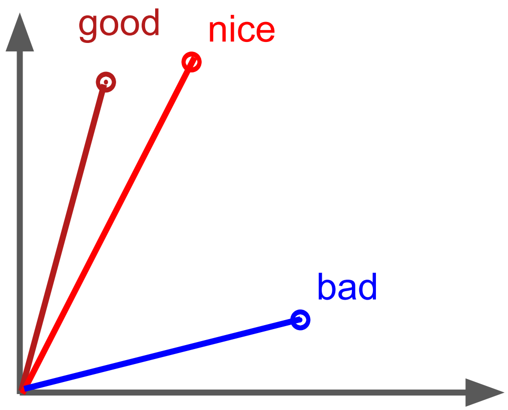
CS 4782 - Midterm Review

Snehal, Adhitya, Sean, Žiga, Lucas

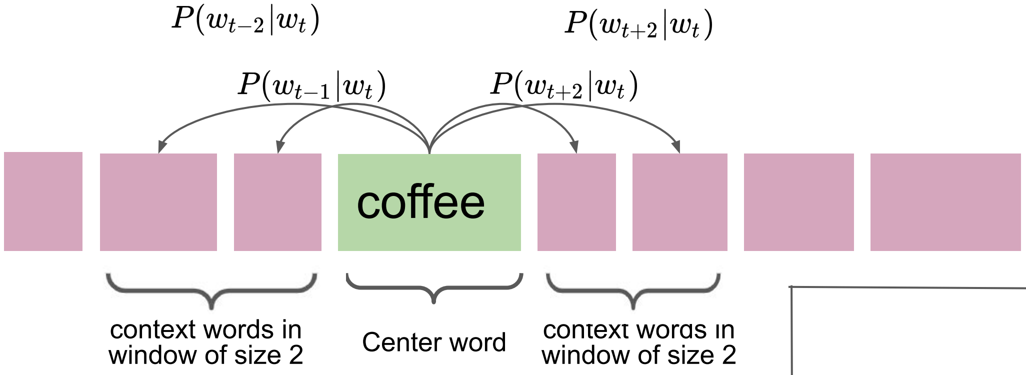
Word Embeddings

Motivated by semantic similarity

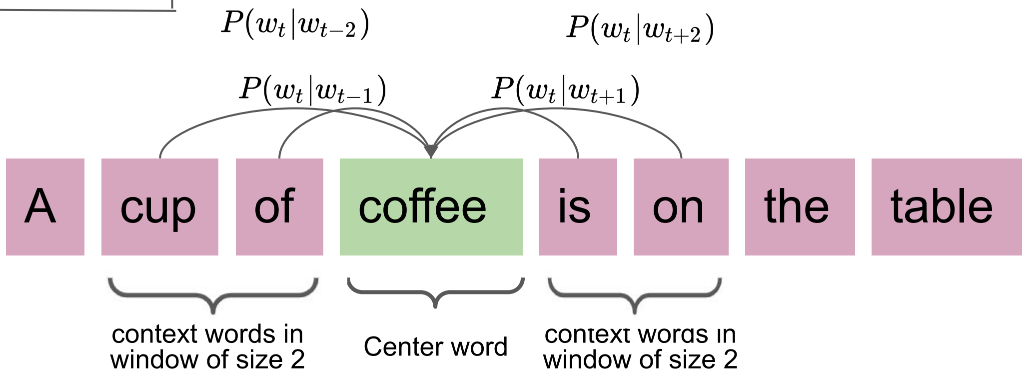


Word2Vec

Skipgram - Predict context from target



Continuous Bag of Words (CBOW) - predict target from context

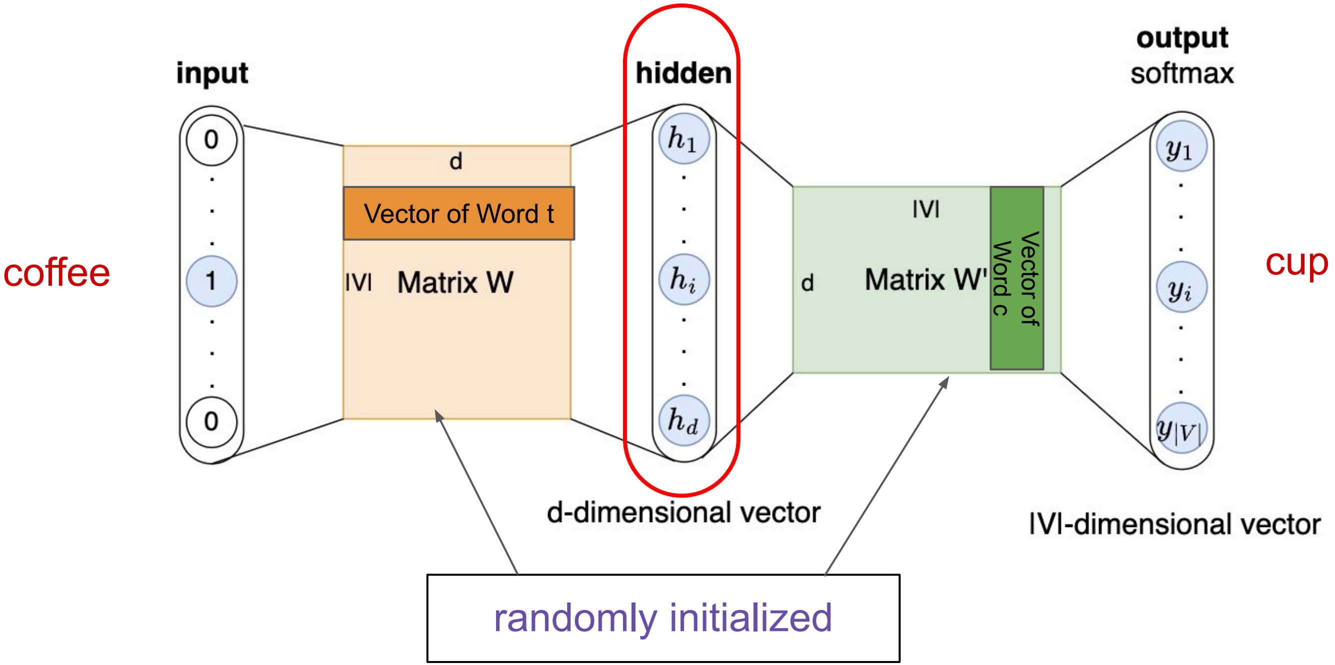


Training Data

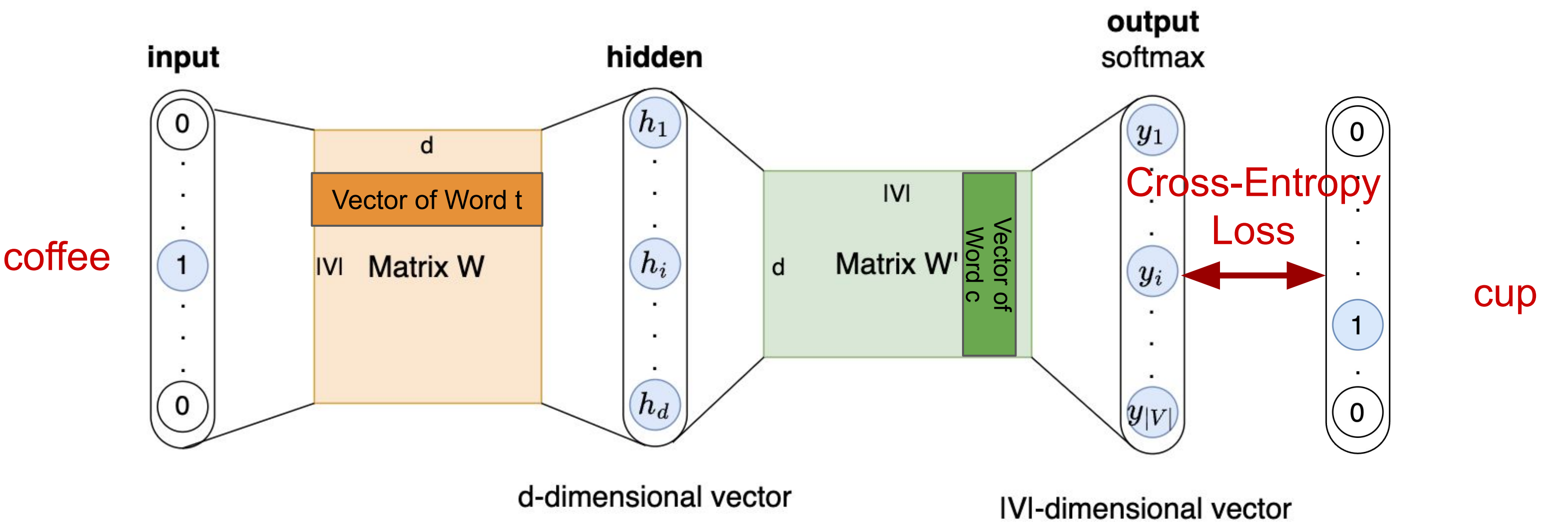
One-hot encoded vectors of all the words in your vocabulary

SkipGram

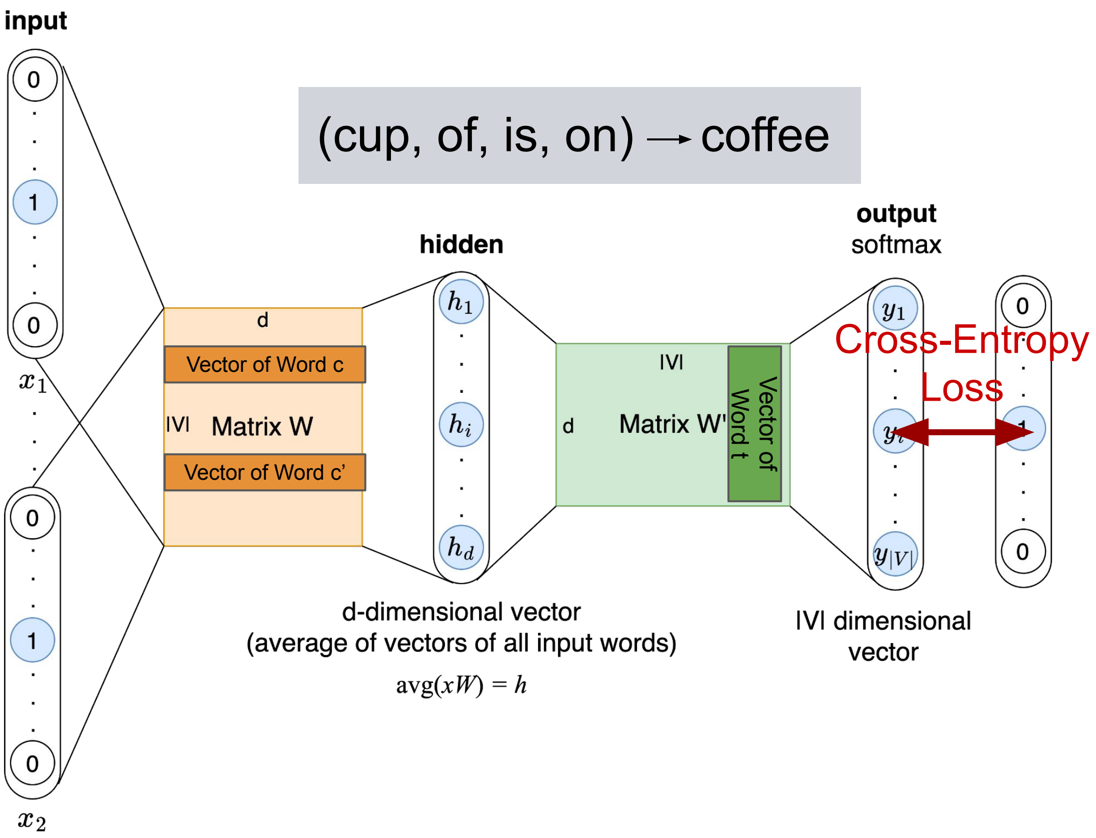
coffee → cup



Calculate Loss and BackPropagate till Minima



CBOW



Calculate Loss and BackPropagate till Minima

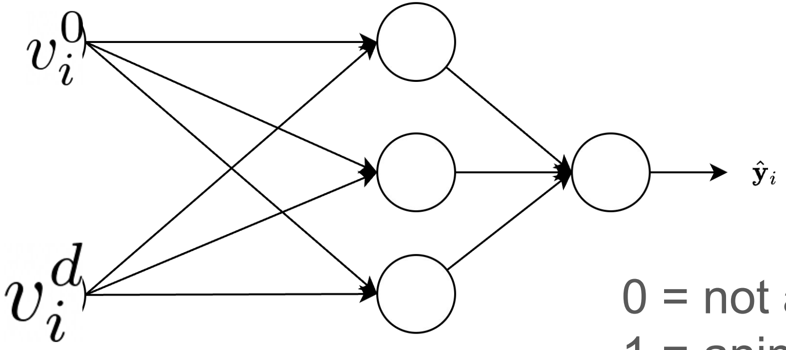
Transitioning to RNNs

Let's simplify!

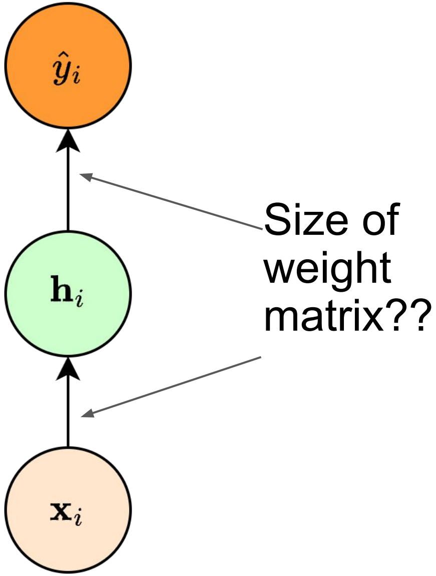
What if we have a single word and a single output?

cat	0.6	0.9	0.1	0.4	-0.7	-0.3	-0.2

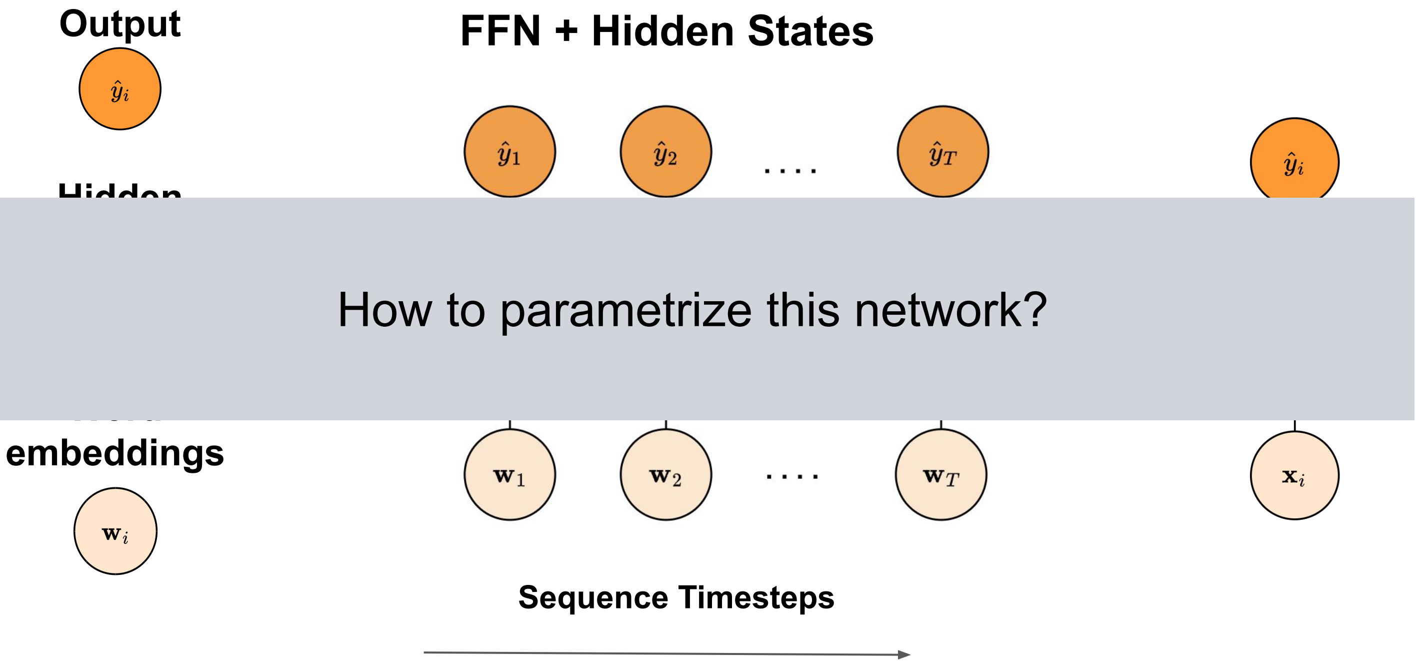
$\mathbf{v}_i \in \mathbb{R}^d$



0 = not animal
1 = animal

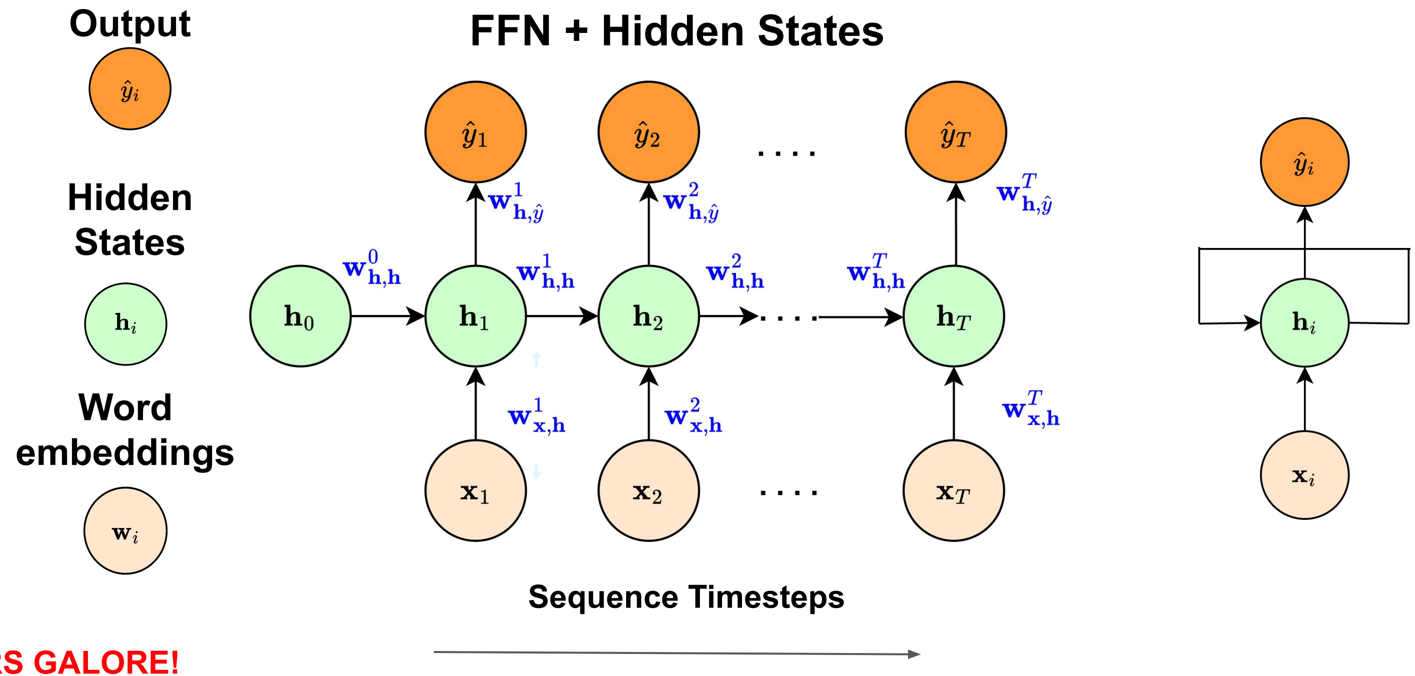


Recurrent neural network (RNN)



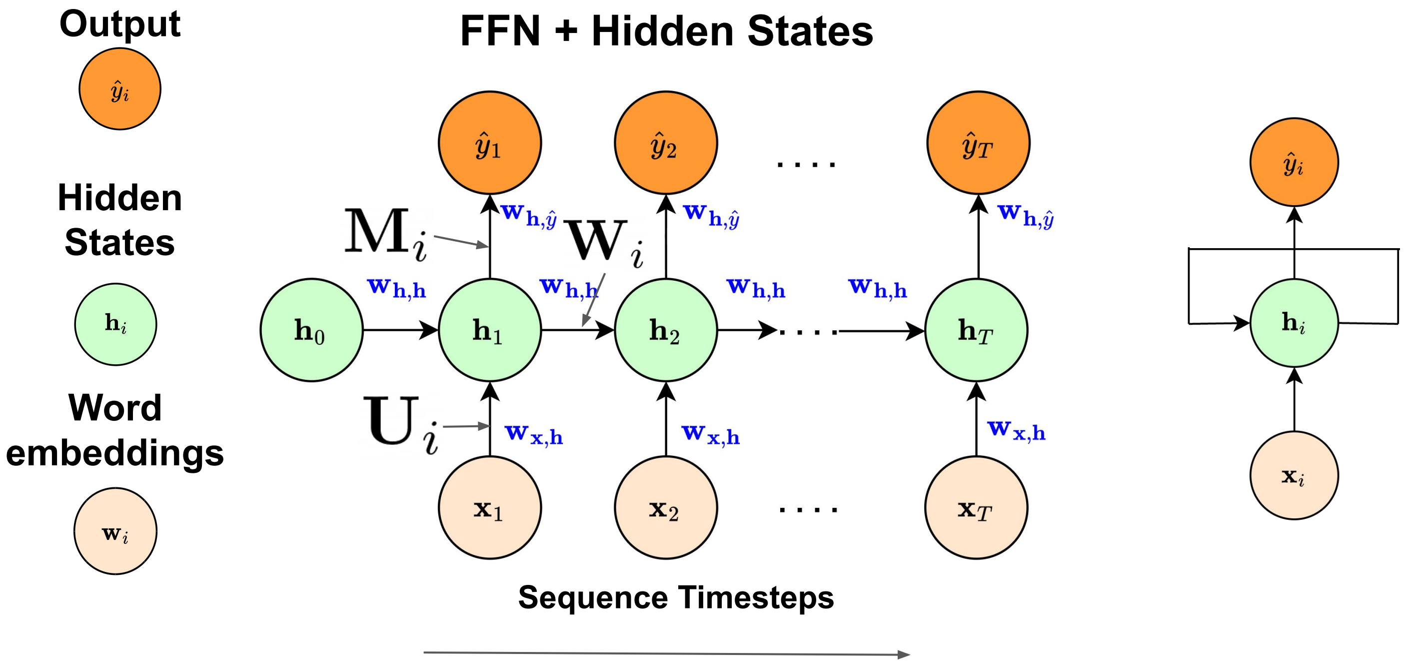
How to parametrize this network?

Maybe we add a weight matrix between every state??



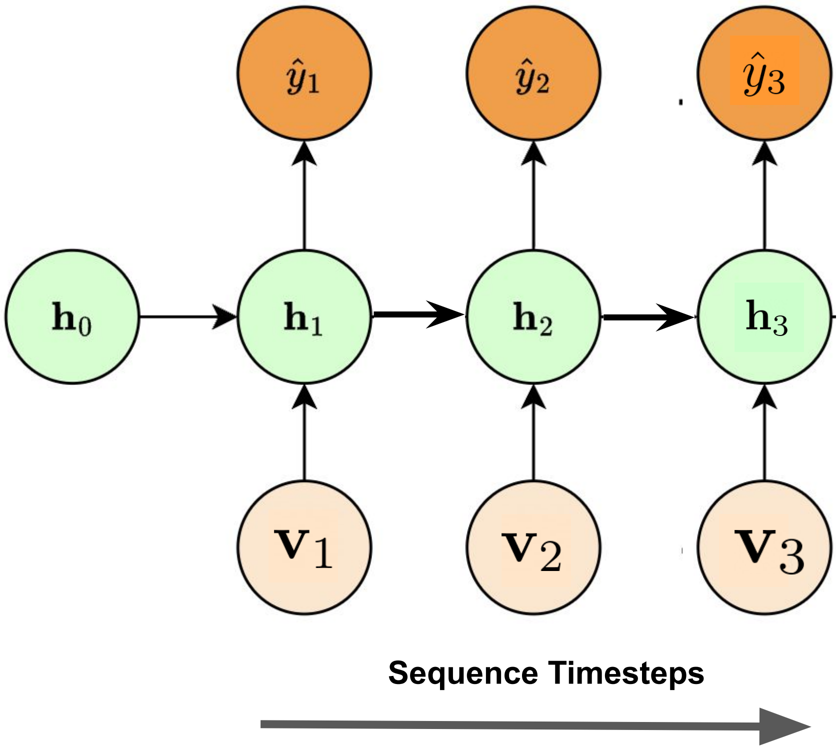
PARAMETERS GALORE!

Solution: Reuse same weight matrices - *wherever possible*



Recurrent neural network (RNN)

Use the same parameters across different timesteps.



Output

$$\hat{y}_i = \mathbf{M}\ \mathbf{h}_i$$

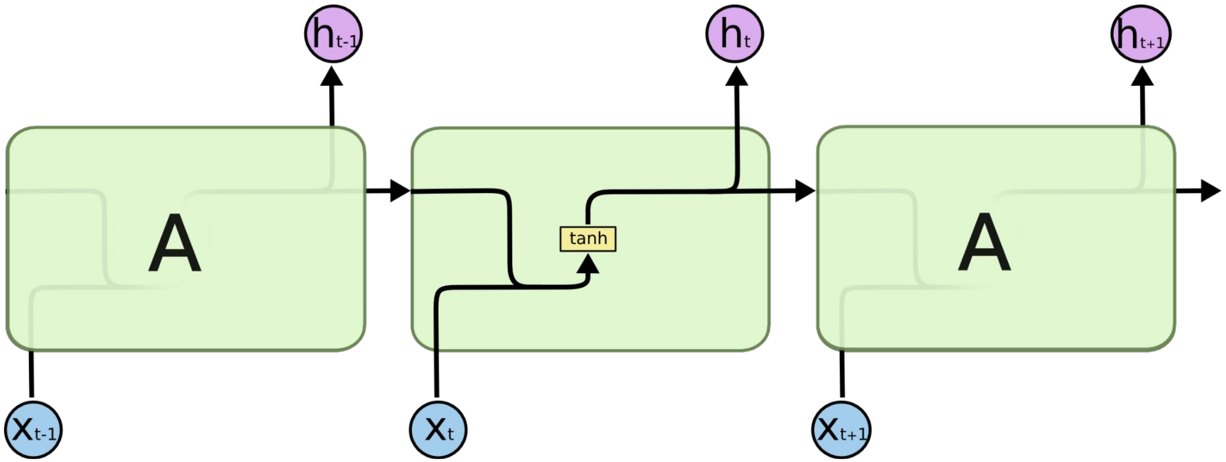
Hidden State

$$\mathbf{h}_i = \sigma(\mathbf{U}\ \mathbf{v}_i + \mathbf{W}\ \mathbf{h}_{i-1})$$

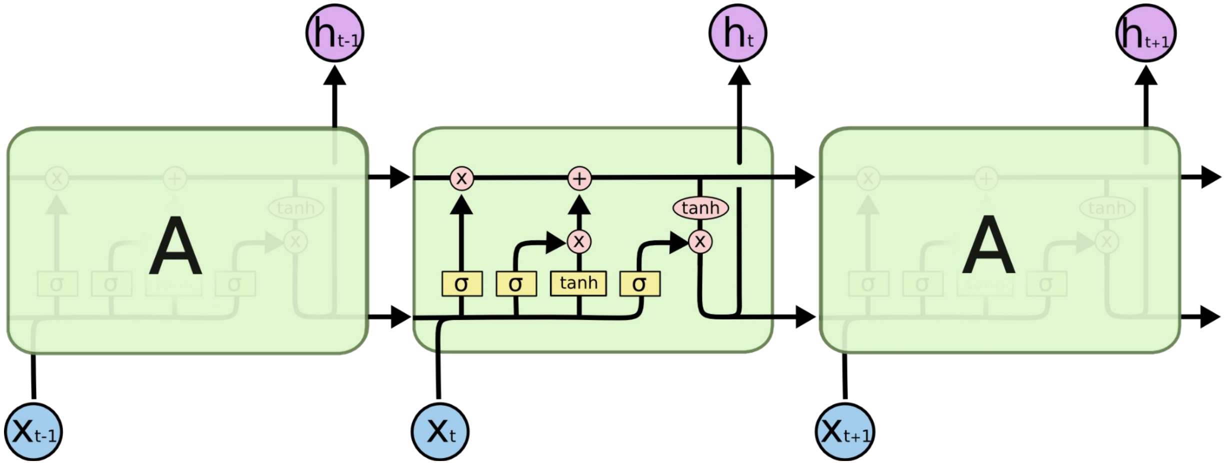
Problems

- **Loooooooong** Context Issues
- Sequential - slow

RNN



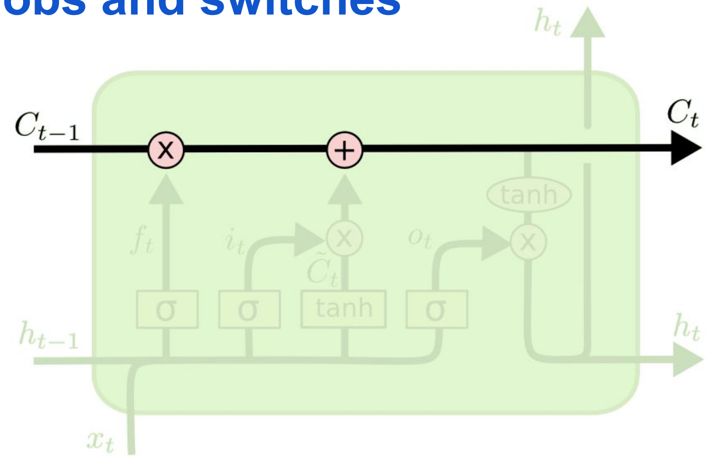
Long-short Term Memory (LSTM)

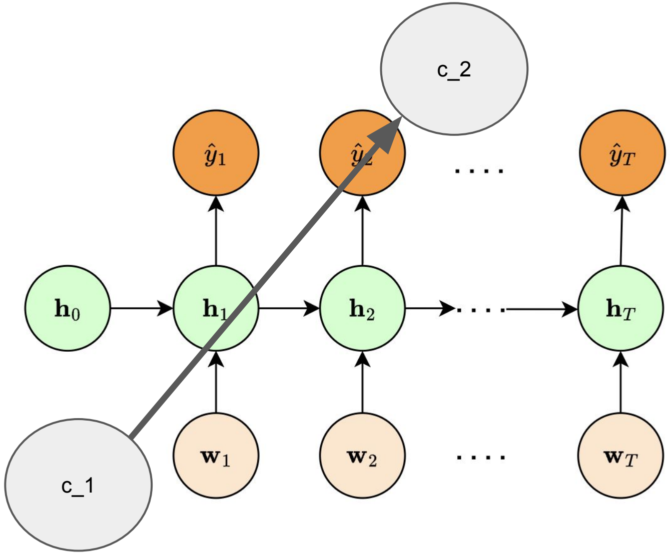


Long-short Term Memory (LSTM)

- Main idea: add a “cell” state that allows information to flow easily
  - Similar to residual connections
  - No repeated matrix multiplications!

Control flow by adding knobs and switches





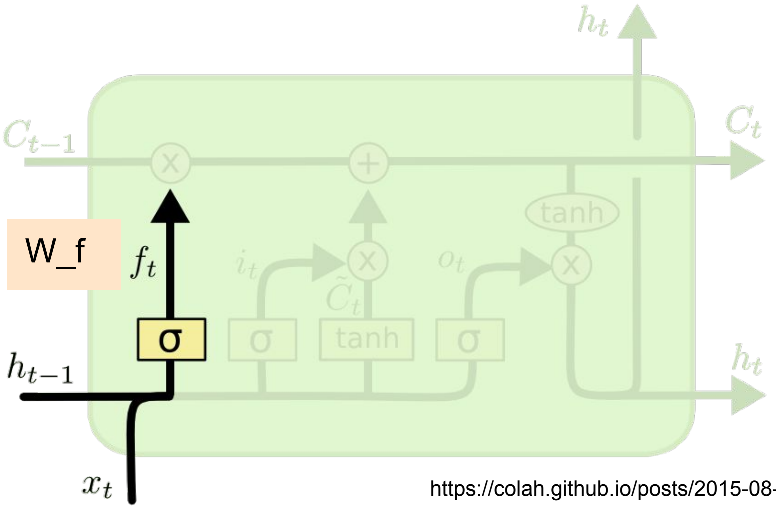
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs- Forget Gate

- Forget gate- function of current input and previous hidden state
- Controls what should be remembered in the cell state

multiply

Think of it like blocking certain dimensions and letting others flow

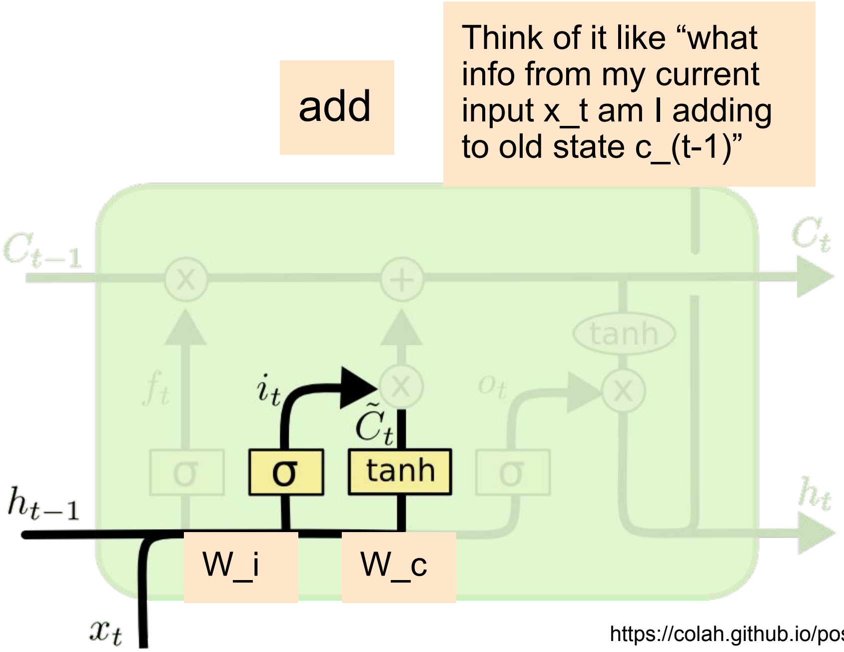


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs- Input Gate

- Input gate- function of current input and previous hidden state
- Decides what information to write to the cell state

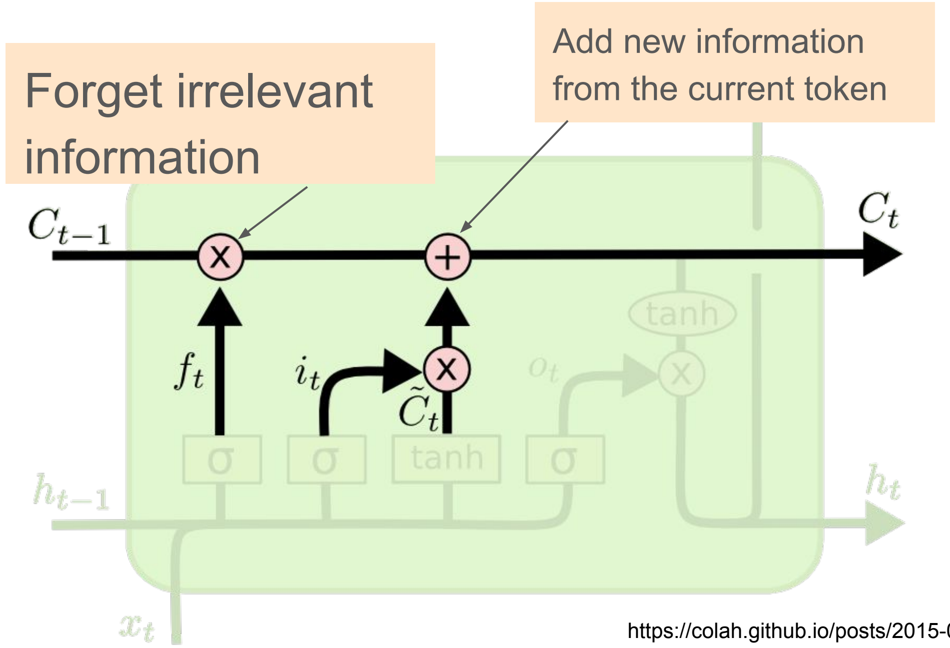


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM- Cell Update

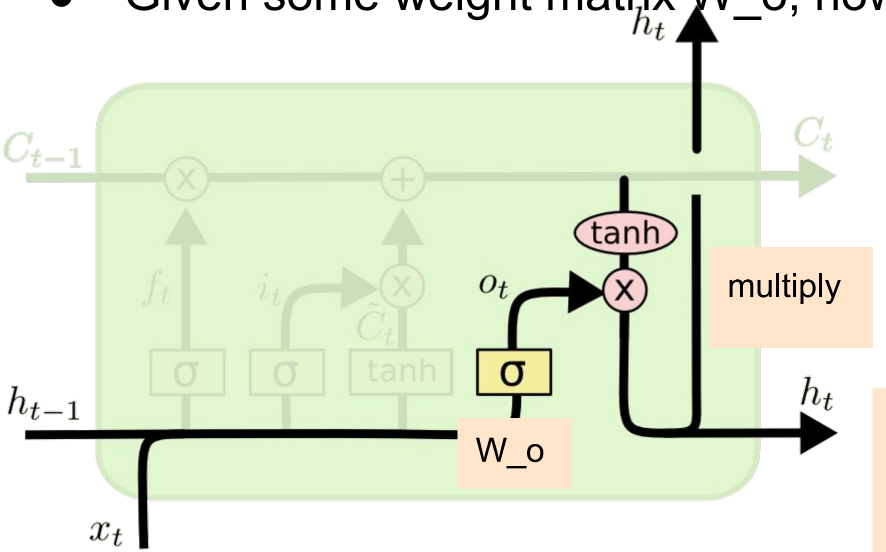


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM- Output Gate

- Output gate- function of current input and previous hidden state
- Controls flow of information from the cell state to the hidden state
- Given some weight matrix W_o, how do we write to o_t and h_t?



$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Think of it like “wanting to maintain a latent hidden space”

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs

- Performs better with long sequences
- But still sequential!!

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

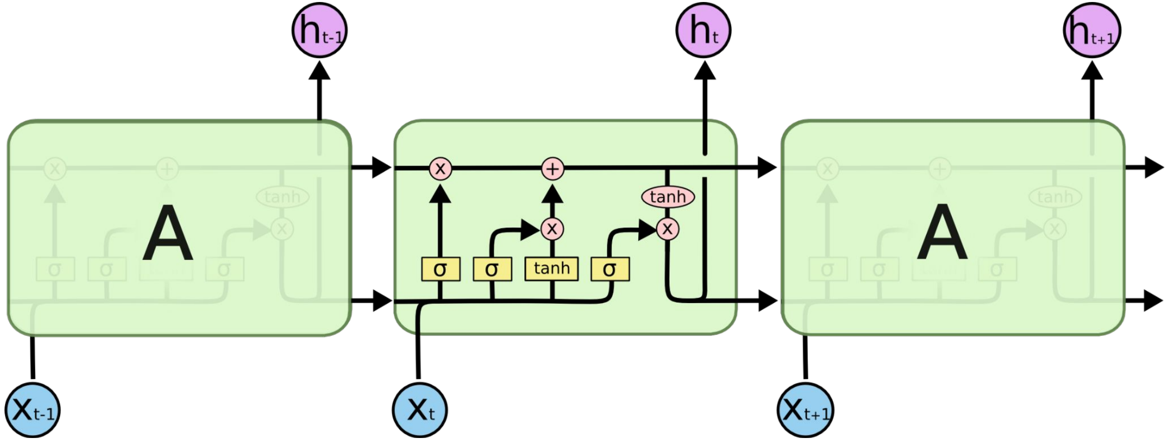
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

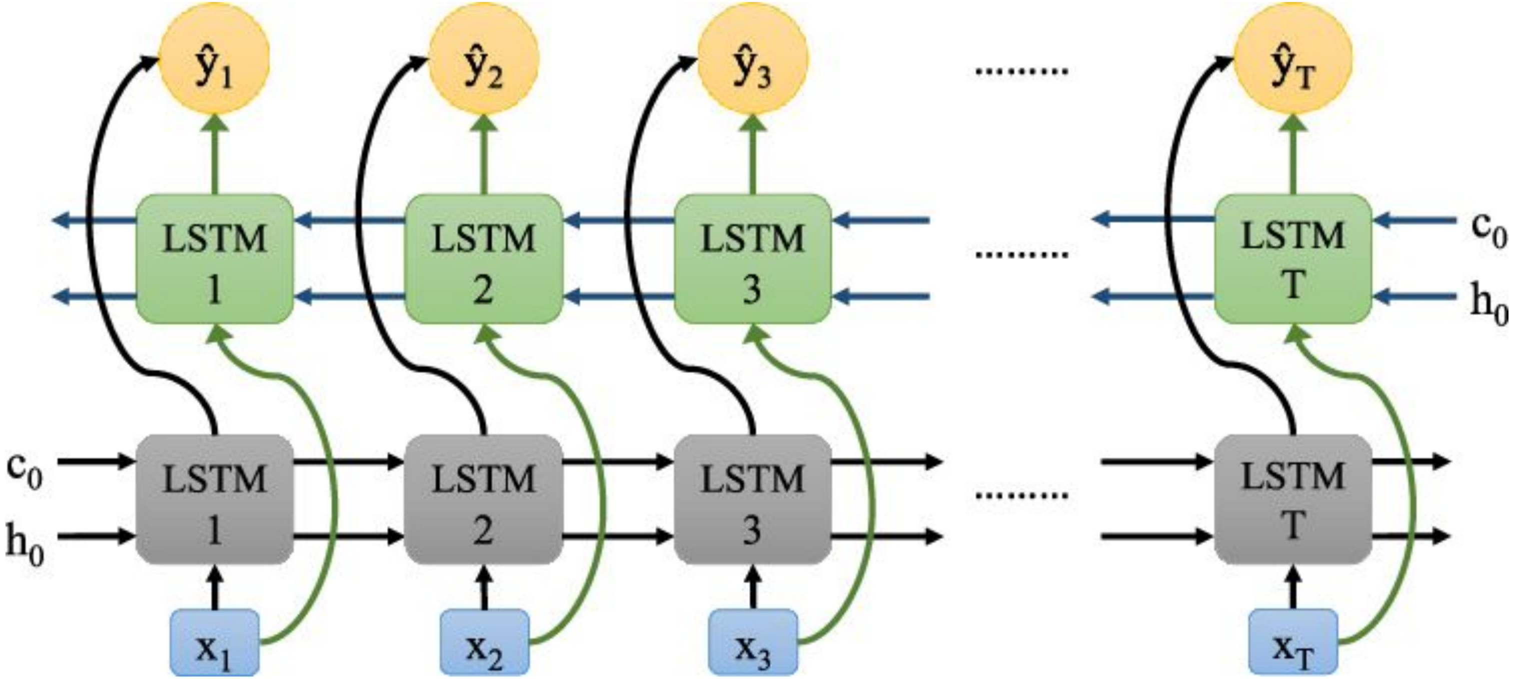


https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Context influences word meaning.

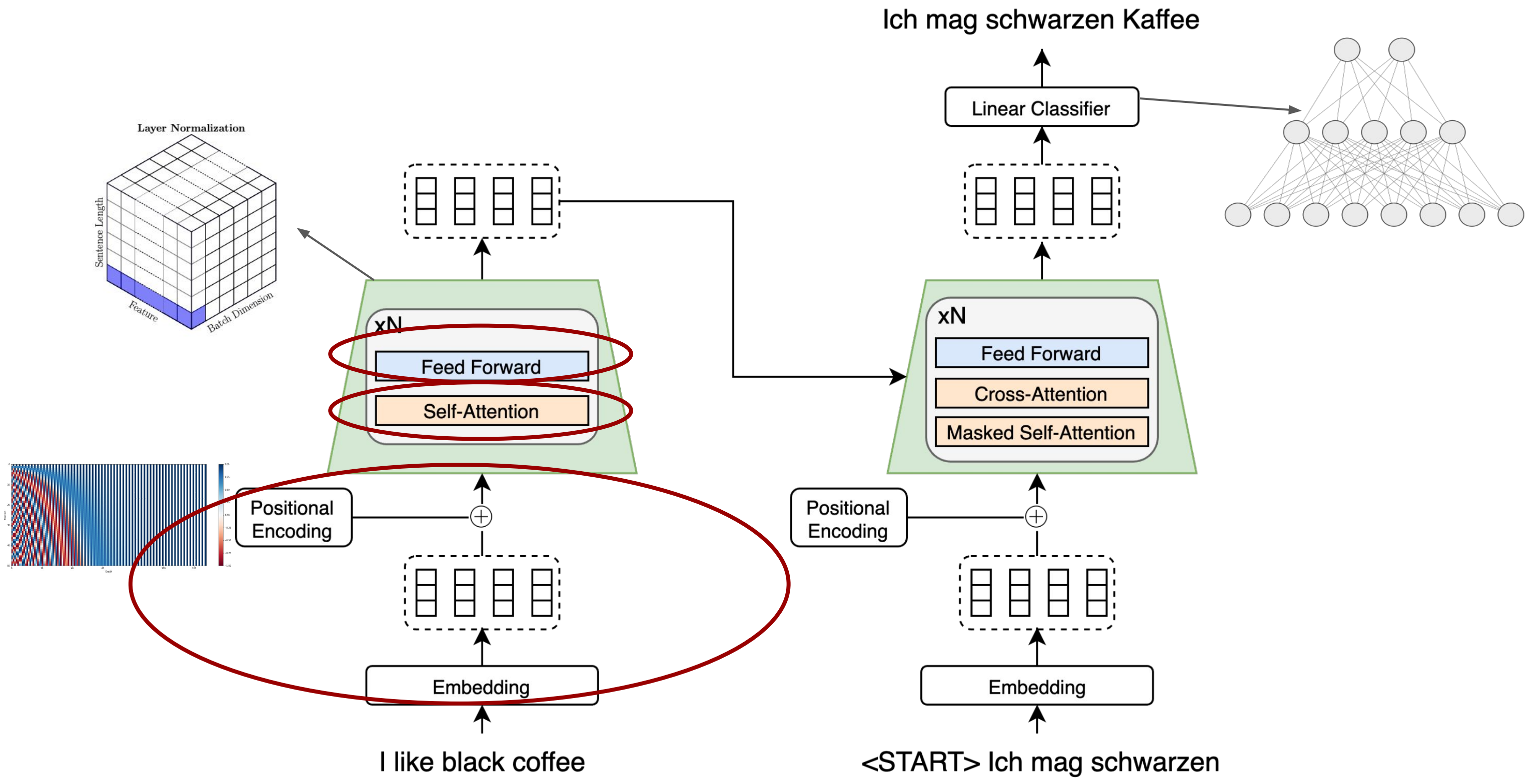
A bat flew out of the dugout, startling the baseball player and making him drop his bat.

Bidirectional LSTM



Transformers

What we cover



RNNs and LSTMs were sequential - Transformers are parallelized!

Token Position

Integer	0	1	2	3	4	5	6	7
Binary	0	0	0	0	1	1	1	1
	0	0	1	1	0	0	1	1
	0	1	0	1	0	1	0	1

Positional Embeddings

For a position pos and embedding dimension i:

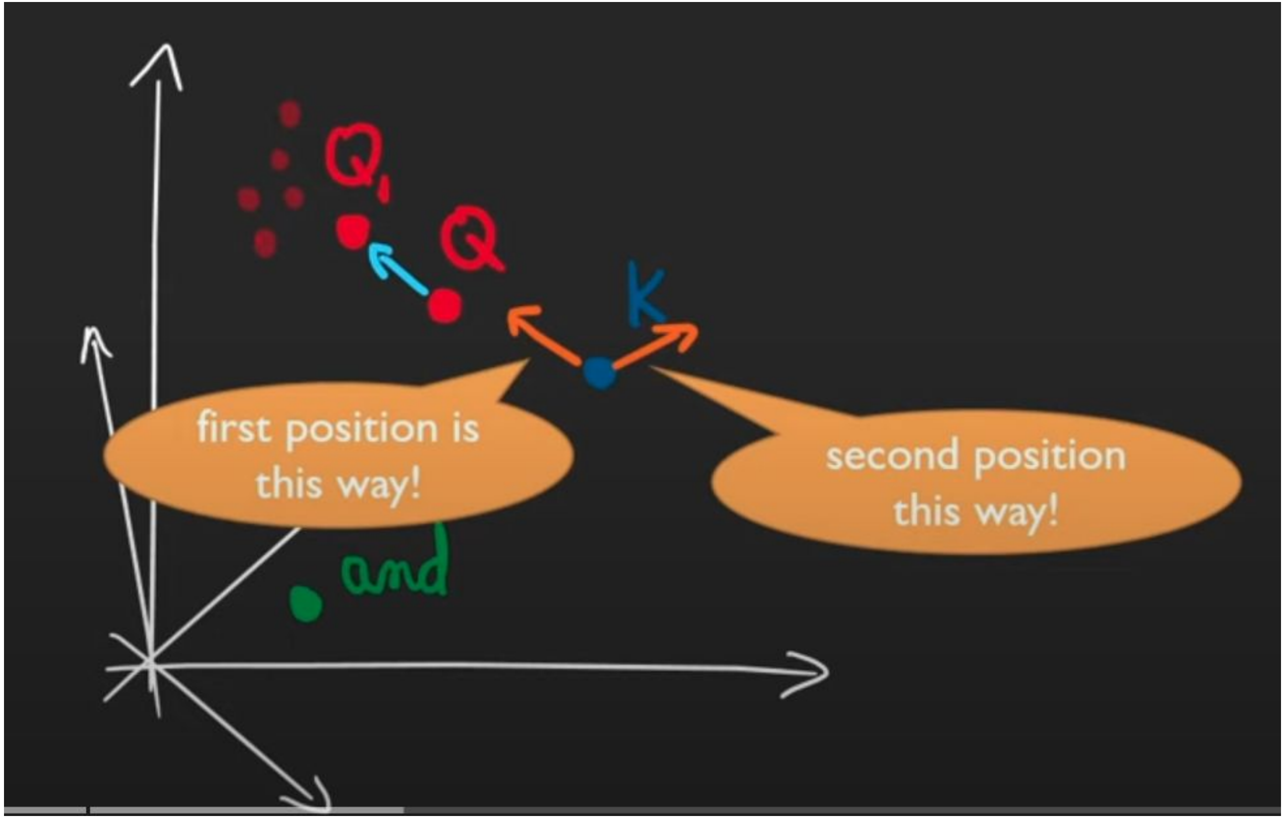
- $PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$
- $PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$

Token Position

Position	0	1	2	3	4	5	6	7
Sinusoid	0.5	0.3	0.3	0.5	0.7	0.9	0.9	0.7
	0.5	0.5	0.7	0.7	0.5	0.5	0.7	0.7
	0	1	0	1	0	1	0	1

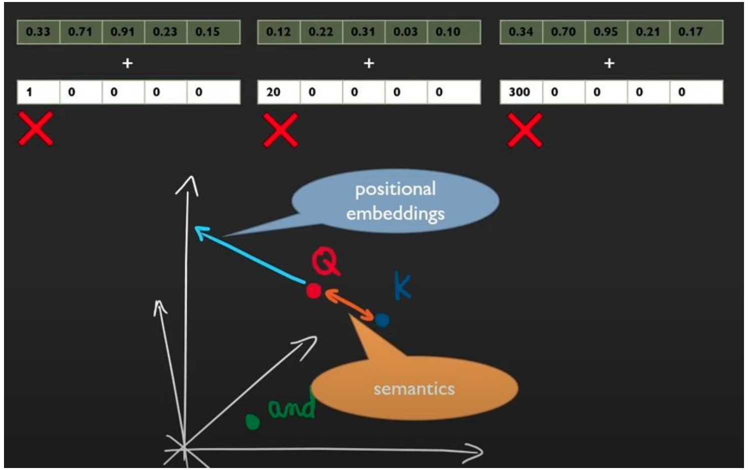
Positional Embeddings

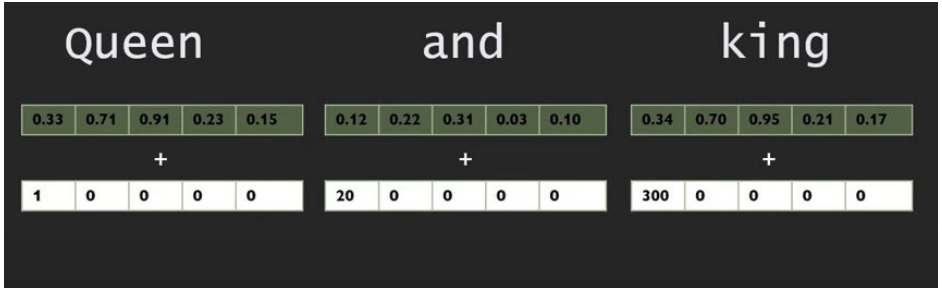
Sentence: “Queen and King”



Some requirements

- Every position should have a unique identifier
- Independent of input
- Independent of sequence length
- Numbers shouldn't be too large



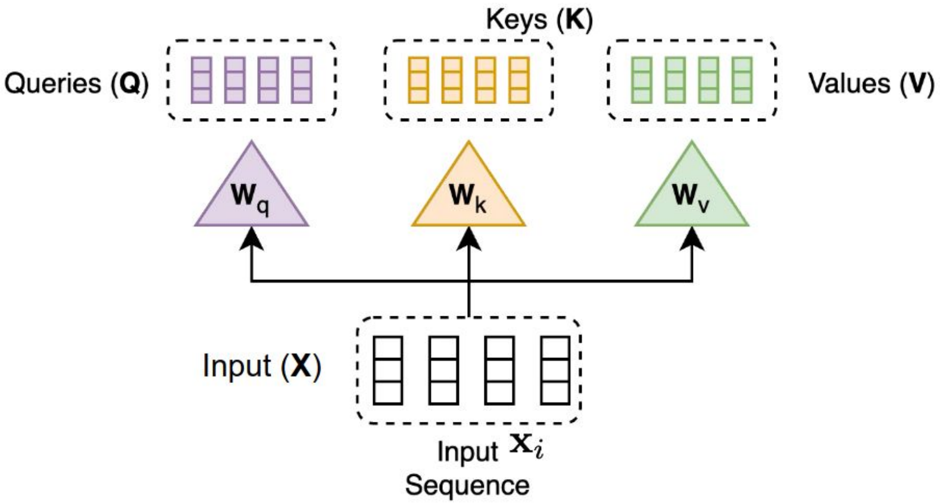


Self-Attention

(N x D) input Matrices

(N x D) output Matrix

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



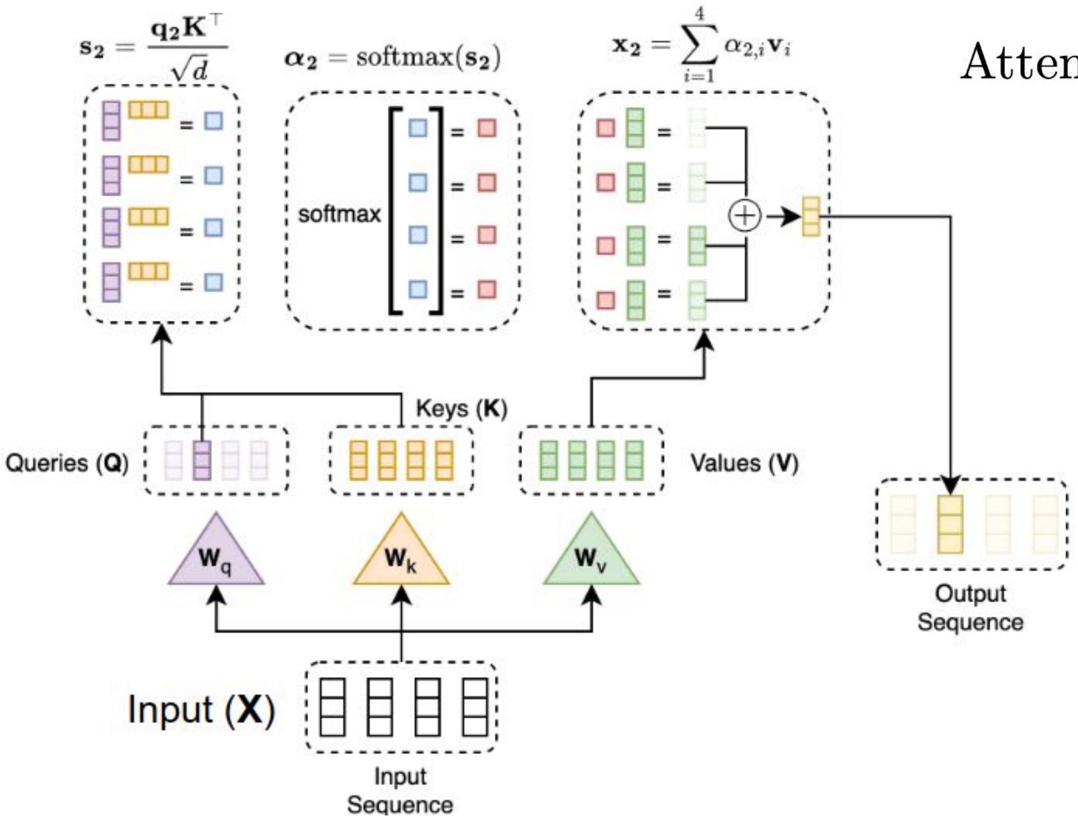
1. Get Q,K,V from the **N** input token using the weights.

   $\mathbf{x}, W_q, W_k, W_v$ (N,) → (N,D) $\{Q, K, V$

2. Use attention to transform this token representation with the other tokens in the sequence.

   (N,D) → (N,D) $\{A$

Self-Attention



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

1. Get `Q,K,V` from the **N** input token using the weights.

   $\mathbf{x}$, W_q, W_k, W_v `(N,)` → `(N,D)` $\{Q, K, V$

2. Use attention to transform this token representation with the other tokens in the sequence.

   `(N,D)` → `(N,D)` $\{A$

Self-Attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Implementation:

$$\text{MultiHead}(\mathbf{X}) = \mathbf{W}_O \text{Concat}(\text{head}_1, \ldots, \text{head}_h)$$

$$\text{where head}_i = \text{Attention}(\mathbf{W}_Q^i \mathbf{X}, \mathbf{W}_K^i \mathbf{X}, \mathbf{W}_V^i \mathbf{X})$$

```
Multi-Head-Attention(input x):
    # Split input into query, key, and value vectors
    q = split_heads(W_q(x))  # (b, h, n, d)
    k = split_heads(W_k(x))  # (b, h, n, d)
    v = split_heads(W_v(x))  # (b, h, n, d)

    # Compute attention scores and apply them to values
    attn_output = compute_attention(q, k, v)

    # Combine attention heads and apply output transformation
    output = W_0(combine_heads(attn_output))

    return output
```

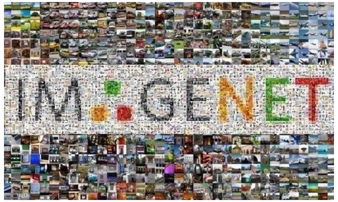
$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i$$

$$\mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i$$

$$\mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i$$

Representation Learning

Task

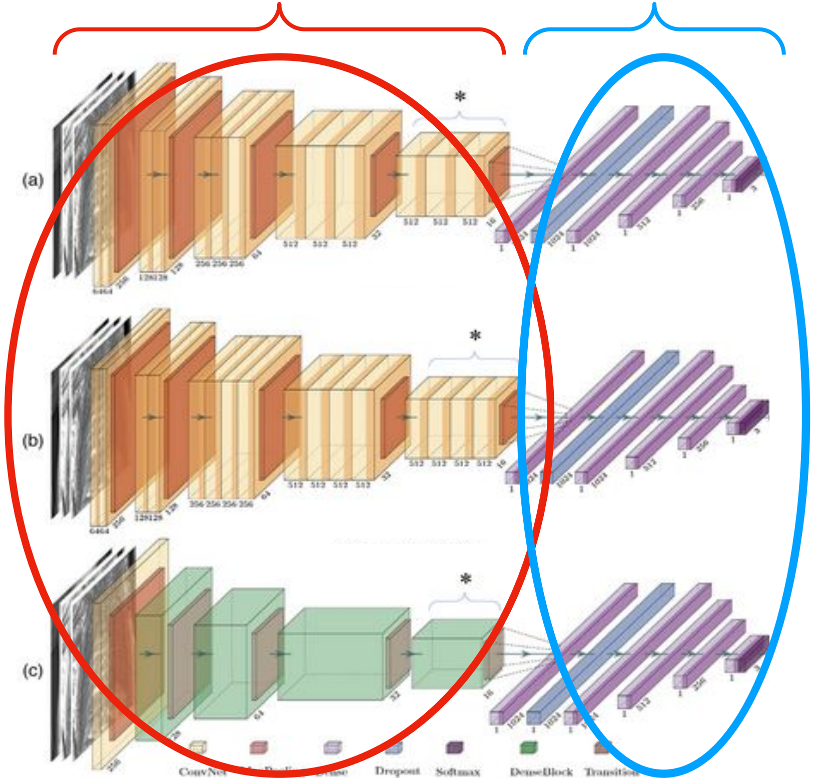


Dataset-*“Agnostic”* Dataset-Specific

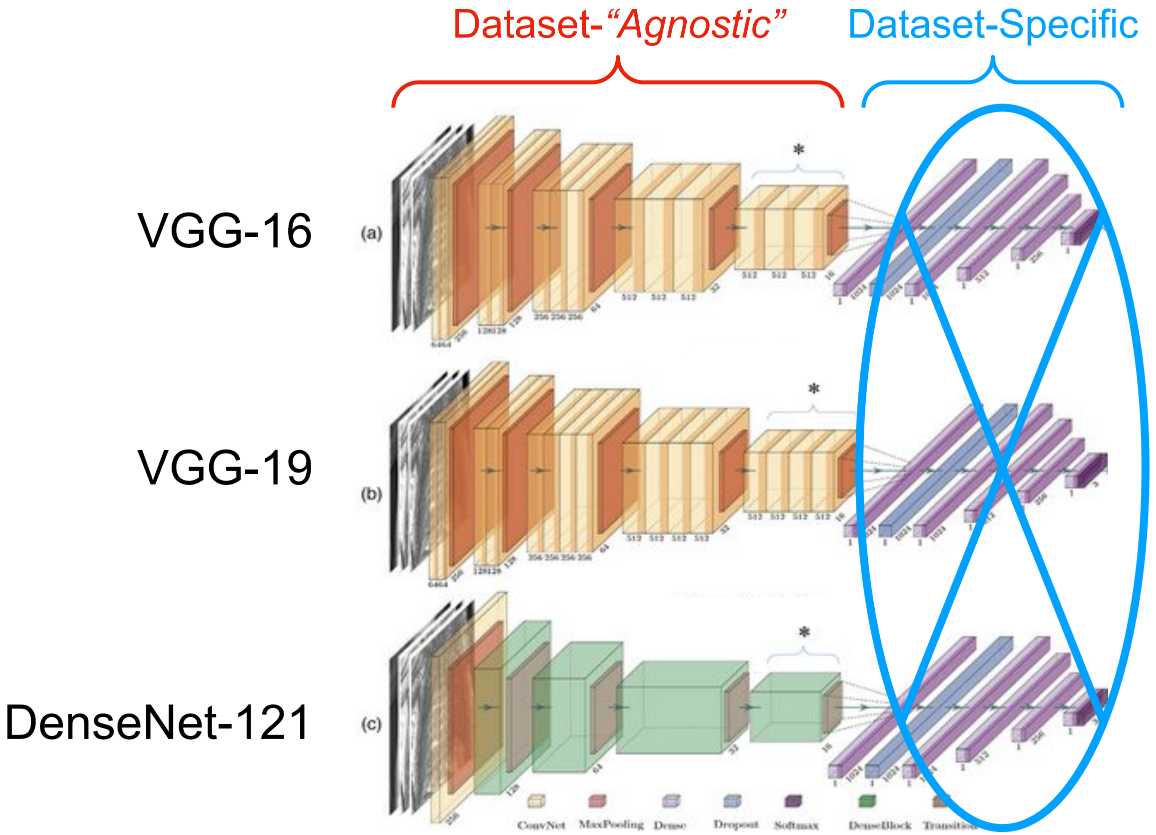
VGG-16

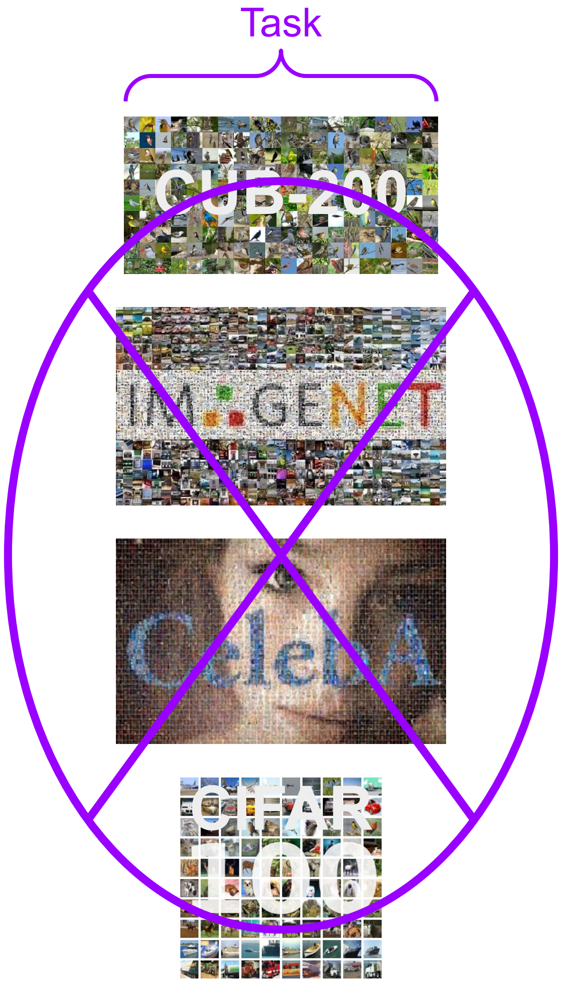
VGG-19

DenseNet-121



Representation Learning



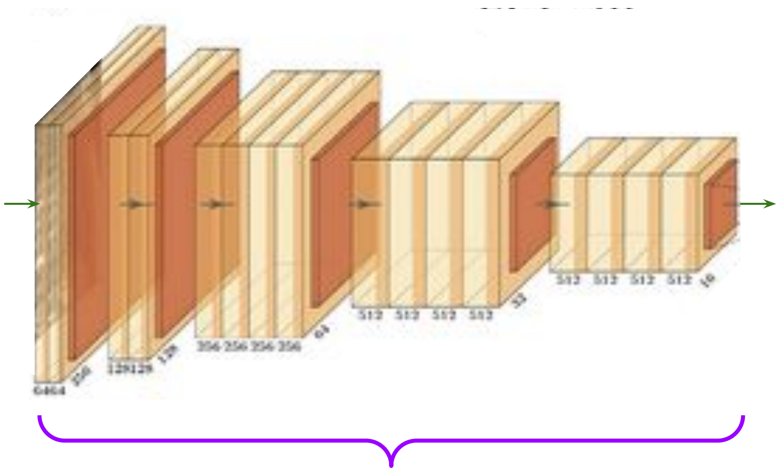


Representation Learning

Unlabeled Image Data

$\mathbb{R}^{224\times224}$

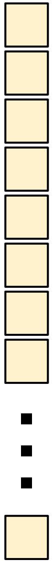
$$f(x) : \mathbb{R}^{224\times224} \rightarrow \mathbb{R}^{768}$$



How to get this useful image embedder?

Image Representation

$\mathbb{R}^{768}$



B.ring
Y.our
O.wn
C.lassifier

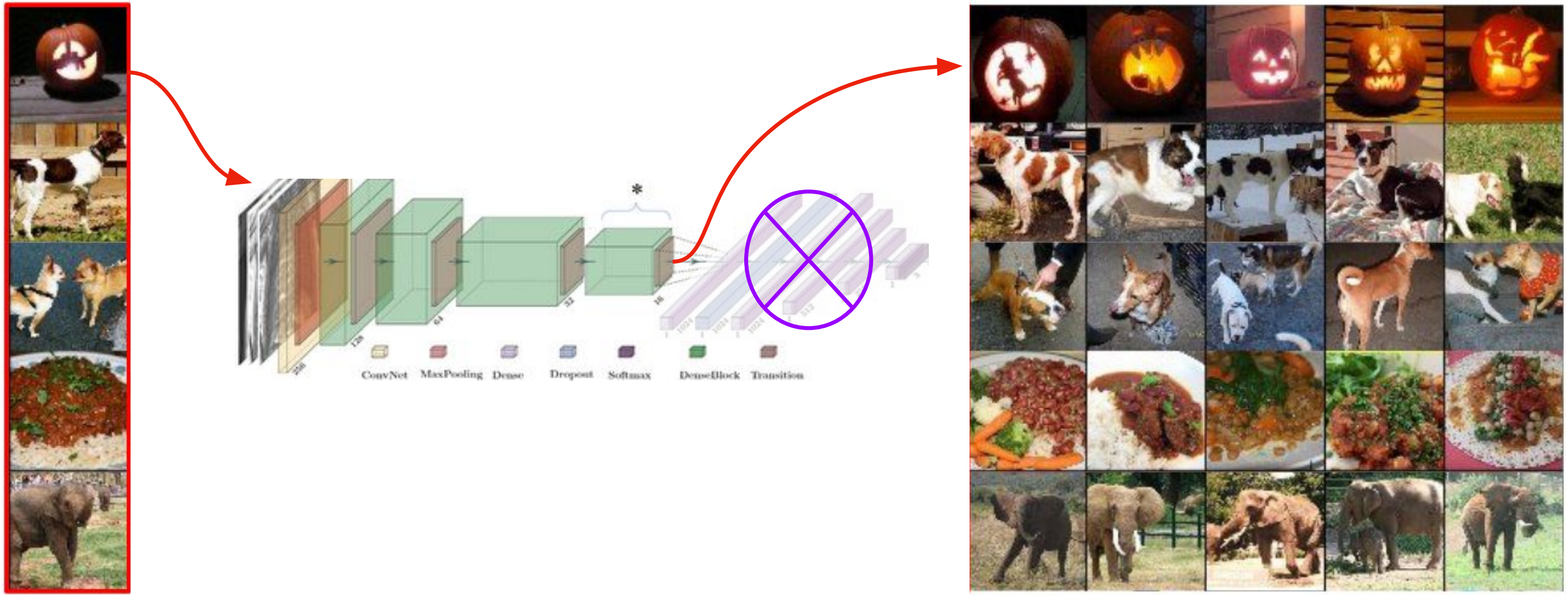
Representation Learning

Q: How to get an image embedder?

Train on ImageNet and throw away the classifier.

➔ Supervised Model Features

Neural Net Features: Nearest Neighbors

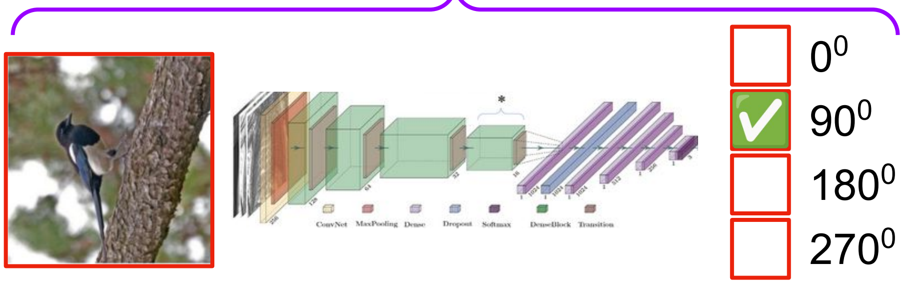


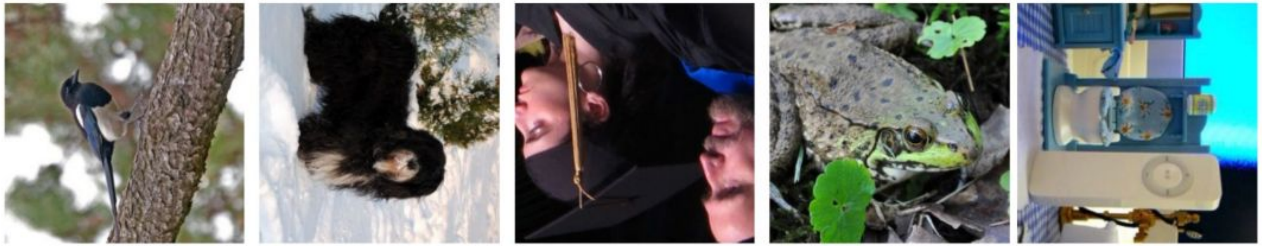
Representation Learning

Q: How to get an image embedder?

- ➔ Supervised Model Features
- ➔ Self-Supervised Learning
  - ◆ Pre-training Task

Pick a Task Correlated with Image Understanding
(Make it easy to label!)





90° rotation | 270° rotation | 180° rotation | 0° rotation | 270° rotation

Figure 1: Images rotated by random multiples of 90 degrees (e.g., 0, 90, 180, or 270 degrees). The core intuition of our self-supervised feature learning approach is that if someone is not aware of the concepts of the objects depicted in the images, he cannot recognize the rotation that was applied to them.

Representation Learning

Q: How to get an image embedder?

➔ Supervised Model Features

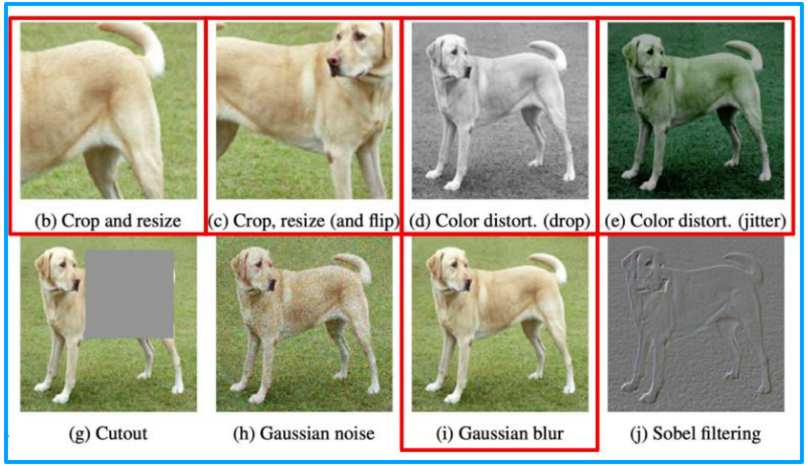
➔ Self-Supervised Learning

- Pre-training Task
- Contrastive Learning

(+negative augmentations)

$\mathbf{x}^-$: **Negative** examples (*different images*)

$\mathbf{x}^+$: **Positive** examples (*augmentations*)



$\mathbf{x}$: Sample

(a) Original

Representation Learning

Q: How to get an image embedder?

➔ Supervised Model Features

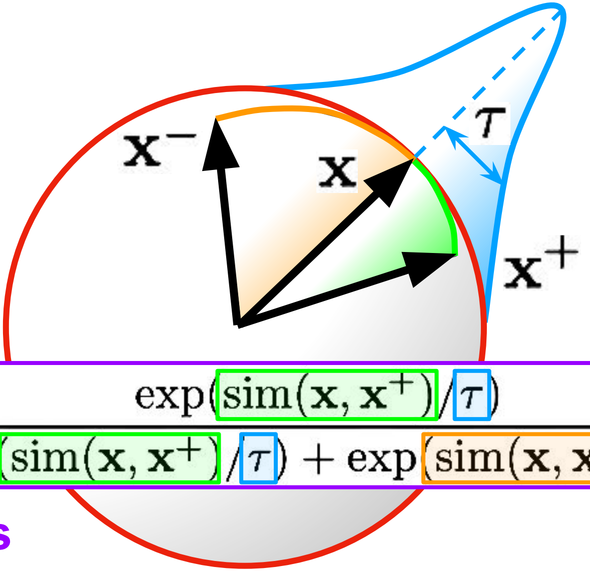
➔ Self-Supervised Learning

- ◆ Pre-training Task
- ◆ Contrastive Learning

$\mathbf{x}$: Sample

$\mathbf{x}^-$: Negative examples

$\mathbf{x}^+$: Positive examples

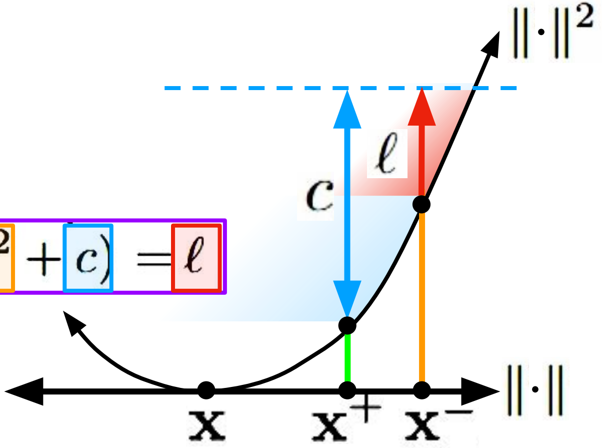


$$-\log\left(\frac{\exp(\mathrm{sim}(\mathbf{x},\mathbf{x}^+)/\tau)}{\exp(\mathrm{sim}(\mathbf{x},\mathbf{x}^+)/\tau)+\exp(\mathrm{sim}(\mathbf{x},\mathbf{x}^-)/\tau)}\right)$$

SimCLR Loss

$$\max(0, \|f(\mathbf{x_i}) - f(\mathbf{x}^+)\|^2 - \|f(\mathbf{x_i}) - f(\mathbf{x}^-)\|^2 + c) = \ell$$

Triplet Loss



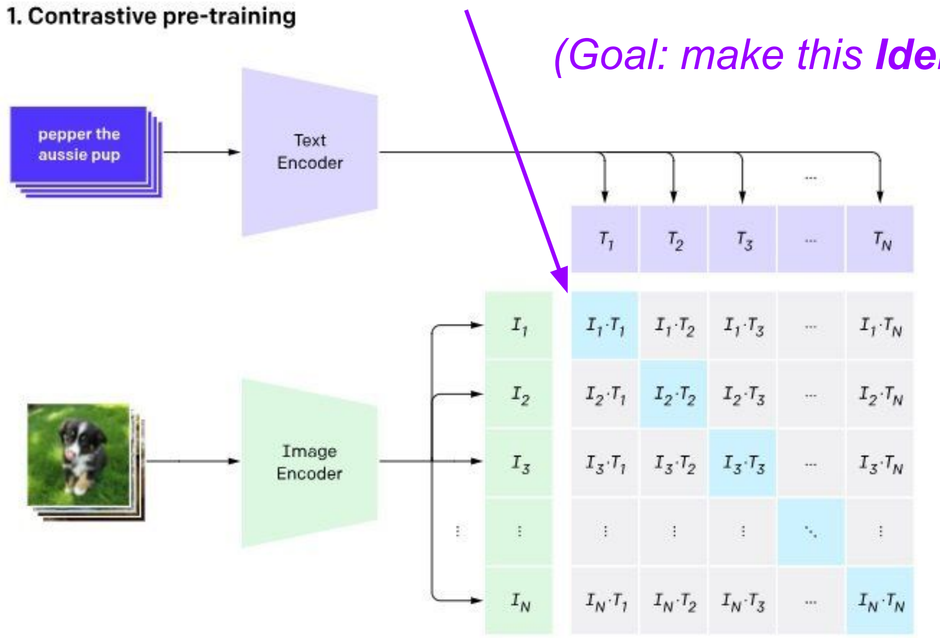
Representation Learning (Multimodal)

Q: How to get ~~an image~~ a multimodal embedder?

- ➔ Supervised Model Features
- ➔ Self-Supervised Learning
  - ◆ Pre-training Task
  - ◆ Contrastive Learning

Cosine similarity between text embedding and image embedding.

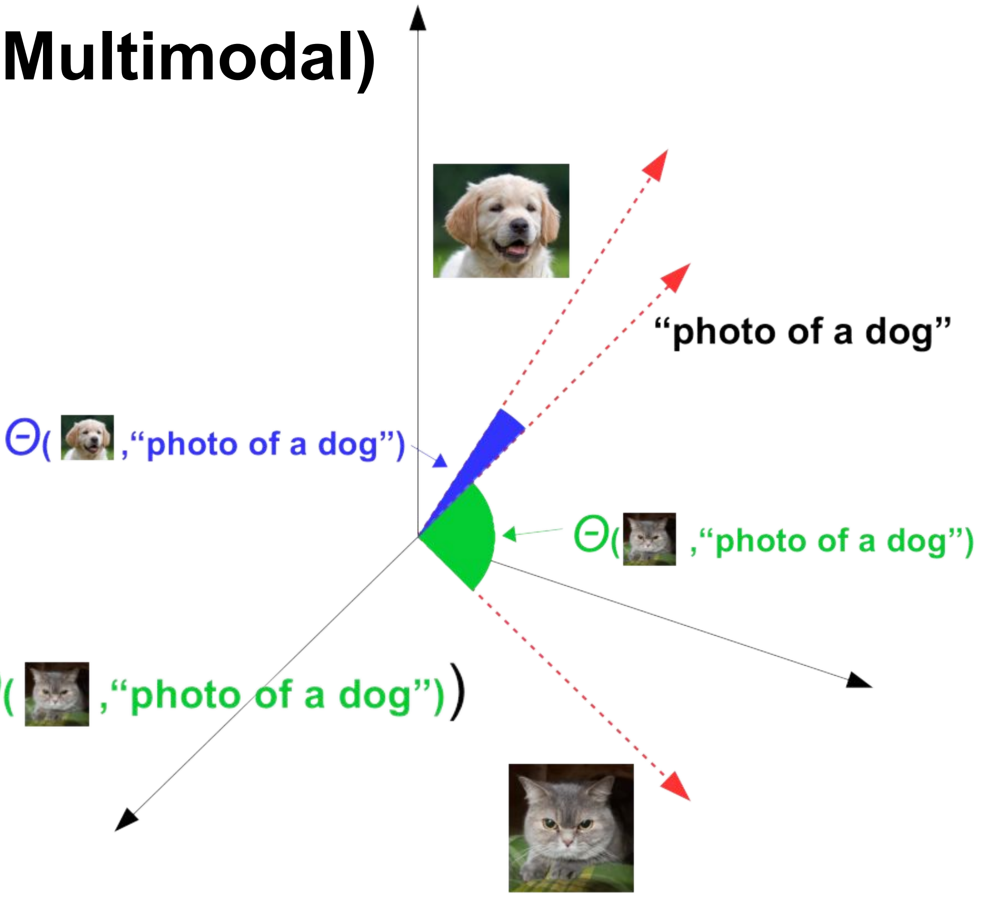
*(Goal: make this **Identity**)*



Representation Learning (Multimodal)

Q: How to get ~~an image~~ a multimodal embedder?

- ➔ Supervised Model Features
- ➔ Self-Supervised Learning
  - ◆ Pre-training Task
  - ◆ Contrastive Learning



cos(Θ(,"photo of a dog")) > cos(Θ(,"photo of a dog"))

Representation Learning

Q: How to get an image embedder?

➔ Supervised Model Features

➔ Self-Supervised Learning

- Pre-training Task
- Contrastive Learning
- Teacher-Student

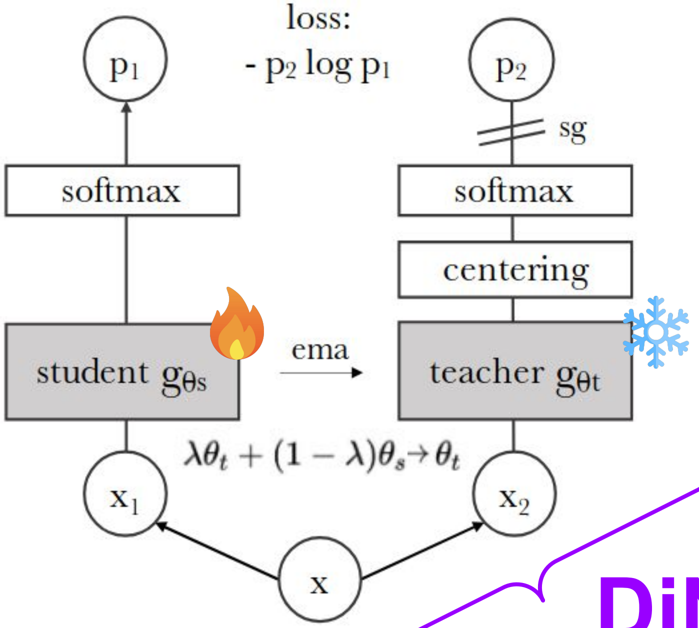
Representation Learning

Q: How to get an image embedder?

➔ Supervised Model Features

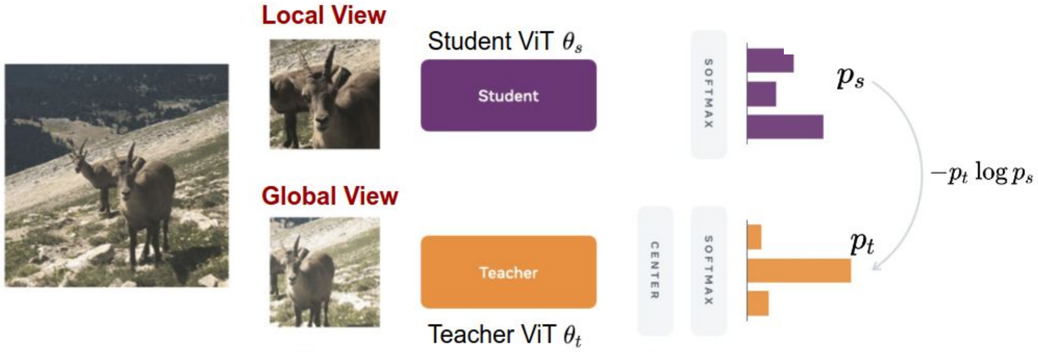
➔ Self-Supervised Learning

- Pre-training Task
- Contrastive Learning
- Teacher-Student



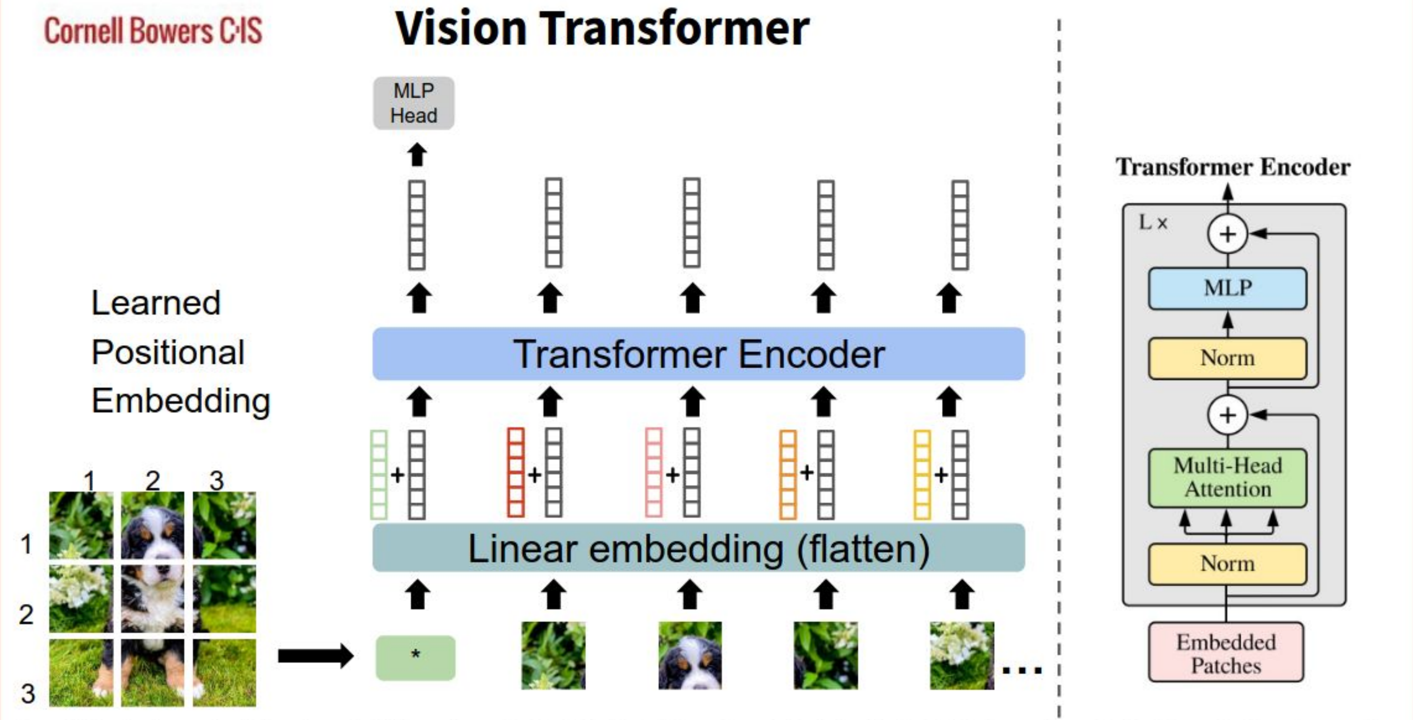
DiNO

"**Di**stillation **No** Labels"



Vision Transformers (ViT)

Frequently pretrained with DiNO, you'll probably see *"DiNO-ViTs"* a lot.

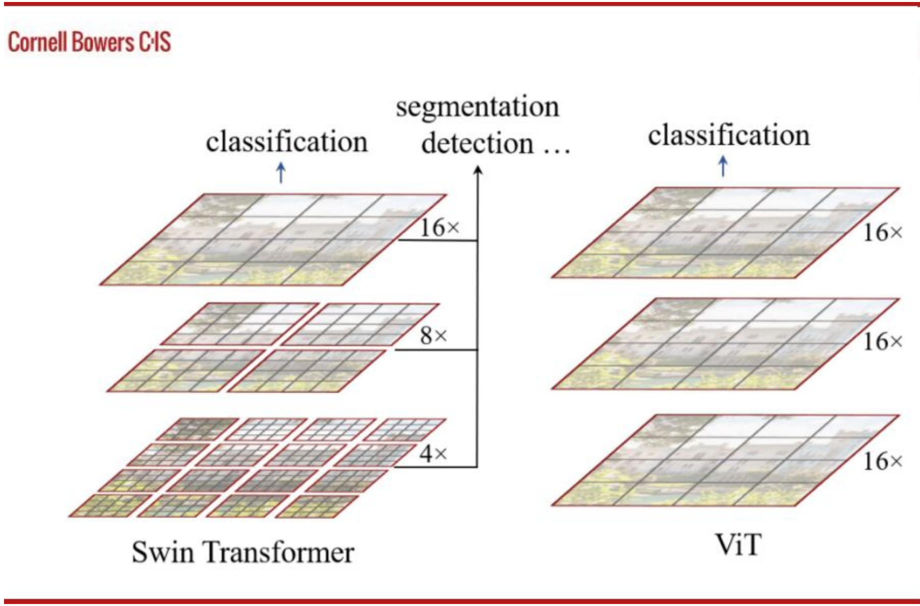


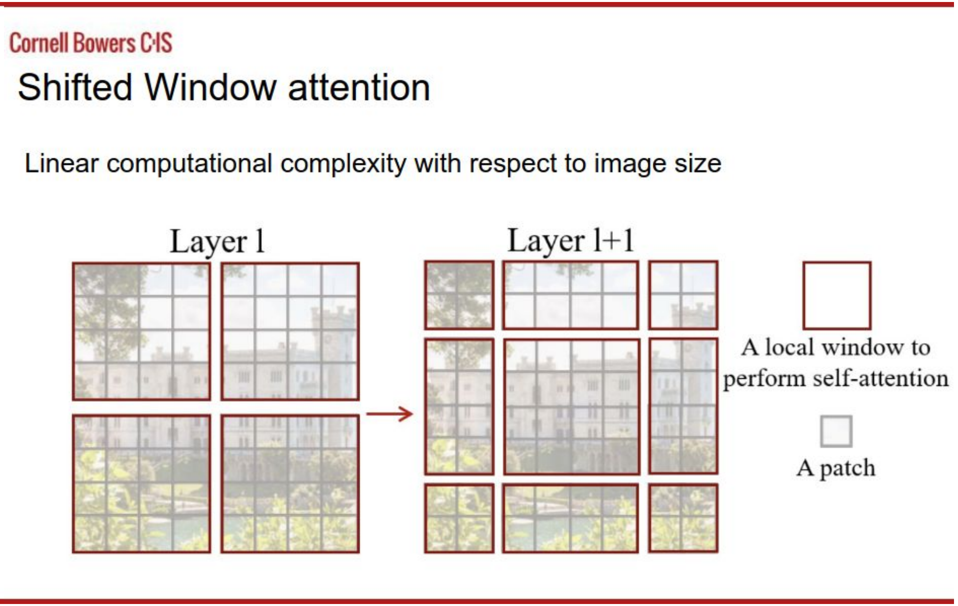
Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929.*

Vision Transformers (ViT)

SWIN “Scaled WINdow”

- Popular for large images and segmentation because of hierarchical features.
- Attention is per-window.
  - Shifted windows allow long-range connections at higher layers.





Vision Transformers (ViT)

DiNO v2

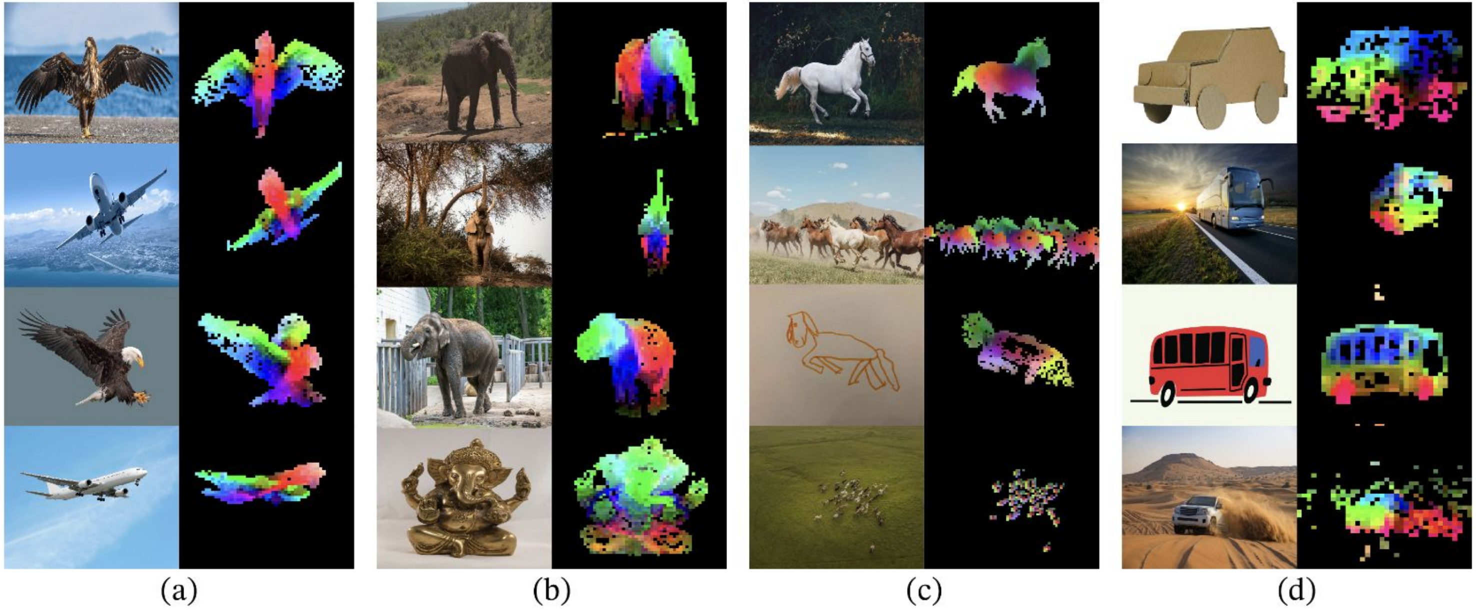
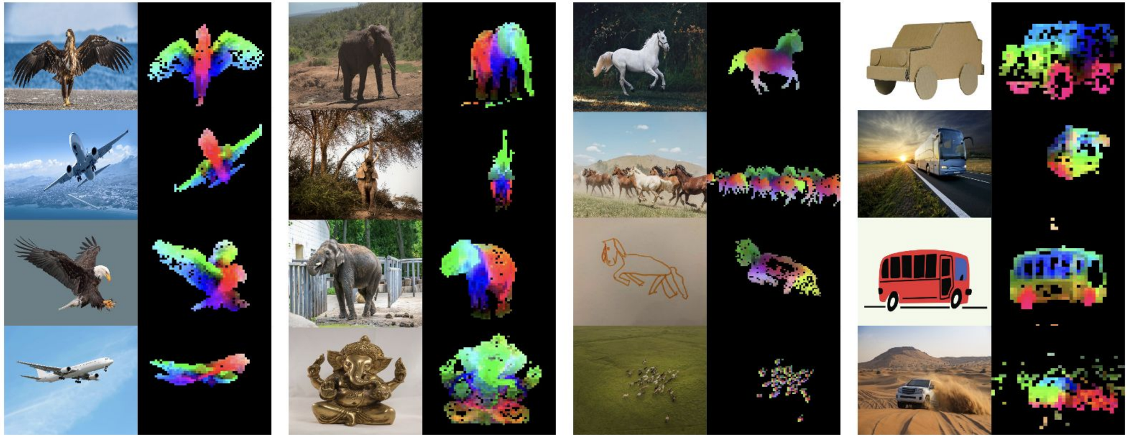


Figure 1: **Visualization of the first PCA components.** We compute a PCA between the patches of the images from the same column (a, b, c and d) and show their first 3 components. Each component is matched to a different color channel. Same parts are matched between related images despite changes of pose, style or even objects. Background is removed by thresholding the first PCA component.

Vision Transformers (ViT)

Figure 1: Visualization of the first PCA components of the image patches.

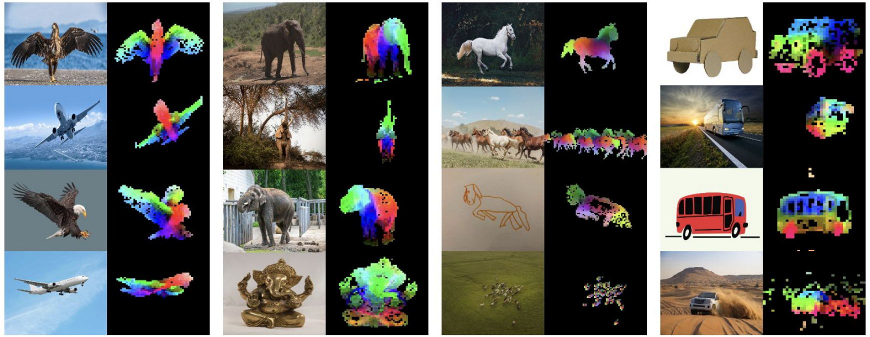


DiNO v2

Q: Why don't we get these visualizations from deep CNNs?

(*NOTE:* You **can** train ResNet-50 with a DiNO, and it works great!)

Vision Transformers (ViT) *(vs deep CNNs)*



class: zebra

class: convertible

ViTs *start* with a global receptive fields.

CNNs *take a while* to combine global features.

- Interpretable long-range dependencies from attention are visible early on.
- DiNO's objective encourages memorizing "parts-of-whole".

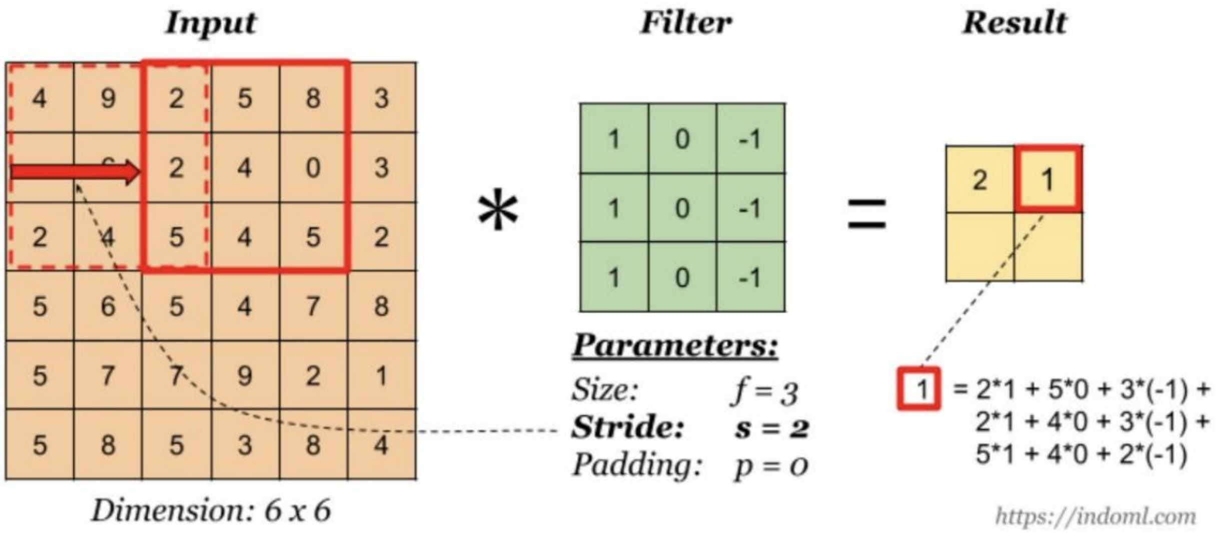
- Creates high level features, but it is hard to determine which pixels they correspond to.
- Must use methods like **Class Activation Mapping** up the network to get to pixel-level effects.

CNN review

HW1 Q4

Output dimension:

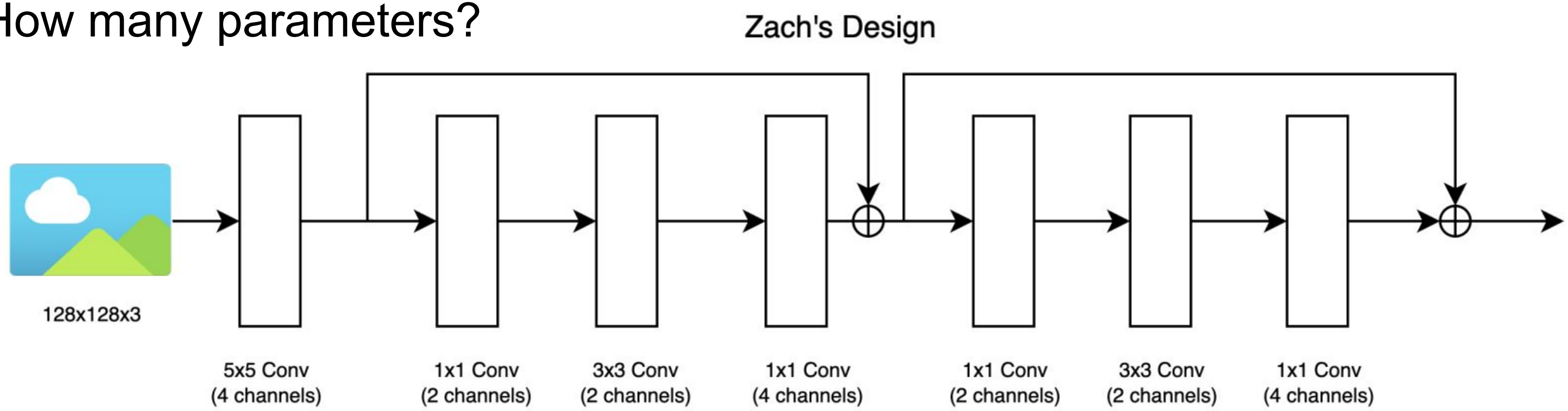
$$(\lfloor \frac{n-k+2p}{s} \rfloor + 1) \times (\lfloor \frac{n-k+2p}{s} \rfloor + 1) \times l$$



Filter with stride (s) = 2

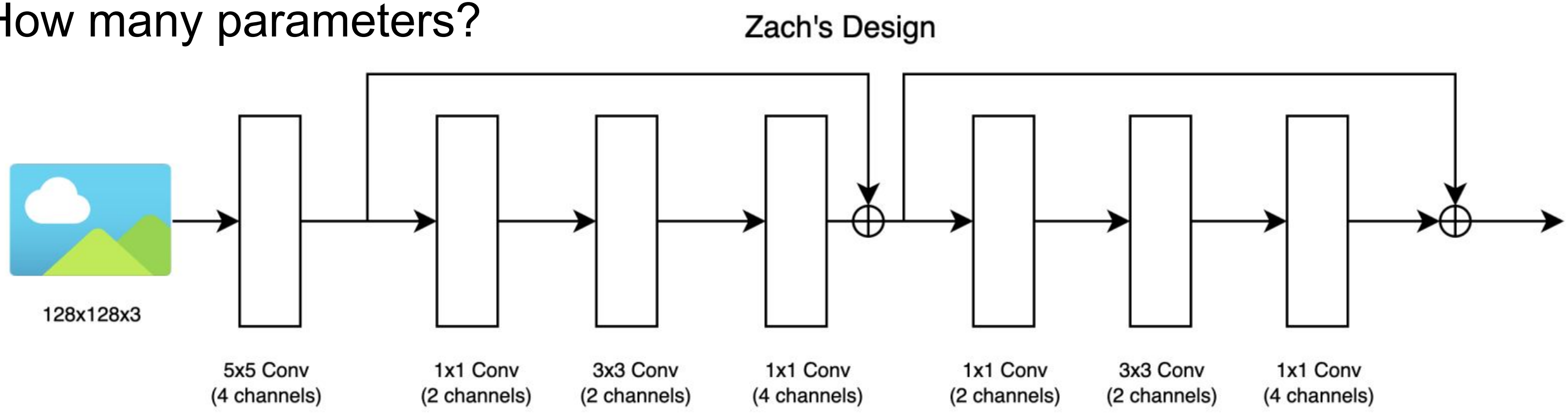
SP24 Prelim Q7

How many parameters?



SP24 Prelim Q7

How many parameters?



Layer 1: (5x5) x 3 x 4 = 300
Layer 2: (1x1) x 4 x 2 = 8
Layer 3: (3x3) x 2 x 2 = 36
Layer 4: (1x1) x 2 x 4 = 8

Layer 5: (1x1) x 4 x 2 = 8
Layer 6: (3x3) x 2 x 2 = 36
Layer 7: (1x1) x 2 x 4 = 8

Total: 404

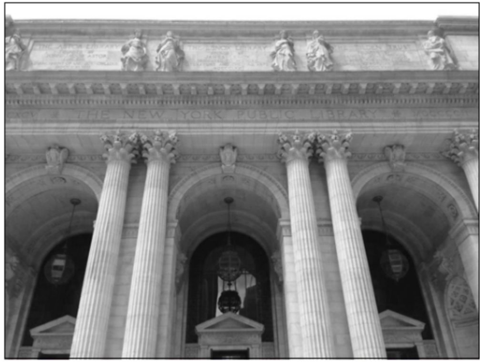
CNN review

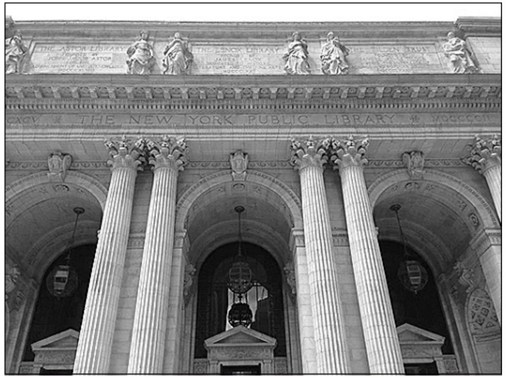
The set of filters is:

$$\left\{\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}\right\}$$

Which output comes from which filter?

Input image:





SP24 Prelim Q5

https://setosa.io/ev/image-kernels/

CNN review

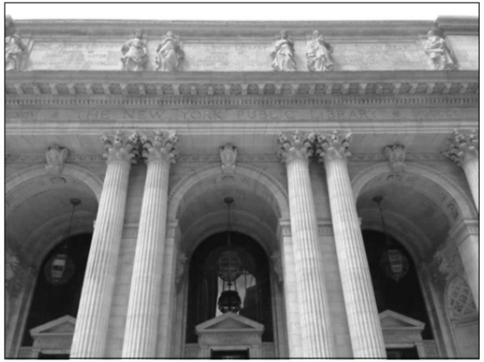
The set of filters is:

Sharpen, Edge, Left sobel

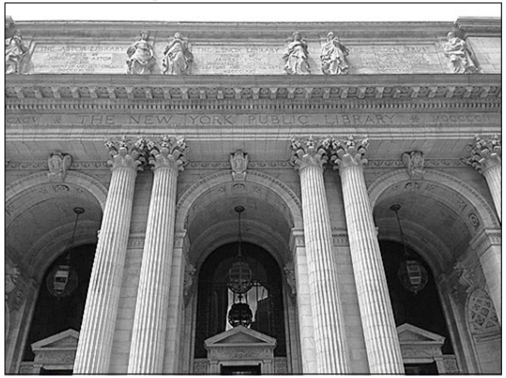
$$\left\{ \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \right\}$$

Which output comes from which filter?

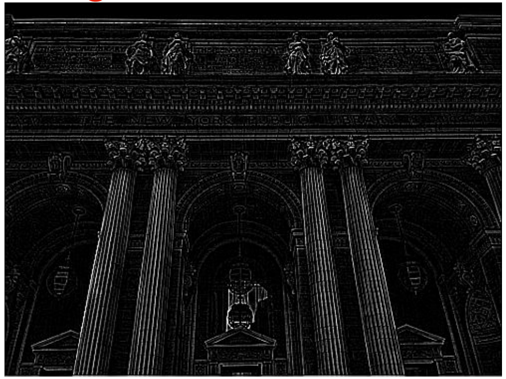
Input image:



Sharpen



Edge



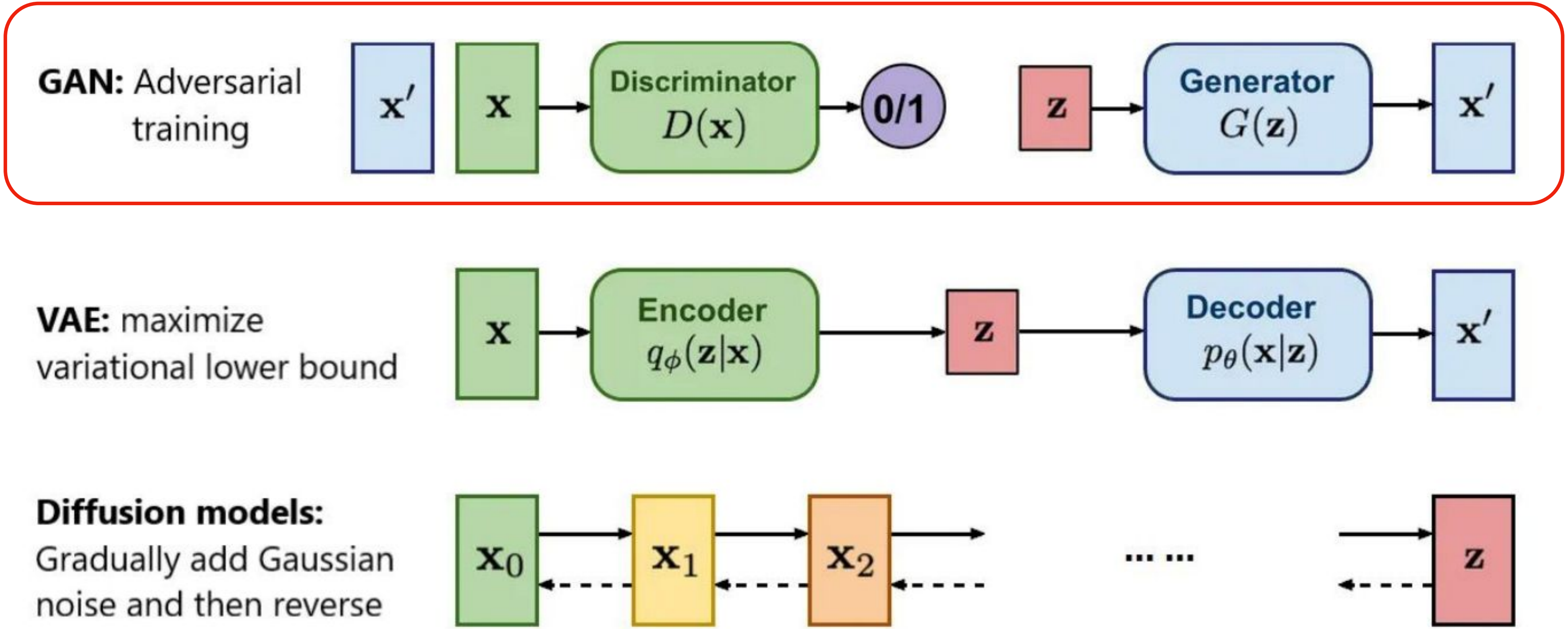
Left sobel

Bottom sobel

SP24 Prelim Q5

https://setosa.io/ev/image-kernels/

GANs

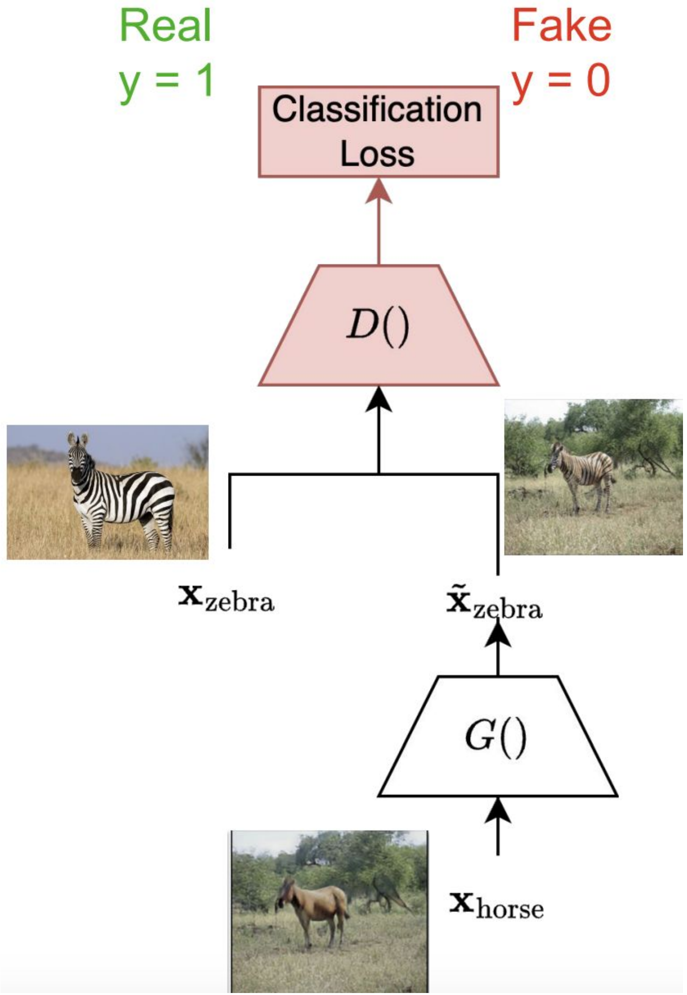

GAN: Adversarial training
x′
x
Discriminator
D(x)
0/1
z
Generator
G(z)
x′
VAE: maximize variational lower bound
x
Encoder
qϕ(z|x)
z
Decoder
pθ(x|z)
x′
Diffusion models: Gradually add Gaussian noise and then reverse
x0
x1
x2
......
z

GANs

- Motivation: Conditional image generation without labelled pairs (Why? Data scarcity)
    - Eg. translate horses into zebras

12 Astonishing Facts About Horses

Zebra Facts | Live Science



GANs

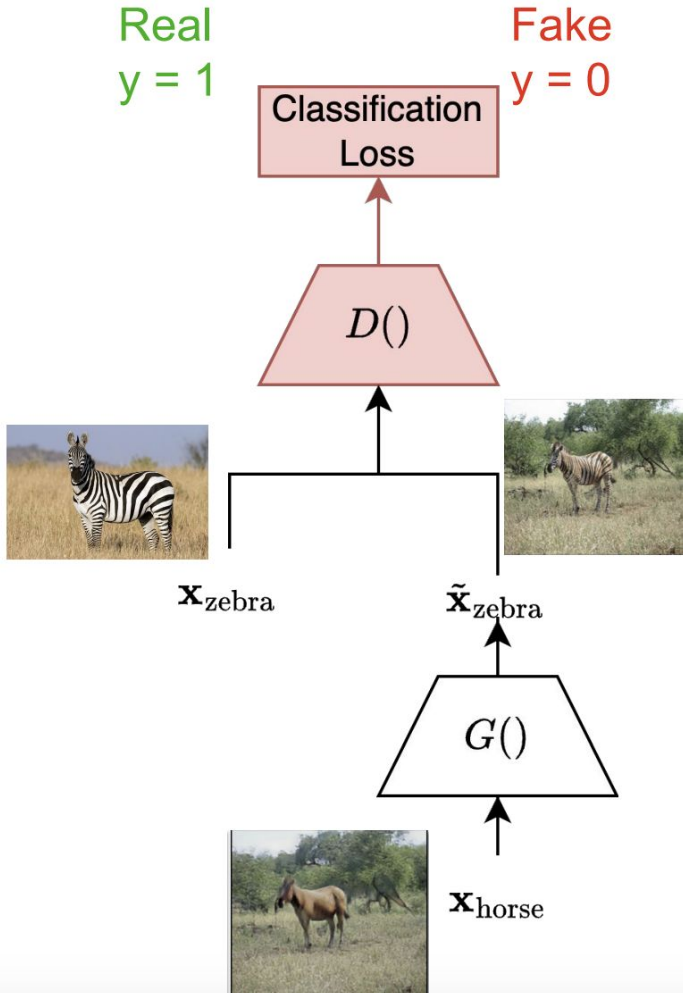
Adversarial Networks

- Generator (G): performs the image translation
  - tries to *fool* the discriminator
- Discriminator (D): predicts whether the image was generated by the generator
  - measures how good the generator did

To train, iteratively:

1. Update discriminator on real/fake images
2. Update generator using feedback from *new* discriminator

Why can't we train them simultaneously?



GANs

Positive example: real image from labelled dataset

Negative example: fake image from generator

Discriminator Loss:

$$L_D = \min_D \left[-\log(D(\mathbf{x})) - \log(1 - D(G(\mathbf{z})))\right]$$

Generator Loss:

$$\max_G \min_D \left[-\log(D(\mathbf{x})) - \log(1 - D(G(\mathbf{z})))\right]$$

$$L_G = -L_D$$

GANs

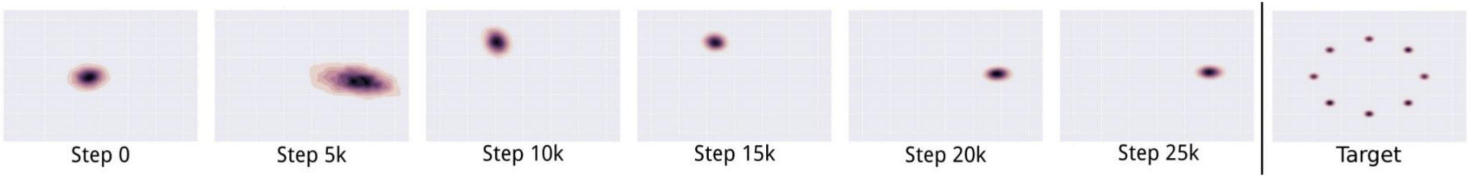
GANs can also be used for *unconditional* image generation

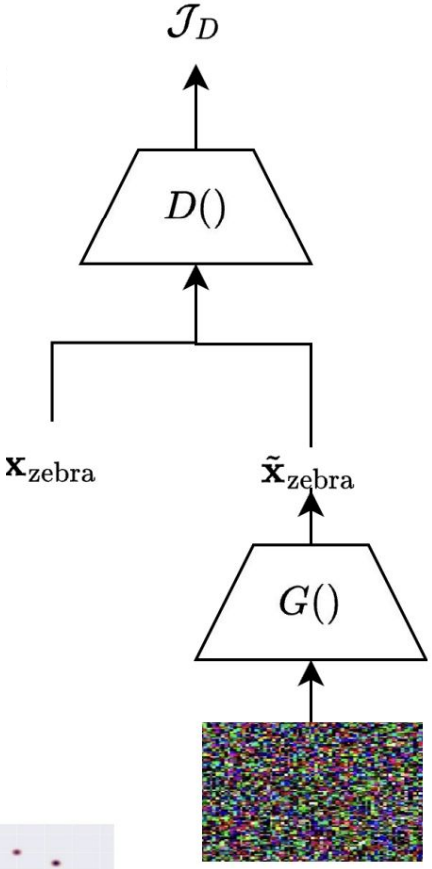
- Sample generator using Gaussian noise
- Results in very high quality images, quickly!

However, GANs experience *Mode Collapse*

- Learn only what best fools the discriminator
- Generated images do not represent full distribution of image class

Mode collapse





Prelim Q2 (Only focus on GANs)

For each of the following statements, indicate whether the statement describes VAEs, GANs, or Diffusion models. You should list all that apply.

1. Trained with a likelihood-based objective.
2. Has a learnable encoder that maps the data to a Gaussian distribution.
3. Has latent variables that must be the same size as the input data.
4. Learns to generate images by fooling a discriminator.
5. Generates high-quality, realistic images.
6. Generates images with a single forward pass of the network.
7. Often suffers from poor diversity among generations.
8. Transforms samples from a unit normal distribution to samples from the data distribution.

Prelim Q2 (Only focus on GANs)

For each of the following statements, indicate whether the statement describes VAEs, GANs, or Diffusion models. You should list all that apply.

1. Trained with a likelihood-based objective.

   N

2. Has a learnable encoder that maps the data to a Gaussian distribution.

   N

3. Has latent variables that must be the same size as the input data.

   N

4. Learns to generate images by fooling a discriminator.

   Y

5. Generates high-quality, realistic images.

   Y

6. Generates images with a single forward pass of the network.

   Y

7. Often suffers from poor diversity among generations.

   Y

8. Transforms samples from a unit normal distribution to samples from the data distribution.

   Y

Brief U-Net Review

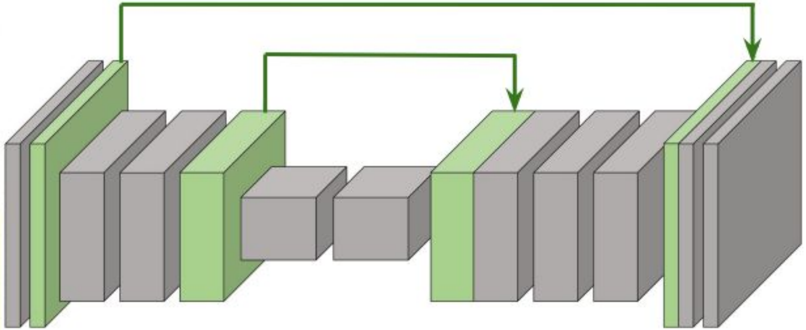
✅ Convolutions — Allow parallelization to extract latent vector for each pixel

✅ Hourglass — Improve efficiency by reducing computations with downsampling

Increase receptive field size by convolving on downsampled feature maps

✅ Skip Connections — Improve prediction quality by combining low-level image features

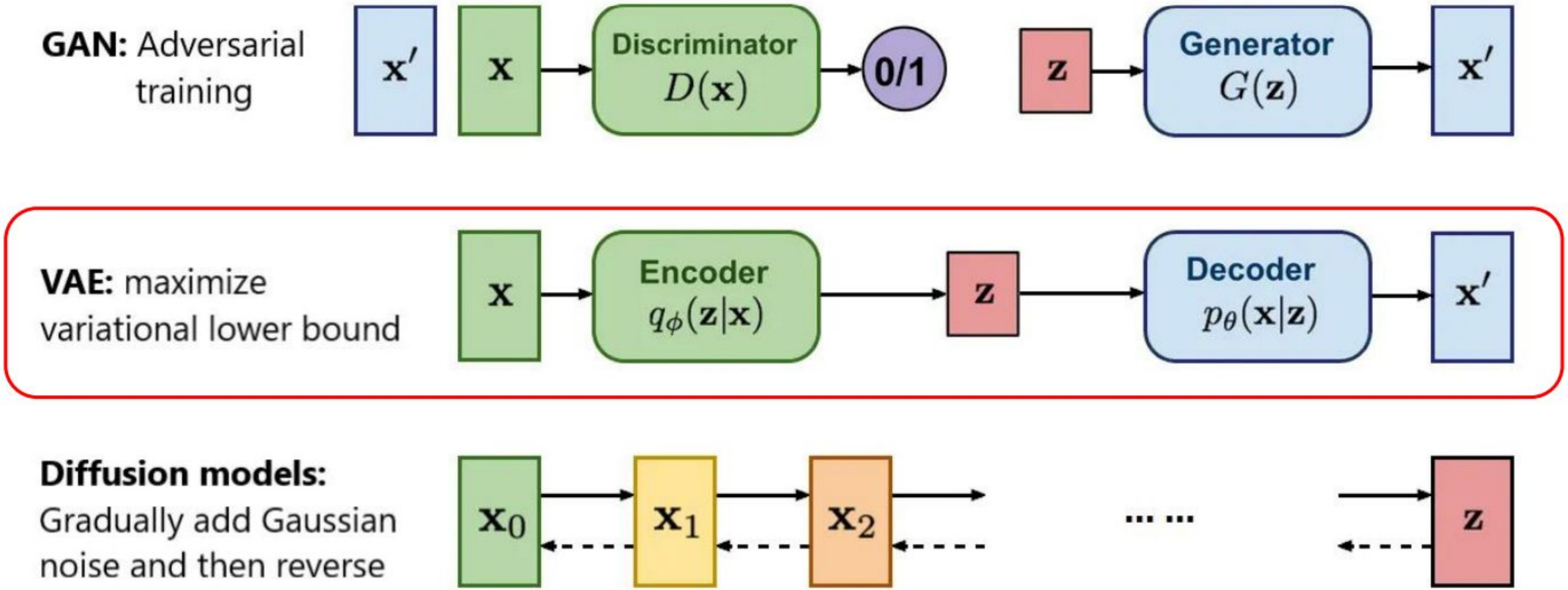
Input Image



Hourglass CNN with Skip Connections

Prediction

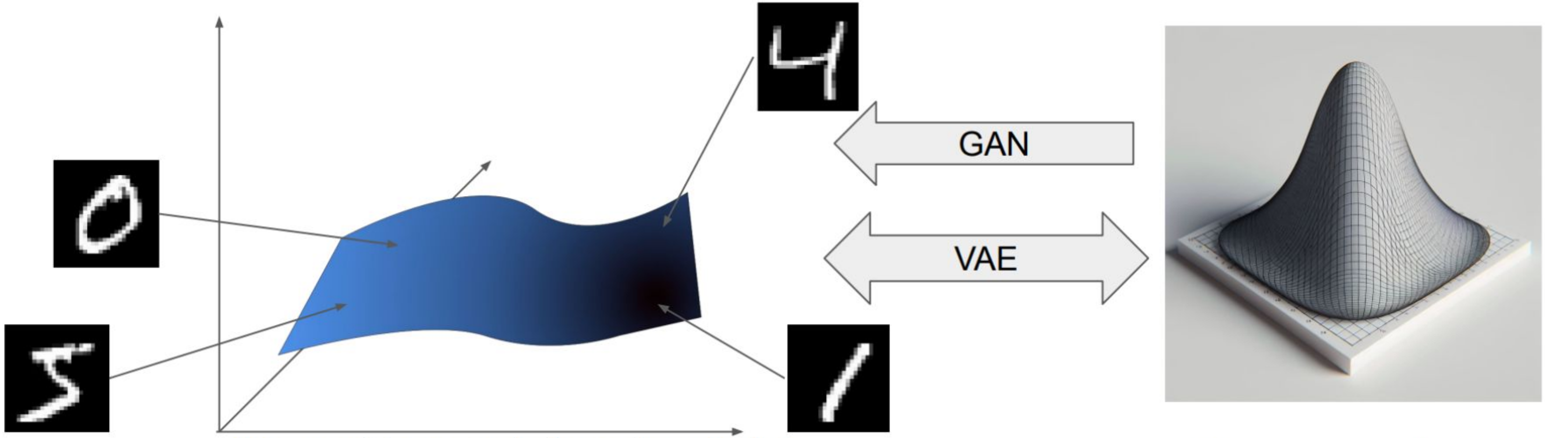
VAEs



GAN: Adversarial training
x'
x
Discriminator
D(x)
0/1
z
Generator
G(z)
x'
VAE: maximize variational lower bound
x
Encoder
qφ(z|x)
z
Decoder
pθ(x|z)
x'
Diffusion models:
Gradually add Gaussian noise and then reverse
x0
x1
x2
z

Data Manifold

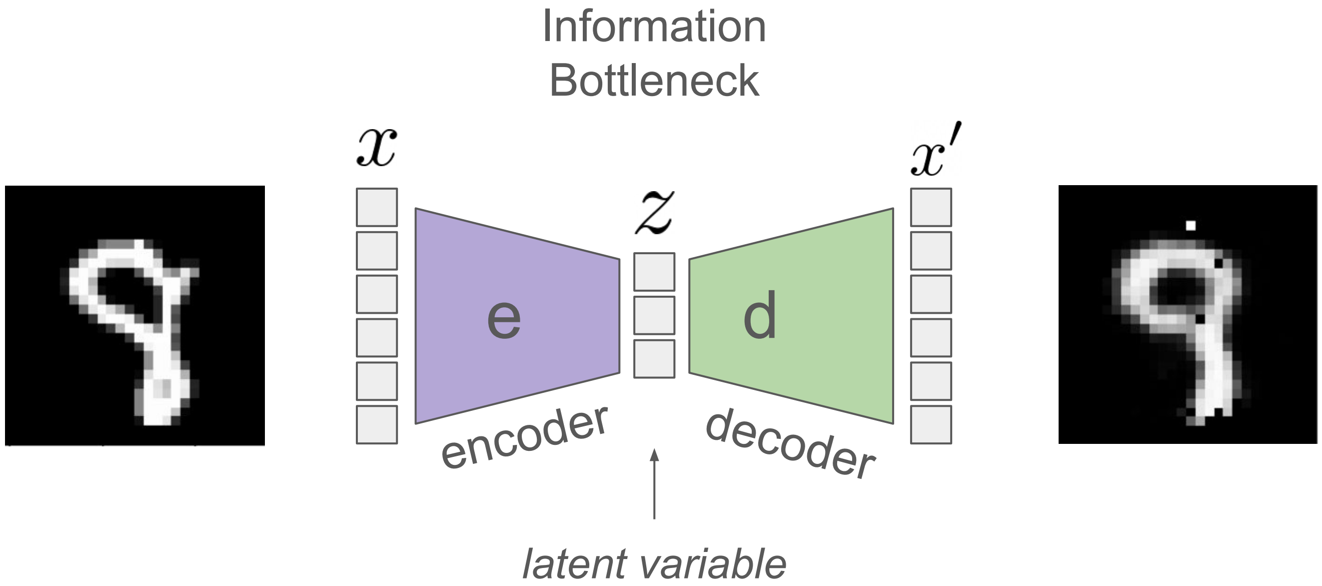
- Data distribution **P(x)** defines a low-dimensional manifold
- Naive random sampling in this space will almost certainly not be on the manifold



We want to sample from the elusive Image Manifold

We **can** sample from a Gaussian Distribution

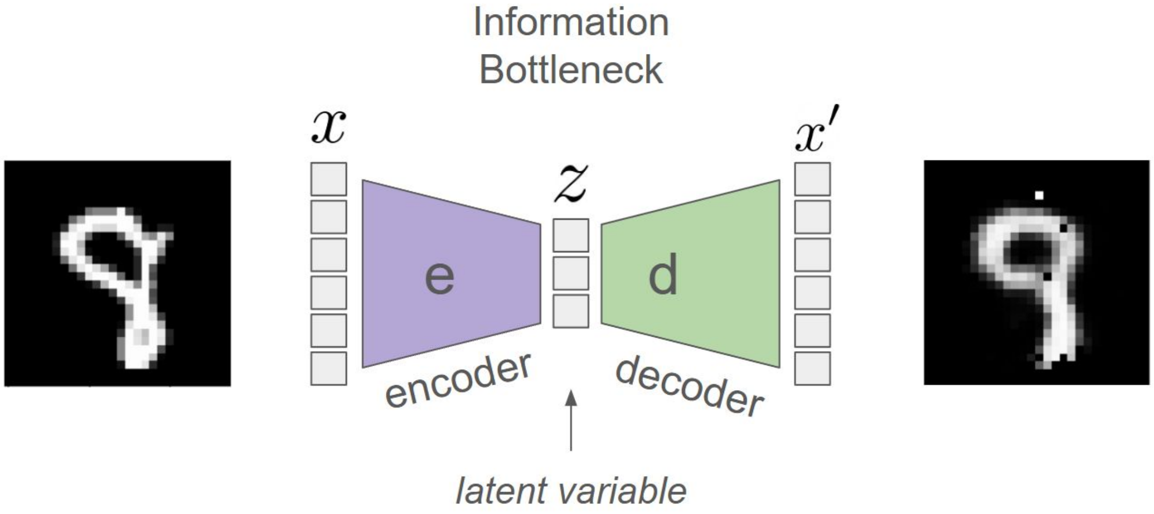
Autoencoders



We typically use MSE or MAE to compute reconstruction loss

Autoencoders -> VAEs

Q: We want to generate images - how can we sample from the latent space?

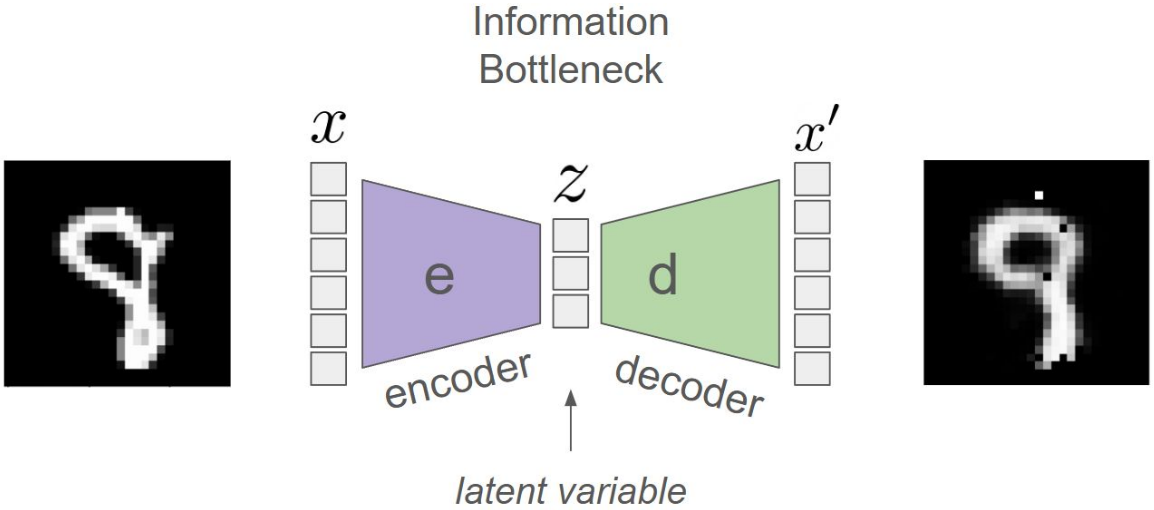


Autoencoders -> VAEs

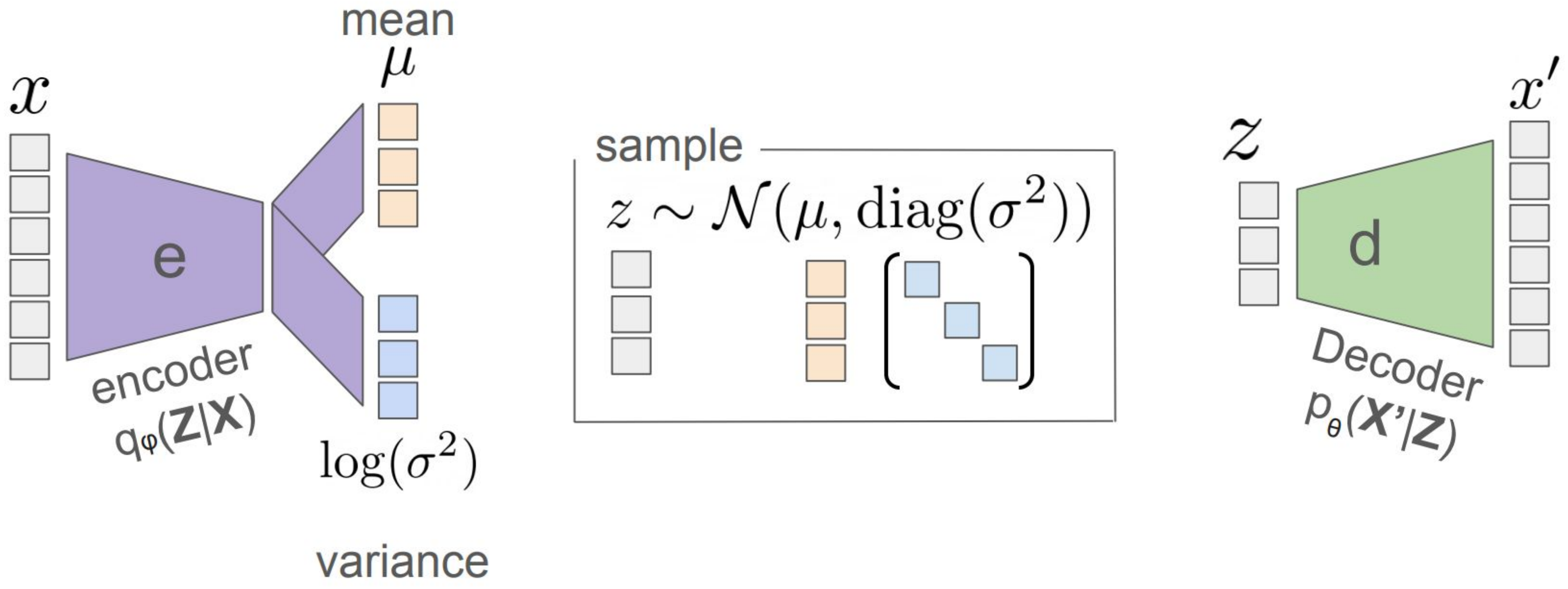
Q: We want to generate images - how can we sample from the latent space?
A: Add regularization to encourage the latent space to approximate a Gaussian.

Foreshadowing ELBO…

$$\underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}))}_{\text{prior matching term}}$$



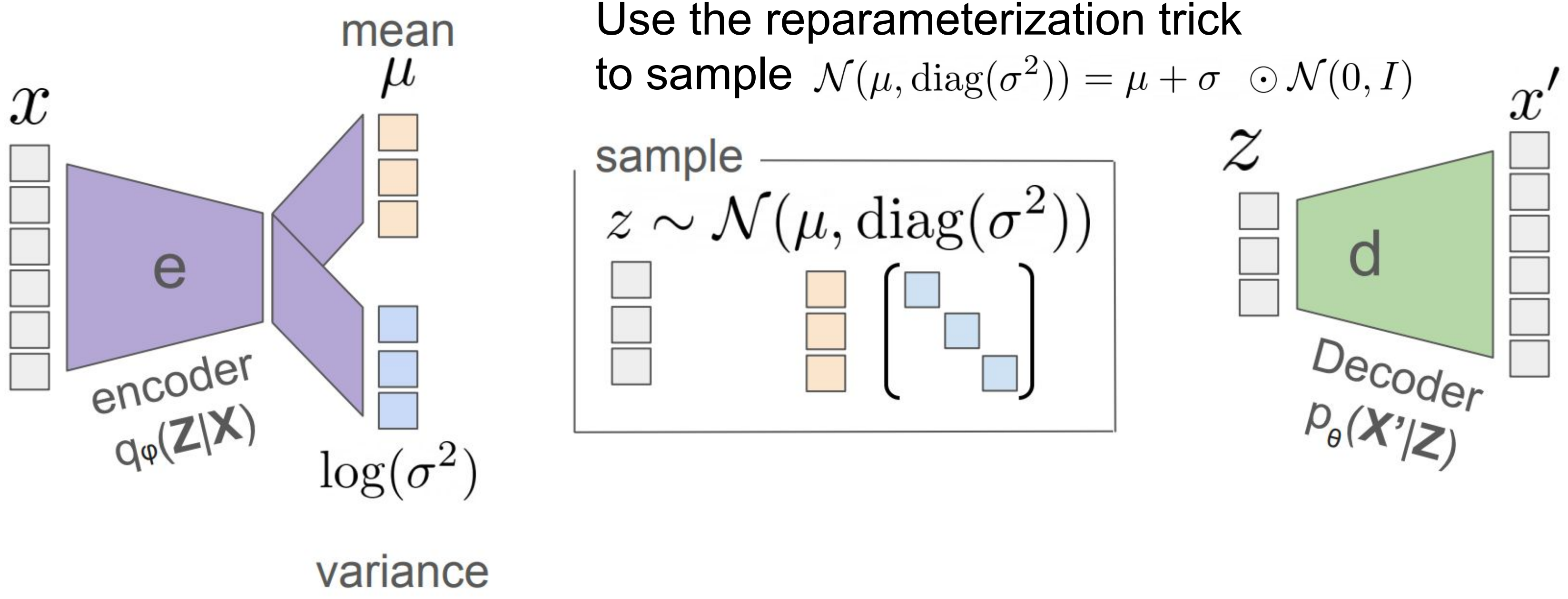
Probabilistic **Encoder** (Gaussian)



$$\max_{\phi,\theta} \mathbb{E}_{z \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log (p_\theta(\mathbf{x}|\mathbf{z}))]$$

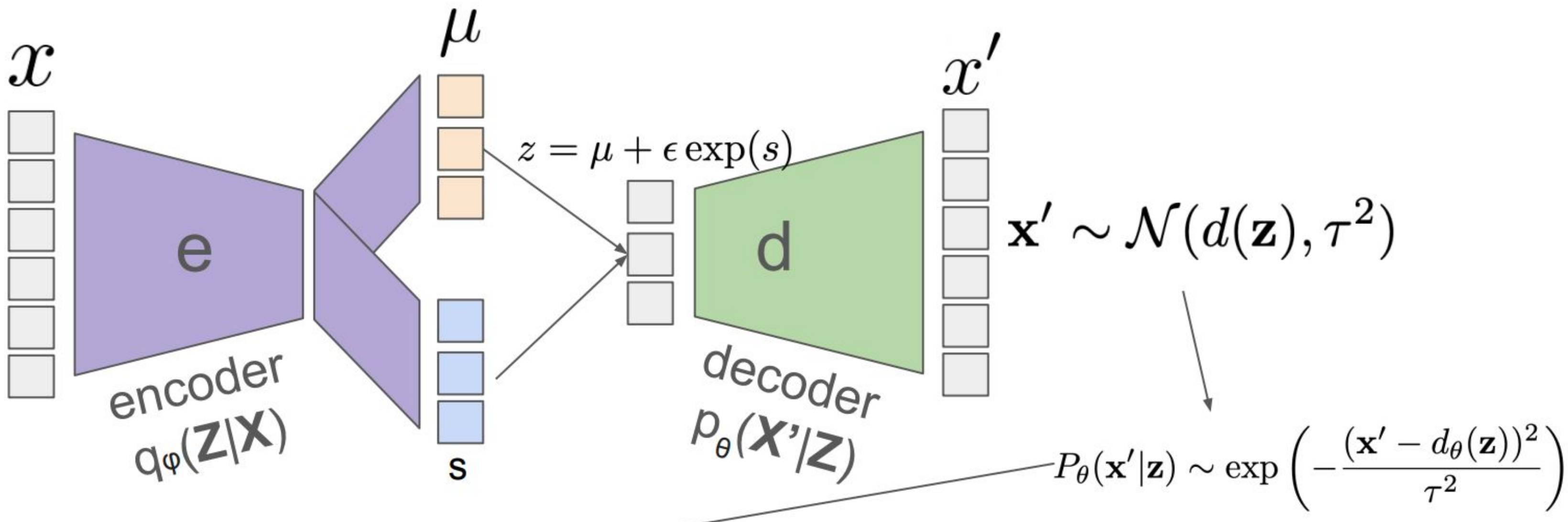
Probabilistic **Encoder** (Gaussian)

Use the reparameterization trick
to sample $\mathcal{N}(\mu, \mathrm{diag}(\sigma^2)) = \mu + \sigma \odot \mathcal{N}(0, I)$



$$\max_{\phi,\theta} \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log(p_\theta(\mathbf{x}|\mathbf{z}))]$$

Probabilistic **decoder** (Gaussian)

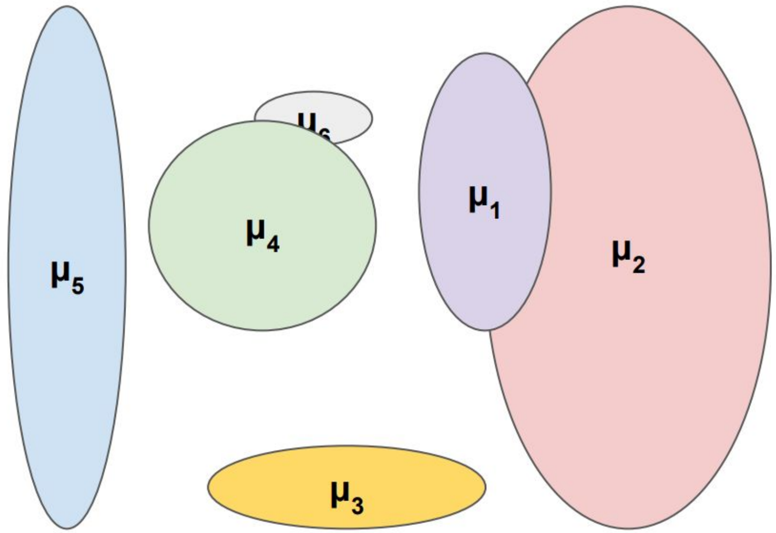


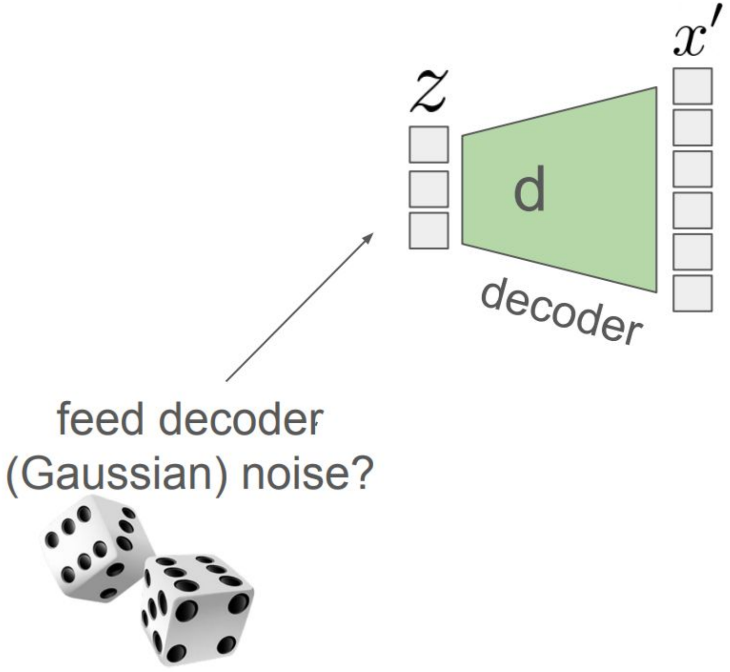
$$\max_{\phi,\theta} \mathbb{E}_{z \sim q_\phi(\mathbf{z}|\mathbf{x})}[\log(p_\theta(\mathbf{x}|\mathbf{z}))] = \min_{\phi,\theta} \mathbb{E}_{z \sim q_\phi(\mathbf{z}|\mathbf{x})}\left[(\mathbf{x} - d_\theta(\mathbf{z}))^2\right]$$

Upshot: Reconstruction formulation results in squared loss

Sampling

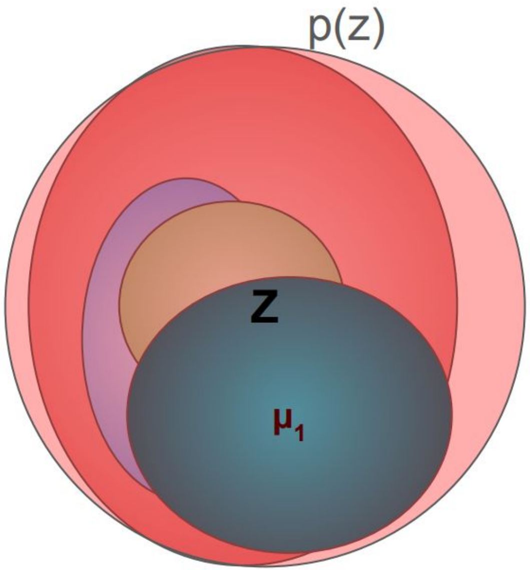
How can we sample, if each sample has its own latent distribution?





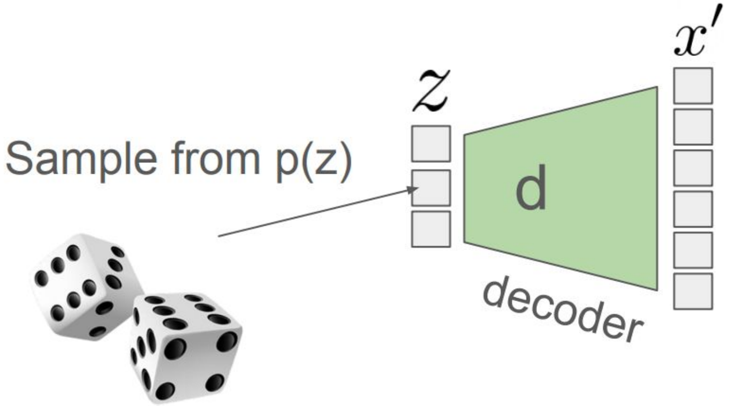
Sampling

Solution: Regularize all distributions to be close to the standard normal ***N(0;I)*.**



maximize

$$\underbrace{\mathbb{E}_{q_{\phi}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q_{\phi}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}))}_{\text{prior matching term}}$$



ELBO

$$\begin{aligned}
\log p(\boldsymbol{x}) &= \log p(\boldsymbol{x}) \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) dz && \text{(Multiply by } 1 = \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) d\boldsymbol{z}\text{)} \\
&= \int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})(\log p(\boldsymbol{x})) dz && \text{(Bring evidence into integral)} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x})\right] && \text{(Definition of Expectation)} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{p(\boldsymbol{z}|\boldsymbol{x})}\right] && \text{(Apply Equation 2)} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z}) q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}|\boldsymbol{x}) q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] && \text{(Multiply by } 1 = \tfrac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\text{)} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z}|\boldsymbol{x})}\right] && \text{(Split the Expectation)} \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] + D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}|\boldsymbol{x})) && \text{(Definition of KL Divergence)} \\
&\geq \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] && \text{(KL Divergence always} \geq 0\text{)}
\end{aligned}$$

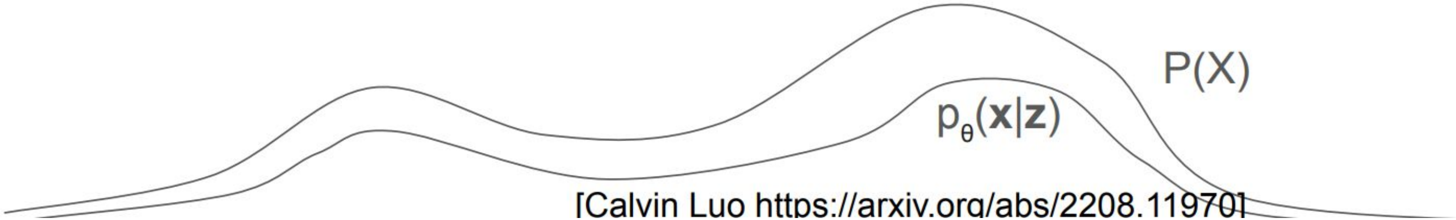
[Calvin Luo https://arxiv.org/abs/2208.11970]

ELBO

$$\begin{aligned}
\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] && \text{(Chain Rule of Probability)}\\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] && \text{(Split the Expectation)}\\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}))}_{\text{prior matching term}} && \text{(Definition of KL Divergence)}
\end{aligned}$$

(We are **maximizing** this lower bound.)

If we maximize $p_\theta(\mathbf{x}|\mathbf{z})$ and minimize the D_{KL} we get close to $P(\mathbf{x})$.



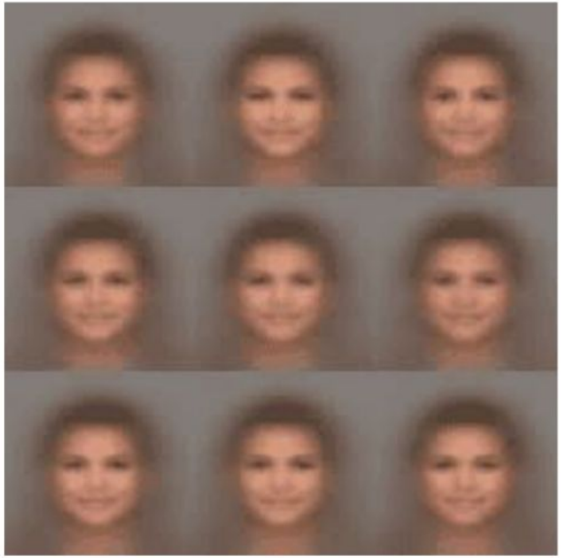
[Calvin Luo https://arxiv.org/abs/2208.11970]

Drawbacks

- Out of the box, generated images can be blurry.

Question: Why? How do GANs fix this problem?

VAE v. GAN



https://borisburkov.net/2022-12-31-1/

Summary

- Generative Image models learn a mapping from the **Standard Normal Gaussian** to the **Image Manifold**
  - GANs learn this through a **discriminator**.
  - VAEs learn it through **variational autoencoders**
- AutoEncoders learn to **compress** and **reconstruct** data
- VAEs make these AutoEncoders **probabilistic**
  - Minimize the **reconstruction loss**
  - Latent space is sampled from Gaussian distributions
  - Sampling is made differentiable with the **Reparameterization Trick**
  - Deviations from the Prior (Standard Normal Gaussian) is penalized by **KL divergence**
- The ELBO is a **lower bound** of P(X)
  - Maximizing the ELBO, and minimizing the KL divergence makes P(x|z) close to P(x)

Prelim Problem (Only focus on VAEs)

For each of the following statements, indicate whether the statement describes VAEs, GANs, or Diffusion models. You should list all that apply.

1. Trained with a likelihood-based objective.
2. Has a learnable encoder that maps the data to a Gaussian distribution.
3. Has latent variables that must be the same size as the input data.
4. Learns to generate images by fooling a discriminator.
5. Generates high-quality, realistic images.
6. Generates images with a single forward pass of the network.
7. Often suffers from poor diversity among generations.
8. Transforms samples from a unit normal distribution to samples from the data distribution.

Prelim Problem (Only focus on VAEs)

For each of the following statements, indicate whether the statement describes VAEs, GANs, or Diffusion models. You should list all that apply.

1. Trained with a likelihood-based objective.

   Y

2. Has a learnable encoder that maps the data to a Gaussian distribution.

   Y

3. Has latent variables that must be the same size as the input data.

   N

4. Learns to generate images by fooling a discriminator.

   N

5. Generates high-quality, realistic images.

   N

6. Generates images with a single forward pass of the network.

   Y

7. Often suffers from poor diversity among generations.

   N

8. Transforms samples from a unit normal distribution to samples from the data distribution.

   Y

Prelim Problem: ELBO

In class, we saw that the ELBO is given as:

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] = \log p(\boldsymbol{x}) - D_{\mathrm{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}|\boldsymbol{x})),$$

where the left-hand-side is the ELBO, $\mathbf{x}$ represents the observed data, and $\mathbf{z}$ represents the latent variables.

a) Show that the ELBO is a lower bound of the log-likelihood of the data.

$$\begin{aligned}\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] &= \log p(\boldsymbol{x}) - D_{\mathrm{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}|\boldsymbol{x})) \\ &\leq \log p(\boldsymbol{x}) && \text{(Non-negativity of KL)}\end{aligned}$$

Prelim Problem: ELBO

b) Rearrange the ELBO to show that that

$$\mathbb{E}_{q_\phi(z|x)}[\log \frac{p(x,z)}{q_\phi(z|x)}] = E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)).$$

Recall that the KL-Divergence is defined as:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right]$$

$$\log p(x) \geq \mathbb{E}_{q_\phi(z|x)}[\log \frac{p(x,z)}{q_\phi(z|x)}]$$

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

$$= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathbb{E}_{q_\phi(z|x)}[\log \frac{p(z)}{q_\phi(z|x)}]$$

$$= E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z))$$

Prelim Problem: ELBO

c) Interpret the two terms in the final ELBO expression in the context of VAEs. What effect does each term have? Write 2-3 sentences.

d) Consider a situation where your variational distribution perfectly matches the true posterior. In other words, $q_\phi(\boldsymbol{z}|\boldsymbol{x}) = p(\boldsymbol{z}|\boldsymbol{x})$. Given this, can you simplify the the ELBO further? Recall that the ELBO is given as

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] = \log p(\boldsymbol{x}) - D_{\mathrm{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}|\boldsymbol{x})).$$

If your variational distribution matches the true posterior, what does maximizing the ELBO accomplish?

Prelim Problem: ELBO

c) Interpret the two terms in the final ELBO expression in the context of VAEs. What effect does each term have? Write 2-3 sentences.

The KL divergence term helps regularize the distribution to be close to the prior distribution. The first term is the expected reconstruction error.

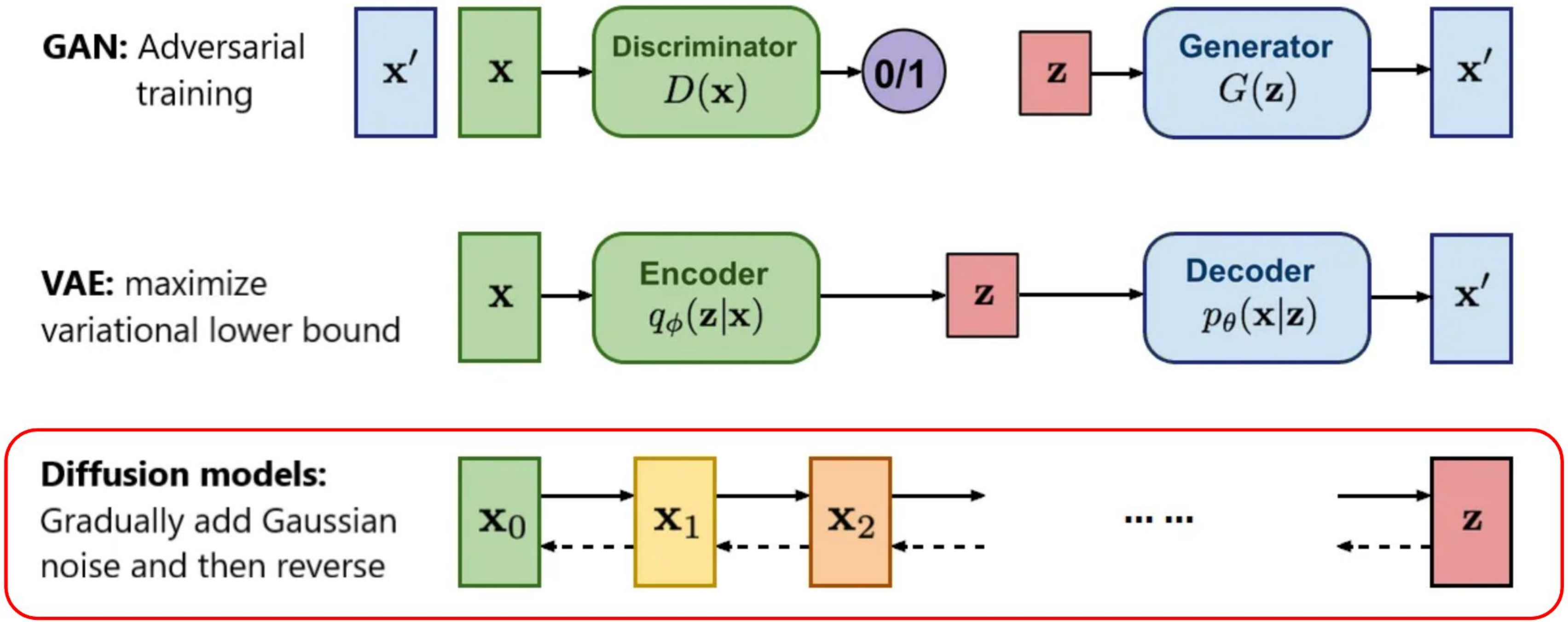
d) Consider a situation where your variational distribution perfectly matches the true posterior. In other words, $q_\phi(\boldsymbol{z}|\boldsymbol{x}) = p(\boldsymbol{z}|\boldsymbol{x})$. Given this, can you simplify the the ELBO further? Recall that the ELBO is given as

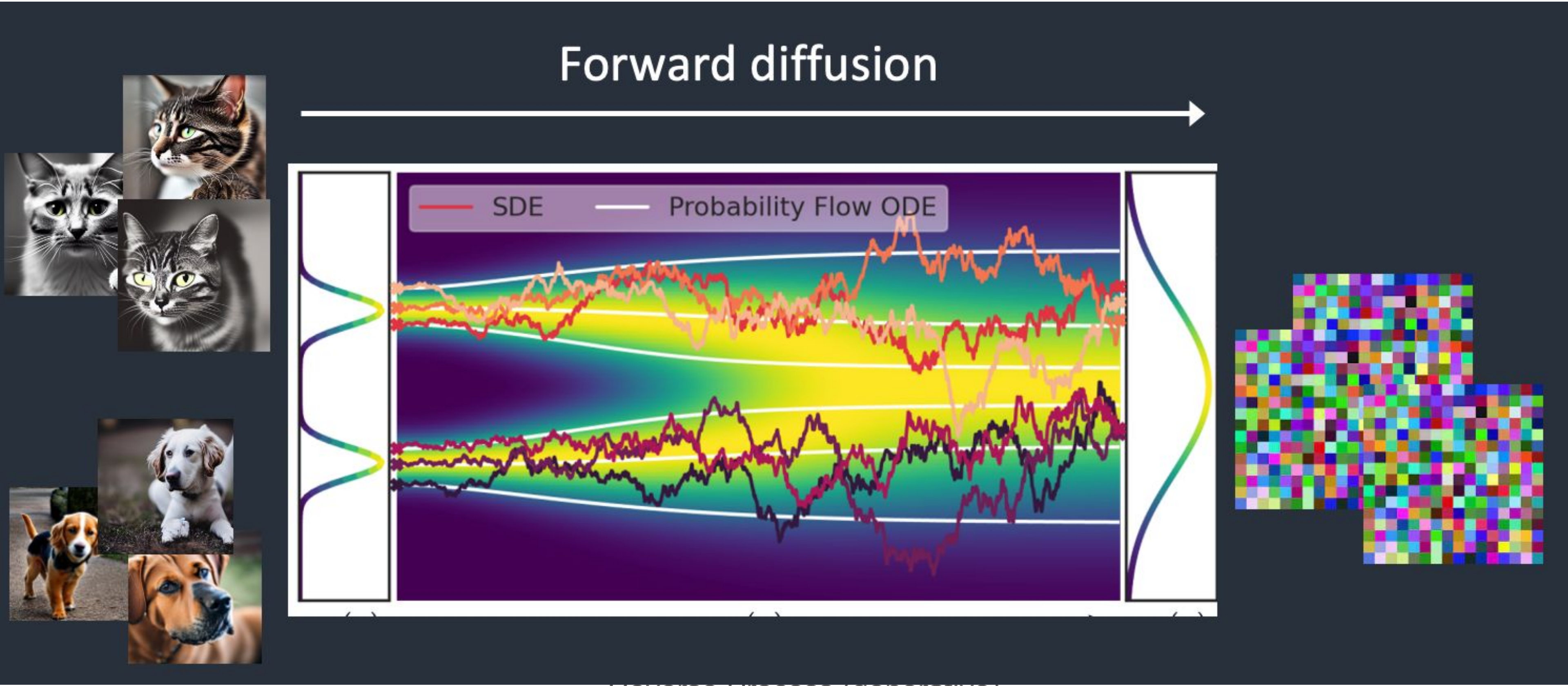
$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] = \log p(\boldsymbol{x}) - D_{\mathrm{KL}}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}|\boldsymbol{x})).$$

If your variational distribution matches the true posterior, what does maximizing the ELBO accomplish?

When the variational distribution perfectly matches the true posterior, the KL divergence term in the ELBO becomes zero since $D_{\mathrm{KL}}(p(\boldsymbol{z}|\boldsymbol{x}) \parallel p(\boldsymbol{z}|\boldsymbol{x})) = 0$. Therefore, the ELBO simplifies to the log-likelihood of the data, $\log p(\boldsymbol{x})$. Maximizing the ELBO is then equivalent to maximizing the log-likelihood of the data.

Diffusion Models





Reverse Process (generative)

Diffusion Models - Main point

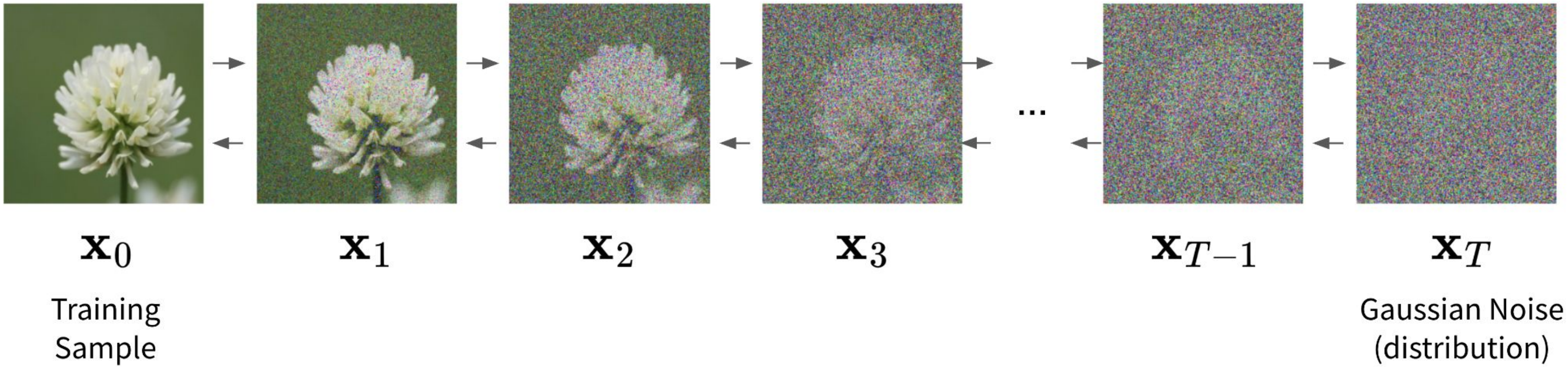
We DEFINE a mapping between a source distribution and Gaussian Distribution (Gaussian Noise)

We LEARN a *reverse* mapping from the Gaussian Distribution to the source distribution.

Forward Process

- “Destroying Data”
- Adding noise forms a **Markov Chain** → **Markov Property**
- **Recall:**

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{x}_0) = q(\mathbf{x}_t | \mathbf{x}_{t-1})$$



Forward Process

- Given a *sampling schedule,* predict the noisy image at timestep **t** from timestep **0**:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})$$

- Usually, we can afford a larger update step when the sample gets noisier so:

$$\bar{\alpha}_1 > \cdots > \bar{\alpha}_T.$$

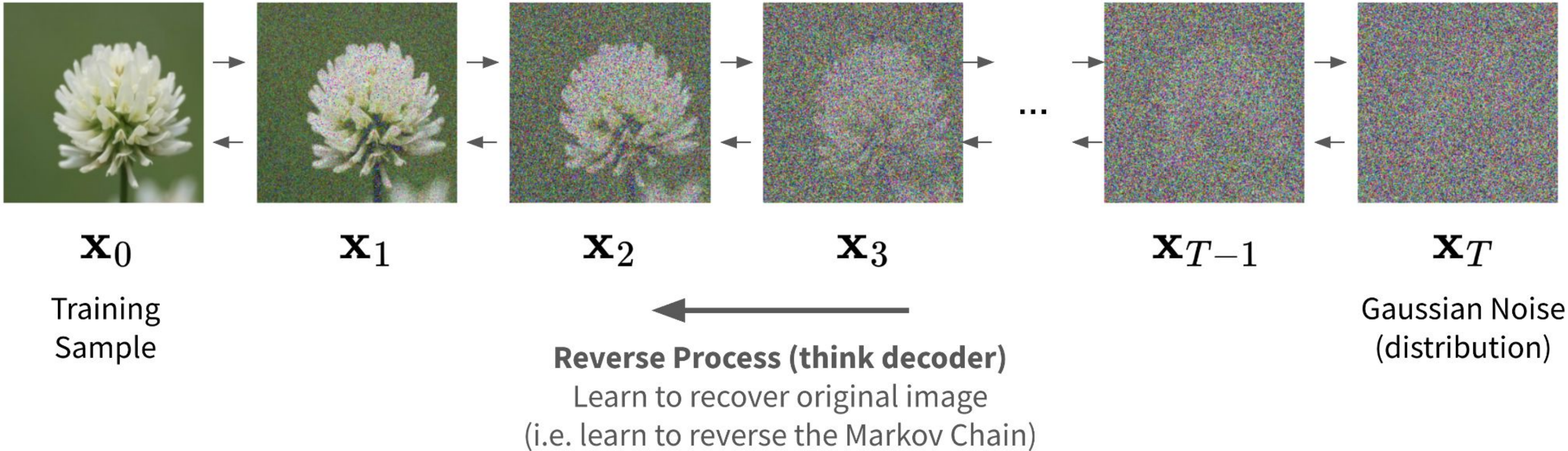
- Want:

$$q(\mathbf{x}_T) \approx \mathcal{N}(0, \mathbf{I})$$

Backward Process

Goal: given an image that was "*noised*" for **t** steps, predict the original image

- Markov chains are not always invertible.
- We ***learn*** the ***inverse (Markov) process***



How do we sample from reverse?

- Sure, try Bayes?
- There's a **problem…**
- **Intractable:** $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1})}{q(\mathbf{x}_t)}$$

$$q(\mathbf{x}_t) = \int q(\mathbf{x}_0)q(\mathbf{x}_1|\mathbf{x}_0)...q(\mathbf{x}_t|\mathbf{x}_{t-1})d\mathbf{x}_0 d\mathbf{x}_1 ... d\mathbf{x}_{t-1}$$

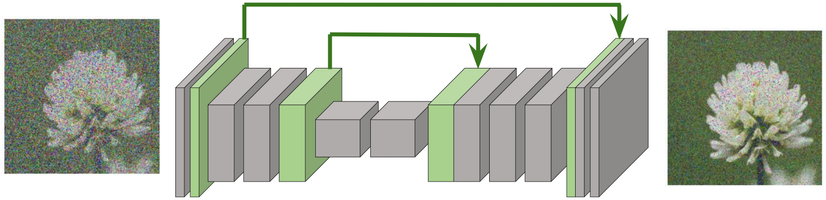
Learn the reverse

We have $\mathbf{x}_0$ during **training**; train a **generative model**

$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is **tractable**

$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$$

$$\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))\right]$$



How do we actually train for the reverse process?

Find the model that **maximizes the likelihood** of the training data

$$\max \ \log p(\boldsymbol{x})$$

$$\log p(\boldsymbol{x}) \geq \underbrace{\mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)\right]}_{\text{reconstruction term}} - \underbrace{D_{\text{KL}}(q(\boldsymbol{x}_T|\boldsymbol{x}_0) \parallel p(\boldsymbol{x}_T))}_{\text{prior matching term}} - \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[D_{\text{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))\right]}_{\text{denoising matching term}}$$

Find the model that maximizes the likelihood of the training data

. $\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[\mathcal{D}_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) \mid\mid p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t))\right]$ is a *denoising matching term*. We learn desired denoising transition step $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)$ as an approximation to tractable, ground-truth denoising transition step $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$. The $q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0)$ transition step can act as a ground-truth signal, since it defines how to denoise a noisy image $\boldsymbol{x}_t$ with access to what the final, completely denoised image $\boldsymbol{x}_0$ should be. This term is therefore minimized when the two denoising steps match as closely as possible, as measured by their KL Divergence.

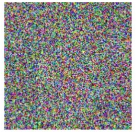
$$\log p(\boldsymbol{x}) \geq \underbrace{\mathbb{E}_{q(\boldsymbol{x}_1|\boldsymbol{x}_0)}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}_0|\boldsymbol{x}_1)\right]}_{\text{reconstruction term}} - \underbrace{D_{\mathrm{KL}}(q(\boldsymbol{x}_T|\boldsymbol{x}_0) \parallel p(\boldsymbol{x}_T))}_{\text{prior matching term}} - \sum_{t=2} \underbrace{\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))\right]}_{\text{denoising matching term}}$$

With large **T,** the prior matching goes to 0

Reparametrization of the noise prediction

$\mu_\theta(\mathbf{x}_t, t)$

$\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$



Recall that we need to learn a neural network to approximate

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Want to train $\boldsymbol{\mu}_\theta$ to get:

$$\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\Big)$$

Since we have **x_t** during training, we can reparametrize to predict $\boldsymbol{\epsilon}_t$

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big)$$

$$\text{Thus } \mathbf{x}_{t-1} = \mathcal{N}(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Reparametrization of the Loss

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)$$

$$\text{Thus } \mathbf{x}_{t-1} = \mathcal{N}\left(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)\right)$$

$$- \sum_{t=2}^{T} \underbrace{\mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[D_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t))\right]}_{\text{denoising matching term}} \longrightarrow L(\theta) = \mathbb{E}_{t,\mathbf{x}_0,\epsilon}\left[||\epsilon - \epsilon_\theta(\mathbf{x}_t, t)||^2\right]$$

Loss is **MSE** of actual predicted loss!

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \mathbf{x}_\theta(\mathbf{x}_t, t)) \approx q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \mathbf{x}_0)$$

Training Algo

Repeat until convergence

1. $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ ← Sample original image from image distribution
2. $t \sim U\{1, 2, \ldots, T\}$ ← Sample random time step uniformly
3. $\epsilon \sim \mathcal{N}(0, 1)$ ← Sample Gaussian noise
4. Optimizer step on $L(\theta) = \mathbb{E}_{t,\mathbf{x}_0,\epsilon}[||\epsilon - \epsilon_\theta(\mathbf{x}_t, t)||^2]$

← Model predicts noise applied at time step t and calculate loss

Sampling Algo

$$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$ ← Sample pure Gaussian noise

For $t = T, T-1 \ldots, 1$

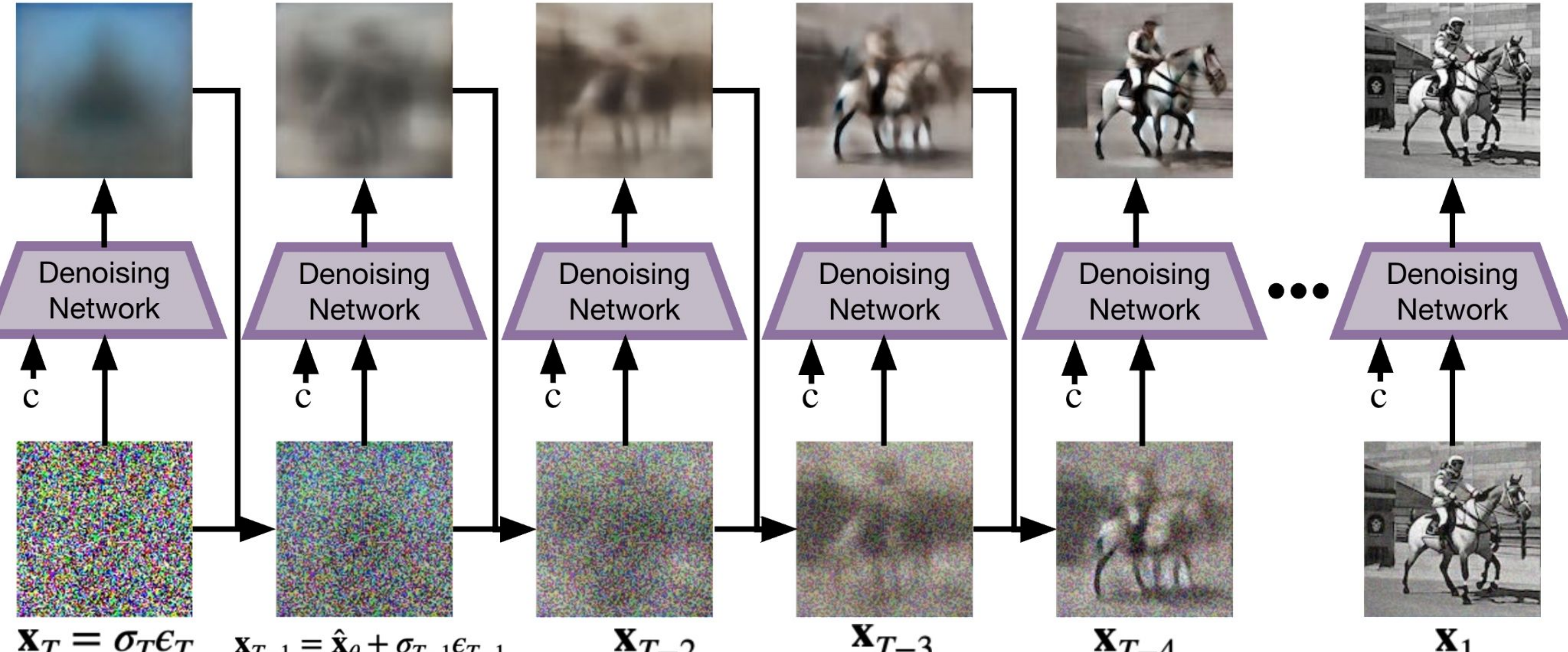
$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \text{ if } t > 1, \text{ else } \mathbf{z} = \mathbf{0}$$ ← Sample Gaussian noise to apply to image

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$$ ← Predict noise applied to image and remove that noise

Return $\mathbf{x}_0$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \mathbf{x}_\theta(\mathbf{x}_t, t))$$

Diffusion Sampling



Alternative Perspective

So far:

Denoising Diffusion Probabilistic Models

Alternative:

Generative Modeling by Estimating Gradients of the Data Distribution

Score-based Models

Langevin dynamics allow you to **sample** from distribution - **even if it is not normalized.**

Would like to model the probability density function:

$$p_\theta(\mathbf{x}) = \frac{e^{-f_\theta(\mathbf{x})}}{Z_\theta}, \text{ where } Z_\theta > 0 \text{ is a normalizing constant s.t. } \int p_\theta(\mathbf{x})d\mathbf{x} = 1$$

Still want to **maximize the log-likelihood**

$$\max_\theta \sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i).$$

Problem: Normalization constant intractable → Approximate the score function:

$$\mathbf{s}_\theta(\mathbf{x}) = \nabla_\mathbf{x} \log p_\theta(\mathbf{x}) = -\nabla_\mathbf{x} f_\theta(\mathbf{x}) - \underbrace{\nabla_\mathbf{x} \log Z_\theta}_{=0} = -\nabla_\mathbf{x} f_\theta(\mathbf{x})$$

What is the score anyway?

- What direction in data space to move in order to further increase its likelihood

Loss?

$$
\begin{aligned}
& \arg\min_{\boldsymbol{\theta}} \mathcal{D}_{\mathrm{KL}}(q(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, \boldsymbol{x}_0) \,||\, p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t)) \\
= & \arg\min_{\boldsymbol{\theta}} \mathcal{D}_{\mathrm{KL}}\left(\mathcal{N}\left(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q\left(t\right)\right) \,||\, \mathcal{N}\left(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\Sigma}_q\left(t\right)\right)\right) & (83) \\
= & \arg\min_{\boldsymbol{\theta}} \frac{1}{2\sigma_q^2(t)} \frac{(1-\alpha_t)^2}{\alpha_t}\left[\left\|\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) - \nabla \log p(\boldsymbol{x}_t)\right\|_2^2\right] & (84)
\end{aligned}
$$

The gradient of x in dataspace, for arbitrary noise level t

Add noise for more accurate scores

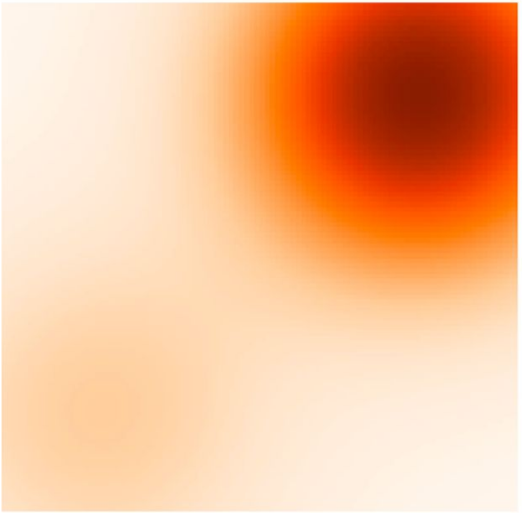
Fisher Divergence

Predicted score!

unknown!

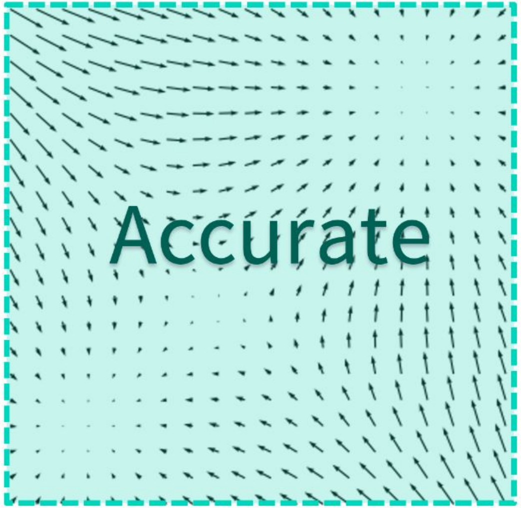
$$\mathbb{E}_{\mathbf{x}}\left[\|\mathbf{s}_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p(x)\|_2^2\right]$$

Perturbed density



Perturbed scores

Estimated scores



Training

Training Objective for noise level t:

$$\sum_{t=1}^{T} \lambda(t)\mathbb{E}_{\mathbf{x},t}[||\mathbf{s}_\theta(\mathbf{x_t}, t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x_t})||_2^2]$$

Using results from denoising score matching [1]:

$$\sum_{t=1}^{T} \lambda(t)\mathbb{E}_{\mathbf{x},t}[||\mathbf{s}_\theta(\tilde{\mathbf{x}}, t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t|\mathbf{x})||_2^2]$$

Using the definition of the pdf of a Gaussian,

$$\sum_{t=1}^{T} \lambda(t)\mathbb{E}_{\mathbf{x},t}[||\mathbf{s}_\theta(\mathbf{x}_t, t) - \frac{\mathbf{x}_t - \mathbf{x}}{\sigma_t^2}||_2^2]$$

Looks familiar?

Okay, saw we learned this cool and fancy score function, what now?

- Just sample through Langevin dynamics!
- starting at any *arbitrary point* in the same space, iteratively follow the score until a mode is reached :)

Conditioning? Guidance?

- Just condition at each step!

$$p(\boldsymbol{x}_{0:T} \mid y) = p(\boldsymbol{x}_T) \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1} \mid \boldsymbol{x}_t, y)$$

$$\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t, y) \approx \nabla \log p(\boldsymbol{x}_t \mid y)$$

Conditional Diffusion — Classifier Guidance

- Use Bayes' rule to decompose the conditional score into the unconditional score a a likelihood term

$$\nabla \log p(\boldsymbol{x}_t \mid y) = \nabla \log \left(\frac{p(\boldsymbol{x}_t) p(y \mid \boldsymbol{x}_t)}{p(y)} \right)$$
$$= \nabla \log p(\boldsymbol{x}_t) + \nabla \log p(y \mid \boldsymbol{x}_t) - \nabla \log p(y)$$
$$= \underbrace{\nabla \log p(\boldsymbol{x}_t)}_{\text{unconditional score}} + \underbrace{\nabla \log p(y \mid \boldsymbol{x}_t)}_{\text{adversarial gradient}}$$

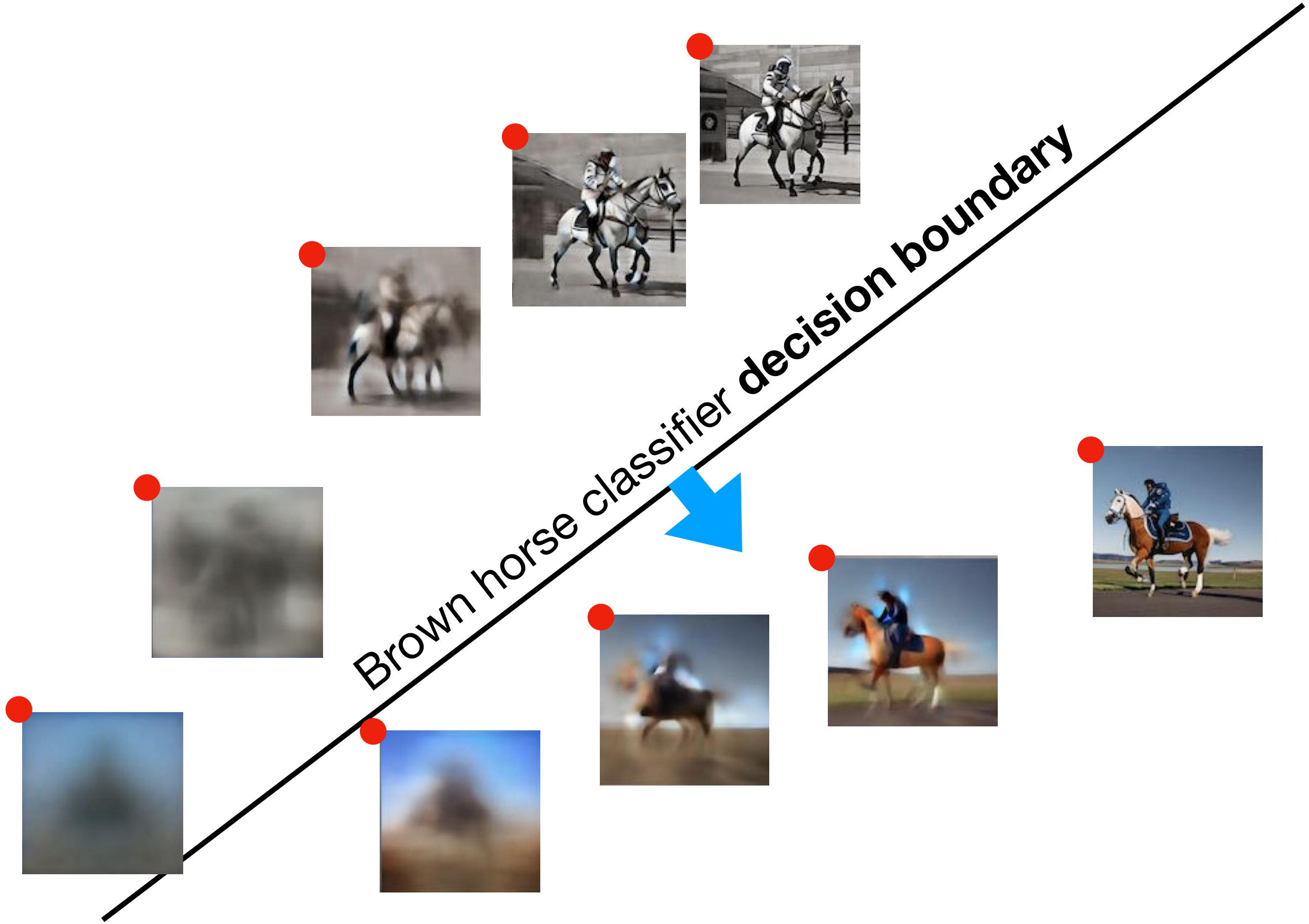
- Only need to train a classifier on noised data
- Use classifier to guide noise!

Can add \gamma

add noise

$$\mathbf{x}_{t-1} = \hat{\mathbf{x}}_\theta(\mathbf{x}_t) + \sigma_{t-1}\epsilon_{t-1} + \alpha \nabla_{\mathbf{x}_t} \log \left(P(y \mid \mathbf{x}_t) \right)$$

de-noise **guide noise**


Brown horse classifier **decision boundary**

Classifier-Free Guidance

- Train a joint conditional and unconditional diffusion model
- Conditioning information is added by concatenating to input or cross attending
- Modified conditional distribution

Randomly drops the condition during training and linearly combines the condition and unconditional output during sampling

$$\log p_t(\mathbf{x}_t|\mathbf{y}) \propto p_t(\mathbf{x}_t|\mathbf{y})\, p_t(\mathbf{y}|\mathbf{x}_t)^{\omega}$$

- Conditional sampling

$$\nabla_{\mathbf{x}_t} \log \tilde{p}_t(\mathbf{x}_t|\mathbf{y}) = \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) + w(\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{y}) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t))$$

Significantly improves quality of conditional models, but decreases diversity

Equivalence of the two views

We have therefore derived three equivalent objectives to optimize a VDM: learning a neural network to predict the original image $\boldsymbol{x}_0$, the source noise $\boldsymbol{\epsilon}_0$, or the score of the image at an arbitrary noise level $\nabla \log p(\boldsymbol{x}_t)$ [2, 10]. The VDM can be scalably trained by stochastically sampling timesteps t and minimizing the norm of the prediction with the ground truth target.

Problem 2

For each of the following statements, indicate whether the statement describes VAEs, GANs, or Diffusion models. You should list all that apply.

1. Trained with a likelihood-based objective.
2. Has a learnable encoder that maps the data to a Gaussian distribution.
3. Has latent variables that must be the same size as the input data.
4. Learns to generate images by fooling a discriminator.
5. Generates high-quality, realistic images.
6. Generates images with a single forward pass of the network.
7. Often suffers from poor diversity among generations.
8. Transforms samples from a unit normal distribution to samples from the data distribution.

Problem 2

For each of the following statements, indicate whether the statement describes VAEs, GANs, or Diffusion models. You should list all that apply.

1. Trained with a likelihood-based objective.

Y

2. Has a learnable encoder that maps the data to a Gaussian distribution.

N

3. Has latent variables that must be the same size as the input data.

Y

4. Learns to generate images by fooling a discriminator.

N

5. Generates high-quality, realistic images.

Y

6. Generates images with a single forward pass of the network.

N

7. Often suffers from poor diversity among generations.

N

8. Transforms samples from a unit normal distribution to samples from the data distribution.

Y

9. CLIP is a constrastive learning algorithm that learns to map paired text and images close together in a shared embedding space.

10. Vision transformers typically require less training data than convolutional neural networks (CNNs) to achieve comparable performance on image classification tasks.

9. CLIP is a constrastive learning algorithm that learns to map paired text and images close together in a shared embedding space.

Solution: True.

10. Vision transformers typically require less training data than convolutional neural networks (CNNs) to achieve comparable performance on image classification tasks.

Solution: False. Vision transformers often require larger amounts of training data compared to CNNs to achieve comparable performance on image classification tasks. CNNs have inductive biases such as translation invariance and local receptive fields, which allow them to learn effective features with less training data. However, when trained on sufficiently large datasets, vision transformers have shown impressive performance and have even outperformed CNNs on various benchmarks.

Problem 9: Image Captioning [9 pts]

Adhitya wants to train a model for image captioning. He plans to train a decoder-only transformer model that generates a natural language description by attending to visual features extracted from an input image. He has access to a pretrained ViT model to extract image features. Adhitya has written most of the network architecute and needs a little help with the cross-attention layer in the decoder (the cross-attention layer is used to attend to the image features).

a) Adhitya has already written code to divide the input images of shape $(h \times w \times 3)$ into k patches and feed the patches to a ViT to produce 128-dimensional features for every patch. The hidden dimension of the decoder is 128. What are the dimensions of the weight matrices W_Q, W_K, W_V in the cross-attention layer?

Problem 9: Image Captioning [9 pts]

Adhitya wants to train a model for image captioning. He plans to train a decoder-only transformer model that generates a natural language description by attending to visual features extracted from an input image. He has access to a pretrained ViT model to extract image features. Adhitya has written most of the network architecute and needs a little help with the cross-attention layer in the decoder (the cross-attention layer is used to attend to the image features).

a) Adhitya has already written code to divide the input images of shape $(h \times w \times 3)$ into k patches and feed the patches to a ViT to produce 128-dimensional features for every patch. The hidden dimension of the decoder is 128. What are the dimensions of the weight matrices W_Q, W_K, W_V in the cross-attention layer?

The dimensions of the weight matrices W_Q, W_K, W_V in the cross-attention layer are:
$W_Q : (128, 128)$ $W_K : (128, 128)$ $W_V : (128, 128)$

Problem 9: Image Captioning [9 pts]

Adhitya wants to train a model for image captioning. He plans to train a decoder-only transformer model that generates a natural language description by attending to visual features extracted from an input image. He has access to a pretrained ViT model to extract image features. Adhitya has written most of the network architecute and needs a little help with the cross-attention layer in the decoder (the cross-attention layer is used to attend to the image features).

b) When you are predicting the first word, what are the shapes of the output matrices obtained after multiplying W_Q, W_K, W_V with their appropriate inputs.

Problem 9: Image Captioning [9 pts]

Adhitya wants to train a model for image captioning. He plans to train a decoder-only transformer model that generates a natural language description by attending to visual features extracted from an input image. He has access to a pretrained ViT model to extract image features. Adhitya has written most of the network architecute and needs a little help with the cross-attention layer in the decoder (the cross-attention layer is used to attend to the image features).

b) When you are predicting the first word, what are the shapes of the output matrices obtained after multiplying W_Q, W_K, W_V with their appropriate inputs.

The shapes of the output matrices after multiplying W_Q, W_K, W_V with their appropriate inputs are:

$Q : (1, 128)$ - The query matrix is obtained by multiplying the decoder's input embedding (or the previous layer's output) with W_Q. $K : (k, 128)$ - The key matrix is obtained by multiplying the ViT's output features with W_K. $V : (k, 128)$ - The value matrix is obtained by multiplying the ViT's output features with W_V.

Problem 9: Image Captioning [9 pts]

Adhitya wants to train a model for image captioning. He plans to train a decoder-only transformer model that generates a natural language description by attending to visual features extracted from an input image. He has access to a pretrained ViT model to extract image features. Adhitya has written most of the network architecute and needs a little help with the cross-attention layer in the decoder (the cross-attention layer is used to attend to the image features).

c) Suppose the ViT model produced 64-dimensional features for every patch (instead of 128-dimeansional features), what would you have to modify to attend to these image features?

Problem 9: Image Captioning [9 pts]

Adhitya wants to train a model for image captioning. He plans to train a decoder-only transformer model that generates a natural language description by attending to visual features extracted from an input image. He has access to a pretrained ViT model to extract image features. Adhitya has written most of the network architecute and needs a little help with the cross-attention layer in the decoder (the cross-attention layer is used to attend to the image features).

c) Suppose the ViT model produced 64-dimensional features for every patch (instead of 128-dimeansional features), what would you have to modify to attend to these image features?

Add a linear layer to project the embeddings to 128 dimensions.