

Generating Sentences by Editing Prototypes

Kelvin Guu*, Tatsunori Hashimoto*, Yonatan Oren, Percy Liang

TACL 2018, appeared at ACL 2018



\begin{**Overview**}

Goal: sentence generation

The goal is to develop a general-purpose sentence generation technique that can be used for applications like language modeling, machine translation, dialogue, etc.

Goal: sentence generation

$p(y)$ \longrightarrow $y = \text{"stocks fell by 2 percent"}$

The goal is to develop a general-purpose sentence generation technique that can be used for applications like language modeling, machine translation, dialogue, etc.

Goal: sentence generation

$p(y)$ \longrightarrow $y = \text{"stocks fell by 2 percent"}$

$x = \text{"死马当活马医"}$ $\xrightarrow{p(y | x)}$ $y = \text{"beating a dead horse"}$

The goal is to develop a general-purpose sentence generation technique that can be used for applications like language modeling, machine translation, dialogue, etc.

Goal: sentence generation

$p(y)$ \longrightarrow $y = \text{"stocks fell by 2 percent"}$

$x = \text{"死马当活马医"}$ $\xrightarrow{p(y | x)}$ $y = \text{"beating a dead horse"}$

$x = \text{"how are you?"}$ $\xrightarrow{p(y | x)}$ $y = \text{"pretty good, you?"}$

The goal is to develop a general-purpose sentence generation technique that can be used for applications like language modeling, machine translation, dialogue, etc.

The status quo

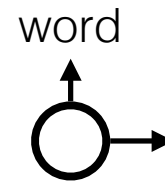
Most existing methods are autoregressive models

- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo



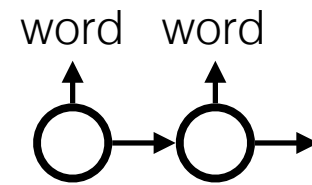
Most existing methods are autoregressive models

- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo



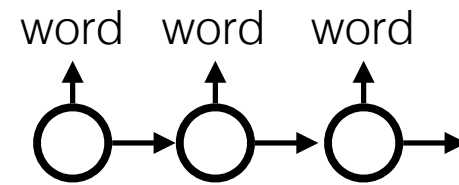
Most existing methods are autoregressive models

- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo



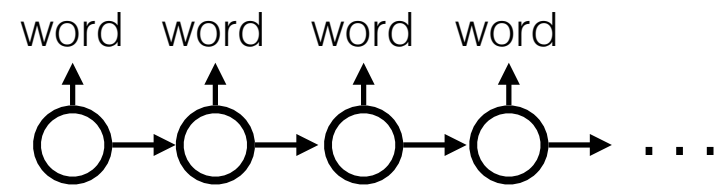
Most existing methods are autoregressive models

- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo



Most existing methods are autoregressive models

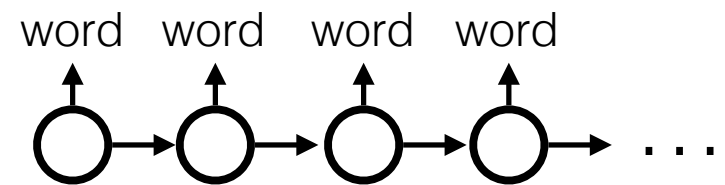
- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo

- left to right
- word by word



Most existing methods are autoregressive models

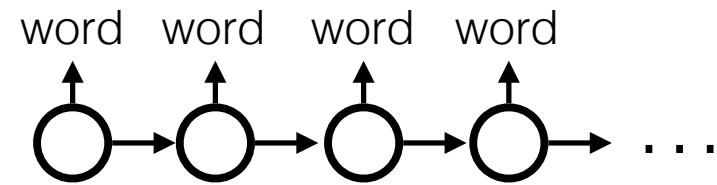
- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo

- left to right
- word by word



Train on wide output distributions

Most existing methods are autoregressive models

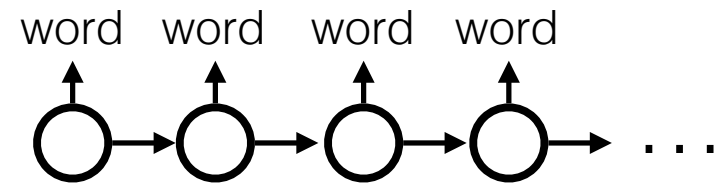
- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo

- left to right
- word by word



Train on wide output distributions

- low diversity

Most existing methods are autoregressive models

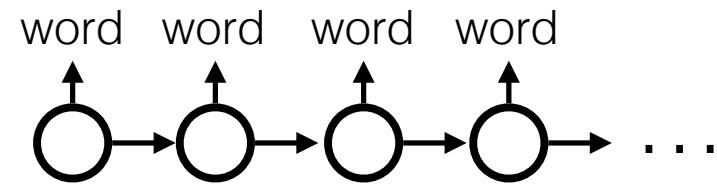
- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo

- left to right
- word by word



Train on wide output distributions

- low diversity
 - the generic utterance problem

Most existing methods are autoregressive models

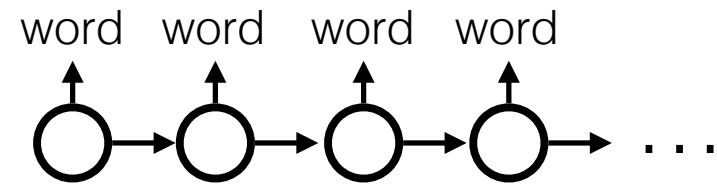
- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo

- left to right
- word by word



Train on wide output distributions

- low diversity
 - the generic utterance problem
 - ("*I don't know*", "*I'm sorry*") [Li+ 2016, Serban+ 2016, Ott+ 2018]

Most existing methods are autoregressive models

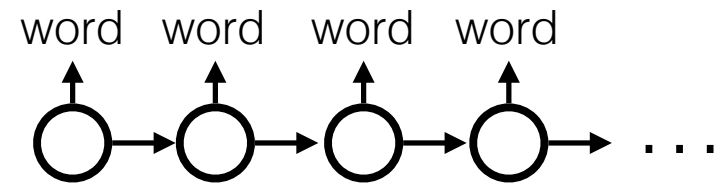
- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

The status quo

- left to right
- word by word



Train on wide output distributions

- low diversity
 - the generic utterance problem
 - (*"I don't know"*, *"I'm sorry"*) [Li+ 2016, Serban+ 2016, Ott+ 2018]
- no semantic control [Hu+ 2017]

Most existing methods are autoregressive models

- Namely, they generate a sentence from scratch, left to right, word by word

When you fit such models to high-entropy "wide" output distributions, you tend to run into two problems

- researchers have found that they often tend to generate highly generic utterances, such as "I don't know" or "I'm sorry"
- And furthermore, there is no notion of semantic control over what gets generated.

Approach: prototype, then edit

To address these problems, we will be proposing a new way to generate sentences.

In this new proposal, we don't generate a sentence from scratch.

Instead, we first grab a sentence from the training set. Let's call that a prototype.

Then you use a sequence to sequence model to edit that sentence into a new sentence.

The prototype enables you to roughly control the semantics of your sentence.

While the seq2seq editor injects additional variation.

And by choosing diverse prototypes, we can avoid the problem of generic utterances.

Approach: prototype, then edit

Overpriced , overrated , and tasteless food .
The food here is ok but not worth the price .
I definitely recommend this restaurante .

Sample from
the training set



Prototype

The food here is ok but not worth the price .

To address these problems, we will be proposing a new way to generate sentences.

In this new proposal, we don't generate a sentence from scratch.

Instead, we first grab a sentence from the training set. Let's call that a prototype.

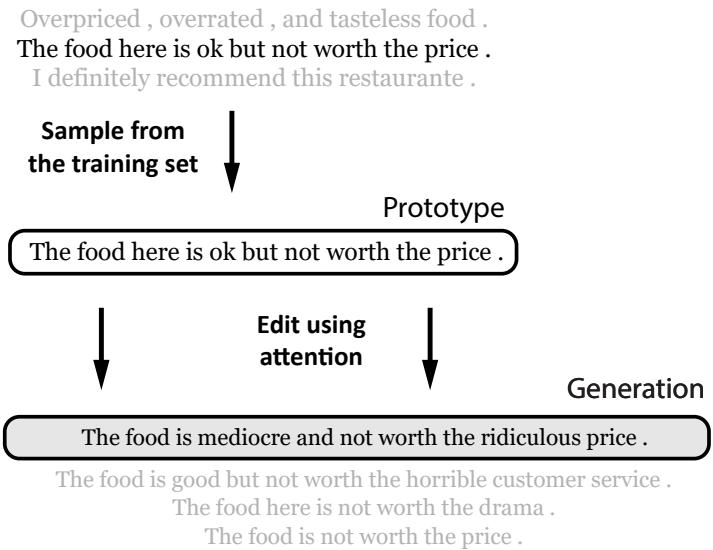
Then you use a sequence to sequence model to edit that sentence into a new sentence.

The prototype enables you to roughly control the semantics of your sentence.

While the seq2seq editor injects additional variation.

And by choosing diverse prototypes, we can avoid the problem of generic utterances.

Approach: prototype, then edit



To address these problems, we will be proposing a new way to generate sentences.

In this new proposal, we don't generate a sentence from scratch.

Instead, we first grab a sentence from the training set. Let's call that a prototype.

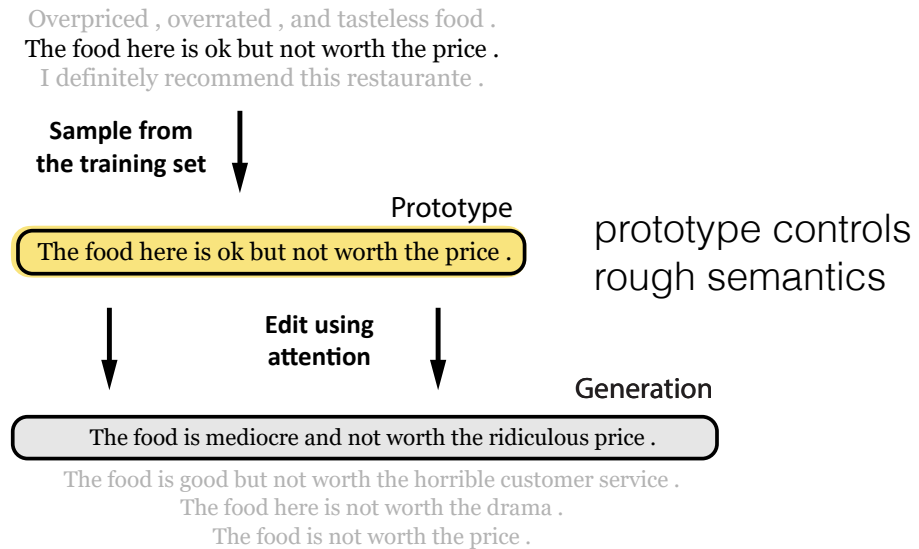
Then you use a sequence to sequence model to edit that sentence into a new sentence.

The prototype enables you to roughly control the semantics of your sentence.

While the seq2seq editor injects additional variation.

And by choosing diverse prototypes, we can avoid the problem of generic utterances.

Approach: prototype, then edit



To address these problems, we will be proposing a new way to generate sentences.

In this new proposal, we don't generate a sentence from scratch.

Instead, we first grab a sentence from the training set. Let's call that a prototype.

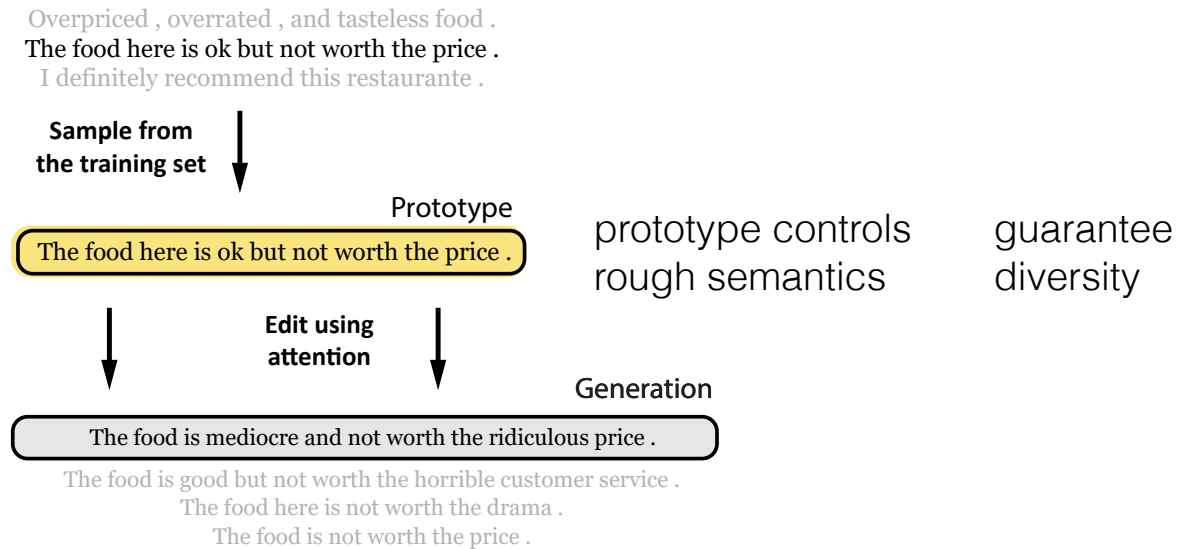
Then you use a sequence to sequence model to edit that sentence into a new sentence.

The prototype enables you to roughly control the semantics of your sentence.

While the seq2seq editor injects additional variation.

And by choosing diverse prototypes, we can avoid the problem of generic utterances.

Approach: prototype, then edit



To address these problems, we will be proposing a new way to generate sentences.

In this new proposal, we don't generate a sentence from scratch.

Instead, we first grab a sentence from the training set. Let's call that a prototype.

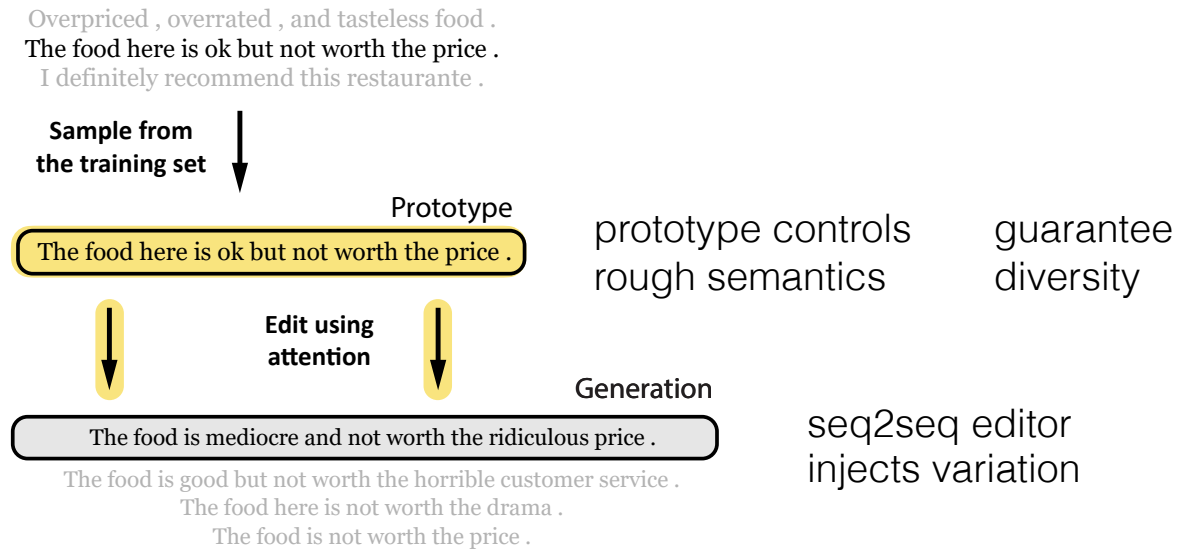
Then you use a sequence to sequence model to edit that sentence into a new sentence.

The prototype enables you to roughly control the semantics of your sentence.

While the seq2seq editor injects additional variation.

And by choosing diverse prototypes, we can avoid the problem of generic utterances.

Approach: prototype, then edit



To address these problems, we will be proposing a new way to generate sentences.

In this new proposal, we don't generate a sentence from scratch.

Instead, we first grab a sentence from the training set. Let's call that a prototype.

Then you use a sequence to sequence model to edit that sentence into a new sentence.

The prototype enables you to roughly control the semantics of your sentence.

While the seq2seq editor injects additional variation.

And by choosing diverse prototypes, we can avoid the problem of generic utterances.

Overview of results

Overview of results

- **More diverse generations**

Overview of results

- **More diverse generations**
- **Higher quality generations** (Mechanical Turk)

Overview of results

- **More diverse generations**
- **Higher quality generations** (Mechanical Turk)
- **Better perplexity** (BillionWord, Yelp reviews)

Overview of results

- **More diverse generations**
- **Higher quality generations** (Mechanical Turk)
- **Better perplexity** (BillionWord, Yelp reviews)
- **Seq2seq edits are semantically interpretable**

Overview of results

- **More diverse generations**
- **Higher quality generations** (Mechanical Turk)
- **Better perplexity** (BillionWord, Yelp reviews)
- **Seq2seq edits are semantically interpretable**
 - preserve semantic similarity

Overview of results

- **More diverse generations**
- **Higher quality generations** (Mechanical Turk)
- **Better perplexity** (BillionWord, Yelp reviews)
- **Seq2seq edits are semantically interpretable**
 - preserve semantic similarity
 - can be used to perform sentence-level analogies

\end{\b{Overview}}

\begin{**Approach**}

prototype, then edit **(formally)**

Here is the new generation process we are proposing, in more formal detail.

First, we sample a prototype sentence from our training set, which I'll call z_p

Then, we sample an edit vector, z_e , from some prior distribution over edits

And finally, we combine the prototype and edit vector to produce a new, edited sentence, y .

To keep track of notation, I'll put this legend on the bottom later slides.

prototype, then edit (**formally**)

Overpriced , overrated , and tasteless food .
The food here is ok but not worth the price .
I definitely recommend this restaurante .

Sample from
the training set



Prototype

The food here is ok but not worth the price .

Here is the new generation process we are proposing, in more formal detail.

First, we sample a prototype sentence from our training set, which I'll call z_p

Then, we sample an edit vector, z_e , from some prior distribution over edits

And finally, we combine the prototype and edit vector to produce a new, edited sentence, y .

To keep track of notation, I'll put this legend on the bottom later slides.

prototype, then edit (**formally**)

Overpriced , overrated , and tasteless food .
The food here is ok but not worth the price .
I definitely recommend this restaurante .

Sample from
the training set



Prototype

The food here is ok but not worth the price .

$$z_p \sim p_{\text{proto}}$$

Here is the new generation process we are proposing, in more formal detail.

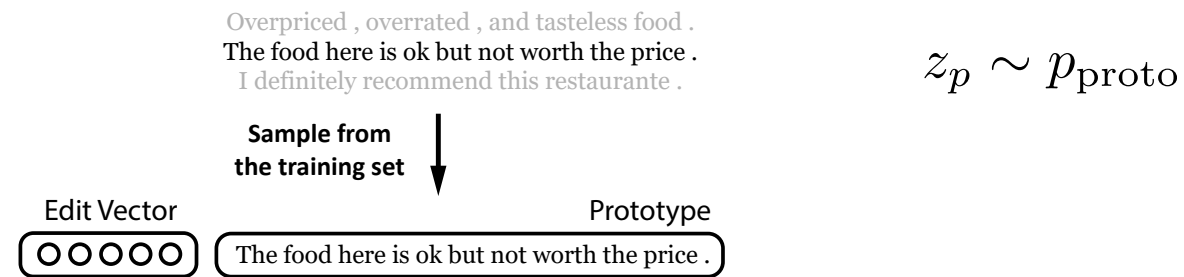
First, we sample a prototype sentence from our training set, which I'll call z_p

Then, we sample an edit vector, z_e , from some prior distribution over edits

And finally, we combine the prototype and edit vector to produce a new, edited sentence, y .

To keep track of notation, I'll put this legend on the bottom later slides.

prototype, then edit (**formally**)



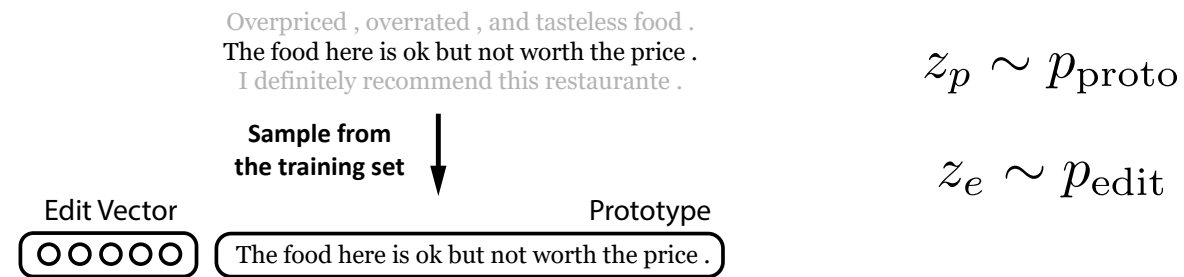
Here is the new generation process we are proposing, in more formal detail.

First, we sample a prototype sentence from our training set, which I'll call z_p
Then, we sample an edit vector, z_e , from some prior distribution over edits

And finally, we combine the prototype and edit vector to produce a new, edited sentence, y .

To keep track of notation, I'll put this legend on the bottom later slides.

prototype, then edit (**formally**)



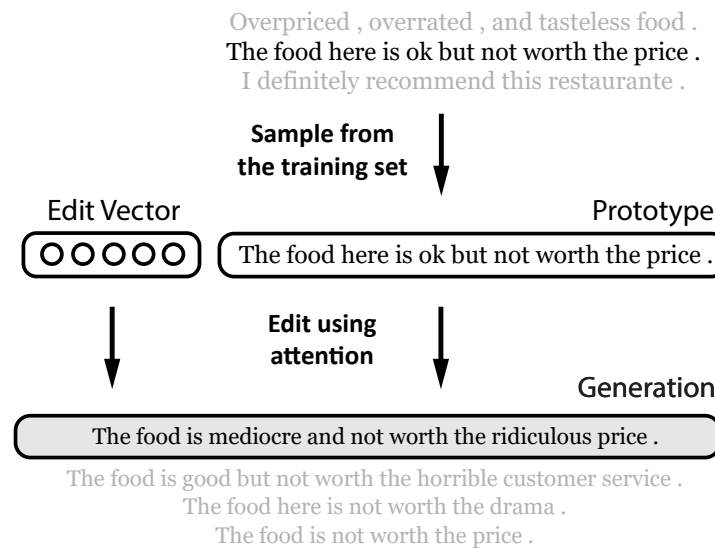
Here is the new generation process we are proposing, in more formal detail.

First, we sample a prototype sentence from our training set, which I'll call z_p
Then, we sample an edit vector, z_e , from some prior distribution over edits

And finally, we combine the prototype and edit vector to produce a new, edited sentence, y .

To keep track of notation, I'll put this legend on the bottom later slides.

prototype, then edit (**formally**)



$$z_p \sim p_{\text{proto}}$$

$$z_e \sim p_{\text{edit}}$$

Here is the new generation process we are proposing, in more formal detail.

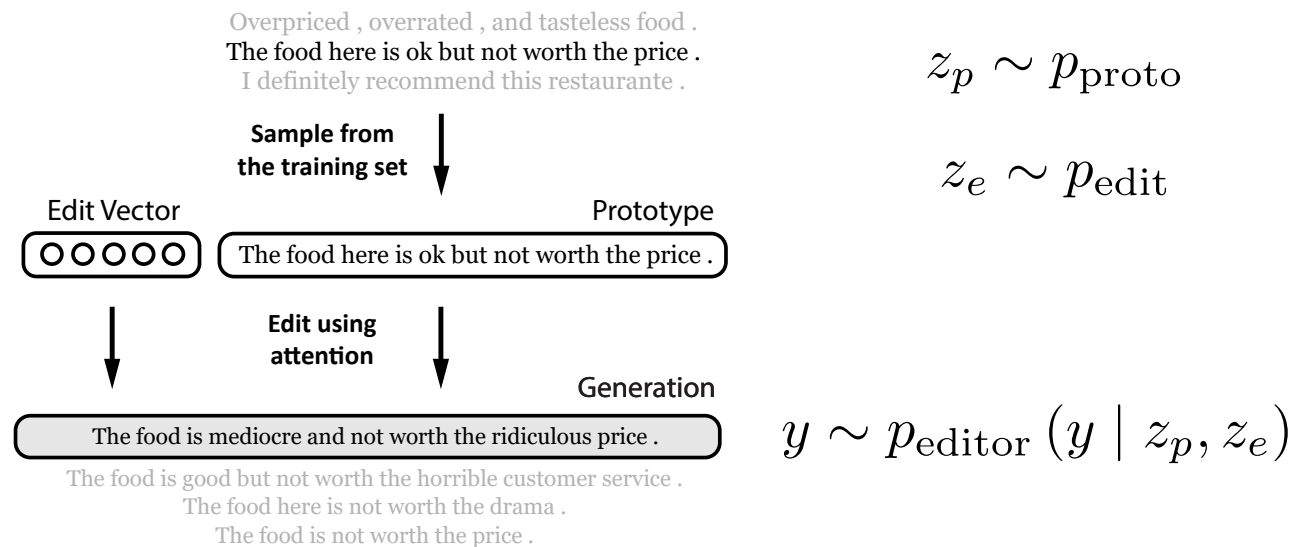
First, we sample a prototype sentence from our training set, which I'll call z_p

Then, we sample an edit vector, z_e , from some prior distribution over edits

And finally, we combine the prototype and edit vector to produce a new, edited sentence, y .

To keep track of notation, I'll put this legend on the bottom later slides.

prototype, then edit (**formally**)



Here is the new generation process we are proposing, in more formal detail.

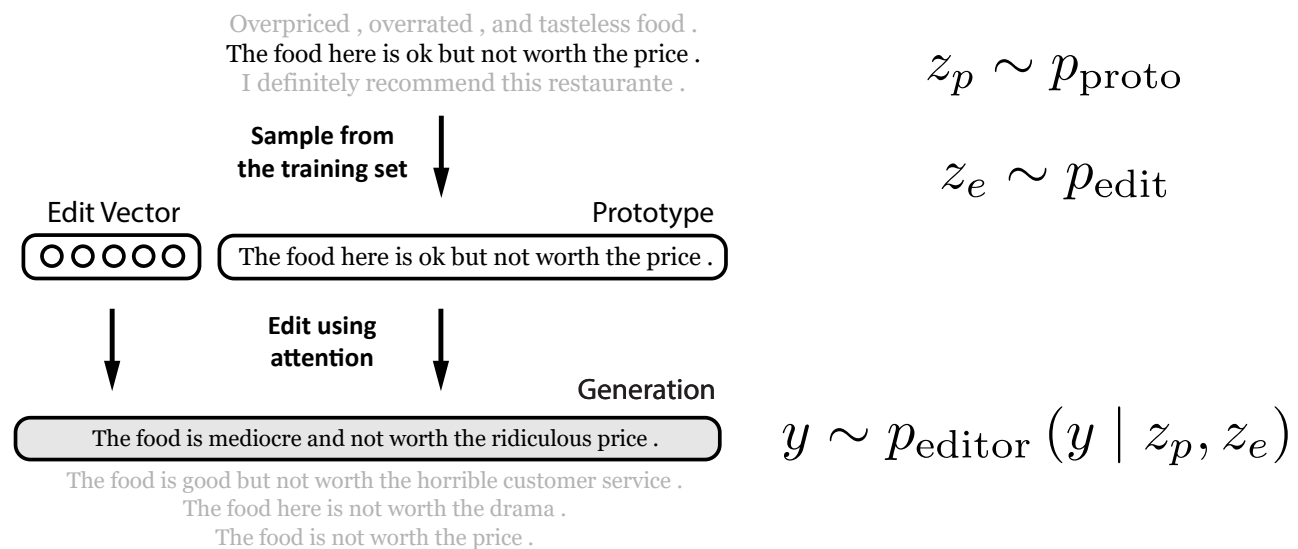
First, we sample a prototype sentence from our training set, which I'll call z_p

Then, we sample an edit vector, z_e , from some prior distribution over edits

And finally, we combine the prototype and edit vector to produce a new, edited sentence, y .

To keep track of notation, I'll put this legend on the bottom later slides.

prototype, then edit (**formally**)



y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Here is the new generation process we are proposing, in more formal detail.

First, we sample a prototype sentence from our training set, which I'll call z_p

Then, we sample an edit vector, z_e , from some prior distribution over edits

And finally, we combine the prototype and edit vector to produce a new, edited sentence, y .

To keep track of notation, I'll put this legend on the bottom later slides.

Intuitions

One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

humans are not pure left-to-right generators

One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

humans are not pure left-to-right generators

- we write a first draft, then edit

One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

humans are not pure left-to-right generators

- we write a first draft, then edit
- we use templates

One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

humans are not pure left-to-right generators

- we write a first draft, then edit
- we use templates
- we plagiarize

One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

humans are not pure left-to-right generators

- we write a first draft, then edit
- we use templates
- we plagiarize



One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

humans are not pure left-to-right generators

- we write a first draft, then edit
- we use templates
- we plagiarize



semi-parametric statistics

One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

humans are not pure left-to-right generators

- we write a first draft, then edit
- we use templates
- we plagiarize



semi-parametric statistics

- we are doing **kernel density estimation** over sentence space

One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

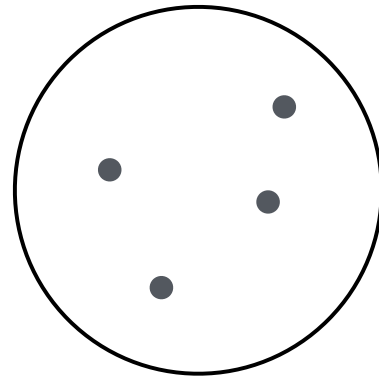
humans are not pure left-to-right generators

- we write a first draft, then edit
- we use templates
- we plagiarize



semi-parametric statistics

- we are doing **kernel density estimation** over sentence space



One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

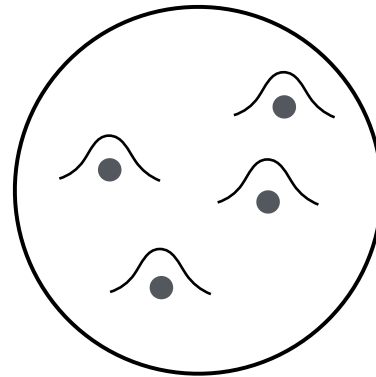
humans are not pure left-to-right generators

- we write a first draft, then edit
- we use templates
- we plagiarize



semi-parametric statistics

- we are doing **kernel density estimation** over sentence space



One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

humans are not pure left-to-right generators

- we write a first draft, then edit
- we use templates
- we plagiarize



semi-parametric statistics

- we are doing **kernel density estimation** over sentence space



One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Intuitions

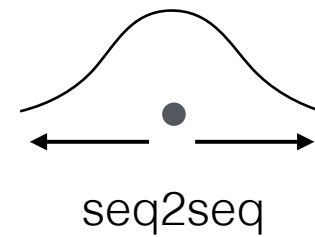
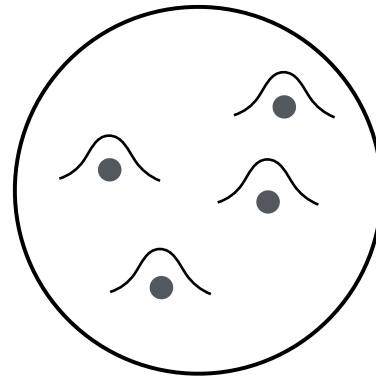
humans are not pure left-to-right generators

- we write a first draft, then edit
- we use templates
- we plagiarize



semi-parametric statistics

- we are doing **kernel density estimation** over sentence space



One observation is that humans are not pure left-to-right generators.

We can also think about this intuition in terms of semi-parametric statistics
namely, we are trying to perform kernel density estimation

Another intuition

Finally, if none of those intuitions excited you...

Ray's comments echo the challenges people have encountered in trying to build models that look like this (left), where a single vector of meaning is meant to generate a sentence.

We sought to avoid this problem, by proposing a model like this (right)

Rather than putting the whole sentence into vector space, We only try to cram the meaning of an edit into a vector.

Another intuition



Professor of Computer Science
The University of Texas at Austin

Finally, if none of those intuitions excited you...

Ray's comments echo the challenges people have encountered in trying to build models that look like this (left), where a single vector of meaning is meant to generate a sentence.

We sought to avoid this problem, by proposing a model like this (right)

Rather than putting the whole sentence into vector space, We only try to cram the meaning of an edit into a vector.

Another intuition



Professor of Computer Science
The University of Texas at Austin

*You can't cram the meaning of a
whole %&!\$ing sentence into
a single \$&!*ing vector!*

[Ray Mooney, ACL 2014]

Finally, if none of those intuitions excited you...

Ray's comments echo the challenges people have encountered in trying to build models that look like this (left), where a single vector of meaning is meant to generate a sentence.

We sought to avoid this problem, by proposing a model like this (right)

Rather than putting the whole sentence into vector space, We only try to cram the meaning of an edit into a vector.

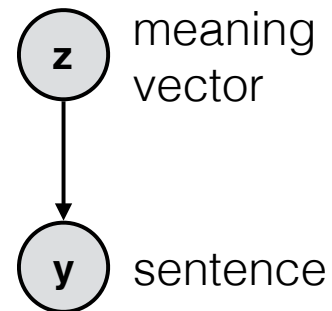
Another intuition



Professor of Computer Science
The University of Texas at Austin

You can't cram the meaning of a whole sentence into a single vector!

[Ray Mooney, ACL 2014]



Finally, if none of those intuitions excited you...

Ray's comments echo the challenges people have encountered in trying to build models that look like this (left), where a single vector of meaning is meant to generate a sentence.

We sought to avoid this problem, by proposing a model like this (right)

Rather than putting the whole sentence into vector space, We only try to cram the meaning of an edit into a vector.

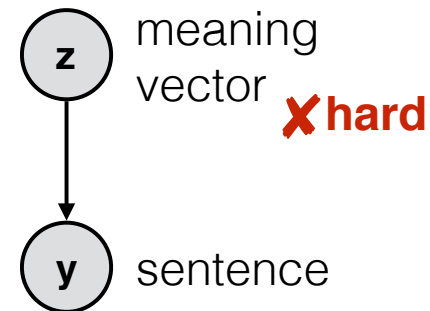
Another intuition



Professor of Computer Science
The University of Texas at Austin

*You can't cram the meaning of a whole %&!\$ing sentence into a single \$&!*ing vector!*

[Ray Mooney, ACL 2014]



Finally, if none of those intuitions excited you...

Ray's comments echo the challenges people have encountered in trying to build models that look like this (left), where a single vector of meaning is meant to generate a sentence.

We sought to avoid this problem, by proposing a model like this (right)

Rather than putting the whole sentence into vector space, We only try to cram the meaning of an edit into a vector.

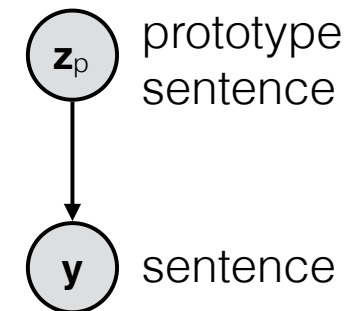
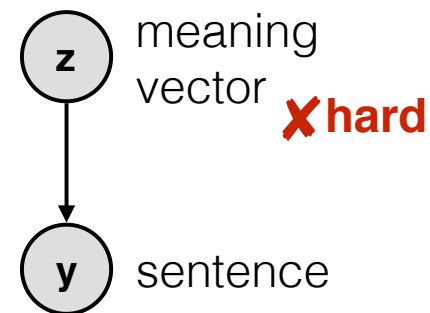
Another intuition



Professor of Computer Science
The University of Texas at Austin

*You can't cram the meaning of a whole %&!\$ing sentence into a single \$&!*ing vector!*

[Ray Mooney, ACL 2014]



Finally, if none of those intuitions excited you...

Ray's comments echo the challenges people have encountered in trying to build models that look like this (left), where a single vector of meaning is meant to generate a sentence.

We sought to avoid this problem, by proposing a model like this (right)

Rather than putting the whole sentence into vector space, We only try to cram the meaning of an edit into a vector.

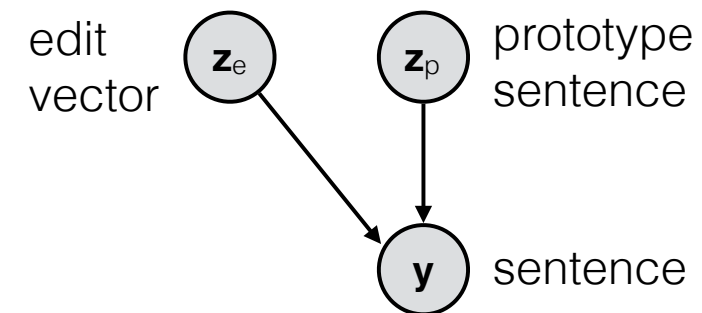
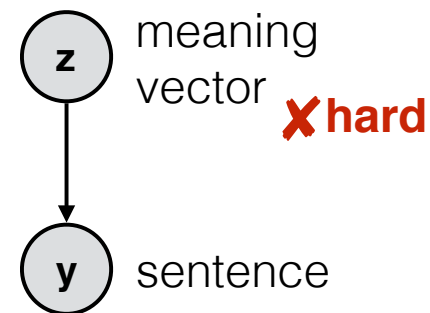
Another intuition



Professor of Computer Science
The University of Texas at Austin

*You can't cram the meaning of a whole %&!\$ing sentence into a single \$&!*ing vector!*

[Ray Mooney, ACL 2014]



Finally, if none of those intuitions excited you...

Ray's comments echo the challenges people have encountered in trying to build models that look like this (left), where a single vector of meaning is meant to generate a sentence.

We sought to avoid this problem, by proposing a model like this (right)

Rather than putting the whole sentence into vector space, We only try to cram the meaning of an edit into a vector.

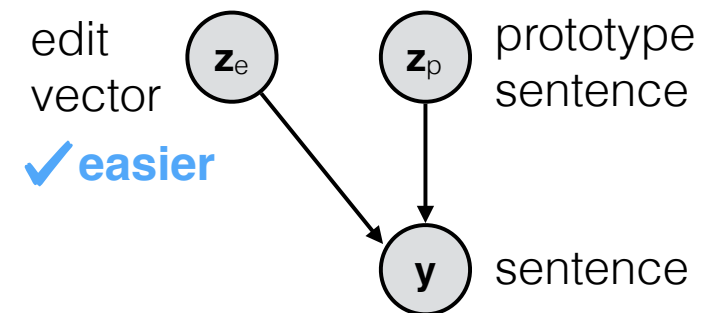
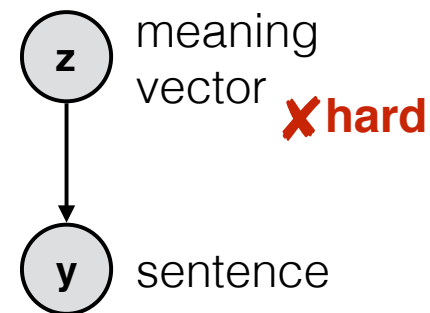
Another intuition



Professor of Computer Science
The University of Texas at Austin

*You can't cram the meaning of a whole %&!\$ing sentence into a single \$&!*ing vector!*

[Ray Mooney, ACL 2014]



Finally, if none of those intuitions excited you...

Ray's comments echo the challenges people have encountered in trying to build models that look like this (left), where a single vector of meaning is meant to generate a sentence.

We sought to avoid this problem, by proposing a model like this (right)

Rather than putting the whole sentence into vector space, We only try to cram the meaning of an edit into a vector.

Training objective

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

$$p(y)$$

y = output sentence **z**_p = prototype sentence **z**_e = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

$$p(y) = \sum_{z_p} p(y \mid z_p) p_{\text{proto}}(z_p)$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

$$p(y) = \sum_{z_p} \underbrace{p(y | z_p)}_{\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e} p_{\text{proto}}(z_p)$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

maximize



$$p(y) = \sum_{z_p} \underbrace{p(y | z_p) p_{\text{proto}}(z_p)}_{\text{Marginalize over } z_p}$$

$$\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

maximize

$$p(y) = \sum_{z_p} \underbrace{p(y | z_p) p_{\text{proto}}(z_p)}_{\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e} \quad \text{expensive}$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

maximize



$$p(y) = \sum_{z_p} \underbrace{p(y | z_p) p_{\text{proto}}(z_p)}_{\text{expensive}}$$

$$\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e \quad \text{intractable}$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

maximize

$$p(y) = \sum_{z_p} \underbrace{p(y | z_p) p_{\text{proto}}(z_p)}_{\text{expensive}} \int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e \quad \text{intractable}$$

key tool: **ELBO** (evidence lower bound)
[Dempster+ '77, Jordan+ '99, Kingma+ '13]

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution


To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

 **maximize**

$$p(y) = \sum_{z_p} \underbrace{p(y | z_p) p_{\text{proto}}(z_p)}_{\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e} \quad \begin{array}{l} \text{expensive} \\ \text{intractable} \end{array}$$

key tool: **ELBO** (evidence lower bound)
[Dempster+ '77, Jordan+ '99, Kingma+ '13]

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound


- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective


The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

maximize

 $p(y) = \sum_{z_p} p(y | z_p) p_{\text{proto}}(z_p)$ expensive

 $\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e$ intractable

key tool: **ELBO** (evidence lower bound)
[Dempster+ '77, Jordan+ '99, Kingma+ '13]

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

maximize



$$p(y) = \sum_{z_p} p(y | z_p) p_{\text{proto}}(z_p)$$

expensive



$$\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e$$

intractable

key tool: **ELBO** (evidence lower bound)

[Dempster+ '77, Jordan+ '99, Kingma+ '13]

- more computationally tractable

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

Training objective

maximize



$$p(y) = \sum_{z_p} p(y | z_p) p_{\text{proto}}(z_p)$$

expensive



$$\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e$$

intractable

key tool: **ELBO** (evidence lower bound)

[Dempster+ '77, Jordan+ '99, Kingma+ '13]

- more computationally tractable
- bias towards semantically interpretable edits

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

The generative story we just saw gives us the following overall likelihood of a sentence

- First, we marginalize out over all prototypes that could have produced this sentence.
- And for each prototype, we marginalize out over all edit vectors that could have produced the resulting edit.

In the ideal world, we would then simply maximize the likelihood of this model

- But note that the summation over all prototypes in the training set is very expensive
- And the integration over all possible edit vectors is completely intractable with no closed form solution

To deal with these two problems, we will be applying an important tool from variational inference, called the evidence lower bound

- We will use the ELBO, to derive a lower bound for each of these expressions
- We can then maximize the lower bound instead of the original objective

The new approximate objective will serve two purposes

- be more computationally tractable
- and it will actually bias the model towards more semantically meaningful edits

ELBO (in general)

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z

When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

ELBO (in general)

$$\log p(y)$$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z


When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

ELBO (in general)

$$\log p(y)$$

$$\int_z p(y | z) p(z) dz$$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z

When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

ELBO (in general)

$$\underbrace{\log p(y)}_{\int_z p(y|z)p(z)dz} \geq \int_z \log p(y|z) q(z) dz - KL(q(z) || p(z))$$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z

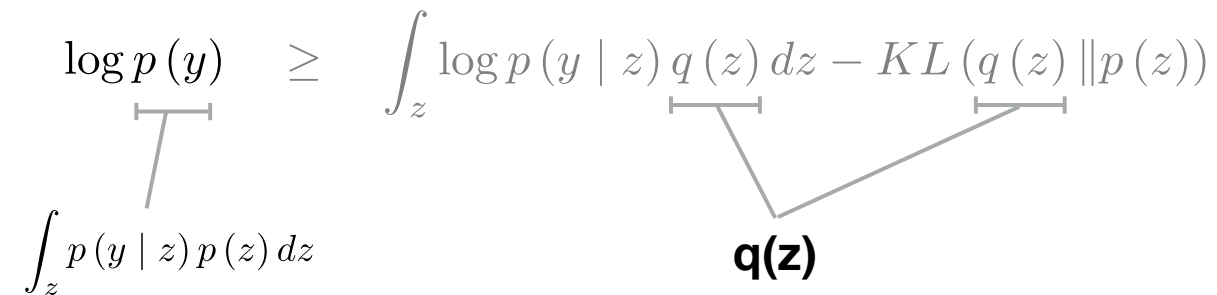
When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

ELBO (in general)

$$\log p(y) \geq \int_z \log p(y | z) q(z) dz - KL(q(z) || p(z))$$


$\int_z p(y | z) p(z) dz$

q(z)

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z

When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

ELBO (in general)

$$\log p(y) \geq \int_z \log p(y | z) q(z) dz - KL(q(z) || p(z))$$

$\int_z p(y | z) p(z) dz$

q(z)

you choose q(z)

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z

When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

ELBO (in general)

$$\log p(y) \geq \int_z \log p(y | z) q(z) dz - KL(q(z) || p(z))$$

$\int_z p(y | z) p(z) dz$

q(z)

you choose q(z)

- add helpful biases to the model

y = output sentence **z**_p = prototype sentence **z**_e = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z

When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

ELBO (in general)

$$\underbrace{\log p(y)}_{\int_z p(y|z)p(z)dz} \geq \underbrace{\int_z \log p(y|z) q(z) dz}_{\mathbf{q(z)}} - \underbrace{KL(q(z) || p(z))}_{\text{grey}}$$

you choose q(z)

- add helpful biases to the model
- tightness of the lower bound

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z

When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

ELBO (in general)

$$\underbrace{\log p(y)}_{\int_z p(y|z)p(z)dz} \geq \underbrace{\int_z \log p(y|z) q(z) dz}_{\mathbf{q(z)}} - \underbrace{KL(q(z) || p(z))}_{\text{grey}}$$

you choose $q(z)$

- add helpful biases to the model
- tightness of the lower bound $q(z) \approx p(z|y)$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z

When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

ELBO (in general)

$$\log p(y) \geq \int_z \log p(y | z) q(z) dz - KL(q(z) || p(z))$$

$\int_z p(y | z) p(z) dz$

q(z)

you choose q(z)

- add helpful biases to the model
- tightness of the lower bound

$$q(z) \approx p(z | y)$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

To use the ELBO, let's first understand what it is.

If you have a log-likelihood function $p(y)$

And it involves a latent variable z

Then ELBO says you can lower bound the likelihood with this expression

I made the RHS grey because you don't need to look too hard at this right now

The only important thing to note about the RHS is that it introduces a new distribution q over the latent variable z

When you use the ELBO, you get to **choose** what q is.

- You can choose q to deliberately inject helpful biases into the model
- And your choice of q also affects the tightness of the lower bound
 - In particular, the lower bound is perfectly tight when q matches the true posterior distribution over z 's

All of our innovation in the next few slides will come from how we design Q

And we will be using this key criterion right here to guide us in designing Q

Training objective

maximize



$$p(y) = \sum_{z_p} \underbrace{p(y | z_p)}_{\text{expensive}} p_{\text{proto}}(z_p)$$



$$\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e \quad \text{intractable}$$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

So, let's start by applying the ELBO to our sum over prototypes.

ELBO on prototypes

$$p(y) = \sum_{z_p} p(y \mid z_p) p_{\text{proto}}(z_p)$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO to that expression gives us the lower bound in gray.

Again, no need to look too hard at this.

What matters to make this a good lower bound, is that we must choose q to match the true posterior.

In this expression, the prototype sentence is the latent variable.

Note that Q is a distribution over prototype sentences.

And the posterior is a distribution over prototypes, given the output y.

But what does the true posterior look like?

ELBO on prototypes

$$\begin{aligned} p(y) &= \sum_{z_p} p(y \mid z_p) p_{\text{proto}}(z_p) \\ &\geq \sum_{z_p} \log p(y \mid z_p) q(z_p) - KL(q(z_p) \parallel p_{\text{proto}}(z_p)) \end{aligned}$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO to that expression gives us the lower bound in gray.

Again, no need to look too hard at this.

What matters to make this a good lower bound, is that we must choose q to match the true posterior.

In this expression, the prototype sentence is the latent variable.

Note that Q is a distribution over prototype sentences.

And the posterior is a distribution over prototypes, given the output y.

But what does the true posterior look like?

ELBO on prototypes

$$\begin{aligned} p(y) &= \sum_{z_p} p(y \mid z_p) p_{\text{proto}}(z_p) \\ &\geq \sum_{z_p} \log p(y \mid z_p) q(z_p) - KL(q(z_p) \parallel p_{\text{proto}}(z_p)) \end{aligned}$$

$$q(z_p) \approx p(z_p \mid y)$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO to that expression gives us the lower bound in gray.

Again, no need to look too hard at this.

What matters to make this a good lower bound, is that we must choose q to match the true posterior.

In this expression, the prototype sentence is the latent variable.

Note that Q is a distribution over prototype sentences.

And the posterior is a distribution over prototypes, given the output y.

But what does the true posterior look like?

ELBO on prototypes

$$\begin{aligned} p(y) &= \sum_{z_p} p(y \mid z_p) p_{\text{proto}}(z_p) \\ &\geq \sum_{z_p} \log p(y \mid z_p) q(z_p) - KL(q(z_p) \parallel p_{\text{proto}}(z_p)) \end{aligned}$$

$$q(z_p) \approx p(z_p \mid y) \text{ ?}$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO to that expression gives us the lower bound in gray.

Again, no need to look too hard at this.

What matters to make this a good lower bound, is that we must choose q to match the true posterior.

In this expression, the prototype sentence is the latent variable.

Note that Q is a distribution over prototype sentences.

And the posterior is a distribution over prototypes, given the output y.

But what does the true posterior look like?

$q(z)$ over prototypes

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

In particular, z_p should probably be in some neighborhood of y
where the neighborhood is all sentences with high token overlap
and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned.
Also, q depends on y , which is perfectly okay.

q(z) over prototypes

Question

$$q(z_p) \approx p(z_p | y)$$

y = output sentence **z**_p = prototype sentence **z**_e = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

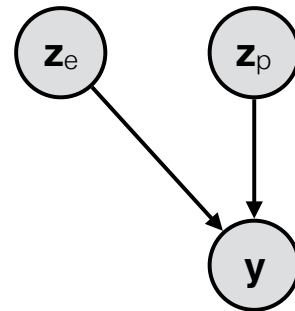
In particular, z_p should probably be in some neighborhood of y
where the neighborhood is all sentences with high token overlap
and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned.
Also, q depends on y , which is perfectly okay.

q(z) over prototypes

Question

$$q(z_p) \approx p(z_p | y)$$



y = output sentence **z_p** = prototype sentence **z_e** = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

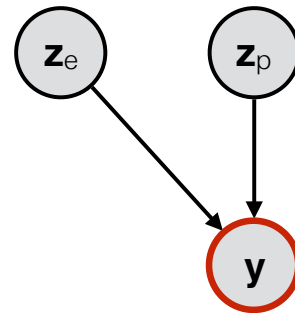
In particular, z_p should probably be in some neighborhood of y where the neighborhood is all sentences with high token overlap and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned. Also, q depends on y , which is perfectly okay.

q(z) over prototypes

Question

$$q(z_p) \approx p(z_p | y)$$



y = output sentence **z_p** = prototype sentence **z_e** = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

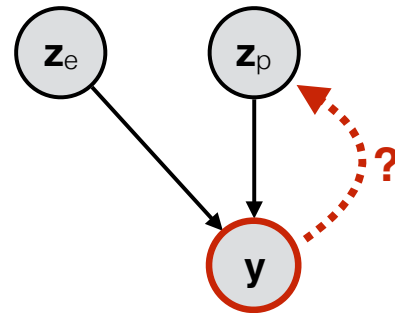
In particular, z_p should probably be in some neighborhood of y where the neighborhood is all sentences with high token overlap and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned. Also, q depends on y , which is perfectly okay.

q(z) over prototypes

Question

$$q(z_p) \approx p(z_p | y)$$



\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

In particular, z_p should probably be in some neighborhood of y where the neighborhood is all sentences with high token overlap and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned. Also, q depends on y , which is perfectly okay.

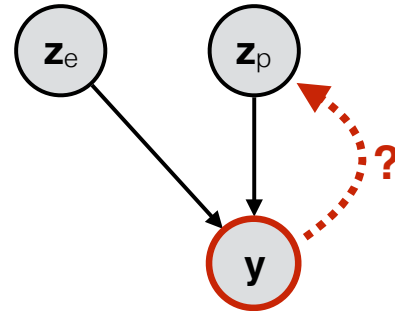
q(z) over prototypes

Question

$$q(z_p) \approx p(z_p | y)$$

Answer

prototype \mathbf{z}_p was probably not too different from \mathbf{y} .



\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

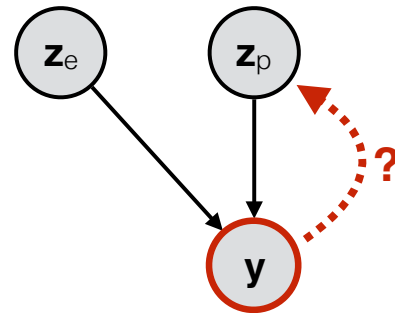
In particular, z_p should probably be in some neighborhood of y where the neighborhood is all sentences with high token overlap and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned. Also, q depends on y , which is perfectly okay.

q(z) over prototypes

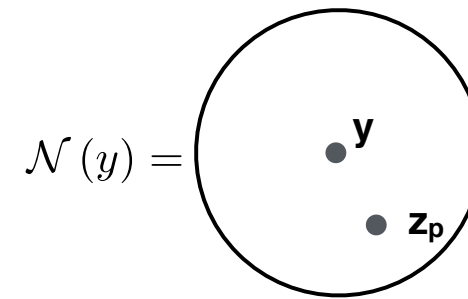
Question

$$q(z_p) \approx p(z_p | y)$$



Answer

prototype z_p was probably not too different from y .



y = output sentence z_p = prototype sentence z_e = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

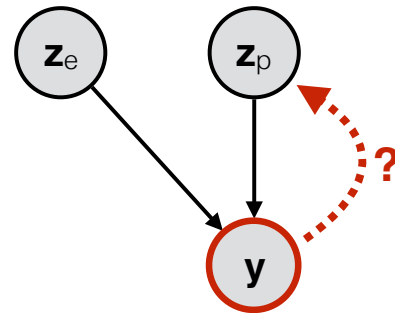
In particular, z_p should probably be in some neighborhood of y where the neighborhood is all sentences with high token overlap and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned. Also, q depends on y , which is perfectly okay.

q(z) over prototypes

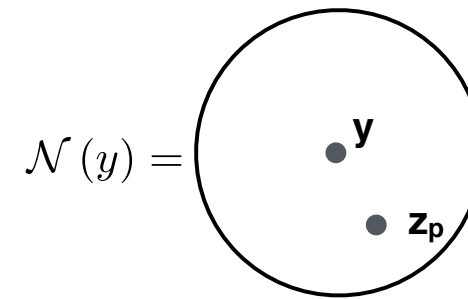
Question

$$q(z_p) \approx p(z_p | y)$$



Answer

prototype z_p was probably not too different from y .



$\mathbf{N}(y)$ = all sentences with high token overlap

y = output sentence z_p = prototype sentence z_e = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

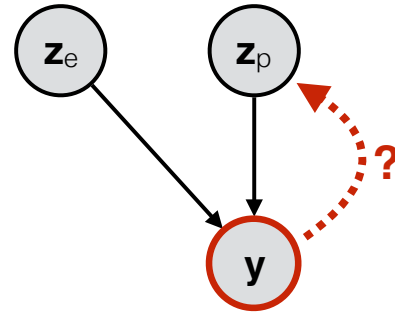
In particular, z_p should probably be in some neighborhood of y where the neighborhood is all sentences with high token overlap and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned. Also, q depends on y , which is perfectly okay.

q(z) over prototypes

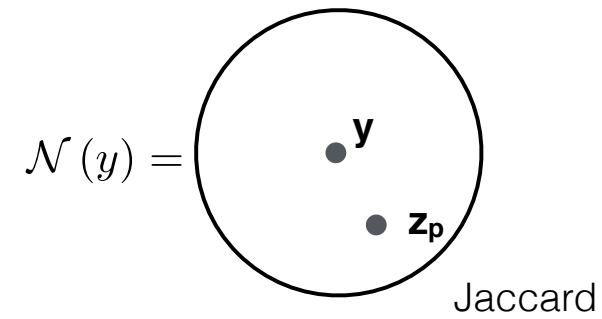
Question

$$q(z_p) \approx p(z_p | y)$$



Answer

prototype z_p was probably not too different from y .



$\mathbf{N}(y)$ = all sentences with high token overlap

y = output sentence z_p = prototype sentence z_e = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

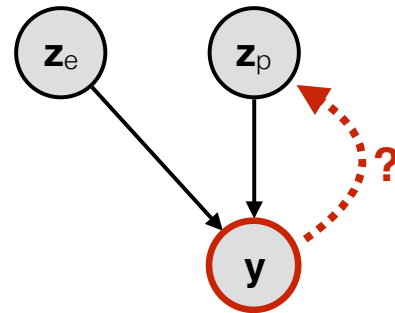
In particular, z_p should probably be in some neighborhood of y where the neighborhood is all sentences with high token overlap and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned. Also, q depends on y , which is perfectly okay.

q(z) over prototypes

Question

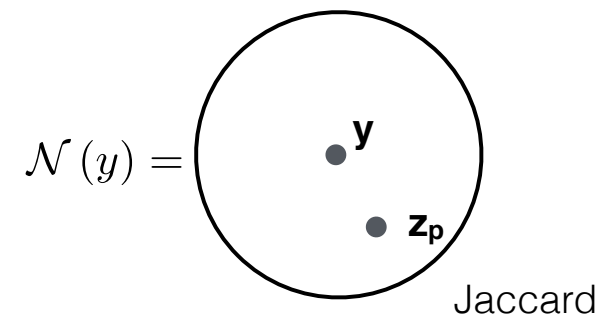
$$q(z_p) \approx p(z_p | y)$$



Answer

prototype z_p was probably not too different from y .

$$q(z_p) := \text{Uniform}(\mathcal{N}(y))$$



$\mathbf{N}(y)$ = all sentences with high token overlap

y = output sentence z_p = prototype sentence z_e = edit vector

In plain English, the posterior $p(z | y)$ asks the following question:

- If I see a sentence y , what prototype z_p did it come from?

The answer I propose is that z_p should probably be something not too different from y .

In particular, z_p should probably be in some neighborhood of y where the neighborhood is all sentences with high token overlap and token overlap is measured by a threshold on Jaccard distance

Note that q is just a fixed distribution in this case. It is not learned. Also, q depends on y , which is perfectly okay.

q(z) over prototypes

$$\text{ELBO} = \sum_{z_p} \log p(y \mid z_p) q(z_p) - KL(q(z_p) \parallel p_{\text{proto}}(z_p))$$

y = output sentence **z**_p = prototype sentence **z**_e = edit vector

When we plug this choice of q into the evidence lower bound, we get the following much simpler expression

This looks like a typical seq2seq objective, where we are mapping each prototype in the neighborhood of y, to the sentence y

It's now clear that this particular choice of q encourages the editor to make small edits, since the sentence pairs are not that different

Furthermore, as we know, seq2seq training is computationally tractable using SGD

I also want to point out that there was nothing special about using Jaccard distance. You could have used any distance function, even a machine-learned paraphrase distance function, and you would still get this expression.

q(z) over prototypes

$$\text{ELBO} = \sum_{z_p} \log p(y \mid z_p) q(z_p) - KL(q(z_p) \parallel p_{\text{proto}}(z_p))$$

||

$$\frac{1}{|\mathcal{N}(y)|} \sum_{z_p \in \mathcal{N}(y)} \log p(y \mid z_p) + C$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

When we plug this choice of q into the evidence lower bound, we get the following much simpler expression

This looks like a typical seq2seq objective, where we are mapping each prototype in the neighborhood of y, to the sentence y

It's now clear that this particular choice of q encourages the editor to make small edits, since the sentence pairs are not that different

Furthermore, as we know, seq2seq training is computationally tractable using SGD

I also want to point out that there was nothing special about using Jaccard distance. You could have used any distance function, even a machine-learned paraphrase distance function, and you would still get this expression.

q(z) over prototypes

$$\text{ELBO} = \sum_{z_p} \log p(y | z_p) q(z_p) - KL(q(z_p) || p_{\text{proto}}(z_p))$$

||

$$\frac{1}{|\mathcal{N}(y)|} \sum_{z_p \in \mathcal{N}(y)} \log p(y | z_p) + C$$

Looks like typical **sequence-to-sequence** objective

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

When we plug this choice of q into the evidence lower bound, we get the following much simpler expression

This looks like a typical seq2seq objective, where we are mapping each prototype in the neighborhood of y, to the sentence y

It's now clear that this particular choice of q encourages the editor to make small edits, since the sentence pairs are not that different

Furthermore, as we know, seq2seq training is computationally tractable using SGD

I also want to point out that there was nothing special about using Jaccard distance. You could have used any distance function, even a machine-learned paraphrase distance function, and you would still get this expression.

q(z) over prototypes

$$\text{ELBO} = \sum_{z_p} \log p(y | z_p) q(z_p) - KL(q(z_p) || p_{\text{proto}}(z_p))$$

||

$$\frac{1}{|\mathcal{N}(y)|} \sum_{z_p \in \mathcal{N}(y)} \log p(y | z_p) + C$$

Looks like typical **sequence-to-sequence** objective

prototype $\mathbf{z}_p \rightarrow$ output \mathbf{y}

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

When we plug this choice of q into the evidence lower bound, we get the following much simpler expression

This looks like a typical seq2seq objective, where we are mapping each prototype in the neighborhood of y, to the sentence y

It's now clear that this particular choice of q encourages the editor to make small edits, since the sentence pairs are not that different

Furthermore, as we know, seq2seq training is computationally tractable using SGD

I also want to point out that there was nothing special about using Jaccard distance. You could have used any distance function, even a machine-learned paraphrase distance function, and you would still get this expression.

$q(z)$ over prototypes

$$\text{ELBO} = \sum_{z_p} \log p(y | z_p) q(z_p) - KL(q(z_p) || p_{\text{proto}}(z_p))$$

||

$$\frac{1}{|\mathcal{N}(y)|} \sum_{z_p \in \mathcal{N}(y)} \log p(y | z_p) + C$$

Looks like typical **sequence-to-sequence** objective

prototype $\mathbf{z}_p \rightarrow$ output \mathbf{y}



bias towards small edits

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

When we plug this choice of q into the evidence lower bound, we get the following much simpler expression

This looks like a typical seq2seq objective, where we are mapping each prototype in the neighborhood of y , to the sentence y

It's now clear that this particular choice of q encourages the editor to make small edits, since the sentence pairs are not that different

Furthermore, as we know, seq2seq training is computationally tractable using SGD

I also want to point out that there was nothing special about using Jaccard distance. You could have used any distance function, even a machine-learned paraphrase distance function, and you would still get this expression.

q(z) over prototypes

$$\text{ELBO} = \sum_{z_p} \log p(y \mid z_p) q(z_p) - KL(q(z_p) \parallel p_{\text{proto}}(z_p))$$

||

$$\frac{1}{|\mathcal{N}(y)|} \sum_{z_p \in \mathcal{N}(y)} \log p(y \mid z_p) + C$$

Looks like typical **sequence-to-sequence** objective

prototype $\mathbf{z}_p \rightarrow$ output \mathbf{y}



bias towards small edits



computationally tractable

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

When we plug this choice of q into the evidence lower bound, we get the following much simpler expression

This looks like a typical seq2seq objective, where we are mapping each prototype in the neighborhood of y, to the sentence y

It's now clear that this particular choice of q encourages the editor to make small edits, since the sentence pairs are not that different

Furthermore, as we know, seq2seq training is computationally tractable using SGD

I also want to point out that there was nothing special about using Jaccard distance. You could have used any distance function, even a machine-learned paraphrase distance function, and you would still get this expression.

Training objective

maximize



$$p(y) = \sum_{z_p} \underbrace{p(y | z_p)}_{\text{expensive}} p_{\text{proto}}(z_p)$$

expensive



$$\int_{z_e} p_{\text{editor}}(y | z_p, z_e) p_{\text{edit}}(z_e) dz_e$$

intractable

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

So that handles the first evidence lower bound.

Now, let's go to the second ELBO. This time, we want to handle the integral over edit vectors.

ELBO on edit vectors

$$\log p(y \mid z_p)$$

y = output sentence **z**_p = prototype sentence **z**_e = edit vector

Applying the ELBO, we get the following bound

There are two terms in this bound, which we can call the reconstruction cost and the KL penalty.

ELBO on edit vectors

$$\log p(y \mid z_p)$$

$$\geq E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y \mid z_p, z_e)] - KL(q(z_e) \parallel p_{\text{edit}}(z_e))$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO, we get the following bound

There are two terms in this bound, which we can call the reconstruction cost and the KL penalty.

ELBO on edit vectors

$$\log p(y \mid z_p)$$

$$\geq \underbrace{E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y \mid z_p, z_e)]}_{\text{reconstruction_cost}} - KL(q(z_e) \parallel p_{\text{edit}}(z_e))$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO, we get the following bound

There are two terms in this bound, which we can call the reconstruction cost and the KL penalty.

ELBO on edit vectors

$$\log p(y \mid z_p)$$

$$\geq \underbrace{E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y \mid z_p, z_e)]}_{\text{reconstruction_cost}} - \underbrace{KL(q(z_e) \parallel p_{\text{edit}}(z_e))}_{\text{KL_penalty}}$$


y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO, we get the following bound

There are two terms in this bound, which we can call the reconstruction cost and the KL penalty.

ELBO on edit vectors

sample \mathbf{z}_e from $\mathbf{q}(\mathbf{z}_e)$ $\log p(y | z_p)$


$$\geq \underbrace{E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y | z_p, z_e)]}_{\text{reconstruction_cost}} - \underbrace{KL(q(z_e) || p_{\text{edit}}(z_e))}_{\text{KL_penalty}}$$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

Applying the ELBO, we get the following bound

There are two terms in this bound, which we can call the reconstruction cost and the KL penalty.

ELBO on edit vectors

sample \mathbf{z}_e from $\mathbf{q}(\mathbf{z}_e)$ $\log p(y | z_p)$

$\mathbf{z}_p, \mathbf{z}_e \longrightarrow \mathbf{y}$

$$\geq \underbrace{E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y | z_p, z_e)]}_{\text{reconstruction_cost}} - \underbrace{KL(q(z_e) || p_{\text{edit}}(z_e))}_{\text{KL_penalty}}$$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

Applying the ELBO, we get the following bound

There are two terms in this bound, which we can call the reconstruction cost and the KL penalty.

ELBO on edit vectors

sample \mathbf{z}_e from $\mathbf{q}(\mathbf{z}_e)$ $\log p(y | z_p)$

$\mathbf{z}_p, \mathbf{z}_e \longrightarrow \mathbf{y}$

$$\geq \underbrace{E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y | z_p, z_e)]}_{\text{reconstruction_cost}} - \underbrace{KL(q(z_e) || p_{\text{edit}}(z_e))}_{\text{KL_penalty}}$$

measures how well we can
reconstruct \mathbf{y} from prototype
 \mathbf{z}_p and edit \mathbf{z}_e

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

Applying the ELBO, we get the following bound

There are two terms in this bound, which we can call the reconstruction cost and the KL penalty.

ELBO on edit vectors

sample \mathbf{z}_e from $\mathbf{q}(\mathbf{z}_e)$ $\log p(y | z_p)$

$\mathbf{z}_p, \mathbf{z}_e \longrightarrow \mathbf{y}$

$$\geq \underbrace{E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y | z_p, z_e)]}_{\text{reconstruction_cost}} - \underbrace{KL(q(z_e) || p_{\text{edit}}(z_e))}_{\text{KL_penalty}}$$

measures how well we can reconstruct \mathbf{y} from prototype \mathbf{z}_p and edit \mathbf{z}_e

measures difference between \mathbf{q} and edit prior \mathbf{p}_{edit}

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

Applying the ELBO, we get the following bound

There are two terms in this bound, which we can call the reconstruction cost and the KL penalty.

ELBO on edit vectors

$$\log p(y \mid z_p)$$

$$\geq E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y \mid z_p, z_e)] - KL(q(z_e) \parallel p_{\text{edit}}(z_e))$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO, we get the following bound

This expression is rather large, and warrants some explaining.

What matters is that there are two terms, which we can call the reconstruction cost and the KL penalty.

As before, we need to choose a good definition of q. This time, it needs to match the posterior distribution over edit vectors. Again, let's meditate on what this posterior looks like.

ELBO on edit vectors

$$\log p(y \mid z_p)$$

$$\geq E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y \mid z_p, z_e)] - KL(q(z_e) \parallel p_{\text{edit}}(z_e))$$

$$q(z_e) \approx p(z_e \mid y, z_p)$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO, we get the following bound

This expression is rather large, and warrants some explaining.

What matters is that there are two terms, which we can call the reconstruction cost and the KL penalty.

As before, we need to choose a good definition of q. This time, it needs to match the posterior distribution over edit vectors. Again, let's meditate on what this posterior looks like.

ELBO on edit vectors

$$\log p(y \mid z_p)$$

$$\geq E_{z_e \sim q(z_e)} [\log p_{\text{editor}}(y \mid z_p, z_e)] - KL(q(z_e) \parallel p_{\text{edit}}(z_e))$$

$$q(z_e) \approx p(z_e \mid y, z_p) \text{ ?}$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

Applying the ELBO, we get the following bound

This expression is rather large, and warrants some explaining.

What matters is that there are two terms, which we can call the reconstruction cost and the KL penalty.

As before, we need to choose a good definition of q. This time, it needs to match the posterior distribution over edit vectors. Again, let's meditate on what this posterior looks like.

$q(z)$ over edits

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

In plain English, the posterior is saying:

if I see both prototype sentence \mathbf{z}_p and output \mathbf{y} , what edit vector relates them?

q(z) over edits

Question

$$q(z_e) \approx p(z_e \mid y, z_p)$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

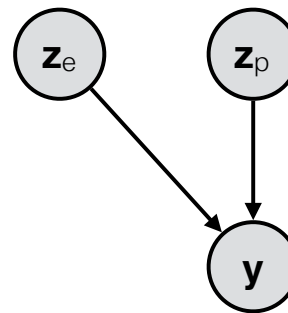
In plain English, the posterior is saying:

if I see both prototype sentence z_p and output y , what edit vector relates them?

q(z) over edits

Question

$$q(z_e) \approx p(z_e \mid y, z_p)$$



y = output sentence **z_p** = prototype sentence **z_e** = edit vector

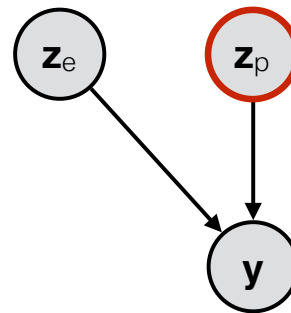
In plain English, the posterior is saying:

if I see both prototype sentence z_p and output y , what edit vector relates them?

q(z) over edits

Question

$$q(z_e) \approx p(z_e \mid y, z_p)$$



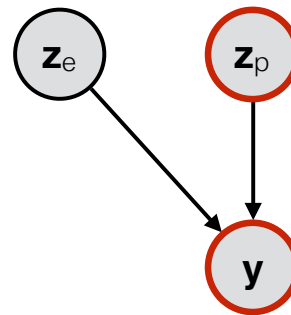
y = output sentence **z_p** = prototype sentence **z_e** = edit vector

In plain English, the posterior is saying:
if I see both prototype sentence z_p and output y , what edit vector relates them?

q(z) over edits

Question

$$q(z_e) \approx p(z_e \mid y, z_p)$$



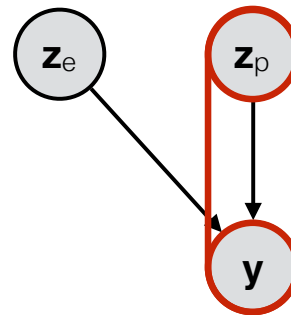
y = output sentence **z_p** = prototype sentence **z_e** = edit vector

In plain English, the posterior is saying:
if I see both prototype sentence z_p and output y , what edit vector relates them?

$q(z)$ over edits

Question

$$q(z_e) \approx p(z_e \mid y, z_p)$$



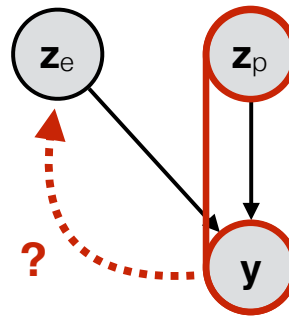
\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

In plain English, the posterior is saying:
if I see both prototype sentence z_p and output y , what edit vector relates them?

q(z) over edits

Question

$$q(z_e) \approx p(z_e \mid y, z_p)$$



\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

In plain English, the posterior is saying:
if I see both prototype sentence z_p and output y , what edit vector relates them?

q(z) over edits

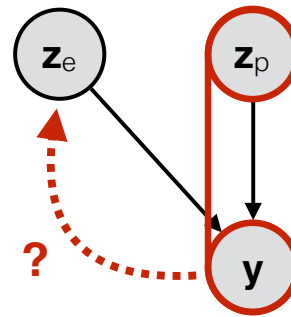
Question

$$q(z_e) \approx p(z_e \mid y, z_p)$$

Answer

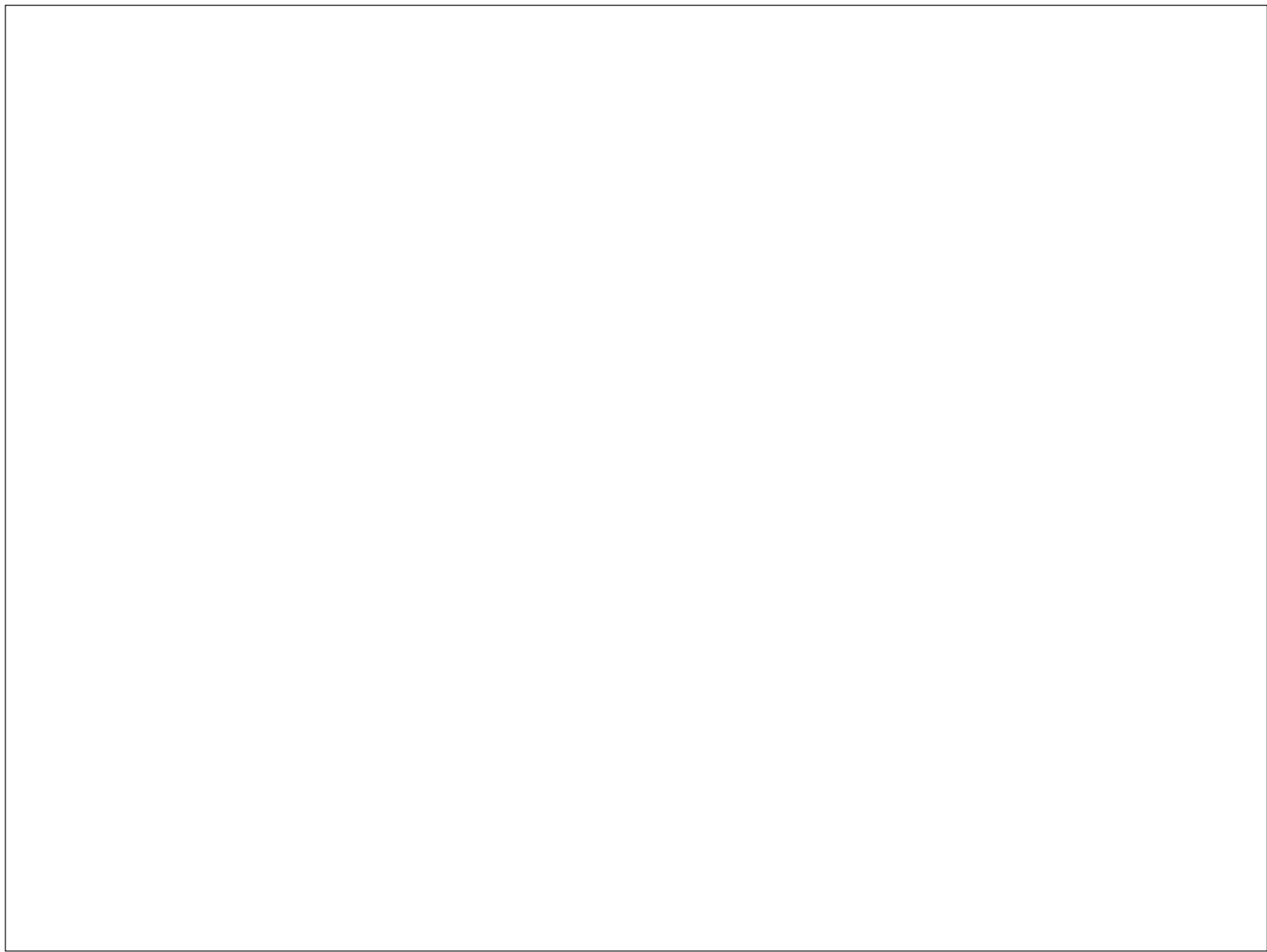
Compare the two sentences.

Figure out which words were **inserted** and **deleted**.
Then sum their word vectors.



y = output sentence z_p = prototype sentence z_e = edit vector

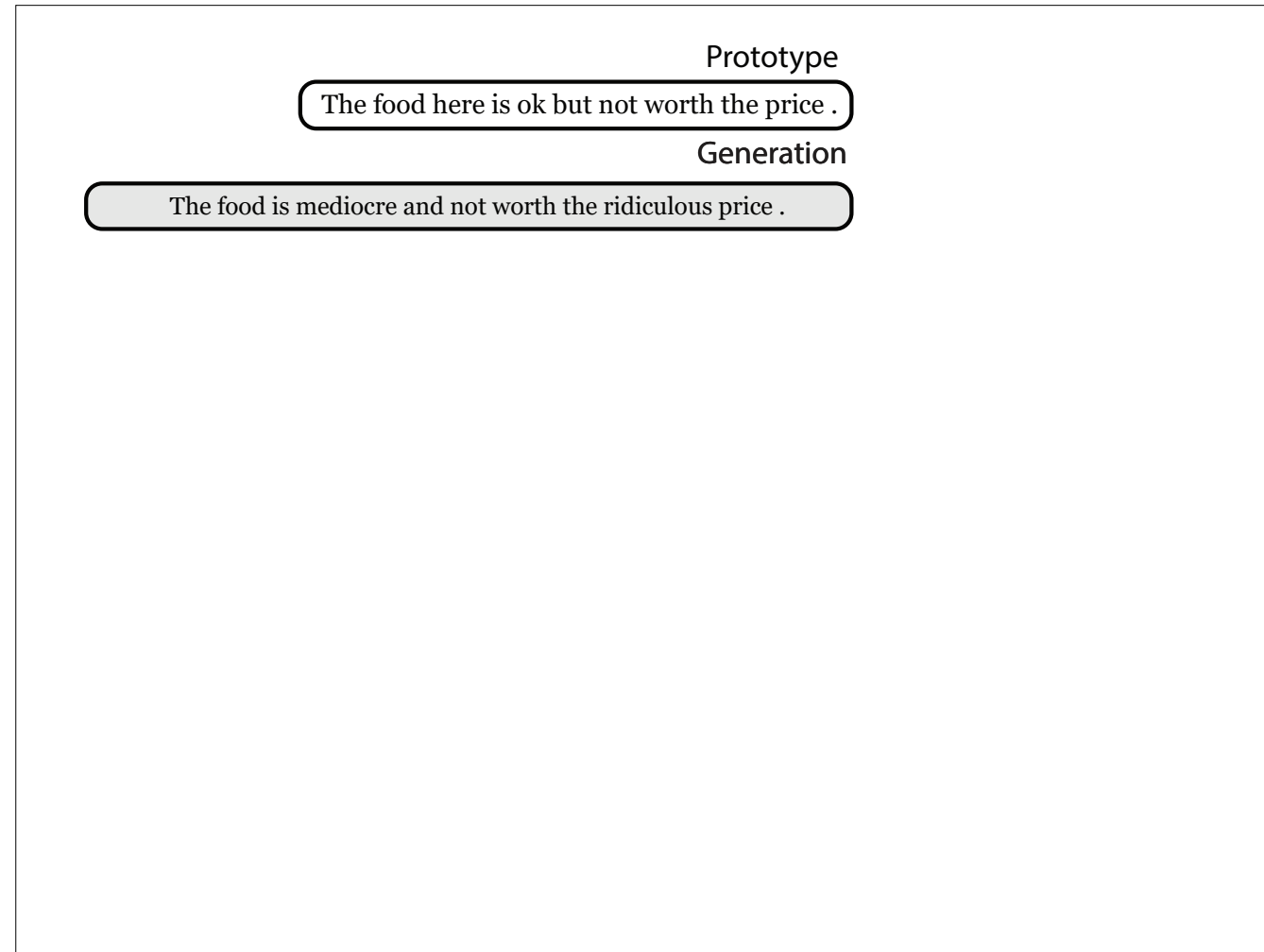
In plain English, the posterior is saying:
if I see both prototype sentence z_p and output y , what edit vector relates them?



Here is an illustration of $q(z_e)$.

So far, I have described a deterministic process to create an edit vector.
But what we want is an actual probability distribution over edit vectors.

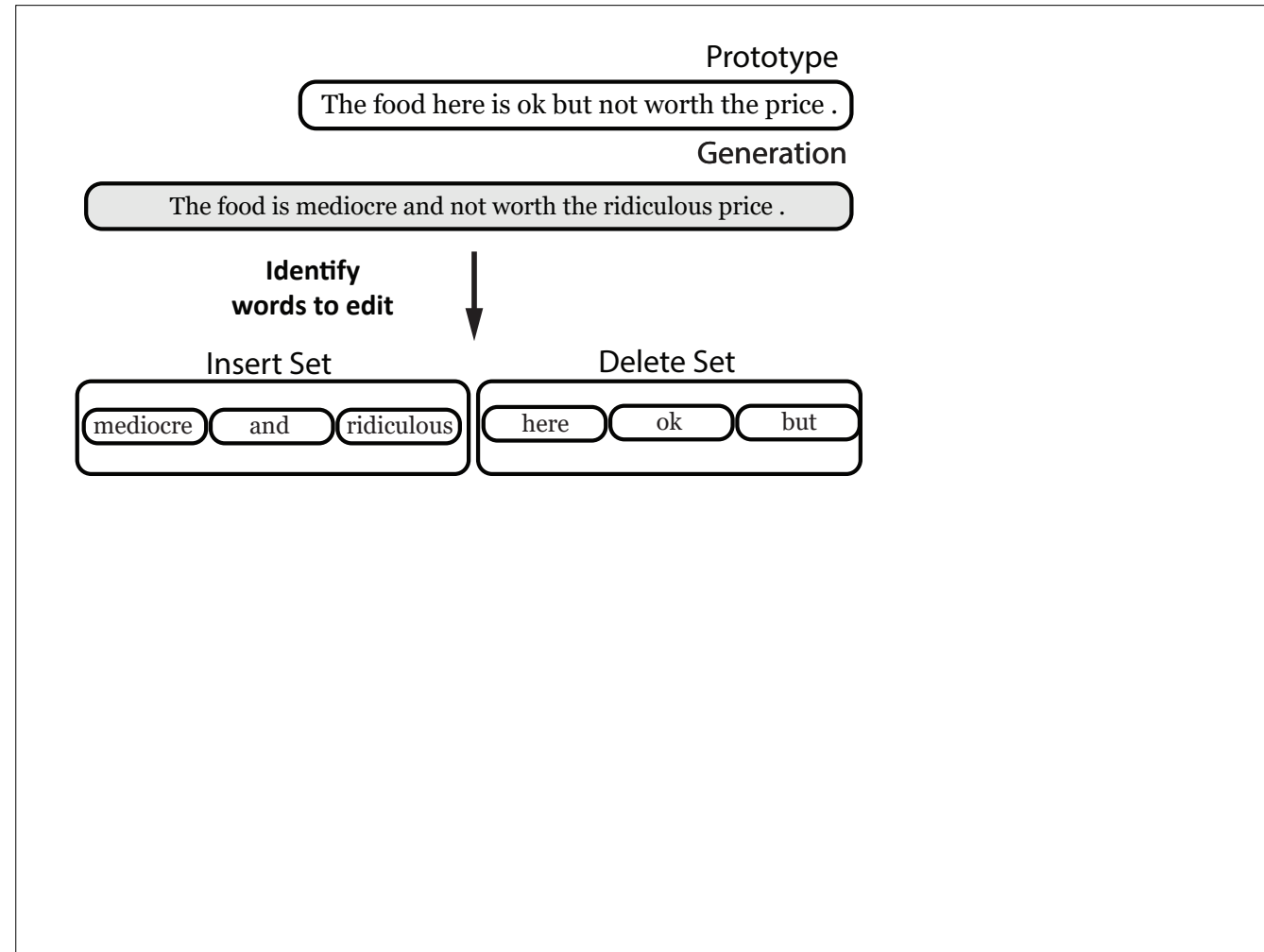
So, as a final step, we will add noise to get our final z_e .
For reference, let's call the pre-noise version \hat{z} .



Here is an illustration of $q(z_e)$.

So far, I have described a deterministic process to create an edit vector.
But what we want is an actual probability distribution over edit vectors.

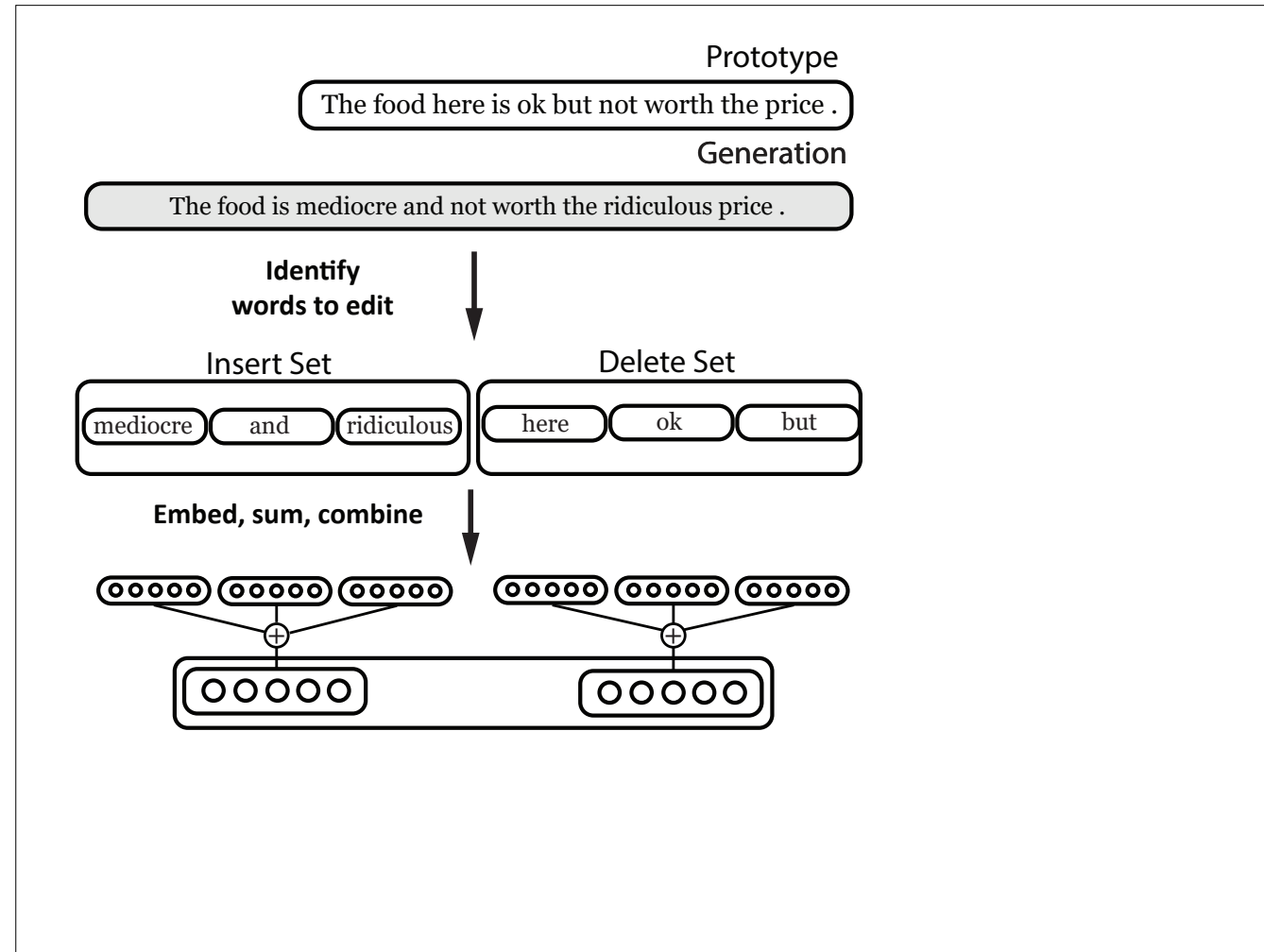
So, as a final step, we will add noise to get our final z_e .
For reference, let's call the pre-noise version \hat{z} .



Here is an illustration of $q(z_e)$.

So far, I have described a deterministic process to create an edit vector.
But what we want is an actual probability distribution over edit vectors.

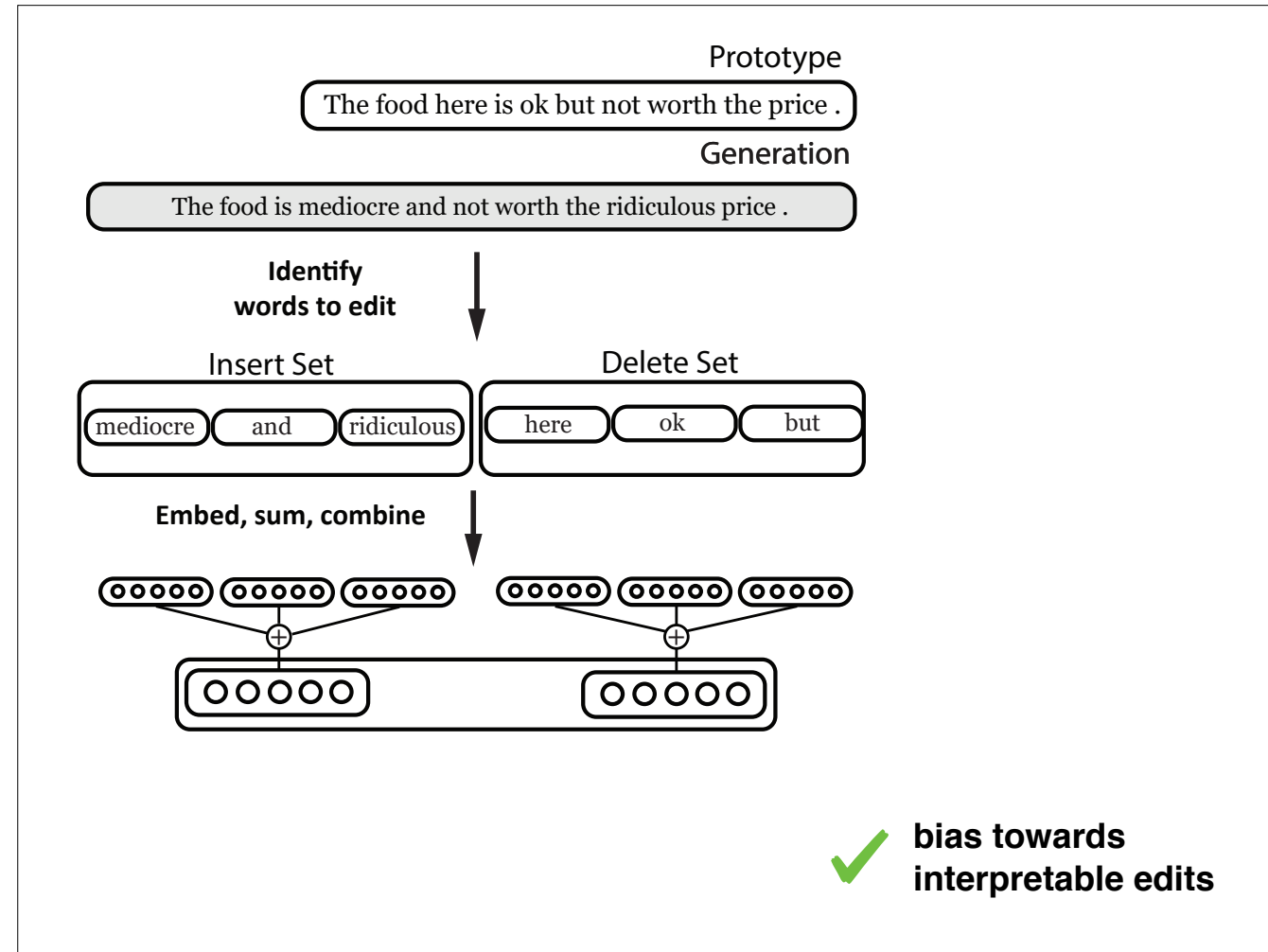
So, as a final step, we will add noise to get our final z_e .
For reference, let's call the pre-noise version z hat.



Here is an illustration of $q(z_e)$.

So far, I have described a deterministic process to create an edit vector.
But what we want is an actual probability distribution over edit vectors.

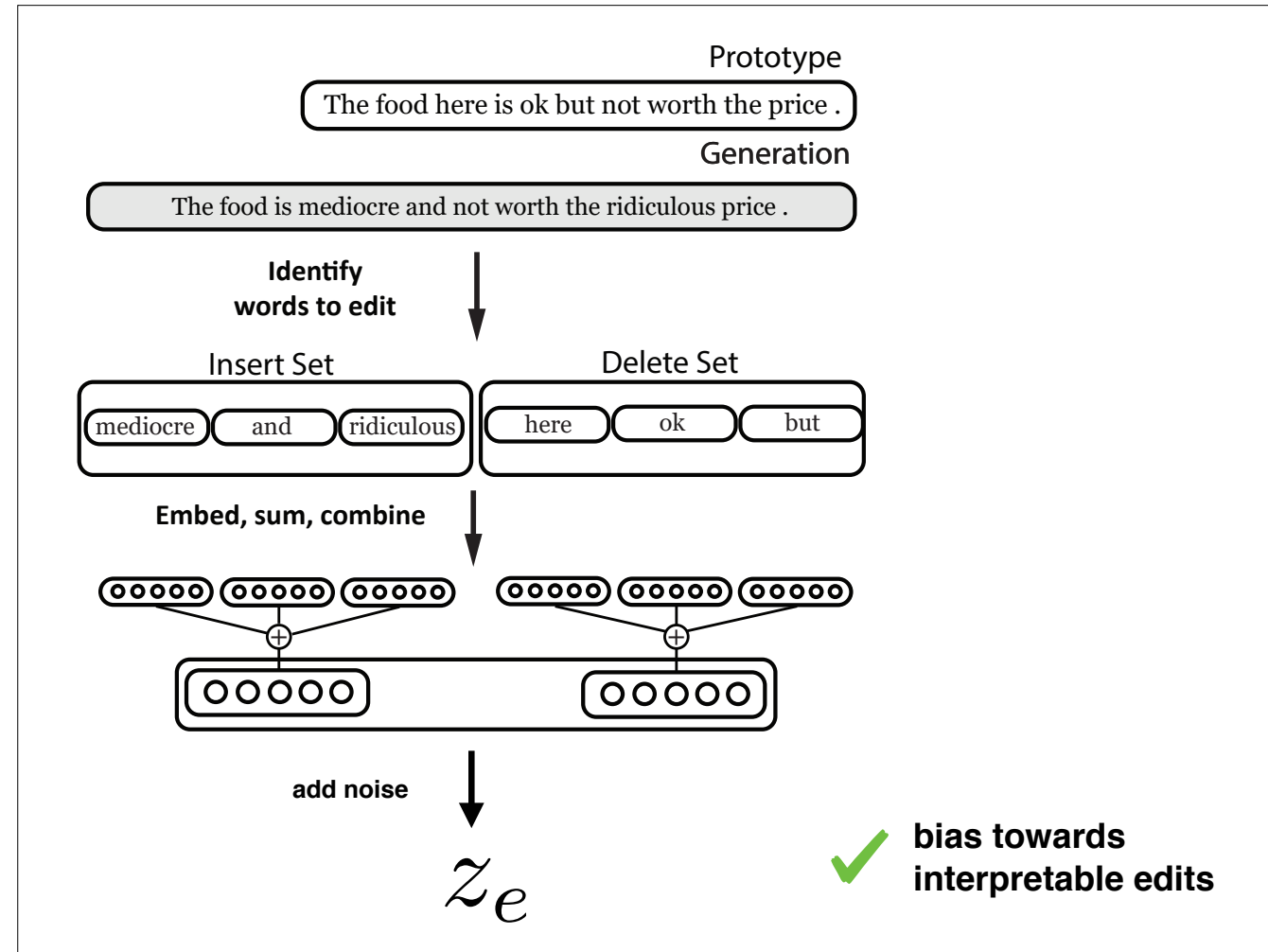
So, as a final step, we will add noise to get our final z_e .
For reference, let's call the pre-noise version \hat{z} .



Here is an illustration of $q(z_e)$.

So far, I have described a deterministic process to create an edit vector.
But what we want is an actual probability distribution over edit vectors.

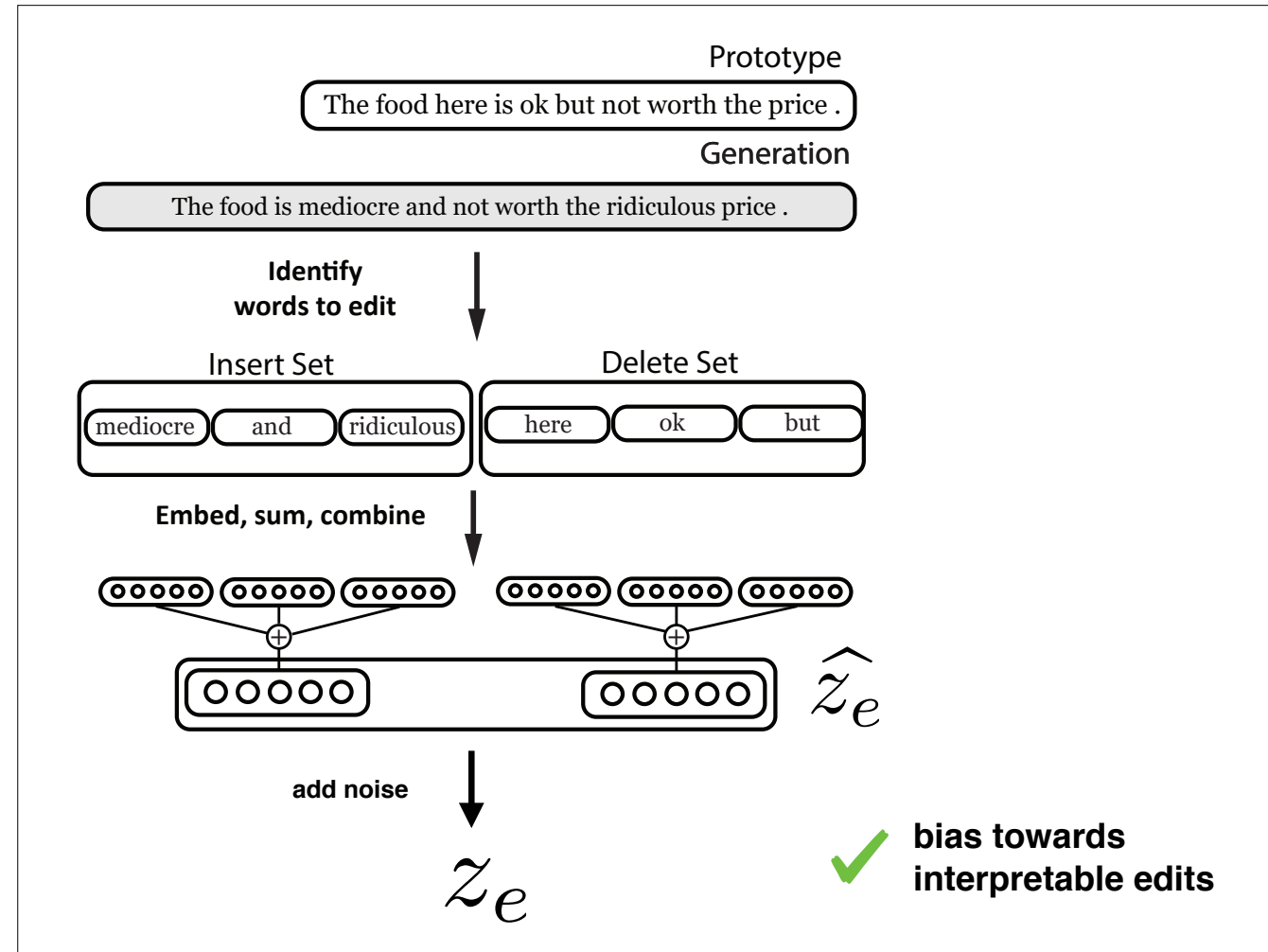
So, as a final step, we will add noise to get our final z_e .
For reference, let's call the pre-noise version \hat{z} .



Here is an illustration of $q(z_e)$.

So far, I have described a deterministic process to create an edit vector.
But what we want is an actual probability distribution over edit vectors.

So, as a final step, we will add noise to get our final z_e .
For reference, let's call the pre-noise version z hat.

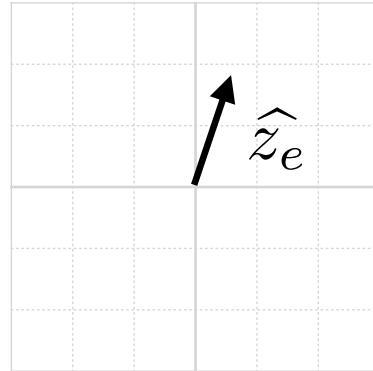


Here is an illustration of $q(z_e)$.

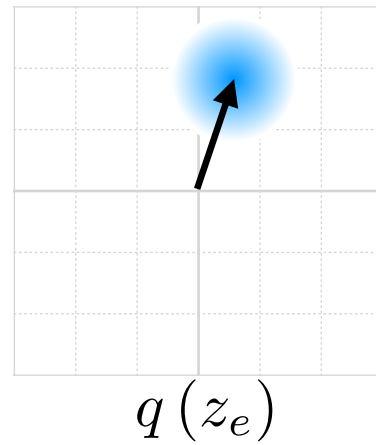
So far, I have described a deterministic process to create an edit vector. But what we want is an actual probability distribution over edit vectors.

So, as a final step, we will add noise to get our final z_e . For reference, let's call the pre-noise version $z_{\hat{e}}$.

How to add noise to \hat{z}_e ?



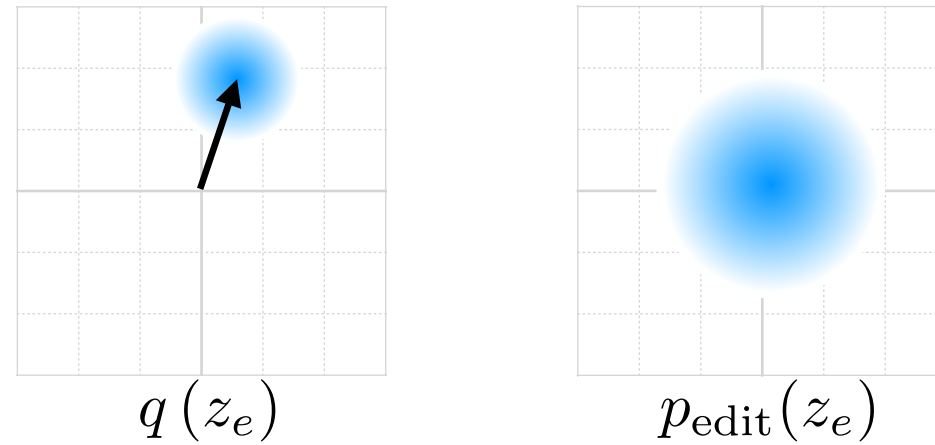
Standard choice (VAE): Gaussian



A standard approach to adding noise (as you would see in the literature on variational autoencoders) is to add Gaussian noise
And then define the prior over z to also be Gaussian.

This is computationally tractable because the reconstruction cost can be estimated using the reparameterization trick, and the KL penalty has a closed form.

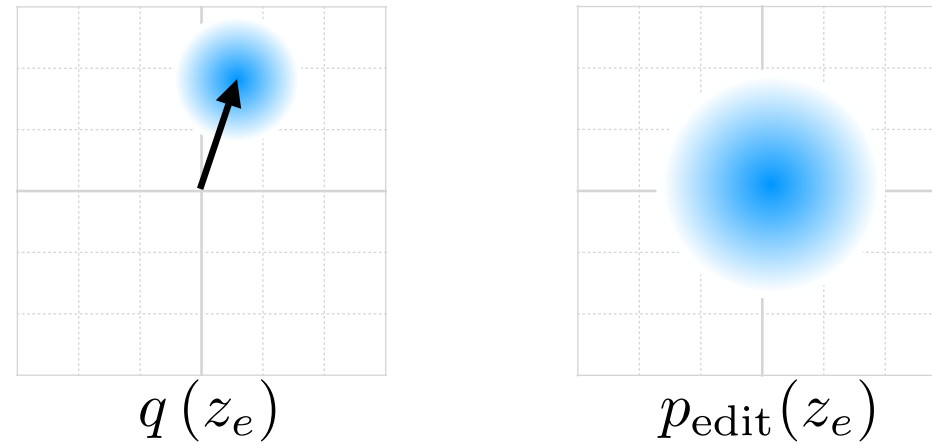
Standard choice (VAE): Gaussian



A standard approach to adding noise (as you would see in the literature on variational autoencoders) is to add Gaussian noise
And then define the prior over z to also be Gaussian.

This is computationally tractable because the reconstruction cost can be estimated using the reparameterization trick, and the KL penalty has a closed form.

Standard choice (VAE): Gaussian

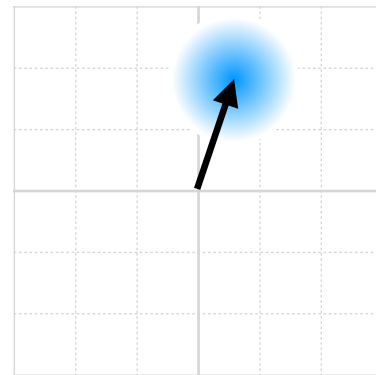


✓ **computationally tractable**

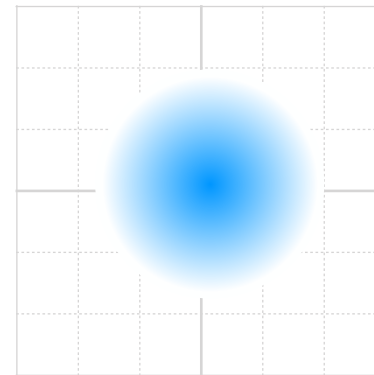
A standard approach to adding noise (as you would see in the literature on variational autoencoders) is to add Gaussian noise
And then define the prior over z to also be Gaussian.

This is computationally tractable because the reconstruction cost can be estimated using the reparameterization trick, and the KL penalty has a closed form.

Standard choice (VAE): Gaussian



$q(z_e)$



$p_{\text{edit}}(z_e)$

$$\text{ELBO} = \text{reconstruction_cost} - \text{KL_penalty}$$

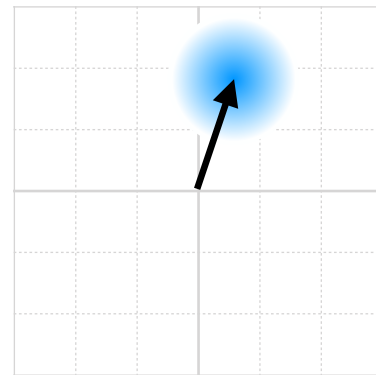


computationally tractable

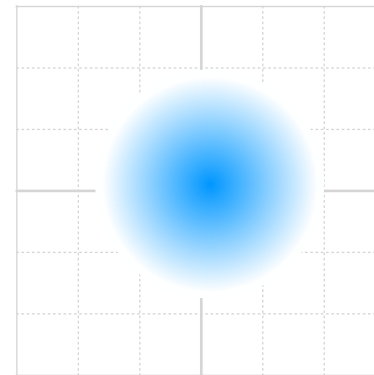
A standard approach to adding noise (as you would see in the literature on variational autoencoders) is to add Gaussian noise
And then define the prior over z to also be Gaussian.

This is computationally tractable because the reconstruction cost can be estimated using the reparameterization trick, and the KL penalty has a closed form.

Standard choice (VAE): Gaussian



$q(z_e)$



$p_{\text{edit}}(z_e)$

$$\text{ELBO} = \text{reconstruction_cost} - \text{KL_penalty}$$

reparameterization trick (VAEs)

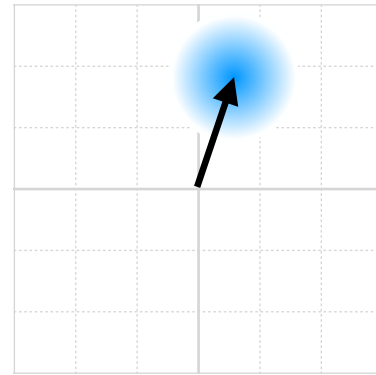


computationally tractable

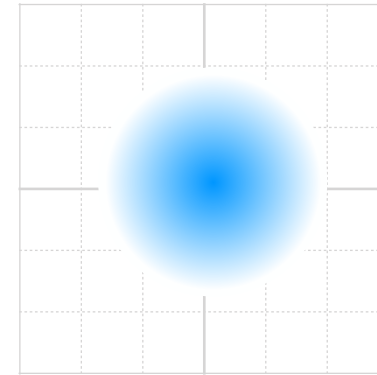
A standard approach to adding noise (as you would see in the literature on variational autoencoders) is to add Gaussian noise
And then define the prior over z to also be Gaussian.

This is computationally tractable because the reconstruction cost can be estimated using the reparameterization trick, and the KL penalty has a closed form.

Standard choice (VAE): Gaussian



$q(z_e)$



$p_{\text{edit}}(z_e)$

$$\text{ELBO} = \text{reconstruction_cost} - \text{KL_penalty}$$

reparameterization trick (VAEs)

(low-variance MC estimate of gradient)

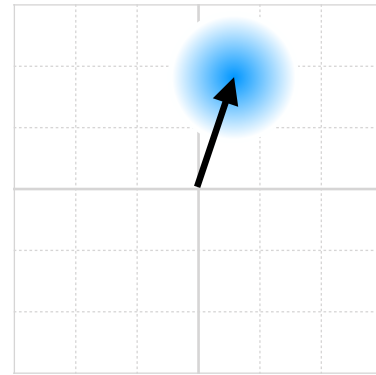


computationally tractable

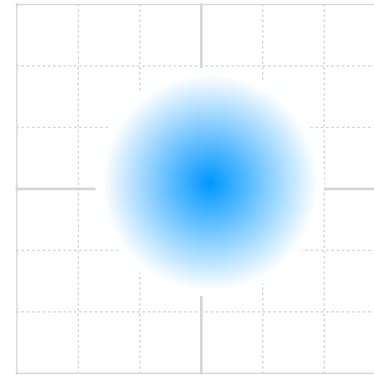
A standard approach to adding noise (as you would see in the literature on variational autoencoders) is to add Gaussian noise
And then define the prior over z to also be Gaussian.

This is computationally tractable because the reconstruction cost can be estimated using the reparameterization trick, and the KL penalty has a closed form.

Standard choice (VAE): Gaussian



$q(z_e)$



$p_{\text{edit}}(z_e)$

$$\text{ELBO} = \text{reconstruction_cost} - \text{KL_penalty}$$

reparameterization trick (VAEs)

(low-variance MC estimate of gradient)

closed form



computationally tractable

A standard approach to adding noise (as you would see in the literature on variational autoencoders) is to add Gaussian noise
And then define the prior over z to also be Gaussian.

This is computationally tractable because the reconstruction cost can be estimated using the reparameterization trick, and the KL penalty has a closed form.

The problem with a Gaussian prior

But we chose not to use a Gaussian Q distribution, because Gaussians have a problem in high dimensions.

In the low dimensional picture, this Gaussian seems well-behaved.
Most of the mass is near the origin, implying that we prefer small edit vectors.

In contrast, a high dimensional Gaussian would look more like this, where most of the probability mass is concentrated on a thin shell.

In fact, it is known that if you sample a point from a Gaussian, its distance from the center follows a Chi distribution.

I've plotted the chi distribution for different dimensions.

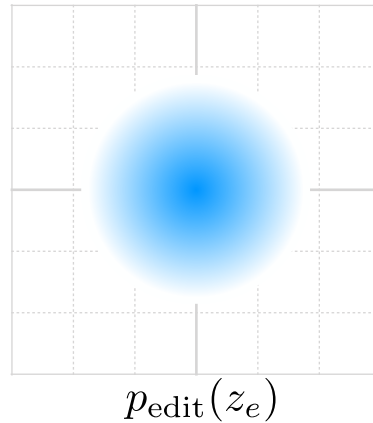
Recall that our edit vector is the sum of word vectors.

This means we are imposing a very heavy prior that the edit should change no more than 12 and no less than 8 words.

Instead, we would like something that is much more uniform over lengths.

The problem with a Gaussian prior

low-dim Gaussian



But we chose not to use a Gaussian Q distribution, because Gaussians have a problem in high dimensions.

In the low dimensional picture, this Gaussian seems well-behaved.
Most of the mass is near the origin, implying that we prefer small edit vectors.

In contrast, a high dimensional Gaussian would look more like this, where most of the probability mass is concentrated on a thin shell.

In fact, it is known that if you sample a point from a Gaussian, its distance from the center follows a Chi distribution.

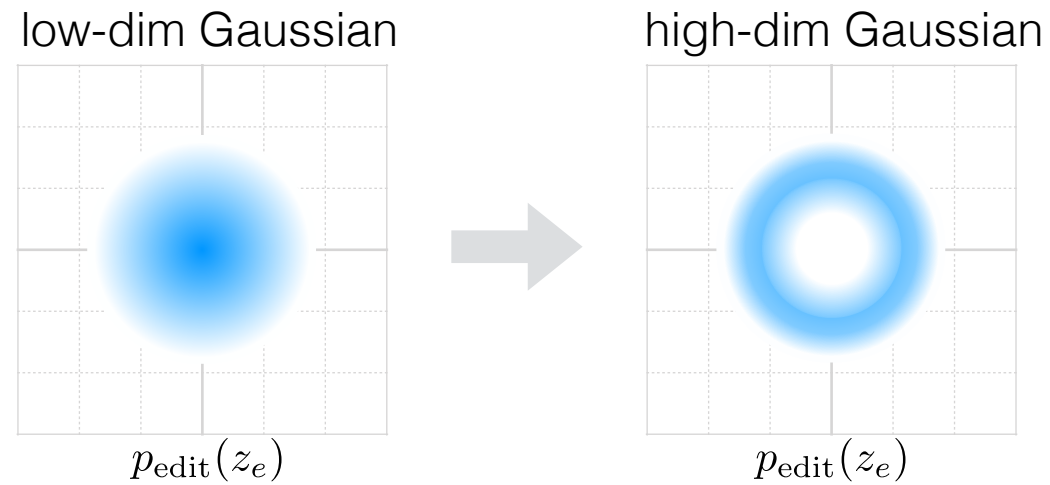
I've plotted the chi distribution for different dimensions.

Recall that our edit vector is the sum of word vectors.

This means we are imposing a very heavy prior that the edit should change no more than 12 and no less than 8 words.

Instead, we would like something that is much more uniform over lengths.

The problem with a Gaussian prior



But we chose not to use a Gaussian Q distribution, because Gaussians have a problem in high dimensions.

In the low dimensional picture, this Gaussian seems well-behaved.
Most of the mass is near the origin, implying that we prefer small edit vectors.

In contrast, a high dimensional Gaussian would look more like this, where most of the probability mass is concentrated on a thin shell.

In fact, it is known that if you sample a point from a Gaussian, its distance from the center follows a Chi distribution.

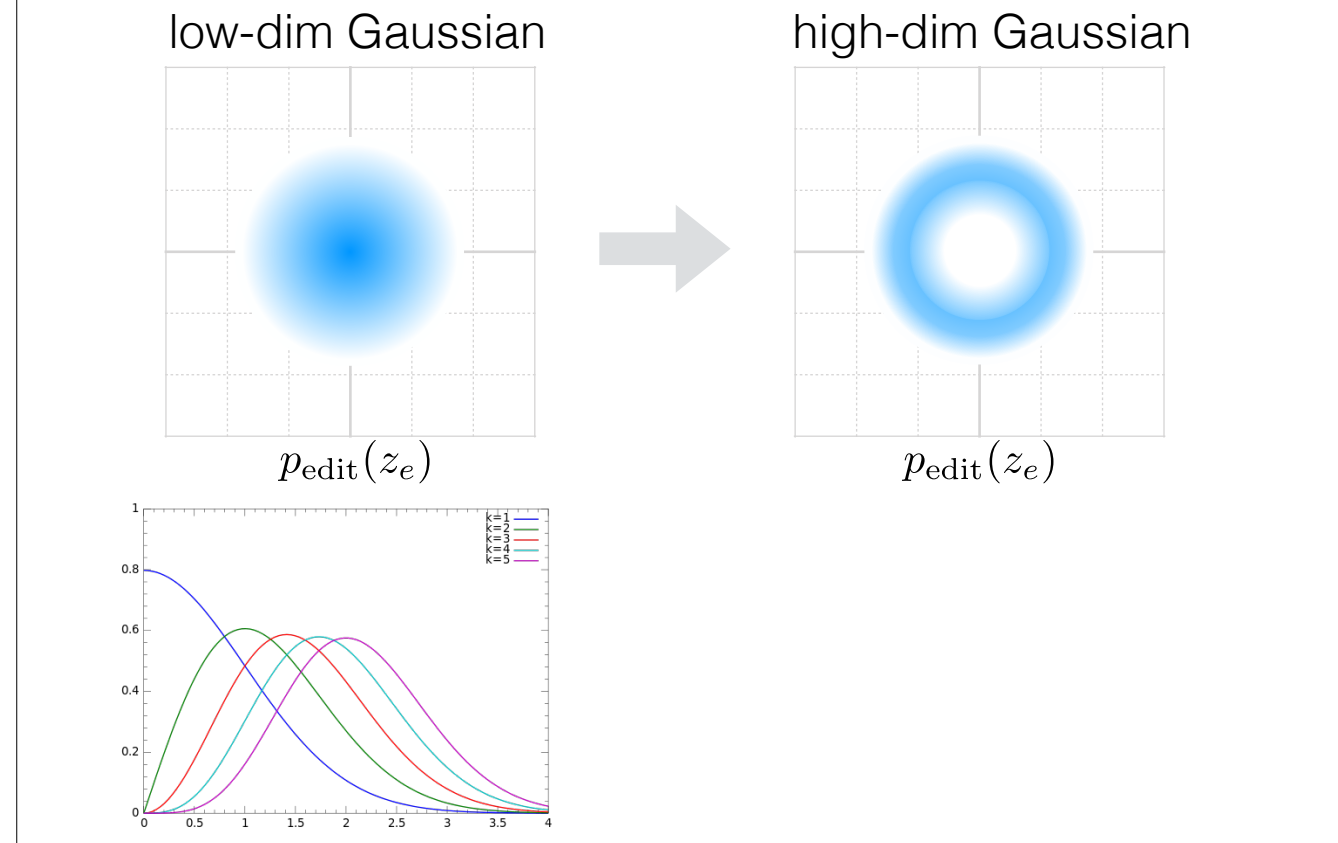
I've plotted the chi distribution for different dimensions.

Recall that our edit vector is the sum of word vectors.

This means we are imposing a very heavy prior that the edit should change no more than 12 and no less than 8 words.

Instead, we would like something that is much more uniform over lengths.

The problem with a Gaussian prior



But we chose not to use a Gaussian Q distribution, because Gaussians have a problem in high dimensions.

In the low dimensional picture, this Gaussian seems well-behaved.
Most of the mass is near the origin, implying that we prefer small edit vectors.

In contrast, a high dimensional Gaussian would look more like this, where most of the probability mass is concentrated on a thin shell.

In fact, it is known that if you sample a point from a Gaussian, its distance from the center follows a Chi distribution.

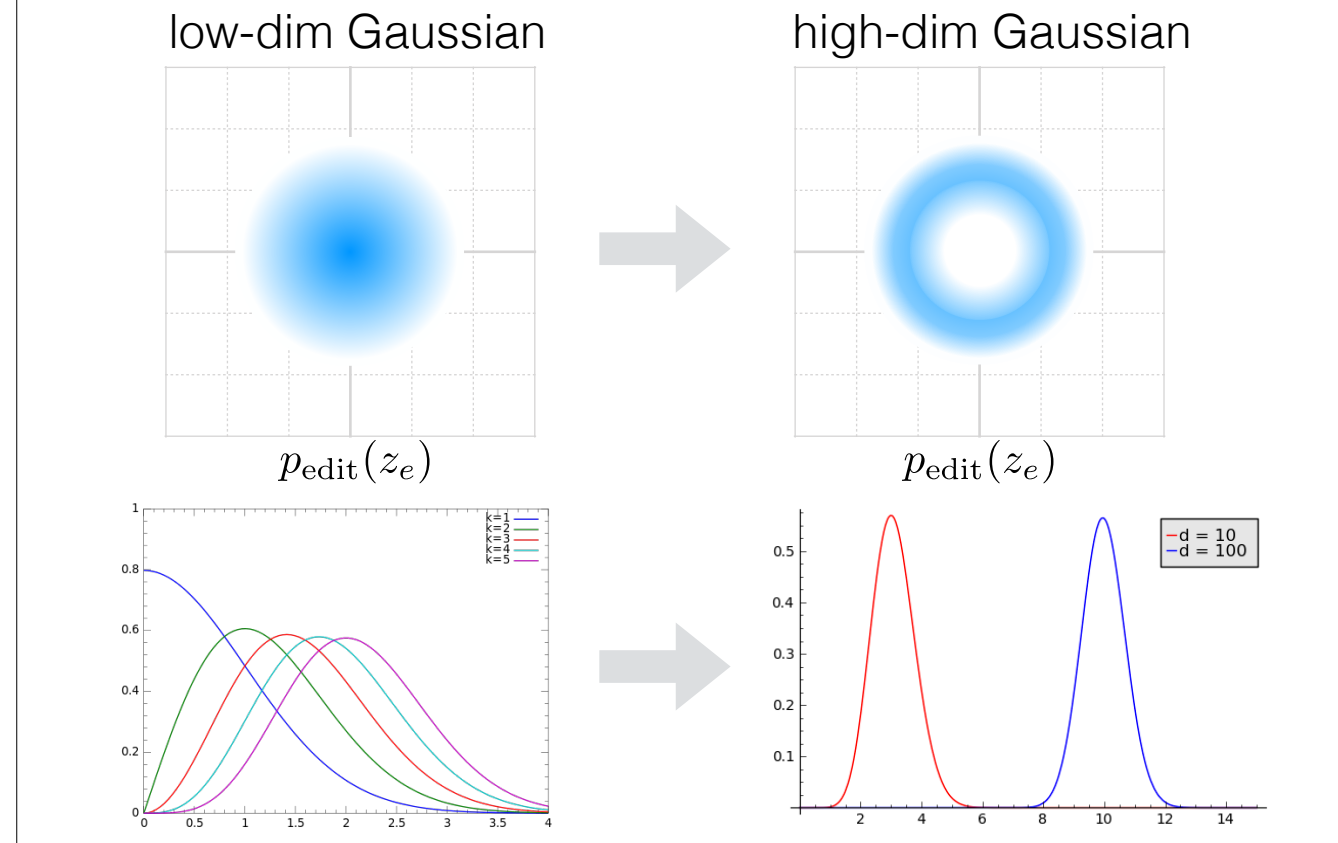
I've plotted the chi distribution for different dimensions.

Recall that our edit vector is the sum of word vectors.

This means we are imposing a very heavy prior that the edit should change no more than 12 and no less than 8 words.

Instead, we would like something that is much more uniform over lengths.

The problem with a Gaussian prior



But we chose not to use a Gaussian Q distribution, because Gaussians have a problem in high dimensions.

In the low dimensional picture, this Gaussian seems well-behaved.
Most of the mass is near the origin, implying that we prefer small edit vectors.

In contrast, a high dimensional Gaussian would look more like this, where most of the probability mass is concentrated on a thin shell.

In fact, it is known that if you sample a point from a Gaussian, its distance from the center follows a Chi distribution.

I've plotted the chi distribution for different dimensions.

Recall that our edit vector is the sum of word vectors.

This means we are imposing a very heavy prior that the edit should change no more than 12 and no less than 8 words.

Instead, we would like something that is much more uniform over lengths.

Better edit prior

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

To fix this problem...

First, we'll propose a different edit prior, which explicitly encodes a uniform distribution over the magnitude of the edit vector.

Then, we need to define a $q(\mathbf{z}_e)$ that is compatible with this edit prior.

We will see how to do that next.

Better edit prior

$$\text{mag} \sim \text{Unif}[0, 10]$$

y = output sentence **z**_p = prototype sentence **z**_e = edit vector

To fix this problem...

First, we'll propose a different edit prior, which explicitly encodes a uniform distribution over the magnitude of the edit vector.

Then, we need to define a $q(\mathbf{z}_e)$ that is compatible with this edit prior.

We will see how to do that next.

Better edit prior

$\text{mag} \sim \text{Unif}[0, 10]$

$\text{dir} \sim \text{unif. over sphere}$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

To fix this problem...

First, we'll propose a different edit prior, which explicitly encodes a uniform distribution over the magnitude of the edit vector.

Then, we need to define a $q(\mathbf{z}_e)$ that is compatible with this edit prior.

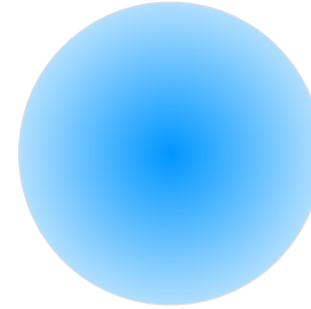
We will see how to do that next.

Better edit prior

$\text{mag} \sim \text{Unif}[0, 10]$

$\text{dir} \sim \text{unif. over sphere}$

$p_{\text{edit}}(z_e)$



\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

To fix this problem...

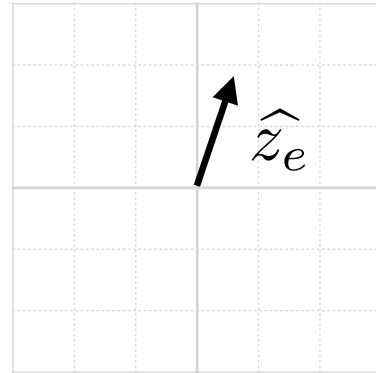
First, we'll propose a different edit prior, which explicitly encodes a uniform distribution over the magnitude of the edit vector.

Then, we need to define a $q(z_e)$ that is compatible with this edit prior.

We will see how to do that next.

Better edit prior

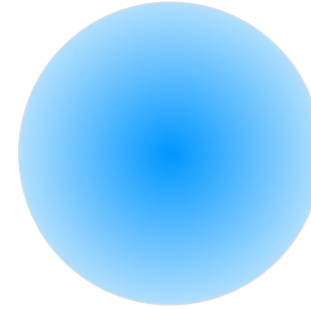
$q(z_e)?$



$\text{mag} \sim \text{Unif}[0, 10]$

$\text{dir} \sim \text{unif. over sphere}$

$p_{\text{edit}}(z_e)$



\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

To fix this problem...

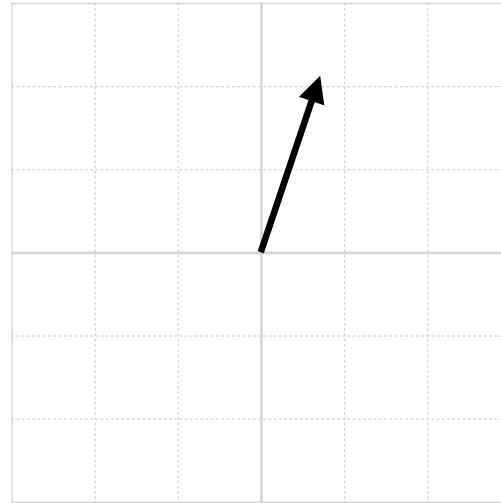
First, we'll propose a different edit prior, which explicitly encodes a uniform distribution over the magnitude of the edit vector.

Then, we need to define a $q(z_e)$ that is compatible with this edit prior.

We will see how to do that next.

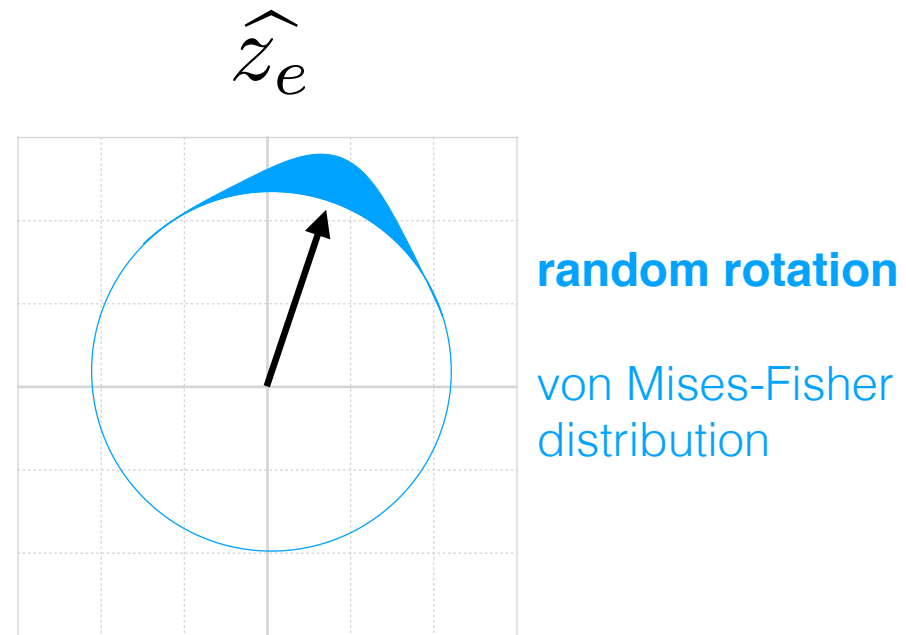
How to add noise to \hat{z}_e ?

\hat{z}_e



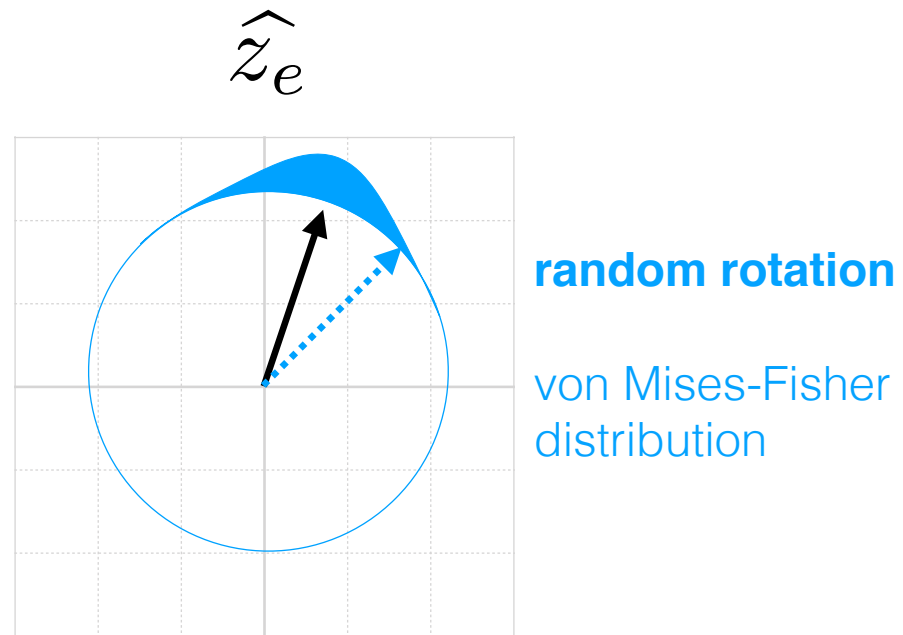
Starting with \hat{z}_e , we first randomly rotate it, using a von Mises Fisher distribution.

How to add noise to \hat{z}_e ?



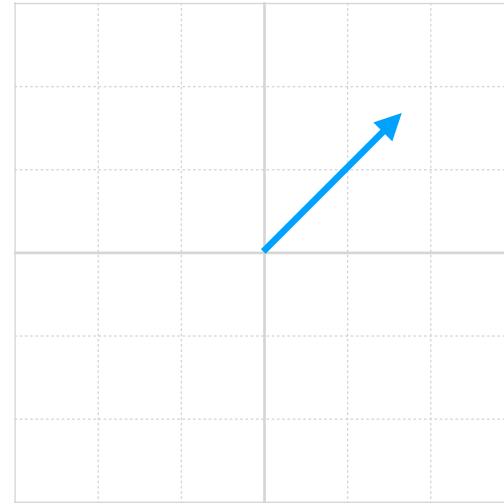
Starting with \hat{z}_e , we first randomly rotate it, using a von Mises Fisher distribution.

How to add noise to \hat{z}_e ?



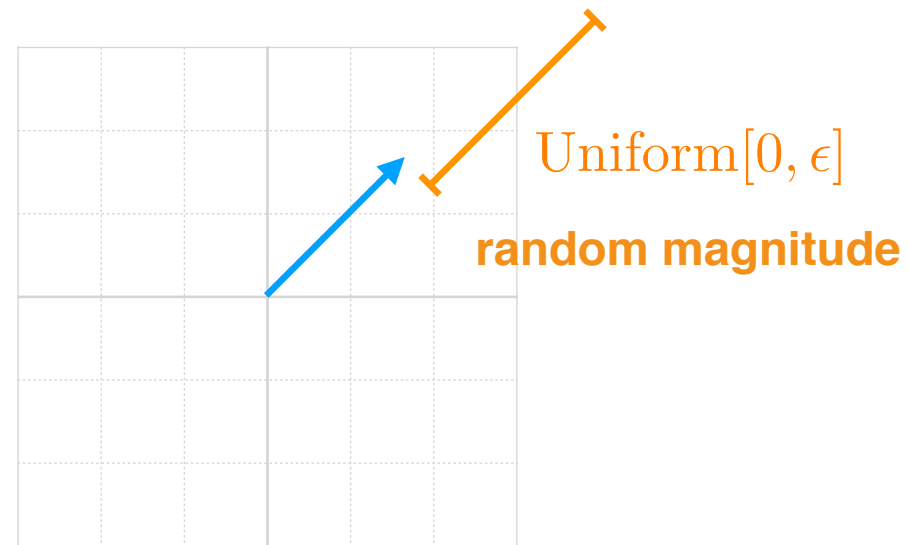
Starting with \hat{z}_e , we first randomly rotate it, using a von Mises Fisher distribution.

How to add noise to \hat{z}_e ?



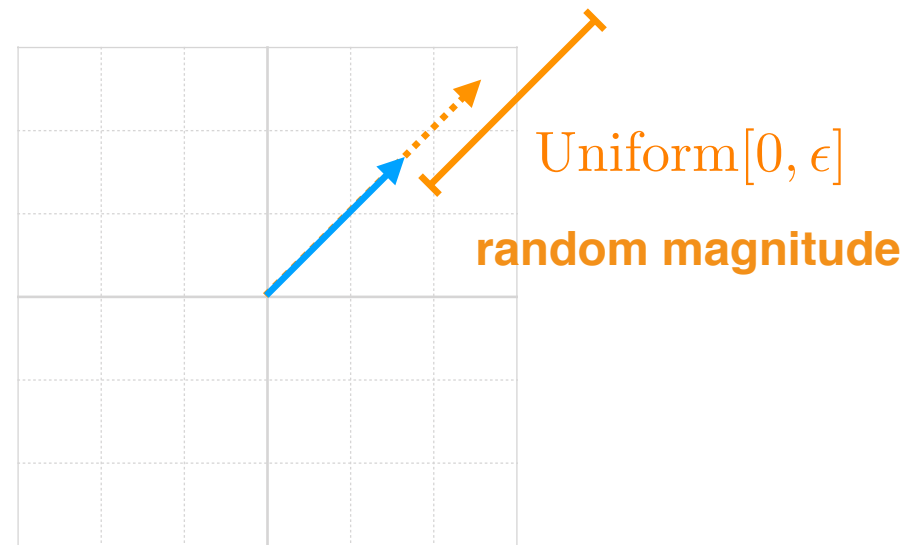
We then randomly perturb its magnitude

How to add noise to \hat{z}_e ?



We then randomly perturb its magnitude

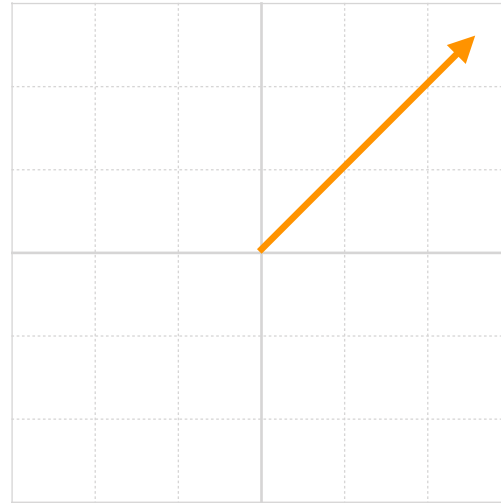
How to add noise to \hat{z}_e ?



We then randomly perturb its magnitude

How to add noise to \hat{z}_e ?

z_e



$q(z)$ over edits

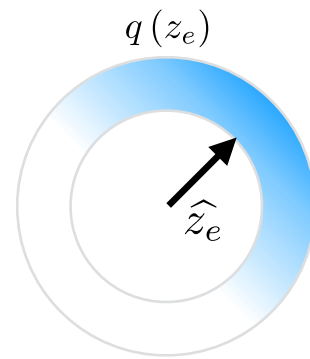
To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of z 's magnitude
and the direction is just a VMF perturbation of z 's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

You can still apply the reparameterization trick
and the KL penalty is now a constant

q(z) over edits



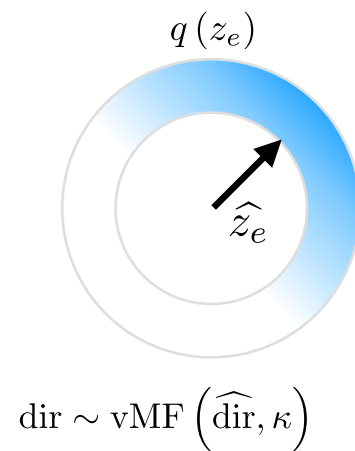
To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of \hat{z} 's magnitude
and the direction is just a VMF perturbation of \hat{z} 's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

You can still apply the reparameterization trick
and the KL penalty is now a constant

q(z) over edits



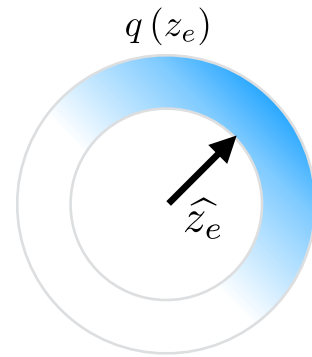
To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of zhat's magnitude
and the direction is just a VMF perturbation of zhat's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

You can still apply the reparameterization trick
and the KL penalty is now a constant

q(z) over edits



$$\text{dir} \sim \text{vMF}(\hat{\text{dir}}, \kappa)$$

$$\text{mag} \sim \text{Unif}[\widehat{\text{mag}}, \widehat{\text{mag}} + \epsilon]$$

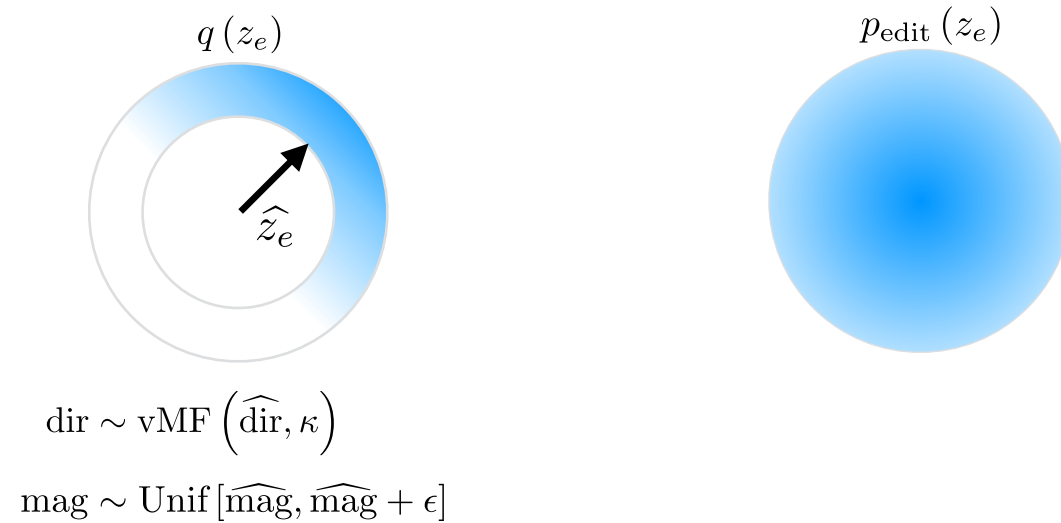
To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of zhat's magnitude
and the direction is just a VMF perturbation of zhat's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

You can still apply the reparameterization trick
and the KL penalty is now a constant

q(z) over edits



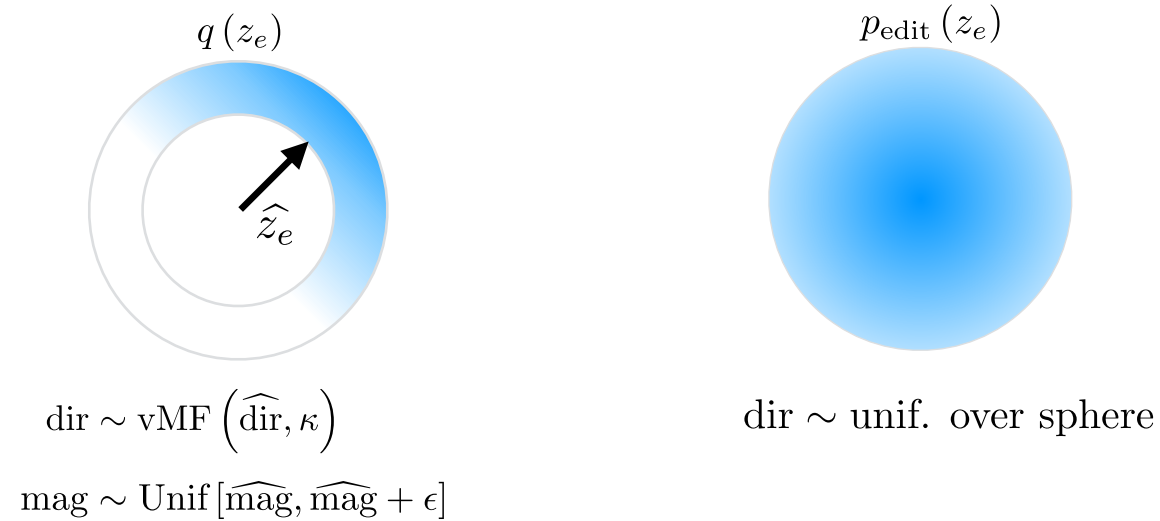
To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of zhat's magnitude
and the direction is just a VMF perturbation of zhat's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

You can still apply the reparameterization trick
and the KL penalty is now a constant

q(z) over edits



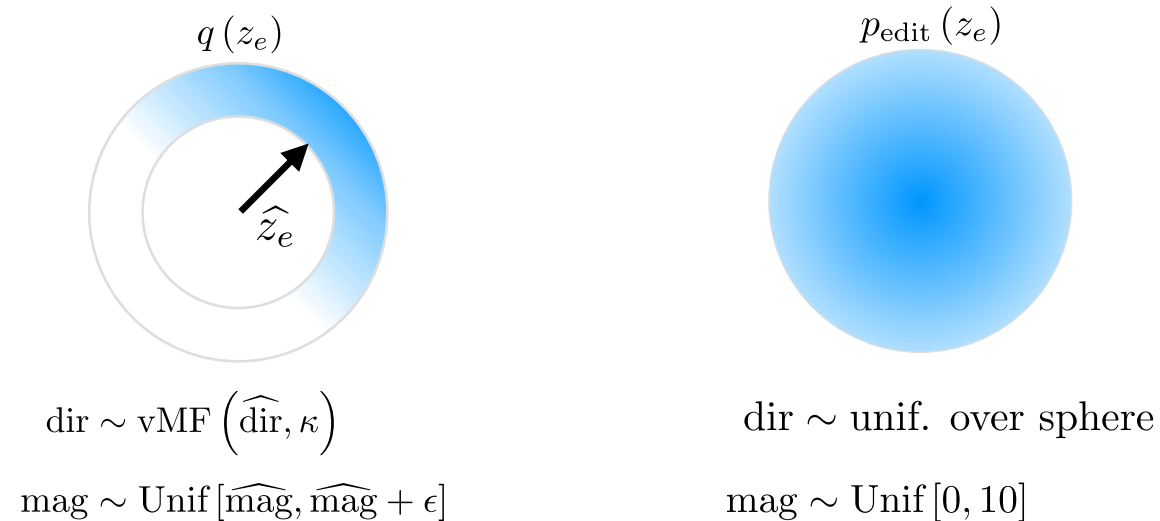
To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of \hat{z} 's magnitude
and the direction is just a VMF perturbation of \hat{z} 's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

You can still apply the reparameterization trick
and the KL penalty is now a constant

q(z) over edits



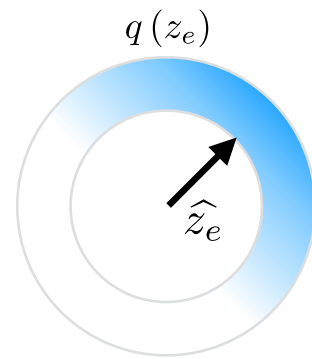
To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of zhat's magnitude
and the direction is just a VMF perturbation of zhat's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

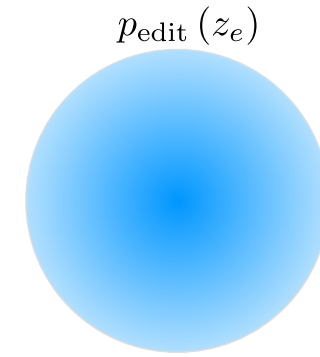
You can still apply the reparameterization trick
and the KL penalty is now a constant

q(z) over edits



$$\text{dir} \sim \text{vMF}(\hat{\text{dir}}, \kappa)$$

$$\text{mag} \sim \text{Unif}[\widehat{\text{mag}}, \widehat{\text{mag}} + \epsilon]$$



$$\text{dir} \sim \text{unif. over sphere}$$

$$\text{mag} \sim \text{Unif}[0, 10]$$

$$\text{ELBO} = \text{reconstruction_cost} - \text{KL_penalty}$$

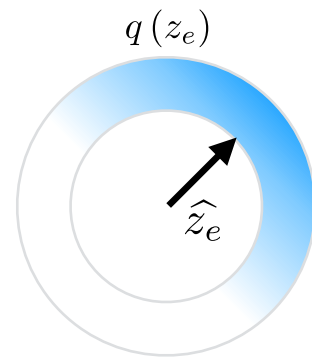
To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of zhat's magnitude
and the direction is just a VMF perturbation of zhat's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

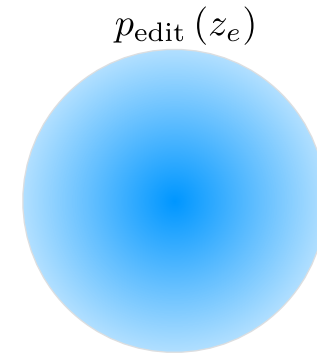
You can still apply the reparameterization trick
and the KL penalty is now a constant

q(z) over edits



$$\text{dir} \sim \text{vMF}(\hat{\text{dir}}, \kappa)$$

$$\text{mag} \sim \text{Unif}[\widehat{\text{mag}}, \widehat{\text{mag}} + \epsilon]$$



$$\text{dir} \sim \text{unif. over sphere}$$

$$\text{mag} \sim \text{Unif}[0, 10]$$

$$\text{ELBO} = \text{reconstruction_cost} - \text{KL_penalty}$$

reparameterization trick (VAEs)

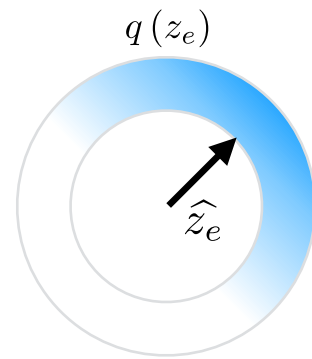
To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of zhat's magnitude
and the direction is just a VMF perturbation of zhat's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

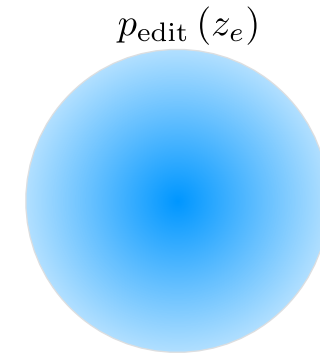
You can still apply the reparameterization trick
and the KL penalty is now a constant

q(z) over edits



$$\text{dir} \sim \text{vMF}(\hat{\text{dir}}, \kappa)$$

$$\text{mag} \sim \text{Unif}[\widehat{\text{mag}}, \widehat{\text{mag}} + \epsilon]$$



$$\text{dir} \sim \text{unif. over sphere}$$

$$\text{mag} \sim \text{Unif}[0, 10]$$

$$\text{ELBO} = \text{reconstruction_cost} - \text{KL_penalty}$$

reparameterization trick (VAEs) just a constant

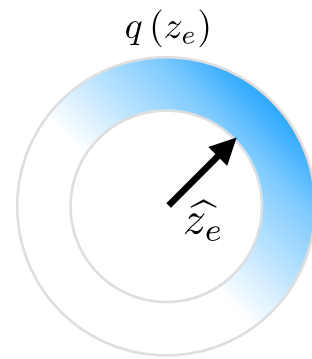
To sum that up, we now have a q distribution which looks like this (left)
 the magnitude is just a uniform perturbation of zhat's magnitude
 and the direction is just a VMF perturbation of zhat's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
 in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
 and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

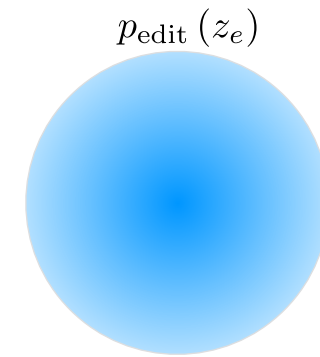
You can still apply the reparameterization trick
 and the KL penalty is now a constant

q(z) over edits



$$\text{dir} \sim \text{vMF}(\widehat{\text{dir}}, \kappa)$$

$$\text{mag} \sim \text{Unif}[\widehat{\text{mag}}, \widehat{\text{mag}} + \epsilon]$$



$$\text{dir} \sim \text{unif. over sphere}$$

$$\text{mag} \sim \text{Unif}[0, 10]$$

$$\text{ELBO} = \text{reconstruction_cost} - \text{KL_penalty}$$

reparameterization trick (VAEs) just a constant

✓ **computationally tractable**

To sum that up, we now have a q distribution which looks like this (left)
the magnitude is just a uniform perturbation of zhat's magnitude
and the direction is just a VMF perturbation of zhat's direction

And we will choose our edit prior to look like this, so that it actually concentrates mass near the origin
in particular, we'll draw a random magnitude uniform 0, 10 (guaranteeing spread out magnitudes)
and we'll draw a direction uniformly over the sphere

This new Q function still has all the nice properties that the standard Gaussian would give you

You can still apply the reparameterization trick
and the KL penalty is now a constant

Summary of training

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

That was a lot of lower bounds, but I want to emphasize that the final training procedure is fairly straightforward.

Summary of training

- Build a training set of lexically similar sentence pairs (\mathbf{z}_p , \mathbf{y})

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

That was a lot of lower bounds, but I want to emphasize that the final training procedure is fairly straightforward.

Summary of training

- Build a training set of lexically similar sentence pairs (\mathbf{z}_p , \mathbf{y})
- For each pair of sentences (\mathbf{z}_p , \mathbf{y})

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

That was a lot of lower bounds, but I want to emphasize that the final training procedure is fairly straightforward.

Summary of training

- Build a training set of lexically similar sentence pairs (\mathbf{z}_p , \mathbf{y})
- For each pair of sentences (\mathbf{z}_p , \mathbf{y})
 1. identify words that differ between \mathbf{z}_p and \mathbf{y}

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

That was a lot of lower bounds, but I want to emphasize that the final training procedure is fairly straightforward.

Summary of training

- Build a training set of lexically similar sentence pairs (\mathbf{z}_p , \mathbf{y})
- For each pair of sentences (\mathbf{z}_p , \mathbf{y})
 1. identify words that differ between \mathbf{z}_p and \mathbf{y}
 2. embed those words into a vector

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

That was a lot of lower bounds, but I want to emphasize that the final training procedure is fairly straightforward.

Summary of training

- Build a training set of lexically similar sentence pairs (\mathbf{z}_p , \mathbf{y})
- For each pair of sentences (\mathbf{z}_p , \mathbf{y})
 1. identify words that differ between \mathbf{z}_p and \mathbf{y}
 2. embed those words into a vector
 3. add noise to get edit vector \mathbf{z}_e

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

That was a lot of lower bounds, but I want to emphasize that the final training procedure is fairly straightforward.

Summary of training

- Build a training set of lexically similar sentence pairs (\mathbf{z}_p , \mathbf{y})
- For each pair of sentences (\mathbf{z}_p , \mathbf{y})
 1. identify words that differ between \mathbf{z}_p and \mathbf{y}
 2. embed those words into a vector
 3. add noise to get edit vector \mathbf{z}_e
 4. train seq2seq mapping (\mathbf{z}_p , \mathbf{z}_e) \rightarrow \mathbf{y} $p_{\text{editor}}(y \mid z_p, z_e)$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

That was a lot of lower bounds, but I want to emphasize that the final training procedure is fairly straightforward.

Summary of training

- Build a training set of lexically similar sentence pairs (\mathbf{z}_p , \mathbf{y})
- For each pair of sentences (\mathbf{z}_p , \mathbf{y})
 1. identify words that differ between \mathbf{z}_p and \mathbf{y}
 2. embed those words into a vector
 3. add noise to get edit vector \mathbf{z}_e
 4. train seq2seq mapping (\mathbf{z}_p , \mathbf{z}_e) $\rightarrow \mathbf{y}$ $p_{\text{editor}}(y \mid z_p, z_e)$
 5. update $\mathbf{q}(\mathbf{z}_e)$

\mathbf{y} = output sentence \mathbf{z}_p = prototype sentence \mathbf{z}_e = edit vector

That was a lot of lower bounds, but I want to emphasize that the final training procedure is fairly straightforward.

\end{\bstrong}

```
\begin{Results}
```

Now we can see how this stuff actually works.

i had the fried whitefish taco which was decent, but i've had much better.	i had the <unk> and the fried carnitas tacos, it was pretty tasty, but i've had better.
"hash browns" are unseasoned, frozen potato shreds burnt to a crisp on the outside and mushy on the inside.	the hash browns were crispy on the outside, but still the taste was missing.
i'm not sure what is preventing me from giving it <cardinal> stars, but i probably should.	i'm currently giving <cardinal> stars for the service alone.
quick place to grab light and tasty teriyaki.	this place is good and a quick place to grab a tasty sandwich.
sad part is we've been there before and its been good.	i've been here several times and always have a good time.

Before diving into the results, I want to show you some raw generations from the model.

Note that these are just for randomly sampled edit vectors.

Later on, we will consider edit vectors that we directly control, which allow for more precise editing behavior.

Prototype z_p

i had the fried whitefish taco which was decent, but i've had much better.	i had the <unk> and the fried carnitas tacos, it was pretty tasty, but i've had better.
"hash browns" are unseasoned, frozen potato shreds burnt to a crisp on the outside and mushy on the inside.	the hash browns were crispy on the outside, but still the taste was missing.
i'm not sure what is preventing me from giving it <cardinal> stars, but i probably should.	i'm currently giving <cardinal> stars for the service alone.
quick place to grab light and tasty teriyaki.	this place is good and a quick place to grab a tasty sandwich.
sad part is we've been there before and its been good.	i've been here several times and always have a good time.

Before diving into the results, I want to show you some raw generations from the model.

Note that these are just for randomly sampled edit vectors.

Later on, we will consider edit vectors that we directly control, which allow for more precise editing behavior.

Prototype z_p

Output y

i had the fried whitefish taco which was decent, but i've had much better.	i had the <unk> and the fried car-nitas tacos, it was pretty tasty, but i've had better.
"hash browns" are unseasoned, frozen potato shreds burnt to a crisp on the outside and mushy on the inside.	the hash browns were crispy on the outside, but still the taste was missing.
i'm not sure what is preventing me from giving it <cardinal> stars, but i probably should.	i'm currently giving <cardinal> stars for the service alone.
quick place to grab light and tasty teriyaki.	this place is good and a quick place to grab a tasty sandwich.
sad part is we've been there before and its been good.	i've been here several times and al-ways have a good time.

Before diving into the results, I want to show you some raw generations from the model.

Note that these are just for randomly sampled edit vectors.

Later on, we will consider edit vectors that we directly control, which allow for more precise editing behavior.

Prototype \mathbf{z}_p (random edit vector)

Output \mathbf{y}

i had the fried whitefish taco which was decent, but i've had much better.	i had the <unk> and the fried car-nitas tacos, it was pretty tasty, but i've had better.
"hash browns" are unseasoned, frozen potato shreds burnt to a crisp on the outside and mushy on the inside.	the hash browns were crispy on the outside, but still the taste was missing.
i'm not sure what is preventing me from giving it <cardinal> stars, but i probably should.	i'm currently giving <cardinal> stars for the service alone.
quick place to grab light and tasty teriyaki.	this place is good and a quick place to grab a tasty sandwich.
sad part is we've been there before and its been good.	i've been here several times and al-ways have a good time.

Before diving into the results, I want to show you some raw generations from the model.

Note that these are just for randomly sampled edit vectors.

Later on, we will consider edit vectors that we directly control, which allow for more precise editing behavior.

Overview of results

- **More diverse generations**
- **Higher quality generations**
- **Better perplexity** (BillionWord, Yelp reviews)
- **Edits are semantically meaningful**
 - preserve semantic similarity
 - can be used to perform sentence-level analogies

Here is the summary of the results I showed you earlier

We're going to start with semantics, because that is most interesting.

Overview of results

- **More diverse generations**
- **Higher quality generations**
- **Better perplexity** (BillionWord, Yelp reviews)
- ✓ **Edits are semantically meaningful**
 - preserve semantic similarity
 - can be used to perform sentence-level analogies

Here is the summary of the results I showed you earlier

We're going to start with semantics, because that is most interesting.

Edits are semantically meaningful

$$y \sim p_{\text{editor}}(y \mid z_p, z_e)$$

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

As described earlier, one sub-component of our model is an editor, which transforms one sentence into another.

A cool feature is that we can actually plug in any edit vector we want, and control the direction of the edits.

Up next, I'll be seeing just how much control we have.

Edits are semantically meaningful

$$y \sim p_{\text{editor}}(y \mid z_p, z_e)$$



plug in your own edit vector!

y = output sentence **z**_p = prototype sentence **z**_e = edit vector

As described earlier, one sub-component of our model is an editor, which transforms one sentence into another.

A cool feature is that we can actually plug in any edit vector we want, and control the direction of the edits.

Up next, I'll be seeing just how much control we have.

Edits are semantically meaningful

$$y \sim p_{\text{editor}}(y \mid z_p, z_e)$$



semantic control

plug in your own edit vector!

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

As described earlier, one sub-component of our model is an editor, which transforms one sentence into another.

A cool feature is that we can actually plug in any edit vector we want, and control the direction of the edits.

Up next, I'll be seeing just how much control we have.

Edits are semantically meaningful

$$y \sim p_{\text{editor}}(y \mid z_p, z_e)$$



semantic control

plug in your own edit vector!

semantic smoothness:

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

As described earlier, one sub-component of our model is an editor, which transforms one sentence into another.

A cool feature is that we can actually plug in any edit vector we want, and control the direction of the edits.

Up next, I'll be seeing just how much control we have.

Edits are semantically meaningful

$$y \sim p_{\text{editor}}(y \mid z_p, z_e)$$



semantic control

plug in your own edit vector!

semantic smoothness:

small magnitude edit vector should cause small changes

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

As described earlier, one sub-component of our model is an editor, which transforms one sentence into another.

A cool feature is that we can actually plug in any edit vector we want, and control the direction of the edits.

Up next, I'll be seeing just how much control we have.

Edits are semantically meaningful

$$y \sim p_{\text{editor}}(y \mid z_p, z_e)$$



semantic control

plug in your own edit vector!

semantic smoothness:

small magnitude edit vector should cause small changes

consistent edit behavior:

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

As described earlier, one sub-component of our model is an editor, which transforms one sentence into another.

A cool feature is that we can actually plug in any edit vector we want, and control the direction of the edits.

Up next, I'll be seeing just how much control we have.

Edits are semantically meaningful

$$y \sim p_{\text{editor}}(y \mid z_p, z_e)$$



semantic control

plug in your own edit vector!

semantic smoothness:

small magnitude edit vector should cause small changes

consistent edit behavior:

apply the same edit vector to different sentences

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

As described earlier, one sub-component of our model is an editor, which transforms one sentence into another.

A cool feature is that we can actually plug in any edit vector we want, and control the direction of the edits.

Up next, I'll be seeing just how much control we have.

Edits are semantically meaningful

$$y \sim p_{\text{editor}}(y \mid z_p, z_e)$$



semantic control

plug in your own edit vector!

semantic smoothness:

small magnitude edit vector should cause small changes

consistent edit behavior:

apply the same edit vector to different sentences
should cause semantically analogous edits

y = output sentence **z_p** = prototype sentence **z_e** = edit vector

As described earlier, one sub-component of our model is an editor, which transforms one sentence into another.

A cool feature is that we can actually plug in any edit vector we want, and control the direction of the edits.

Up next, I'll be seeing just how much control we have.

Semantic smoothness

What we will do is use the editor to take a random walk in sentence space.

We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness

**random walk in
sentence space**

What we will do is use the editor to take a random walk in sentence space.

We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness

**random walk in
sentence space**

●
 z_p

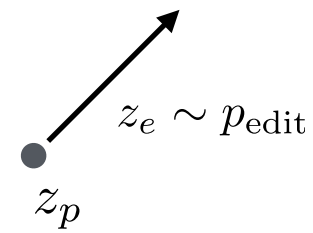
What we will do is use the editor to take a random walk in sentence space.

We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence
And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



**random walk in
sentence space**

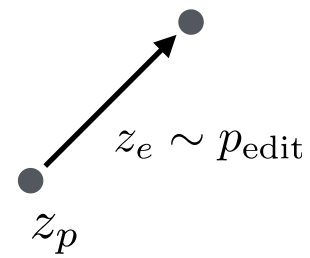
What we will do is use the editor to take a random walk in sentence space.

We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence
And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



**random walk in
sentence space**

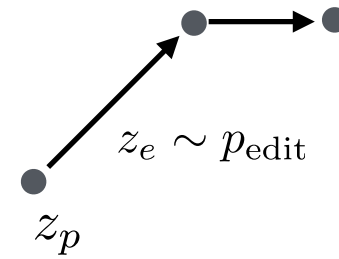
What we will do is use the editor to take a random walk in sentence space.

We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence
And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



**random walk in
sentence space**

What we will do is use the editor to take a random walk in sentence space.

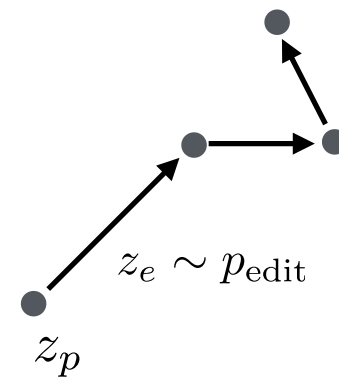
We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



**random walk in
sentence space**

What we will do is use the editor to take a random walk in sentence space.

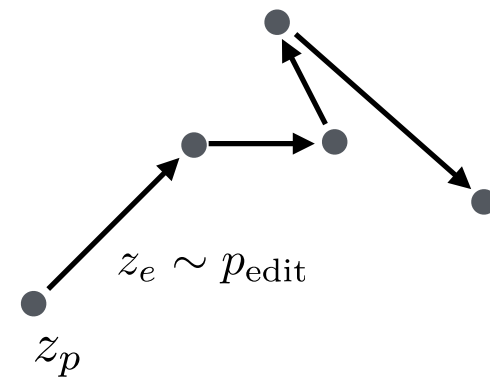
We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



**random walk in
sentence space**

What we will do is use the editor to take a random walk in sentence space.

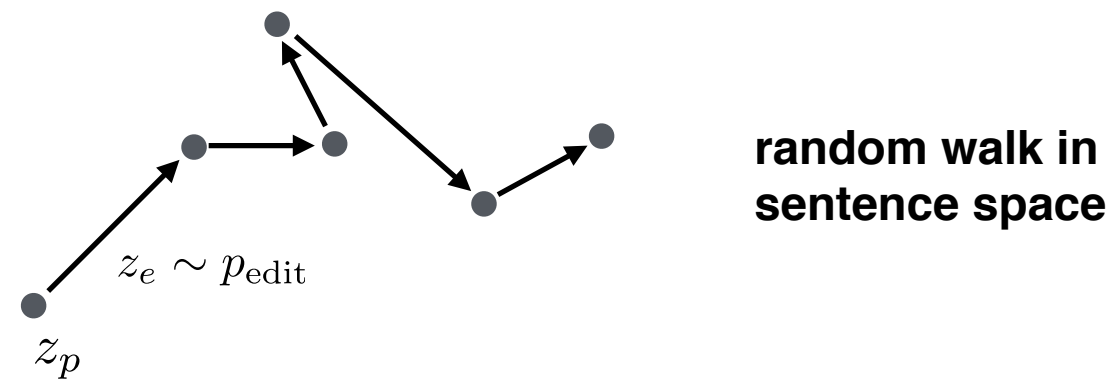
We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



What we will do is use the editor to take a random walk in sentence space.

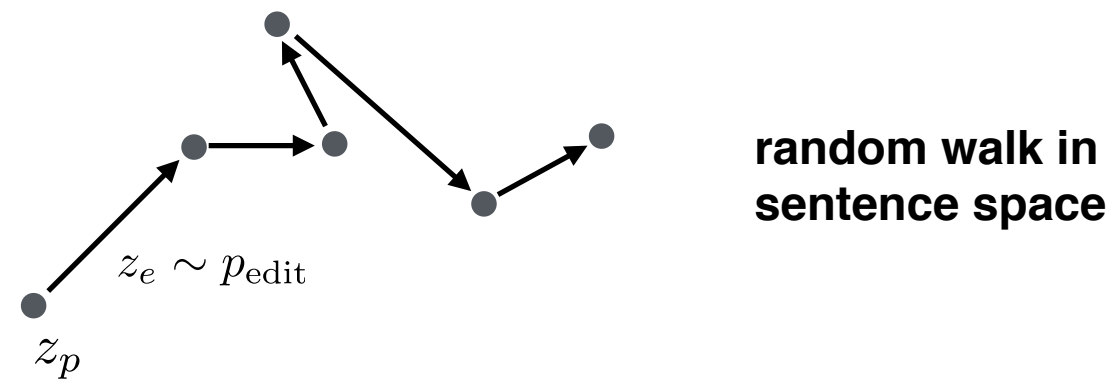
We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



• ice cream was one of the best i've ever tried .

What we will do is use the editor to take a random walk in sentence space.

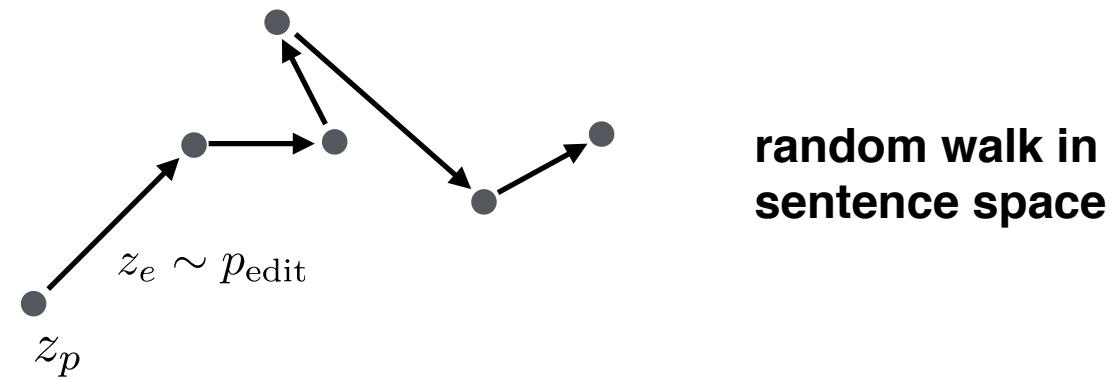
We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



- ice cream was one of the best i've ever tried .
- some of the best ice cream we've ever had .

What we will do is use the editor to take a random walk in sentence space.

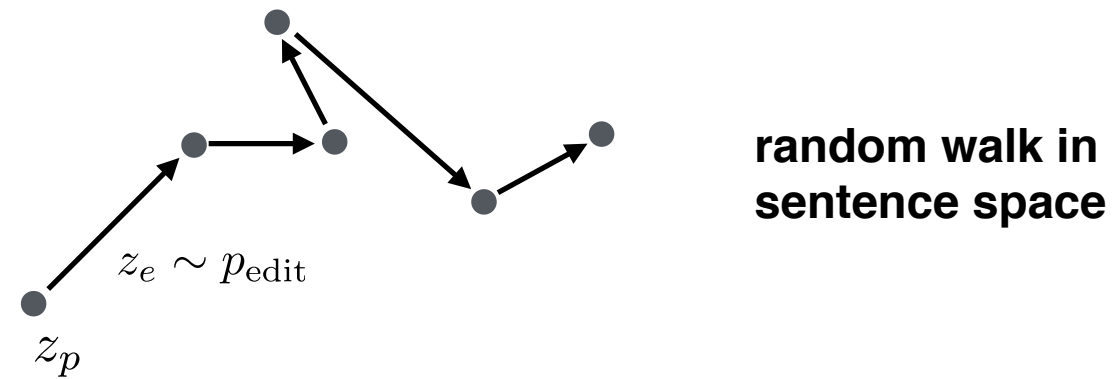
We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



- ice cream was one of the best i've ever tried .
- some of the best ice cream we've ever had .
- just had the best ice - cream i've ever had !

What we will do is use the editor to take a random walk in sentence space.

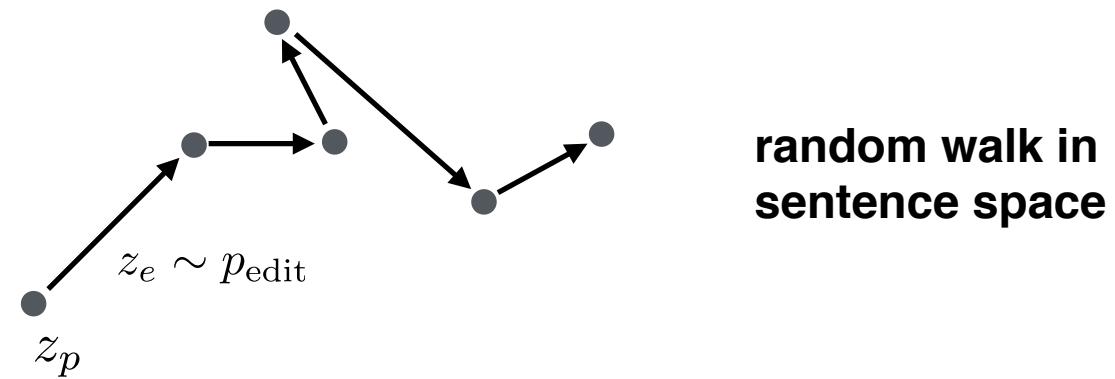
We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



- ice cream was one of the best i've ever tried .
- some of the best ice cream we've ever had .
- just had the best ice - cream i've ever had !
- some of the best pizza i've ever tasted !

What we will do is use the editor to take a random walk in sentence space.

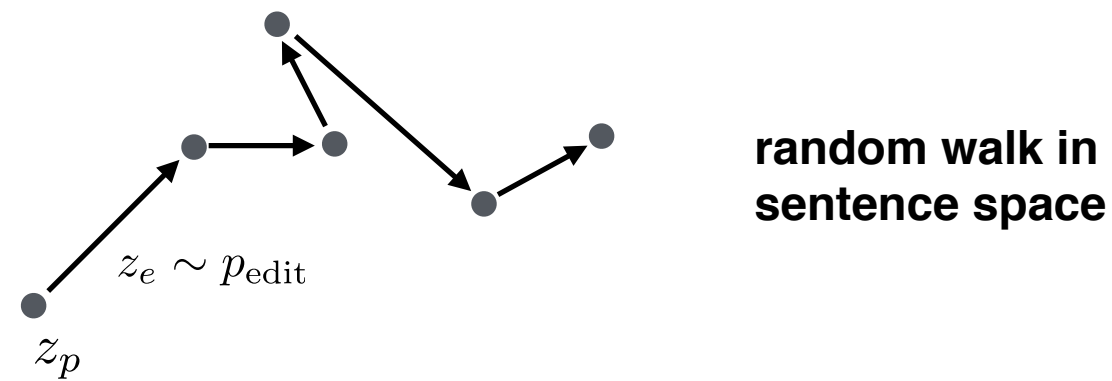
We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Semantic smoothness



- ice cream was one of the best i've ever tried .
- some of the best ice cream we've ever had .
- just had the best ice - cream i've ever had !
- some of the best pizza i've ever tasted !
- that was some of the best pizza i've had in the area .

What we will do is use the editor to take a random walk in sentence space.

We'll start with some prototype sentence z_p

Then apply some random edit vector z_e , to get a new sentence

And then rinse and repeat

Here is an example of what we get when we do this

Turkers: how jumpy is each step?

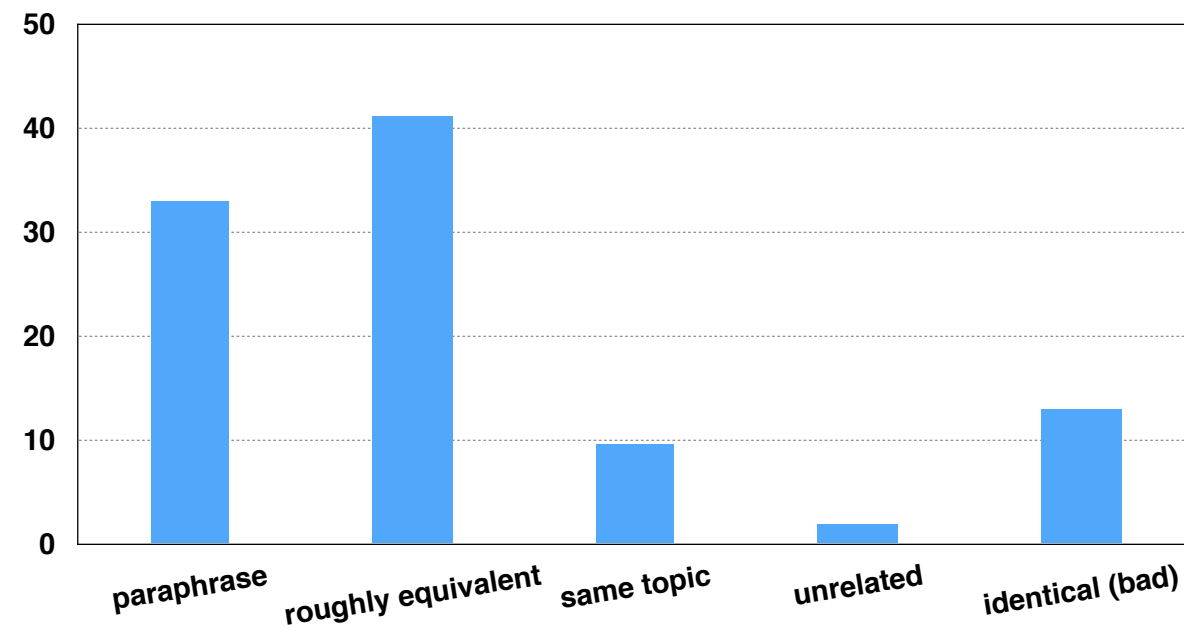
We then asked Turkers to rate how jumpy each step of the walk was

We got the following interesting result.

Over 30% of our steps resulted in actual paraphrases, while 40% were roughly equivalent.

Sometimes the editor did not do anything at all, which is the bad "identical" category on the right

Turkers: how jumpy is each step?



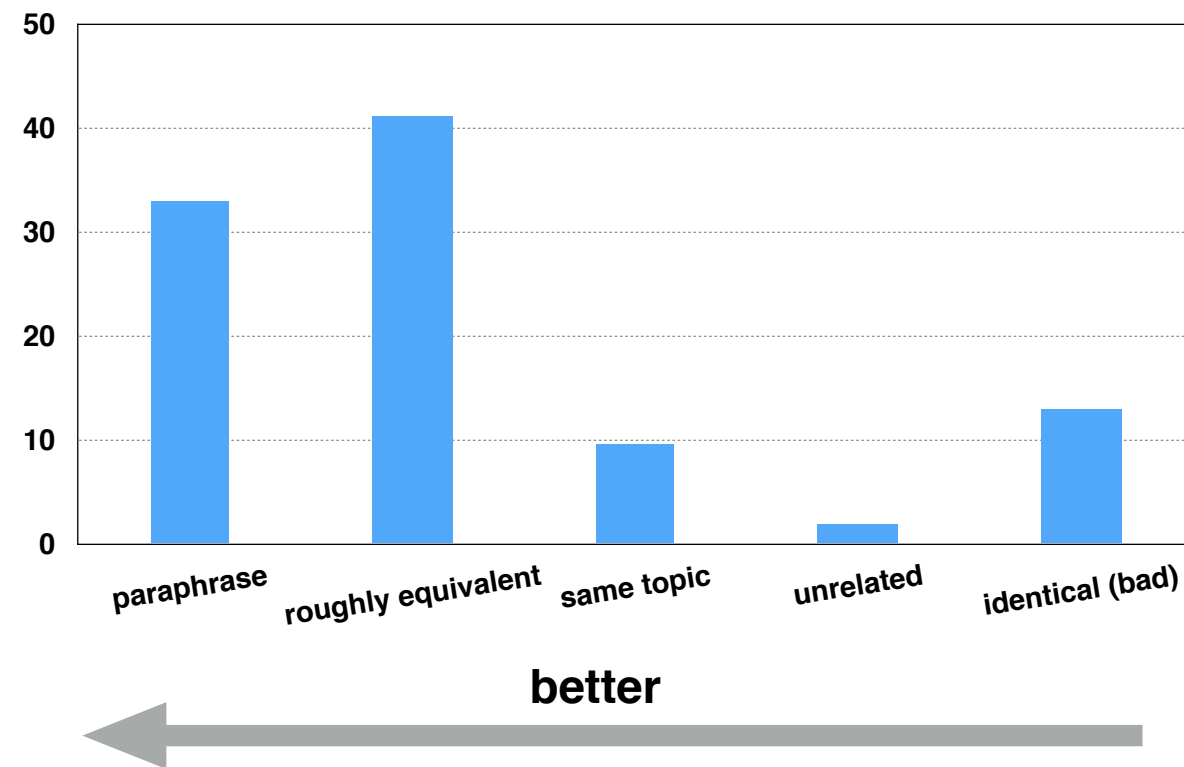
We then asked Turkers to rate how jumpy each step of the walk was

We got the following interesting result.

Over 30% of our steps resulted in actual paraphrases, while 40% were roughly equivalent.

Sometimes the editor did not do anything at all, which is the bad "identical" category on the right

Turkers: how jumpy is each step?



We then asked Turkers to rate how jumpy each step of the walk was

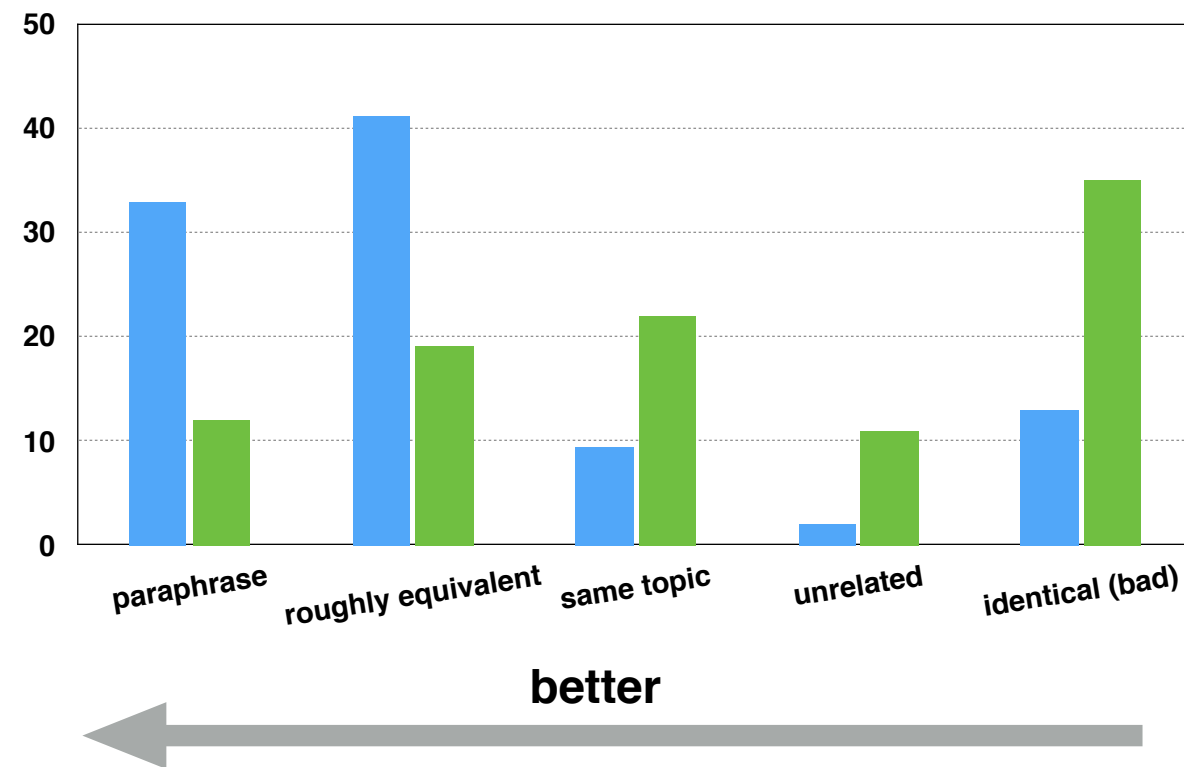
We got the following interesting result.

Over 30% of our steps resulted in actual paraphrases, while 40% were roughly equivalent.

Sometimes the editor did not do anything at all, which is the bad "identical" category on the right

Turkers: how smooth is the random walk?

blue = NeuralEditor **green** = SVAE [Bowman+ 2015]



We also compared our approach to the sentence VAE, another common approach for embedding sentences into vector space

In this comparison, we really tried hard to tune the SVAE to make small steps

But essentially, if we made the step size large, we would always end up with "unrelated" or "same topic" at best, and if we made the step size too small, we would always end up with "identical"

Consistent edit behavior

So we know that the magnitude of the edit vector is quite meaningful. But what about the direction of the edit vector?

To explore this, we thought about sentence analogies.

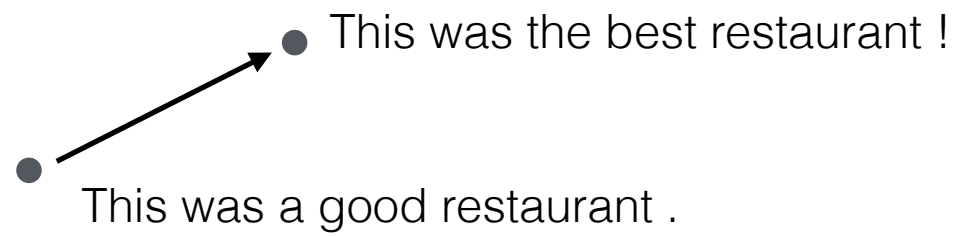
Consistent edit behavior

- This was a good restaurant .

So we know that the magnitude of the edit vector is quite meaningful. But what about the direction of the edit vector?

To explore this, we thought about sentence analogies.

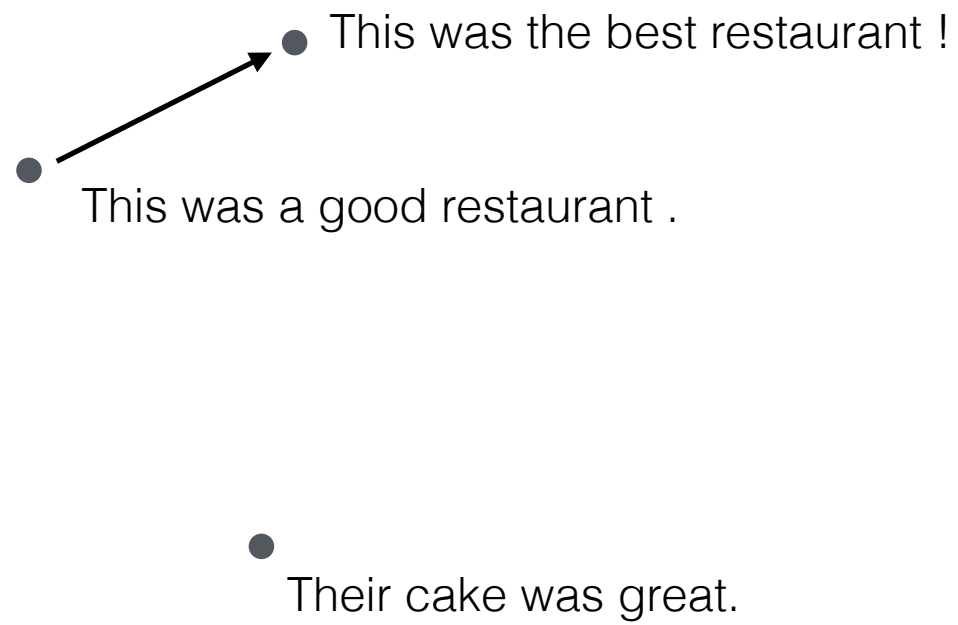
Consistent edit behavior



So we know that the magnitude of the edit vector is quite meaningful. But what about the direction of the edit vector?

To explore this, we thought about sentence analogies.

Consistent edit behavior



So we know that the magnitude of the edit vector is quite meaningful. But what about the direction of the edit vector?

To explore this, we thought about sentence analogies.

Consistent edit behavior



So we know that the magnitude of the edit vector is quite meaningful. But what about the direction of the edit vector?

To explore this, we thought about sentence analogies.

Sentence analogy dataset

Unfortunately, when we started doing this experiment, there was no readily available sentence analogy dataset.

So, we induced a sentence dataset from lexical analogies. Here's how that works.

We start with pairs of words that are analogous.

We then looked in the Yelp review corpus for pairs of sentences that were different by only those word pairs (allowing for reordering and stopwords)

We found at least 10 sentence pairs for each category of analogy

Sentence analogy dataset

Lexical analogies [Mikolov+ 2013]

Unfortunately, when we started doing this experiment, there was no readily available sentence analogy dataset.

So, we induced a sentence dataset from lexical analogies. Here's how that works.

We start with pairs of words that are analogous.

We then looked in the Yelp review corpus for pairs of sentences that were different by only those word pairs (allowing for reordering and stopwords)

We found at least 10 sentence pairs for each category of analogy

Sentence analogy dataset

Lexical analogies [Mikolov+ 2013]

is $\xrightarrow{\text{past tense}}$ **was**

Unfortunately, when we started doing this experiment, there was no readily available sentence analogy dataset.

So, we induced a sentence dataset from lexical analogies. Here's how that works.

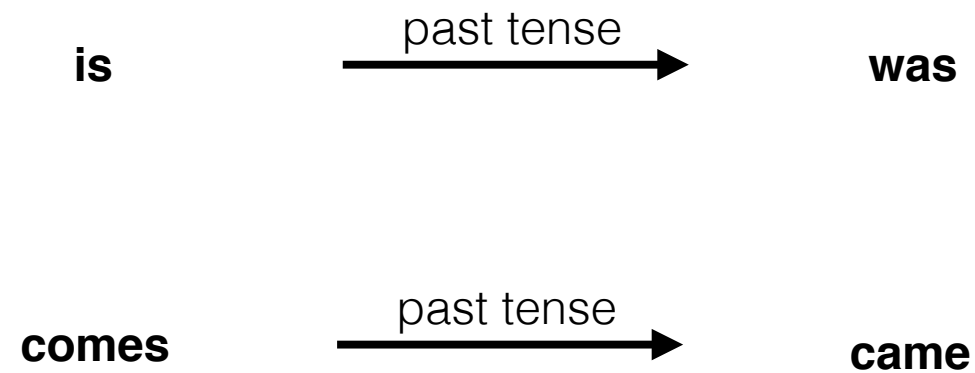
We start with pairs of words that are analogous.

We then looked in the Yelp review corpus for pairs of sentences that were different by only those word pairs (allowing for reordering and stopwords)

We found at least 10 sentence pairs for each category of analogy

Sentence analogy dataset

Lexical analogies [Mikolov+ 2013]



Unfortunately, when we started doing this experiment, there was no readily available sentence analogy dataset.

So, we induced a sentence dataset from lexical analogies. Here's how that works.

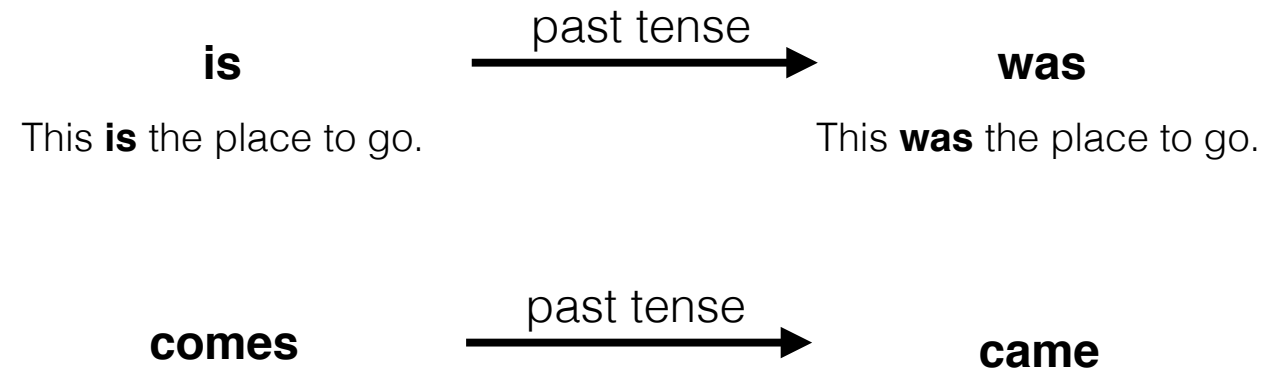
We start with pairs of words that are analogous.

We then looked in the Yelp review corpus for pairs of sentences that were different by only those word pairs (allowing for reordering and stopwords)

We found at least 10 sentence pairs for each category of analogy

Sentence analogy dataset

Lexical analogies [Mikolov+ 2013]



Unfortunately, when we started doing this experiment, there was no readily available sentence analogy dataset.

So, we induced a sentence dataset from lexical analogies. Here's how that works.

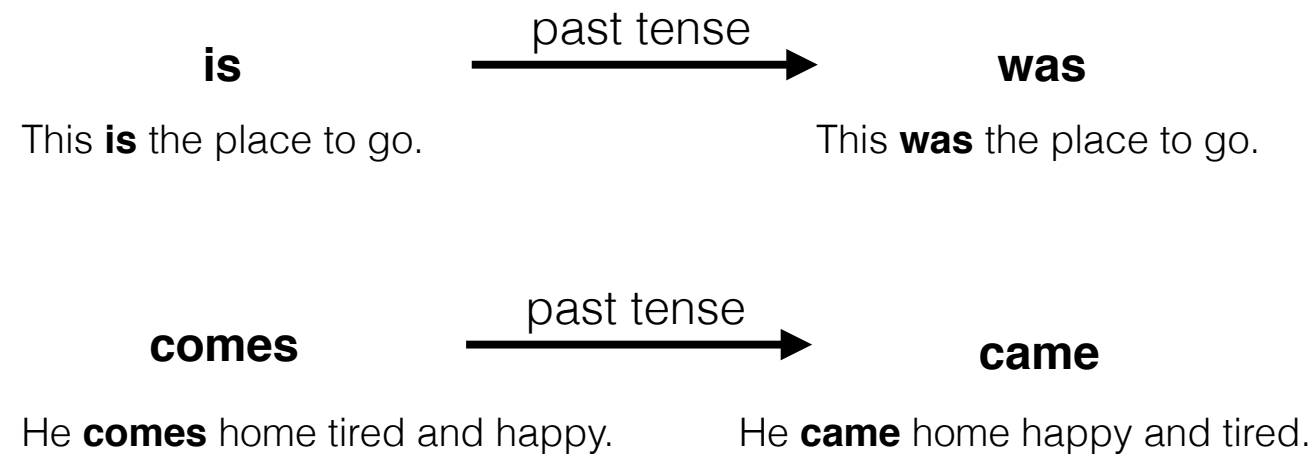
We start with pairs of words that are analogous.

We then looked in the Yelp review corpus for pairs of sentences that were different by only those word pairs (allowing for reordering and stopwords)

We found at least 10 sentence pairs for each category of analogy

Sentence analogy dataset

Lexical analogies [Mikolov+ 2013]



Unfortunately, when we started doing this experiment, there was no readily available sentence analogy dataset.

So, we induced a sentence dataset from lexical analogies. Here's how that works.

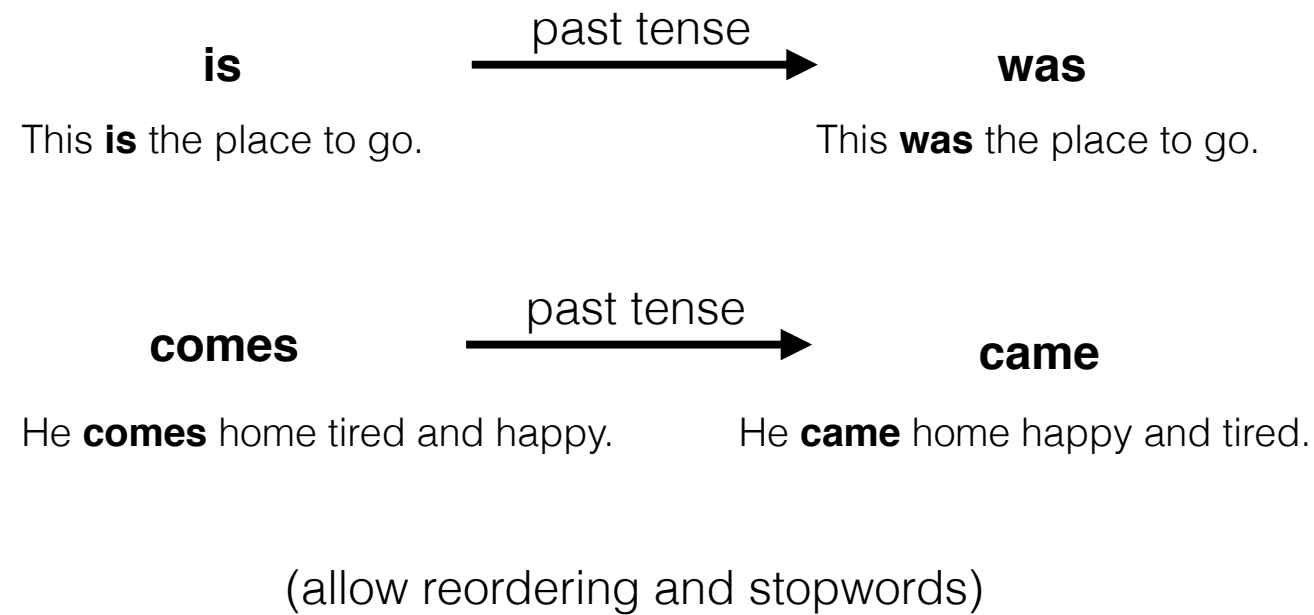
We start with pairs of words that are analogous.

We then looked in the Yelp review corpus for pairs of sentences that were different by only those word pairs (allowing for reordering and stopwords)

We found at least 10 sentence pairs for each category of analogy

Sentence analogy dataset

Lexical analogies [Mikolov+ 2013]



Unfortunately, when we started doing this experiment, there was no readily available sentence analogy dataset.

So, we induced a sentence dataset from lexical analogies. Here's how that works.

We start with pairs of words that are analogous.

We then looked in the Yelp review corpus for pairs of sentences that were different by only those word pairs (allowing for reordering and stopwords)

We found at least 10 sentence pairs for each category of analogy

Sentence analogy dataset

This **is** the place to go.

This **was** the place to go.

He **comes** home tired and happy.

He **came** home happy and tired.

Then we completely throw away the word pairs.

Given a seed pair, we use our model to compute the edit vector between the two sentences.

We then go to a new sentence, call it the prototype, and apply the SAME edit vector.

Ideally, it should produce the same sentence as the true y

Sentence analogy dataset

This **is** the place to go. $\xrightarrow{\hat{z}_e}$ This **was** the place to go.

He **comes** home tired and happy. He **came** home happy and tired.

Then we completely throw away the word pairs.

Given a seed pair, we use our model to compute the edit vector between the two sentences.

We then go to a new sentence, call it the prototype, and apply the SAME edit vector.

Ideally, it should produce the same sentence as the true y

Sentence analogy dataset

This **is** the place to go. $\xrightarrow{\hat{z}_e}$ This **was** the place to go.

z_p
He **comes** home tired and happy. He **came** home happy and tired.

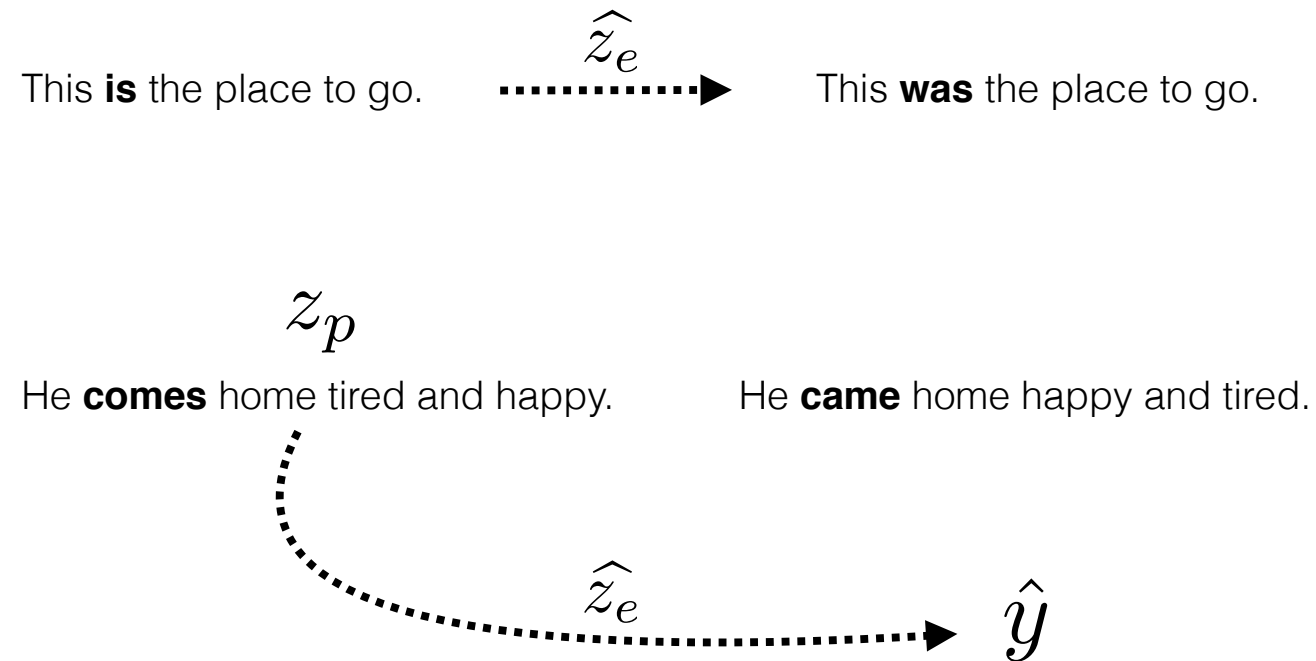
Then we completely throw away the word pairs.

Given a seed pair, we use our model to compute the edit vector between the two sentences.

We then go to a new sentence, call it the prototype, and apply the SAME edit vector.

Ideally, it should produce the same sentence as the true y

Sentence analogy dataset



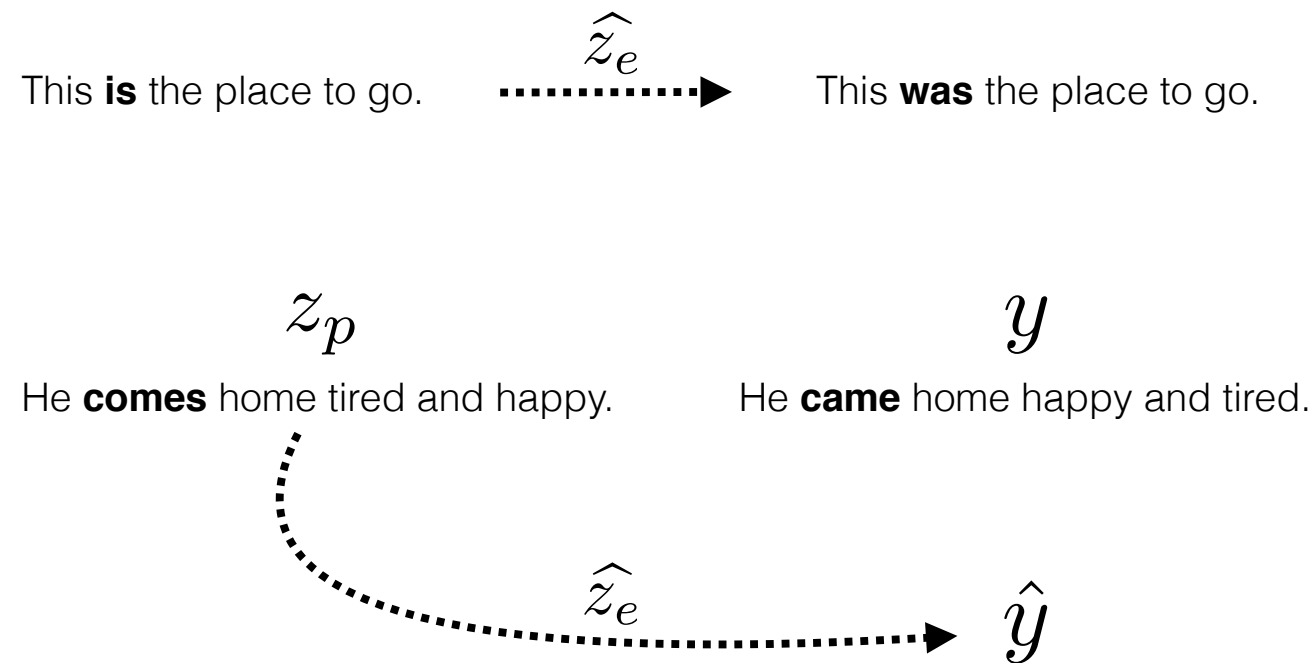
Then we completely throw away the word pairs.

Given a seed pair, we use our model to compute the edit vector between the two sentences.

We then go to a new sentence, call it the prototype, and apply the SAME edit vector.

Ideally, it should produce the same sentence as the true y

Sentence analogy dataset



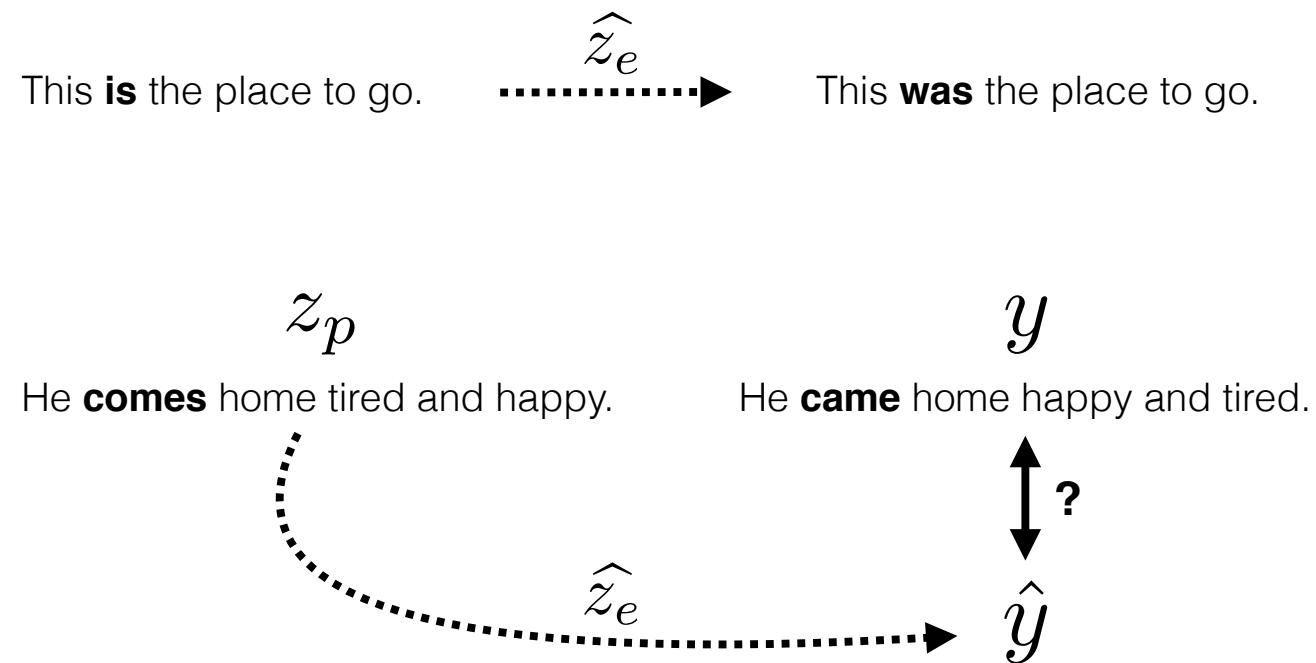
Then we completely throw away the word pairs.

Given a seed pair, we use our model to compute the edit vector between the two sentences.

We then go to a new sentence, call it the prototype, and apply the SAME edit vector.

Ideally, it should produce the same sentence as the true y

Sentence analogy dataset



Then we completely throw away the word pairs.

Given a seed pair, we use our model to compute the edit vector between the two sentences.

We then go to a new sentence, call it the prototype, and apply the SAME edit vector.

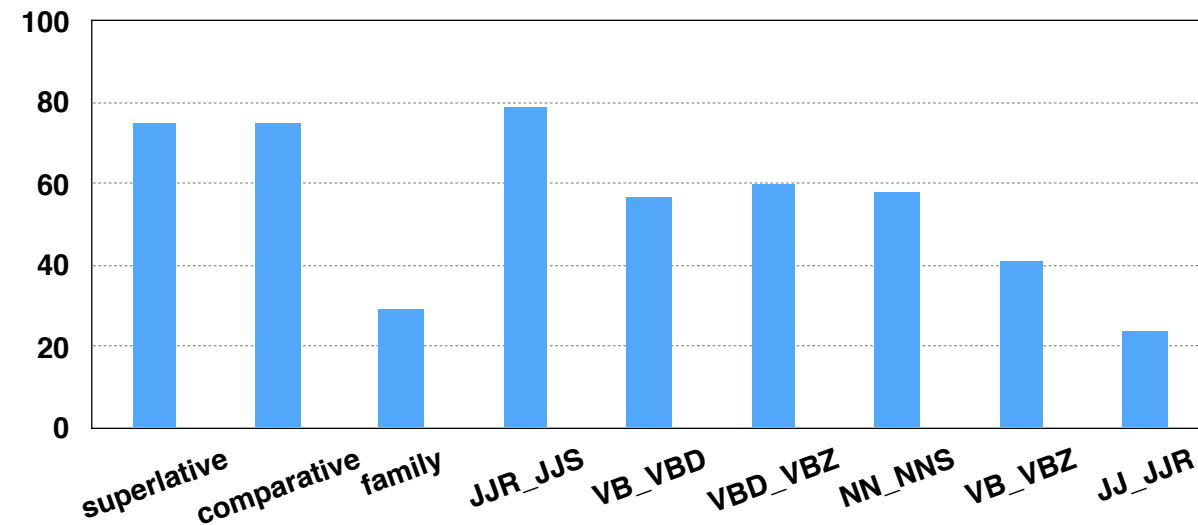
Ideally, it should produce the same sentence as the true y

Sentence analogy results

Note: we need to generate an entire sentence that exactly matches the gold sentence.

Since that is quite hard, we consider it a win if the right sentence shows up in the top-10 beam elements of our model.

Sentence analogy results

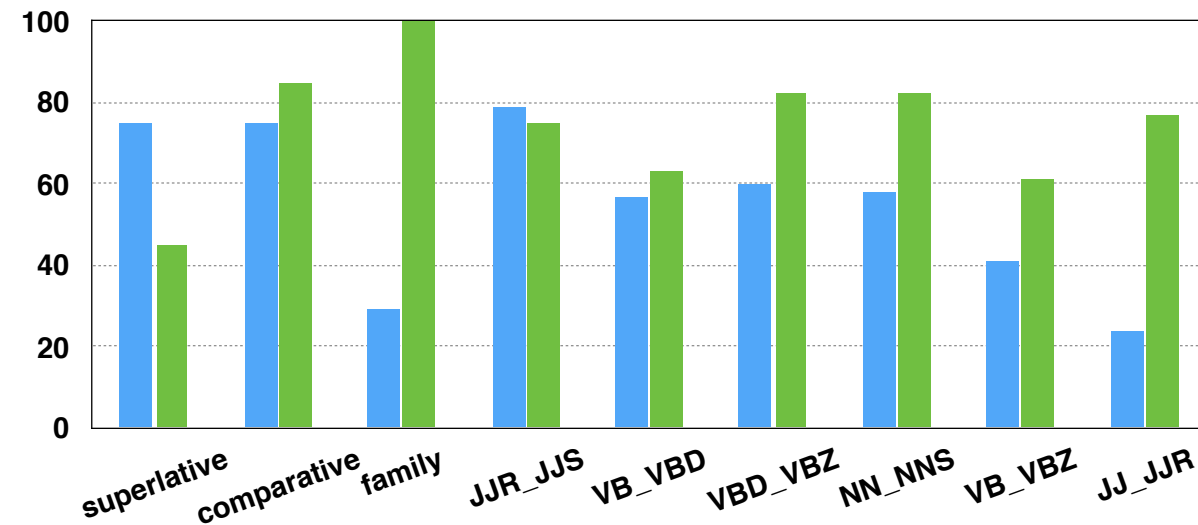


Exact sentence match (top-10 outputs)

Note: we need to generate an entire sentence that exactly matches the gold sentence.

Since that is quite hard, we consider it a win if the right sentence shows up in the top-10 beam elements of our model.

Sentence analogy results



blue = exact sentence match (top-10 outputs)

green = exact word match (GloVE)

Interestingly, you can compare our sentence-level analogy performance

to word-level analogy performance by GloVe vectors

Note: these really aren't strictly comparable. GloVe only needs to predict a word. We need to predict an entire sentence.

Results

- **More diverse generations**
- **Higher quality generations**
- **Better perplexity** (BillionWord, Yelp reviews)
- ✓ **Edits are semantically meaningful**
 - preserve semantic similarity
 - can be used to perform sentence-level analogies

That concludes the investigation of semantics.

But for those of you who don't care about semantics, we also have more standard perplexity results

Results

- **More diverse generations**
- **Higher quality generations**
- ✓ **Better perplexity** (BillionWord, Yelp reviews)
- ✓ **Edits are semantically meaningful**
 - preserve semantic similarity
 - can be used to perform sentence-level analogies

That concludes the investigation of semantics.

But for those of you who don't care about semantics, we also have more standard perplexity results

Perplexity

We evaluated perplexity on the Yelp review corpus and the Billion Word Benchmark.

We compared the NeuralEditor against a standard neural language model.

The NeuralEditor has extra parameters in the encoder, but the same flexibility in the decoder.

Furthermore, we backoff to a standard NLM, because sometimes we encounter a sentence that simply doesn't look like anything in the training corpus.

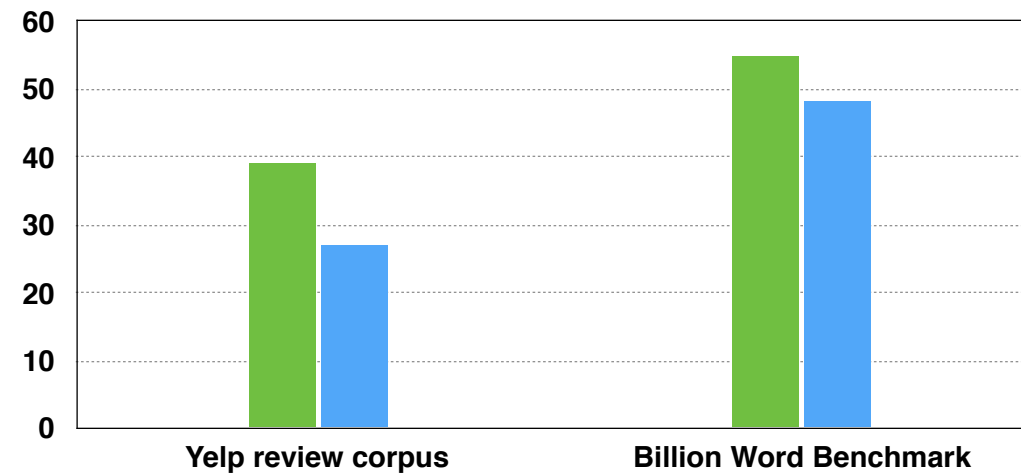
In both cases, we get a nice drop in perplexity.

Yelp is less surprising, because people tend to say the same kinds of things over and over again.

But our results on the Billion Word Benchmark suggest that even news text can benefit.

We also have more results in the paper showing that even if you ensemble a standard NLM with Kneser-Ney, or some sort of pure memorization model, you don't get the same benefits.

Perplexity



We evaluated perplexity on the Yelp review corpus and the Billion Word Benchmark.

We compared the NeuralEditor against a standard neural language model.

The NeuralEditor has extra parameters in the encoder, but the same flexibility in the decoder.

Furthermore, we backoff to a standard NLM, because sometimes we encounter a sentence that simply doesn't look like anything in the training corpus.

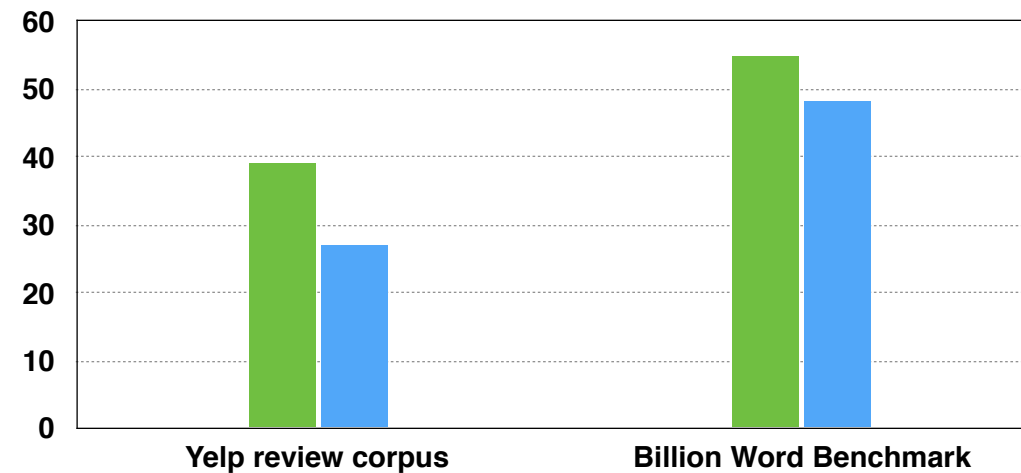
In both cases, we get a nice drop in perplexity.

Yelp is less surprising, because people tend to say the same kinds of things over and over again.

But our results on the Billion Word Benchmark suggest that even news text can benefit.

We also have more results in the paper showing that even if you ensemble a standard NLM with Kneser-Ney, or some sort of pure memorization model, you don't get the same benefits.

Perplexity



green = standard NLM

blue = NeuralEditor (**same** decoder architecture)
+ backoff to standard NLM

We evaluated perplexity on the Yelp review corpus and the Billion Word Benchmark.

We compared the NeuralEditor against a standard neural language model.

The NeuralEditor has extra parameters in the encoder, but the same flexibility in the decoder.

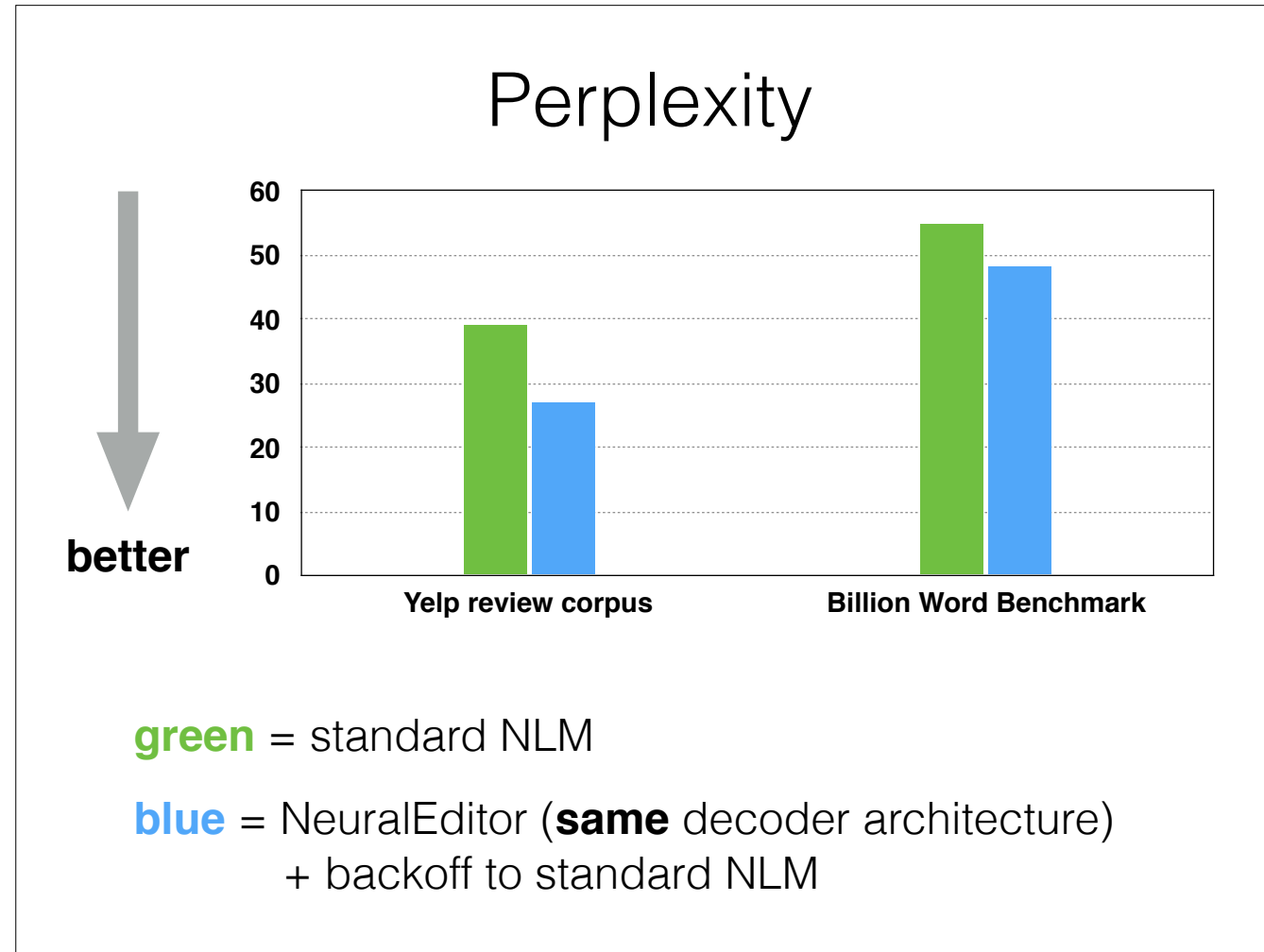
Furthermore, we backoff to a standard NLM, because sometimes we encounter a sentence that simply doesn't look like anything in the training corpus.

In both cases, we get a nice drop in perplexity.

Yelp is less surprising, because people tend to say the same kinds of things over and over again.

But our results on the Billion Word Benchmark suggest that even news text can benefit.

We also have more results in the paper showing that even if you ensemble a standard NLM with Kneser-Ney, or some sort of pure memorization model, you don't get the same benefits.



We evaluated perplexity on the Yelp review corpus and the Billion Word Benchmark.

We compared the NeuralEditor against a standard neural language model.

The NeuralEditor has extra parameters in the encoder, but the same flexibility in the decoder.

Furthermore, we backoff to a standard NLM, because sometimes we encounter a sentence that simply doesn't look like anything in the training corpus.

In both cases, we get a nice drop in perplexity.

Yelp is less surprising, because people tend to say the same kinds of things over and over again.

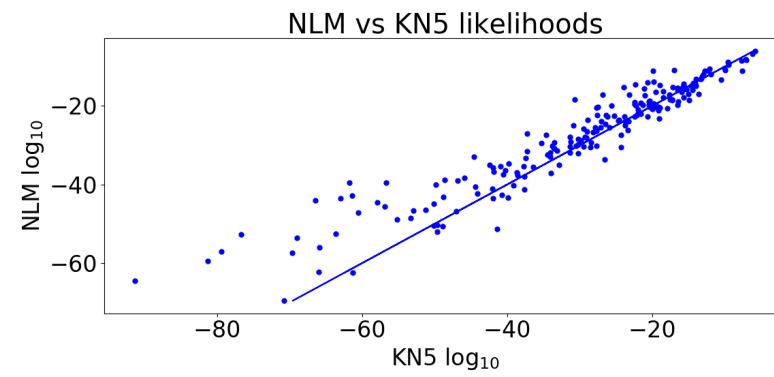
But our results on the Billion Word Benchmark suggest that even news text can benefit.

We also have more results in the paper showing that even if you ensemble a standard NLM with Kneser-Ney, or some sort of pure memorization model, you don't get the same benefits.

Perplexity (closer look)

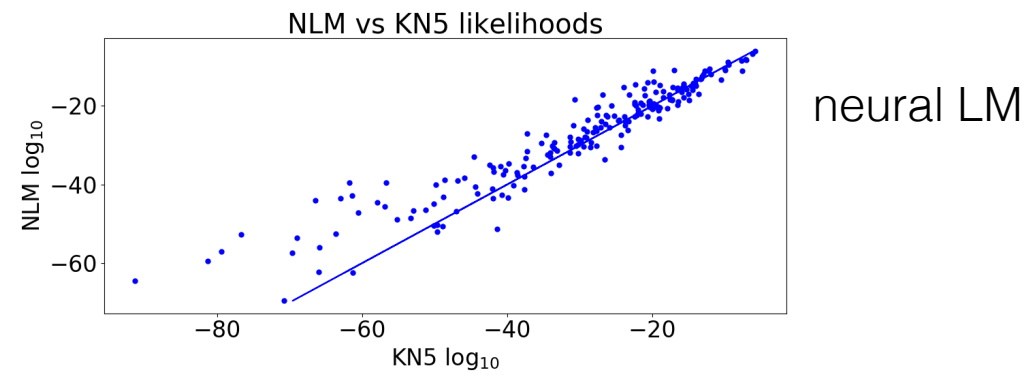
I want to take a closer look at the perplexity, to understand why the NeuralEditor is so helpful.

Perplexity (closer look)



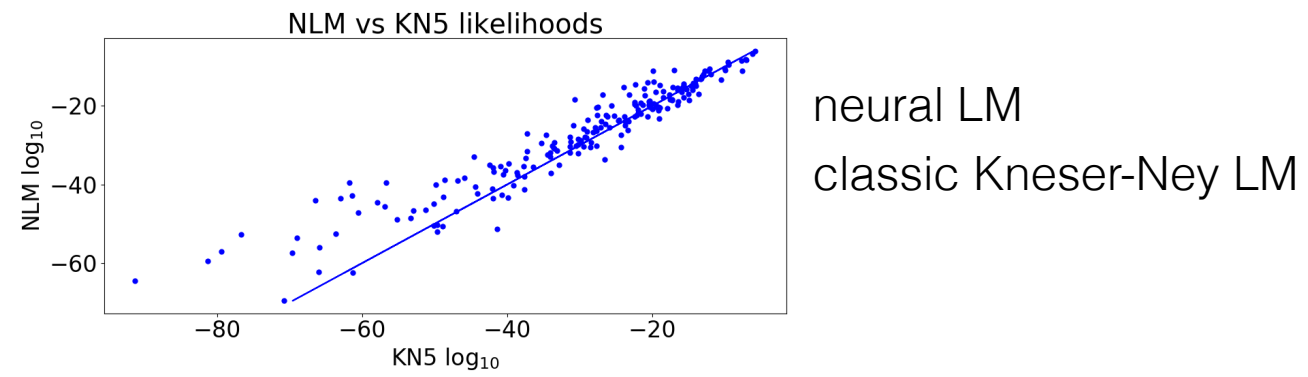
I want to take a closer look at the perplexity, to understand why the NeuralEditor is so helpful.

Perplexity (closer look)



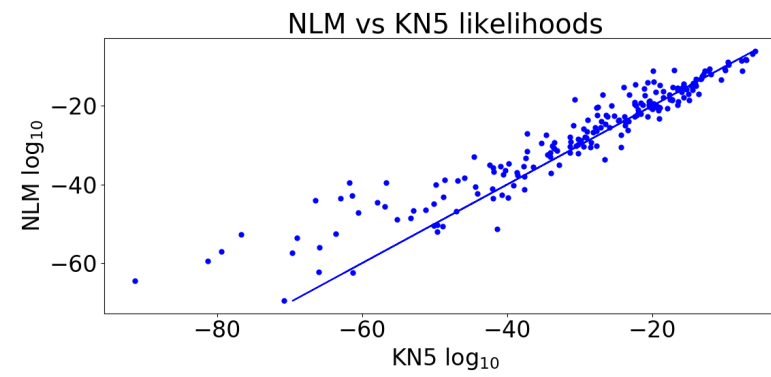
I want to take a closer look at the perplexity, to understand why the NeuralEditor is so helpful.

Perplexity (closer look)



I want to take a closer look at the perplexity, to understand why the NeuralEditor is so helpful.

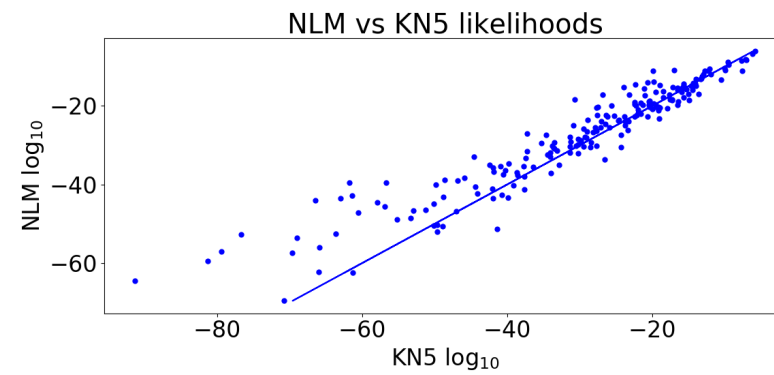
Perplexity (closer look)



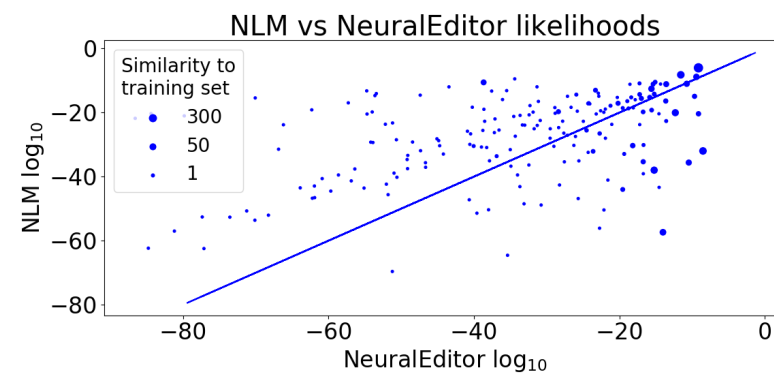
neural LM
classic Kneser-Ney LM
similar

I want to take a closer look at the perplexity, to understand why the NeuralEditor is so helpful.

Perplexity (closer look)

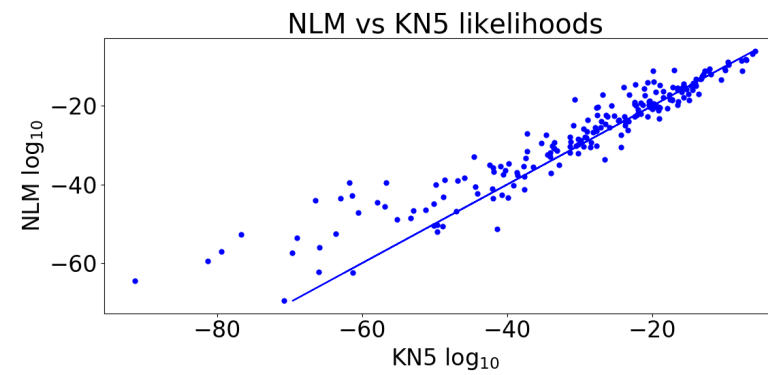


neural LM
classic Kneser-Ney LM
similar

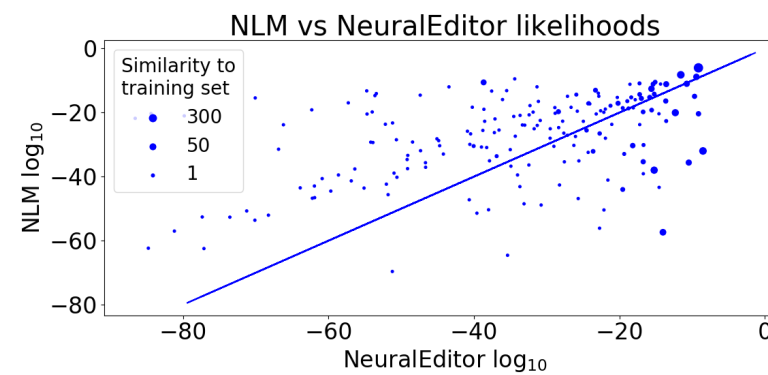


I want to take a closer look at the perplexity, to understand why the NeuralEditor is so helpful.

Perplexity (closer look)



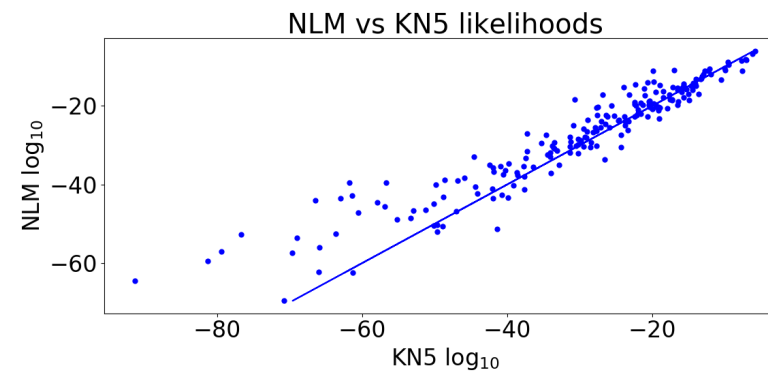
neural LM
classic Kneser-Ney LM
similar



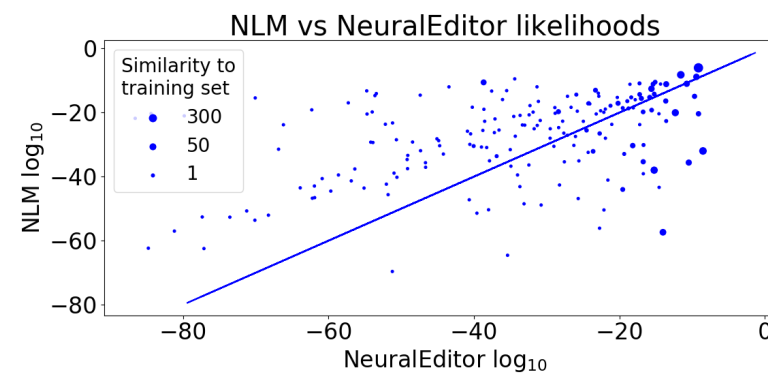
neural LM

I want to take a closer look at the perplexity, to understand why the NeuralEditor is so helpful.

Perplexity (closer look)



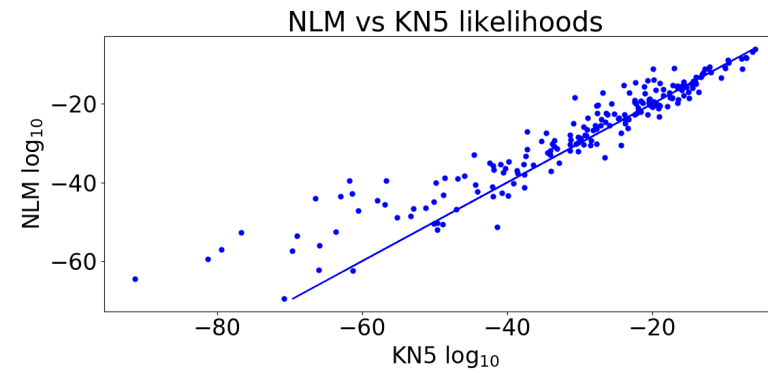
neural LM
classic Kneser-Ney LM
similar



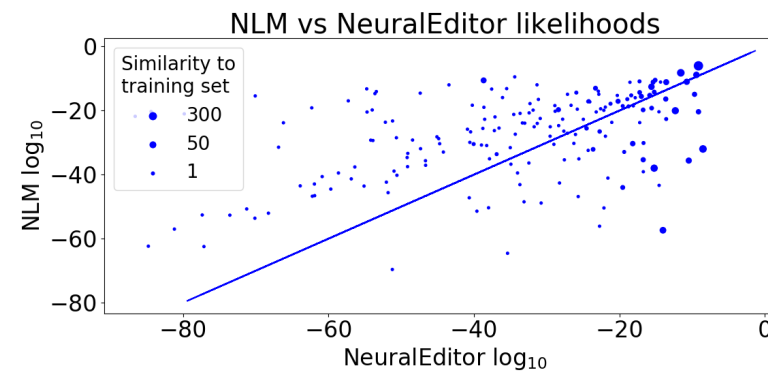
neural LM
NeuralEditor

I want to take a closer look at the perplexity, to understand why the NeuralEditor is so helpful.

Perplexity (closer look)



neural LM
classic Kneser-Ney LM
similar



neural LM
NeuralEditor
different

I want to take a closer look at the perplexity, to understand why the NeuralEditor is so helpful.

Results

- **More diverse generations**
- **Higher quality generations**
- ✓ **Better perplexity** (BillionWord, Yelp reviews)
- ✓ **Edits are semantically meaningful**
 - preserve semantic similarity
 - can be used to perform sentence-level analogies

Up next, we'll look at diversity and quality of generations.

Results

- ✓ **More diverse generations**
- ✓ **Higher quality generations**
- ✓ **Better perplexity** (BillionWord, Yelp reviews)
- ✓ **Edits are semantically meaningful**
 - preserve semantic similarity
 - can be used to perform sentence-level analogies

Up next, we'll look at diversity and quality of generations.

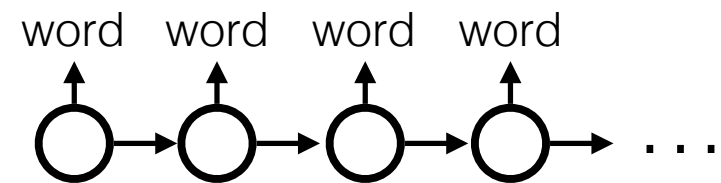
Naive way to increase diversity

A very naive way to increase the diversity of a language model is as follows

Since each time step produces a softmax distribution over words, we can raise the temperature of that softmax, to increase the diversity of the samples

The problem, as we will see, is that raising the temperature starts to allow ungrammatical sequences

Naive way to increase diversity

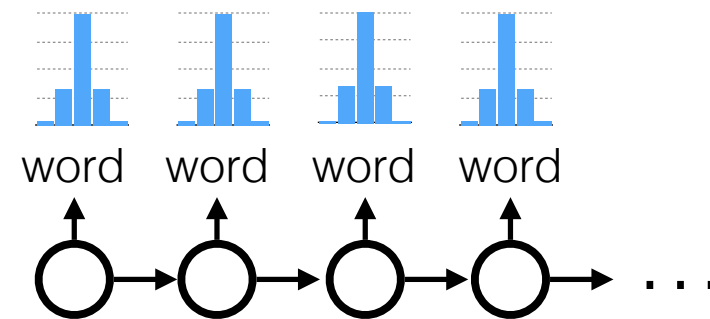


A very naive way to increase the diversity of a language model is as follows

Since each time step produces a softmax distribution over words, we can raise the temperature of that softmax, to increase the diversity of the samples

The problem, as we will see, is that raising the temperature starts to allow ungrammatical sequences

Naive way to increase diversity

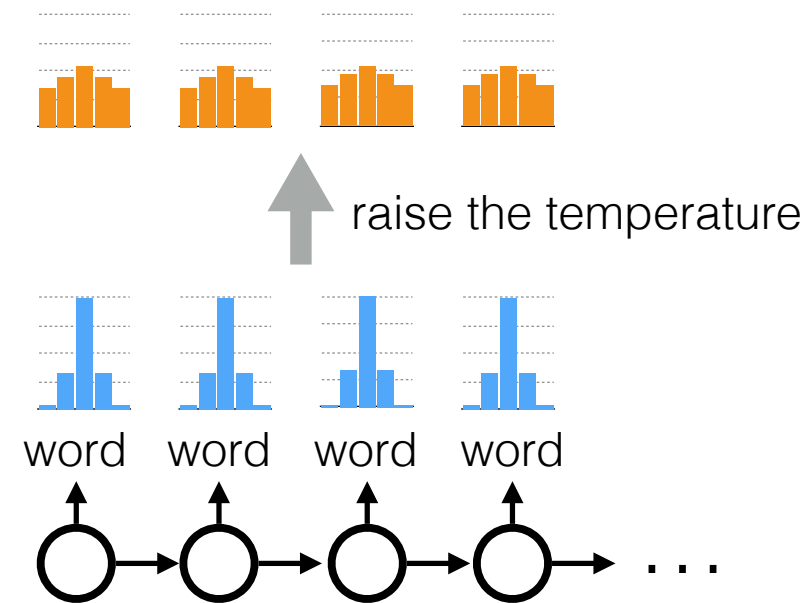


A very naive way to increase the diversity of a language model is as follows

Since each time step produces a softmax distribution over words, we can raise the temperature of that softmax, to increase the diversity of the samples

The problem, as we will see, is that raising the temperature starts to allow ungrammatical sequences

Naive way to increase diversity

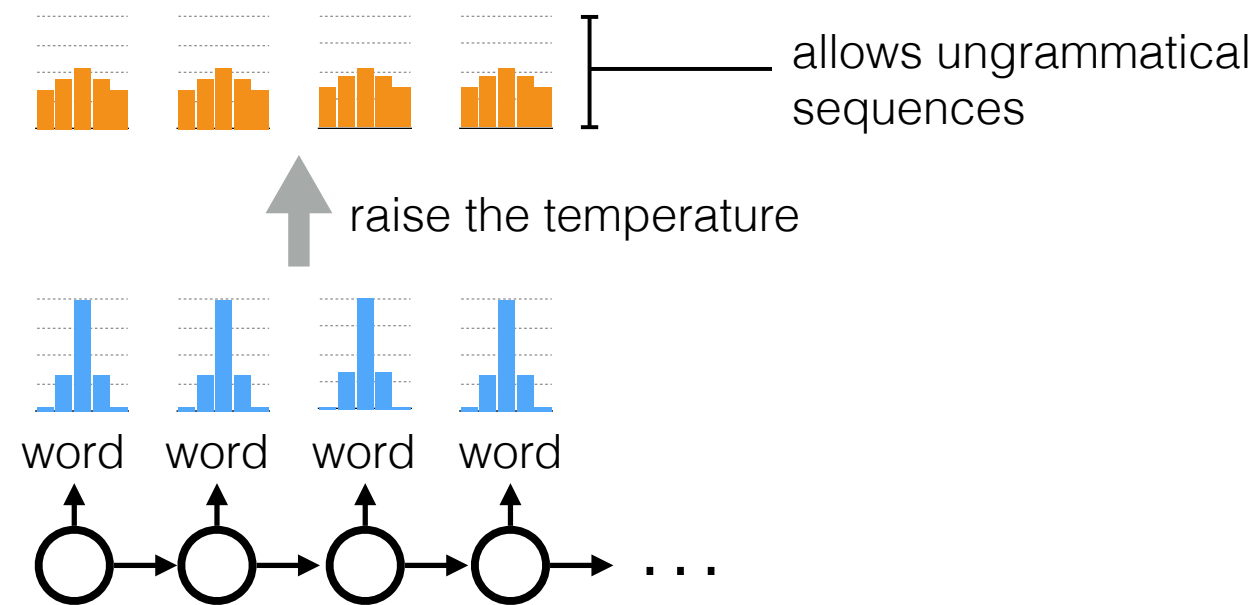


A very naive way to increase the diversity of a language model is as follows

Since each time step produces a softmax distribution over words, we can raise the temperature of that softmax, to increase the diversity of the samples

The problem, as we will see, is that raising the temperature starts to allow ungrammatical sequences

Naive way to increase diversity



A very naive way to increase the diversity of a language model is as follows

Since each time step produces a softmax distribution over words, we can raise the temperature of that softmax, to increase the diversity of the samples

The problem, as we will see, is that raising the temperature starts to allow ungrammatical sequences

Increasing diversity of NeuralEditor

In contrast, we have other options available when using the neural editor

There are actually two ways to increase diversity

First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Increasing diversity of NeuralEditor

$$z_p \sim p_{\text{proto}}$$

In contrast, we have other options available when using the neural editor

There are actually two ways to increase diversity

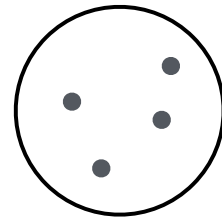
First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Increasing diversity of NeuralEditor

$$z_p \sim p_{\text{proto}}$$



In contrast, we have other options available when using the neural editor

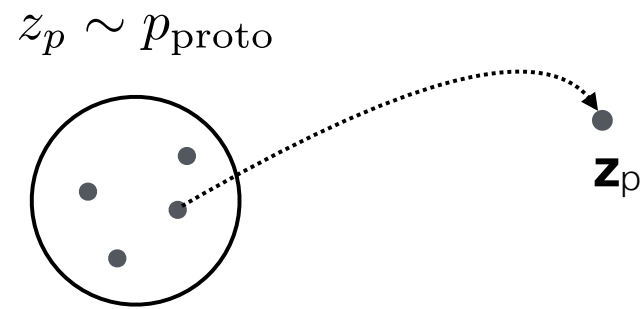
There are actually two ways to increase diversity

First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Increasing diversity of NeuralEditor



In contrast, we have other options available when using the neural editor

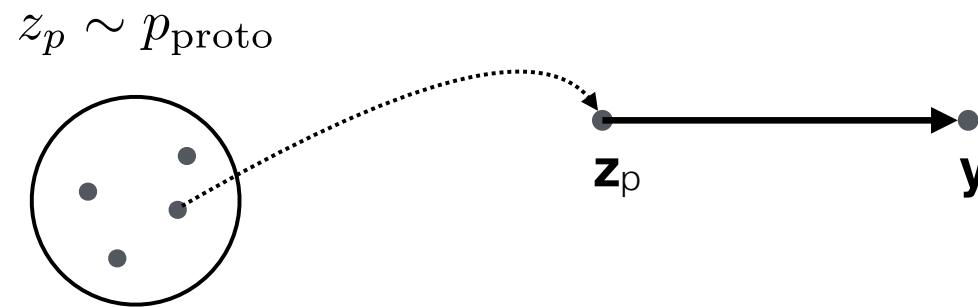
There are actually two ways to increase diversity

First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Increasing diversity of NeuralEditor



In contrast, we have other options available when using the neural editor

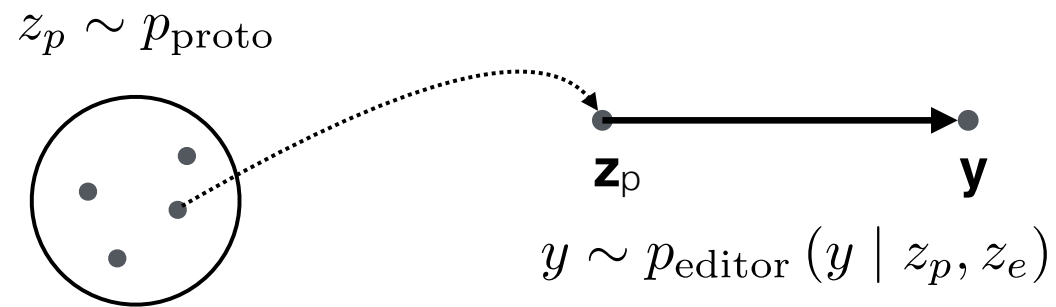
There are actually two ways to increase diversity

First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Increasing diversity of NeuralEditor



In contrast, we have other options available when using the neural editor

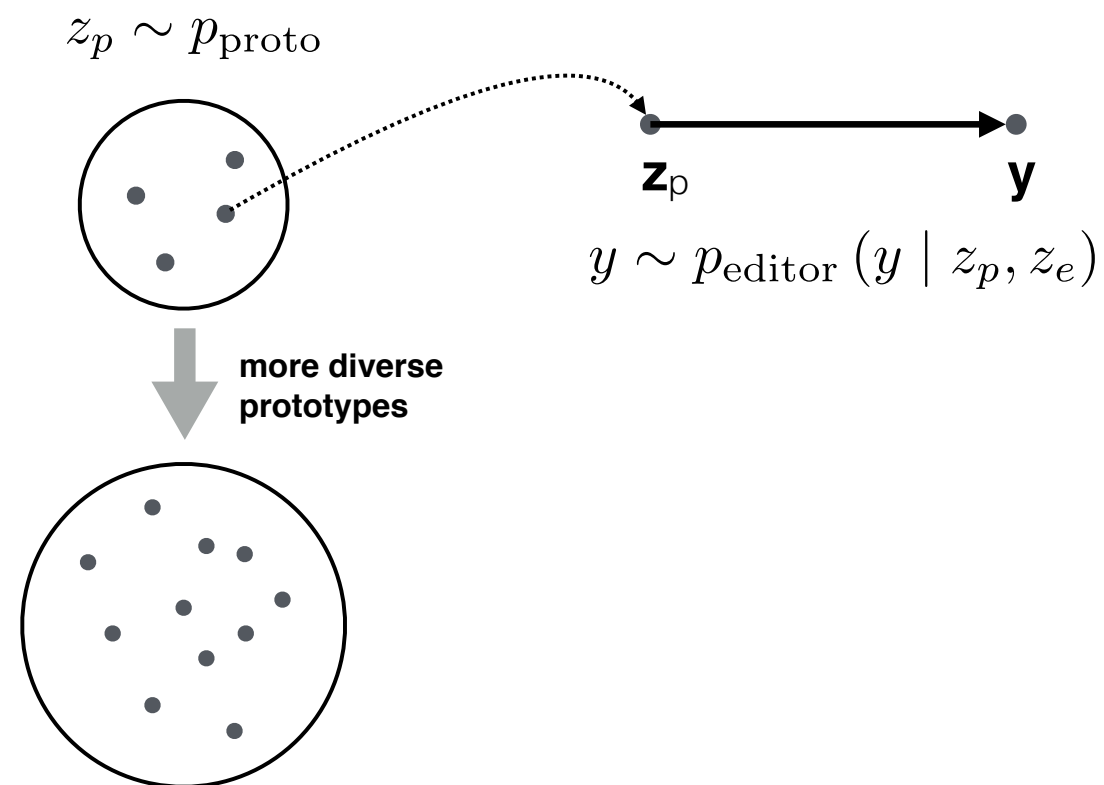
There are actually two ways to increase diversity

First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Increasing diversity of NeuralEditor



In contrast, we have other options available when using the neural editor

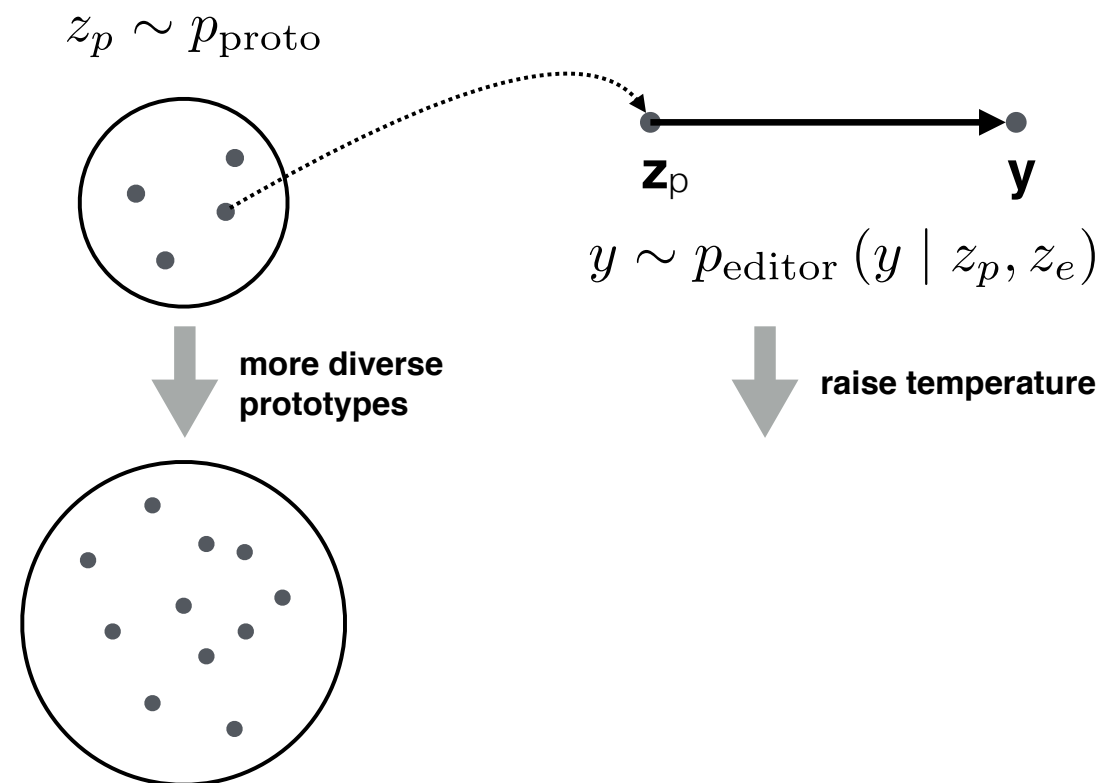
There are actually two ways to increase diversity

First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Increasing diversity of NeuralEditor



In contrast, we have other options available when using the neural editor

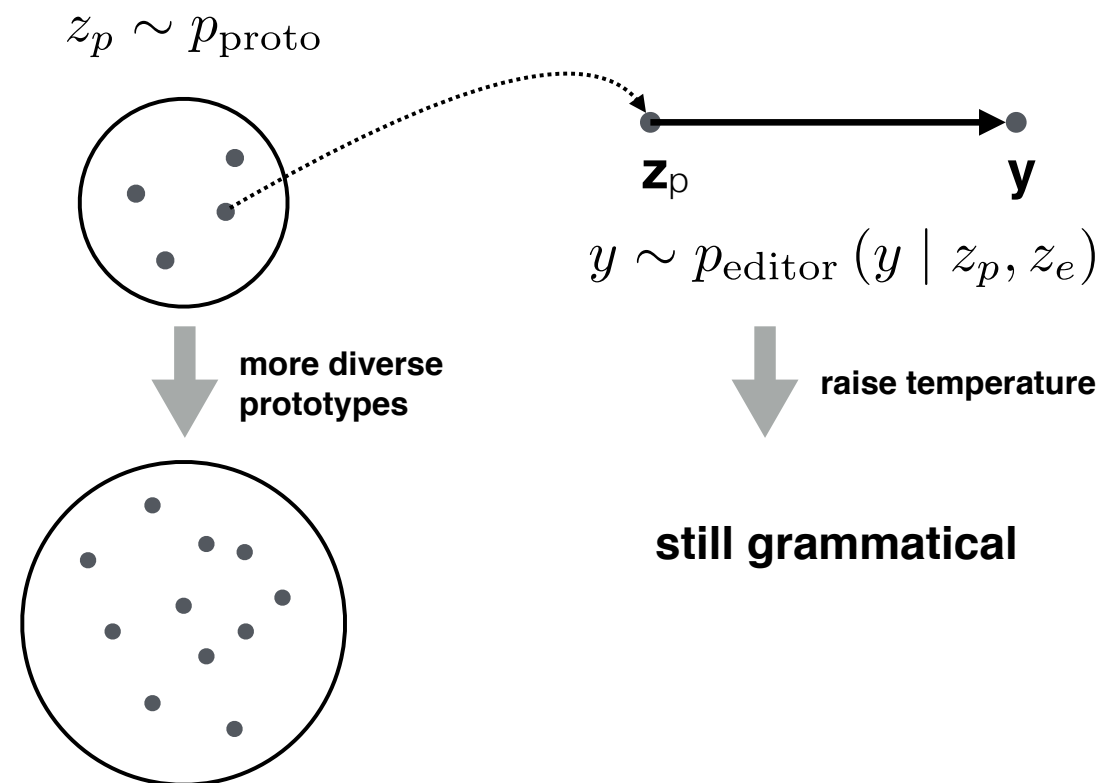
There are actually two ways to increase diversity

First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Increasing diversity of NeuralEditor



In contrast, we have other options available when using the neural editor

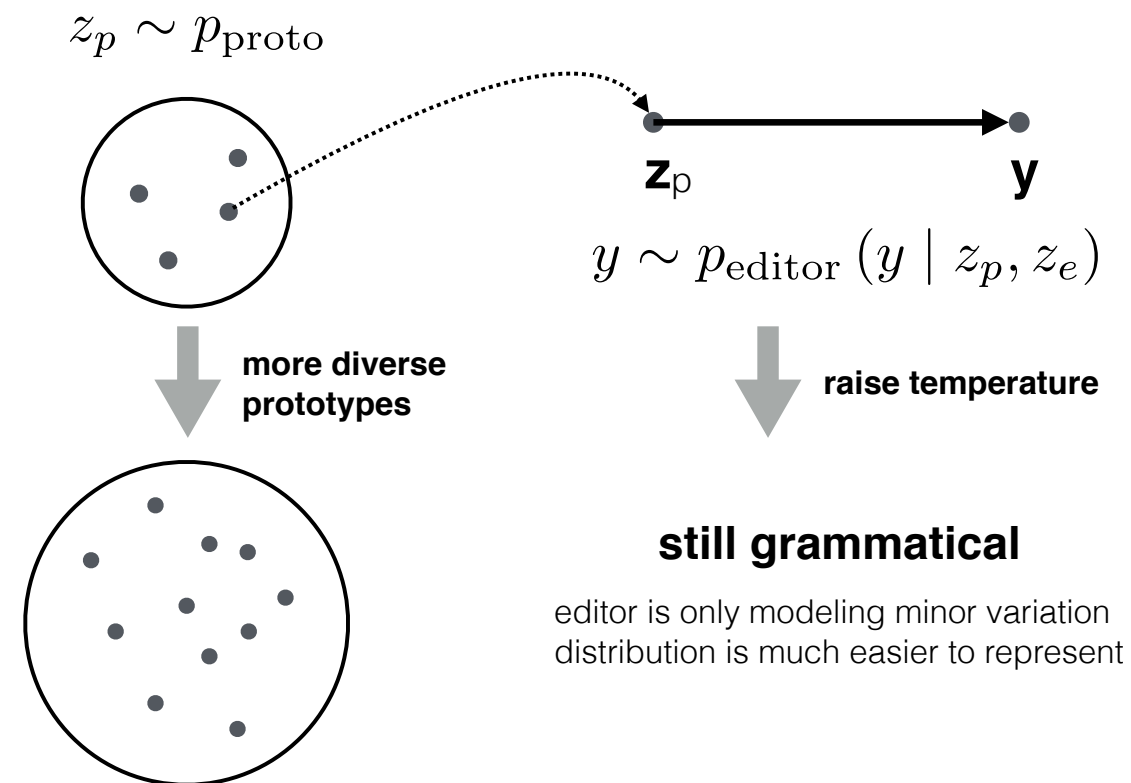
There are actually two ways to increase diversity

First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Increasing diversity of NeuralEditor



In contrast, we have other options available when using the neural editor

There are actually two ways to increase diversity

First, we can control the diversity of our prototypes. If we simply make our prototype set more diverse, we can increase diversity without even touching the editor.

Second, we can still raise the temperature of the editor. Fortunately, even this still turns out to be quite grammatical.

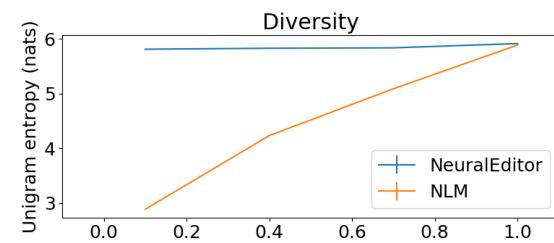
Unlike the full neural language model, the editor is only modeling minor variation around one sentence, so the distribution is much easier to represent.

Diversity: NLM vs NeuralEditor

Let's look at the results empirically.

Here is a plot of temperature vs diversity.

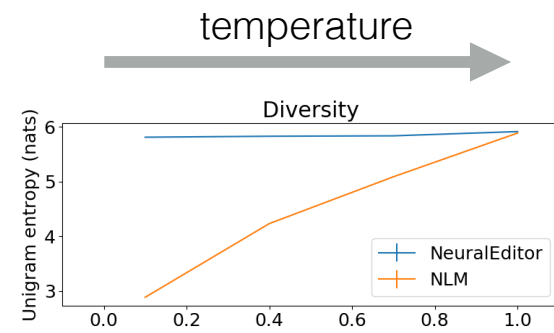
Diversity: NLM vs NeuralEditor



Let's look at the results empirically.

Here is a plot of temperature vs diversity.

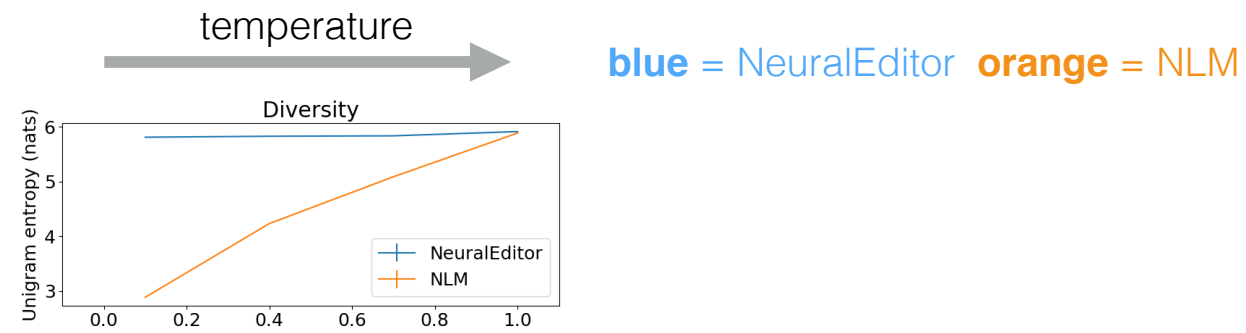
Diversity: NLM vs NeuralEditor



Let's look at the results empirically.

Here is a plot of temperature vs diversity.

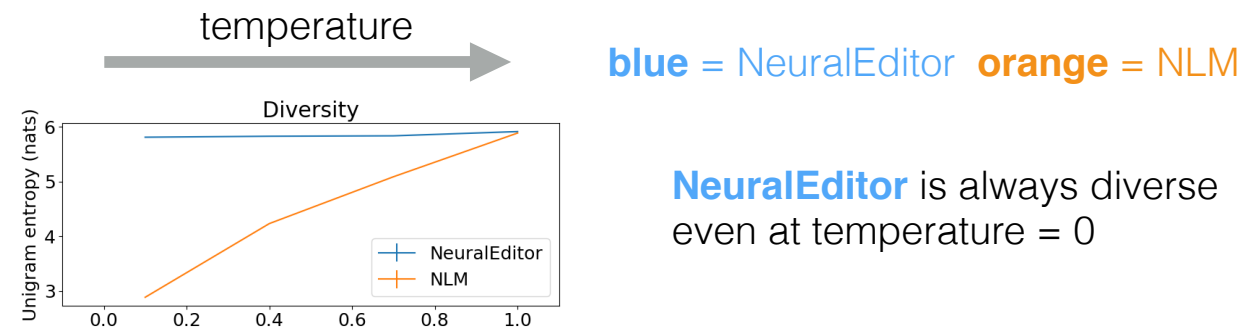
Diversity: NLM vs NeuralEditor



Let's look at the results empirically.

Here is a plot of temperature vs diversity.

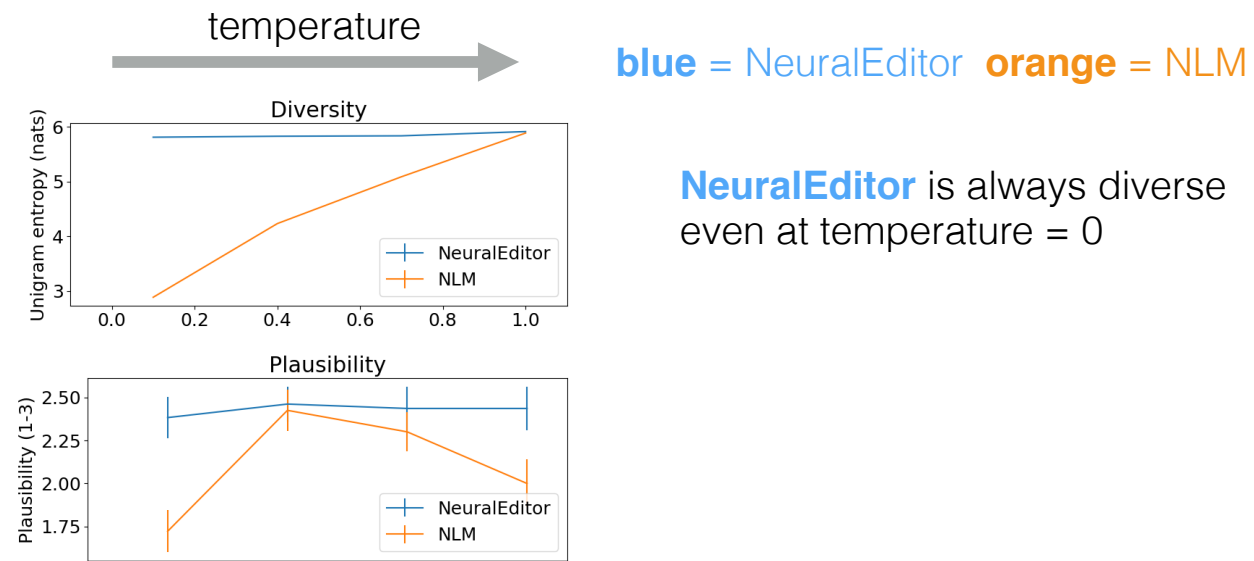
Diversity: NLM vs NeuralEditor



Let's look at the results empirically.

Here is a plot of temperature vs diversity.

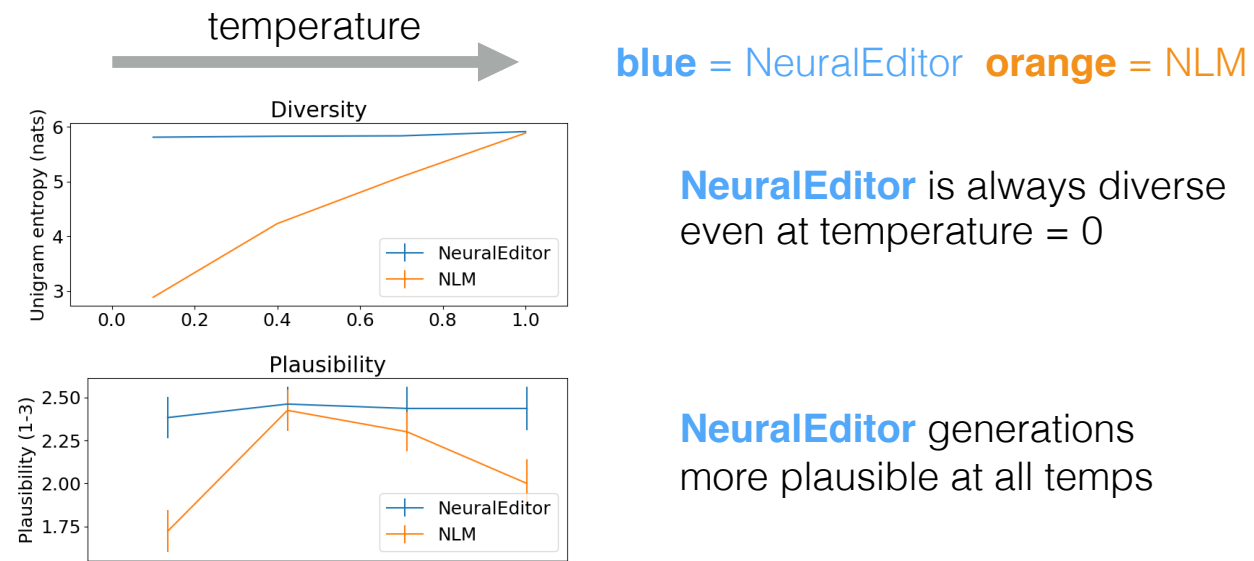
Diversity: NLM vs NeuralEditor



Let's look at the results empirically.

Here is a plot of temperature vs diversity.

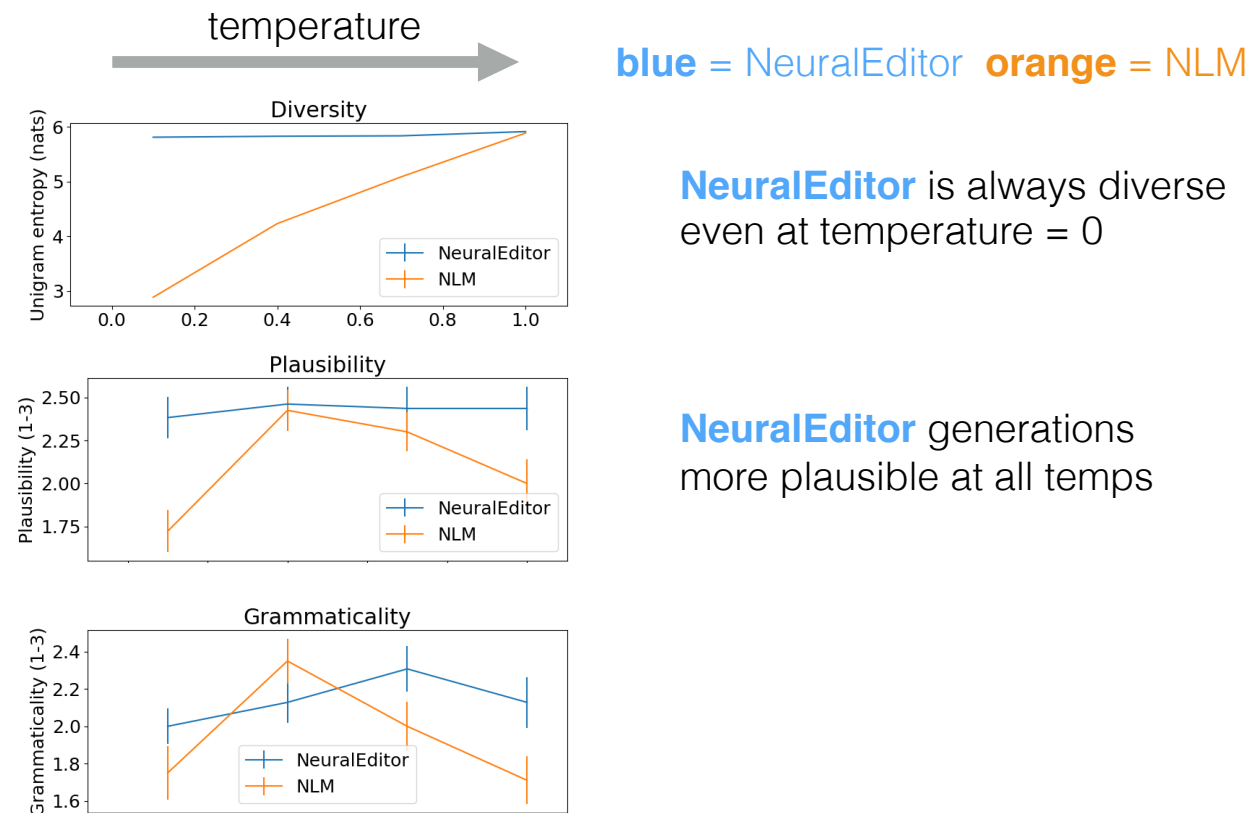
Diversity: NLM vs NeuralEditor



Let's look at the results empirically.

Here is a plot of temperature vs diversity.

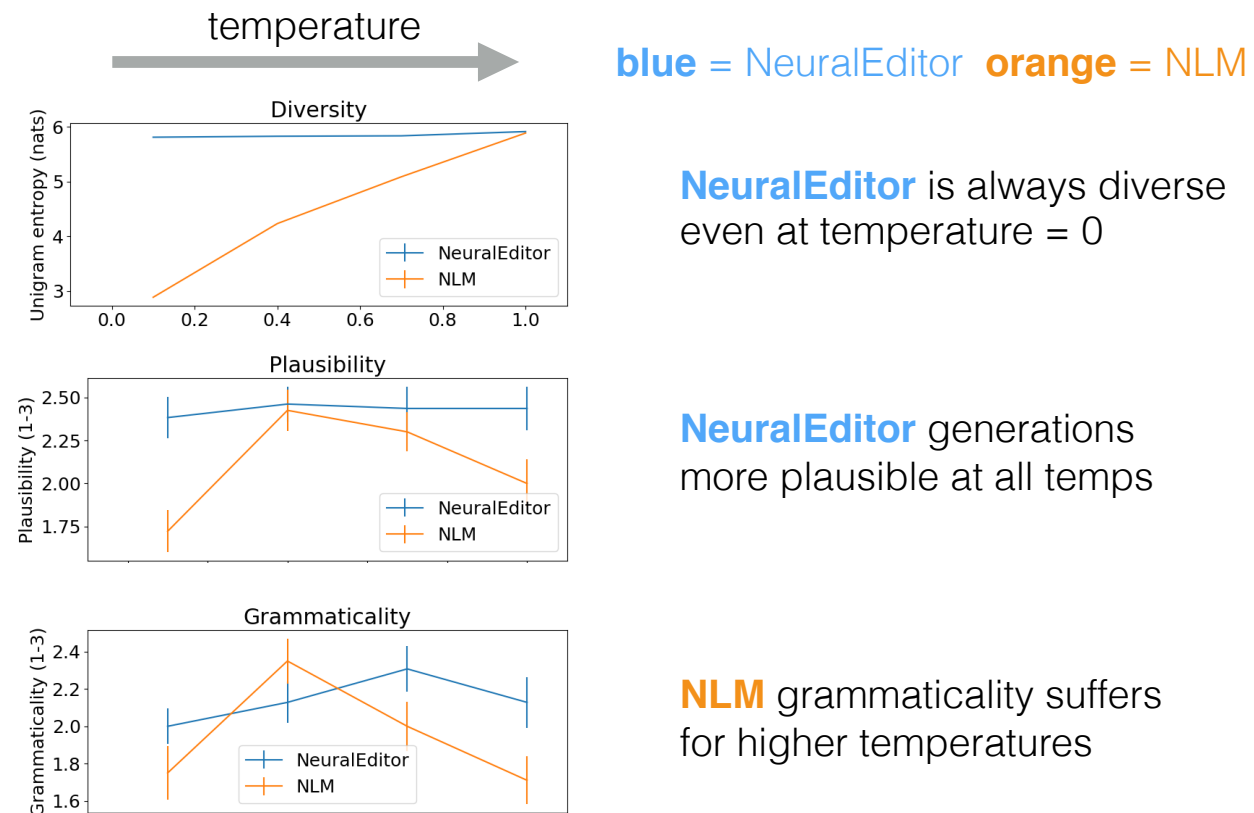
Diversity: NLM vs NeuralEditor



Let's look at the results empirically.

Here is a plot of temperature vs diversity.

Diversity: NLM vs NeuralEditor



Let's look at the results empirically.

Here is a plot of temperature vs diversity.

Results

- ✓ **More diverse generations**
- ✓ **Higher quality generations**
- ✓ **Better perplexity** (BillionWord, Yelp reviews)
- ✓ **Edits are semantically meaningful**
 - preserve semantic similarity
 - can be used to perform sentence-level analogies

\end{\b{Results}}