CS598PS – Machine Learning for Signal Processing

Privacy-Preserving Machine Learning

(+ more by request)

24 November 2020

Today's lecture

- Secure Multiparty Computations
  - Some cryptography basics
  - Applying SMC to MLSP
- Some items by request
  - Generative models
  - Reinforcement learning
  - Attention/Transformers
  - Others ...

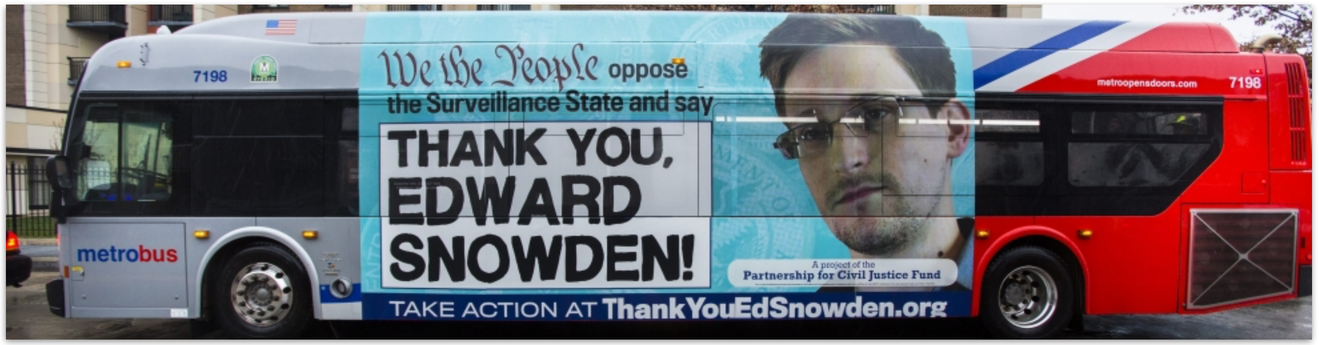
A problem

- Machine learning can be a great service
  - Give me your data, I'll give you some insight

- But, you can't ask everyone for their data
  - Sending your voice mail for transcription?
  - Camera feed from your house for security?
  - Sharing medical data with your phone?

What would be nice

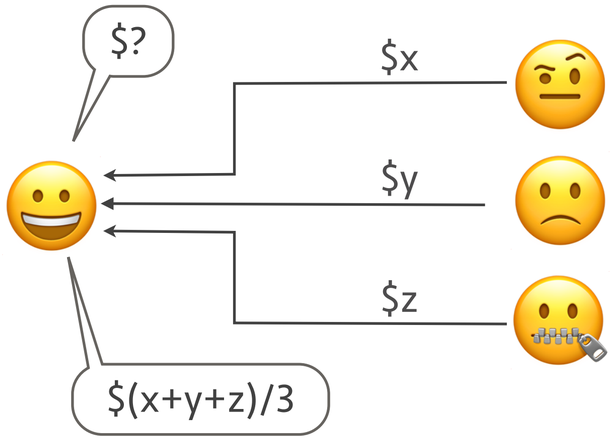
- In a perfect world there would be absolute trust
  - But do you really trust anyone with your data?



- In the real world
  - We want to ensure the privacy of our data
  - Others want to ensure privacy of their algorithms & data
    - Can these constraints co-exist?

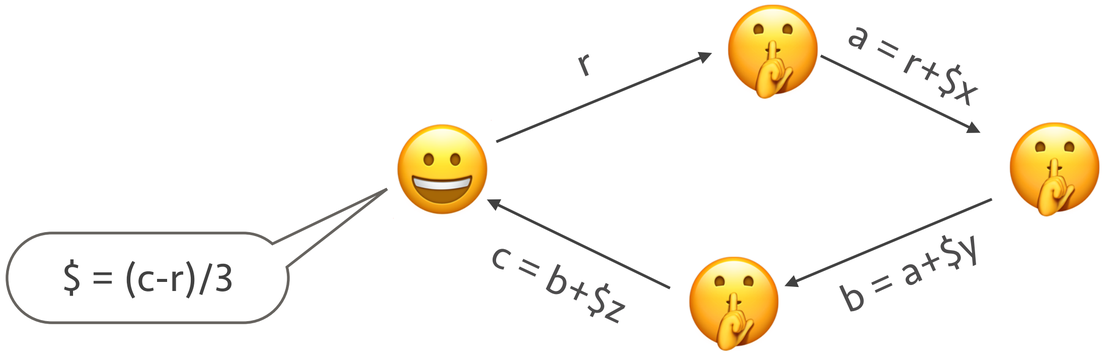
Starting simple

- Doing some statistics with private data
- E.g. finding the average person's salary in the room?
  - The unacceptable way: Ask everyone for their income



A more socially acceptable approach

- Obfuscate dollar amounts with noise
  - Give a random number to next person, ask them to add their salary, pass the sum to the next person, and repeat
  - I get back sum of salaries plus my known random number
    - Remove random number and divide by number of people!
    - No private data has been shared!

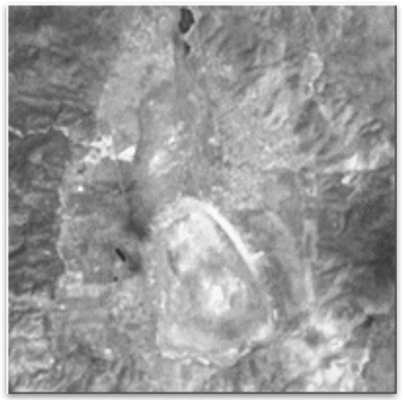


A signal-friendly example

- Your government wants you to examine the differences between satellite images over time
  - But the images are classified

Before

After

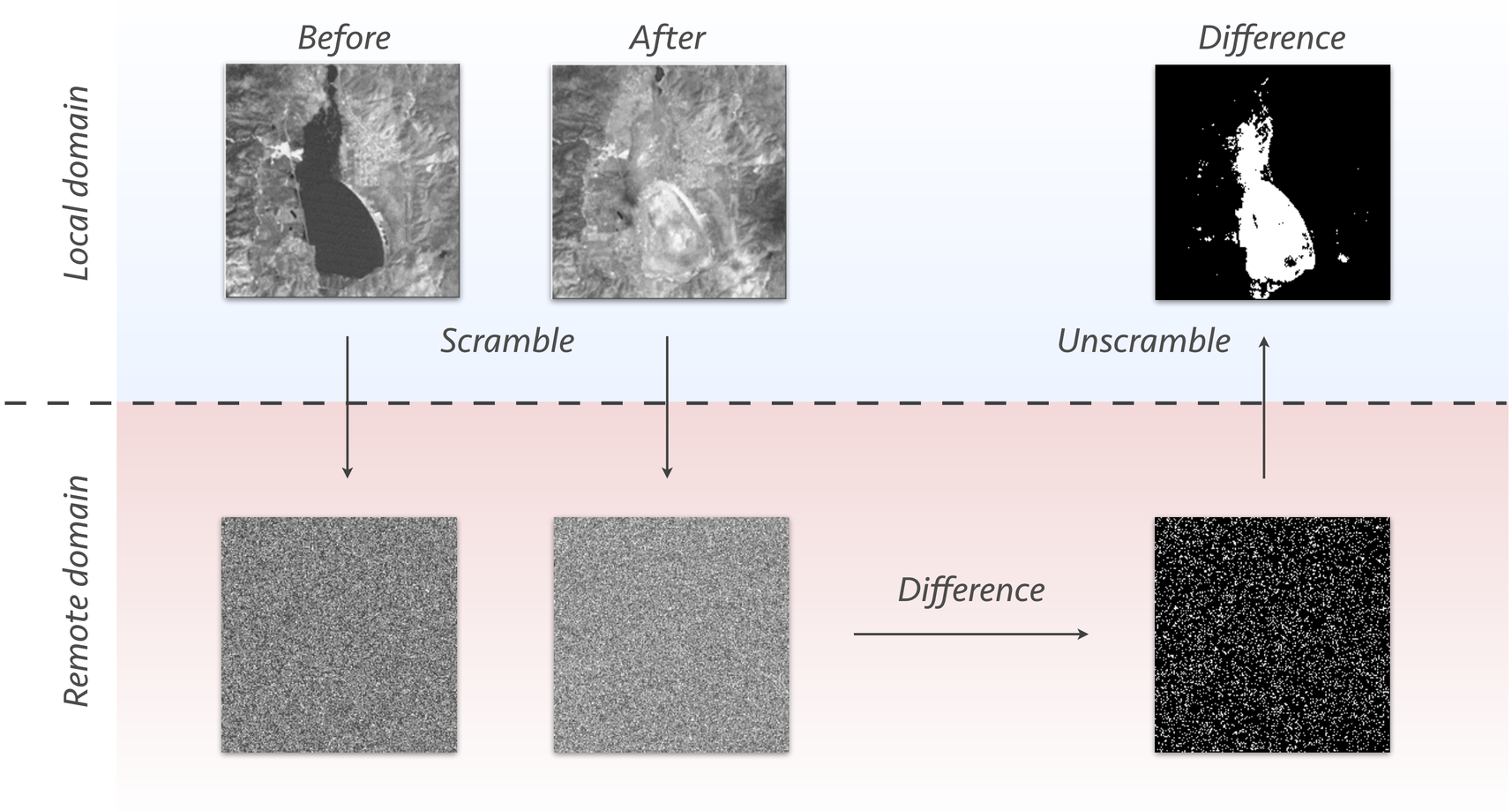


Difference

  - How do we do this one?

Using noise to obfuscate the data

- Permute the pixels before sending them for processing!



Basic processing primitives

- The vast majority of MLSP uses only a few operations
  - Dot product, greater than, maxarg, …
- If we define secure versions of these, then we can construct more complex MLSP operations
  - While maintaining data privacy

The setup

- We will use two collaborators: Alice and Bob
  - They are not malicious, but curious
    - i.e. Won't go out of their way to cheat, but are happy to inspect the data
  - There is no trusted 3rd party
    - i.e. they cannot trust a third person to mediate
- Both have private data that they do not want to share
  - But they need each other's data to perform a computation
- They do not have infinite computing resources
  - Otherwise they could easily cheat

Multiple scenario cases

- Blind transaction
  - Alice wants to protect her data; Bob cannot see the data, but can see the process output
    - Even if Bob or a third party hacker is malicious they can't deduce the data
- Double-blind transaction
  - Alice wants to protect her data and the identity of the results; Bob cannot see the data nor the process output
    - Even if Bob or a third party hacker is malicious he can't deduce anything
- But there's more
  - Alice is malicious and wants to send specific data to reverse engineer Bob's algorithm
  - Alice has data which needs processing, and Bob's result can be sent over to Carl who analyses blindly to sent to Darren who then …

A simple exchange for now

- Alice has vector $\mathbf{x}$, Bob has vector $\mathbf{w}$ and threshold θ
  - They do not want to share their data with each other
  - They want to compute whether $\mathbf{w}^\top \mathbf{x} > \theta$

- Why this operation?
  - It is the core operation for most classifiers (more later)

A cryptography diversion

- The Paillier cryptosystem
  - Provides an encoding: $y = \mathrm{E}[x]$ and it's inverse: $x = \mathrm{E}^{-1}[y]$
    - x is referred to as the *plaintext* and y as the *ciphertext*
- Has a hugely useful property
  - $\mathrm{E}[x]\mathrm{E}[y] = \mathrm{E}[x+y]$
  - This is known as *homomorphic encryption*
    - An operation on ciphertext corresponds to an operation in plaintext
    - Many other cryptosystems with similar properties

Some details

- Encryption: $c = \mathrm{E}[x] = g^x\, r^n \bmod n^2$, $x \in \mathbb{Z}_n$, $r \in \mathbb{Z}^*_n$
  - $n = p\, q$, where p, q are equal length primes
  - g is a random integer $\mathbb{Z}^*_{n^2}$, r is a random integer $\mathbb{Z}^*_n$
  - Public key: $\{n, g\}$
- Decryption: $x = \mathrm{E}[c] = \mathrm{L}(c^\lambda \bmod n^2) m \bmod n$
  - $\lambda = \mathrm{lcd}(p-1, q-1)$, $m = \mathrm{L}(g^\lambda \bmod n^2)^{-1} \bmod n$, $\mathrm{L}(x) = (x-1)/n$
  - Private key is: $\{\lambda, m\}$

Homomorphic properties

- Adding plaintexts via ciphertexts
  - $\mathrm{E}^{-1}[\ \mathrm{E}[x_1]\ \mathrm{E}[x_2] \bmod n^2\ ] = x_1 + x_2 \bmod n$
  - $\mathrm{E}^{-1}[\ \mathrm{E}[x_1]\ g^{x_2} \bmod n^2\ ] = x_1 + x_2 \bmod n$

- Multiplying plaintexts via ciphertexts
  - $\mathrm{E}^{-1}[\ \mathrm{E}[x_1]^{x_2} \bmod n^2] = x_1 x_2 \bmod n$
  - $\mathrm{E}^{-1}[\ \mathrm{E}[x_2]^{x_1} \bmod n^2] = x_1 x_2 \bmod n$

Using Paillier for a secure inner product

- Desired operation:
  - Alice has vector $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$
  - Bob has vector $\mathbf{w} = \{w_1, w_2, \ldots, w_N\}$
  - Wanted outcome: Bob obtains $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]$

- In the end, Bob should have result but cannot see it

The SIP protocol

- Alice:
  - Generates an $\mathrm{E}[]$ and $\mathrm{E}^{-1}[]$, sends $\mathrm{E}[]$ to Bob
  - Encrypts $\mathbf{x}$ and sends all $\mathrm{E}[x_i]$ to Bob
- Bob:
  - Computes homomorphic element-wise multiplication
    - $\mathrm{E}[x_i]^{w_i} = \mathrm{E}[x_i w_i]$
  - Bob computes homomorphic summation
    - $\prod \mathrm{E}[x_i w_i] = \mathrm{E}[\sum x_i w_i] = \mathrm{E}[\mathbf{w}^\top \mathbf{x}]$

Now we're stuck

- A couple of issues
  - Bob has $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]$ but he can't see the result
    - That's a feature not a bug!
  - Alice can decrypt it, but then she will know $\mathbf{w}$
    - That would be a bug, not a feature!

- To protect Bob's data we need to "mask" the result

Masking

- In *masking/blinding* we obfuscate our data by adding random numbers (as we did with earlier examples)

- In our case we can perform *additive masking*
  - Bob can add a secret random number ρ to $\mathbf{w}^\top \mathbf{x}$
    - $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]\mathrm{E}[\rho] = \mathrm{E}[\mathbf{w}^\top \mathbf{x} + \rho]$

Constructing a classifier

- Use the SIP protocol:
  - Bob masks the obtained dot product: $\mathbf{E}[\mathbf{w}^\top \mathbf{x}]$
  - Bob sends the result $\mathbf{E}[\mathbf{w}^\top \mathbf{x} + \rho]$ to Alice
- Outcome:
  - Alice can obtain $\mathbf{w}^\top \mathbf{x} + \rho$, but has not clue what $\mathbf{w}$ is
  - Bob has no clue what $\mathbf{x}$ is
- To classify we need to compare $\mathbf{w}^\top \mathbf{x} + \rho$ with $\theta + \rho$

Performing a secure comparison

- Yao's millionaires' problem
  - Alice and Bob want to compare their assets, but not to reveal them
- More formally
  - Alice has x dollars
  - Bob has y dollars
  - Alice and Bob only need to find out if $x > y$
    - But Alice can't learn y and Bob can't learn x
- In our case instead of dollars we compare $\mathbf{w}^\top \mathbf{x} + \rho$ with $\theta + \rho$

Representing the values to compare

- Alice & Bob decide on a range of numbers to compare
  - e.g. \$1m, \$2m, ..., \$10m, each value represented by $i = \{1, \dots, 10\}$
- Alice and Bob then have i_a and i_b dollars
  - They need to know if $i_a > i_b$
- A minor issue: This is a discrete and bound set
  - Bad for us since the dot product and threshold are real
    - But we can always quantize

Secure comparison algorithm

Alice's data is red
Bob's data is blue
Shared data is yellow

- Alice sends her public key to Bob
  - Bob can thus compute $\mathrm{E}[]$
- Bob computes $c = \mathrm{E}[x]$
  - x being a random integer of N bits
- Bob transmits to Alice $v = c + 1 - i_b$ ⟵ *This ensures that Bob's number remains private*
  - i_b is Bob's representation of assets

Secure comparison algorithm

Alice's data is red
Bob's data is blue
Shared data is yellow

- Alice generates a set of 10 numbers ← *Because we are considering 10 possible values*
  - $y_{i=1,\ldots,10} = \mathrm{E}^{-1}[v + i - 1]$
  - The number corresponding to y_{i_b} will be equal to x
- Alice generates a random prime p ← *This ensures that we don't leak Alice's private key*
  - And computes $z_i = y_i \bmod p$
    - p is $N/2$-bits and must result in z_i's, that are apart by at least 2
- Alice sends p and $\mathbf{u} = \{z_1, \ldots, z_{i_a-1}, z_{i_a}, 1+z_{i_a+1}, \ldots, 1+z_{10}\}$

Secure comparison algorithm

Alice's data is red
Bob's data is blue
Shared data is yellow

- Bob computes $g = x \bmod p$
  - x was Bob's original random number, p is Alice's prime
- Bob compares g with the i_b'th element of $\mathbf{u}$
  - if $u_{i_b} = g$, then $i_a \geq i_b$
  - Otherwise $i_a < i_b$
- Neither party gets to share their original number!

Back to our original problem

- Alice has data $\mathbf{x}$, Bob has classifier $\{\mathbf{w}, \theta\}$
  - Step 1. Perform secure inner product
    - Bob obtains $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]$
    - Bob sends to Alice $\mathrm{E}[\mathbf{w}^\top \mathbf{x}]\mathrm{E}[\rho] = \mathrm{E}[\mathbf{w}^\top \mathbf{x} + \rho]$
  - Step 2. Perform secure comparison
    - Alice compares $\mathbf{w}^\top \mathbf{x} + \rho$ with Bob's $\theta + \rho$, gets inequality result
- Alice keeps both data and classification outcome private
  - At the expense of extra overhead …

So what we do with this?

- Linear classifiers!
  - These are essentially the previous formula!

- What about more complex classifiers?
  - e.g. a Gaussian likelihood classifier
  - This isn't a linear operation

Making the Gaussian a dot product

- Gaussian log likelihood:

$$\log P(\mathbf{x};\mu,\Sigma) = -\frac{1}{2}\left(\mathbf{x}-\mu\right)^{\top}\Sigma^{-1}\left(\mathbf{x}-\mu\right) - \frac{d}{2}\log 2\pi - \frac{1}{2}\log\left|\Sigma_i\right|$$

- Equivalent to: $g(\mathbf{x}) = \mathbf{x}^{\top}\cdot\mathbf{W}\cdot\mathbf{x} + \mathbf{w}^{\top}\cdot\mathbf{x} + w$
  - Where: $\mathbf{W} = -\frac{1}{2}\Sigma^{-1},\ \mathbf{w} = \Sigma^{-1}\cdot\mu,\ w = -\frac{1}{2}\mu^{\top}\cdot\Sigma^{-1}\cdot\mu - \frac{1}{2}\log\left|\Sigma\right| - \frac{1}{2}\log 2\pi$
- Which can be simplified to: $g(\mathbf{x}) = \tilde{\mathbf{x}}\cdot\tilde{\mathbf{W}}\cdot\tilde{\mathbf{x}}$
  - Concatenate 1 to $\mathbf{x}$ and include $\mathbf{w}$, w into $\mathbf{W}$

One more step

- But that's not a single product (we can do better)
- Instead we can do: $g(\mathbf{x}) = \tilde{\mathbf{W}} \cdot \tilde{\mathbf{x}}$
  - with:

$$\tilde{\mathbf{x}} = [1, x_1, x_2, \cdots, x_1 x_1, x_1 x_2, x_1 x_3, \cdots x_1 x_N, x_2 x_1, x_2 x_2, x_2 x_3, \cdots, x_N x_N]$$

$$\tilde{\mathbf{W}} = \begin{bmatrix} w, & \Sigma^{-1} \cdot \mu, & -\frac{1}{2}\mathrm{vec}\,\Sigma^{-1} \end{bmatrix}, \quad w = -\frac{1}{2}\mu^{\top}\Sigma^{-1}\mu - \frac{1}{2}\log\left|\Sigma\right| - \frac{1}{2}\log 2\pi$$

- Which only needs a simple secure product operation

Stringing protocols together

- It is often easier to modularize a process
  - Coming up with a secure HMM is hard
  - Coming up with a secure Gaussian, followed by a secure sum, followed by a secure transition regularizer, followed by …, is easier
- To do so we can use *additive shares*
  - At each step the output is additively distributed between parties
  - E.g. for SIP: $z+v=\mathrm{SIP}(x, y)$, Alice has x and z, Bob has y an v
  - We can keep processing keeping all intermediate results hidden

But there are some issues …

- Data needs to be integer-valued
  - Not a huge problem, we can quantize data to desired accuracy
  - Can be a problem with, e.g. secure comparison though
- Encryption/decryption is computationally intensive
  - There is work on specialized hardware for this
- Are these worthwhile ideas?
  - Maybe later, email encryption was just as hopeless a while back

More reading material

- Yao's Millionaires' problem:
  - http://research.cs.wisc.edu/areas/sec/Yao1982.pdf
- Secure Multiparty Computations and Data Mining
  - https://eprint.iacr.org/2008/197.pdf
- Full-blown secure HMM implementation for speech:
  - http://paris.cs.illinois.edu/pubs/smaragdis-tasl07-3.pdf

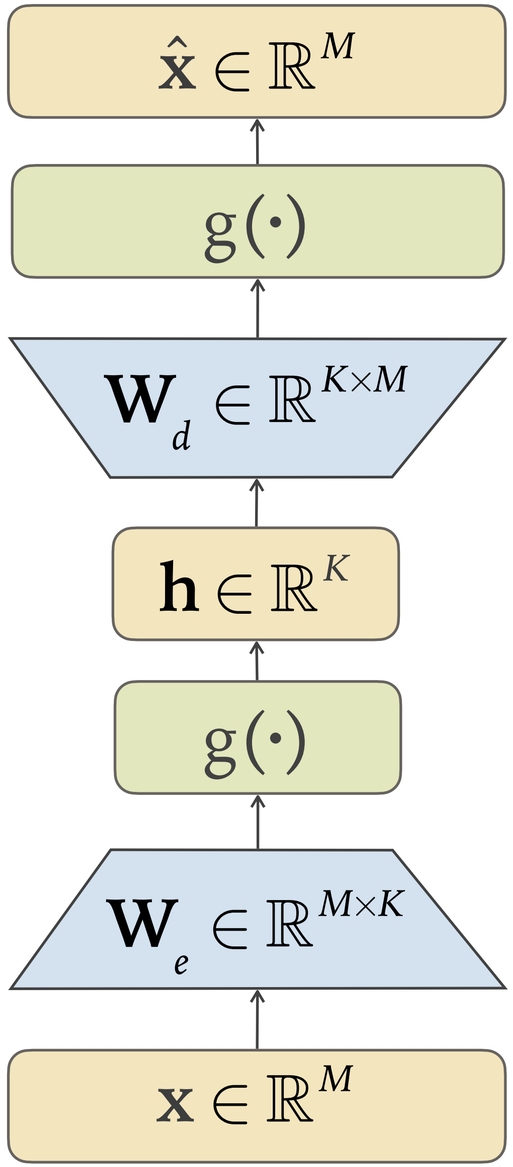
Generative Models

- Generative models are good for synthesizing data
  - E.g., speech synthesis, image generation, videos, etc.

- Two dominant models
  - Variational Auto-Encoders (VAEs)
  - Generative Adversarial Networks (GANs)

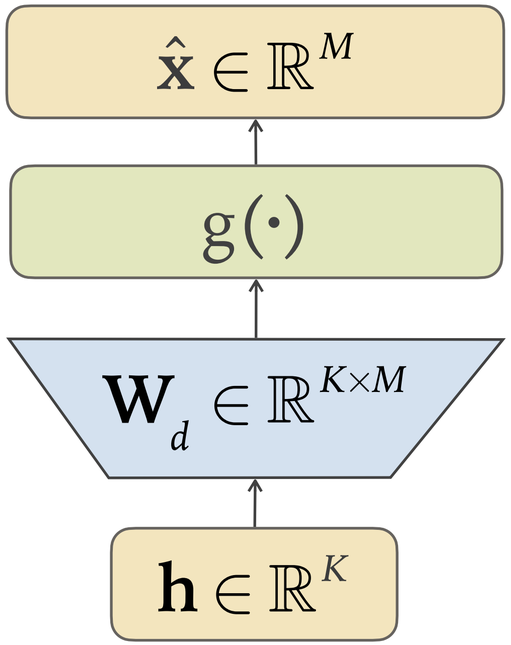
Starting with an autoencoder

- Consider an autoencoder network
  - High-dimensional input $\mathbf{x}$
  - Transformed to a low-dimensional $\mathbf{h}$
    - Via an *encoder* $\mathrm{g}(\mathbf{W}_e \cdot \mathbf{x})$
  - And back to high-dim approximation of input
    - Via a *decoder* $\mathrm{g}(\mathbf{W}_d \cdot \mathbf{h})$
- $\mathbf{h}$ is a low-dim representation of $\mathbf{x}$
  - Like PCA, but not orthogonal



Hallucinating new data

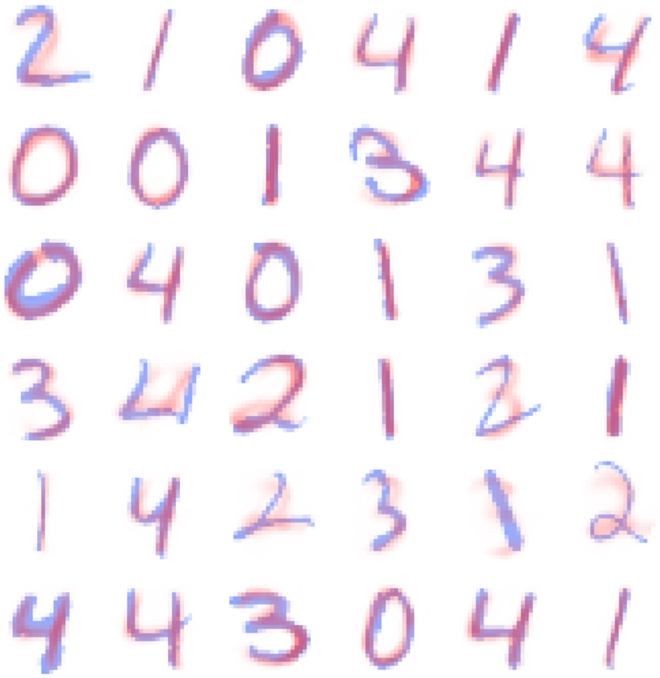
- Trained autoencoder has two parts
  - Encoder: from high-d to low-d
  - Decoder: from low-d to high-d
- The decoder on its own can be used to generate new data
  - Generate random values for $\mathbf{h}$
  - Pass them through decoder



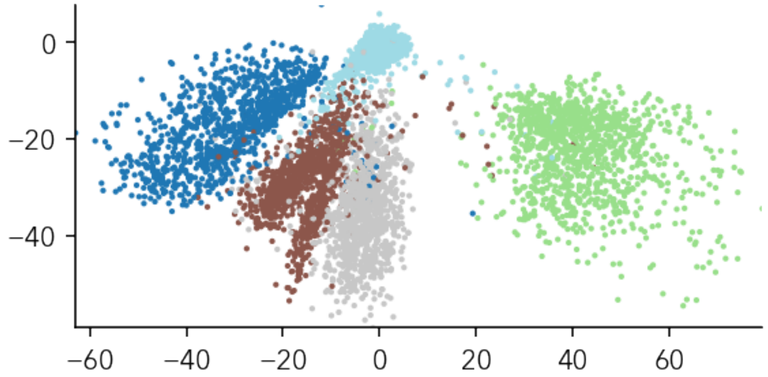
MNIST example

- Train on digits {0,1,2,3,4}, use 2d latent representation
  - Latent dimension is not easy to understand

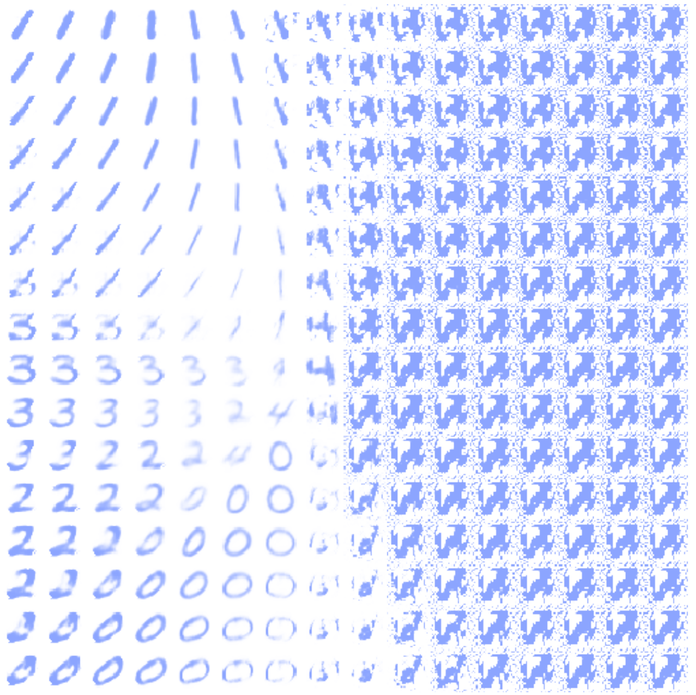
Input/approximations



Latent projection



Latent space

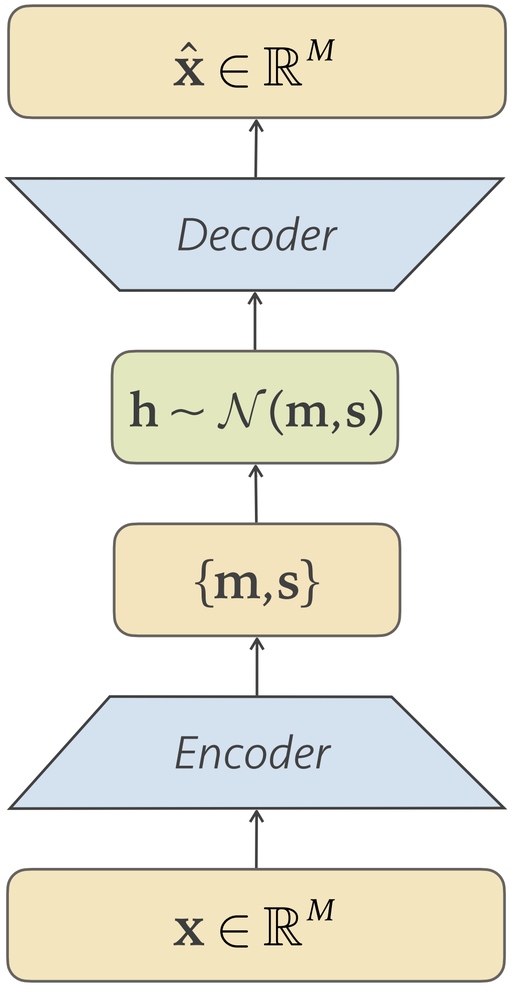


Some problems

- How can we sample plausible $\mathbf{h}$ values?
  - There are gaps which generate garbage data
- How is $\mathbf{h}$ distributed?
  - What range do I sample from?
- What if multiple $\mathbf{x}$'s collapse to a single $\mathbf{h}$?
  - Can I guarantee that the encoder is smooth?

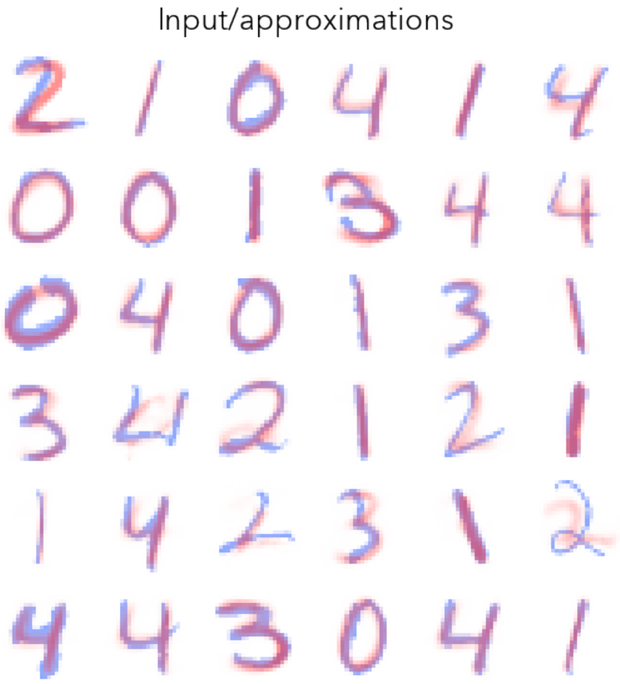
Variational Auto-Encoder (VAE)

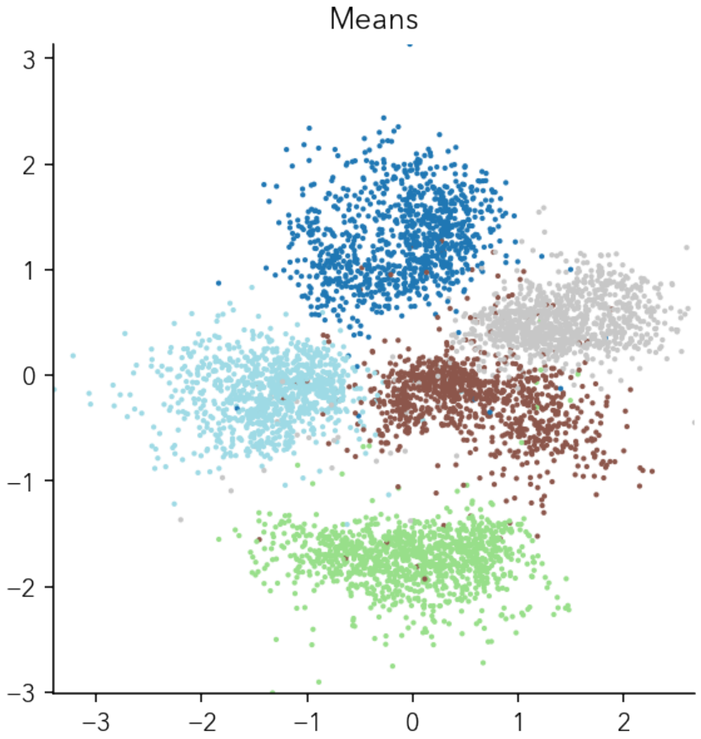
- Force $\mathbf{h}$ to be Gaussian distributed
  - Helps us to know its structure
  - Creates a more efficient encoding (more later)
- How to do this?
  - Encoder maps each input $\mathbf{x}$ to a mean and variance
  - We sample a Gaussian with them
  - We present that sample to the decoder
- We additionally minimize: $\mathrm{KL}\Big(\mathcal{N}(\mathbf{m},\mathbf{s}) \,\|\, \mathcal{N}(\mathbf{0},\mathbf{I})\Big)$
  - i.e. tend to generate $\mathbf{m} = \mathbf{0}$, $\mathbf{s} = \mathbf{I}$

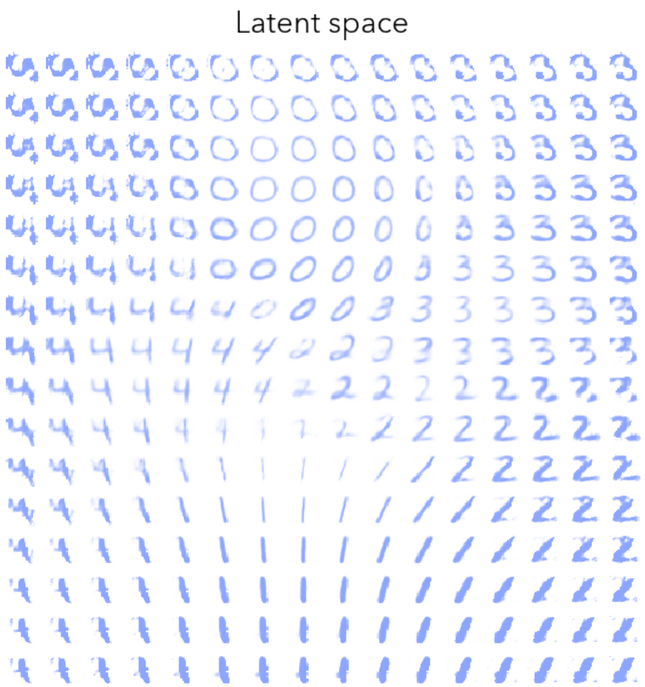


VAE MNIST example

- More convenient results
  - New latent representation is more compact and predictable
  - Sampled latent space generates continuous outputs







Key points to make

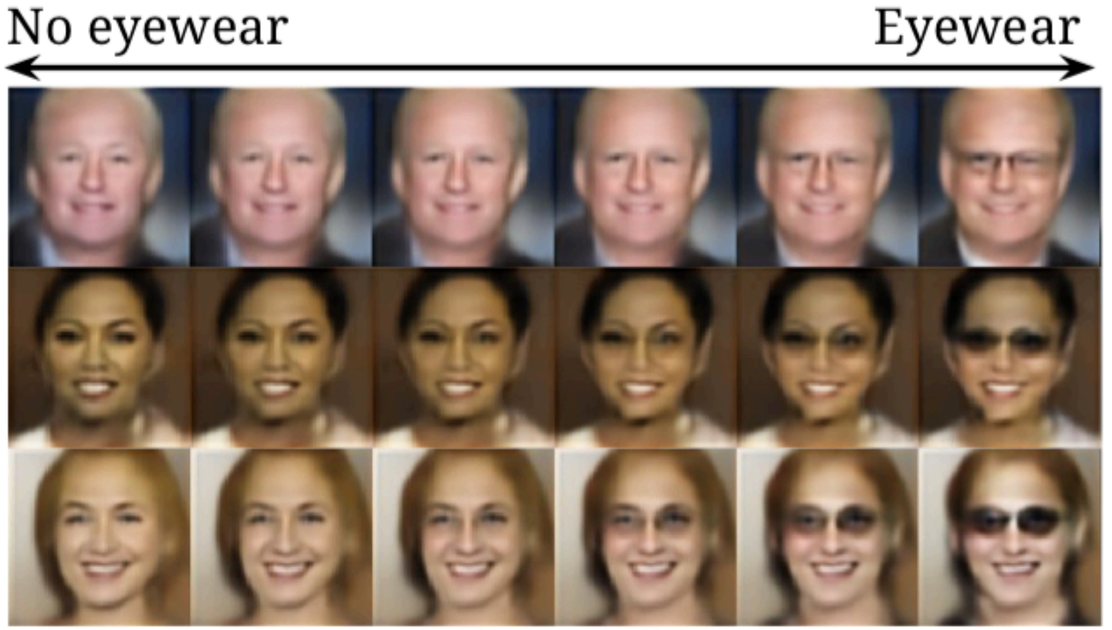
- We now have a distribution for $\mathbf{h}$
  - We know where to sample from
  - We additionally minimize gaps in the latent space

- The sampling process creates a more efficient encoder
  - Noise in the process forces the same data point to map to an entire region, and not a single point (we avoid overfitting)

Conditional VAEs

- If we also have labels for our data we can use them
  - Provide label as additional input to the decoder
  - This allows us to specify elements of data to generate

- E.g. Attribute2Image model
  - Learn faces with attributes
  - Sample and impose constraints



VAEs in action

- Popular for generating images
  - Generate new images from old
  - Generate new images with custom attributes (e.g. age, gender, …)
- Also used for making music
  - Interpolate between melodies
  - Generate new sequences from old
    - Can use RNNs in VAE model too!

An alternative approach

- Use a neural network to teach a neural network
  - Avoid selecting a specific cost function
  - Instead of comparing inputs/outputs, use a teacher network

- Why is this good?
  - We don’t have to use a simplistic measure (e.g. L2 reconstruction)
  - We can have a constantly evolving model

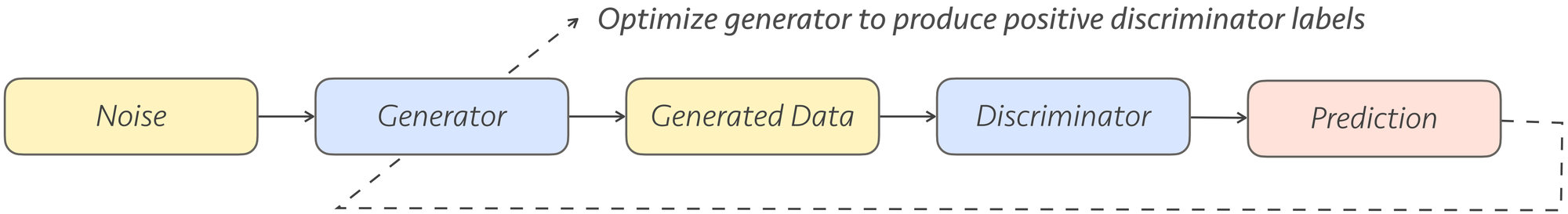
Generative Adversarial Networks (GANs)

- GANs are composed out of two networks
  - The *Generator*, creates new data points from noise input
  - The *Discriminator*, classifies data points as generated or real

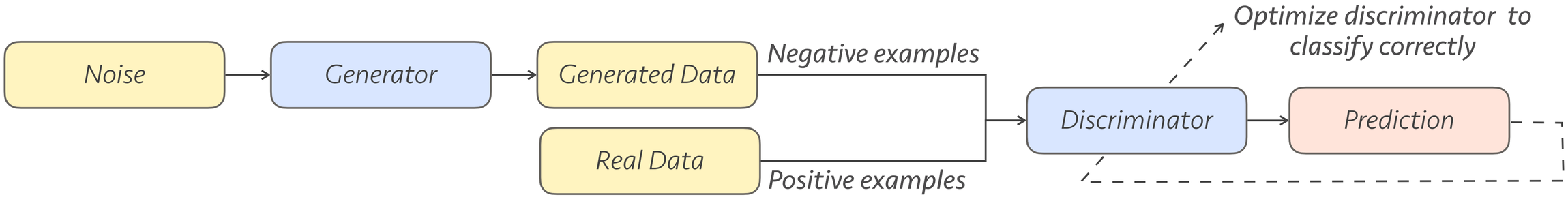
- Adversarial training process
  - Train the generator such that it fools the discriminator
  - Train the discriminator to detect generator's data
    - Through this process they both constantly improve each other

Training a GAN

- Train the generator to fool the discriminator
  - Improves the quality of generated data



- Train the discriminator to detect generated data
  - Makes the discriminator pickier, which will improve generator



How to set this up

- Original formulation, minimax setup

$$\max D = E_{\mathbf{x}} \log\Big[D(\mathbf{x})\Big] + E_{\mathbf{z}} \log\Big[1 - D\Big(G(\mathbf{z})\Big)\Big]$$

$$\min G = E_{\mathbf{z}} \log\Big[1 - D\Big(G(\mathbf{z})\Big)\Big]$$

- Wasserstein GANs ⟵ *Work better!*
  - Boils down to a simpler problem
    - Train the discriminator *N* times using regular classification loss
    - Train the generator *M* times using discriminator's loss
  - Get rid of the logs while we're at it

Some problems

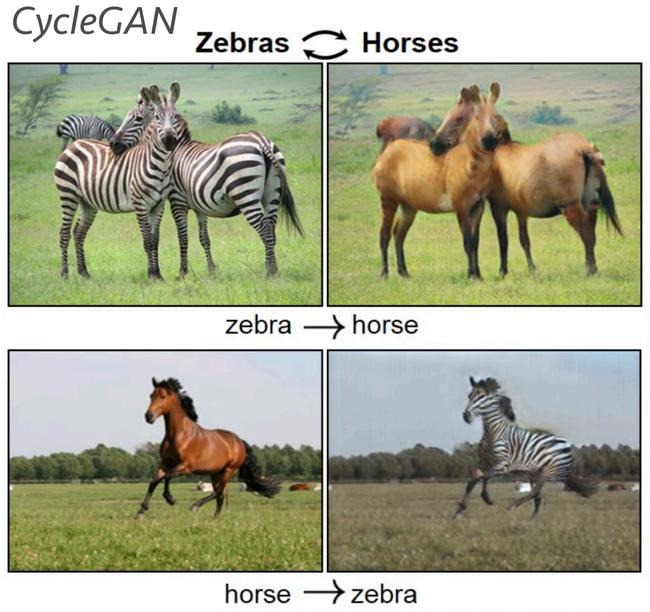
- Generated data are not guaranteed to be right
  - They will most likely just fool the discriminator (is that enough?)
- Mode collapse
  - The generator constantly produces one passable data point
    - There is no guarantee of having variance in the generated data
  - Training can be tricky
- Cannot map to a describable latent space

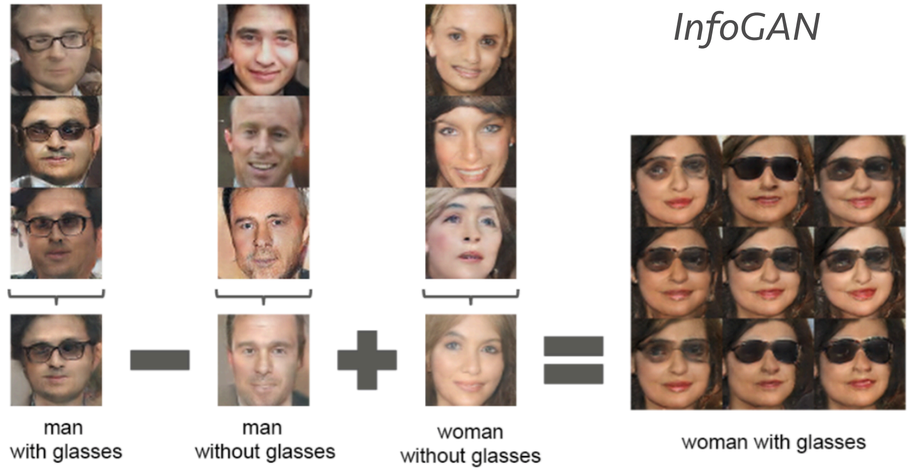
GANs in action

- Very popular in vision/graphics
  - E.g. fake celebrity generation
- Some success in other domains
  - Very useful when we cannot easily describe the data we want to make

Many elaborations

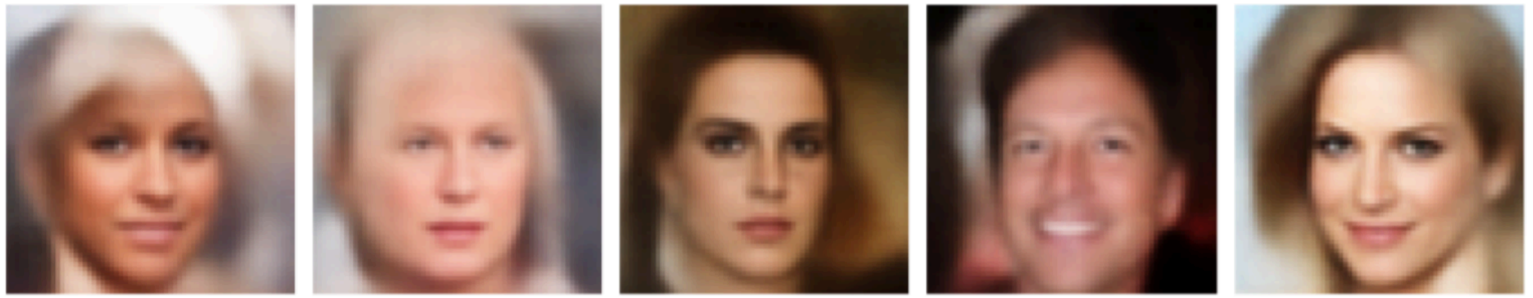
- CycleGAN
  - Make an autoencoder, provide additional output to a discriminator of other style
  - Then run it backwards
- InfoGAN
  - Force part of representation to use a latent code that adds information
- Many more …





VAEs or GANs?

- VAEs are known to produce “fuzzier” outputs



- GANs are known for sharper (but weirder) outputs

By request

- Attention and Transformers
  - An alternative to CNNs/RNNs
- Quick intro to reinforcement learning
  - The basics
  - Deep Q learning
  - Playing games
- Regression and estimation
- What's hot in ML/DL for audio/speech today?

Re-examining CNNs

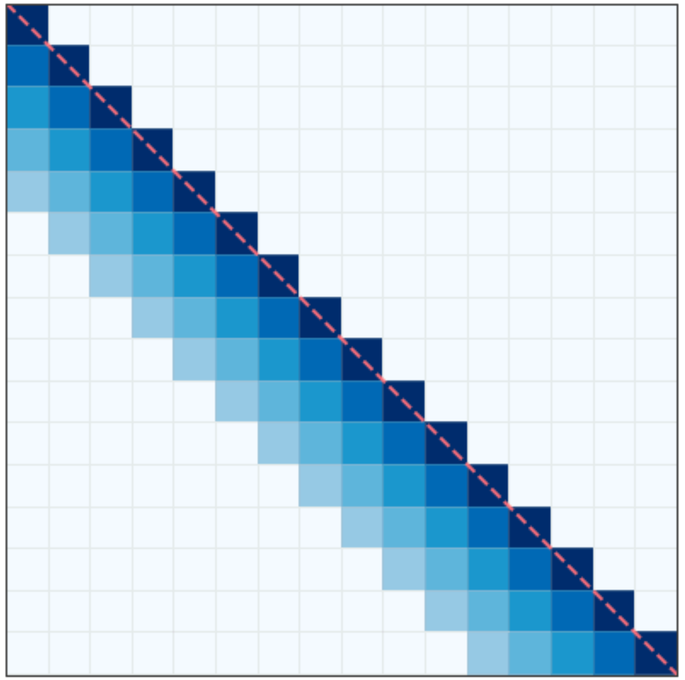
- Convolution is a linear operation
  - i.e. it can be done with matrix multiplications

$$\begin{bmatrix} \vdots \\ z_{t-1} \\ z_t \\ z_{t+1} \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & & & & & \\ x_2 & x_1 & x_0 & & & \\ & x_2 & x_1 & x_0 & & \\ & & x_2 & x_1 & x_0 & \\ & & & \ddots & & \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ y_{t-1} \\ y_t \\ y_{t+1} \\ \vdots \end{bmatrix}$$

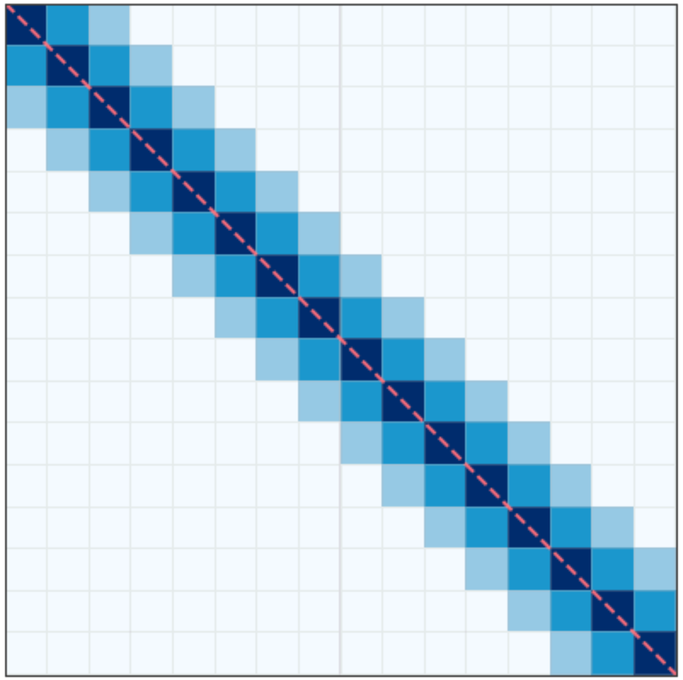
Different types of convolution matrices

- The type of convolution we use results in a different structured matrix

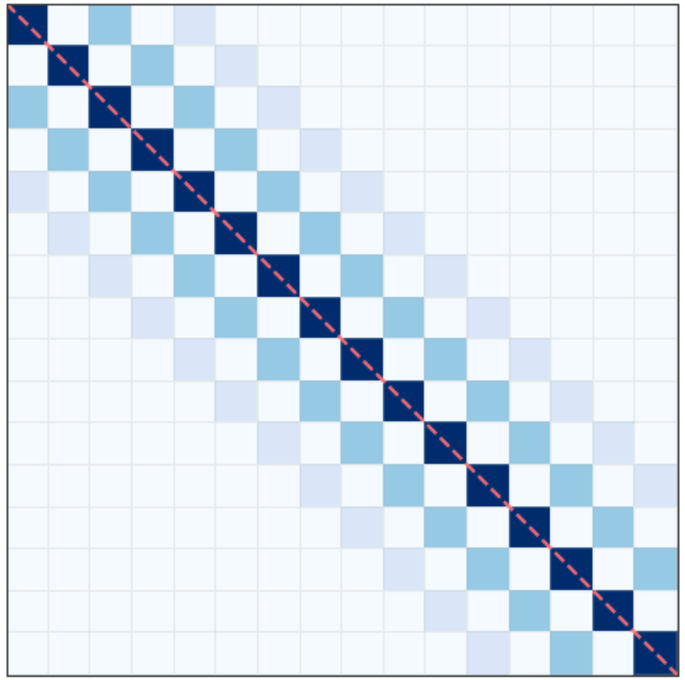
Causal 5-pt conv



Non-causal 5-pt conv



Non-causal 3-pt with dilation



What if used other structures?

- Block-diagonal? Toeplitz? Checkerboard?
  - They will perform some form of structured convolution

- Or how about we use a full matrix?
  - Which will use all the input data!
    - Surely that must be better!

Core attention idea

- Use all the available samples, not just the immediate neighborhood

- Simplest form:

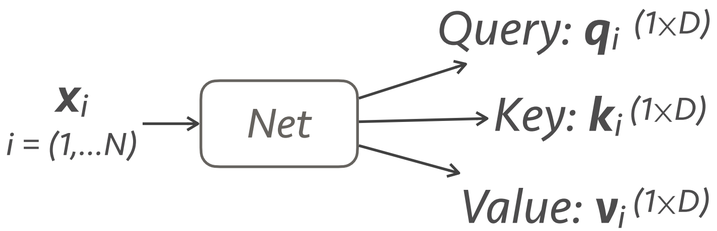
$$\mathbf{y} = \mathbf{W} \cdot \begin{bmatrix} \cdots & x_{t-1} & x_t & x_{t+1} & \cdots \end{bmatrix}^\top$$

  - Good thing: The receptive field is really wide!
  - Bad thing: Hey, that won't work for varying length inputs!!

A length-independent attention model

- Multi-step process:

1. Create three versions of each input $\mathbf{x}_i$ using a neural net



2. Compute similarity of queries $\mathbf{q}_i$ with keys $\mathbf{k}_i$

$$softmax(\mathbf{Q}\cdot\mathbf{K}^T)=\mathbf{S}\ (N\times N)$$

Tells you how similar each sample is with each other

3. Use that to compute layer's output

$$\mathbf{S}\cdot\mathbf{V}=\mathbf{Y}\ (N\times D)$$

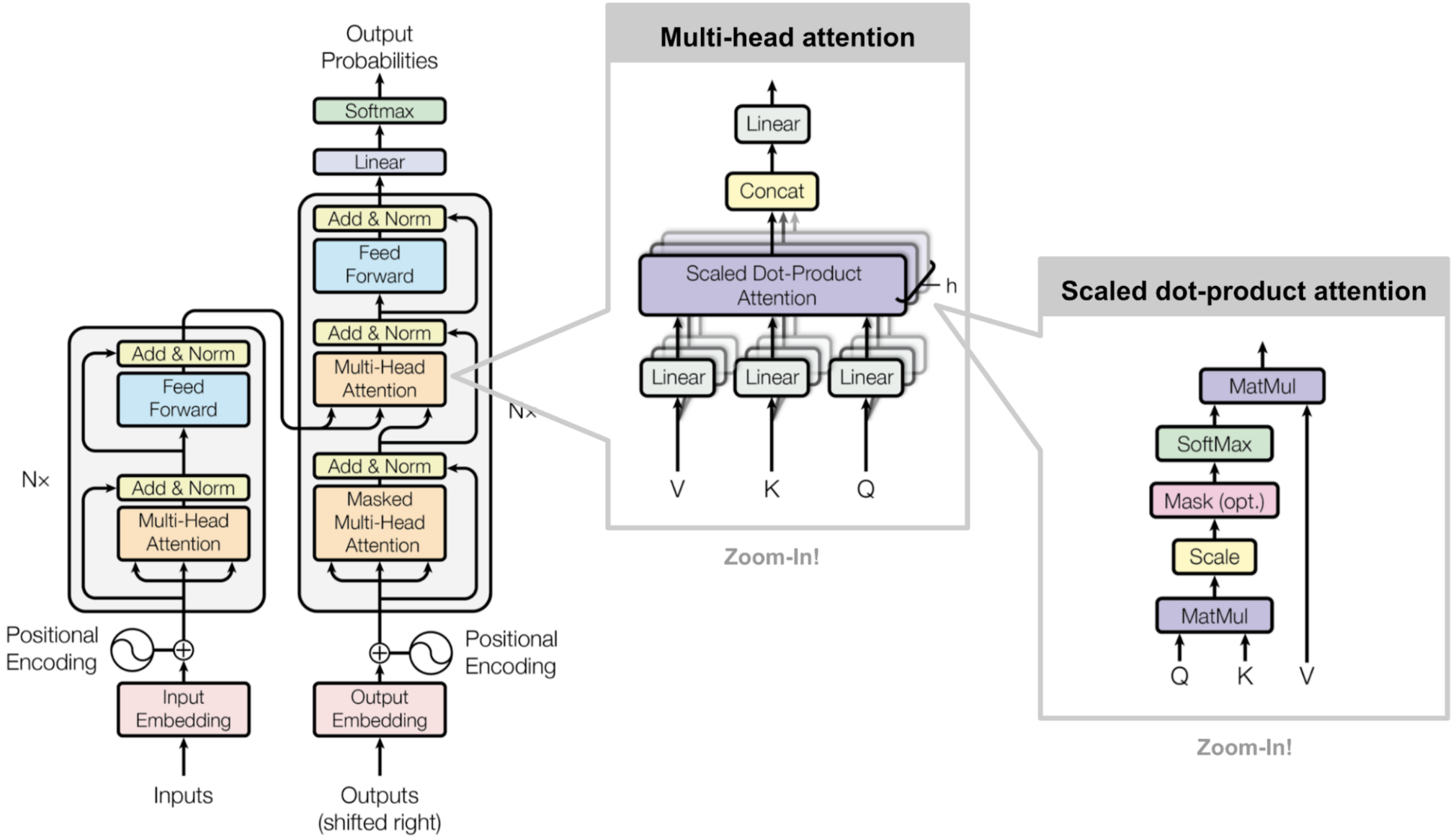
- A simplified way to see it: $\mathbf{Y} = softmax(\mathbf{X}\cdot\mathbf{X}^T)\cdot\mathbf{X}$
  - So that's similar to clustering in some sense
- Why use query/key idea?
  - Learns a transform based on data, and breaks similarity symmetry

So what's new here?

- Now we can deal with arbitrary length inputs
  - Before the learned transform size was fixed to the input length
    - We abstract that by transforming the inputs to queries/keys

- But there is still no concept of time!!
  - To get time you need to use *positional encodings*
    - Usually sinusoids used as a timestamp

Full Transformer model



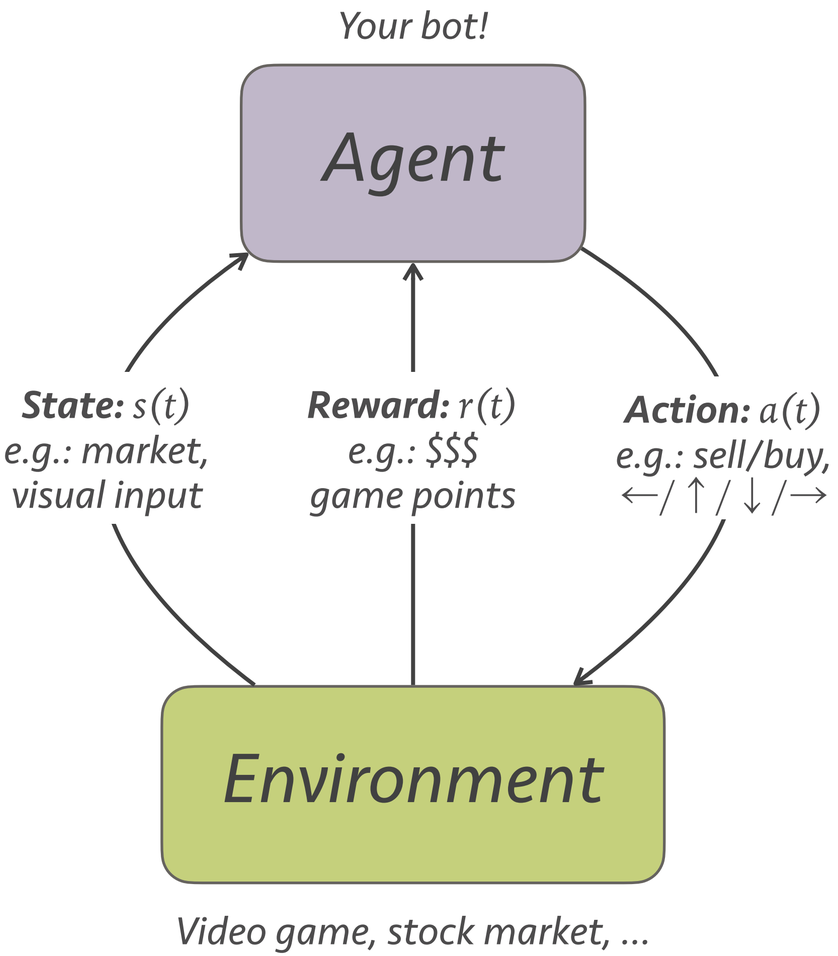
Reinforcement Learning

- Make actions to maximize cumulative rewards
  - E.g. navigate a maze (which way to go?)
  - Play at the stock market (how to spend money?)
  - Survive in a dynamic environment (Super Mario, robots)

- Key uniqueness: Rewards are not instantaneous
  - We cannot train on input/output pairs
  - Often seemingly costly decisions are good (e.g. a large investment)

Key setup

- Environment
  - The place where you deploy RL
  - State: Description of environment
  - Rewards: What you optimize
    - Will not come instantaneously
- Agent
  - This is your software
  - Action: How to react to state
  - Observation: How it sees the state



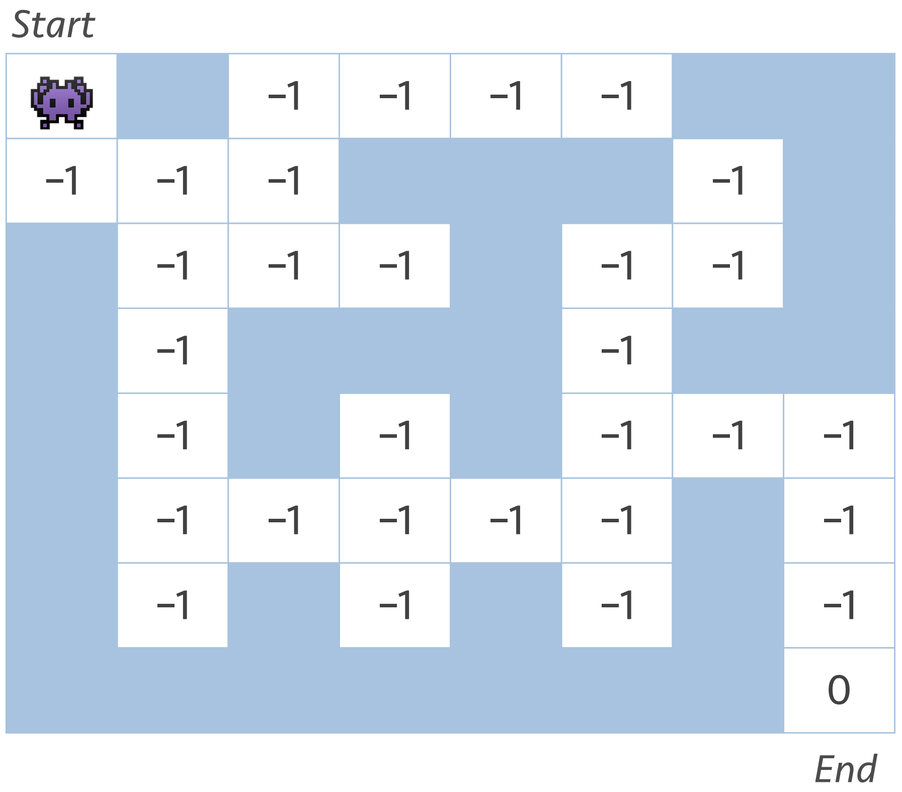
More formally

- Policy: $a = \pi(s)$
  - How to behave based on the state

- Value function: $V_\pi(s, a) = \mathbb{E}[r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} \ldots \,|\, s, a]$
  - How good is a given s?
  - $0 \leq \gamma \leq 1$ is the discount factor
    - Helps focus a bit more to the near future
    - The far future is more uncertain

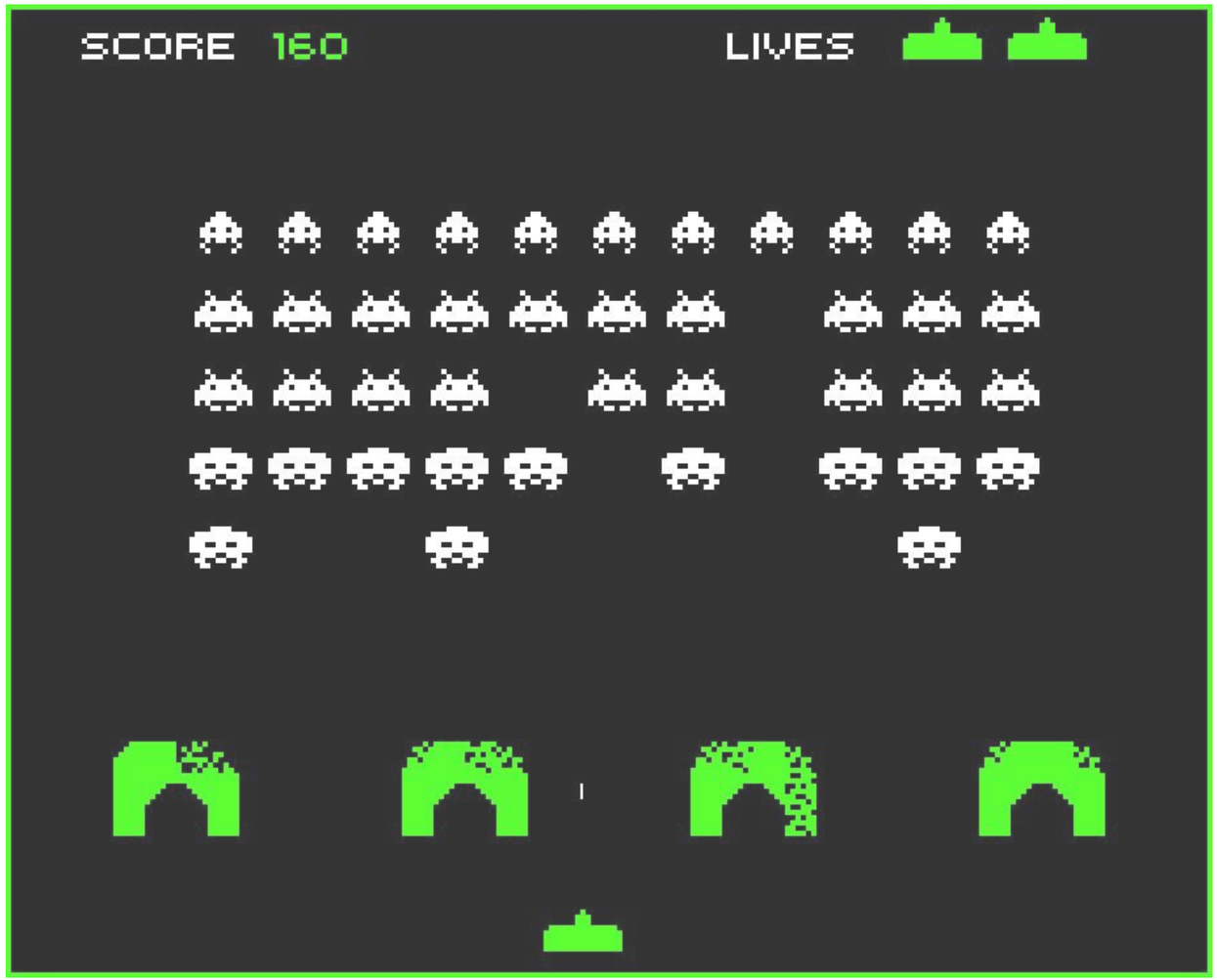
A simple example

- Navigate a maze
  - What are the actions?
  - What is the state?
  - What is the reward?



A harder example

- Actions
  - ← , → , or Fire
- State
  - What you see!
    - How do we use it?
- Reward
  - Game score!

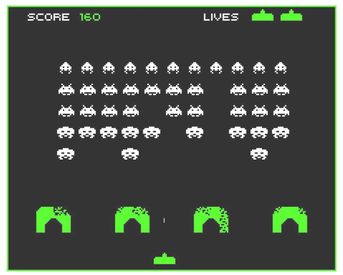


Various approaches

- We can focus on learning various things
  - Find the policy function π that maximizes rewards
    - Policy-based RL
  - Approximate the value function
    - Value-based RL
  - Learn a model of the environment and plan accordingly
    - Model-based RL
- Deep RL
  - Approximate any or all of the above with a deep network

A deep Q-learning example

- Video game playing
  - Inputs: pixels / Output: Q-function (value based on action)
  - Policy: $\pi(s) = \text{argmax}_a Q(s, a)$
  - Loss function:
    - $L_t = [\underbrace{r_t + \max_a Q(s_{t+1}, a_{t+1})}_{\textit{Target}} - \underbrace{Q(s_t, a_t)}_{\textit{Prediction}}]^2$



Deep Q Network

$Q(s, \leftarrow)$

$Q(s, \rightarrow)$

$Q(s, fire)$

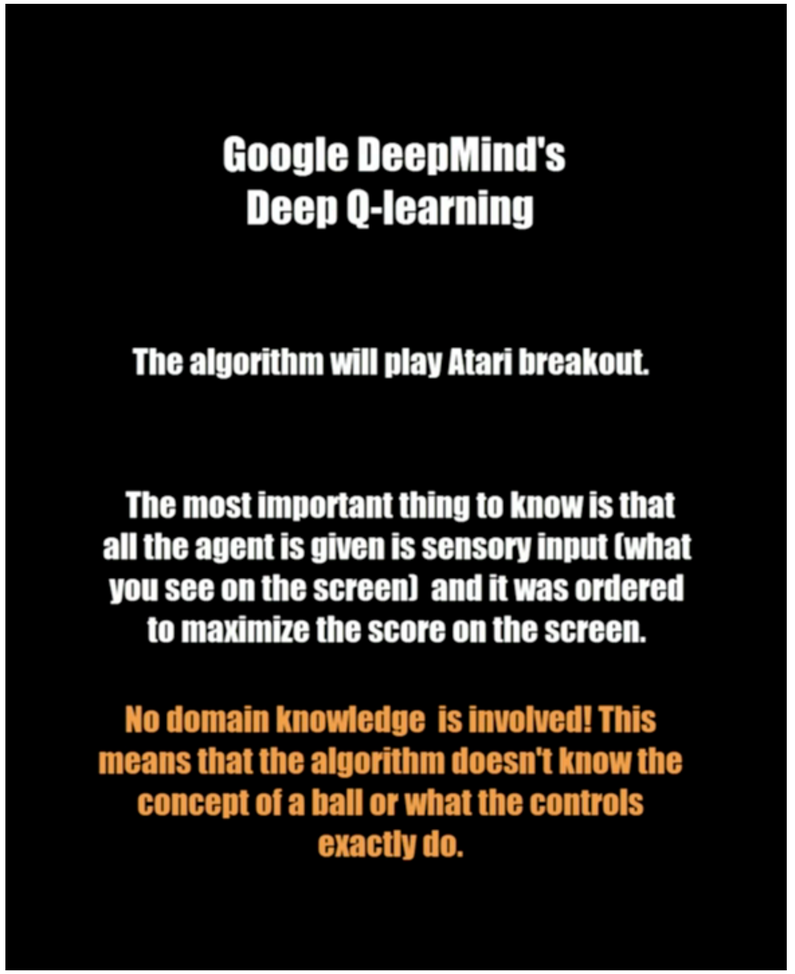
1. *Do 1st pass with s_t, get Q*
2. *Do 2nd pass with s_{t+1} for all actions*
3. *Use Q of best action as target*

Lots of interesting applications

- Lots of RL in the news lately
  - Game playing (Atari, Go, chess), smart robots (driving, moving, folding), finance ($$$), …

Learning to move

Playing video games



Not that simple …

- Front-end tweaking
  - Use last few frames as state, make frames grayscale/smaller
- Experience replay
  - Keep a collection of $\{s_t, a_t, s_{t+1}, a_{t+1}\}$ and randomly train on them
  - Avoids local minima
- Exploration-Exploitation
  - Dilemma: settle for an ok Q approximation or try something else?
  - ε-greedy exploration: with probability ε do something random
- Lots more to tweak, and many more approaches …

Regularization, overfitting, and regression

- The more parameters you have the more data you need
  - Or, you can *regularize*
- Simple with tractable models
  - Lots of work on relevant bounds for estimators
- A bit messy for neural nets
  - Difficult systems to analyze thoroughly

NN regularization tricks

- Common approaches
  - Weight regularization, e.g. min L2/L1
    - or make your model smaller!
  - Dropout
  - Data augmentation
- Rich literature on what works
  - But this is mostly on a per-case basis
    - If you work on generic problems that great, otherwise …

Regression or classification?

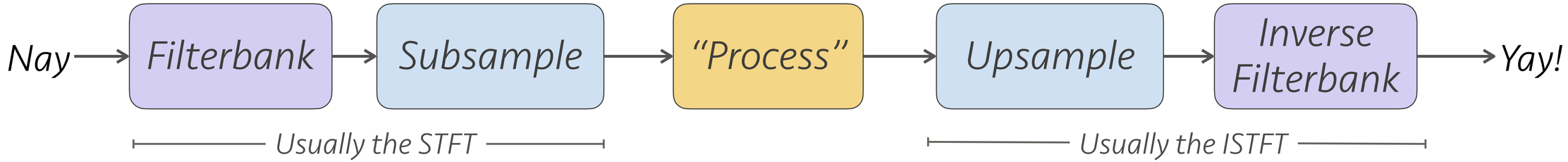
- Neural nets are bad at continuous-value regression
  - Most of their power comes from saturating non-linearities
  - Easier to output a range of large values that saturate

- Popular case: WaveNet
  - Deepified linear prediction
  - Output is not $x(t+1)$, but a classifier on it's *N*-bit value

Trends in ML/DL for speech/audio

- Research-wise: Deepify eveything!
  - Take existing algorithms and convert them to a neural net

Typical audio algorithm



Deepified version

Areas of growth

- Huge increase in sound recognition
  - DCASE workshop - exponential growth
- Lots of new work on generative models
  - OpenAI Jukebox
  - Google's TacoTron


Pop in the style of Katy Perry


Metal in the style of Rage


Speech with priors


Speech without priors

- Rethinking classical DSP
  - Letting go of linear/Gaussian/etc systems