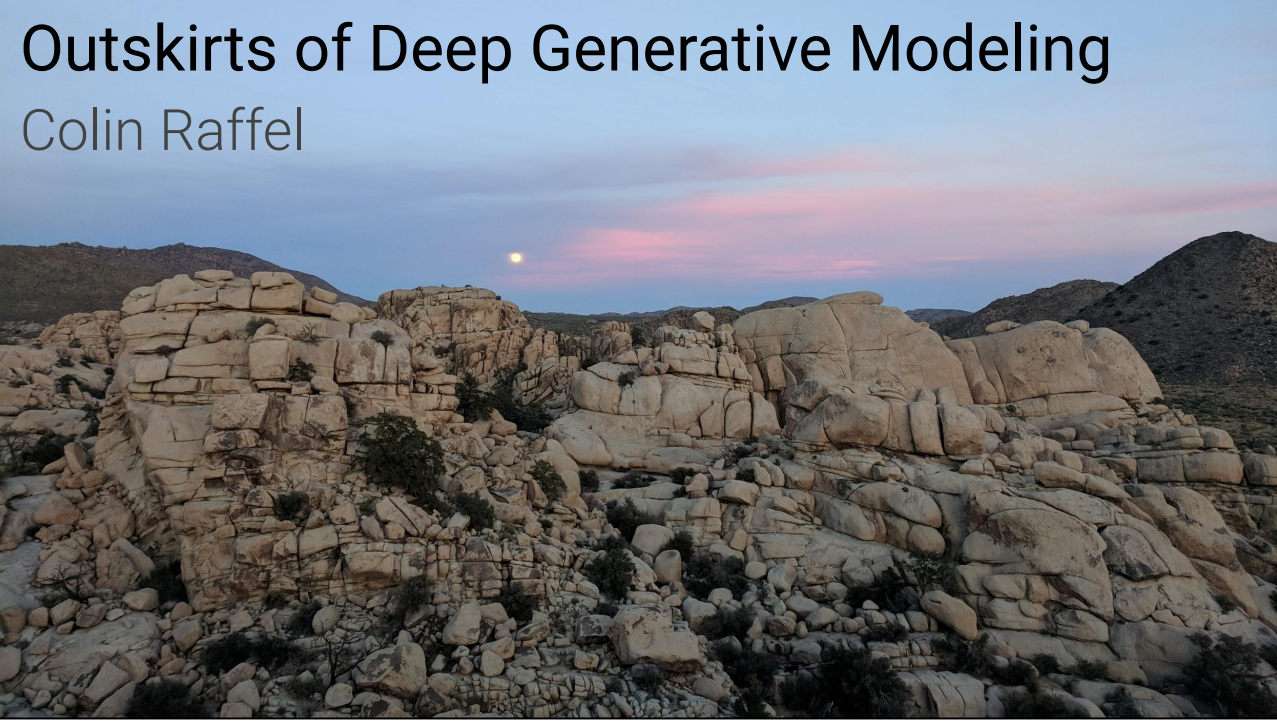
Outskirts of Deep Generative Modeling

Colin Raffel



You are viewing the PDF version of these slides; any animations will appear as static images.

In this talk I'll provide an overview of the dominant approaches and describe my own work in the field, which is focused on sequential data and representation learning. I'll wrap up with my outlook on the research problems we need to solve in order to apply generative modeling to new and important real-world problems.

So, what is generative modeling? Let's start with a example. Let's say I go out into the world and collect a dataset of natural images. So, first I get this picture of some cyclists...

And then some pictures of vegetables

And then a few more photos

Dataset of natural images

And eventually I have a large collection of natural images. In generative modeling, we say that these images are **samples from some true distribution** - the distribution of natural images. The goal of generative modeling is to take this training set and learn a model which approximates this true distribution. That is, I hope to be able to draw new samples from my model which appear like real natural images, but are totally new and are not simply copies of entries from my training set.

Dataset of natural images

Generated image, from “Large Scale GAN Training for High Fidelity Natural Image Synthesis”, Brock et al.

Here's an example of an image produced by a recent generative model developed by some researchers at DeepMind. The generative model has made up this dog completely on its own - it’s only input is a random noise vector and the label “dog”. It has never seen this particular dog before in the training set, either. To my eyes at least, it's hard to tell that this is a generated image and not a real picture of a dog.

Dataset of natural images

Generated image, from "Large Scale GAN Training for High Fidelity Natural Image Synthesis", Brock et al.

Here's another image from the same model, this one of a monarch butterfly..

Dataset of natural images

Generated image, from "Large Scale GAN Training for High Fidelity Natural Image Synthesis", Brock et al.

And here's another one. I for one would be happy to eat this made-up cheeseburger. The fact that the model is able to produce these images suggests that it has learned about the structure of the natural world. The model has done this without explicit supervision - it was simply shown millions of images and discovered this structure itself.

Dataset of natural images

Generated image, from "Large Scale GAN Training for High Fidelity Natural Image Synthesis", Brock et al.

Of course, this model is not perfect -- sometimes it generates things like this quasi-car.

Dataset of natural images

Generated image, from “Large Scale GAN Training for High Fidelity Natural Image Synthesis”, Brock et al.

Even worse, it hasn't really figured out what human faces look like. This is likely because the dataset it was trained on has a relatively low proportion of images of people’s faces. So while the results are quite impressive, there's still work to be done.

== treaty of paris (1763)

the treaty of paris, also known as the treaty of 1763, was signed on 10 february 1763 by the kingdoms of great britain, france and spain, with portugal in agreement, after great britain's victory over france and spain during the seven years' war.

the signing of the treaty formally ended the seven years' war, known as the french and indian war in the north american theatre,[1] and marked the beginning of an era of british dominance outside europe.[2] great britain and france each returned much of the territory that they had captured during the war, but great britain gained much of france's possessions in north america. additionally, great britain agreed to protect roman catholicism in the new world...

This same paradigm can be applied to other domains outside of natural images. For example, let's say we collect articles from Wikipedia.

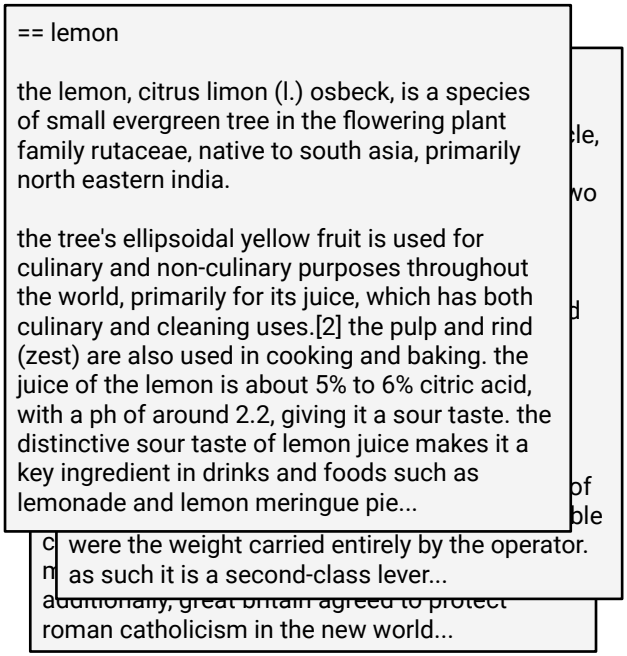
==

roman catholicism in the new world...

== wheelbarrow

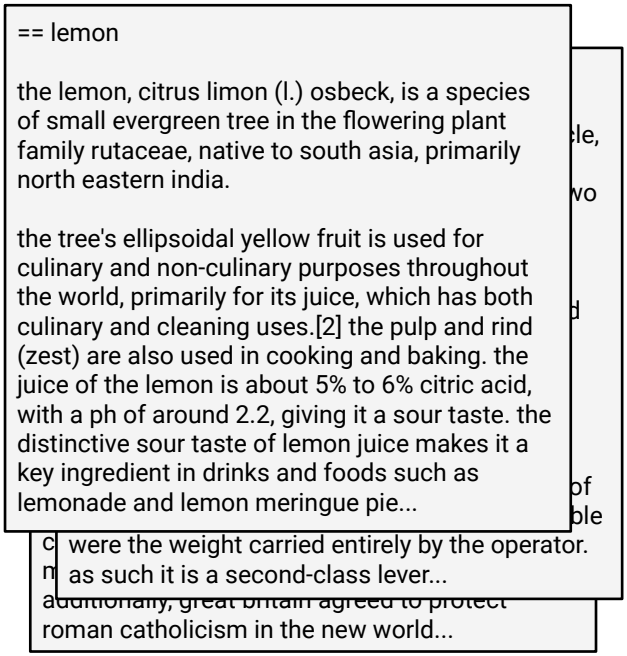
a wheelbarrow is a small hand-propelled vehicle, usually with just one wheel, designed to be pushed and guided by a single person using two handles at the rear, or by a sail to push the ancient wheelbarrow by wind. the term "wheelbarrow" is made of two words: "wheel" and "barrow." "barrow" is a derivation of the old english "bearwe" which was a device used for carrying loads.

the wheelbarrow is designed to distribute the weight of its load between the wheel and the operator, so enabling the convenient carriage of heavier and bulkier loads than would be possible were the weight carried entirely by the operator. as such it is a second-class lever...



Dataset of Wikipedia articles

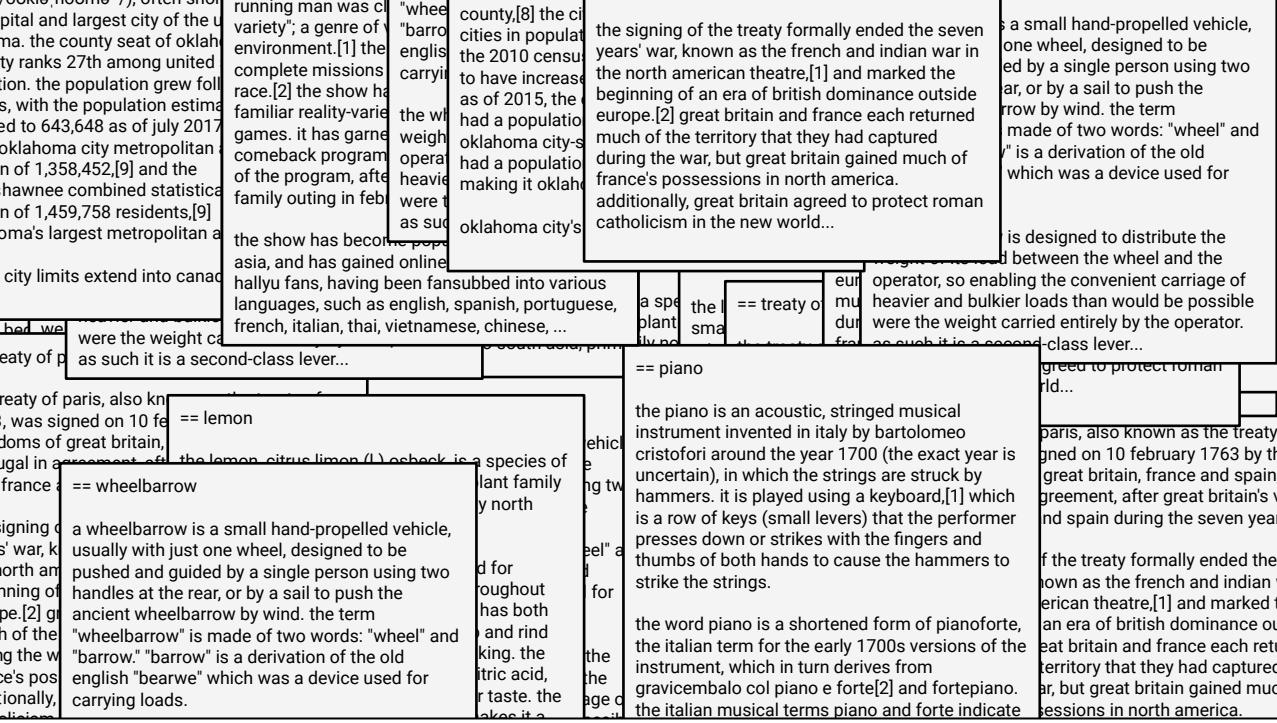
Once we have a large dataset of articles…



Dataset of Wikipedia articles

Generated article, from "Generating Wikipedia by Summarizing Long Sequences", Liu et al.

we can train a generative model to generate new Wikipedia articles. This example comes from work done by researchers at Google Brain. In this case, the generated article discusses a made-up album called Wings over Kansas. The model was only given the title of the article and filled in the rest. You can see that the model has learned not only basic sentence structure, grammar, and spelling, but also the kind of content which is typical in Wikipedia articles. It also stays on theme, though it does end up generating some nonsense near the end.



While it's fun to train a model to generate new made-up Wikipedia articles from scratch, it's not terribly useful on its own since there are already so many Wikipedia articles out there already. One of the things that makes generative models useful in practice is that they provide a means to leverage and discover structure in large unlabeled datasets. In machine learning, we often find gains simply from collecting more data. In many applications like modeling text or natural images, unlabeled data is almost limitless.

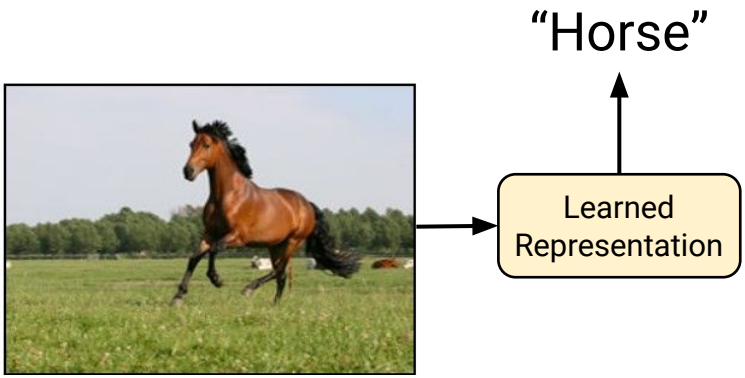
DATASET	METRIC	OUR RESULT	PREVIOUS RECORD	HUMAN
Winograd Schema Challenge	accuracy (+)	**70.70%**	63.7%	92%+
LAMBADA	accuracy (+)	**63.24%**	59.23%	95%+
LAMBADA	perplexity (-)	**8.6**	99	~1-2
Children's Book Test Common Nouns (validation accuracy)	accuracy (+)	**93.30%**	85.7%	96%
Children's Book Test Named Entities (validation accuracy)	accuracy (+)	**89.05%**	82.3%	92%
Penn Tree Bank	perplexity (-)	**35.76**	46.54	unknown
WikiText-2	perplexity (-)	**18.34**	39.14	unknown
enwik8	bits per character (-)	**0.93**	0.99	unknown
text8	bits per character (-)	**0.98**	1.08	unknown
WikiText-103	perplexity (-)	**17.48**	18.3	unknown

From "Language Models are Unsupervised Multitask Learners", Radford et al.

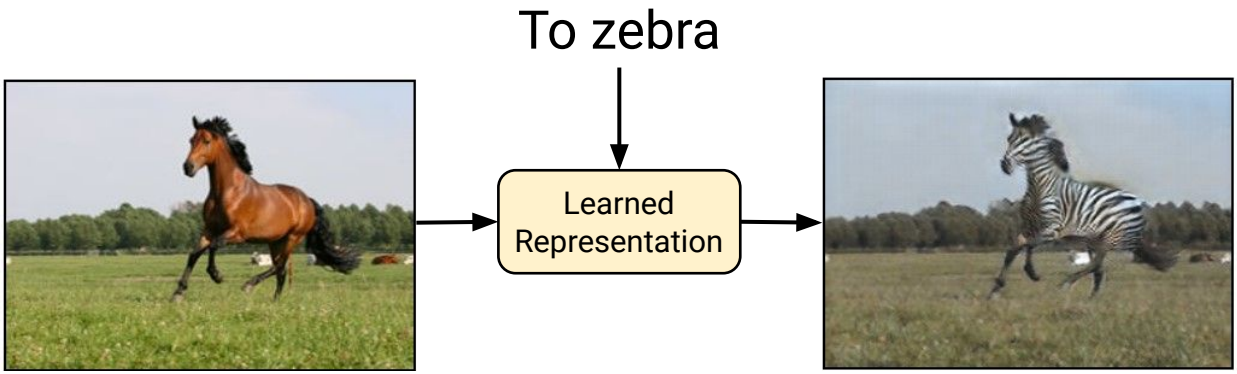
A good example of the power of this is some recent work from OpenAI, where they demonstrate that by training a large generative text model on a huge corpus of documents from the web, they obtain state-of-the-art results on many common text benchmarks. These results are obtained with a simple type of generative model which is particularly effective at learning about the structure of text and language from unlabeled data alone. These impressive results are largely thanks to the fact that they were able to trained a large generative model on lots of unlabeled data.

Learned Representation

Another common way to use generative models is for representation learning. In representation learning, our goal is to find a way to transform raw data into a format which is more amenable for downstream tasks.

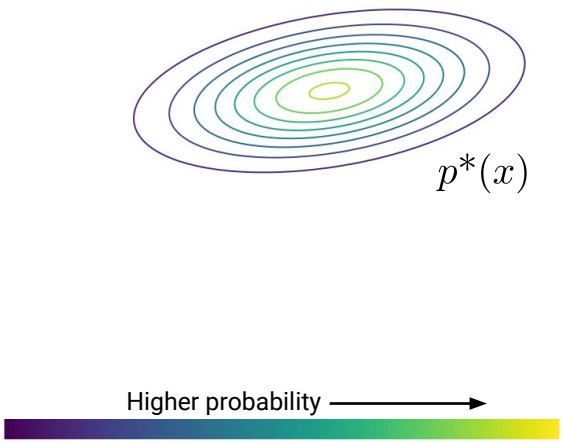


For example, it might be easier to classify the contents of this image as being a photo of a horse by using this learned representation than to perform classification from the raw pixels alone. The hope with generative models is that they can uncover structure in the true distribution which is helpful for these downstream tasks.

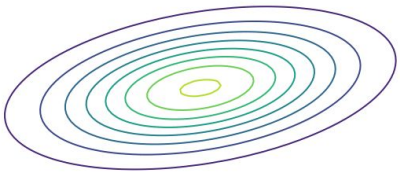


From "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", Zhu et al.

In some cases, we can also use the learned representation of a generative model to transform existing data. In this example, Zhu et al. show how a generative model can be used to learn to transform images of horses to images of zebras. Again, the model learns to do this without supervision, which allows it to leverage large collections of unlabeled data. In this talk, I'll discuss ways that **generative models can learn representations for transforming sequences,** and show how being able to perform this kind of transformation implies that the **learned representation has captured important structure about the true distribution**.

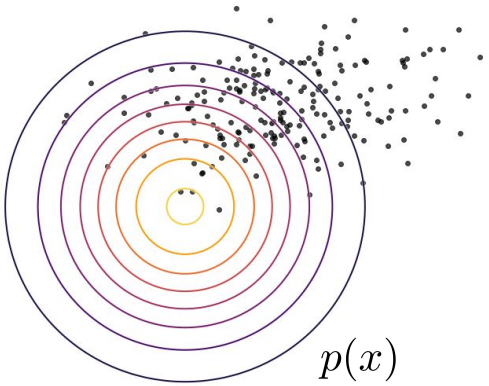


To get an idea as to how these models were created, let's start with the basics. As I mentioned, we start by assuming there is some true underlying distribution which we call p star x. Here the true distribution is simply a Gaussian in two dimensions. The contour lines show the probability of different regions, from low probability in purple to higher in yellow.

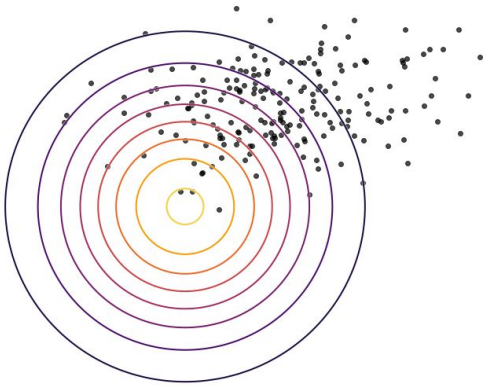


$$x \sim p^*(x)$$

Now, as I mentioned in practice we don't have access to the true distribution. Instead, we have access to samples from it. In this case, let's say we are given a few hundred samples from the true distribution, shown as black dots here.

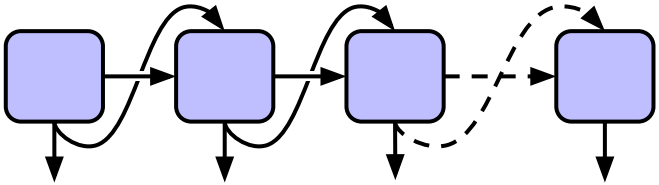


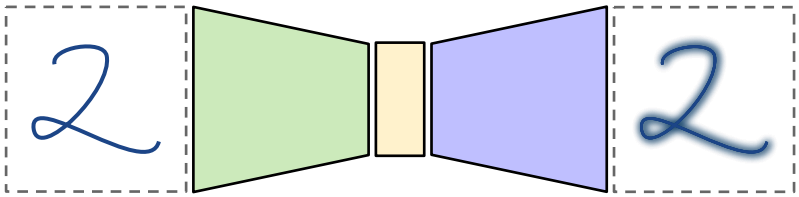
Now, our goal is to train our generative model, which I'm calling p(x), to recover the true distribution. We'll do this by changing the parameters of the generative model so that it matches the samples we drew from the true distribution.

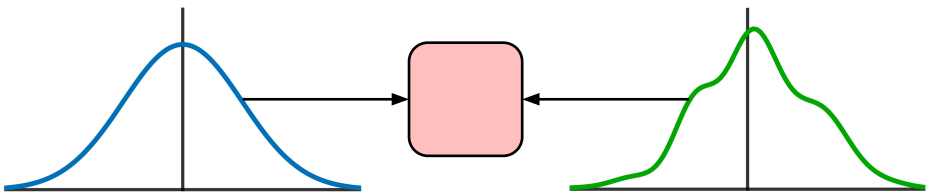


$$\min \mathrm{KL}(p^* || p) \rightarrow \max \mathbb{E}_{x \sim p^*}[\log p(x)]$$

The most common way to do this is to use a divergence which measures how well our model fits the samples. A common choice is the KL divergence. We can see here that if we adjust the parameters of our model to minimize the KL divergence between the model and the samples, we end up with a good fit. If you're familiar, minimizing the KL divergence in this way is equivalent to maximum likelihood estimation, which has a simple intuition: We want to modify our model so that it assigns high probability to all of the samples from the true distribution.







Ok, now that we've had some introduction and motivation on generative modeling, I'll move on to discussing some of the dominant paradigms in this field. First, I'll talk about autoregressive models, which are trained for next-step prediction. Then, I'll discuss variational autoencoders, which learn to map their input to a compressed representation. Finally, I'll discuss adversarial learning, which allows us to fit models from which we can only draw samples. Throughout the talk, I'll cover my work which addresses various shortcomings of these methods, enables new applications, and provides insight into their behavior.

The

To introduce autoregressive models, let's do a small exercise. I'd like you to predict what word comes next in this sentence. So, starting with the word the...

The cat

---

then cat...

The cat sat

---

sat...

The cat sat on

---

on...

The cat sat on the

---

the...

The cat sat on the mat.

---

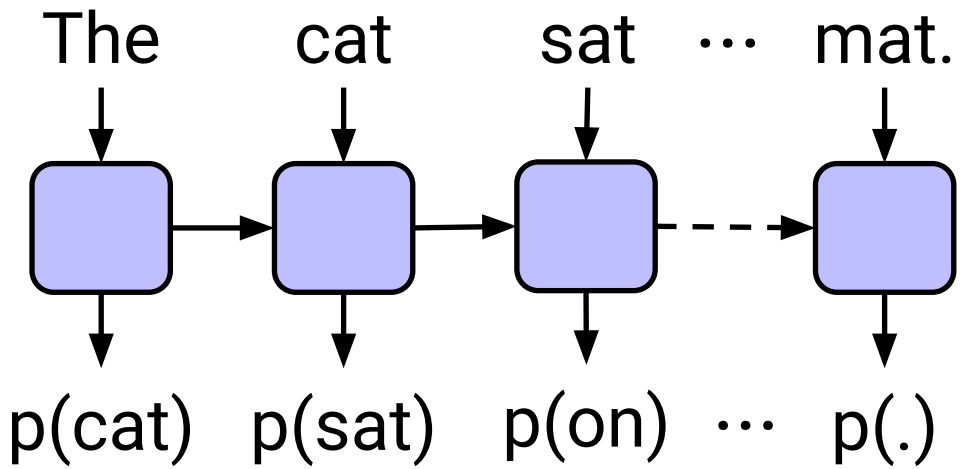
mat. Great, so many of you figured out pretty quickly where this sentence was going.

p(mat|The,cat,sat,on,the)

What you all were doing, roughly, was predicting a probability distribution over the next word in the sentence given the previous words. By the end of the sentence, you probably were predicting that the word "mat" had a probability near 1 and every other word had relatively low probability.

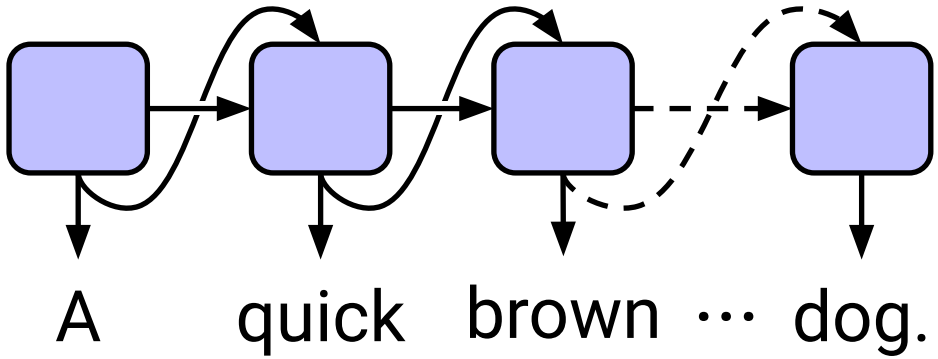
$$p(y_t | y_{t-1}, y_{t-2}, \ldots, y_2, y_1)$$

This is, in fact, fundamentally what autoregressive models do. Given a sequence, which here we are calling y_1, y_2, ... and so on, an autoregressive model tries to predict y_t from all of the sequence entries before y_t.



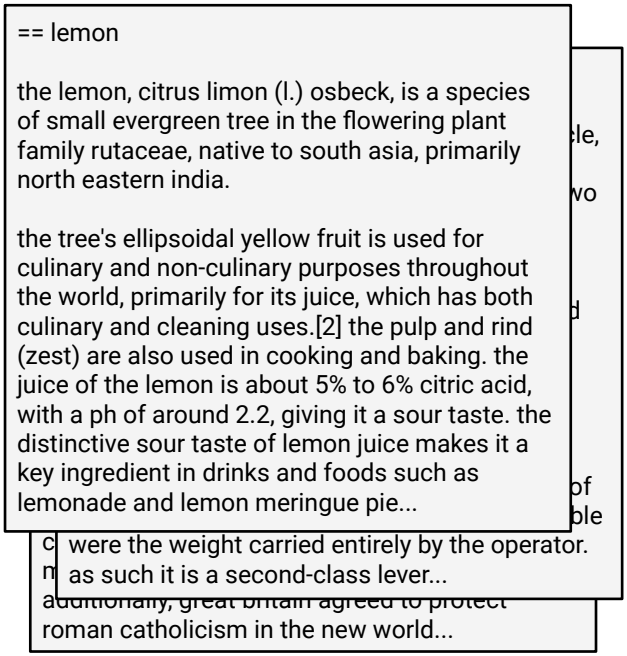
$$\max \mathbb{E}[\log p(y_t | y_{t-1}, y_{t-2}, \ldots, y_2, y_1)]$$

The autoregressive models I'll be discussing today are implemented with a particular kind of model called a recurrent neural network. Without getting into details, a recurrent neural network ingests an entry in the sequence at each timestep and uses this new input to update an internal state. It then outputs a probability distribution over the next entry in the sequence. These models are trained with maximum likelihood - we draw sample sequences from our training set, and adjust the parameters of the recurrent neural network so that it assigns a higher probability to the correct output, given the past inputs. So overall the framework is unchanged from the simple 2D Gaussian case we discussed earlier - we have a model, we have a training set, and we train our model to assign high probability to the training set.



$$y_1 \sim p(y_1), y_2 \sim p(y_2|y_1), y_3 \sim p(y_3|y_2, y_1) \dots$$

Now, to use an autoregressive model as a generative model, we can proceed like so: First, we compute the probability distribution of the first output, and then sample a first element for our sequence. Then, we take that sample and feed it back in to the model, and then compute the probability distribution over the second sequence element. Then, we sample the second sequence entry and feed it back in, and so on, continuing to generate our sequence by sampling from the model sequentially.



Dataset of Wikipedia articles

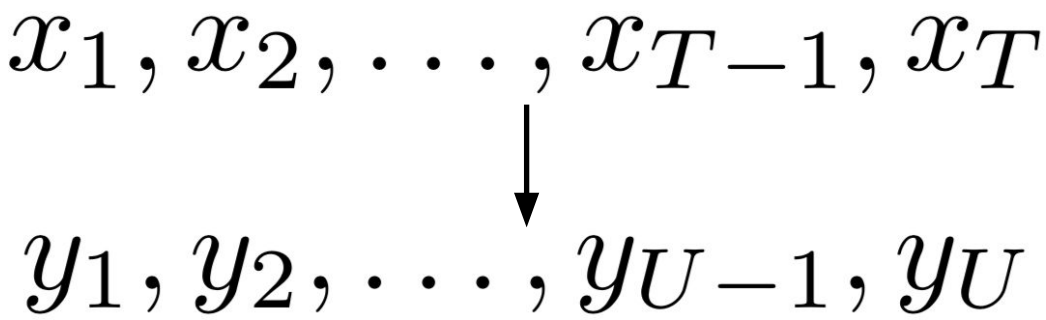
Generated article, from "Generating Wikipedia by Summarizing Long Sequences", Liu et al.

This is actually how the Wikipedia article we saw at the beginning was generated -- a large autoregressive model was trained to predict the next word in a collection of Wikipedia articles, and then a new article was sampled word-by-word in the same procedure we just described. So while the overall approach is somewhat simple, it can be very powerful.

John is a big orange cat.

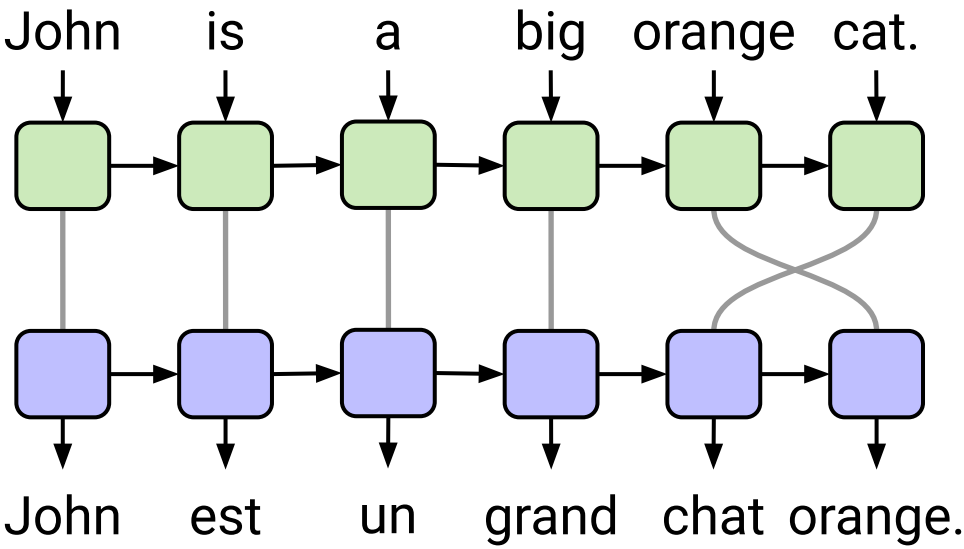
John est un grand chat orange.

In most applications, it's not so important that we are simply able to generate new sequence. We often want to do more complex things, like transform an existing sequence. For example, we might want to take an English sentence and have our generative model predict the French translation of it.



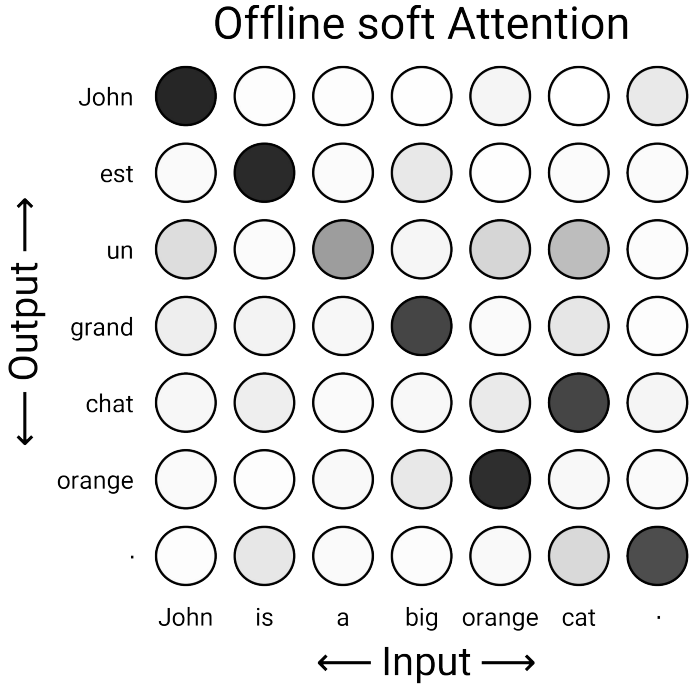
$$\max \mathbb{E}[\log p(y_t | y_{t-1}, \ldots, y_1, x_T, \ldots, x_1)]$$

We can achieve this with only a small modification over the standard autoregressive model. Say we have an input sequence x_1, x_2, and so on, and we're trying to transform it into a new output sequence y_1, y_2, ... To train a model to do this, we will still predict the entries of y one entry at a time, but we will condition not only on the previous entries of y but also on the entirety of the sequence x. So effectively we are just providing the input sequence as additional information, but we are still doing next-step prediction on the output sequence.



$$\max \mathbb{E}[\log p(y_t | y_{t-1}, \ldots, y_1, x_T, \ldots, x_1)]$$

In practice this is usually achieved through what is called an attention mechanism. The attention mechanism allows the model to refer back to entries in the input sequence as it produces its output sequence. To do so, the input sequence is fed through one recurrent neural network, called an encoder. Then, the output sequence is generated by another recurrent neural network called the decoder. At each output timestep, the decoder produces a soft weight over the entries of the input. This weight corresponds to how much the decoder is conditioning on, or paying attention to, a given entry of the input sequence.



One of the nice things about using attention is that it allows us to visualize the behavior of our model. This diagram shows the attention produced by the decoder at each output timestep, as the model translates the english sentence on the bottom into the French translation on the left. Along the y axis are the output words; along the x axis are the input words. Darker cells indicate a higher weight. You can see that when the model is translating a given word, it tends to assign a high weight to it, and in particular when the adjective/noun order swaps in French the attention weights indicate this. You can see from this diagram that t**he decoder assigns a weight to every entry in the input at every output timestep**. This causes attention to have a quadratic time and space complexity. You can also see that in order to start producing the output, the model must have finished processing the entire input sequence, since it must assign a weight to every input entry even at the first output timestep. This prevents attention-based models from being applied in online settings, where we want to transform an input sequence as it's still being generated.

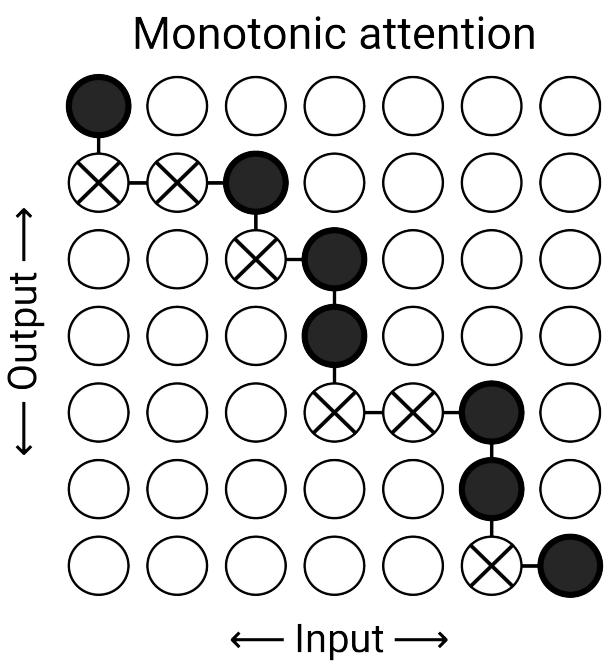
Is there a way to use attention to transform a sequence as it's being generated?

Online and Linear-Time Attention by Enforcing Monotonic Alignments, ICML 2017.
Monotonic Chunkwise Attention, ICLR 2018.
Monotonic Infinite Lookback Attention for Simultaneous Machine Translation, ACL 2019 (to appear).

Wei Li | Naveen Ari | Thang Luong | Peter J. Liu | Ron J. Weiss | Douglas Eck | Ruoming Pang | Semih Yavuz | Chung-Cheng Chiu | Wolfgang Machery | Colin Cherry

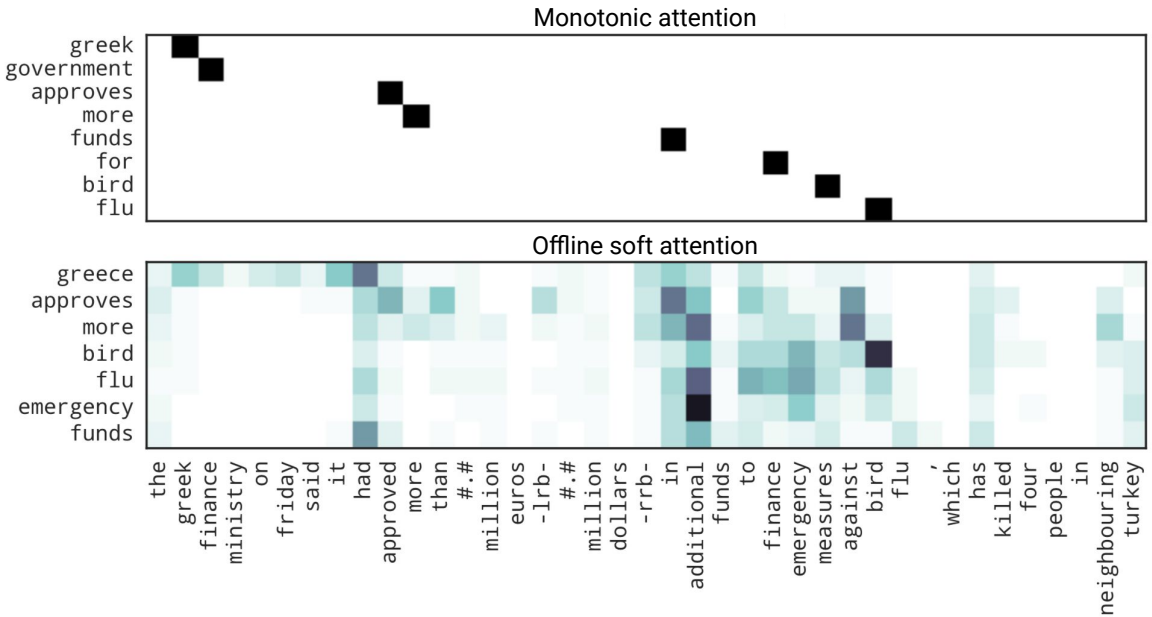
---

In this part of the talk, I'll discuss my work which addresses these drawbacks of standard offline attention. In this series of papers, I developed attention mechanisms which allow for **online and linear-time transformation of sequences**, all while providing **better interpretability and without sacrificing accuracy**.



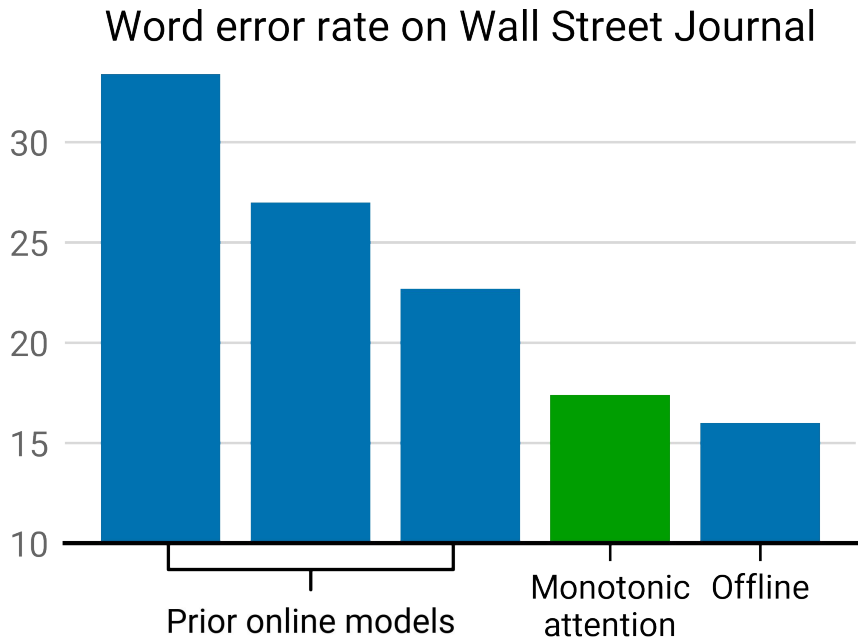
Online and Linear-Time Attention by Enforcing Monotonic Alignments, ICML 2017.

The first attention mechanism I developed, called monotonic attention, explicitly processes the input sequence from left-to-right instead of assigning a weight to every entry in the input at each output timestep. For each entry in the input sequence, we make a binary decision as to whether to stop and attend to that entry or to ingest a new element from the input sequence. Once we attend to a given entry in the input, we output a new element in the output sequence and start from where we left off at the next output timestep. This means we only ever make one pass over the input sequence which makes this approach linear-time. We also ingest the input on-the-fly, making this applicable to online sequence transduction problems.



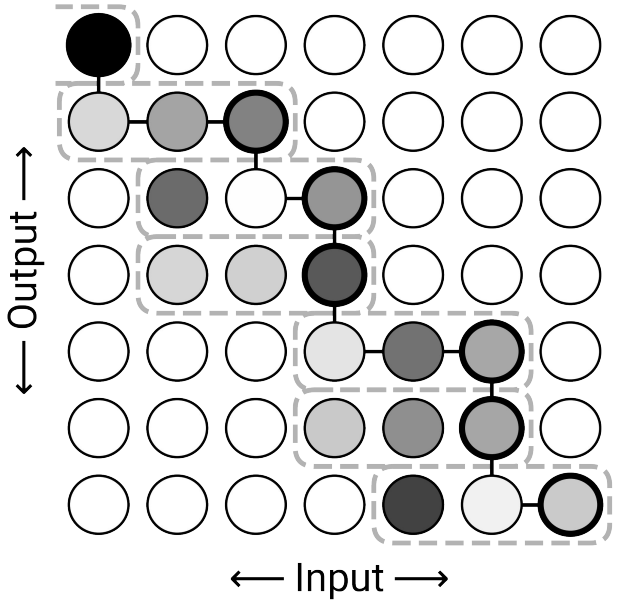
Online and Linear-Time Attention by Enforcing Monotonic Alignments, ICML 2017.

Here's an example of what happens when we apply this approach to a sentence shortening task, where the goal is to take in a long sentence and output a shortened version of it. In this case, the monotonic attention shown on the top works mostly how we'd expect - the model attends to "greek" when it outputs "greek", "finance" when it outputs "government", "approved" when it outputs "approves", and so on. The offline soft attention model, shown on the bottom, is much harder to interpret. By forcing the model to attend to a single entry in the input, we can end up with a more interpretable input-output alignment.



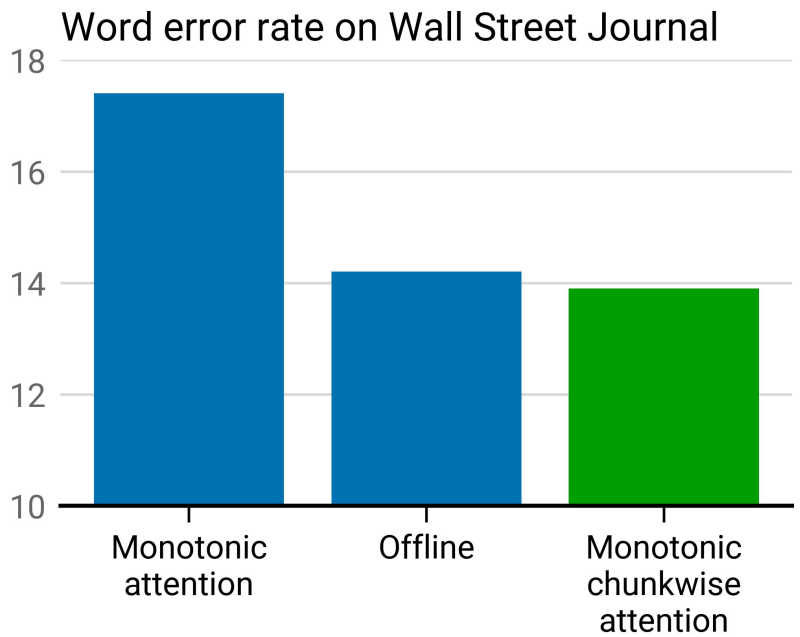
Online and Linear-Time Attention by Enforcing Monotonic Alignments, ICML 2017.

When we applied this approach to a common speech recognition task, we found that it outperformed all existing online sequence transduction models by a significant amount. However, in terms of absolute performance it still lagged slightly behind just using standard, offline soft attention. This was a common trend - we were not able to completely recover the performance of standard attention using monotonic attention.



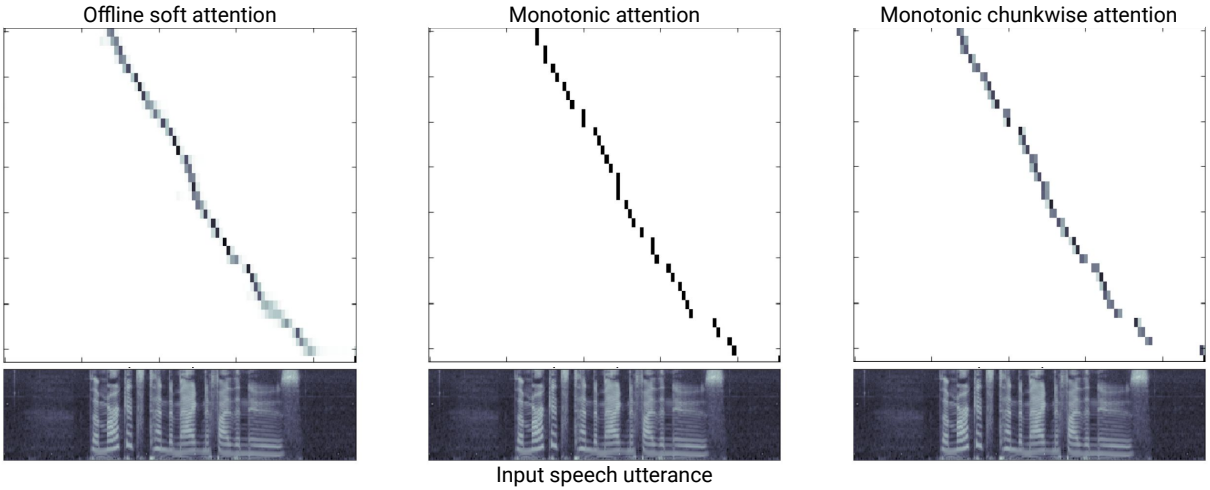
Monotonic Chunkwise Attention, ICLR 2018.

To get around this issue, we introduced a modification to monotonic attention where instead of attending to a single entry in the input, the model attends to a small fixed-length chunk of the input sequence at each output timestep. In this schematic, I'm showing a chunk size of three. The start position of the chunk moves according to a monotonic attention mechanism, but then the model is allowed to assign a non-zero weight to entries in the chunk preceding the element chosen by monotonic attention. This provides the model with some flexibility, while still being online and linear time.



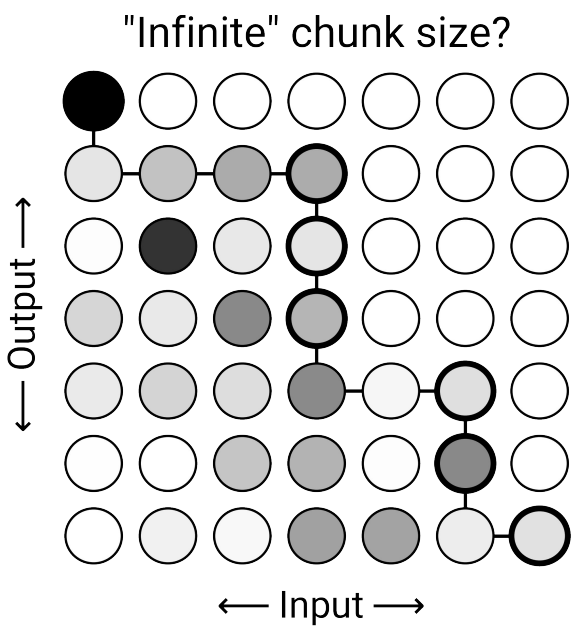
Monotonic Chunkwise Attention, ICLR 2018.

With this modification, we found that we were actually able close the gap between monotonic attention and offline soft attention, and in fact we slightly outperformed standard offline soft attention on the same speech recognition task. This is likely because the monotonic constraint provides an implicit bias which is well-matched to the needs of speech recognition.



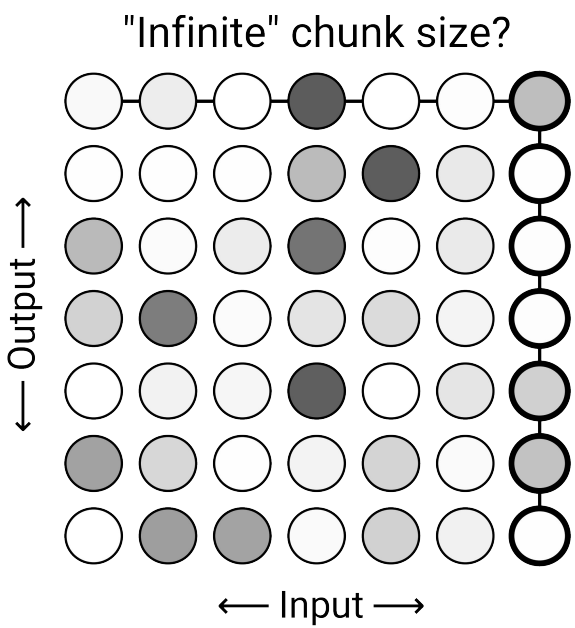
Monotonic Chunkwise Attention, ICLR 2018.

Here are examples of the different attention mechanisms on this speech task. The input is a speech utterance, whose spectrogram is shown on the bottom. On the left, we see standard offline soft attention. The input-output alignment is mostly monotonic, though the model does spread out its attention a little bit at each output timestep. In the middle is monotonic attention, which is forced to attend to a single entry in the input at each output timestep. Finally, on the right is monotonic chunkwise attention, which allows the model to attend to a few frames at a time.



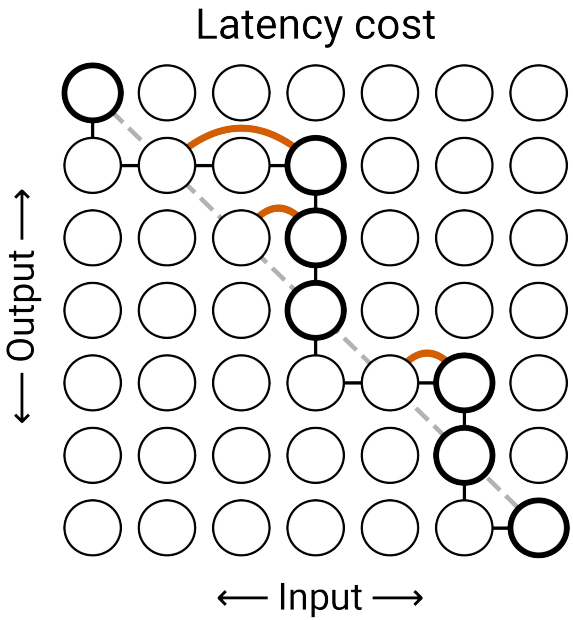
Monotonic Infinite Lookback Attention for Simultaneous Machine Translation, under review.

A shortcoming of monotonic chunkwise attention is that only allows a limited amount of reordering in the input-output alignment. More flexibility in being able to reorder can be useful when doing machine translation and the word ordering between languages is different. A natural extension would be to allow the model to attend to the entirety of the input sequence preceding the monotonic attention location, producing a sort of “infinite chunk size”, as is shown here. While this would no longer be a linear-time attention mechanism, it still would allow for online transformation of sequences. Note that **in order to allow for reordering, we will need to introduce some latency**, but ideally we will be able to trade-off latency for quality and still achieve roughly “streaming” transduction even with some reordering.



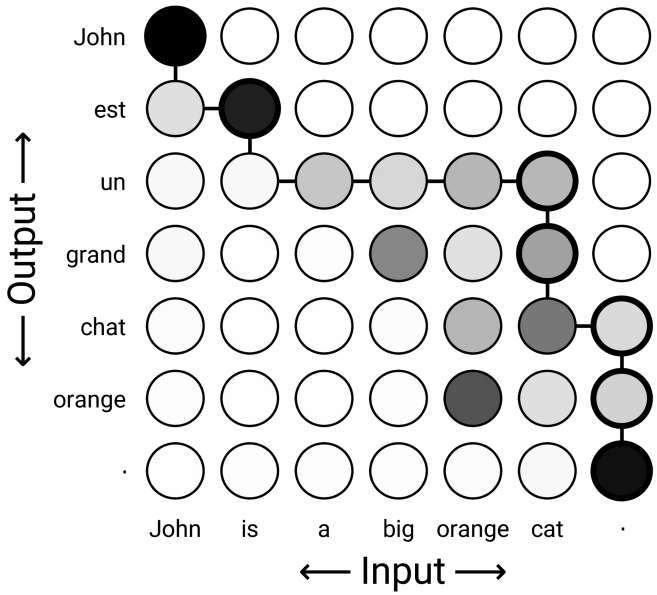
Monotonic Infinite Lookback Attention for Simultaneous Machine Translation, under review.

Unfortunately, naively applying this idea results in the pathology shown here. At the first output timestep, the monotonic attention mechanism simply ingests the entire input sequence. At subsequent output timesteps, the model is free to attend to the entire input sequence. This effectively makes the model regress to standard, offline attention, and makes the use of a monotonic attention mechanism useless.



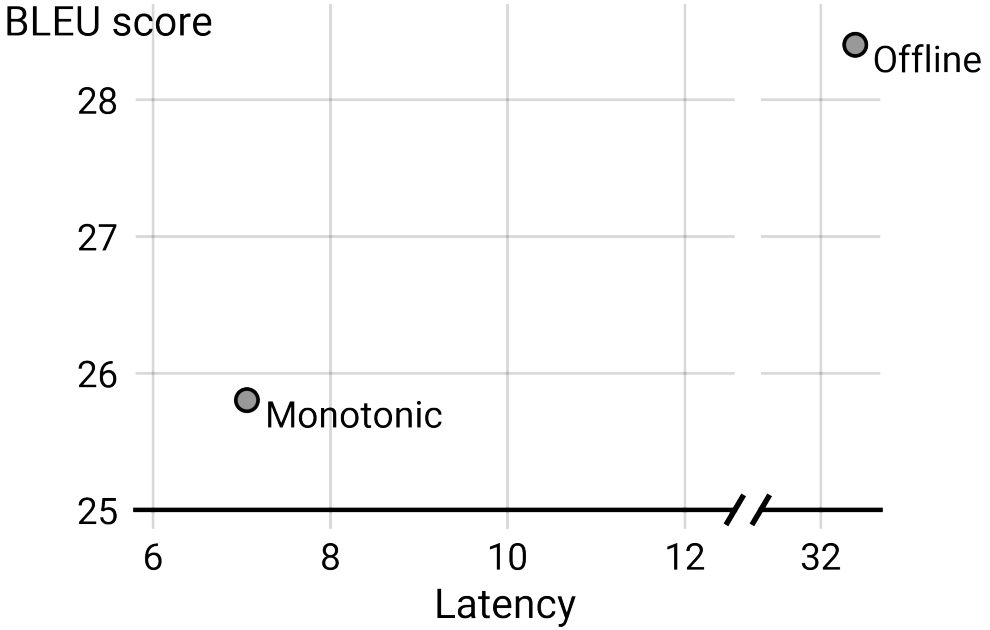
Monotonic Infinite Lookback Attention for Simultaneous Machine Translation, under review.

To address this issue, we impose an additional cost on the model. To motivate this cost, consider that the lowest-latency case is one where the model produces an element of the output sequence every time it ingests a new entry from the input sequence. This would be represented by an alignment that follows the diagonal line shown in this figure. Any extent to which the model ingests more of the input sequence than this implies some latency that we could ideally to get rid of. So, we can produce a latency cost by simply measuring the amount to which the monotonic attention mechanism attends to entries off the diagonal. This is shown as the red arcs in this figure. In this case, the latency cost would be 2, 1, and 1 timesteps, corresponding to those three red arcs. We can encourage the model to be lower latency by increasing the weight of this additional cost while training the model.



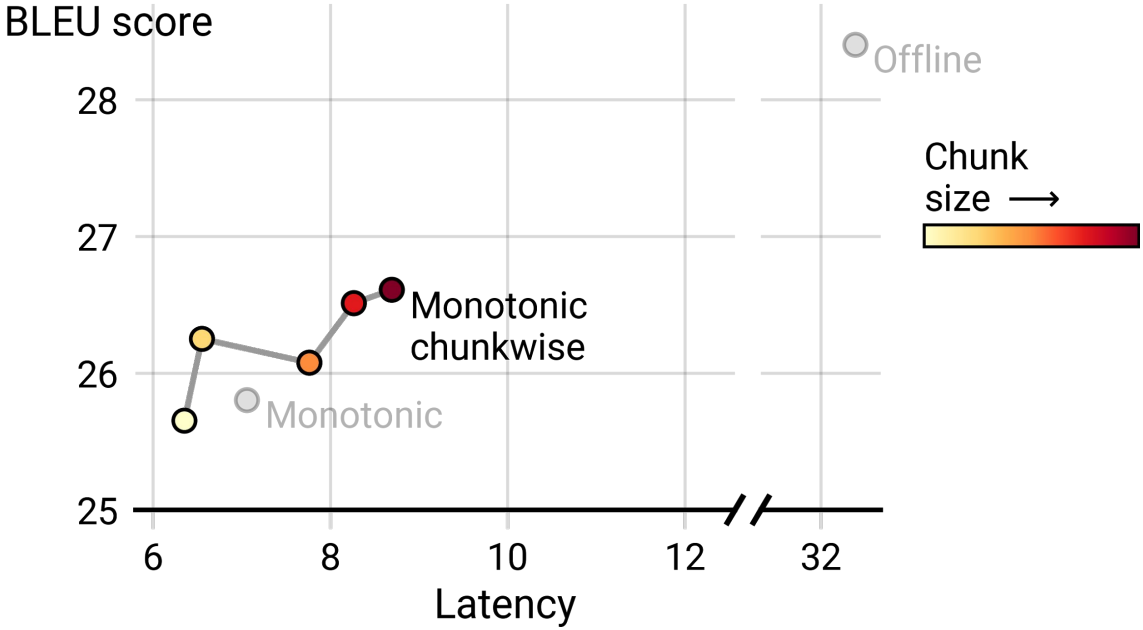
Monotonic Infinite Lookback Attention for Simultaneous Machine Translation, under review.

So, if we apply monotonic attention, infinite lookback, and our latency cost, we get monotonic infinite lookback attention, and here's the alignment it produces for the sentence we were looking at earlier. You can see that it does manage to produce a low-latency alignment and still get the translation correct. Notice how the monotonic attention chose to ingest the entire input sequence up to the word "cat" when producing the third french word, the article en, which is a translation of the English word "a". This is because the model needs to know what noun the article refers to before it can choose the gender of the article, whether it should be en or oon. If the monotonic attention mechanism did not ingest all of these tokens, it would have had to guess the gender of the noun when producing the translation of the article "a".



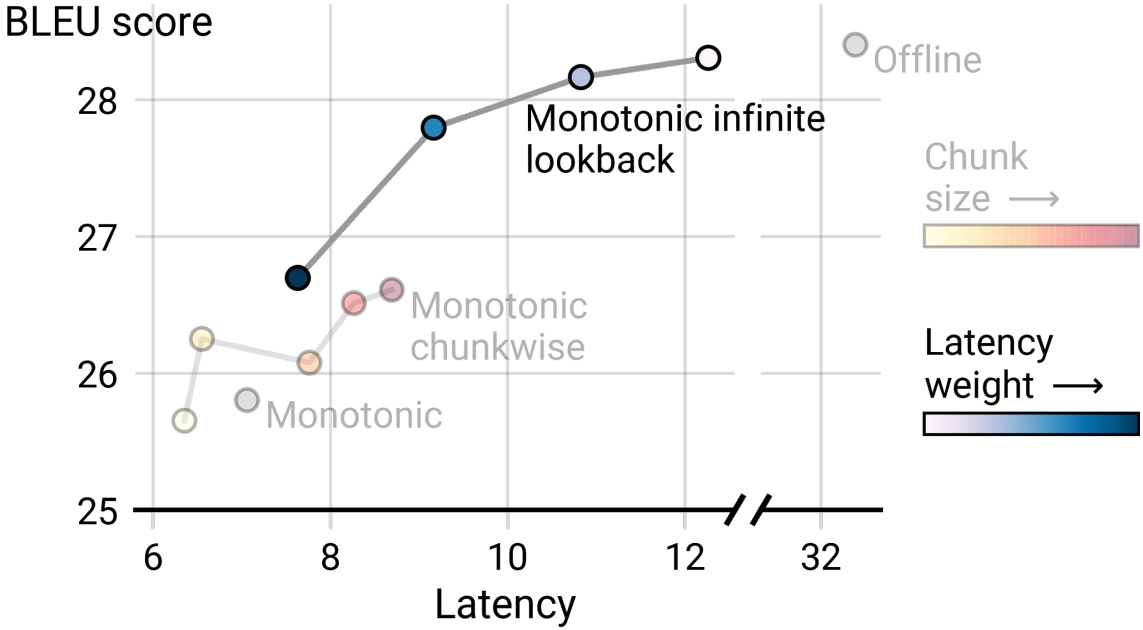
Monotonic Infinite Lookback Attention for Simultaneous Machine Translation, under review.

So, how well does this work in general? We can characterize how well these different systems perform by measuring their accuracy against their latency. For example, on this standard English to French translation corpus, a standard offline attention model has a latency of 32 words, corresponding to the average input sequence length. Monotonic attention manages to have much lower latency of around 7 words, but its BLEU score (a measure of translation quality) is much worse.



Monotonic Infinite Lookback Attention for Simultaneous Machine Translation, under review.

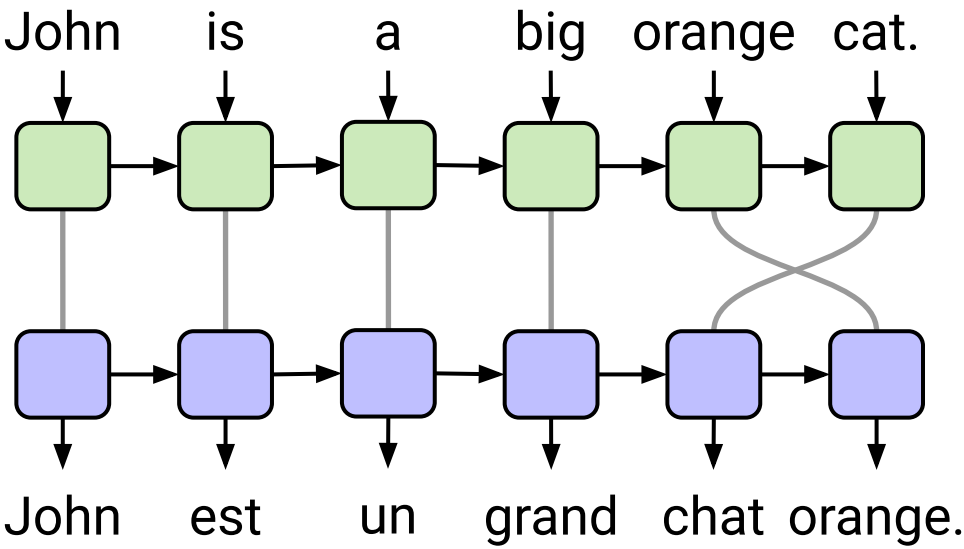
If we apply monotonic chunkwise attention to this task, we can see that we can get some degree of a trade-off between latency and quality. As we increase the chunk size, the model is given more context and is able to translate with better quality, albeit with higher latency.



Monotonic Infinite Lookback Attention for Simultaneous Machine Translation, under review.

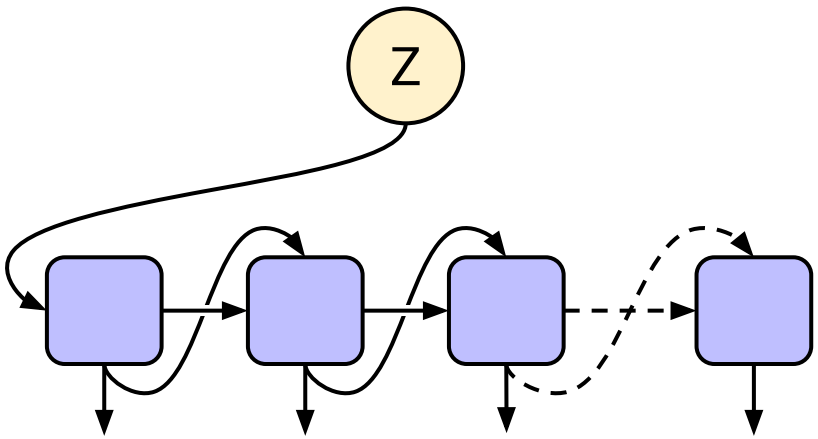
Finally, if we apply monotonic infinite lookback attention to this task, we can see that as we decrease the latency weight the latency increases and the model quality also increases. This provides a direct trade-off between how “online” the model is and how well it’s translating. Importantly, we can see that we can roughly recover the performance of an offline attention mechanism with a much lower average latency of around 12.

$$p(y_t|y_{t-1}, \ldots, y_1, x_T, \ldots, x_1)$$



So far we have been talking about conditional autoregressive models which predict each element of the output sequence based on past entries of the output and conditioned on some input sequence. By conditioning on an input sequence, we can control what the generative model produces.

$$p(y_t|y_{t-1}, \ldots, y_1, z)$$



Now, let's move on to a different kind of conditioning. Rather than conditioning on an input sequence, we will condition on a stochastic latent variable which we will call z. We want this latent variable z to **capture the important high-level characteristics of a given datapoint**. Just like when we were conditioning on an input, the output is mostly determined once we sample a value for z - the model only needs to fill in the lower-level details. As I'll show, this also makes the latent variable a useful learned representation for intuitive manipulation of data and other downstream tasks.

$$\log p(x) = \log \int p(x|z)p(z)\,\mathrm{d}z$$

So, if we build up a model in this way, the log probability of our model is decomposed as shown here. We can obtain the log probability of x only if we marginalize out z, which x depends on. Specifically, in order to compute log p(x), we need to integrate out all possible values of our latent variable.

$$\log p(x) = \log \int p(x|z)p(z)\,\mathrm{d}z$$

Unfortunately, we can't do this in practice - it's simply too expensive to consider every possible value that the latent variable can take.

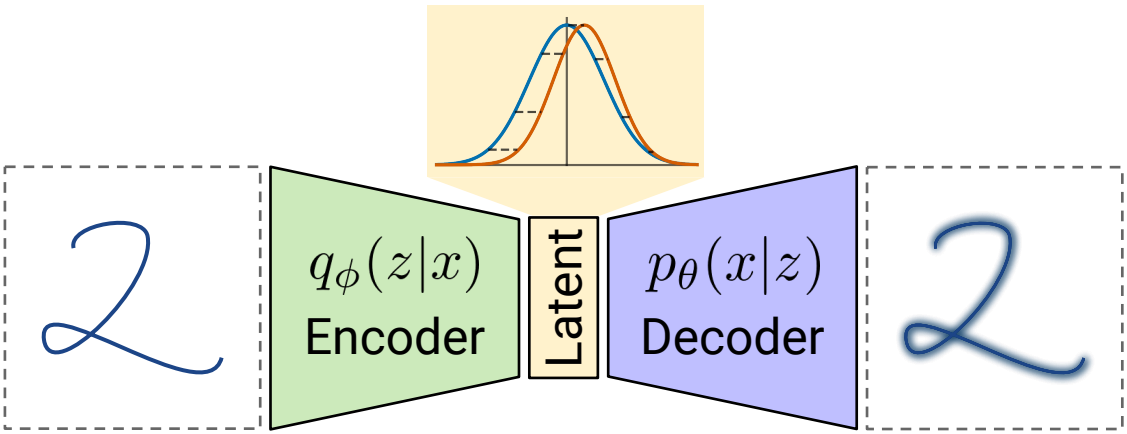
$$\log p(x) \geq \log p(x) - \mathrm{KL}(q_\phi(z|x)||p(z|x))$$

So instead, we'll **introduce a model q phi** which can predict the value of the latent variable for a given input. This is called an approximate posterior, because it approximates the true posterior of the latent variable given the input. Ideally, we want this approximate posterior to match the true posterior p(z | x), which is expressed by measuring the KL divergence between the two. We want this KL divergence to be small, and the log probability to be high, so we can subtract the KL divergence from our log probability as shown here. This forms a lower bound on the log probability since the KL divergence is non-negative. Making this approximation is called variational inference.

$$\log p(x) \geq \log p(x) - \mathrm{KL}(q_\phi(z|x)||p(z|x))$$
$$= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)||p(z))$$

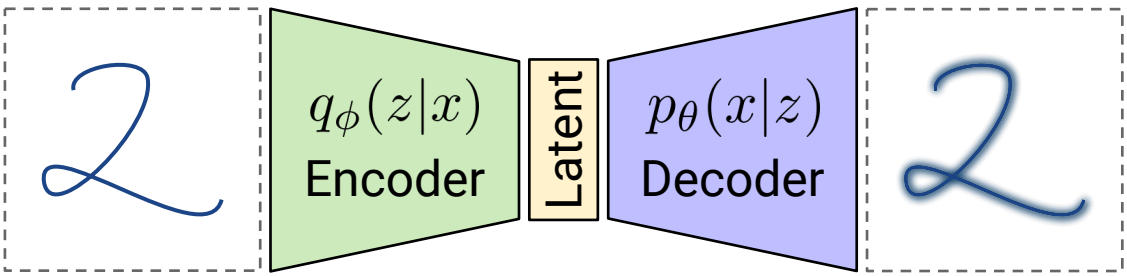
After some algebra, we can show this is equivalent to the following expression, which is called the ELBO or evidence lower bound. Let's unpack this expression a bit.

$$\log p(x) \geq \log p(x) - \mathrm{KL}(q_\phi(z|x)||p(z|x))$$
$$= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\phi(z|x)||p(z))$$

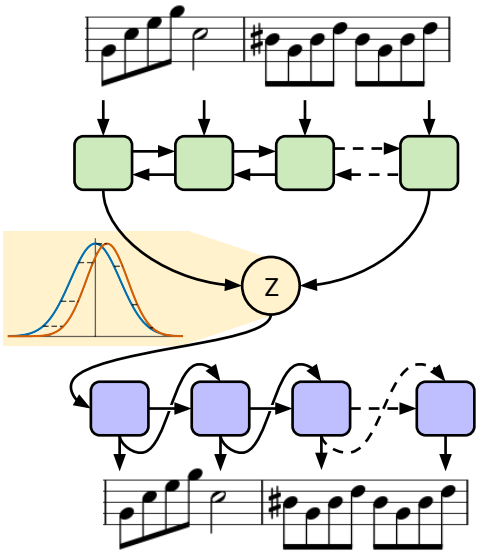


The form of this objective maps pretty directly onto a common neural network architecture called an autoencoder. An autoencoder first takes an input and encodes it to a compressed latent representation. This is achieved by a neural network called the encoder. The encoder produces a distribution over values of the latent code z given an input x, which is simply the approximate posterior we've been talking about q_\phi(z | x). The model then takes the latent code and decodes it back to the output, using a decoder which computes the distribution of data x given the latent variable z. The goal of the autoencoder is to successfully reconstruct its input x after it's been encoded and decoded. This reconstruction objective is reflected in the first term of the ELBO: If we encode an input as Z and then decode it, we want the model to assign high probability to the original datapoint. The second term in the ELBO encapsulates something different - we want the KL divergence between the approximate posterior and some simple prior distribution to be small. In essence, what this is saying is that **we want the distribution of latent codes to to be simple**. This means the latent code can't to store too much information about the input - we just want it to capture the important, high-level global information and then use the decoder to fill in the details.

$$\log p(x) \geq \log p(x) - \mathrm{KL}(q_\phi(z|x)||p(z|x))$$
$$= \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{Reconstruction}} - \underbrace{\mathrm{KL}(q_\phi(z|x)||p(z))}_{\text{Regularization}}$$

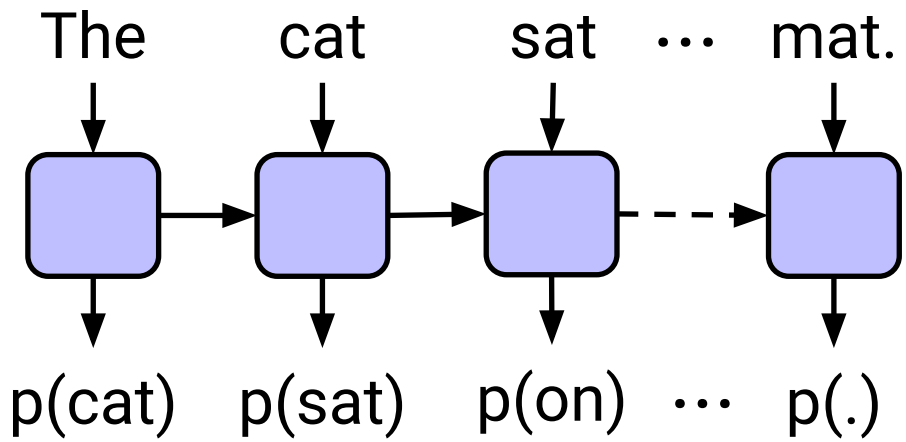


So to summarize, we can think of the first term as requiring that the autoencoder successfully reconstructs its input, and the second term as regularizing the learned representation so that it captures as little information as possible. Using this objective on this kind of model is referred to as a variational autoencoder or VAE



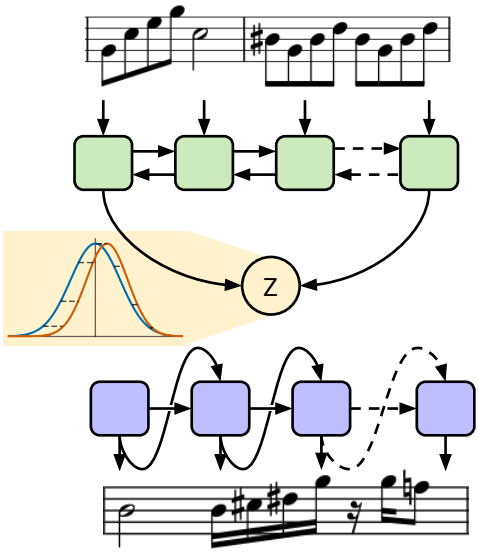
$$\mathbb{E}[\log p_\theta(x_t|x_{t-1},\ldots,x_1,z)] - \mathrm{KL}(q_\theta(z|x_T,\ldots,x_1)||p(z))$$

So for the remainder of this part of the talk, I'm going to be talking about our work on using variational autoencoders to model music. Our goal will be to feed in a sequence of notes, have it map to a compressed latent representation, and then be able to decode the result. Ideally, our latent representation will allow us to perform useful manipulations to input datapoints. Since we are processing sequences, we will continue using recurrent neural networks, here as the encoder and the decoder. In fact, our decoder will be an autoregressive model, which conditions on the latent code by using it as its initial state.



$$p(y_t|y_{t-1}, y_{t-2}, \ldots, y_2, y_1)$$

One thing to notice though is that as we discussed, autoregressive models on their own are quite good at modeling the true distribution. So using an autoregressive model as our decoder has the potential of making our model quite powerful.



$$\mathbb{E}[\log p_\theta(x_t|x_{t-1},\ldots,x_1,z)] - \mathrm{KL}(q_\theta(z|x_T,\ldots,x_1)||p(z))$$

However, what unfortunately ends up happening as a result of using a powerful autoregressive decoder is that the decoder is so good at modeling the data on its own that it effectively ignores the latent code. In other words, it tries to optimize the first term of the ELBO independent of the latent code, and then it drives the second term, the KL divergence, to zero. This is referred to posterior collapse, and it simply means that the model is not making use of the global latent code as we had intended.

Can we avoid posterior collapse and model long sequences by leveraging the hierarchical structure of music?

A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music, ICML 2018.
Learning a Latent Space of Multitrack Measures, NeurIPS Creativity Workshop 2018.


Adam Roberts

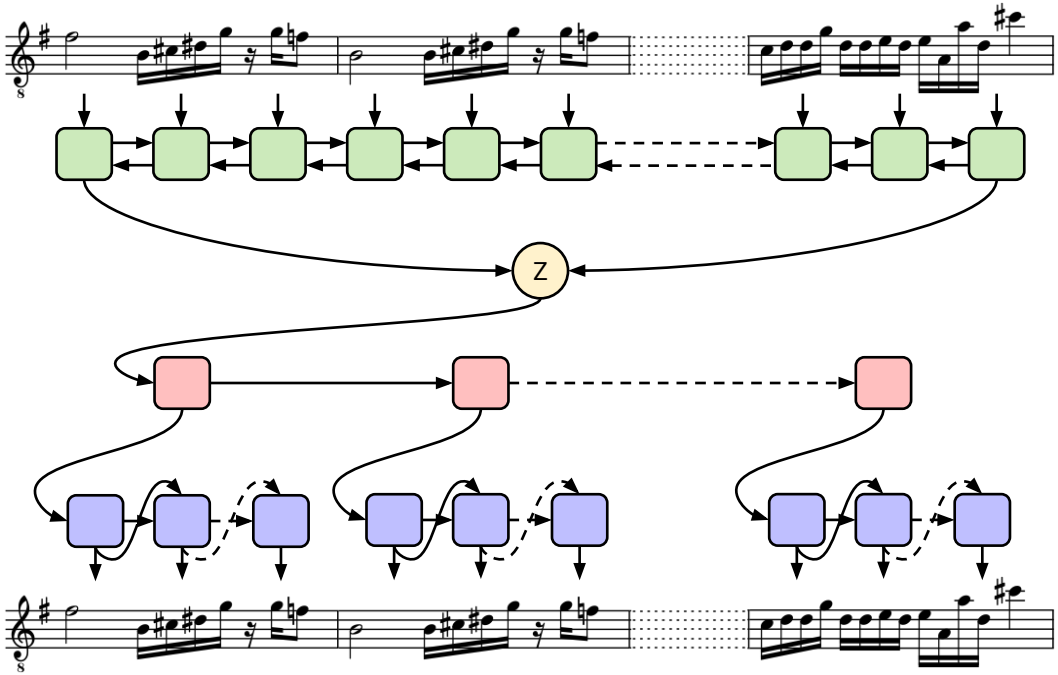

Jesse Engel


Fjord Hawthorne


Douglas Eck

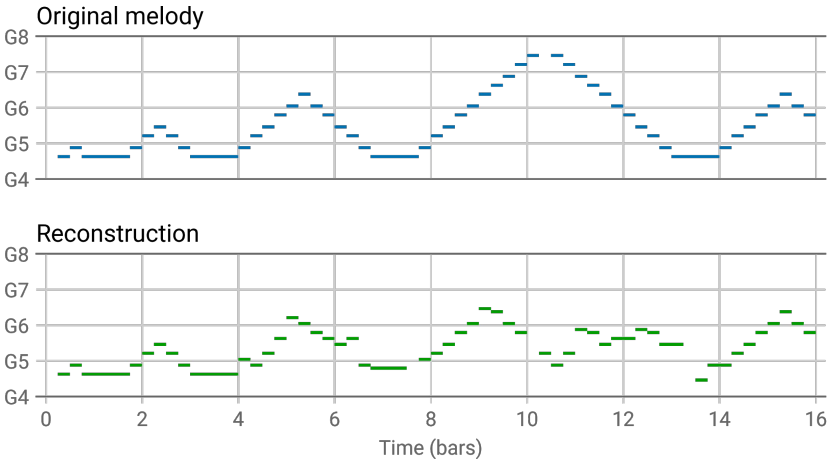

Ian Simon

In this part of the talk, I'll discuss my work on making VAEs able to model long sequences without exhibiting posterior collapse. To achieve this, I'll introduce hierarchical model architectures which explicitly reflect the structure of music.



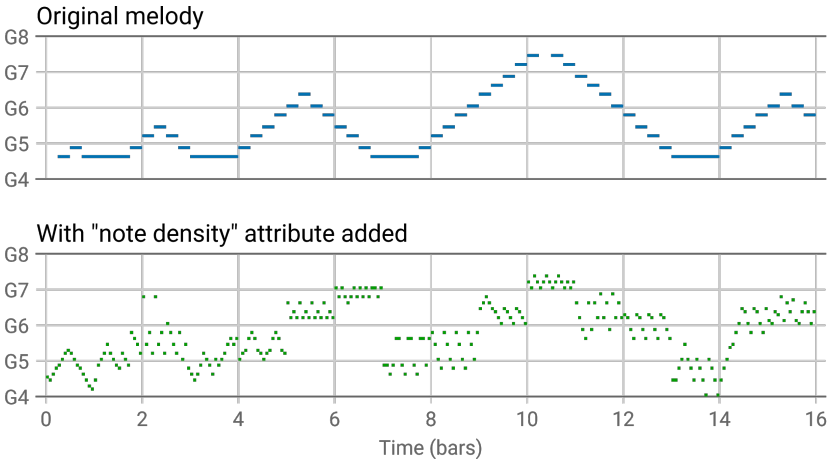
A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music, ICML 2018.

Here is a diagram of our approach. As before, the encoder is an RNN which processes a sequence. We then use the RNN's final hidden state to parametrize a distribution over latent codes. Then, after sampling a latent code, instead of decoding it directly, we first pass it to an intermediate recurrent neural network which outputs a sequence of latent codes, one for each bar or measure in the music being modeled. The decoder then decodes each bar independently - it is **not allowed to propagate any information between bars**, so it relies on the latent codes in order to model any longer-term structure. Our hope is that the latent code will encapsulate the high-level properties of a piece of music, like its rhythmic and harmonic structure. In doing so, we are effectively forcing the model to use the latent code by leveraging the fact that music is naturally decomposed into individual bars. As a result of this we were able to model sequences with hundreds of timesteps, which was longer than anyone had previously shown with this kind of model.



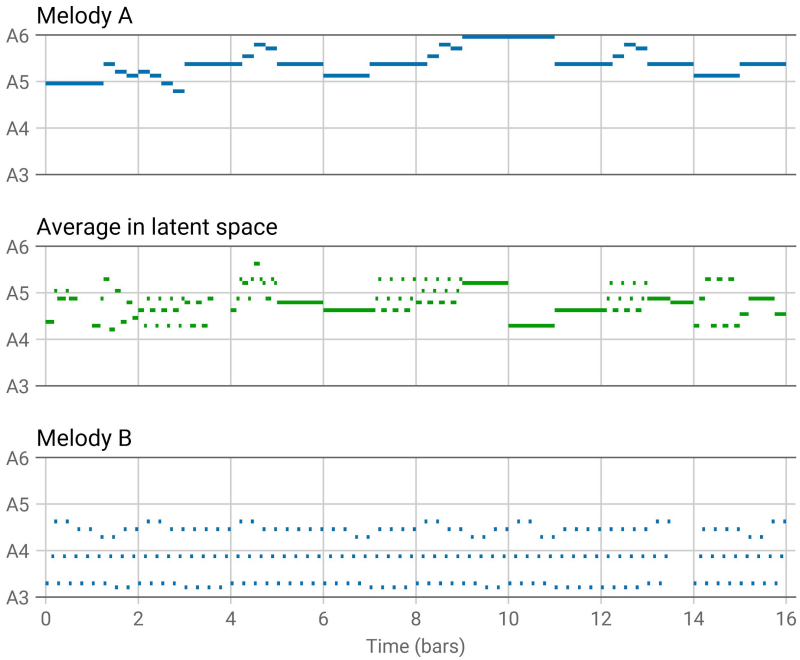
A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music, ICML 2018.

As a simple test of this model, we can see how well it reconstructs an example input melody. To do this, we fed in the melody on the top into our model to obtain a latent code, and then decoded the result. **The plots here are called piano rolls**; they reflect which notes are being played on the y axis at a given point in time on the x axis. On the top we see the original melody, and this is what it sounds like. I'll only play the first few bars, just for brevity. ... On the bottom is the reconstruction, which you can see is not perfect. Let's listen to that. ... As you can hear, the model got the timing, key, and overall contour of the melody right but did not get the details correct. In some ways, this is ideal - we want our latent code to only capture the high-level characteristics, which is all the decoder has access to.



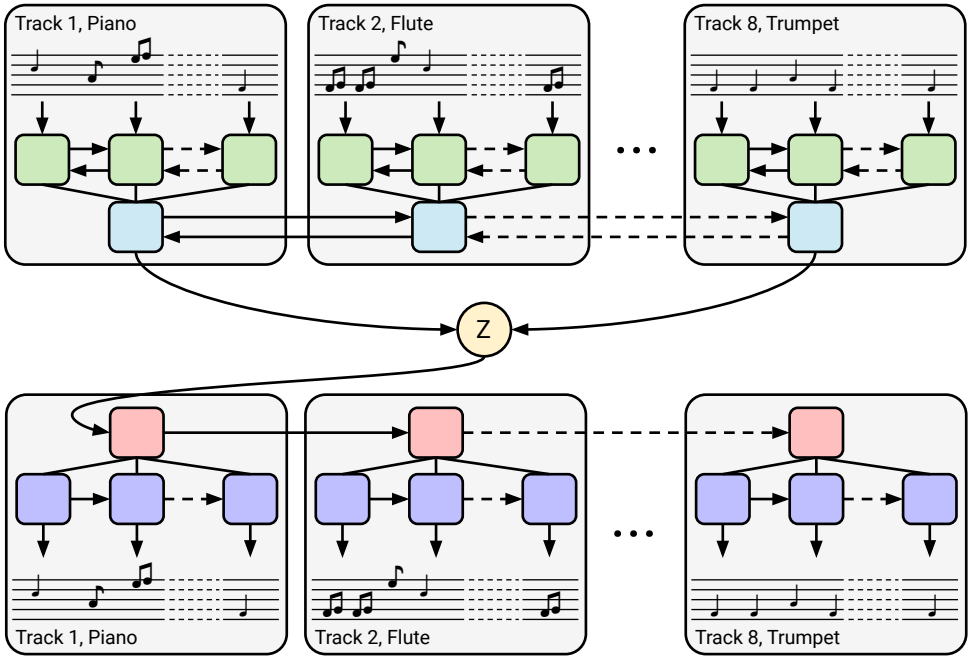
A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music, ICML 2018.

With an effective latent space, one useful thing we can do is called attribute vector arithmetic. To do this, we collect a bunch of examples which have some particular attribute, obtain their latent codes, and then average them all together to obtain an attribute vector. To apply this attribute to a given datapoint, we encode the datapoint and add the attribute to its latent code and then decode the result. Here we are showing the effect of a "note density" attribute vector which was computed by averaging together the latent codes for melodies with many frequent notes. So, if we add this attribute vector to the encoding from the previous example, here's the result ... Notice that it did not simply repeat notes - it created a musically plausible arpeggiation of the original melody.



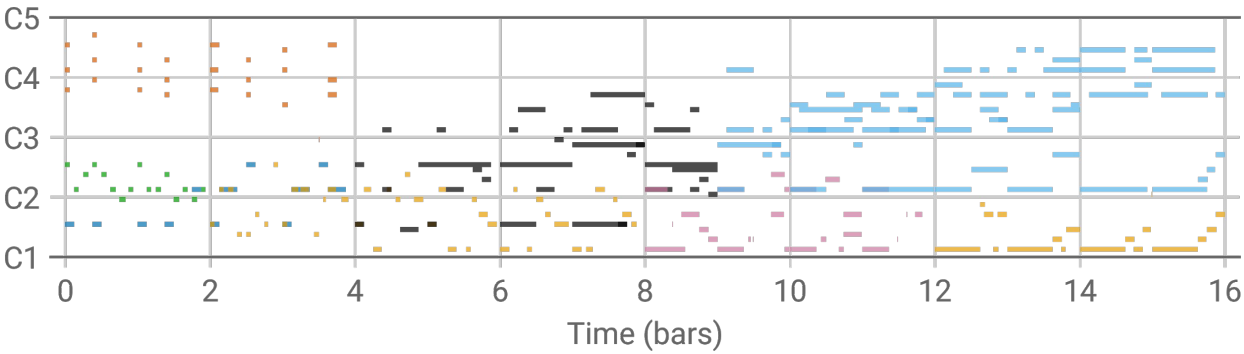
A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music, ICML 2018.

Another possibility is to interpolate between datapoints by mixing their latent codes. We can do this by taking two inputs, encoding them to produce latent codes, computing a convex combination of the latent codes, and then decoding the result. On the top we have one original melody, which sounds like this .... and on the bottom we have another original melody, which sounds like this... The first melody is in a high register with slow notes; the second is in a low register, in a different key, with more frequent notes. If we take the latent vectors for these two melodies, average them together, and decode the result, what we get is on the middle which sounds like this: ... You can hear that the model has managed to incorporate aspects of both melodies, despite them being quite different. Specifically, the model has found a composite key for the two melodies despite them being in quite different keys; the melody incorporates some arpeggiated sections as well as longer held notes; and the melody is in a middle register.



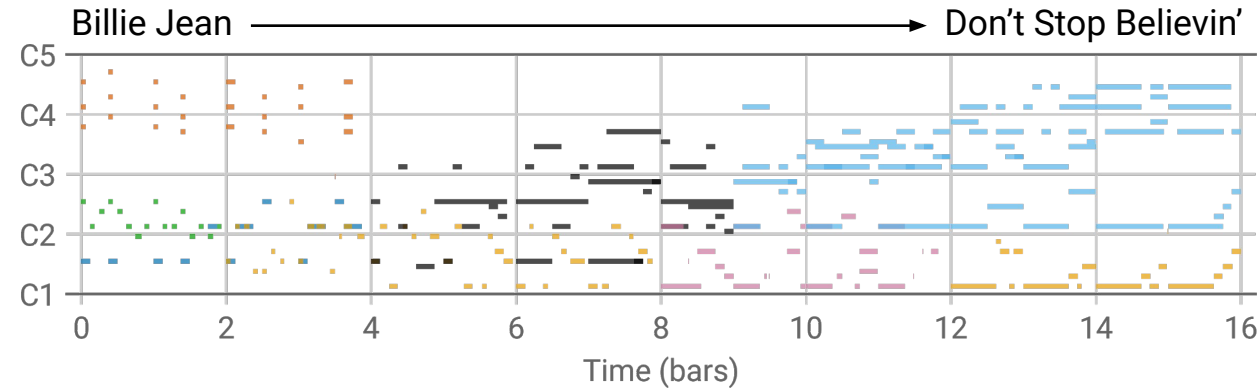
Learning a Latent Space of Multitrack Measures, NeurIPS Creativity Workshop 2018.

So far, I have been showing how we can more effectively model sequences with a variational autoencoder by breaking up the sequence into a natural temporal hierarchy. Another natural hierarchy for music is the hierarchy of instruments. We also explored modeling this more explicitly, which is diagrammed here - instead of producing individual latent codes for each measure in a long sequence, we produce separate latent codes for each track, or instrument, in a song. Again, these individual latent codes are computed based on a single global latent code, which makes the semantic manipulations we just explored possible in this setting too.



Learning a Latent Space of Multitrack Measures, NeurIPS Creativity Workshop 2018.

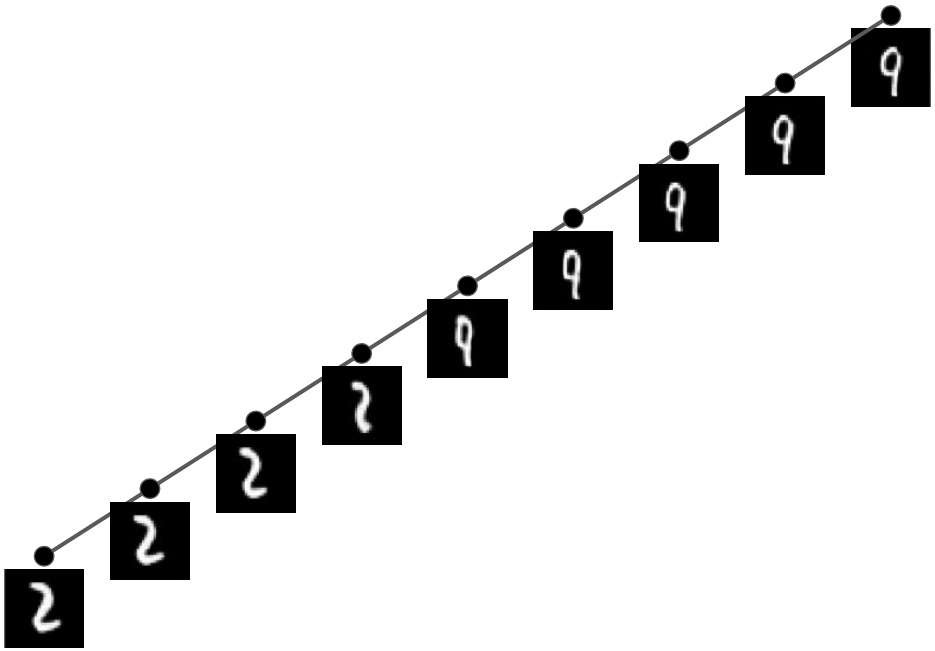
Now that we can model multiple instruments, we can actually model real music. To give an example, I'll play a gradual interpolation between the first measure of two popular songs. To produce this interpolation, we compute the latent codes for the two measures, and then compute a sequence of latent codes by mixing in more of one latent code and less and less of the other. Then, we decode each latent code along this interpolated sequence. Here's what that sounds like. ... Can anyone figure out the songs?



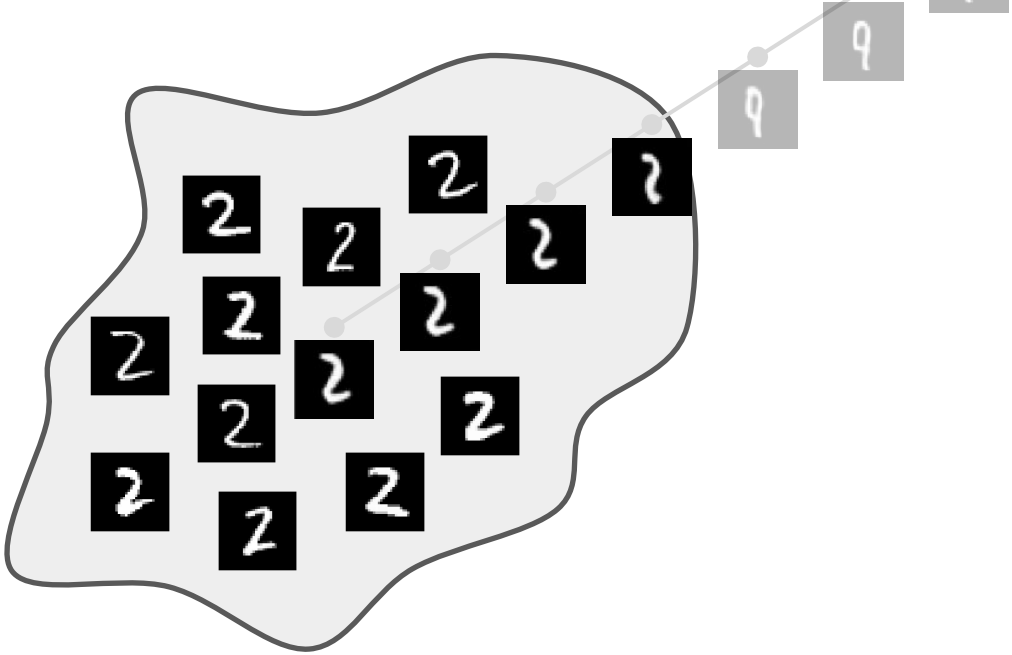
Learning a Latent Space of Multitrack Measures, NeurIPS Creativity Workshop 2018.

That’s right, it's billy jean and don’t stop believin. I think the model does a reasonable job of finding a plausible path between these two quite different measures of music -- it sounds smooth and each measure along the interpolation sounds realistic to me, apart from the point in the middle where the drummer kind of gives up and thrashes on notes. I would argue that this is a non-trivial operation even for a human to do at all, and as a result **this kind of model can open up new forms of creative control for musicians.**

In fact, MusicVAE has been integrated into Ableton Live, which is one of the most popular pieces of software for music creation. This allows musicians to perform creative manipulations of their music, like interpolating between two drum beats, directly in their normal music software.



Ok, so I have just shown a few examples of interpolating between datapoints. Let's explore this idea a bit more. What we're doing in these interpolations is taking two anchor points and getting their latent codes. Then, we are moving across the line between their two codes in latent space, so to speak, and decoding the result at various points along this line. In the example on the screen, I'm showing what interpolation looks like between two handwritten digits. You can see that the two slowly morphs into a 9. This is the behavior we want from an interpolation - we want the transition to be smooth, with realistic-looking points along the way.



What can interpolation tell us about the space of latent codes learned by the model? If the transition is semantically smooth, it means that points nearby in latent space decode to semantically similar points. Ideally, this could suggest that nearby points have some important high-level similarities. For example, we might hope that all of the 2 digits are close together in latent space. This could help for downstream tasks, like classifying between digits.

Does being able to interpolate imply a good representation for other downstream tasks?

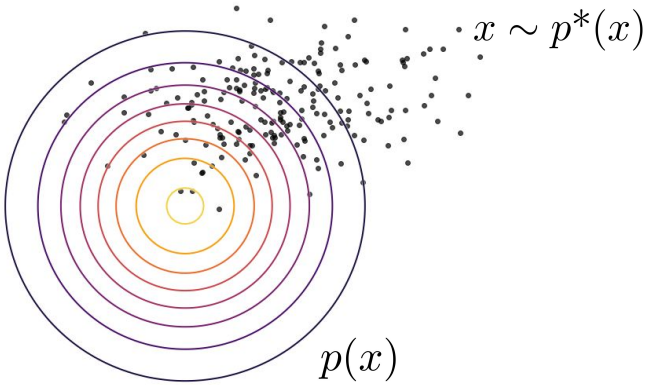
Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer, ICLR 2019.


David Berthelot


Aurko Roy

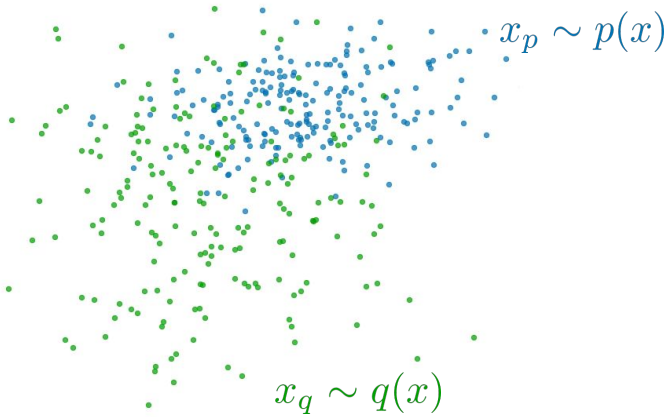

Ian Goodfellow

In this last portion of the talk, I'll describe my work which directly attacks this question. To do so, I developed a regularization strategy for autoencoders which encourages improved interpolation quality, and then tested whether applying the regularizer resulted in better performance on downstream tasks.



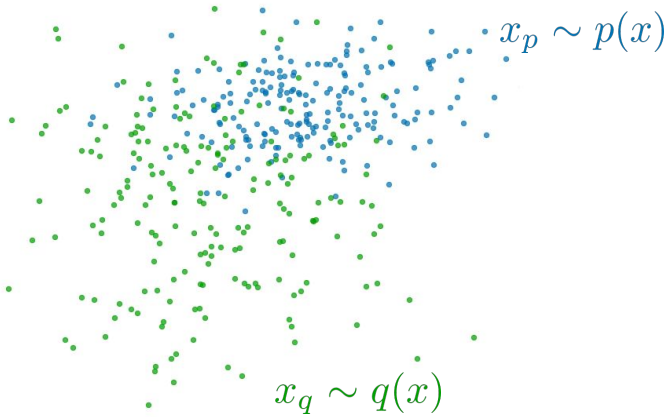
$$\min \mathrm{KL}(p^* || p) \rightarrow \max \mathbb{E}_{x \sim p^*}[\log p(x)]$$

In order to introduce this next bit of work, I need to explain the last major paradigm of deep generative modeling that I will discuss today. As I discussed at the beginning of the talk, our general approach is to minimize a divergence between our model p(x) the true distribution p star (x), using only samples from the true distribution.



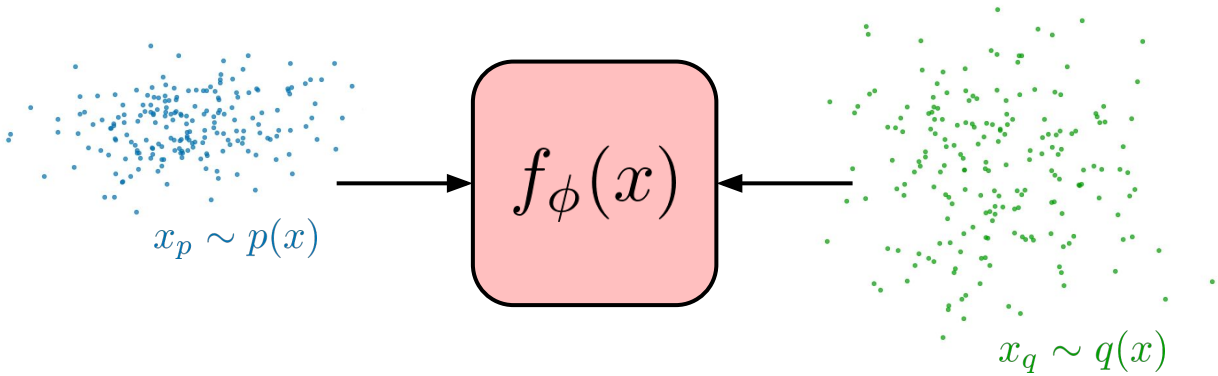
$$\min \mathrm{KL}(p \| q) \rightarrow \max \mathbb{E}_{x_p \sim p}[\log q(x_p)] ?$$

Now, let's consider the case where we neither have access to the true distribution nor our model - we can only draw samples from either; we can't compute the probability of a given datapoint under our model or the true distribution. Remember that before, we used the KL divergence between our model and the true distribution to fit the model, which resulted in maximum likelihood estimation. So, in the most generic case, we might similarly hope to compute a divergence between two generic distributions p and q using only samples from each and do something like maximum likelihood.



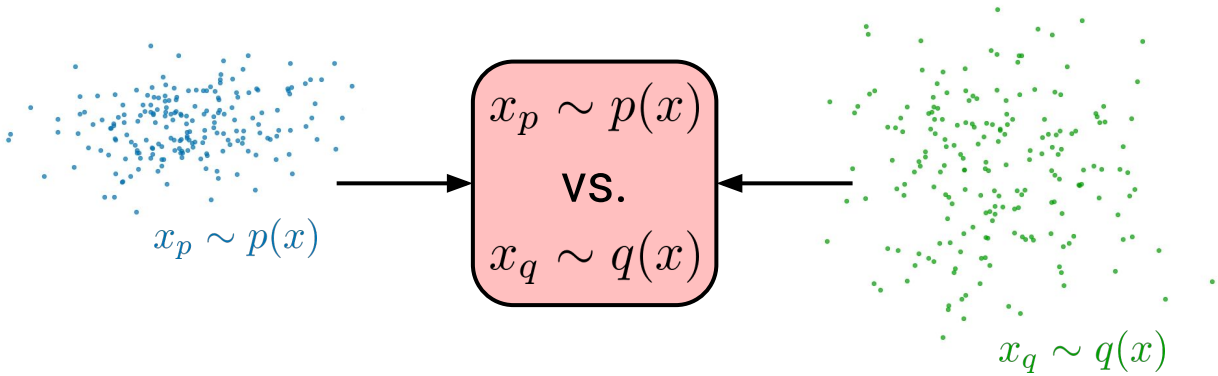
$$\min \mathrm{KL}(p \| q) \rightarrow \max \mathbb{E}_{x_p \sim p}[\log \cancel{q(x_p)}]$$

Unfortunately, we can't do this using the KL divergence and maximum likelihood as we have been, because we can't compute the probability of a sample under our model.



$$\min_{\phi} \max \mathbb{E}_{x_p \sim p, x_q \sim q} V(f_\phi(x_p), f_\phi(x_q))$$

A powerful method for addressing this issue is adversarial training. The simplest way to think of adversarial training is that it learns a function f_\phi which forms a sort of implicit divergence between the two distributions. We call f_\phi the “critic”.



$$\min \max_{\phi} \mathbb{E}_{x_p \sim p, x_q \sim q} V(f_\phi(x_p), f_\phi(x_q))$$

In practice, the critic f_\phi is typically optimized so that it can easily distinguish between samples from p and samples from q. Then, we can modify one of our distributions to match the other by modifying its parameters to make f_\phi unable to distinguish between the two distributions. If f_\phi is sufficiently powerful and can't tell them apart, then they must be equivalent.

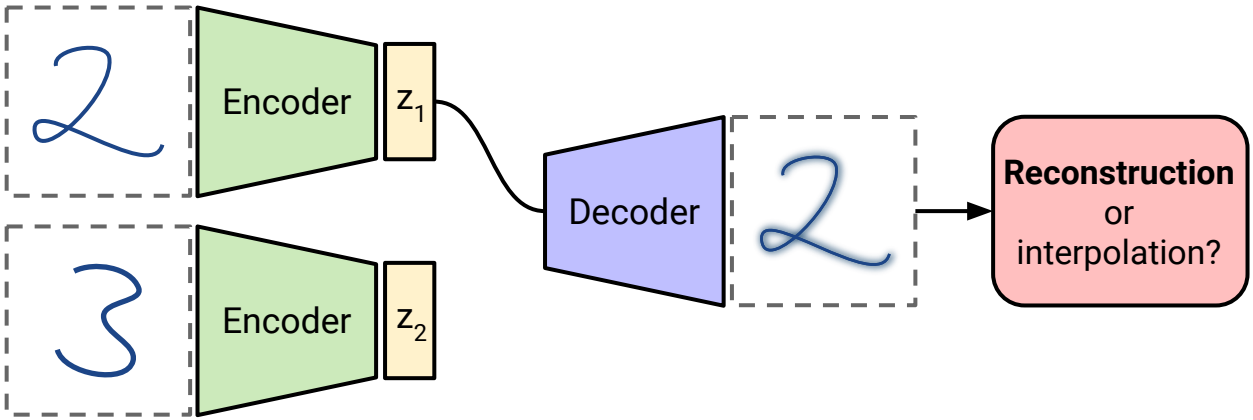
$$x_p \sim p(x)$$

vs.

$$x_q \sim q(x)$$

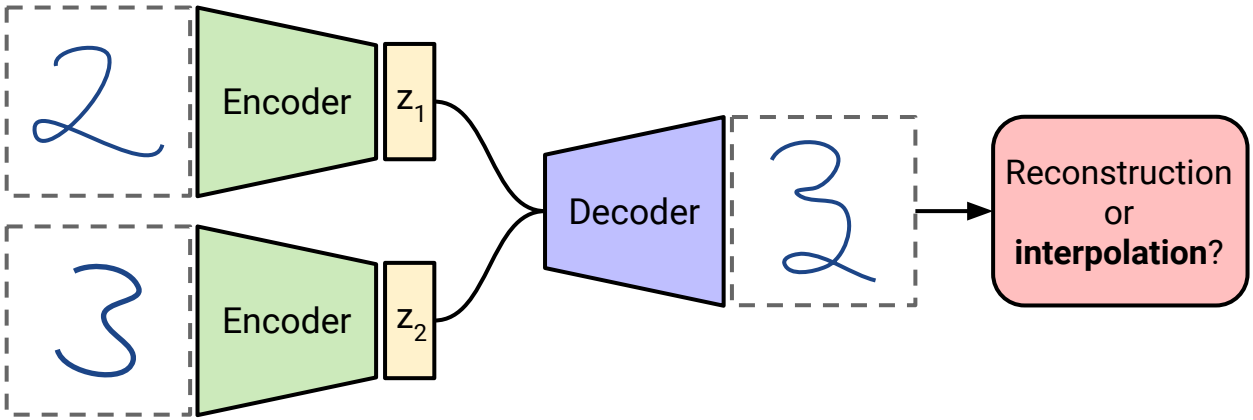
$$\min_{\phi} \max \mathbb{E}_{x_p \sim p, x_q \sim q} V(f_\phi(x_p), f_\phi(x_q))$$

This is the framework behind Generative Adversarial Networks or GANs, which generated the impressive pictures from early in this talk. In a GAN, we train a critic to distinguish between the distributions of real and fake images - that is, to distinguish between samples from our model and samples from the true distribution. Fake images are generated by a neural network, which is trained to fool the critic into thinking that fake images are real. Note that we are still minimizing a quantity shown in the red box at the bottom - this can be thought of loosely as a divergence, and is commonly called an adversarial divergence. This framework can provide a powerful method for learning generative models because both the divergence and our model are realized as arbitrary neural networks.



Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer, ICLR 2019.

So, back to the idea of evaluating whether good interpolations imply good representations. To test this idea, we won't be using GANs; instead, we will use the framework of adversarial learning to create a regularizer which encourages an autoencoder to produce good interpolations. When we encode an input and decode the result without modifying the latent code, the critic will be trained to recognize it as a reconstruction.

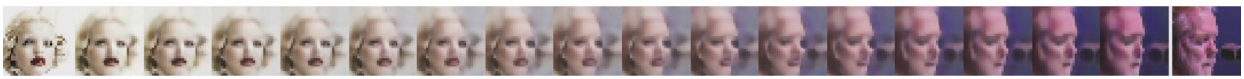


Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer, ICLR 2019.

Then, when we decode a mixture between two latent codes, the critic will be trained to classify it as an interpolation. If the autoencoder successfully fools the critic into thinking that interpolated points are reconstructions, it suggests that the interpolations might be more realistic. As a result, this process can be seen as having the simple function of making interpolations look more realistic. We call this approach ACAI, which stands for "adversarial constraint for autoencoder interpolation", and is also the name of a small tasty berry from Brazil.

Standard autoencoder

Standard autoencoder



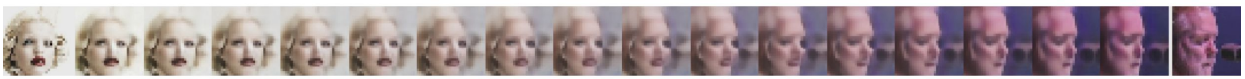
Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer, ICLR 2019.

So, let's test this idea. Here are interpolations between colored digits on the top and between people's faces on the bottom, as produced by a simple autoencoder with no regularization to change the interpolation behavior. You can see that around the middle the autoencoder ends up producing some unrealistic interpolation - on the top, it just sort of fades out one digit in pixel space; on the bottom the right half of the face sort of disappears.

Standard autoencoder

With ACAI

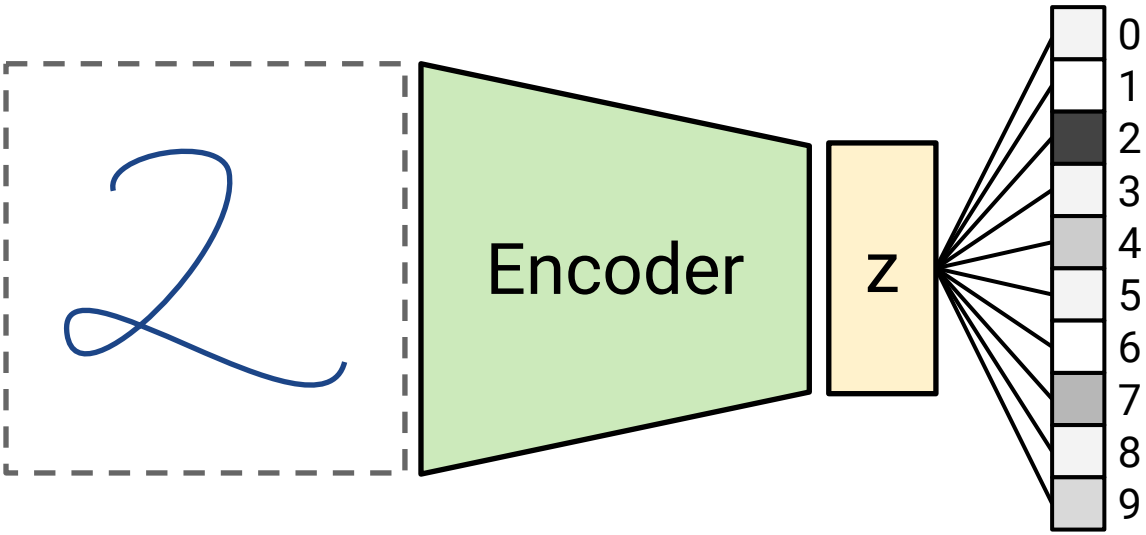
Standard autoencoder



With ACAI

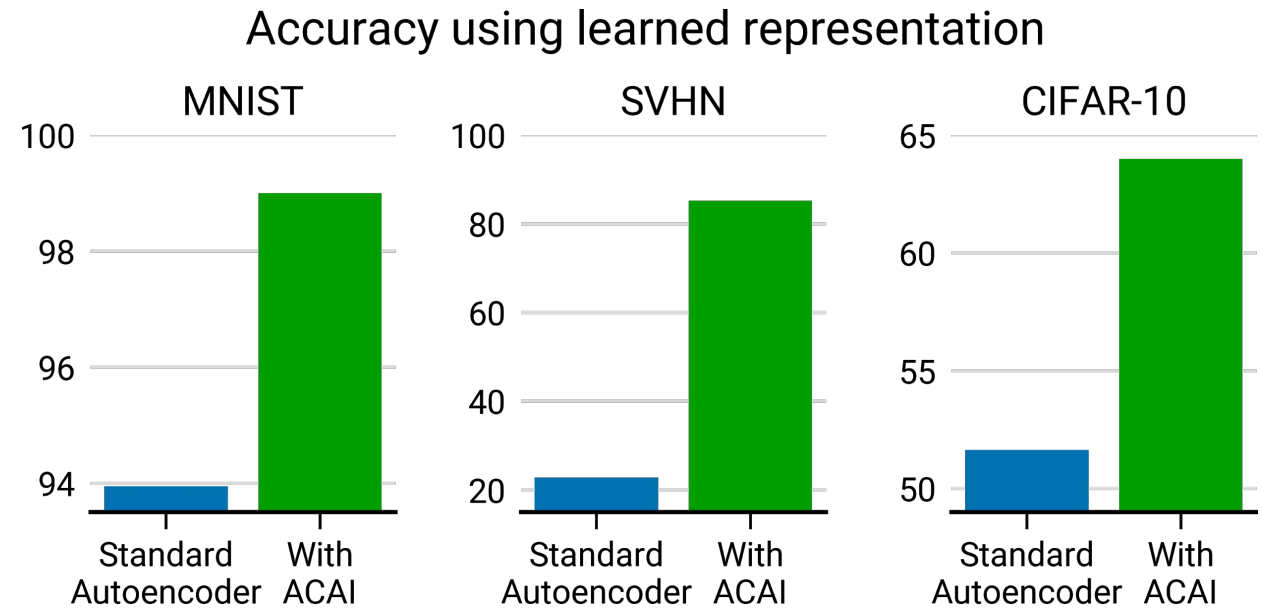
Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer, ICLR 2019.

When we add ACAI, you can see that the interpolated points become realistic. There is no pixel-fading on the top, and on the bottom we see something closer to rotation. So, we are successfully able to make interpolations more realistic with ACAI.



Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer, ICLR 2019.

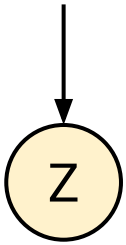
So, now let's try to answer the question as to whether better interpolations produces better representations. To do so, we'll perform a simple test: If we train a simple, single-layer classifier using the latent representations as input, does the classifier perform better or worse with ACAI? Since the classifier is simple, it can only perform well if the learned representation has successfully clustered together different classes.



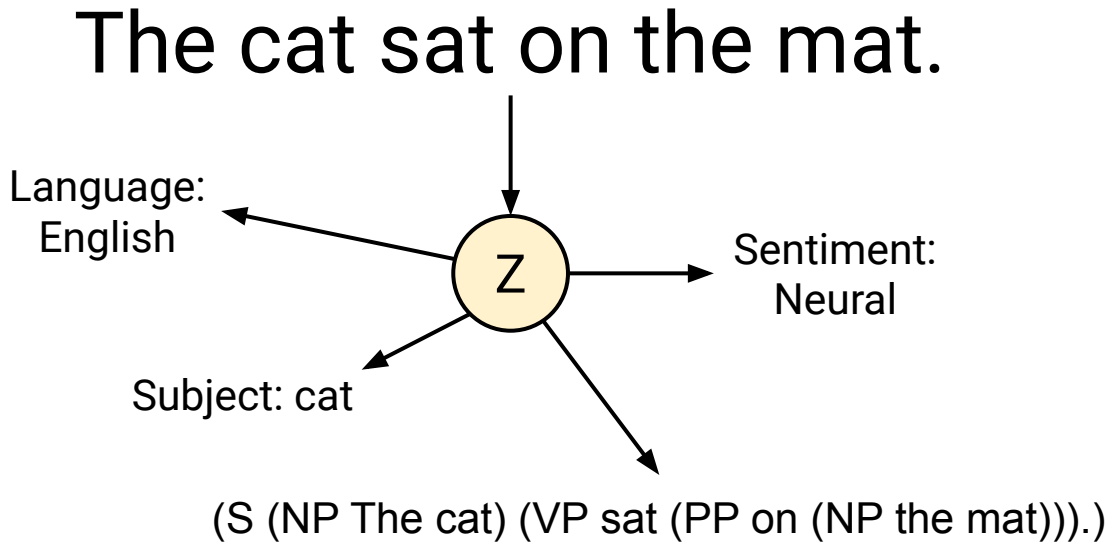
Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer, ICLR 2019.

When we do this, we indeed find that adding ACAI improves performance significantly across a variety of common benchmark datasets. In the most extreme example, the classification performance increases from around 20% to 80%. These results suggest that indeed there is a link between high-quality interpolations and useful representations.

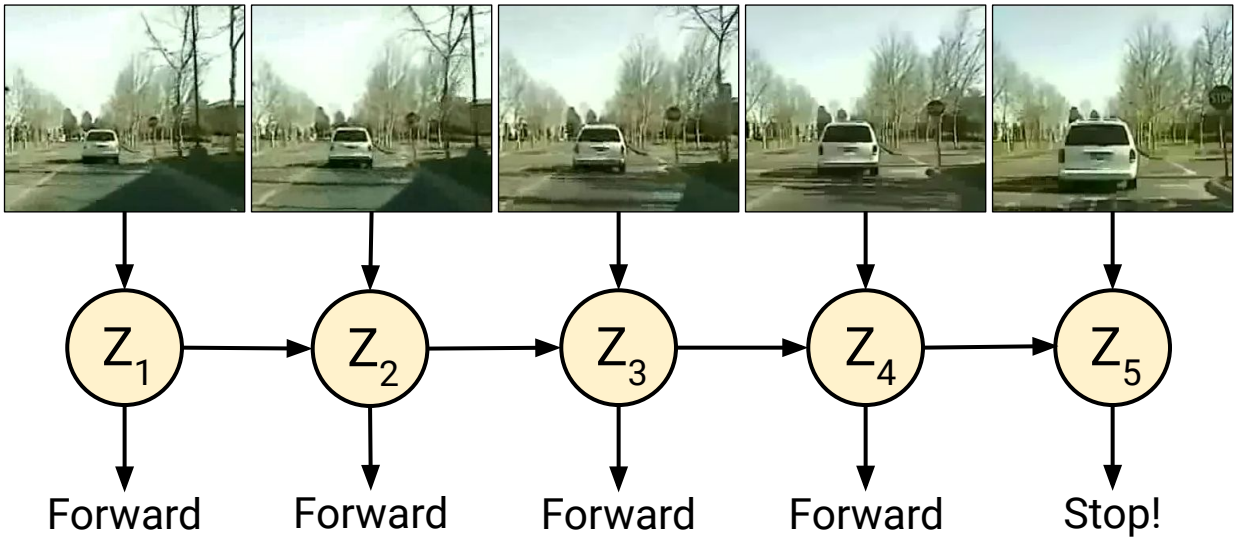
The cat sat on the mat.



So, to recap, we have mainly been talking about generative models which learn to map their input to some latent space, which we call z. This mapping is typically learned without supervision. Our hope is that z captures the most useful information about its input.

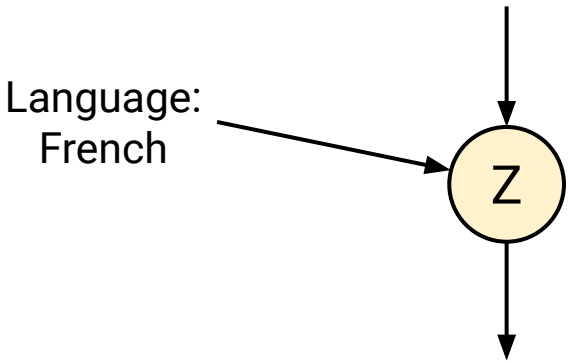


One way to test whether the latent code z is useful is to see if we can decode useful high-level information from it. For example, if we map this sentence to z, we might be able to easily find that its an english sentence about a cat with neutral sentiment. This is what I mean when I say we are capturing the high-level information. We also might hope to be able to easily perform downstream tasks like grammatically parsing the sentence.



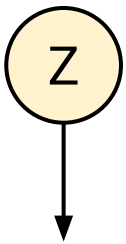
Being able to learn this kind of latent space in sequential prediction models has applications outside those that I mentioned today. For example, if we train a model which not only can successfully predict what comes next but also provide a useful representation, then inferences could more easily be made about how to interact with the world. This has application in model-based reinforcement learning and other applications like self-driving cars.

The cat sat on the mat.



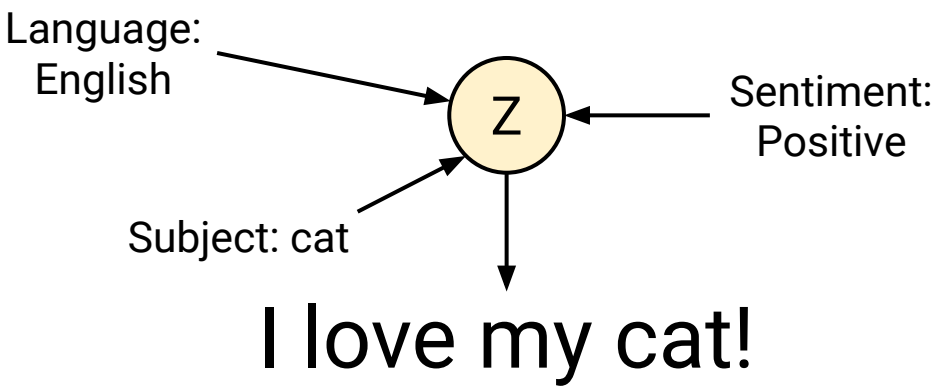
Le chat s’est assis sur le tapis.

In this talk, we have been focused not only on representation learning but more generally on generative models. Broadly speaking, what we want out of generative models is the ability to generate new data. One of the powerful implications of this is that we can generate new data conditioned on some inputs. For example, we can condition on an input English sentence, ask the model to generate a sentence in French, and decode the result, in hopes of producing a French translation.

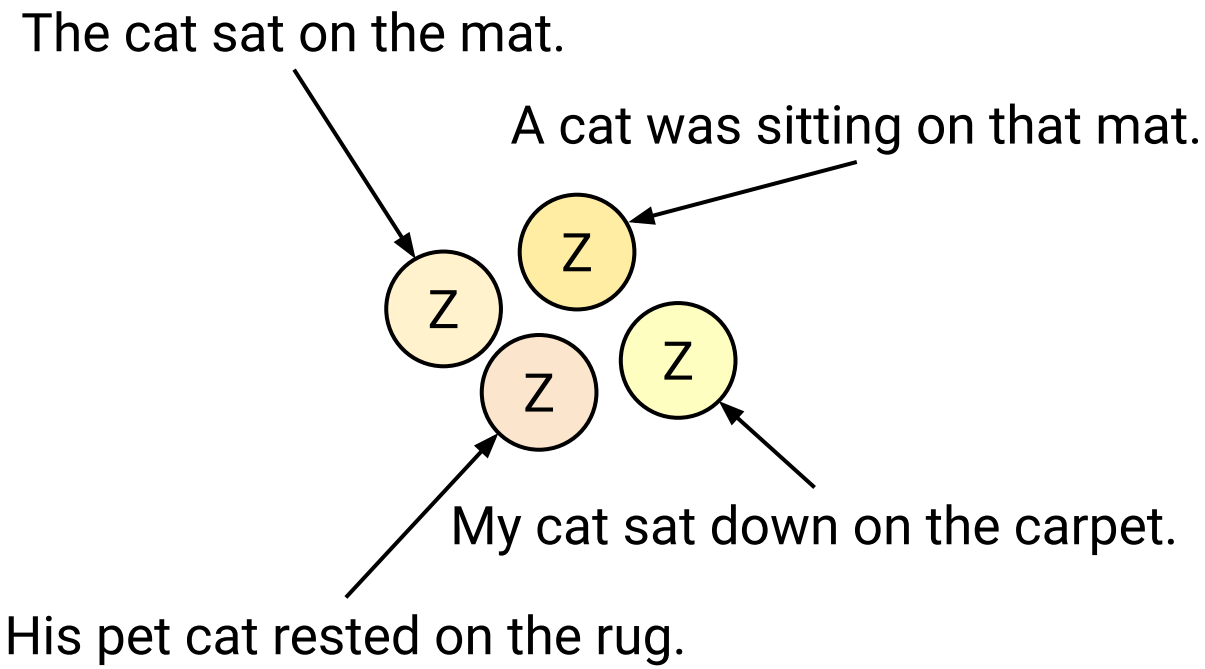


A quick brown fox jumps over the lazy dog.

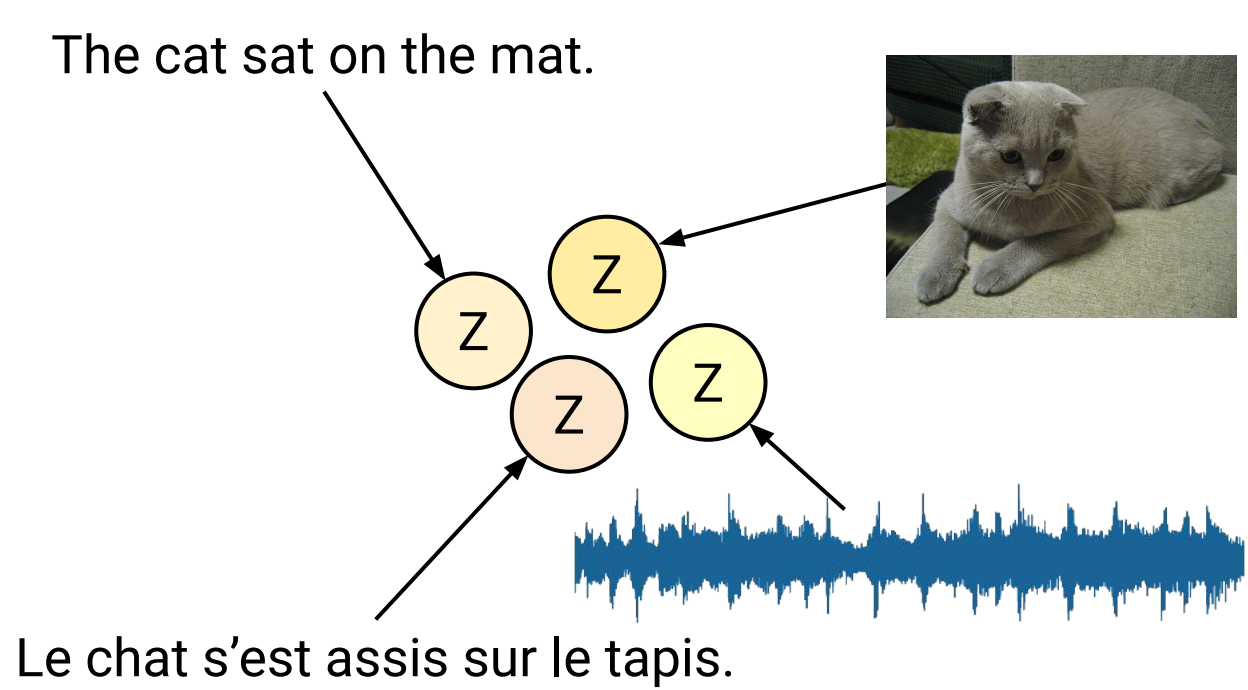
Beyond that, we also often use generative models for generating new data, for example generating a plausible English sentence unconditionally.



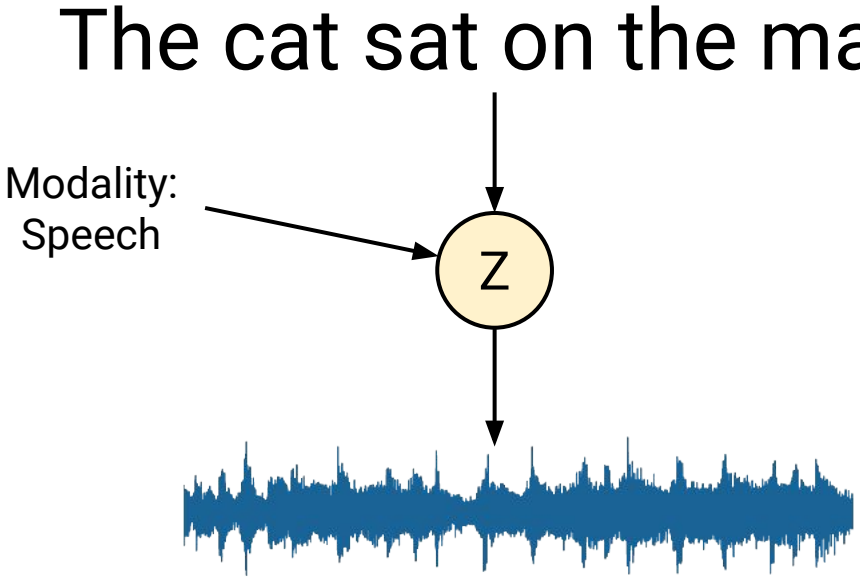
But I would argue generative models are most useful when we have some intuitive control over the data-generation process. Ideally, our model can discover the important structure and attributes of a dataset and provide control over them, so that for example we can generate a sentence which is specifically in English, has positive sentiment, and concerns a cat.



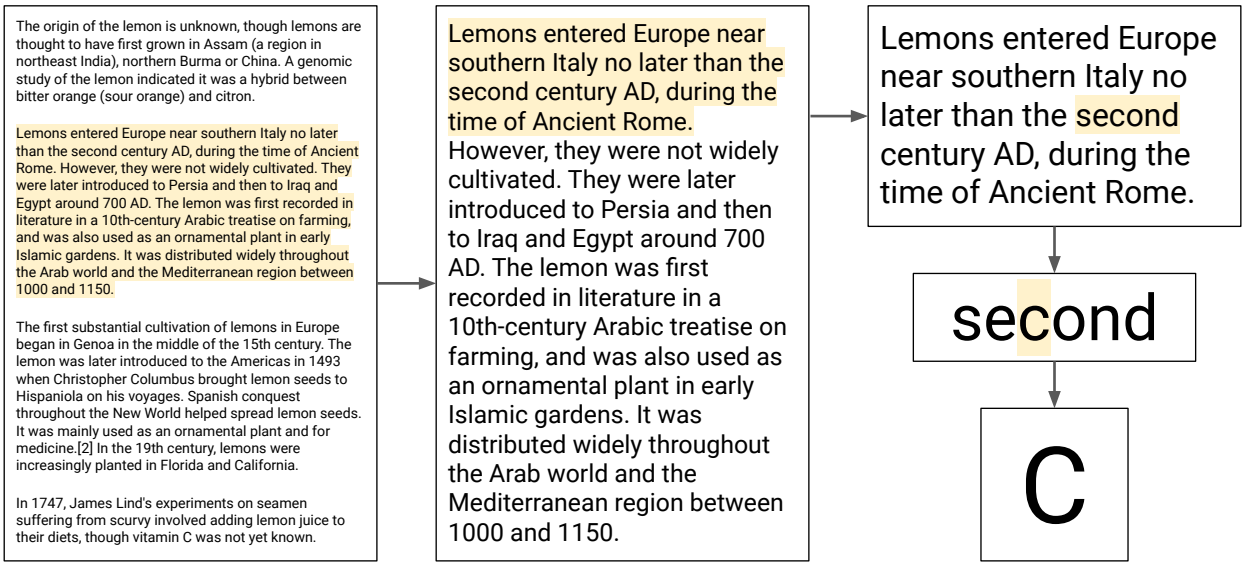
Similarly, we also want our latent representation to capture the high-level semantics of data. For example, if we compute the latent encoding for a variety of different sentences which all have the same meaning, we might hope their representations are close together or otherwise similar, and our work has shown some degree to which this is true.



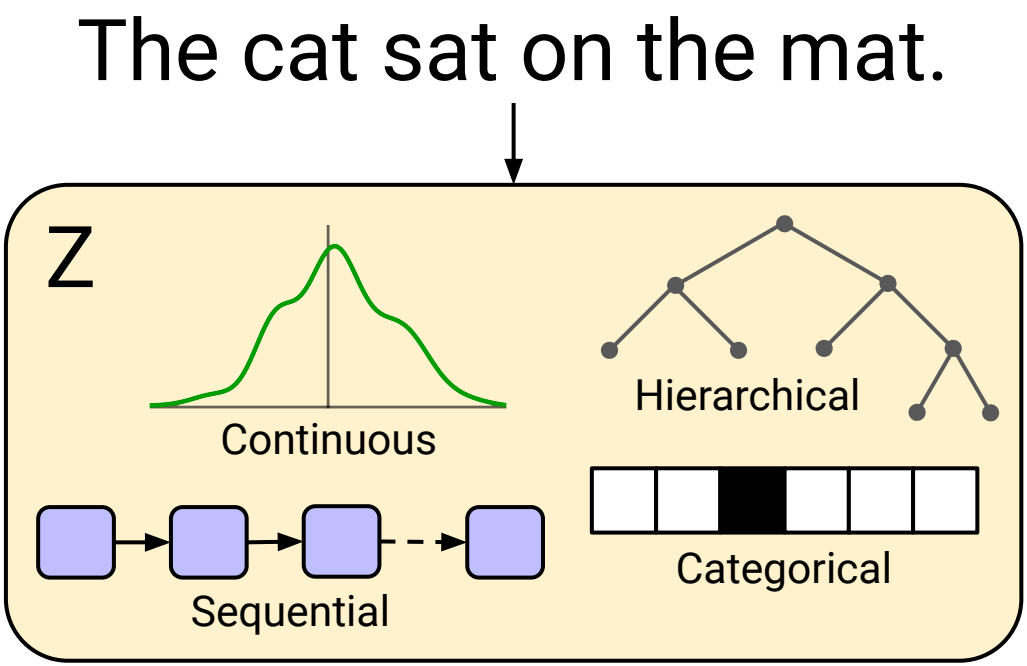
However, I think we should go beyond this. In the future, I'm interested in developing generative models which can can discover shared structure across modalities. Fundamentally, the image on the right and the sentence on the top have the same latent content - they are just expressed in different modalities, one as text and one as a natural image. Similarly, one of the most important factors in a spoken utterance are the words that were spoken. All of these different modalities have one thing in common: their latent content. If we can learn without supervision to discover these important latent factors, we should be able to find this shared structure across different modalities, too. By leveraging the large amount of unlabeled data in each modality, our model might gain better high-level knowledge which could create benefits across tasks.



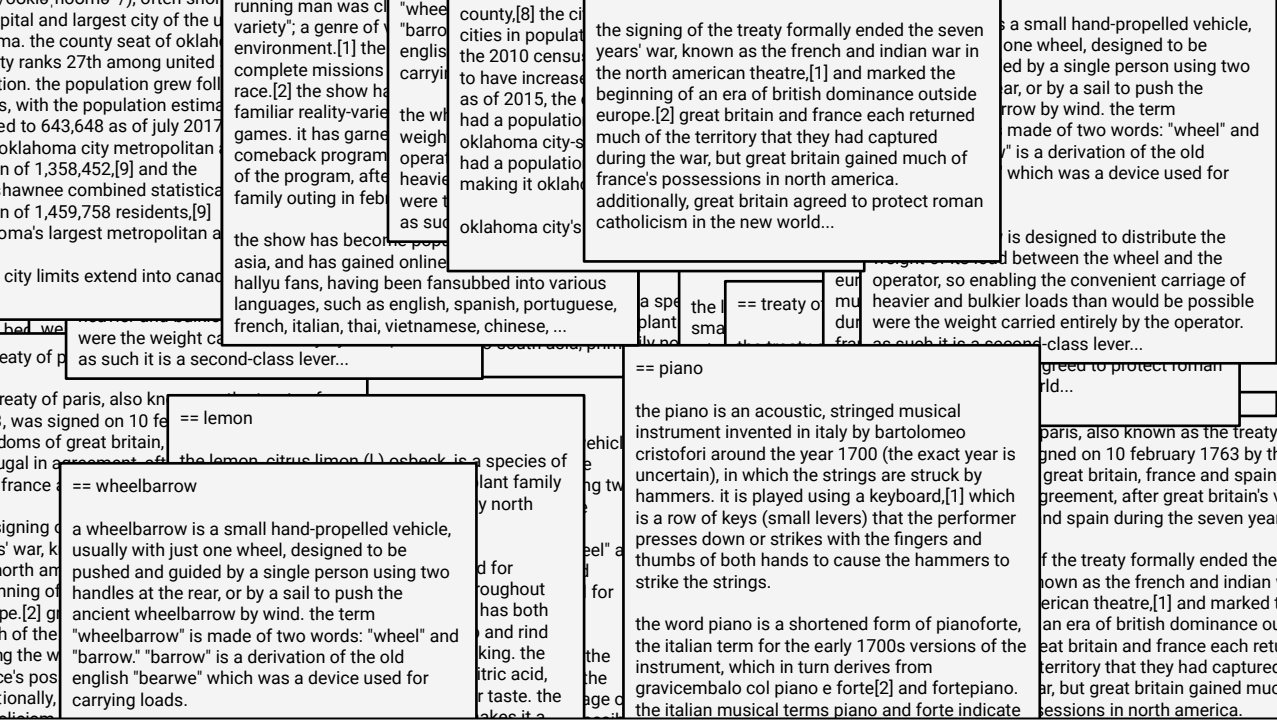
Why would we want to do this? Well, if we are able to create a generative model which represents data in different modalities with the same latent representation, and which can decouple the meaning of a piece of data from the way it was expressed, we might hope to be able to learn to perform useful transformations like converting text to speech by encoding a sentence and generating a speech utterance with the same meaning. While there has been great success in training this kind of model when provided with many transcribed spoken utterances, there is much more unpaired speech and text which we might hope to leverage when training a generative model.



I'll sketch a few problems I think need to be solved before we can start using generative models in these ways. First, as I showed in our MusicVAE work one important ingredient which determines the success of a generative model is the architecture of the model itself. One place where current models lack is in discovering the natural hierarchical structure in sequences on their own. For MusicVAE, we got around this by baking in the structure directly to the model, but for other domains the structure might not be known ahead of time. As an example, text documents are naturally structured into paragraphs, which are comprised of sentences, which are comprised of words, which are made up of characters. Our models should be able to naturally discover this kind of structure on its own. **That way, when we apply this class of approaches to new domains where we don't know the structure ahead of time, we can use these models to teach us new things about our data.**



Similarly, most of the latent representations I discussed today are simple and continuous. However, some of the structure we've been discussing is not fundamentally continuous. For example, the words underlying a spoken utterance are fundamentally a sequence of categorical latent variables. Similarly, the a grammatical parsing of a sentence is hierarchical. To realize the full potential of generative models, we need to be able to incorporate this prior structure into their latent space, too.



Finally, probably the biggest promise of generative models is their ability to leverage and discover structure in large unlabeled datasets. In machine learning, we often find gains simply from collecting more data. In many applications like modeling text, unlabeled data is almost limitless. It's natural to expect the availability of this data to make our learned models more powerful, and ultimately more useful in downstream tasks. As a result, I strongly believe we should investigate methods for scaling up our models and ingesting huge datasets so that we can make use of these largely untapped data sources.

References

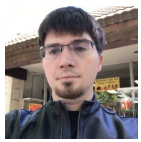
Online and Linear-Time Attention by Enforcing Monotonic Alignments, ICML 2017.
Monotonic Chunkwise Attention, ICLR 2018.
Monotonic Infinite Lookback Attention for Simultaneous Machine Translation, under review.
A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music, ICML 2018.
Learning a Latent Space of Multitrack Measures, NeurIPS Creativity Workshop 2018.
Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer, ICLR 2019.


Wolfgang Machery


Colin Cherry


David Berthelot

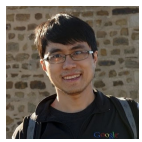

Chung-Cheng Chiu


Ian Goodfellow


Adam Roberts


Jesse Engel


Fjord Hawthorne


Ruoming Pang


Wei Li


Naveen Ari


Aurko Roy


Thang Luong


Peter J. Liu


Ron J. Weiss


Douglas Eck


Ian Simon


Semih Yavuz

I'd like to thank my coauthors, who are shown on the screen here, for working with me on the papers I presented today, which I have listed here at the top. And thanks to you all for your time and attention.