Advanced ML:
Unsupervised learning with autoregressive and latent variable models

https://github.com/janchorowski/JSALT2019_tutorials

Jan Chorowski
University of Wrocław

With slides from TMLSS 2018
(Jan Chorowski, Urlich Paquet)

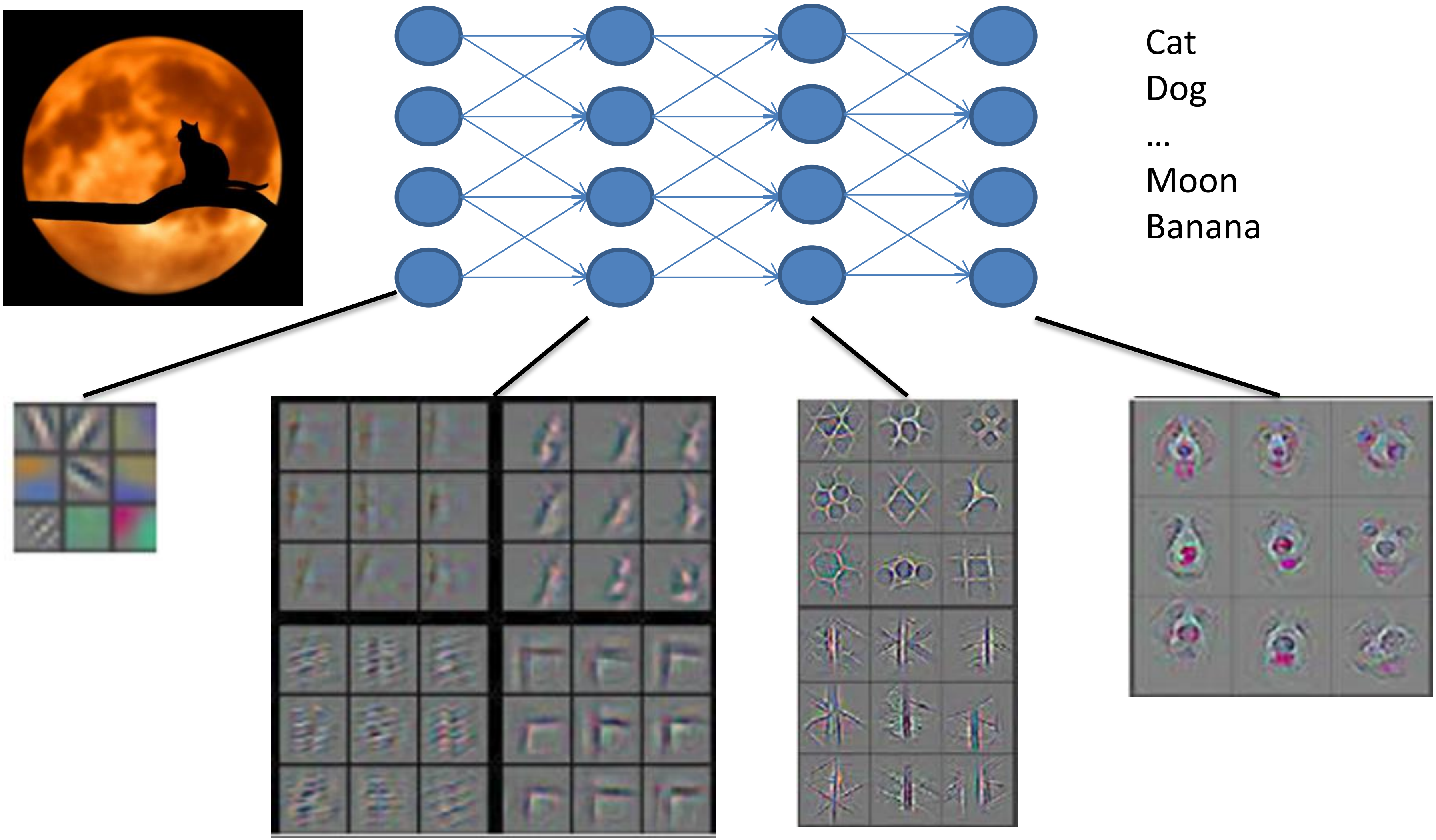
Outline

- Why unsupervised learning
- Autoregressive models
  - Intuitions
  - Dilated convolutions
    RNNs (Intermezzo: LSTM training dynamics)
    Transformers
  - Beyond sequences, summary
- Latent variable models
  - VAE
  - Normalizing flows
- Autoregressive + latent variable: why and how?

Outline

- **Why unsupervised learning**
- Autoregressive models
  - Intuitions
  - Dilated convolutions
    RNNs (Intermezzo: LSTM training dynamics)
    Transformers
  - Beyond sequences, summary
- Latent variable models
  - VAE
  - Normalizing flows
- Autoregressive + latent variable: why and how?

Deep Model = Hierarchy of Concepts



M. Zieler, “Visualizing and Understanding Convolutional Networks”

Deep Learning history: 1986

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

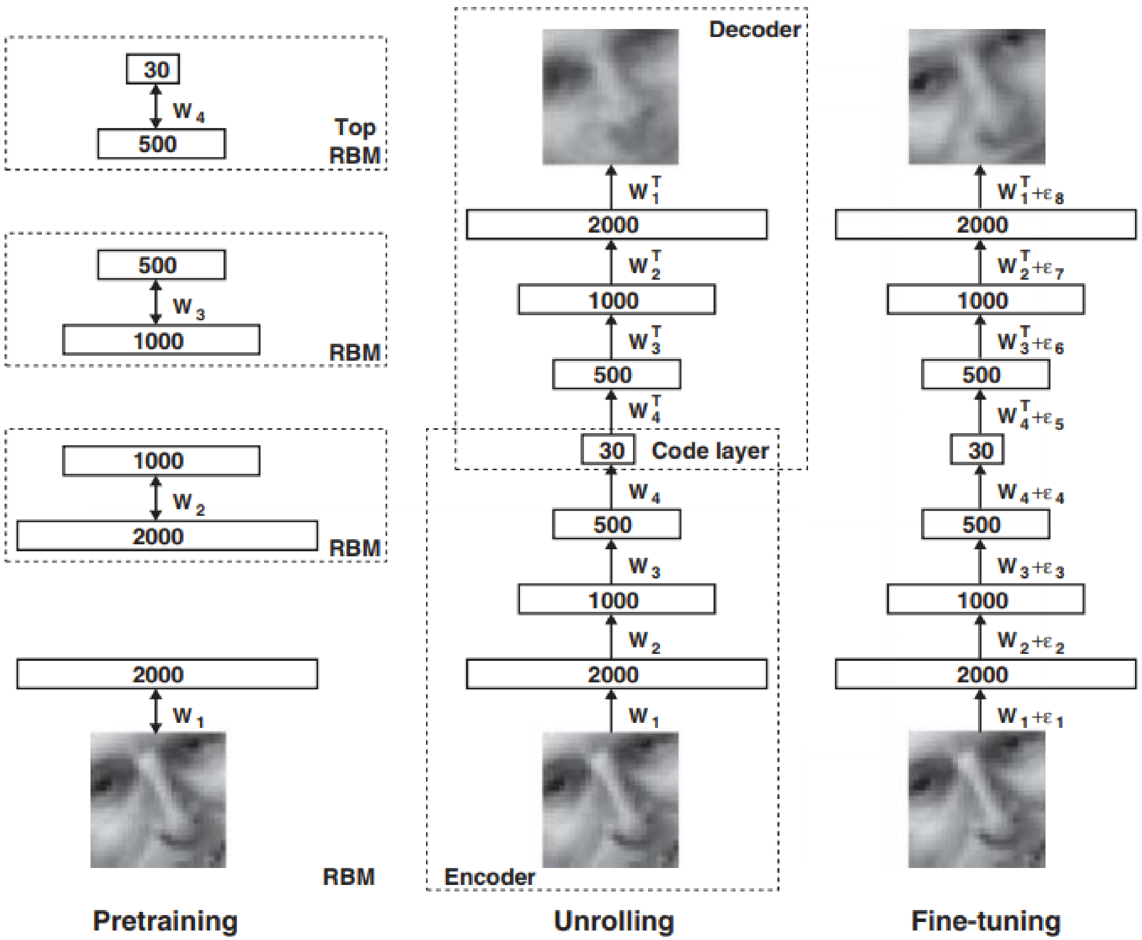
* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

'hidden' units which are not part of the input or output come to represent important features of the task domain

Deep Learning history: 2006

Stacked RBMs



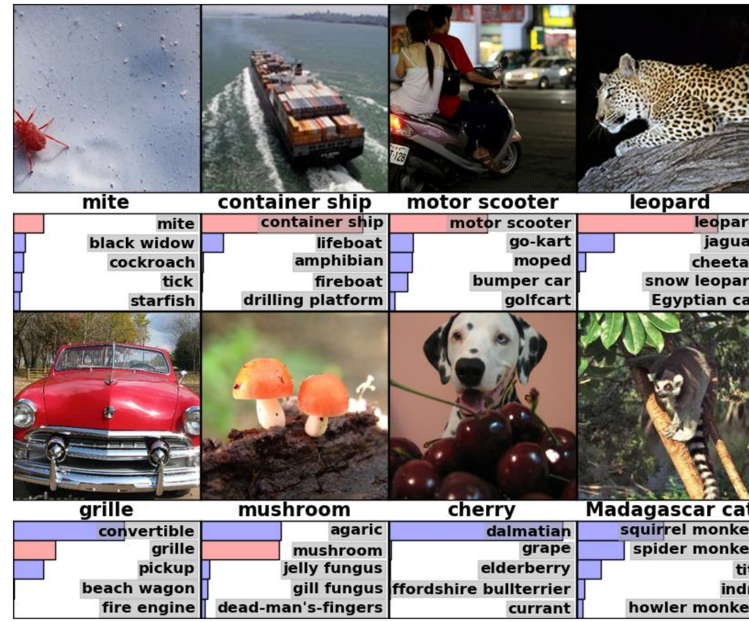
Hinton, Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks"

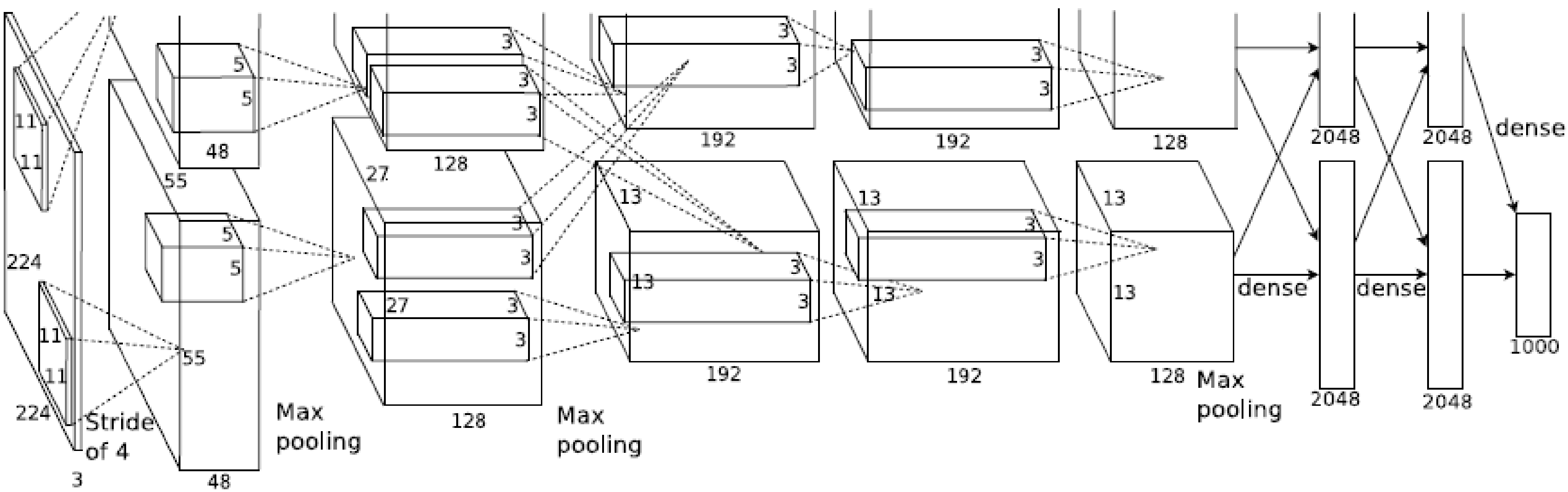
Deep Learning history: 2012

2012: Alexnet

SOTA on Imagenet

Fully supervised training





Deep Learning Recipe

1. Get a massive, labeled dataset $D = \{(x, y)\}$:
   - Comp. vision: Imagenet, 1M images
   - Machine translation: EuroParlamanet data, CommonCrawl, several million sent. pairs
   - Speech recognition: 1000h (LibriSpeech), 12000h (Google Voice Search)
   - Question answering: SQuAD, 150k questions with human answers
   - …
2. Train model to $\mathrm{maximize} \log p(y|x)$

Value of Labeled Data

- Labeled data is crucial for deep learning
- But labels carry little information:
  - Example:
    An ImageNet model has 30M weights, but
    ImageNet is about 1M images from 1000 classes
    Labels: 1M * 10bit = 10Mbits

    Raw data: (128 x 128 images): ca 500 Gbits!

Value of **Un**labeled Data

"The brain has about 10^{14} synapses and we only live for about 10^9 seconds. So we have a lot more parameters than data. This motivates the idea that we must do a lot of unsupervised learning since the perceptual input (including proprioception) is the only place we can get 10^5 dimensions of constraint per second."

Geoff Hinton

https://www.reddit.com/r/MachineLearning/comments/2lmo0l/ama_geoffrey_hinton/

Unsupervised learning recipe

1. Get a massive ~~labeled~~ dataset $D = \{x\}$
   Easy, unlabeled data is nearly free

2. Train model to…???

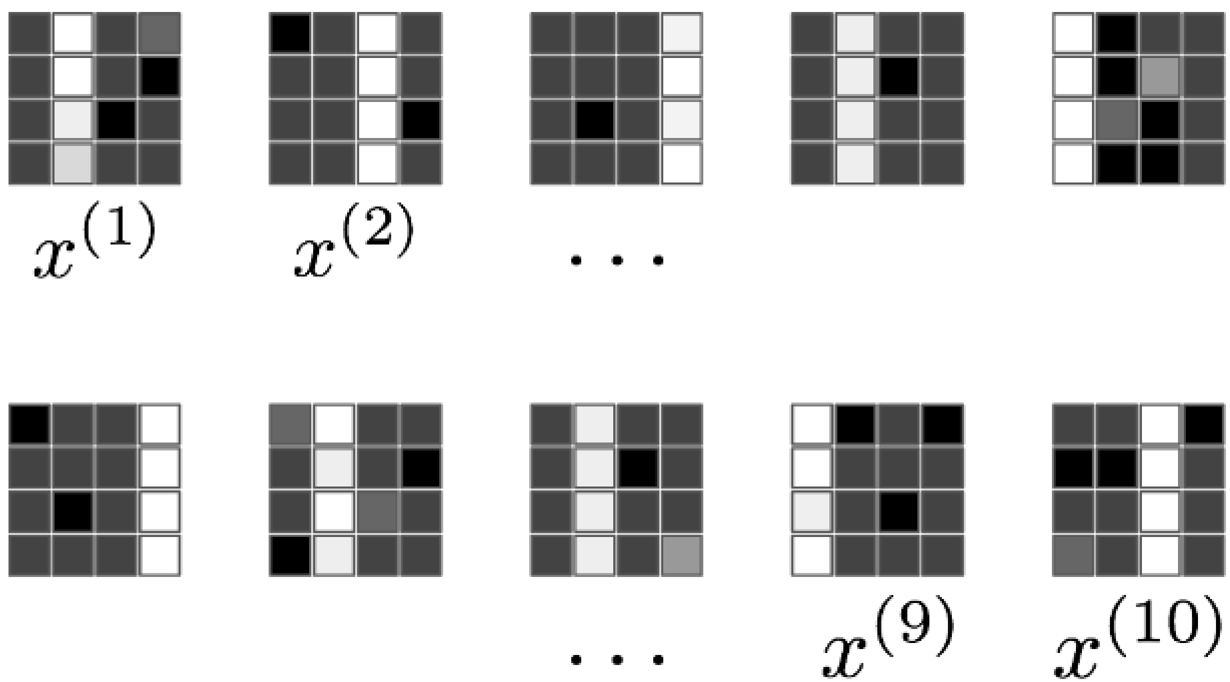
   What is the task?
   What is the loss function?

Unsupervised learning goals

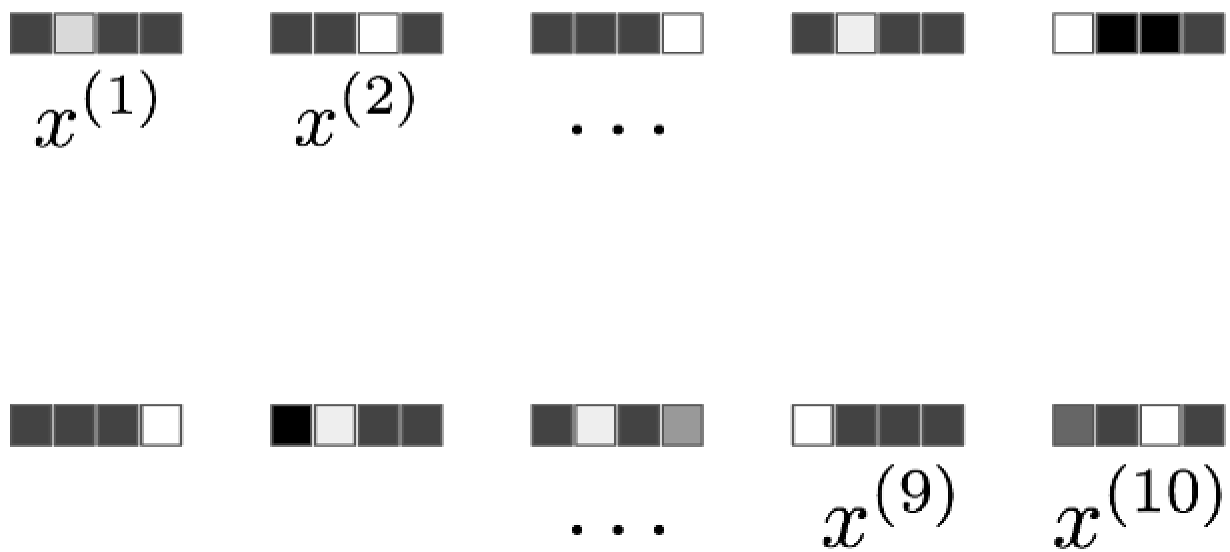
- Learn a data representation:
  - Extract features for downstream tasks
  - Describe the data (clusterings)
- Data generation / density estimation
  - What are my data
  - Outlier detection
  - Super-resolution
  - Artifact removal
  - Compression

2 minute exercise

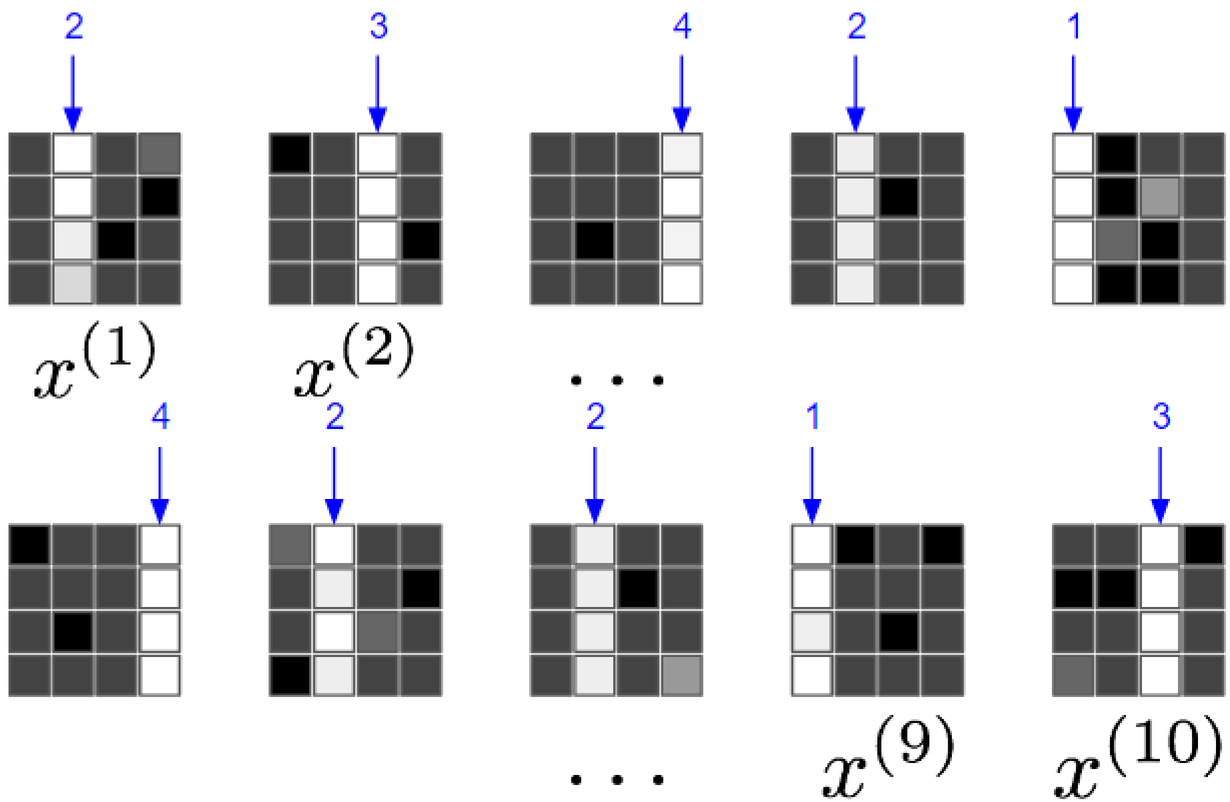
Talk to your friend next to you, and tell him or her everything you can about this data set:



The rows are correlated



Latent representation

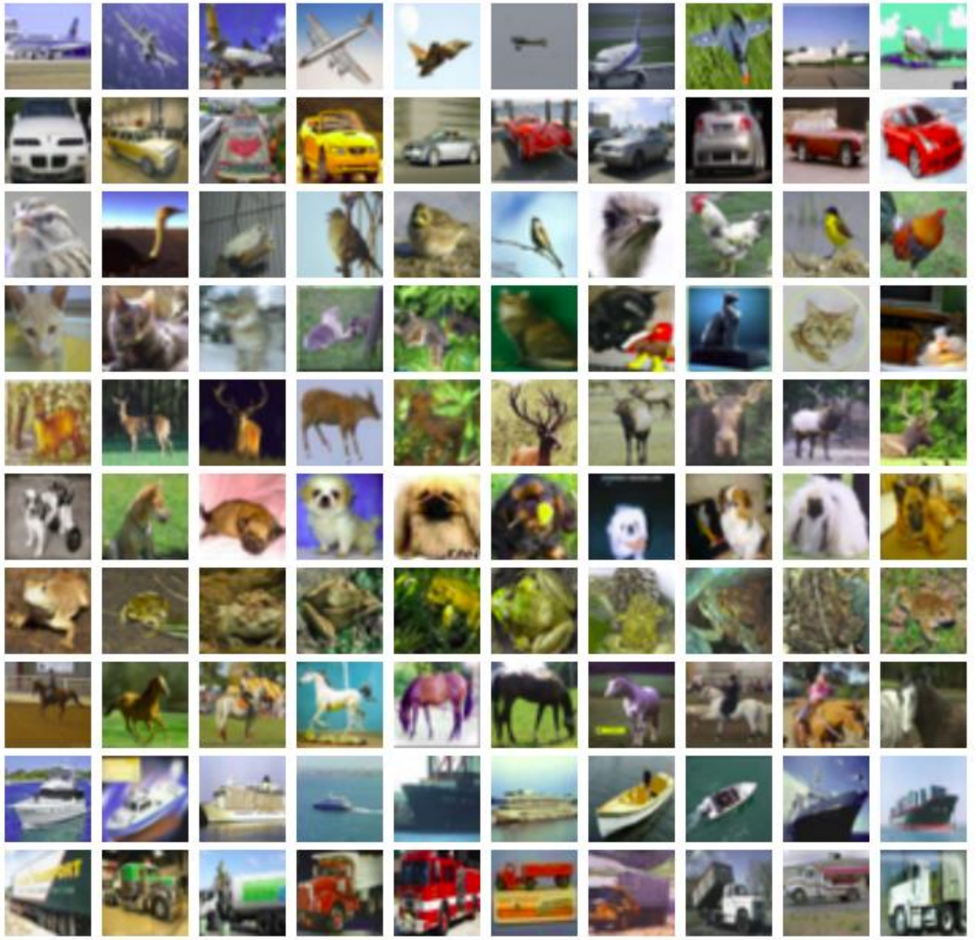


Outline

- Why unsupervised learning
- **Autoregressive models**
  - **Intuitions**
  - Dilated convolutions
    RNNs (Intermezzo: LSTM training dynamics)
    Transformers
  - Beyond sequences, summary
- Latent variable models
  - VAE
  - Normalizing flows
- Autoregressive + latent variable: why and how?

Learning high dimensional distributions is hard

- Assume we work with small (32x32) images
- Each data point is a real vector of size $32 \times 32 \times 3$
- Data occupies only a tiny fraction of $\mathbb{R}^{32\times 32\times 3}$
- Impossible to directly fit standard prob. distribution!



Divide and conquer

What is easier:

1. Write a paragraph at once?
2. Guess the next word?

Mary had a little
Mary had a little **dog**.
Mary had a little dog. **It**
Mary had a little dog. It **was**
Mary had a little dog. It was **cute**

Chain rule

Decompose probability of n-dimensional data points into product of n conditional probabilities

$$
\begin{aligned}
p(x) &= p(x_1, x_2, \dots, x_n) \\
&= p(x_1)p(x_2|x_1) \dots p(x_n|x_1, x_2, \dots, x_{n-1}) \\
&= \prod_t p(x_t|x_1, x_2, \dots, x_{t-1})
\end{aligned}
$$

Learning $p(x_t|x_1, \dots, x_{t-1})$

The good:

It looks like supervised learning!

The bad:

We need to handle variable-sized input.

Large n -> sparse data

Consider an n-gram LM:

$$p(x_t | x_{t-k+1}, \dots, x_{t-1}) = \frac{\# x_{t-k:t}}{\# x_{t-k:t-1}}$$

$$\begin{aligned} p(\text{a lion is chasing a llama}) = & p(\text{a}) \times p(\text{lion}|\text{a}) \times p(\text{is}|\text{a lion}) \\ & \times p(\text{chasing}|\text{lion is}) \times p(\text{a}|\text{is chasing}) \\ & \times \underbrace{p(\text{llama}|\text{chasing a})}_{=0} = 0 \end{aligned}$$

As $n \to \infty$ n-grams become unique:

- Probability of all but one continuation is 0
- Can’t model long contexts

Solution:

Use a representation in which “llama ≈ animal”

Learning $p(x_t|x_1, \dots, x_{t-1})$ #1

Assume distant past doesn't matter,

$$p(x_t|x_1, \dots, x_{t-1}) \approx p(x_t|x_{t-k+1}, \dots, x_{t-1})$$

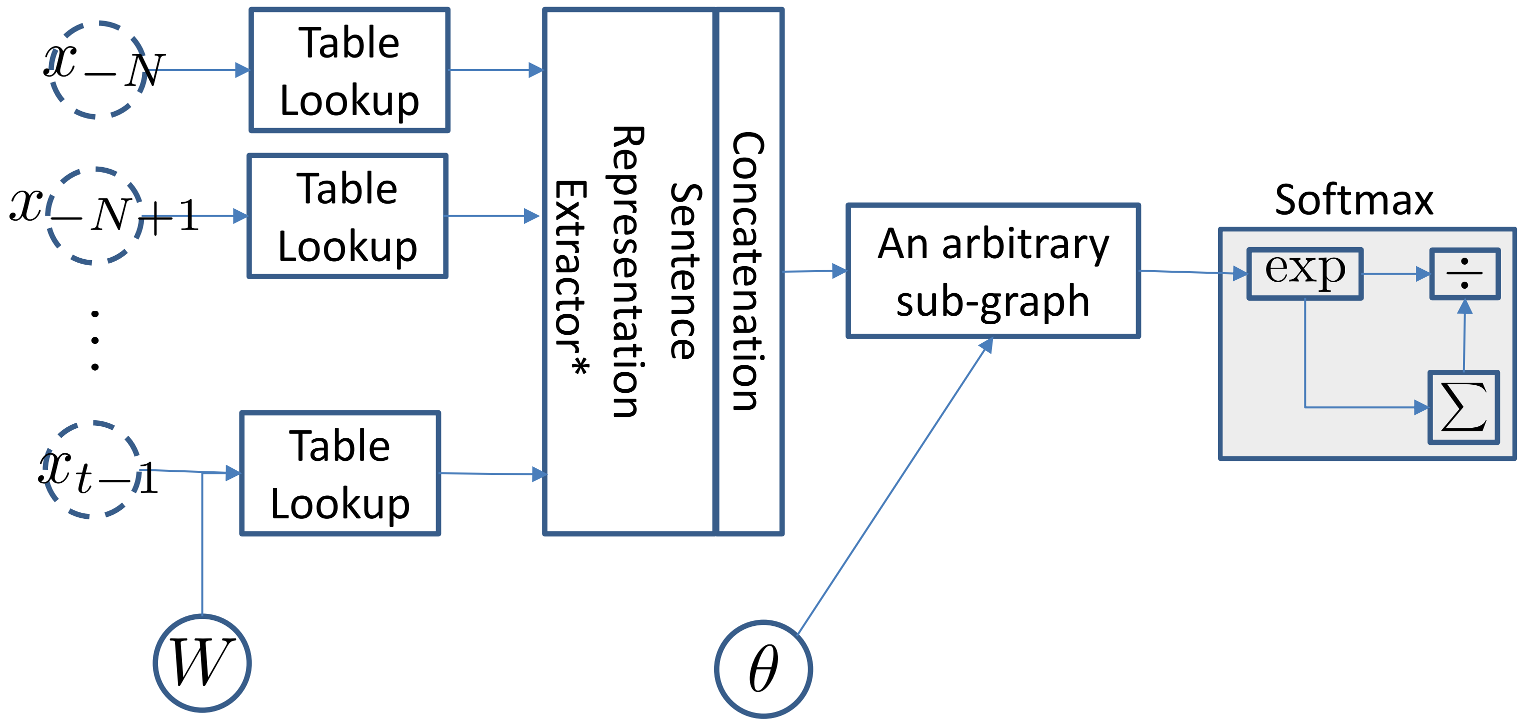
Quite popular:

- n-gram text models (estimate p by counting)
- FIR filters

How to efficiently handle large k?

Neural LM

Compute $p(x_t | x_{t-k+1}, \dots, x_{t-1})$ with a Neural Net



Looks somewhat like a convolution.

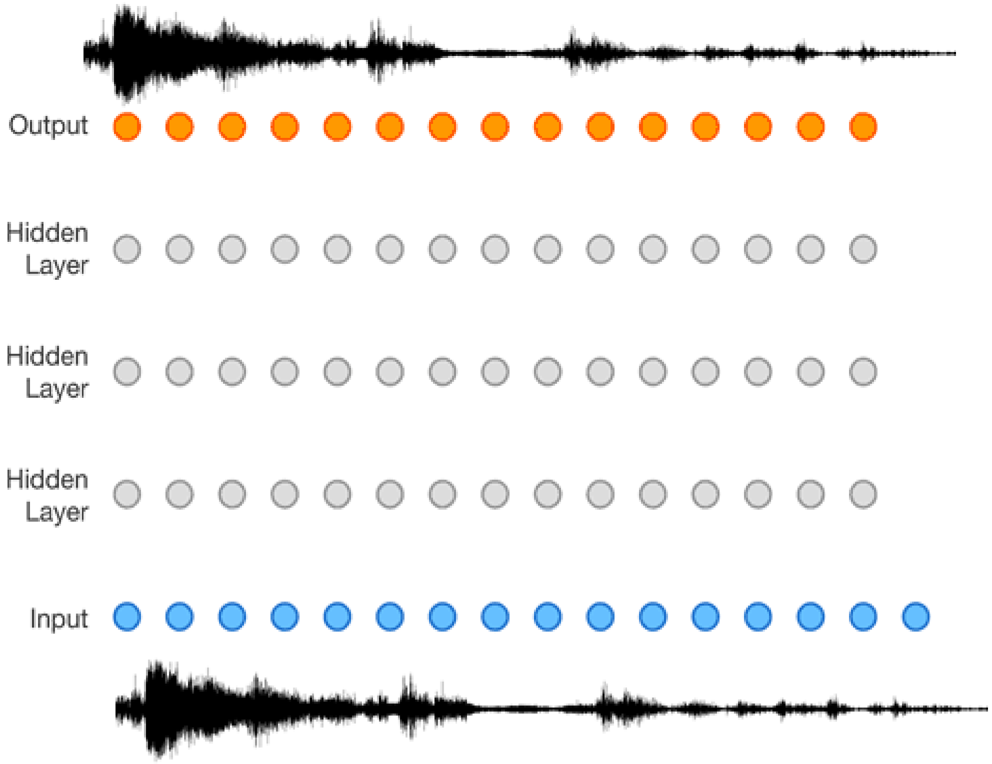
Still: large “n” => many params, how to fix?

Y. Bengio et al “A Neural Probabilistic Language Model”, NIPS 2011

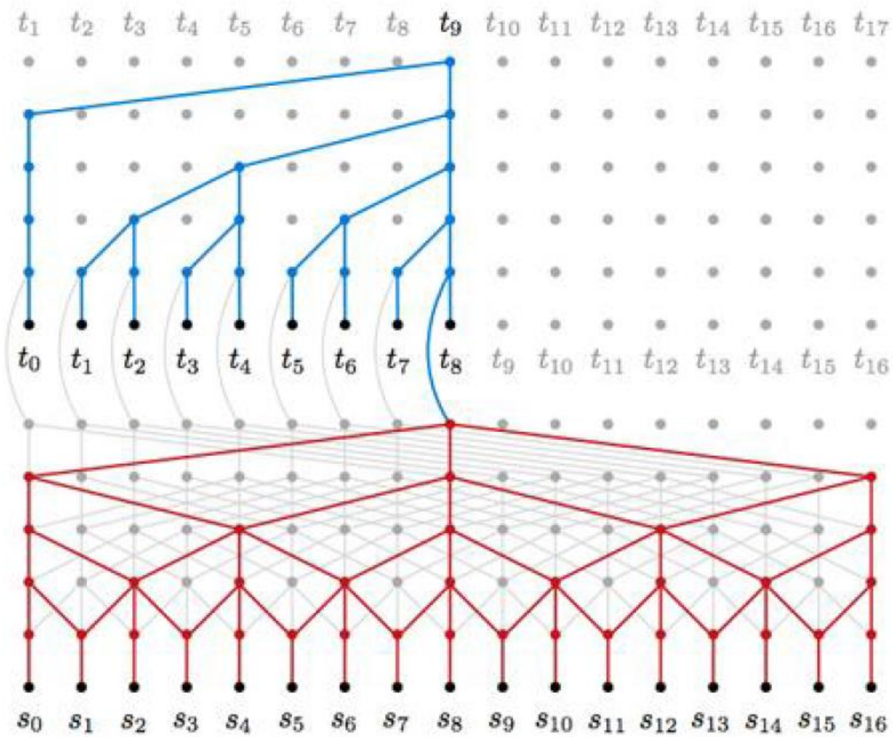
Dilated Convolutions

Large (but finite) receptive fields

Few parameters



WaveNet



ByteNet

https://arxiv.org/abs/1609.03499 https://arxiv.org/abs/1610.10099

Outline

- Why unsupervised learning
- Autoregressive models
  - Intuitions
  - Dilated convolutions
    RNNs (Intermezzo: LSTM training dynamics)
    Transformers
  - Beyond sequences, summary
- Latent variable models
  - VAE
  - Normalizing flows
- Autoregressive + latent variable: why and how?

Learning $p(x_t|x_1, \ldots, x_{t-1})$ #2

Introduce a hidden (i.e. not directly observed) state h_t.

h_t summarizes $x_1, \ldots, x_{t-1}$.

Compute h_t using this **recurrence**:

$$h_t = f(h_{t-1}, x_t)$$
$$p(x_{t+1}|x_1, \ldots, x_t) = g(h_t)$$

Recurrences are powerful

Input: a sequence of bits 1,0,1,0,0,1

Output: the parity

Solution:

The hidden state will be just 1 bit – parity so far:

$$h_0 = 0$$
$$h_t = XOR(h_{t-1}, x_t)$$
$$y_T = h_T$$

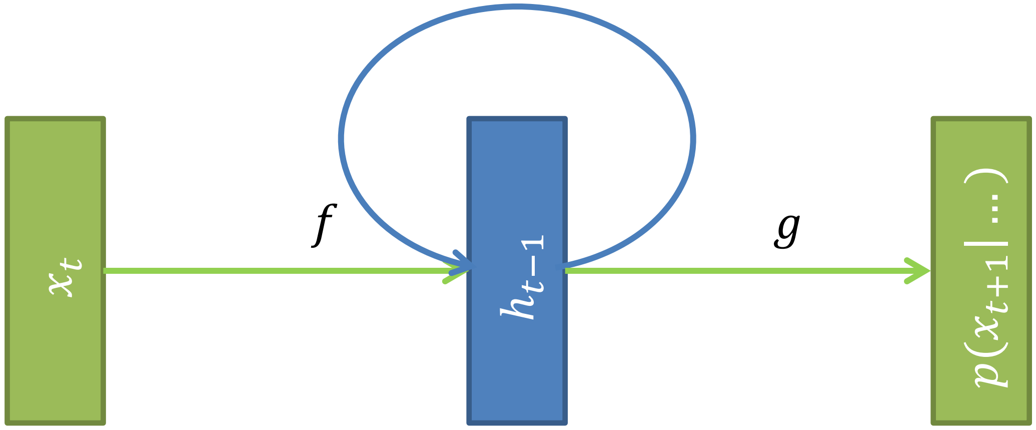
Constant operating memory!

Works for arbitrary long sequences!

Recurrent Neural Networks (RNNs)

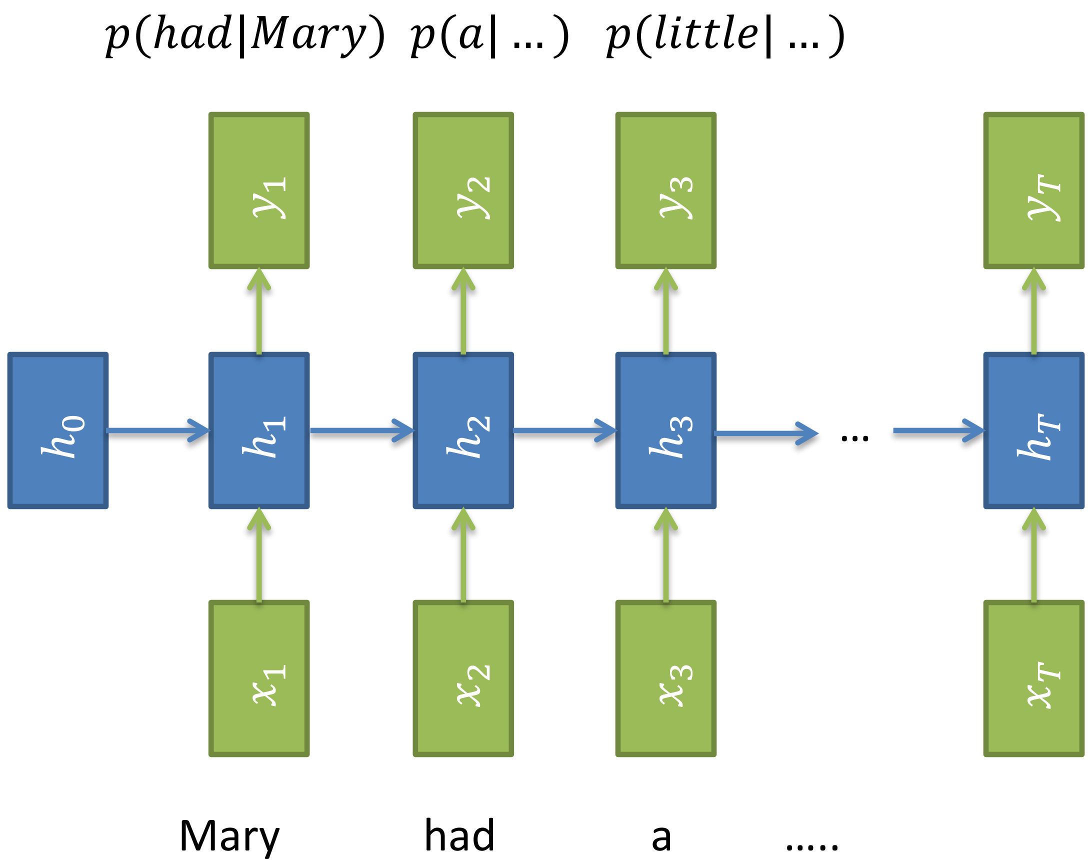
$$h_t = f(h_{t-1}, x_t)$$
$$p(x_{t+1} | x_1, \dots, x_t) = g(h_t)$$

f, g are implemented as feedforward neural nets.



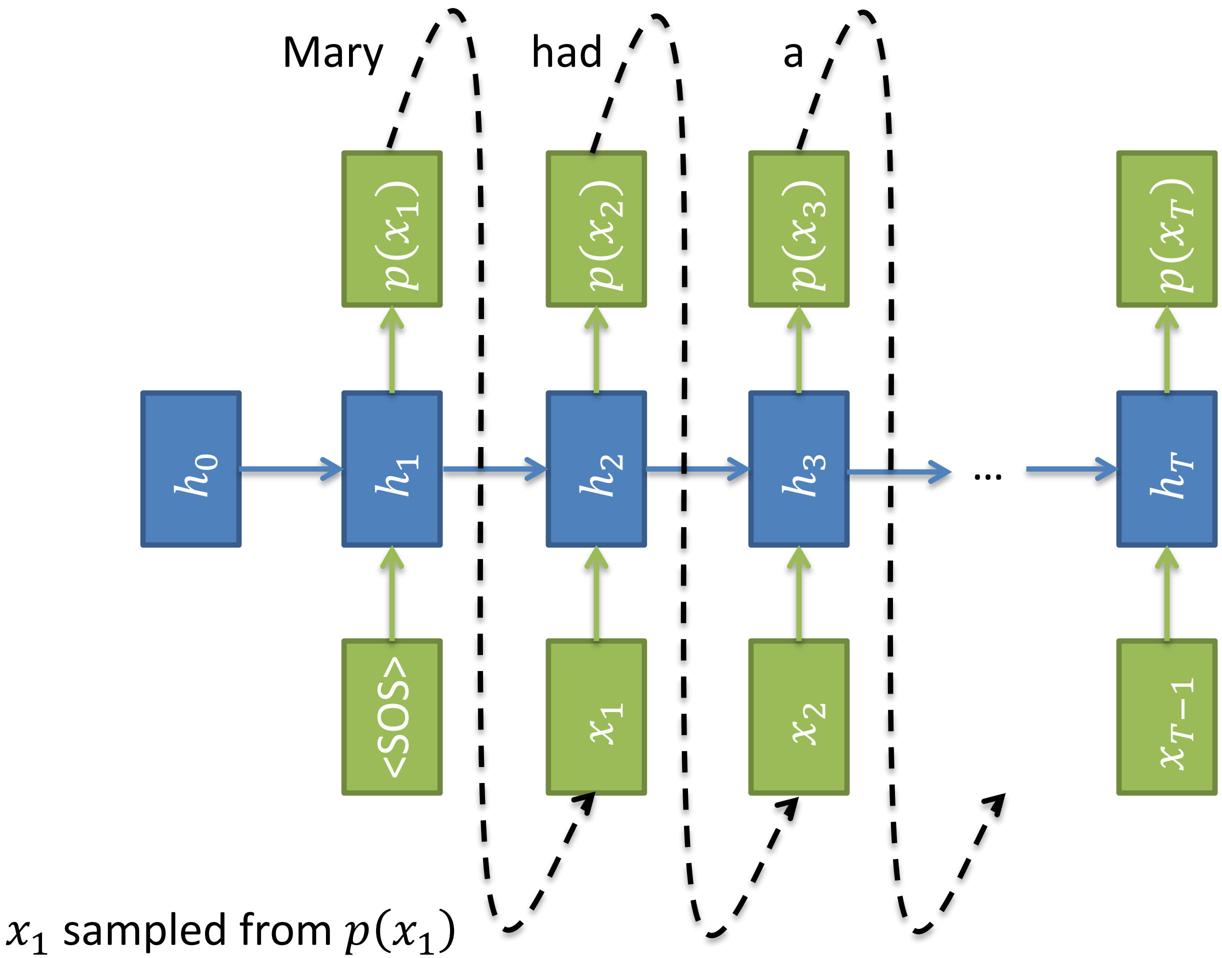
RNNs naturally handle sequences

Training: unroll and supervise at every step:

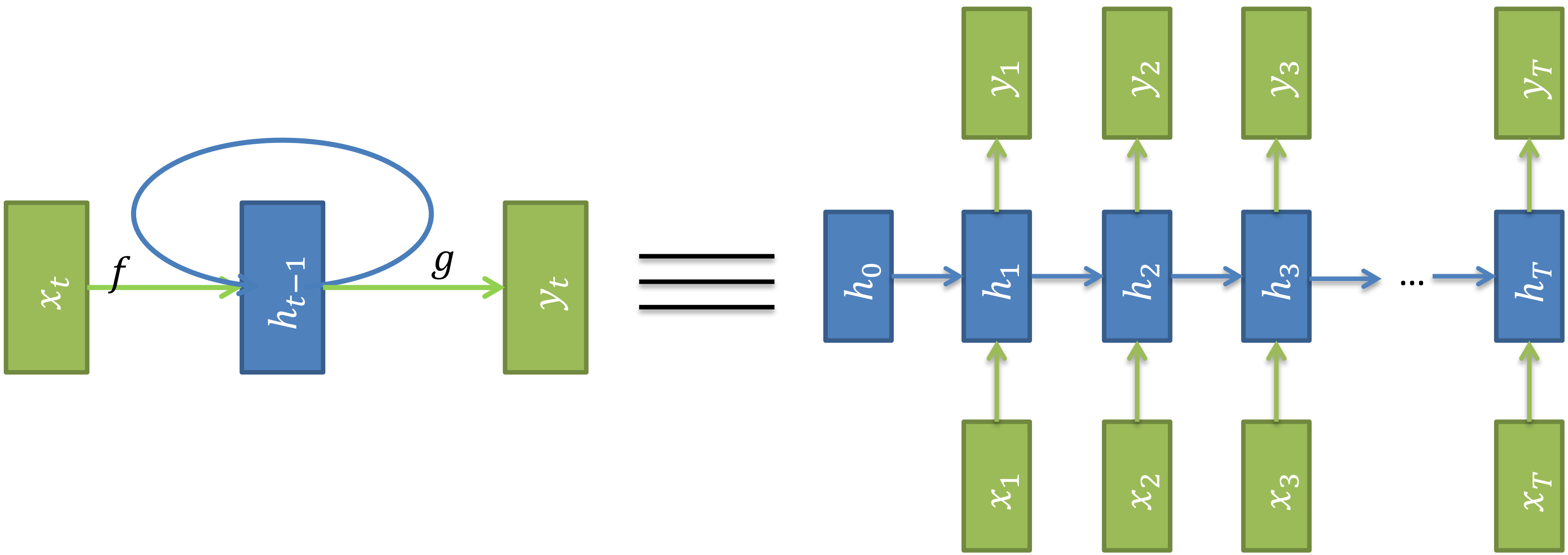


RNNs naturally handle sequences

Generation: sample one element at a time



RNNs are dynamical systems



RNN is a very deep network, all “arrows” compute the same function!

This impacts training!

RNN behavior - intuitions

A linear dynamical system computes

$$h_t = w \cdot h_{t-1} = w^t h_0$$

When:

1. $w > 1$ it diverges: $\lim_{t \to \infty} h_t = \mathrm{sign}(\mathrm{h}_0) \cdot \infty$
2. $w = 1$ it is constant $\lim_{t \to \infty} h_t = \mathrm{h}_0$
3. $-1 < w < 1$ it decays to 0: $\lim_{t \to \infty} h_t = 0$
4. $w = -1$ flips between $\pm h_0$
5. $w < -1$ diverges, changes sign at each step

A multidimensional linear dyn. system

$$h_t \in \mathbb{R}^n, W \in \mathbb{R}^{n \times n}$$
$$h_t = W h_{t-1}$$

Compute the eigendecomposition of W:

$$W = Q \Lambda Q^{-1} = Q \begin{bmatrix} \lambda_{11} & & \\ & \ddots & \\ & & \lambda_{nn} \end{bmatrix} Q^{-1}$$

Then

$$h_t = W h_{t-1} = W^n h_0 = Q \Lambda^{\mathrm{n}} Q^{-1} h_0 =$$
$$= Q \begin{bmatrix} \lambda_{11}^n & & \\ & \ddots & \\ & & \lambda_{nn}^n \end{bmatrix} Q^{-1} h_0$$

A multidimensional dyn. system cont.

$$h_t = W h_{t-1} = W^n h_0 = Q \Lambda^{\mathrm{n}} Q^{-1} h_0$$

If largest eigenvalue has norm:

1. >1, system diverges
2. =1, system is stable or oscillates
3. <1, system decays to 0

This is similar to the scalar case.

A nonlinear dynamical system

$$h_t = \tanh(w \cdot h_{t-1})$$

Output is bounded (can't diverge!)

If $w > 1$ has 3 fixed points: $0, \pm h_f$. Starting the iteration form $h_0 \neq 0$ it ends at $\pm h_f$ (effectively remembers the sign).

If $w \leq 1$ has one fixed point: 0

A nonlinear dynamical system

$$h_t = \tanh(w \cdot h_{t-1}), w > 1$$

Gradients in RNNs

Recall RNN equations:

$$h_t = f(h_{t-1}, x_t; \theta)$$
$$y_t = g(h_t)$$

Assume supervision only at the last step:

$$\mathrm{L} = e(y_T)$$

The gradient is

$$\frac{\partial L}{\partial \theta} = \sum_t \frac{\partial L}{\partial h_T} \boldsymbol{\frac{\partial h_T}{\partial h_t}} \frac{\partial h_t}{\partial \theta}$$

Trouble with $\frac{\partial h_T}{\partial h_t}$

$\frac{\partial h_T}{\partial h_t}$ measures how much h_T changes when h_t changes.

Another look at $\frac{\partial h_T}{\partial h_t}$

Let:

$$h_t = \tanh(w \cdot h_{t-1})$$

Then:

$$\frac{\partial h_t}{\partial h_0} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \dots \frac{\partial h_1}{\partial h_0}$$

$$\frac{\partial h_{i+1}}{\partial h_i} = \tanh{}'(wh_t)\, w$$

$$\frac{\partial h_t}{\partial h_0} = \prod_{i=0}^{t-1} \tanh{}'(wh_i)\, w = w^t \prod_{i=0}^{t-1} \tanh{}'(wh_i)$$

Backward phase is linear!

Vanishing gradient

If $\frac{\partial h_T}{\partial h_t} = 0$ the network forgets all information from time t when it reaches time T.

This makes it impossible to discover correlations between inputs at distant time steps.

This is a modeling problem.

Exploding gradient

When $\frac{\partial h_T}{\partial h_t}$ is large $\frac{\partial \mathrm{Loss}}{\partial \theta}$ will be large too.

Making a gradient step $\theta \leftarrow \alpha \frac{\partial \mathrm{Loss}}{\partial \theta}$ can drastically change the network, or even destroy it.

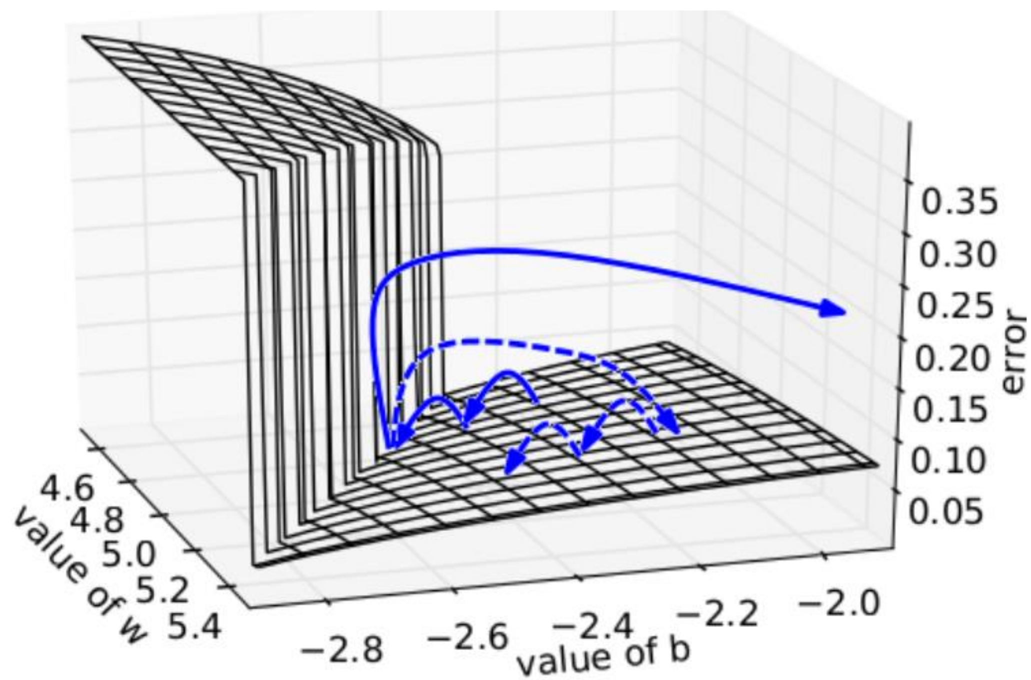
This is not a problem of information flow, but of training stability!

Exploding gradient intuition #1

$$h_t = \tanh(2h_{t-1} + b)$$

Exploding gradient intuition #2

$$h_0 = \sigma(0.5)$$
$$h_t = \sigma(w h_{t-1} + b)$$
$$L = (h_{50} - 0.7)^2$$



R. Pascanu et al. https://arxiv.org/abs/1211.5063

Summary: trouble with $\frac{\partial h_T}{\partial h_t}$

RNNs are difficult to train because:

$\frac{\partial h_T}{\partial h_t}$ can be 0 – vanishing gradient

$\frac{\partial h_T}{\partial h_t}$ can be ∞ – exploding gradient

With both phenomena governed by the spectral norm of the weight matrix (norm of the largest eigenvalue, magnitude of w in the scalar case).

Exploding gradient solution

Don’t do large steps.

Pick a gradient norm threshold and scale down all larger gradients.

This prevents the model from doing a large learning update and destroying itself.

Vanishing gradient solution: LSTM, the RNN workhorse

$\frac{\partial h_T}{\partial h_t} \approx 0$: step T forgots information from t

The dynamical system

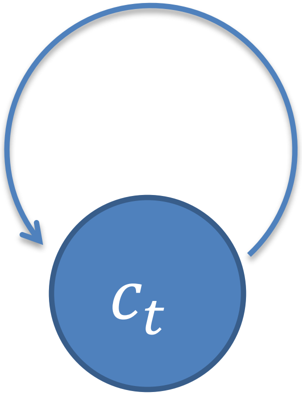
$$h_t = w h_{t-1}$$

maximally preserves information when $w = 1$

LSTM introduces a memory cell c_t that will keep information forever:

$$c_t = c_{t-1}$$

Memory cell

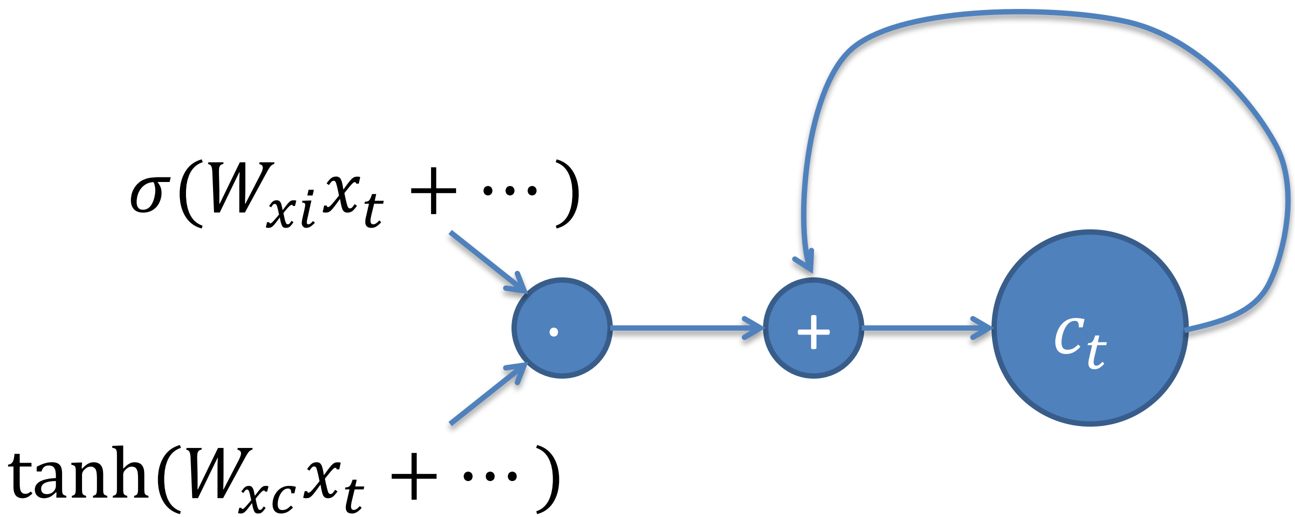


Memory cell preserves information

$$c_t = c_{t-1}$$

$$\frac{\partial c_T}{\partial c_t} = 1$$

Gates



Gates selectively load information into the memory cell:

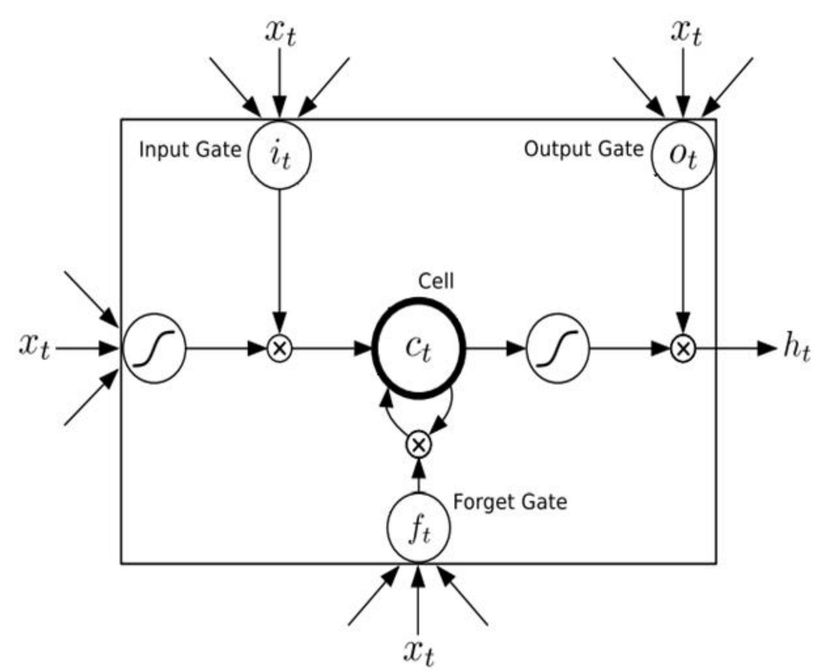
$$i_t = \sigma(W_{xi}x_t + \cdots)$$
$$c_t = c_{t-1} + i_t \cdot \tanh(W_{xc}x_t + \cdots)$$

LSTM: the details

Hidden state is a pair of:
-c_t information in the cell, hidden from the rest of the network
-h_t information extracted form the cell into the network

Update equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right)$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$c_t = i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) + f_t c_{t-1}$$
$$h_t = o_t \tanh c_t$$



RNNs summary

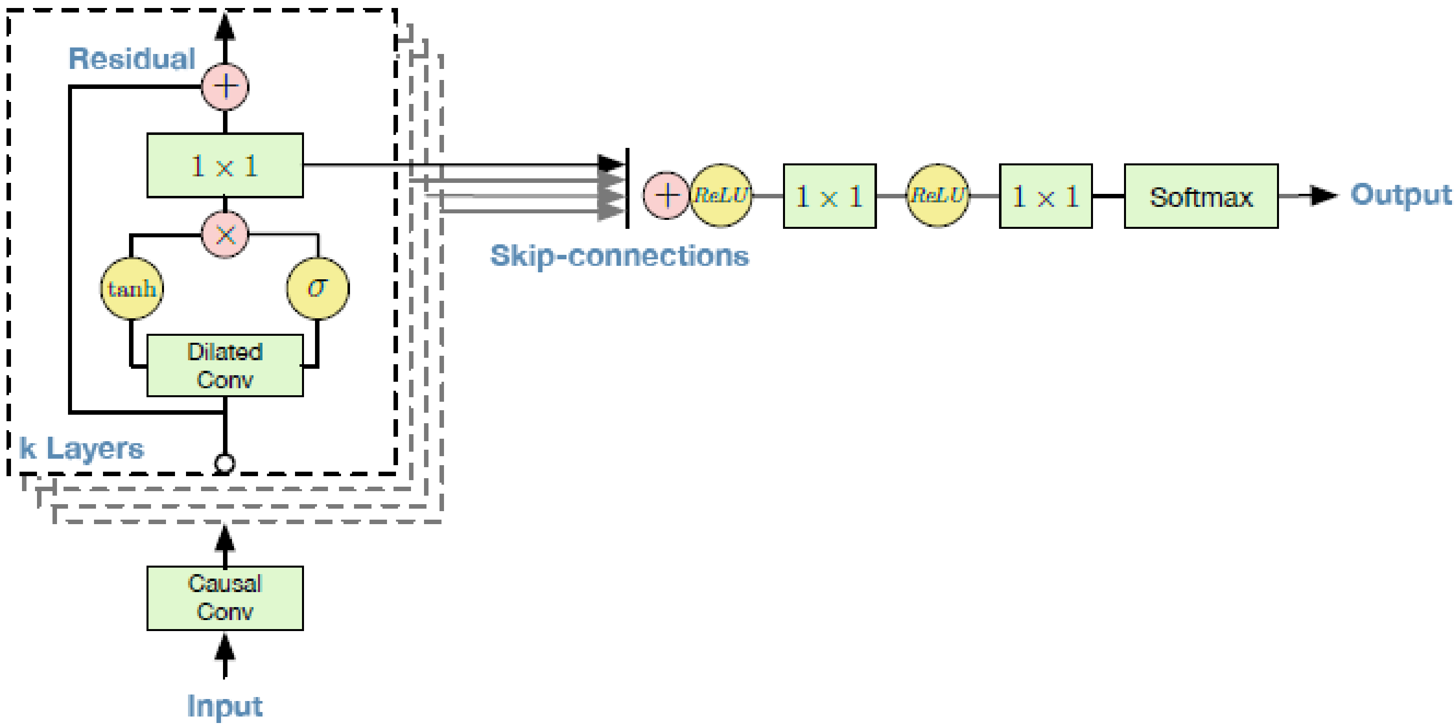
LSTM/GRU can remember long histories:

- #params and memory footprint independent of sequence length
- gates control information flow
- each element is processed in constant time:
  - RNN needs to cram all past into a vector
  - in practice this limits the receptive field

LSTMs gave us gating!

Gates decouple **if** from **what**

WaveNet & Conv models for text use gates too!



Outline

- Why unsupervised learning
- Autoregressive models
  - Intuitions
  - Dilated convolutions
    RNNs (Intermezzo: LSTM training dynamics)
    Transformers
  - Beyond sequences, summary
- Latent variable models
  - VAE
  - Normalizing flows
- Autoregressive + latent variable: why and how?

Learning $p(x_t|x_1, \dots, x_{t-1})$ #3

Reduce $x_1, \dots, x_{t-1}$ to a fixed-size representation

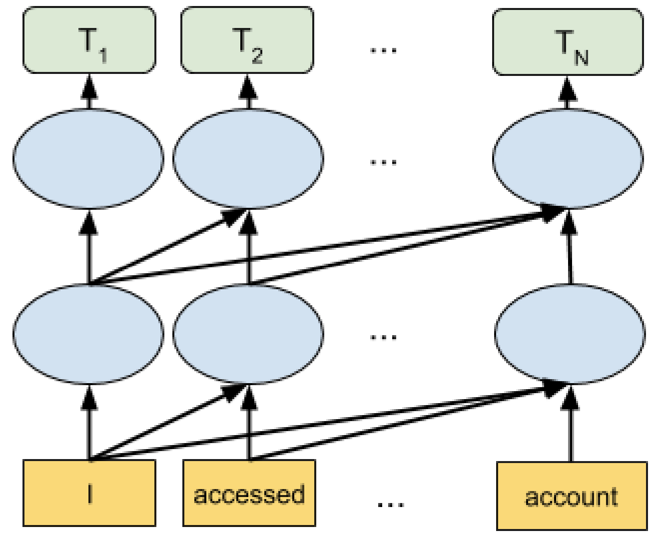
E.g. take weighted average of $x_1 \dots x_{t-1}$

This works really well with self-attention!

Attention ~ unbounded convolution

Slide by L. Kaiser https://nlp.stanford.edu/seminar/details/lkaiser.pdf

Attention details



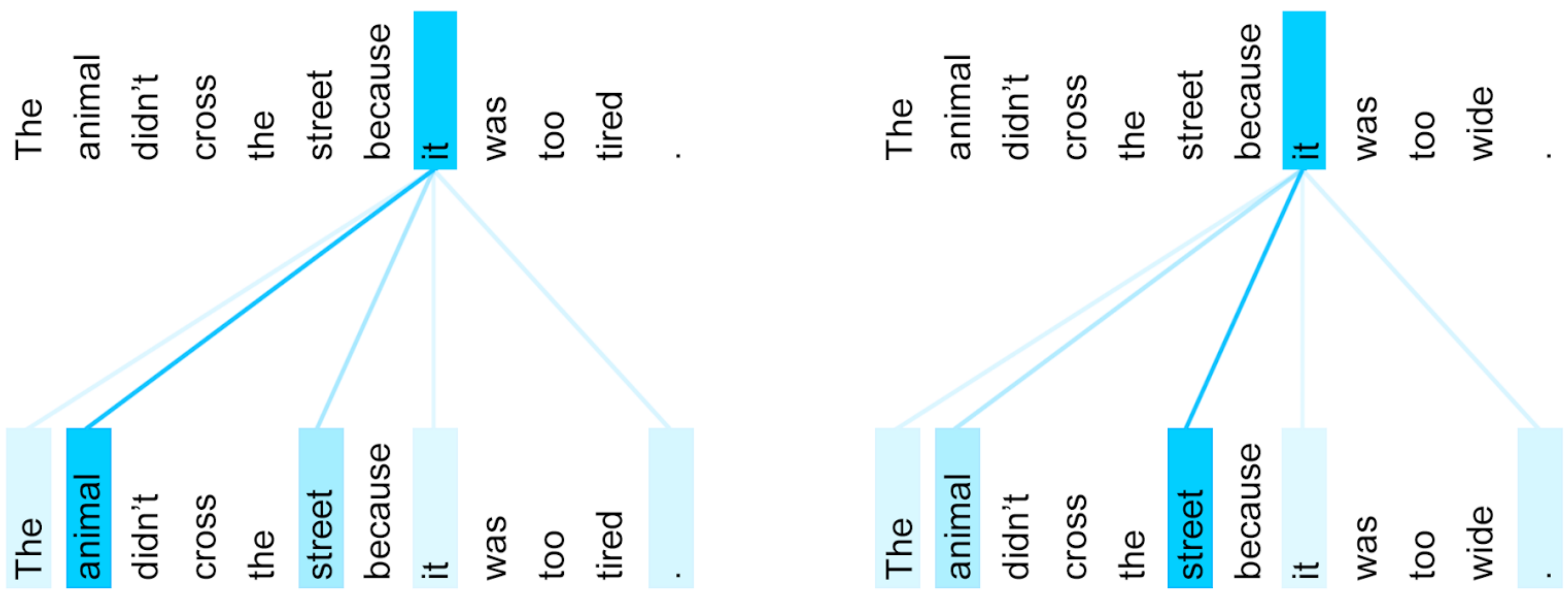
The past encodings form a list of key-value pairs: $[(k_i, V_i)], k \in \mathbb{R}^d, V \in \mathbb{R}^D$

At step t we start with a query $q \in \mathbb{R}^d$

1. Match q to all keys: $\alpha_i = \frac{e^{q^T k_i}}{\sum_j e^{q^T k_j}}$
2. Compute an average: $R = \sum_i \alpha_i V_i$

Attention in action

- This is part of an encoder in NMT. The attention can look forward and backward!



https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html?m=1

GPT2 Transformer in action

The JSALT summer school is about …

GPT2 Transformer in action

The JSALT summer school is about developing creative, innovative ways to solve global challenges, whether the problem is creating sustainable energy systems, promoting economic development, improving global health or making the world more energy independent.

The JSALT summer school is funded by an international grant for its education. Students enter the JSALT program on the basis of merit, with the objective of applying their talents to global development issues. They learn how to: develop, design and produce sustainable and integrated energy systems, promote economic development, increase efficiency of energy, develop a healthy environment for the health and well being of people and the environment, develop sustainable, clean and abundant

https://talktotransformer.com/

Self Attention Summary

params independent from history length + lots of parameter reuse:

- effectively processes very long histories

Trades compute time for better performance!

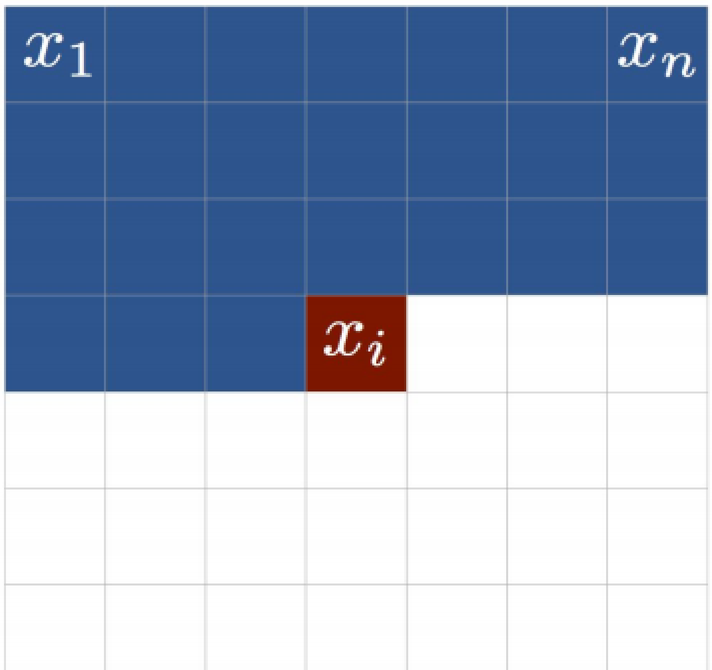
- no need to cram all past into a vector!
- n-th step requires $O(n)$ ops

Outline

- Why unsupervised learning
- Autoregressive models
  - Intuitions
  - Dilated convolutions
    RNNs (Intermezzo: LSTM training dynamics)
    Transformers
  - **Beyond sequences, summary**
- Latent variable models
  - VAE
  - Normalizing flows
- Autoregressive + latent variable: why and how?

Beyond sequences

PixelRNN: A "language model for images"



Pixels generated left-to-right, top-to-bottom.

Cond. probabilities estimated using recurrent or convolutional neural networks.

van den Oord, A., et al. "Pixel Recurrent Neural Networks." ICML (2016).

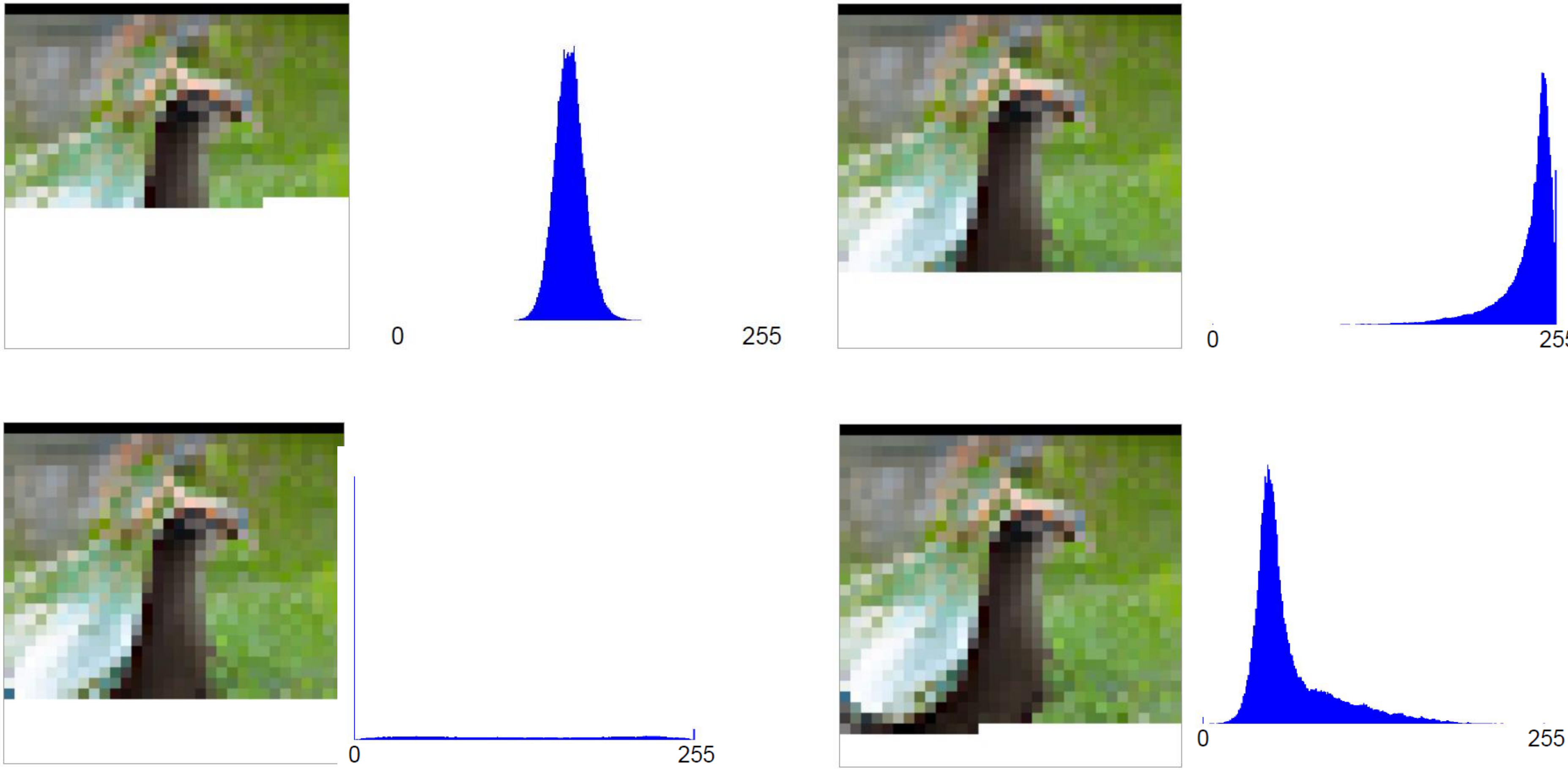
Modeling pixels

How to model pixel values:

- A Gaussian with fixed st. dev?
- A Gaussian with tunable st. dev?
- A distribution over discrete levels [0,1,2,…255]?

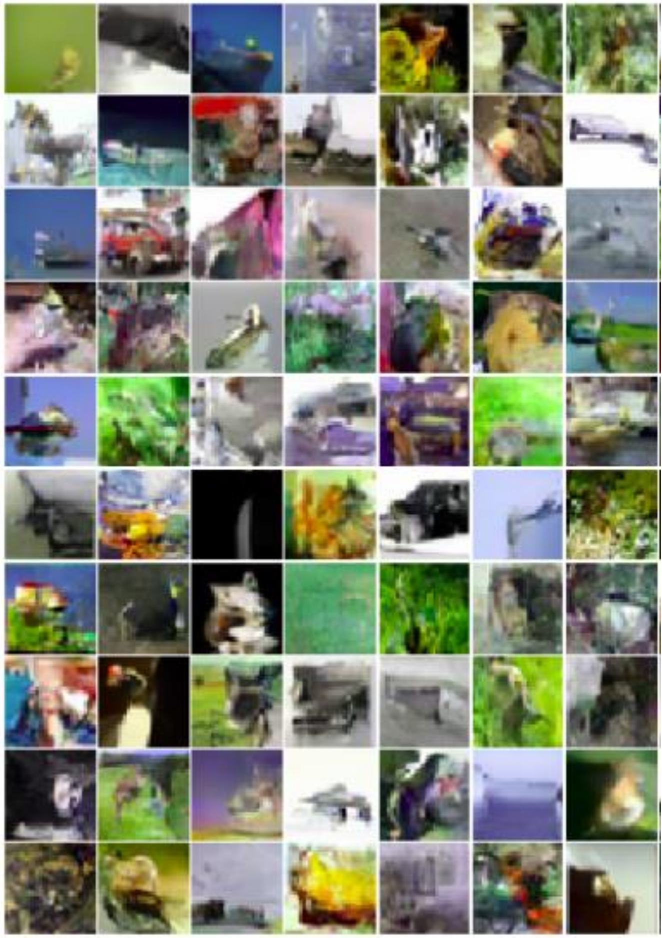
What are the implications?

Modeling pixel values



Model works best with a flexible distribution: better to use a SoftMax over pixel values!

PixelCNN samples & completions



Salimans et al, "A PixelCNN Implementation with Discretized Logistic Mixture Likelihood and Other Modifications"
van den Oord, A., et al. "Pixel Recurrent Neural Networks." ICML (2016).

Autoregressive Models Summary

The good:

- Easy to exploit correlations in data.
- Reduce data generation to many small decisions
  - Simple define (just pick an ordering)
  - Trains like fully supervised
  - Model operations are deterministic, randomness needed during generation
- Often SOTA log-likelihood

Autoregressive Model Summary

The bad:

- Train/test mismatch (teacher forcing): trained on ground truth sequences but applied to own predictions
- Generation requires $O(n)$ steps (Training can be sometimes parallelized)
- No compact intermediate data representation, not obvious how to use for downstream tasks.

Outline

- Why unsupervised learning
- Autoregressive models
  - Intuitions
  - Dilated convolutions
    RNNs (Intermezzo: LSTM training dynamics)
    Transformers
  - Beyond sequences, summary
- **Latent variable models**
  - **VAE**
  - Normalizing flows
- Autoregressive + latent variable: why and how?

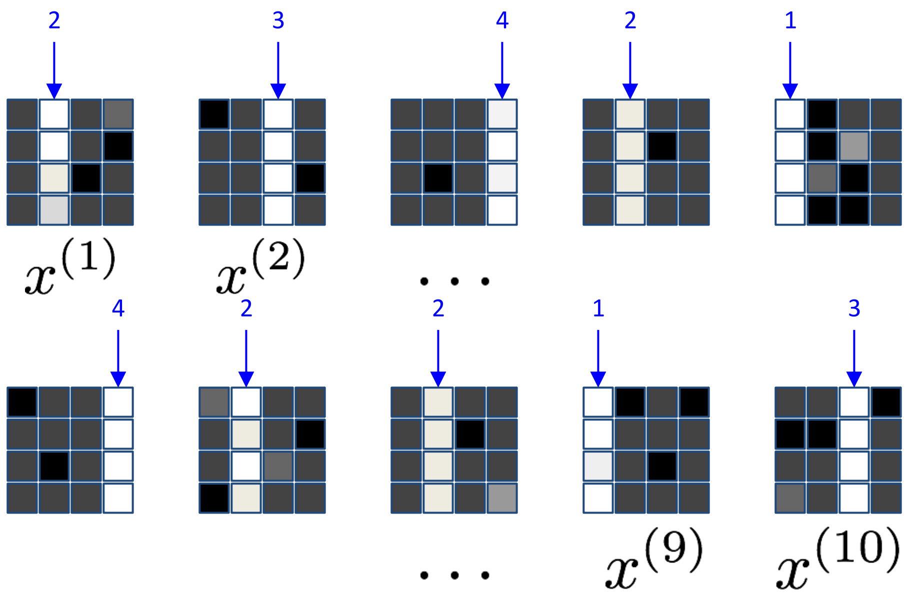
Ad break

I took the following slides from Ulrich Paquet

See https://www.youtube.com/watch?v=xTsnNcctvmU for a recording of a his explanation!

Latent Variable Models

Intuition: the data have a simple structure



Data structure

We can capture most of the variability in the data through **one** number

$$z^{(n)} = 1 \text{ or } 2, 3, 4$$

for each image *n*, even though each image is 16 dimensional

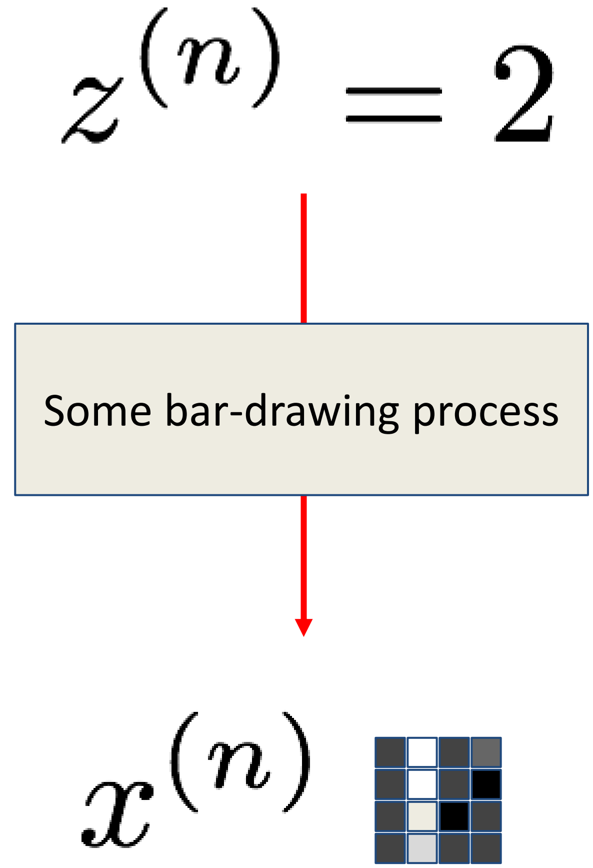
How?

How?

Take $z^{(n)} = 2$

Draw bar in column 2 of image

Et voila! You have $x^{(n)}$

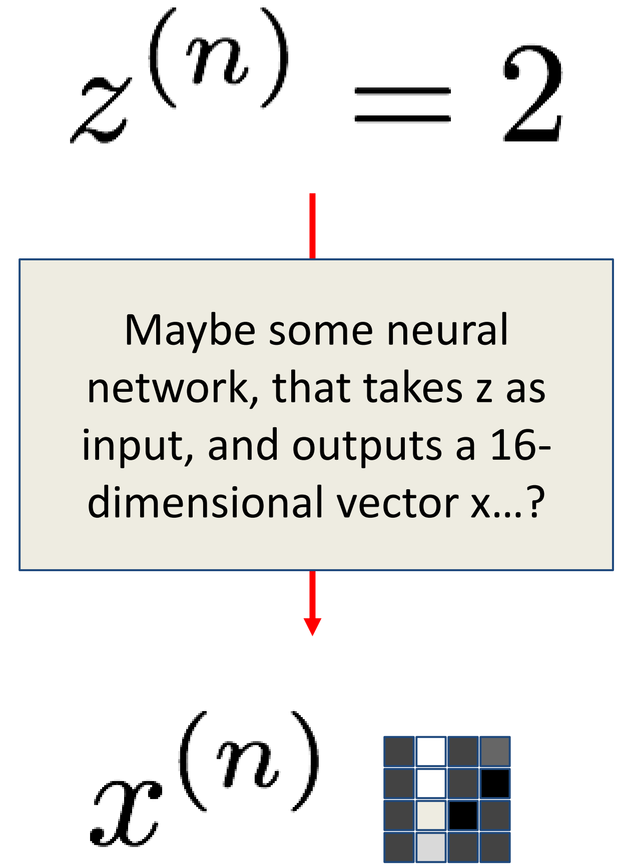


How?

Take $z^{(n)} = 2$

Draw bar in column 2 of image

Et voila! You have $x^{(n)}$



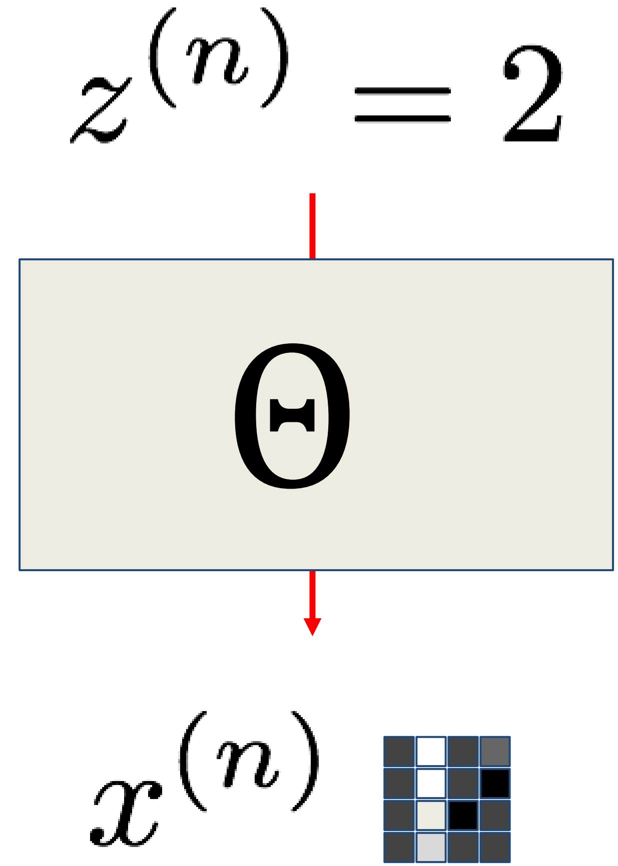
Exercise

Write or draw a function (like a multi-layer perceptron) that takes $z \in \mathbb{R}$ and produces x

Is your input one-dimensional?

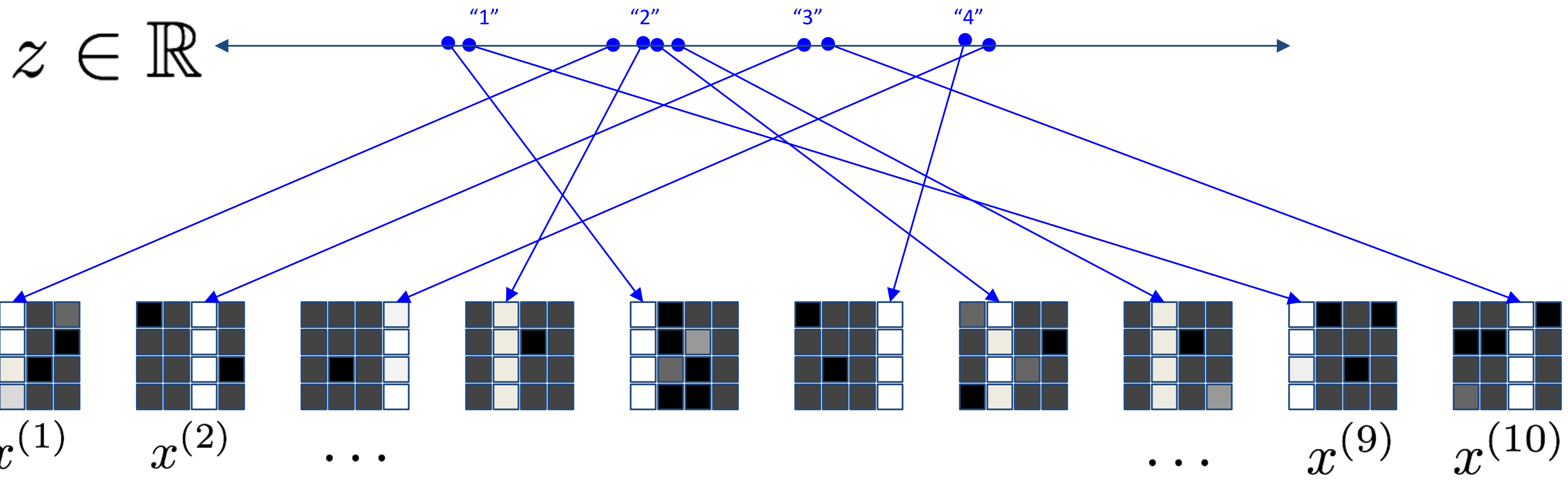
Is your output 16-dimensional?

Identify all the "tunable" parameters Θ of your function



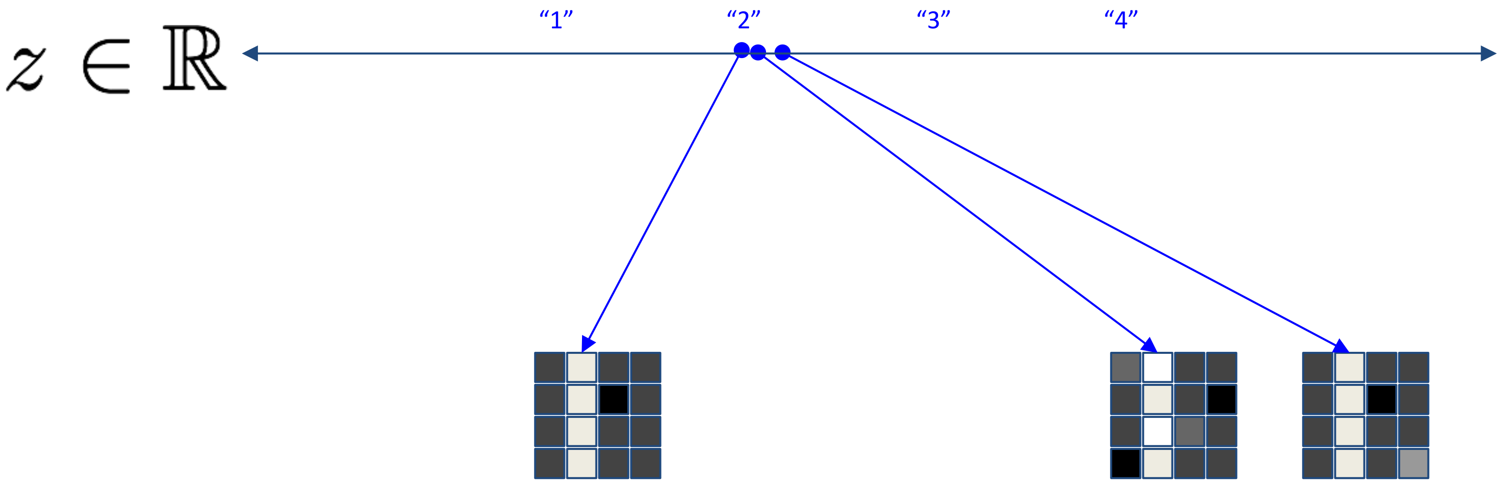
Data manifold

The 16-dimensional images live on a 1-dimensional manifold, plus some "noise"



and noise

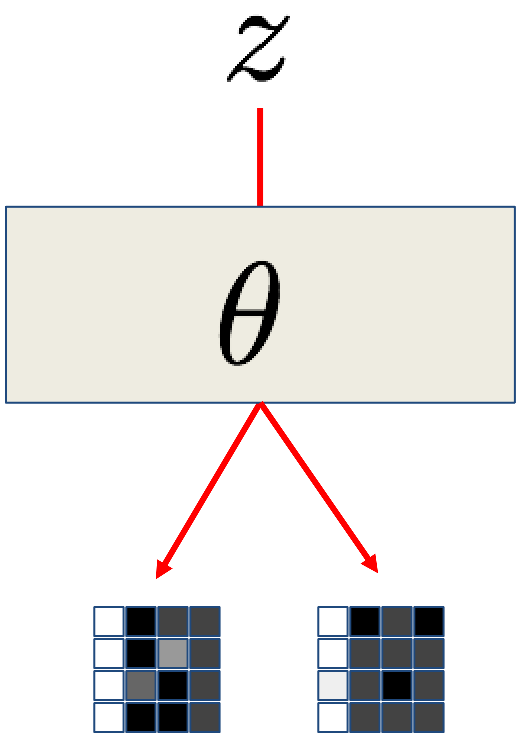
The 16-dimensional images live on a 1-dimensional manifold, plus some “noise”



Exercise

Change the neural network to take z and produce a distribution over x:

$$p_\Theta(x|z)$$



Generation and Inference

Generation:

$$p(z)$$
$$p_\Theta(x|z)$$

Inference:

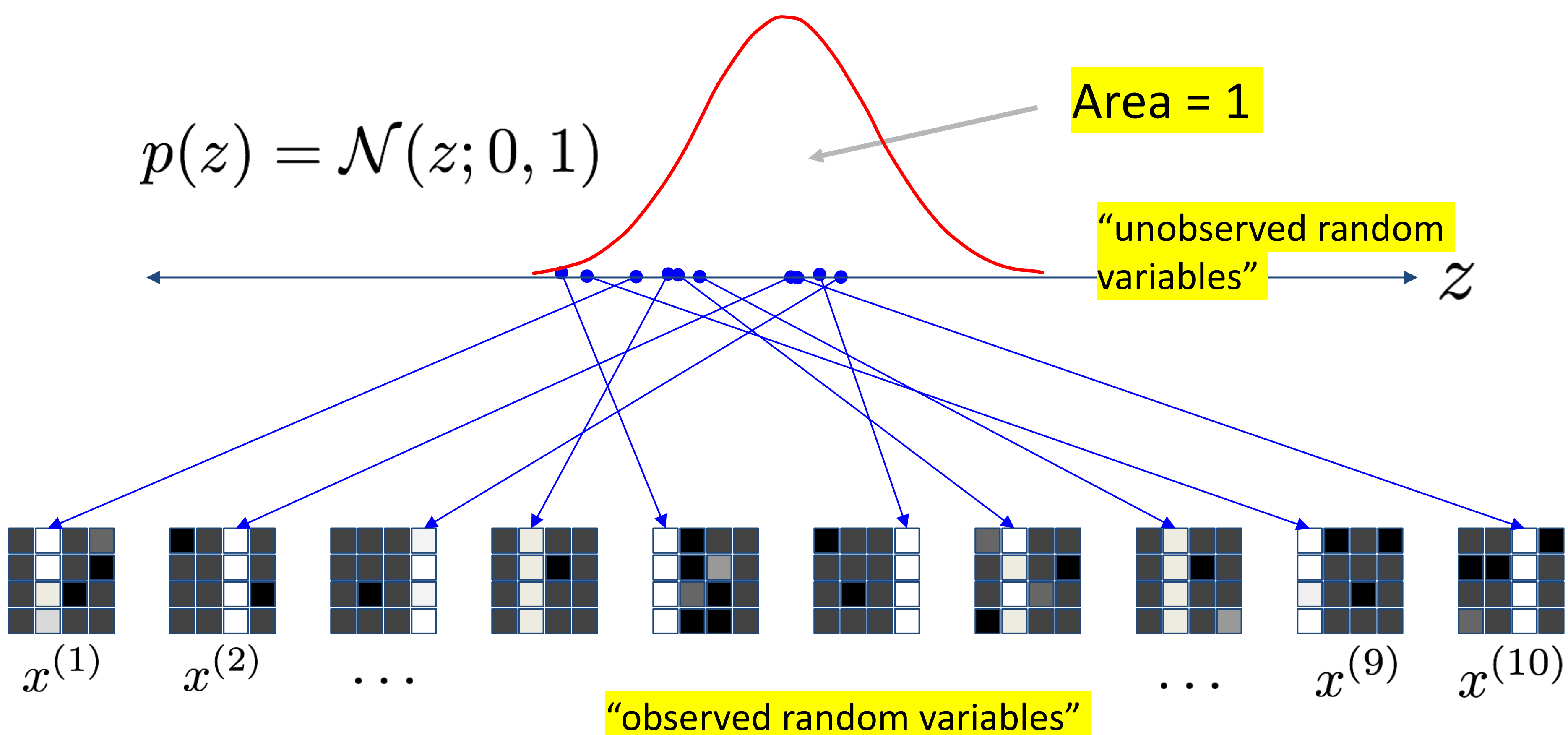
$$p_\Theta(z|x) = ????$$

Bayes says:

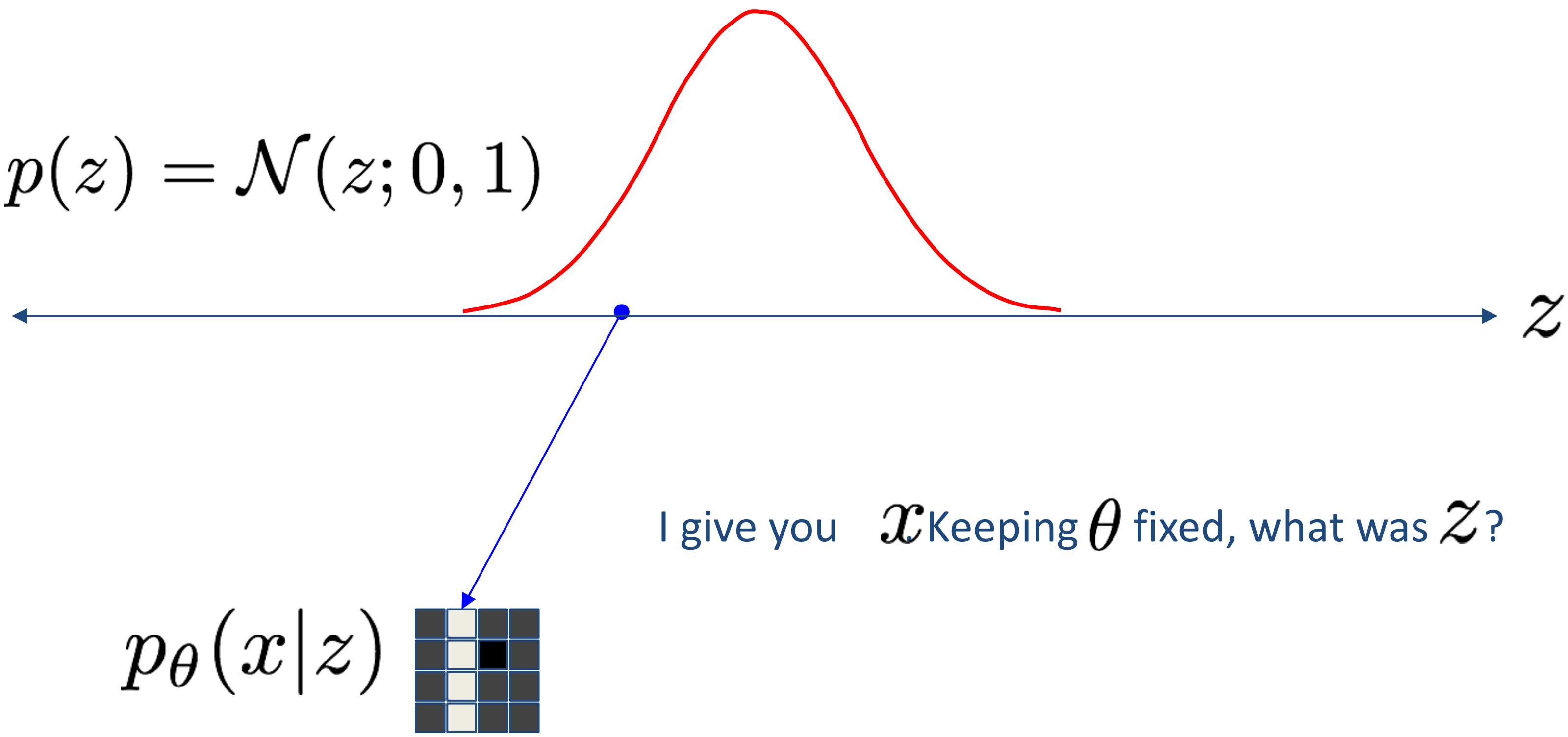
$$p_\Theta(z|x) = \frac{p_\Theta(x|z)p(z)}{\int dz' p_\Theta(x|z')p(z')}$$

But often it’s intractable ☹

Inference starts with priors

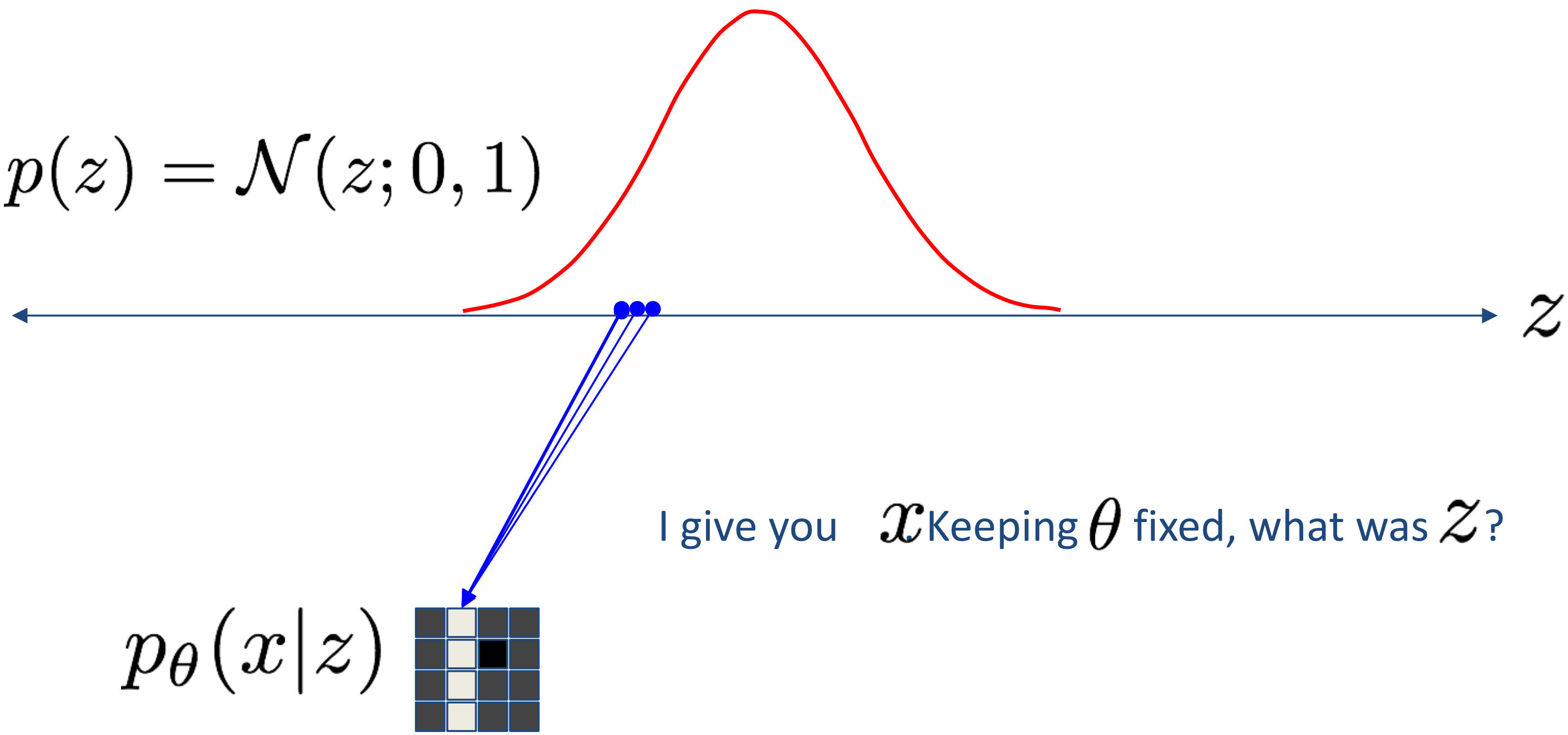


Inference



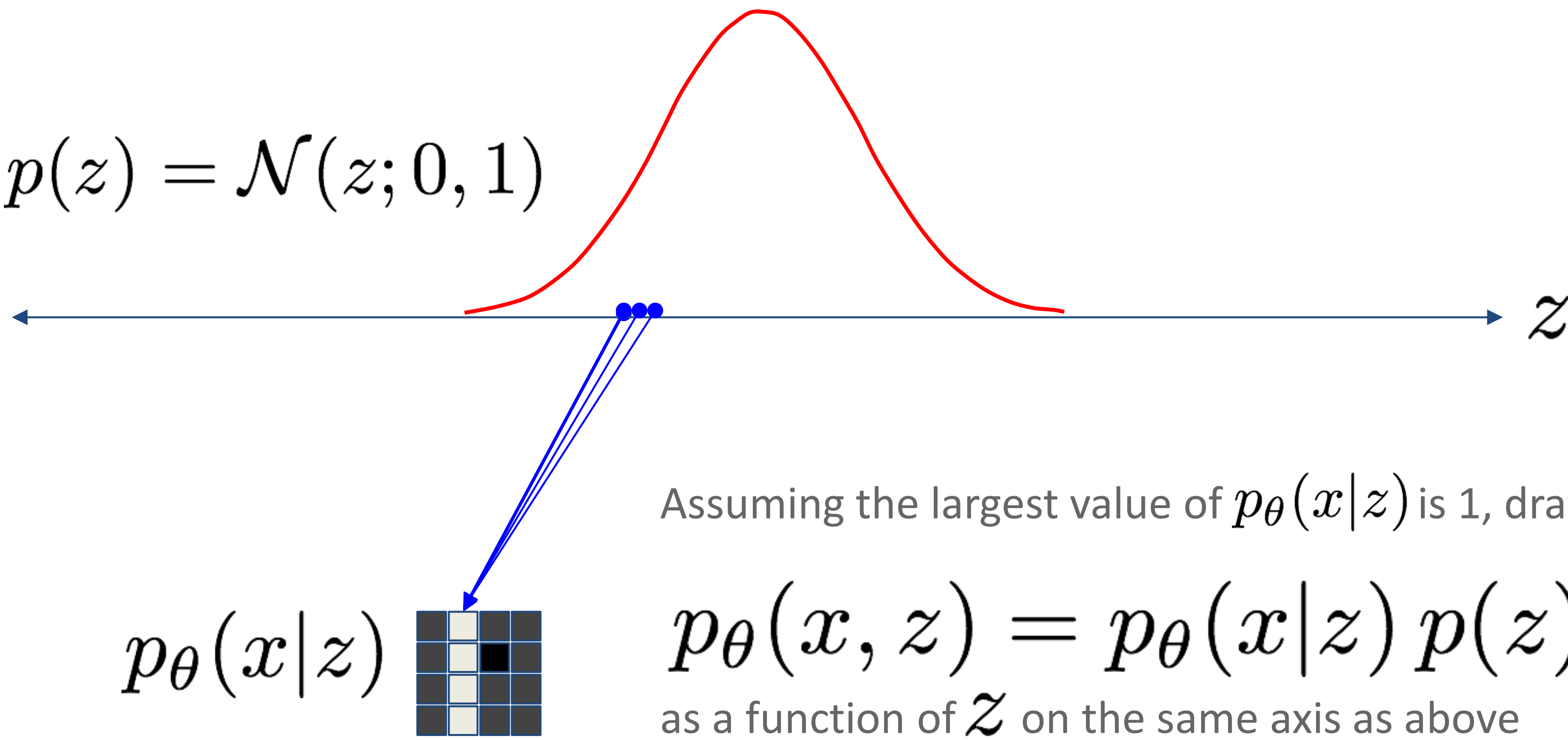
I give you x Keeping θ fixed, what was z?

Inference



I give you x Keeping θ fixed, what was z?

Exercise

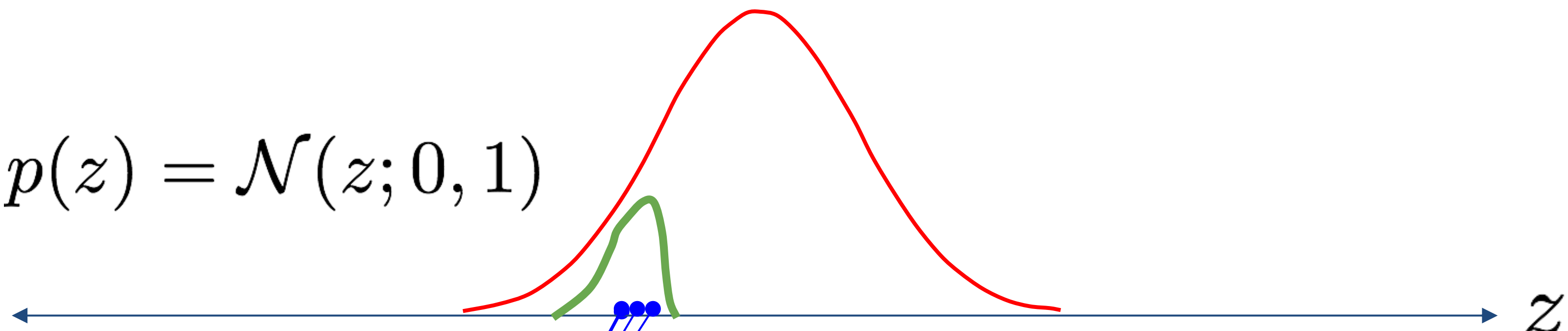


Assuming the largest value of $p_\theta(x|z)$ is 1, draw

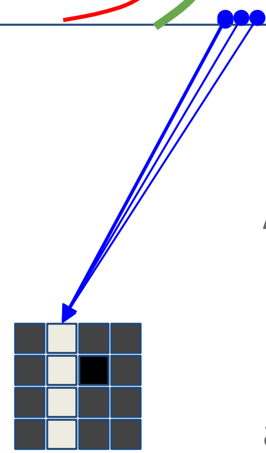
$$p_\theta(x, z) = p_\theta(x|z)\, p(z)$$

as a function of z on the same axis as above

Joint density (with x observed)



$p_\theta(x|z)$

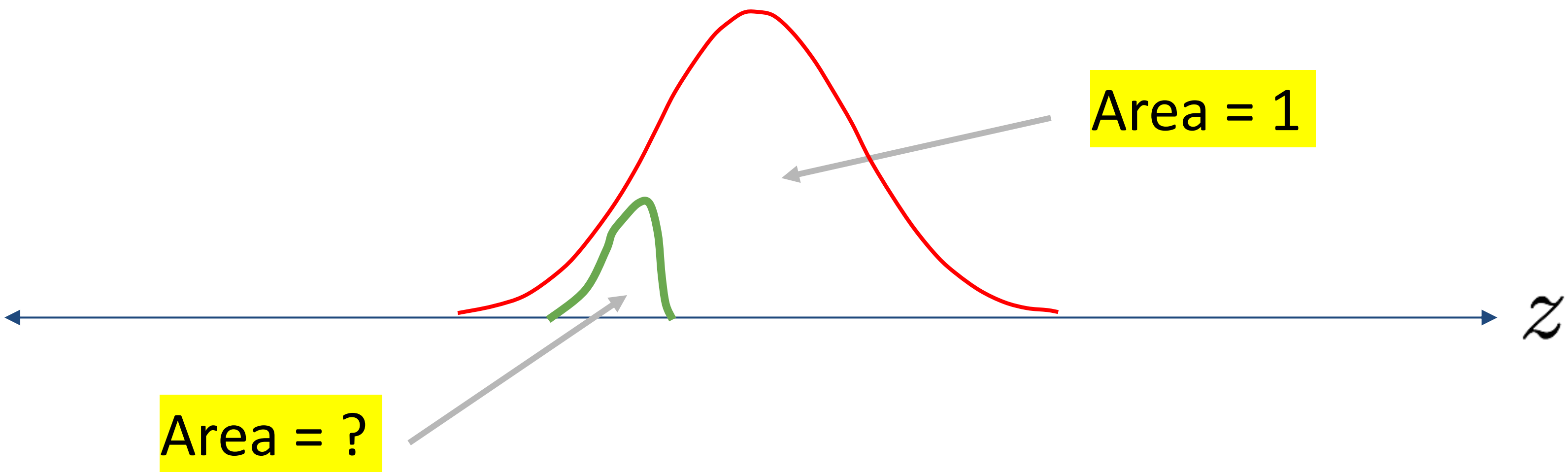


Assuming the largest value of $p_\theta(x|z)$ is 1, draw

$$p_\theta(x, z) = p_\theta(x|z)\, p(z)$$

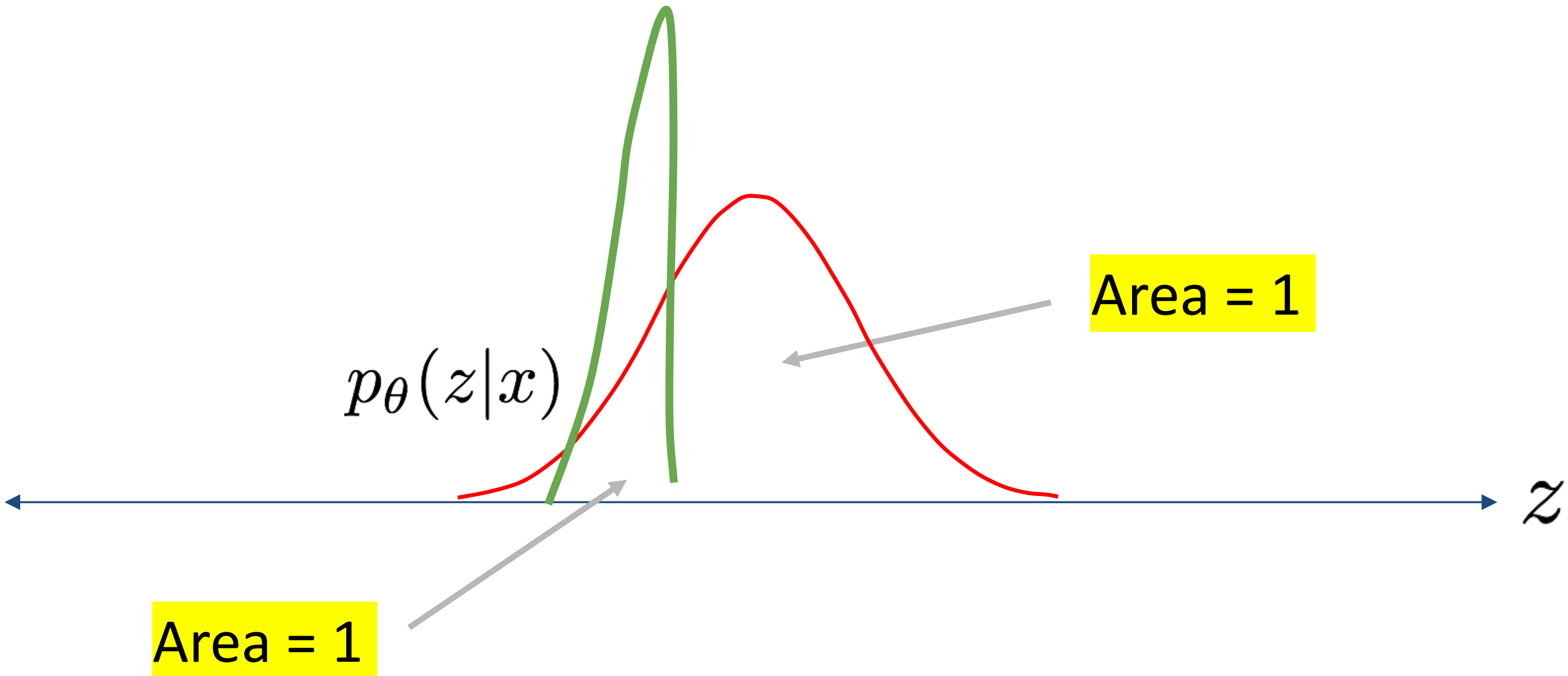
as a function of z on the same axis as above

Joint density (with x observed)



1-minute exercise:
what is the area?

Posterior

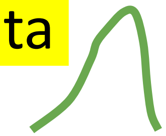


$$p_\theta(z|x) = \frac{p_\theta(x|z)\,p(z)}{p_\theta(x)}$$

Dividing by the marginal likelihood (evidence) scales the area back to 1...

Evidence of all data points

Area for data point *n*



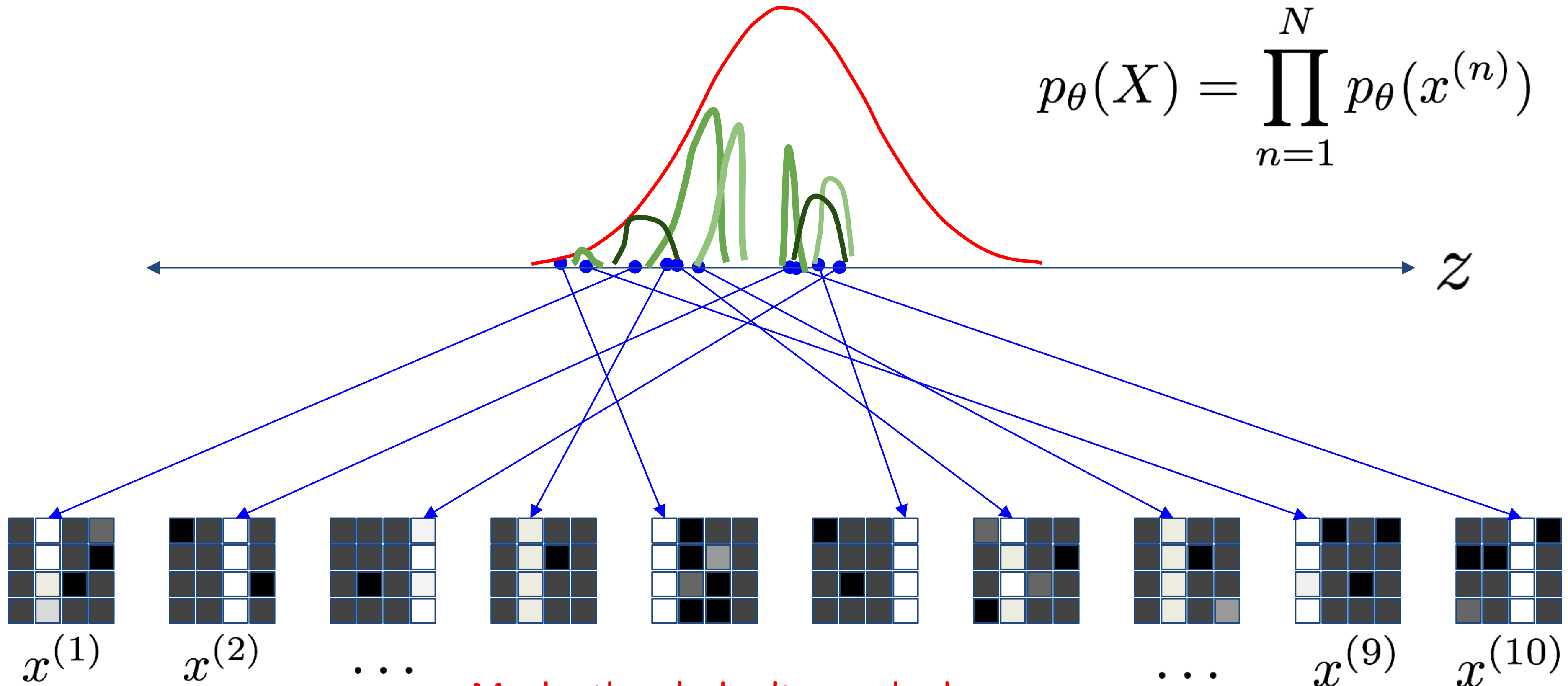
$$p_\theta(X) = \prod_{n=1}^{N} p_\theta(x^{(n)})$$

$$\log p_\theta(X) = \sum_{n=1}^{N} \log p_\theta(x^{(n)})$$

Evidence for all data points

The product of the areas underneath the green curves

$$p_\theta(X) = \prod_{n=1}^{N} p_\theta(x^{(n)})$$

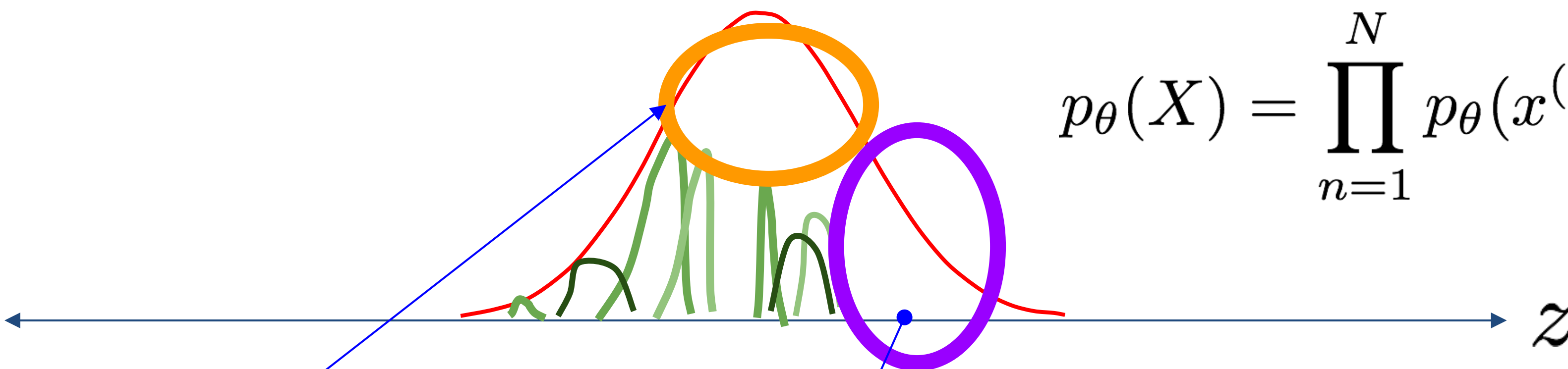


Maybe the x's don't even look that nice, when sampled with

Maximizing the evidence

The product of the areas underneath the green curves

$$p_\theta(X) = \prod_{n=1}^{N} p_\theta(x^{(n)})$$



By changing θ we can make the evidence for these data points bigger...

These z's don't generate images like the ones in the data set...

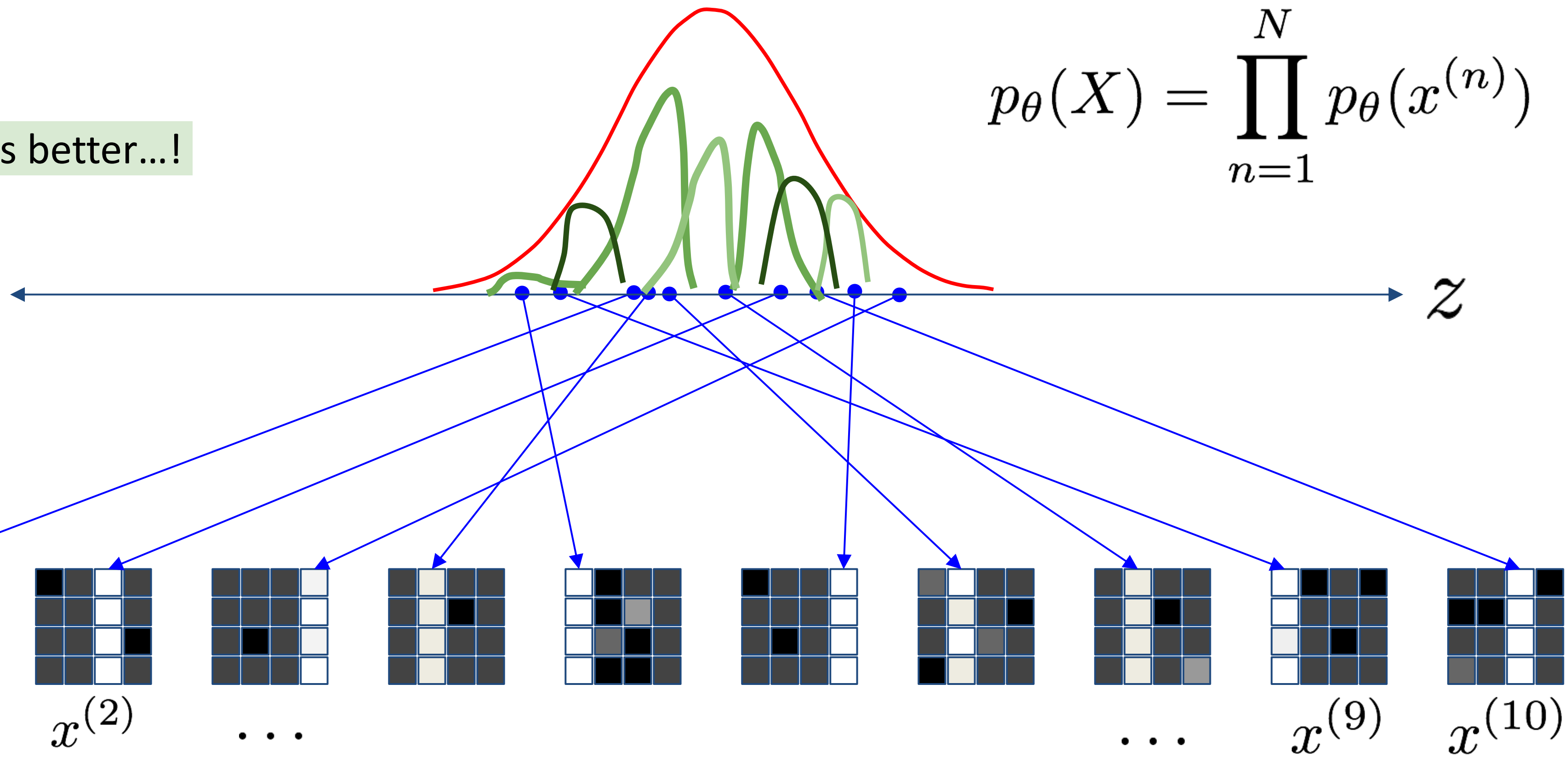
(With this θ, the prior doesn't capture the data manifold well)

Maximizing the evidence

The product of the areas underneath the green curves

$$p_\theta(X) = \prod_{n=1}^{N} p_\theta(x^{(n)})$$

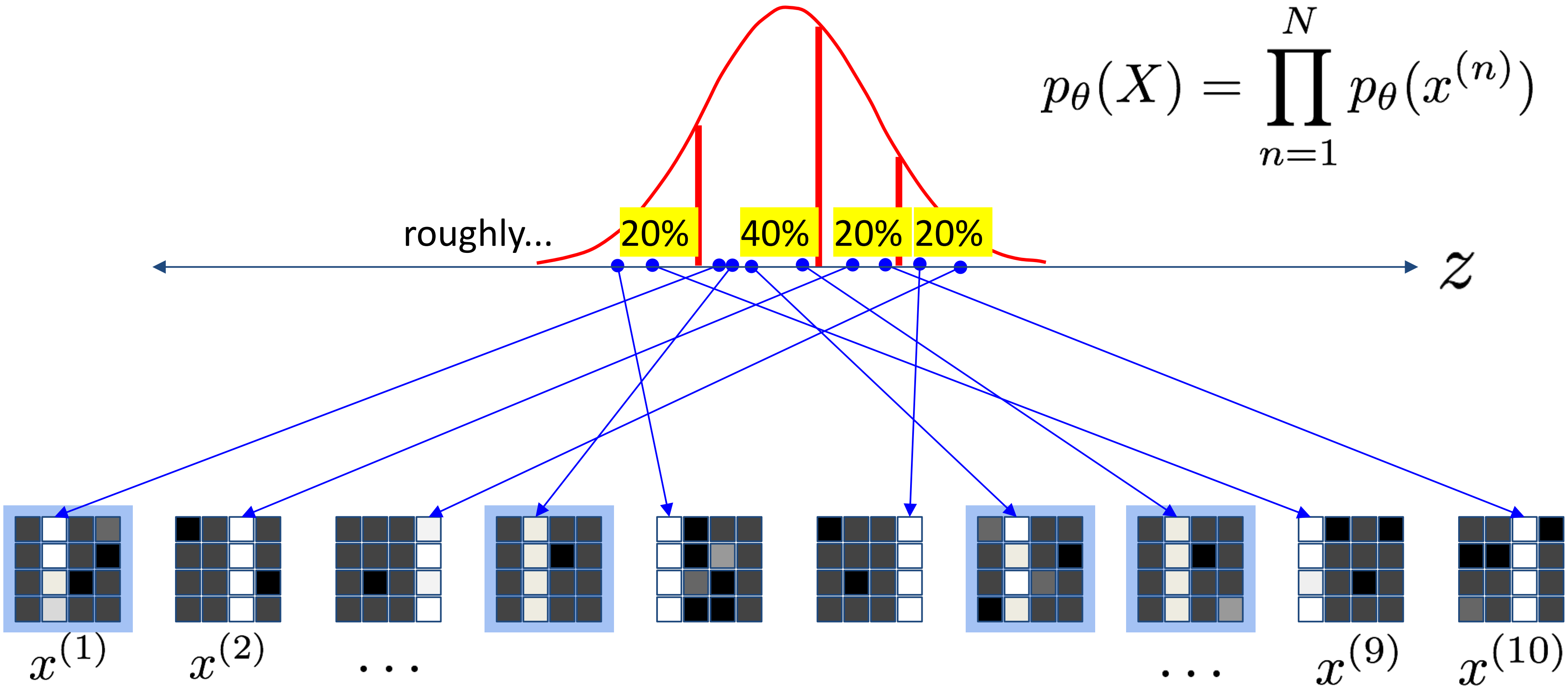
That's better…!



For the sharp-sighted

The product of the areas underneath the green curves

$$p_\theta(X) = \prod_{n=1}^{N} p_\theta(x^{(n)})$$



Generation and Learning

Generation:

$$p(z)$$
$$p_\Theta(x|z)$$

Training by max log-likelihood

$$\arg\max_\Theta \log p_\Theta(x)$$

But

$$p_\Theta(x) = \int dz\, p_\Theta(x|z)p(z)$$

Approximate likelihood optimization

Our approach:

- Lower bound $\log p_\Theta(x)$
- Push the lower-bound up…
  … hoping to increase $\log p_\Theta(x)$

Exercise

Jensen's inequality

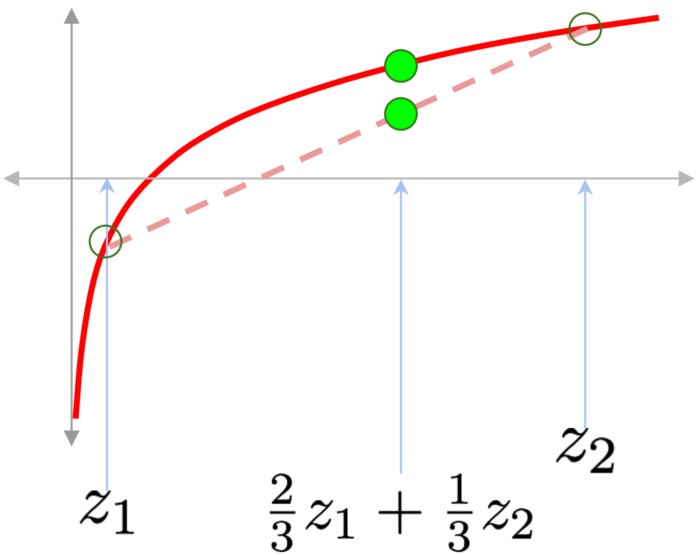
Draw log(...) as a function, convince yourself that

$$\log\left(\frac{2}{3}z_1 + \frac{1}{3}z_2\right) \geq \frac{2}{3}\log z_1 + \frac{1}{3}\log z_2$$

is true for any (nonnegative) setting of z_1 and z_2.

Jensen's inequality

$$\log\left(\tfrac{2}{3}z_1 + \tfrac{1}{3}z_2\right) \geq \tfrac{2}{3}\log(z_1) + \tfrac{1}{3}\log(z_2)$$



$$\log \int dz\, q(z) f(z) \geq \int dz q(z) \log f(z)$$

ELBO: A likelihood bound

$$\log p_\Theta(x) = \log \int dz\, p_\Theta(x,z) =$$

$$= \log \int dz\, q_\Phi(z|x) \frac{p_\Theta(x,z)}{q_\Phi(z|x)}$$

$$\geq \int dz q_\Phi(z|x) \log \frac{p_\Theta(x,z)}{q_\Phi(z|x)}$$

$$= \mathbb{E}_{q_\Phi(z|x)}\left[\frac{p_\Theta(x|z)p(z)}{q_\Phi(z|x)}\right]$$

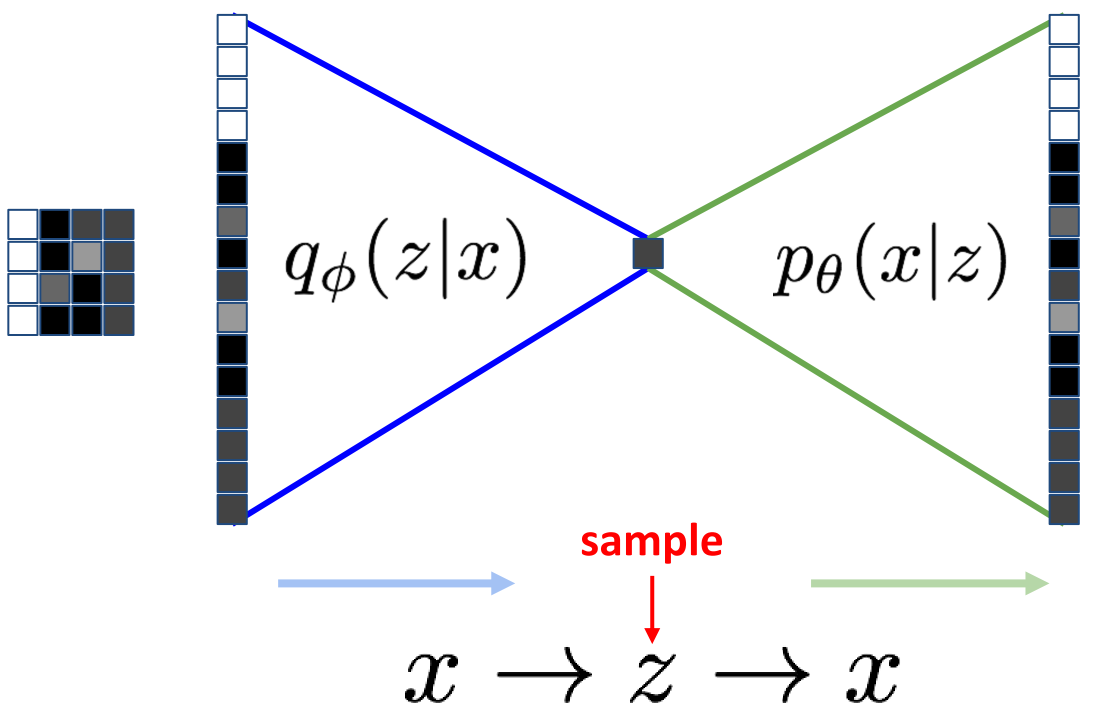
$$= \mathbb{E}_{q_\Phi(z|x)}[p_\Theta(x|z)] - \mathbb{E}_{q_\Phi(z|x)}\left[\frac{q_\Phi(z|x)}{p(z)}\right]$$

$$= \mathbb{E}_{q_\Phi(z|x)}[p_\Theta(x|z)] - KL(q_\Phi(z|x) \parallel p(z))$$

ELBO interpretation

$$\log p_\Theta(x) \geq \mathbb{E}_{q_\Phi(z|x)}[p_\Theta(x|z)] - KL(q_\Phi(z|x) \parallel p(z))$$

$\mathbb{E}_{q_\Phi(z|x)}[p_\Theta(x|z)]$: auto-encoding term!



ELBO optimization

$$\log p_\Theta(x) \geq \mathbb{E}_{q_\Phi(z|x)}[p_\Theta(x|z)] - KL(q_\Phi(z|x) \parallel p(z))$$

ELBO is a function of x, Θ, and Φ

What it means to maximize ELBO over Φ?

It can't change $\log p_\Theta(x)$...

It tries to make the bound tight!

Exercise

Recall Jensen's inequality:

$$\log \int dz\, q(z) f(z) \geq \int dz q(z) \log f(z)$$

When is it an **equality**?

When $f(z) = \mathrm{const}$

When is ELBO tight?

$$\log p_\Theta(x) = \log \int dz\, q_\Phi(z|x) \frac{p_\Theta(x,z)}{q_\Phi(z|x)}$$

$$\geq \int dz q_\Phi(z|x) \log \frac{p_\Theta(x,z)}{q_\Phi(z|x)} = ELBO$$

When $\frac{p_\Theta(x,z)}{q_\Phi(z|x)} = \text{const!}$

What does it mean?

$$\frac{p_\Theta(x,z)}{q_\Phi(z|x)} = \frac{p_\Theta(x|z)p(z)}{q_\Phi(z|x)} = \text{const} \Rightarrow p_\Theta(x|z) = q_\Phi(z|x)$$

ELBO is tight when $q_\Phi(z|x)$ does exact inference!

ELBO optimization

$$\log p_\Theta(x) \geq \mathbb{E}_{q_\Phi(z|x)}[p_\Theta(x|z)] - KL(q_\Phi(z|x) \parallel p(z))$$

ELBO is a function of x, Θ, and Φ

What it means to maximize ELBO over Θ?

Can only affect $\mathbb{E}_{q_\Phi(z|x)}[p_\Theta(x|z)]$!

Makes $p_\Theta(\mathrm{x}|\mathrm{z})$ generate back our x!
This affects $\log p_\Theta(x)$…
…making room for improving q!

ELBO optimization

$$\log p_\Theta(x) \geq \mathbb{E}_{q_\Phi(z|x)}[p_\Theta(x|z)] - KL(q_\Phi(z|x) \parallel p(z))$$

Change Φ to maximize the bound, making $q_\Phi(z|x) \approx p_\Theta(z|x)$

Similar to E step

Change Θ to (if bound sufficiently tight) improve $\log p_\Theta(x)$

Similar to M step

But we tune Φ and Θ at the same time!

ELBO interpretation

ELBO, or evidence lower bound:

$$\log p(x) \geq \mathbb{E}_{z\sim q_\Phi(z|x)}[\log p_\Theta(x|z)] - KL(q_\Phi(z|x) \parallel p(z))$$

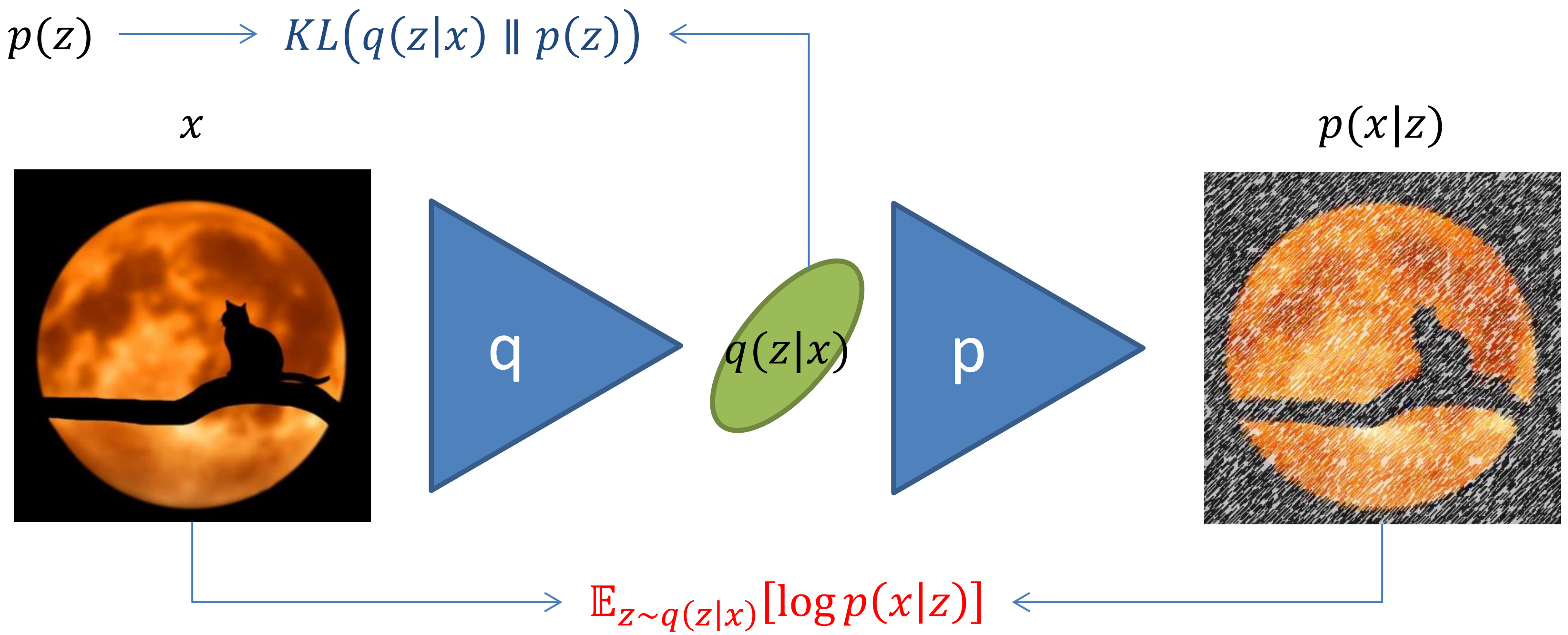
where:

$\mathbb{E}_{z\sim q_\Phi(z|x)}[\log p_\Theta(x|z)]$ reconstruction quality:
how many nats we need to reconstruct x,
when someone gives us $q(z|x)$

$KL(q_\Phi(z|x) \parallel p(z))$ code transmission cost:
how many nats we transmit about x in $q_\Phi(z|x)$ rather than $p(z)$

Interpretation: do well at reconstructing x, limiting the amount of information about x encoded in z.

The Variational Autoencoder



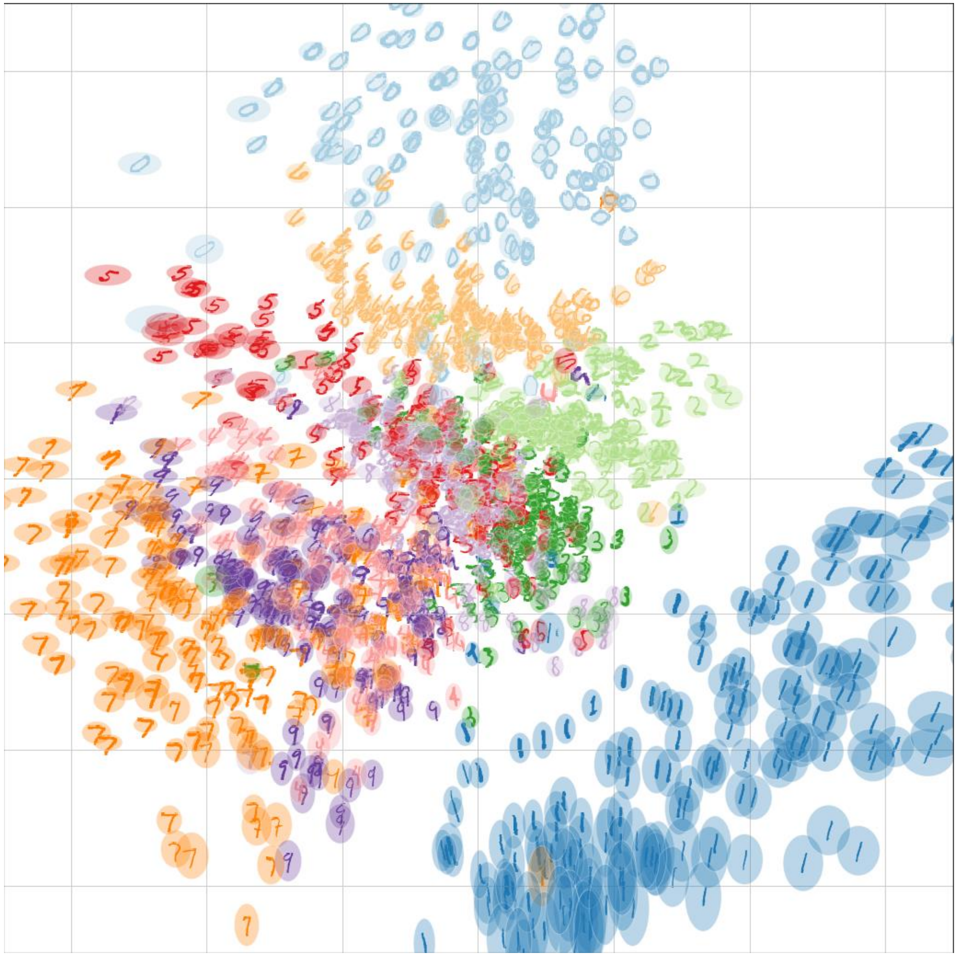
An input x is put through the q network to obtain a distribution over latent code z, $q(z|x)$.

Samples $z_1, \ldots, z_k$ are drawn from $q(z|x)$. They k reconstructions $p(x|z_k)$ are computed using the network p.

VAE is an Information Bottleneck

Each sample is represented as a Gaussian

This discards information (latent representation has low precision)



How to evaluate a VAE

Compute:

$$\log p(x) \geq \mathbb{E}_{z \sim q_\Phi(z|x)}[\log p_\Theta(x|z)] - KL(q_\Phi(z|x) \parallel p(z))$$

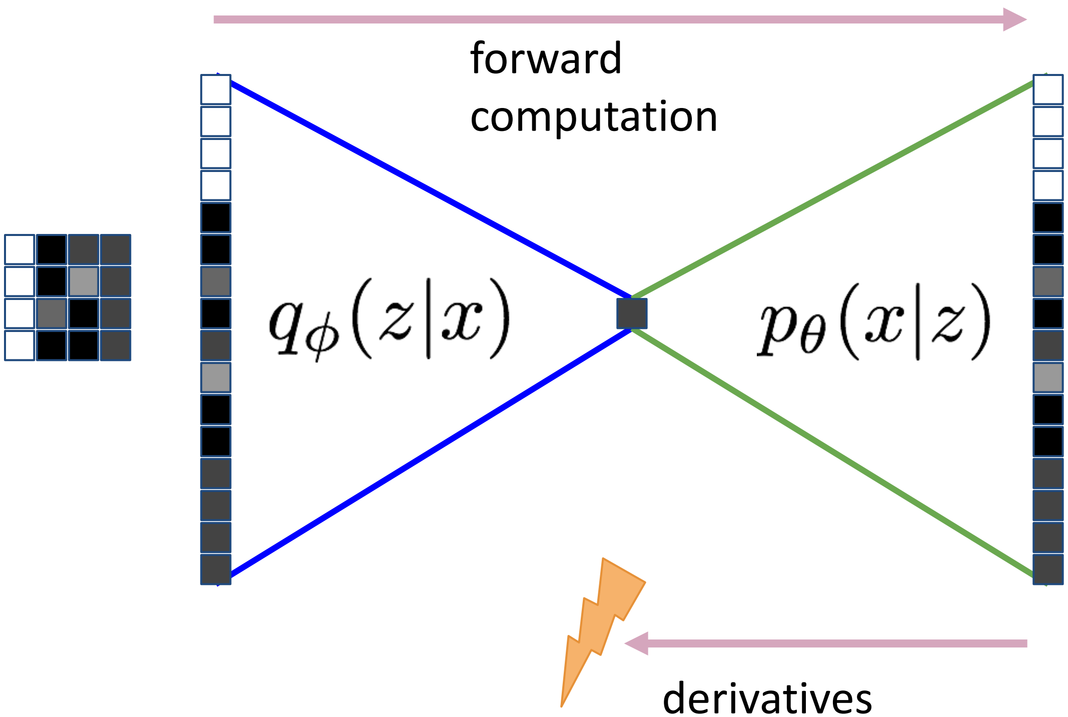
$KL(q_\Phi(z|x) \parallel p(z))$ has closed form for simple $q_\Phi(z|x)$

$\mathbb{E}_{z \sim q_\Phi(z|x)}[\log p_\Theta(x|z)]$ can be approximated:

$$\mathbb{E}_{z \sim q_\Phi(z|x)}[\log p_\Theta(x|z)] \approx \sum_i \log p_\Theta(x|z_i)$$

Where z_i drawn from $q_\Phi(z|x)$

How to train a VAE?



- Forward computation involves drawing samples
- Can't backprop ☹

Reparameterization exercise

Assume that $q_\Phi(z|x) = \mathcal{N}(\mu_z, \sigma_z)$.

Exercise:
you can sample from $\mathcal{N}(0,1)$

Q: how to draw samples from $\mathcal{N}(\mu_z, \sigma_z)$

A:

$$\epsilon_i \sim \mathcal{N}(0,1)$$
$$z_i = \mu_z + \sigma_z \epsilon$$

Reparametrization to the rescue

Assume that $q_\Phi(z|x) = \mathcal{N}(\mu_z, \sigma_z)$.

Then:

$$\mathbb{E}_{z\sim q_\Phi(z|x)}[\log p_\Theta(x|z)]$$
$$= \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}[\log p_\Theta(x|\mu_z + \sigma_z\epsilon)]$$

ϵ is drawn from a fixed distribution.
With ϵ given, the computaiton graph is deterministic -> we can backprop!

Outline

- Why unsupervised learning
- Autoregressive models
  - Intuitions
  - Dilated convolutions
    RNNs (Intermezzo: LSTM training dynamics)
    Transformers
  - Beyond sequences, summary
- Latent variable models
  - VAE
  - **Normalizing flows**
- Autoregressive + latent variable: why and how?

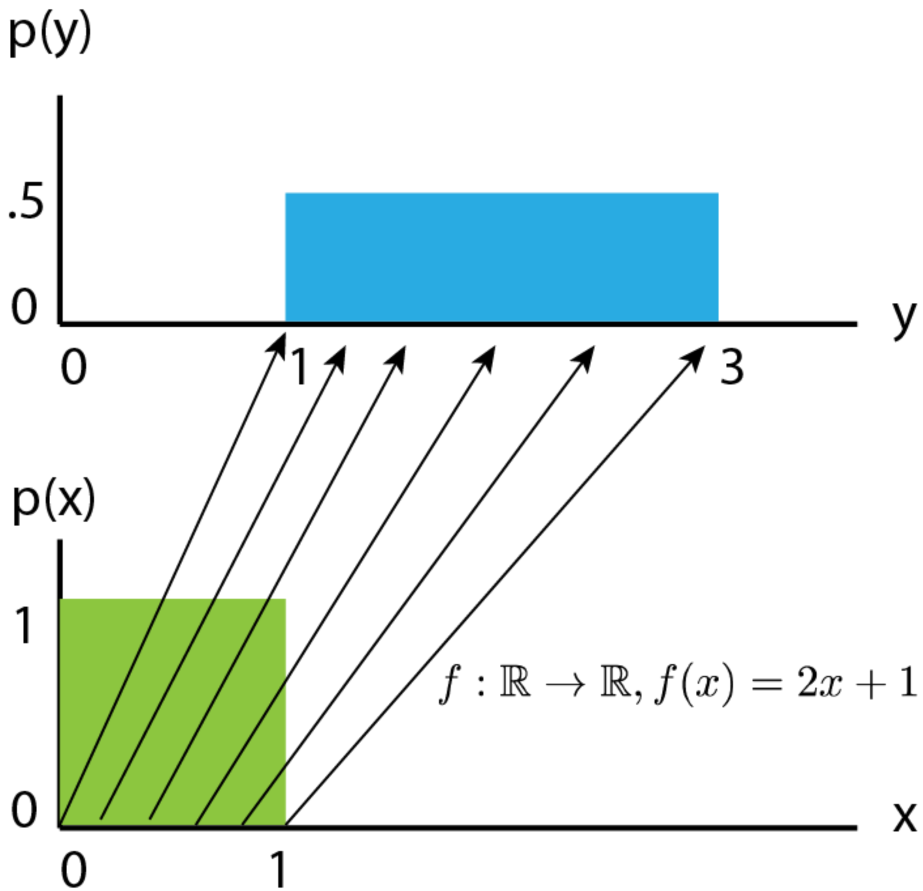
Exercise

$$p(x) = \text{uniform}(0,1) = \begin{cases} 1 \text{ for } x \in (0,1) \\ 0 \text{ otherwise} \end{cases}$$

$$y = 2 * x + 1$$

$$p(y) = ?$$

Transforming distributions



y also has a uniform distribution!

$$p(y) = \text{uniform}(0,2) = \begin{cases} 1/2 \text{ for } y \in (1,3) \\ 0 \text{ otherwise} \end{cases}$$

E Jang https://blog.evjang.com/2018/01/nf1.html

The Jacobian: stretching space

Let $y = f(x)$

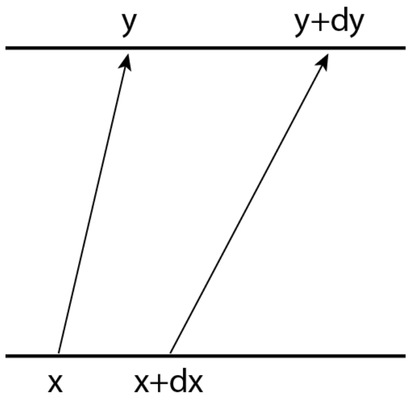
Take a range $(x, x + \Delta x)$

Question:

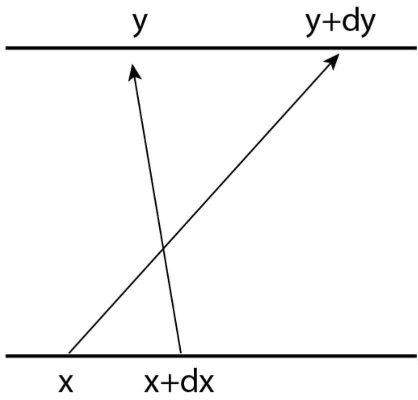
How long is this range?

$$\Delta x$$

$$\frac{dy}{dx} > 0$$



$$\frac{dy}{dx} < 0$$



Question:

How long is $(f(x), f(x + \Delta x))$?

$$f(x + \Delta x) \approx f(x) + \frac{\partial f}{\partial x} \Delta x$$

$$\left| \frac{\partial f}{\partial x} \Delta x \right|$$

https://blog.evjang.com/2018/01/nf1.html

Change of variables

$y = f(x)$, f is a bijection

$$p_x(x) = p_y\big(f(x)\big)\left|\det\frac{\partial f(x)}{\partial x}\right|$$

f transforms $x \pm \frac{\Delta x}{2}$ to y $\pm \frac{\Delta y}{2}$

The space is stretched by

$$\frac{\partial f(x)}{\partial x} \approx \frac{\Delta y}{\Delta x}$$

Idea

Start with $z \sim \mathcal{N}(0,1)$

Then $x = f^{-1}(z)$ (equivalently $\mathrm{z} = f(x)$)

$$p_x(x) = p_z\big(f(x)\big)\left|\det \frac{\partial f(x)}{x}\right|$$

Tractable when:

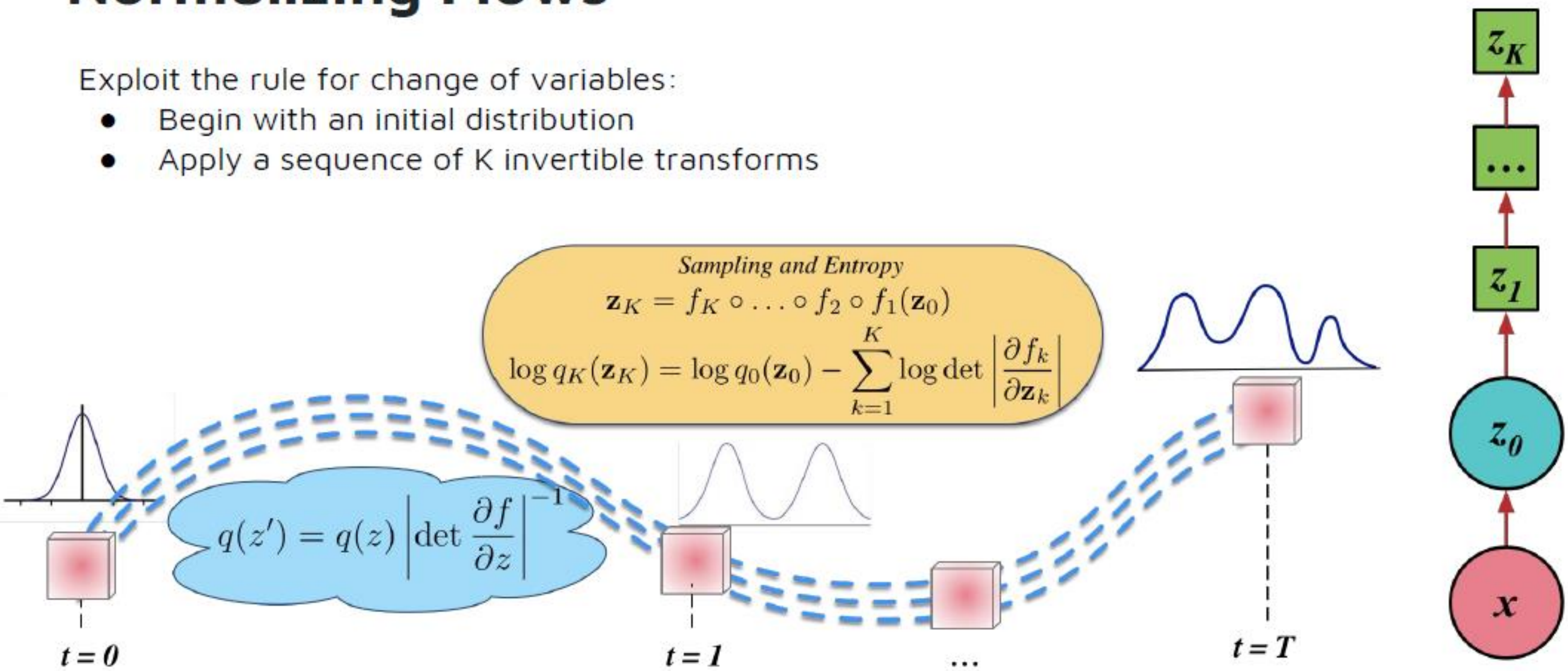
- f is easy to exactly invert
  f and f^{-1} form and exact auto-encoder!
- $\det \frac{\partial f(x)}{x}$ is easy to compute

=> We need special f!

Normalizing Flows

Exploit the rule for change of variables:

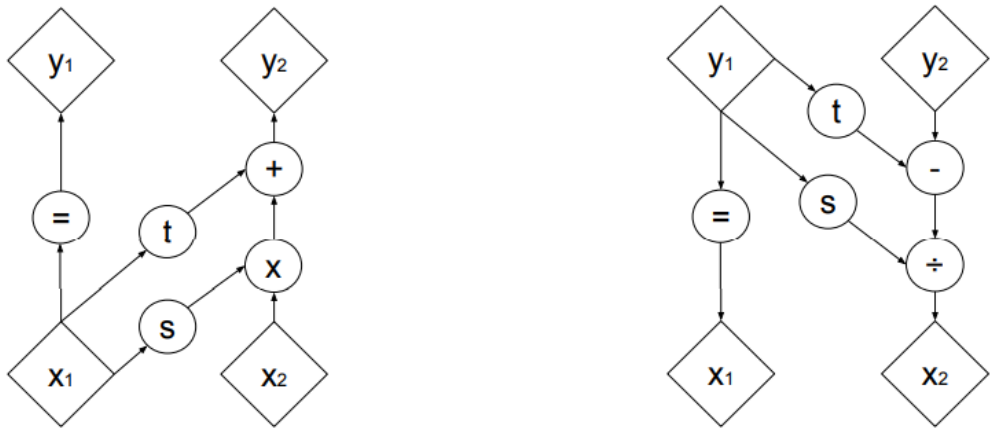
- Begin with an initial distribution
- Apply a sequence of K invertible transforms



Distribution flows through a sequence of invertible transforms

Rezende and Mohamed, 2015

A special form of f



(a) Forward propagation (b) Inverse propagation

$$x_1, x_2 = \mathrm{split}(x)$$
$$y_1 = x_1$$
$$y_2 = x_2 s(x_1) + t(x_1)$$

Trivial to invert!

$\frac{\partial y}{\partial x}$ is diagonal, determinant is easy!

L. Dinh et al "NICE", L. Dinh et al "Real NVP"

Normalizing flows in action

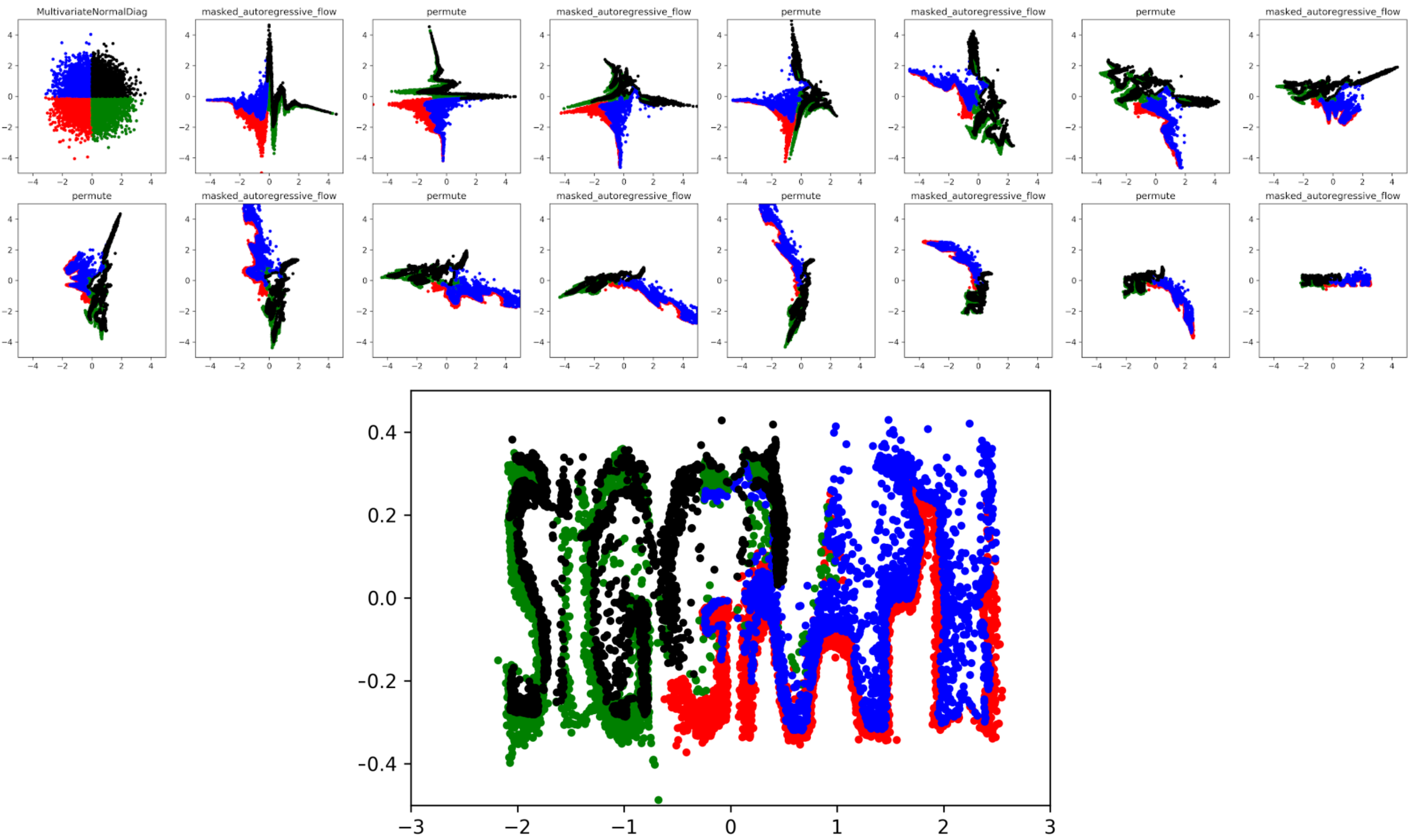
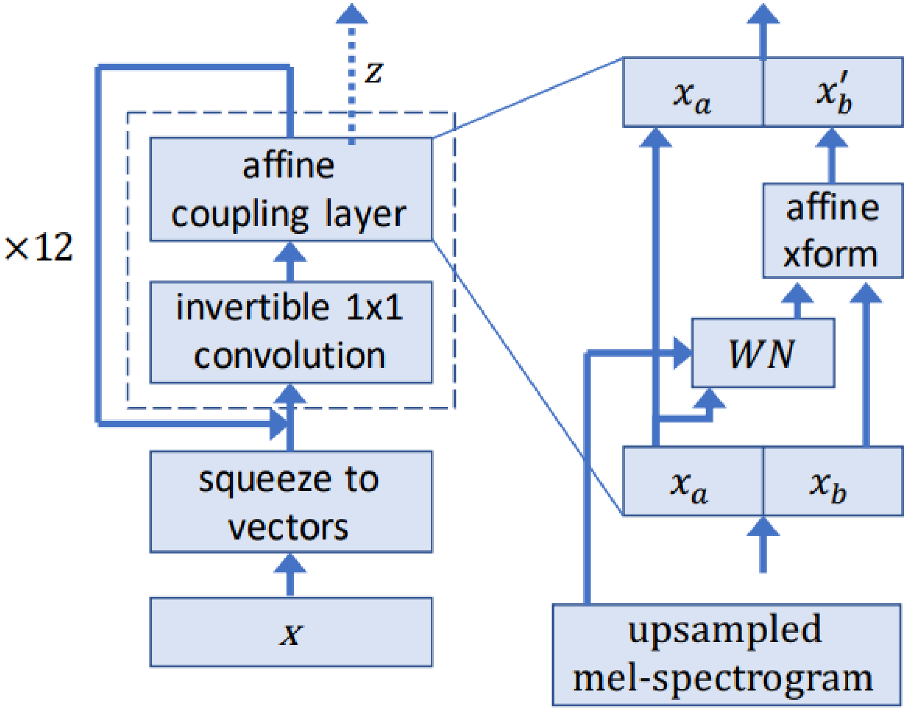


Image credit Eric Jang, https://blog.evjang.com/2018/01/nf2.html

Normalizing flows in action



Kingma et al, “GLOW”

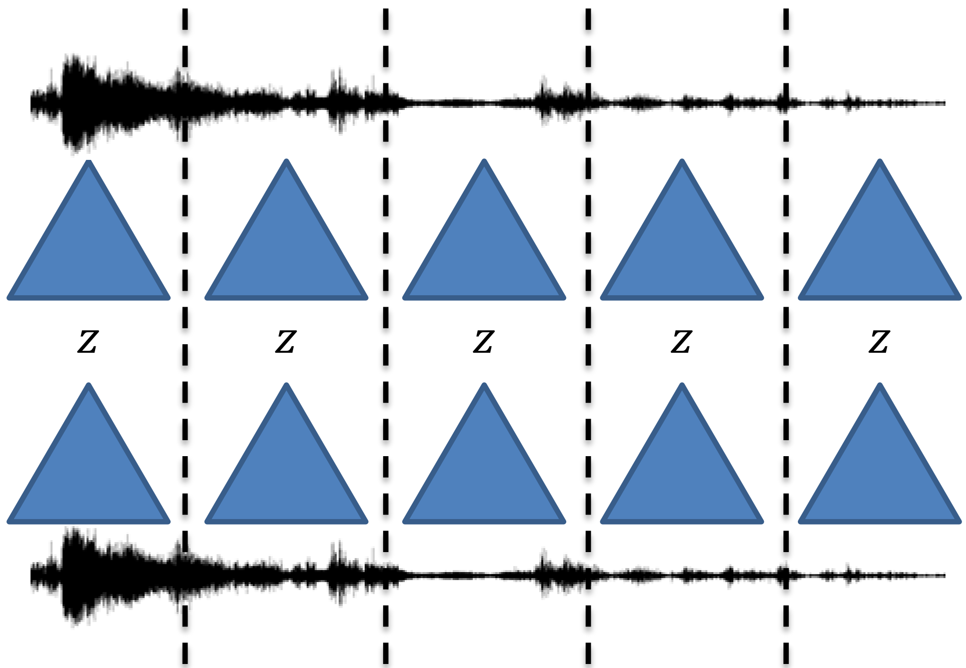
Prenger et al, “WAVEGLOW”

Outline

- Why unsupervised learning
- Autoregressive models
  - Intuitions
  - Dilated convolutions
    RNNs (Intermezzo: LSTM training dynamics)
    Transformers
  - Beyond sequences, summary
- Latent variable models
  - VAE
  - Normalizing flows
- **Autoregressive + latent variable: why and how?**

VAEs and sequential data

To encode a long sequence, we apply the VAE to chunks:

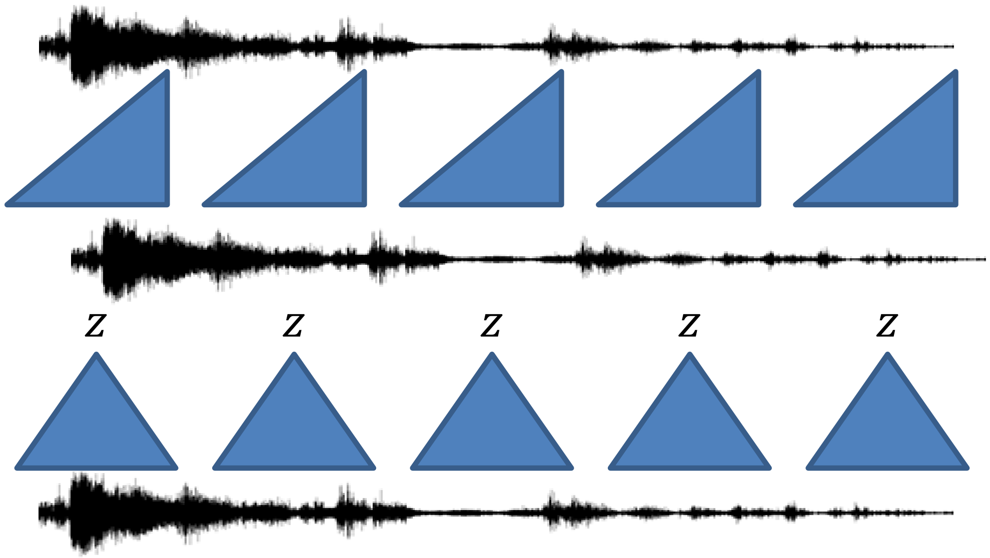


But neighboring chunks are similar!

We are encoding the same information in many zs!

We are wasting capacity!

WaveNet + VAE



A WaveNet reconstructs the waveform using the information from the past

Latent representations are extracted at regular inervals.

The WaveNet uses information from:

1. The past recording
2. The latent vectors z
3. Other conditioning, e.g. about speaker

The encoder transmits in zs only the information that is missing from the past recording .

The whole system is a very low bitrate codec

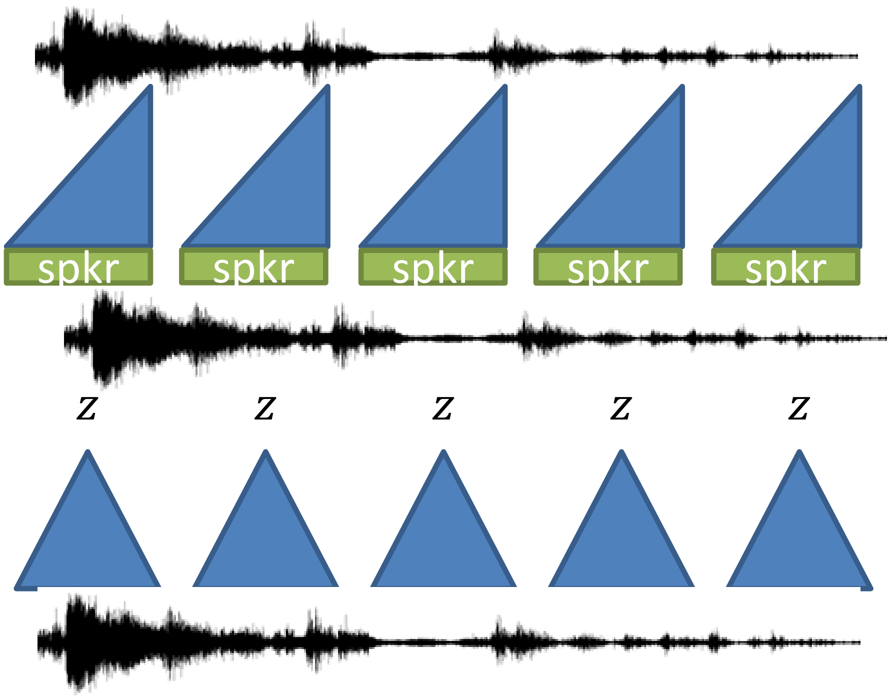
(roughly 0.7kbits/sec, the waveform is 16k Hz* 8bit=128kbit/sec)

van den Oord et al. Neural discrete representation learning

VAE + autoregressive models: latent collapse danger

- Purely Autoregressive models: SOTA log-likelihoods
- Conditioning on latents:
  information passed through bottleneck
  lower reconstruction x-entropy
- In standard VAE model actively tries to
  - reduce information in the latents
  - maxmally use autoregressive information
  => Collapse: latents are not used!
- Solution: stop optimizing KL term
  (free bits), make it a hyperparam (VQVAE)

Model description



WaveNet decoder conditioned on:

- latents extracted at 24Hz-50Hz
- speaker

3 bottleneck evaluated:

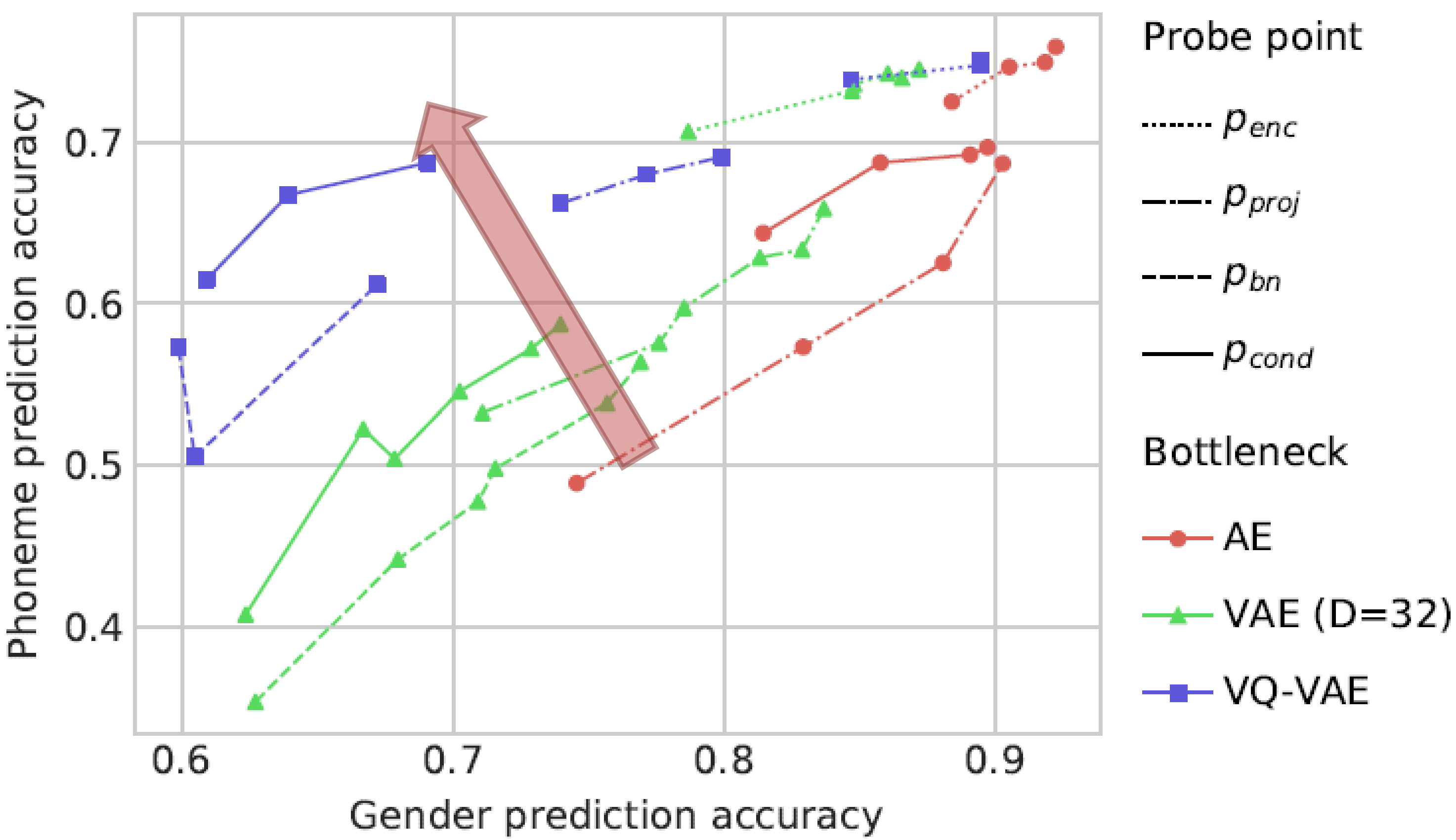
- Dimensionality reduction, max 32 bits/dim
- VAE, $KL\big(q(z|x) \parallel \mathcal{N}(0,1)\big)$ nats (bits)
- VQVAE with K protos: $\log_2 K$ bits

Input:

Waveforms, Mel Filterbanks, MFCCs

Hope: speaker separated form content.

Phonemes vs Gender tradeoff

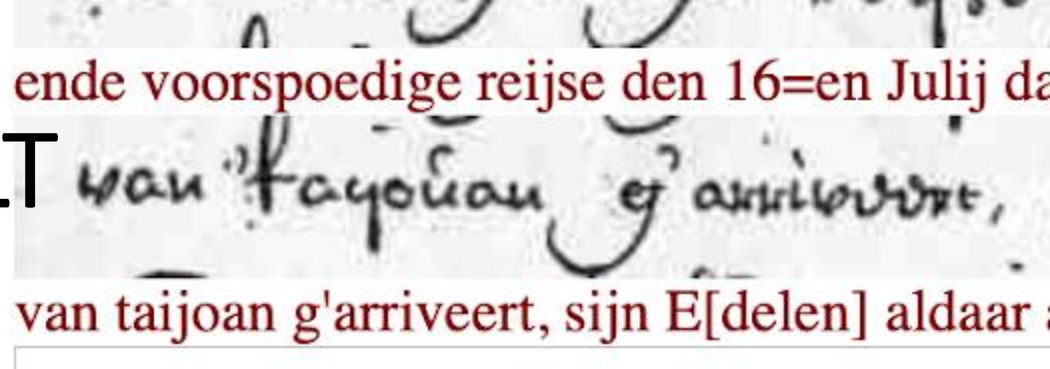


https://arxiv.org/abs/1901.08810

Summary

Model	Evaluate $p(x)$	Sample $p(x)$	Extract latents	Control info in latents
Autoregressive	Exact & Cheap	Exact & Expensive	Impossible	N/A
Latent var, VAE	Lower Bound	Exact & Cheap	Easy	No
Latent var, Norm. Flow	Exact & Cheap	Exact & Cheap	Easy	No
Autoregressive cond on latents	Lower Bound	Exact & Expensive	Easy	Yes

Our topic at JSALT



We will these ideas during JSALT's topic "Distant supervision for representation learning":

- Work on speech and handwriting
- Explore ways of integrating metadata and unlabeled data to control latent representations
- Focus on downstream supervised OCR and ASR tasks under low data conditions

Thank you!

- Questions?

References – other tutorials

- https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf

Backup

ELBO: A lower bound on $\log p(x)$

Let $q(z|x)$ be any distribution. We can show that

$$
\begin{aligned}
&\log p(x) = \\
&= KL\big(q(z|x) \parallel p(z|x)\big) + \mathbb{E}_{z\sim q(z|x)}\left[\log\left(\frac{p(z|x)}{q(z|x)}p(x)\right)\right] \\
&\geq \mathbb{E}_{z\sim q(z|x)}\left[\log\left(\frac{p(z|x)}{q(z|x)}p(x)\right)\right] \\
&= \mathbb{E}_{z\sim q(z|x)}[\log p(x|z)] - KL\big(q(z|x) \parallel p(z)\big)
\end{aligned}
$$

The bound is tight for $p(z|x) = q(z|x)$.

ELBO Derivation pt. 1

$$
\begin{aligned}
KL(q_\phi(z|x)||p_\theta(z|x)) &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[-\log\frac{p_\theta(z|x)}{q_\phi(z|x)}\right] = \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[-\log\frac{p_\theta(z|x)p_\theta(x)}{q_\phi(z|x)p_\theta(x)}\right] = \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[-\log\frac{p_\theta(z|x)p_\theta(x)}{q_\phi(z|x)}\right] + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x)\right] = \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[-\log\frac{p_\theta(x,z)}{q_\phi(z|x)}\right] + \log p_\theta(x)
\end{aligned}
$$

$$
\log p_\theta(x) = KL(q_\phi(z|x)||p_\theta(z|x)) + \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log\frac{p_\theta(x,z)}{q_\phi(z|x)}\right]
$$

ELBO derivation pt. 2

$$
\begin{aligned}
\log p_\theta(x) &\geq \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] = \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log \frac{p_\theta(x|z)p_\theta(z)}{q_\phi(z|x)}\right] = \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - \mathbb{E}_{z\sim q_\phi(z|x)}\left[-\log \frac{p_\theta(z)}{q_\phi(z|x)}\right] \\
&= \mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right] - KL(q_\phi(z|x)||p_\theta(z))
\end{aligned}
$$