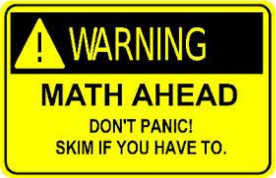


CSC413/2516 Lecture 10: Generative Models & Reinforcement Learning

Jimmy Ba and Bo Wang

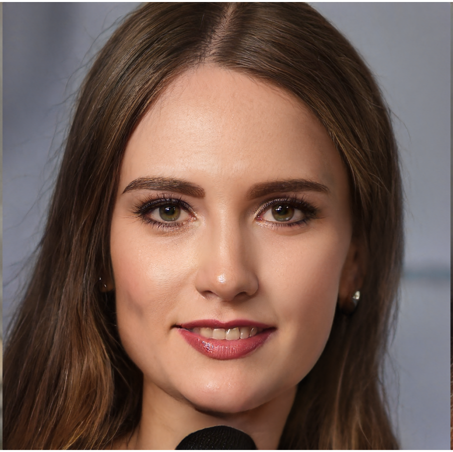
Logistics

Some administrative stuff:

- PA 4 (**most interesting**) is out! (Due April 1st, not a joke!)
- HW 4 (**most 'mathy'**) will be out in March 25th, and due April 08.
- Final Project is due April 12th! (likely to be extended, stay tuned!)

Quiz: Which face image is fake?

A



B

C

Overview

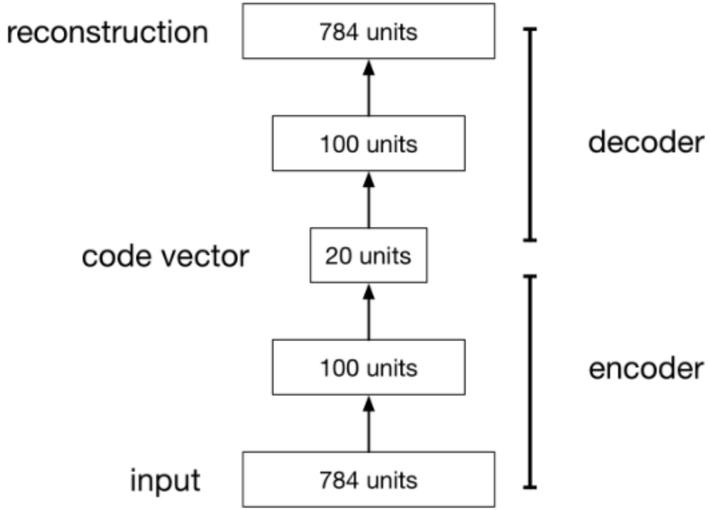
Four modern approaches to generative modeling:

- Autoregressive models (Lectures 3, 7, and 8)
- Generative adversarial networks (last lecture)
- Reversible architectures (last lecture)
- Variational autoencoders (this lecture)

All four approaches have different pros and cons.

Autoencoders

- An autoencoder is a feed-forward neural net whose job it is to take an input **x** and predict **x**.
- To make this non-trivial, we need to add a bottleneck layer whose dimension is much smaller than the input.



Autoencoders

Why autoencoders?

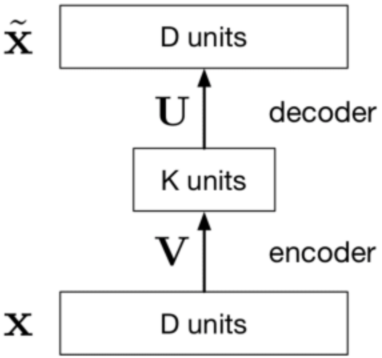
- Map high-dimensional data to two dimensions for visualization
- Compression (i.e. reducing the file size)
  - Note: this requires a VAE, not just an ordinary autoencoder.
- Learn abstract features in an unsupervised way so you can apply them to a supervised task
  - Unlabled data can be much more plentiful than labeled data
- Learn a semantically meaningful representation where you can, e.g., interpolate between different images.

Principal Component Analysis (optional)

- The simplest kind of autoencoder has one hidden layer, linear activations, and squared error loss.

$$\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|^2$$

- This network computes $\tilde{\mathbf{x}} = \mathbf{U}\mathbf{V}\mathbf{x}$, which is a linear function.
- If $K \geq D$, we can choose $\mathbf{U}$ and $\mathbf{V}$ such that $\mathbf{U}\mathbf{V}$ is the identity. This isn't very interesting.



- But suppose $K < D$:
  - $\mathbf{V}$ maps $\mathbf{x}$ to a K-dimensional space, so it's doing dimensionality reduction.
  - The output must lie in a K-dimensional subspace, namely the column space of $\mathbf{U}$.

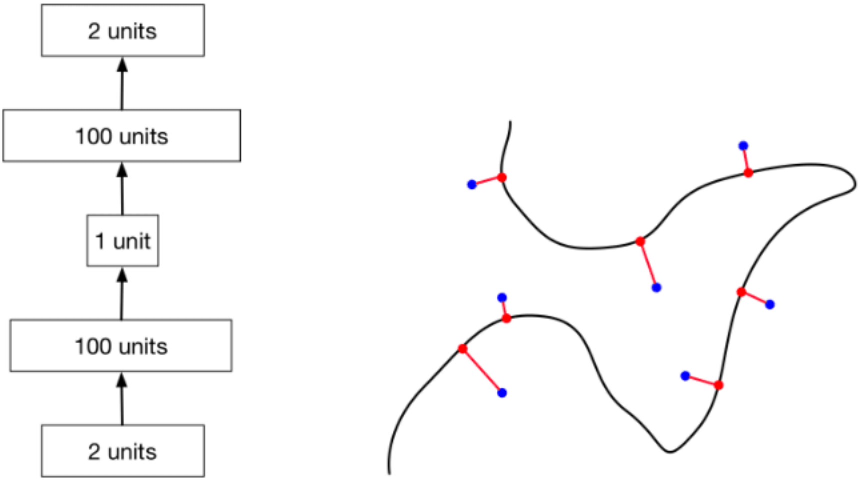
Principal Component Analysis (optional)

- Review from CSC311: linear autoencoders with squared error loss are equivalent to Principal Component Analysis (PCA).
- Two equivalent formulations:
  - Find the subspace that minimizes the reconstruction error.
  - Find the subspace that maximizes the projected variance.
- The optimal subspace is spanned by the dominant eigenvectors of the empirical covariance matrix.

"Eigenfaces"

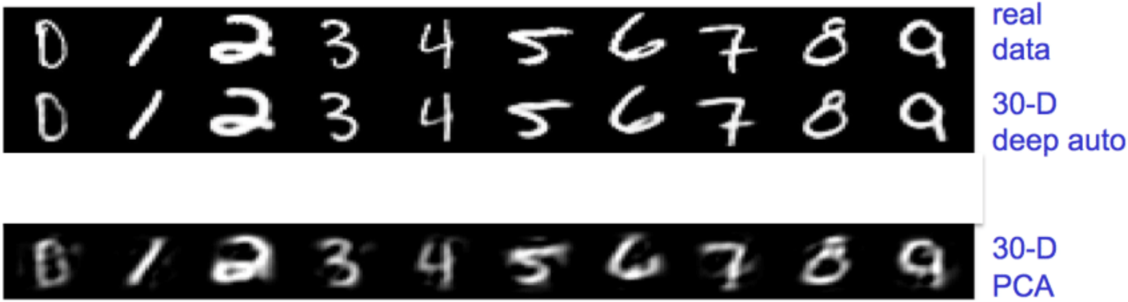
Deep Autoencoders

- Deep nonlinear autoencoders learn to project the data, not onto a subspace, but onto a nonlinear manifold
- This manifold is the image of the decoder.
- This is a kind of nonlinear dimensionality reduction.



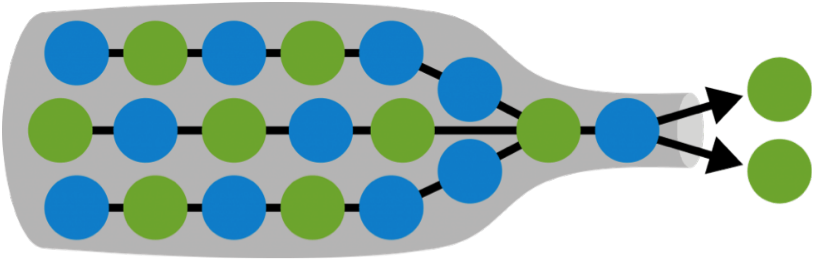
Deep Autoencoders

- Nonlinear autoencoders can learn more powerful codes for a given dimensionality, compared with linear autoencoders (PCA)

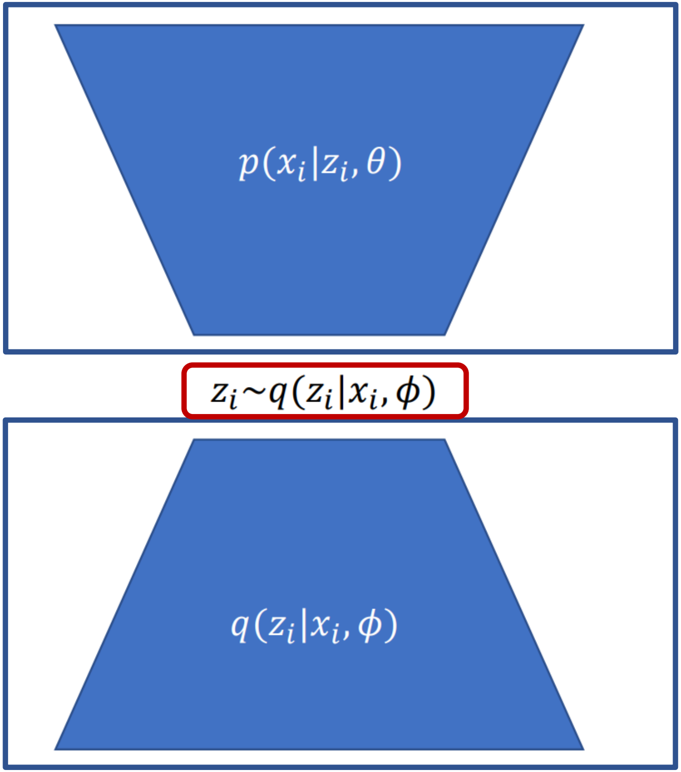


Deep Autoencoders

- Some limitations of autoencoders
  - They're not generative models, so they don't define a distribution
  - How to choose the latent dimension?



Variational Auto-encoder (VAE)

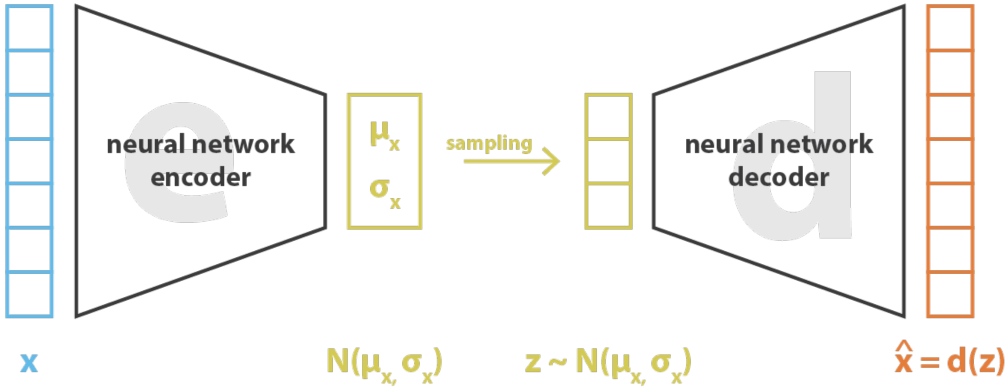


Decoder learns the generative process given the sampled latent vectors.

Sampling process in the middle.

Encoder learns the distribution of latent space given the observations.

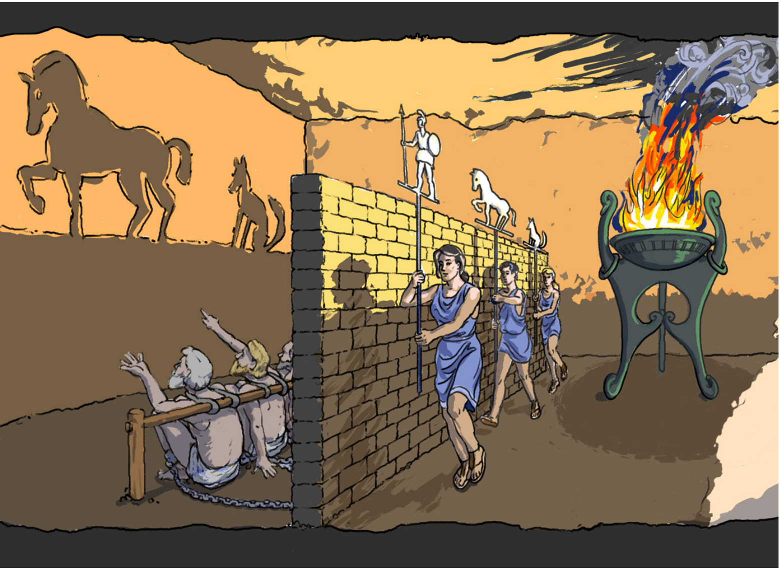
Variational Auto-encoder (VAE)

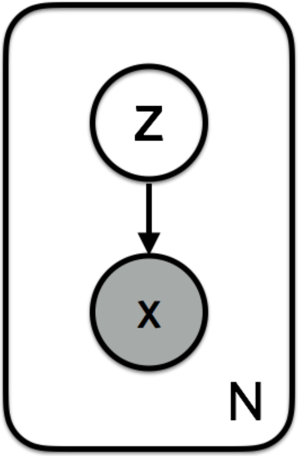


$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[\,N(\mu_x, \sigma_x), N(0, I)\,] = \|x - d(z)\|^2 + \text{KL}[\,N(\mu_x, \sigma_x), N(0, I)\,]$$

Source: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Observation Model





Source: https://iagtm.pressbooks.com/chapter/story-platos-allegory-of-the-cave/

Observation Model

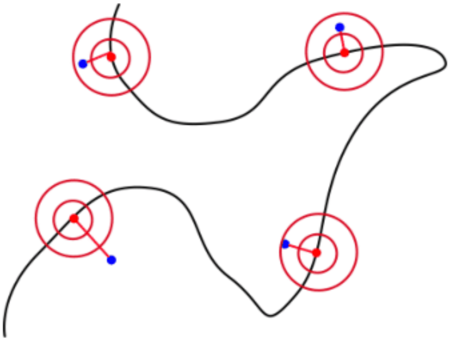
- Consider training a generator network with maximum likelihood.

$$p(\mathbf{x}) = \int p(\mathbf{z}) p(\mathbf{x} \mid \mathbf{z}) \, \mathrm{d}\mathbf{z}$$

- One problem: if $\mathbf{z}$ is low-dimensional and the decoder is deterministic, then $p(\mathbf{x}) = 0$ almost everywhere!
  - The model only generates samples over a low-dimensional sub-manifold of $\mathcal{X}$.
- Solution: define a noisy observation model, e.g.

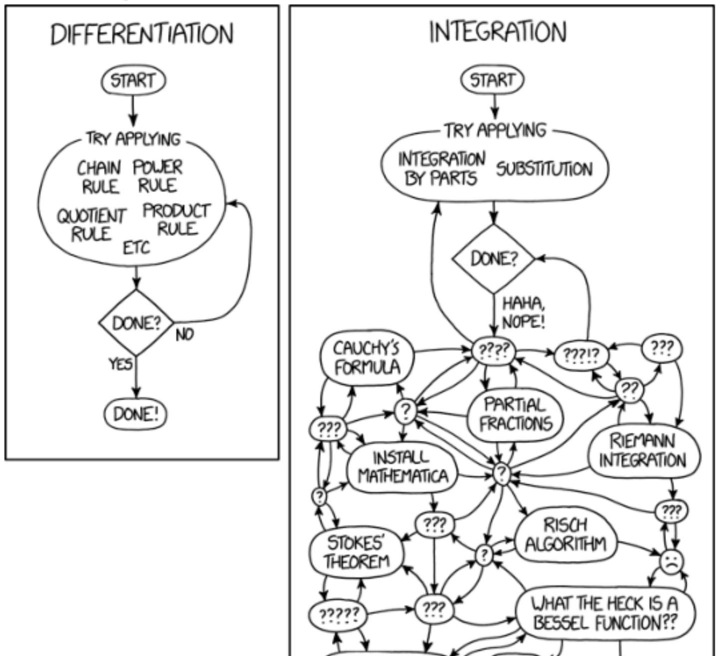
$$p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x}; G_{\boldsymbol{\theta}}(\mathbf{z}), \eta \mathbf{I}),$$

where $G_{\boldsymbol{\theta}}$ is the function computed by the decoder with parameters $\boldsymbol{\theta}$.



Observation Model

- At least $p(\mathbf{x}) = \int p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z}) \, \mathrm{d}\mathbf{z}$ is well-defined, but how can we compute it?
- Integration, according to XKCD:



Observation Model

- At least $p(\mathbf{x}) = \int p(\mathbf{z}) p(\mathbf{x} \mid \mathbf{z}) \, \mathrm{d}\mathbf{z}$ is well-defined, but how can we compute it?
  - The decoder function $G_\theta(\mathbf{z})$ is very complicated, so there's no hope of finding a closed form.
- Instead, we will try to maximize a lower bound on $\log p(\mathbf{x})$.
  - The math is essentially the same as in the EM algorithm from CSC411.

Variational Inference

- We obtain the lower bound using Jensen's Inequality: for a convex function h of a random variable X,

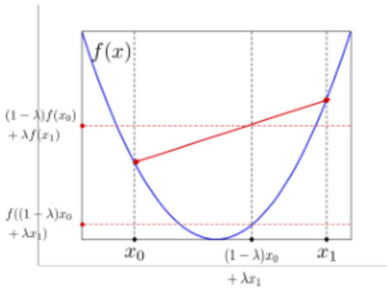
$$\mathbb{E}[h(X)] \geq h(\mathbb{E}[X])$$

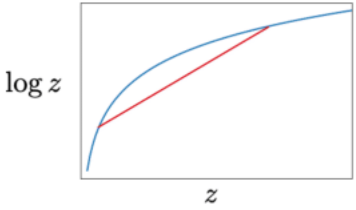
  Therefore, if h is concave (i.e. $-h$ is convex),

$$\mathbb{E}[h(X)] \leq h(\mathbb{E}[X])$$

- The function $\log z$ is concave. Therefore,

$$\mathbb{E}[\log X] \leq \log \mathbb{E}[X]$$





Variational Inference

- Suppose we have some distribution $q(\mathbf{z})$. (We'll see later where this comes from.)
- We use Jensen's Inequality to obtain the lower bound.

Heads-up: You will show the full calculation in HW4.

$$\begin{aligned}
\log p(\mathbf{x}) &= \log \int p(\mathbf{z})\, p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z} \\
&= \log \int q(\mathbf{z})\, \frac{p(\mathbf{z})}{q(\mathbf{z})} p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z} \\
&\geq \int q(\mathbf{z}) \log \left[\frac{p(\mathbf{z})}{q(\mathbf{z})}\, p(\mathbf{x}|\mathbf{z}) \right] \mathrm{d}\mathbf{z} \qquad \text{(Jensen's Inequality)} \\
&= \mathbb{E}_q \left[\log \frac{p(\mathbf{z})}{q(\mathbf{z})} \right] + \mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z}) \right]
\end{aligned}$$

- We'll look at these two terms in turn.

Variational Inference

- The first term we'll look at is $\mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})]$
- Since we assumed a Gaussian observation model,

$$\begin{aligned}\log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; G_{\boldsymbol{\theta}}(\mathbf{z}), \eta \mathbf{I}) \\ &= \log \left[\frac{1}{(2\pi\eta)^{D/2}} \exp\left(-\frac{1}{2\eta} \|\mathbf{x} - G_{\boldsymbol{\theta}}(\mathbf{z})\|^2 \right) \right] \\ &= -\frac{1}{2\eta} \|\mathbf{x} - G_{\boldsymbol{\theta}}(\mathbf{z})\|^2 + \mathrm{const}\end{aligned}$$

- So this term is the expected squared error in reconstructing $\mathbf{x}$ from $\mathbf{z}$. We call it the reconstruction term.

Variational Inference

- The second term is $\mathbb{E}_q\left[\log \frac{p(\mathbf{z})}{q(\mathbf{z})}\right]$.
- This is just $-\mathrm{D}_{\mathrm{KL}}(q(\mathbf{z})\|p(\mathbf{z}))$, where D_{KL} is the Kullback-Leibler (KL) divergence

$$\mathrm{D}_{\mathrm{KL}}(q(\mathbf{z})\|p(\mathbf{z})) \triangleq \mathbb{E}_q\left[\log \frac{q(\mathbf{z})}{p(\mathbf{z})}\right]$$

  - KL divergence is a widely used measure of distance between probability distributions, though it doesn't satisfy the axioms to be a distance metric.
  - More details in tutorial.
- Typically, $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. Hence, the KL term encourages q to be close to $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

Variational Inference

- Hence, we're trying to maximize the variational lower bound, or variational free energy:

$$\log p(\mathbf{x}) \geq \mathcal{F}(\boldsymbol{\theta}, q) = \mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z})\right] - \mathrm{D}_{\mathrm{KL}}(q \| p).$$

- The term "variational" is a historical accident: "variational inference" used to be done using variational calculus, but this isn't how we train VAEs.
- We'd like to choose q to make the bound as tight as possible.
- It's possible to show that the gap is given by:

$$\log p(\mathbf{x}) - \mathcal{F}(\boldsymbol{\theta}, q) = \mathrm{D}_{\mathrm{KL}}(q(\mathbf{z}) \| p(\mathbf{z}|\mathbf{x})).$$

  Therefore, we'd like q to be as close as possible to the posterior distribution $p(\mathbf{z}|\mathbf{x})$.

- Let's think about the role of each of the two terms.
- The reconstruction term

$$\mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})] = -\frac{1}{2\sigma^2}\mathbb{E}_q[\|\mathbf{x} - G_\theta(\mathbf{z})\|^2] + \mathrm{const}$$

is minimized when q is a point mass on

$$\mathbf{z}_* = \arg\min_{\mathbf{z}} \|\mathbf{x} - G_\theta(\mathbf{z})\|^2.$$

- But a point mass would have infinite KL divergence. (Exercise: check this.) So the KL term forces q to be more spread out.

Reparameterization Trick

- To fit q, let's assign it a parametric form, in particular a Gaussian distribution: $q(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K)$ and $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_K^2)$.
- In general, it's hard to differentiate through an expectation. But for Gaussian q, we can apply the reparameterization trick:

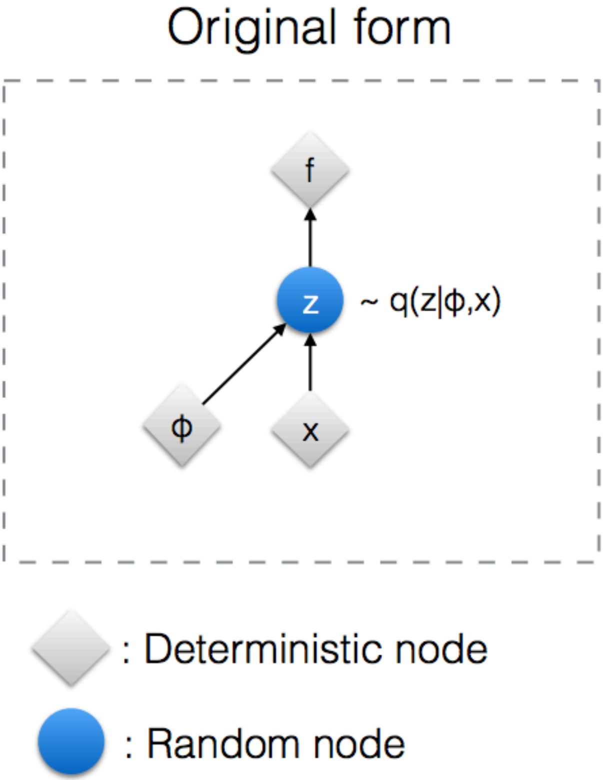
$$z_i = \mu_i + \sigma_i \epsilon_i,$$

  where $\epsilon_i \sim \mathcal{N}(0, 1)$.
- Hence,

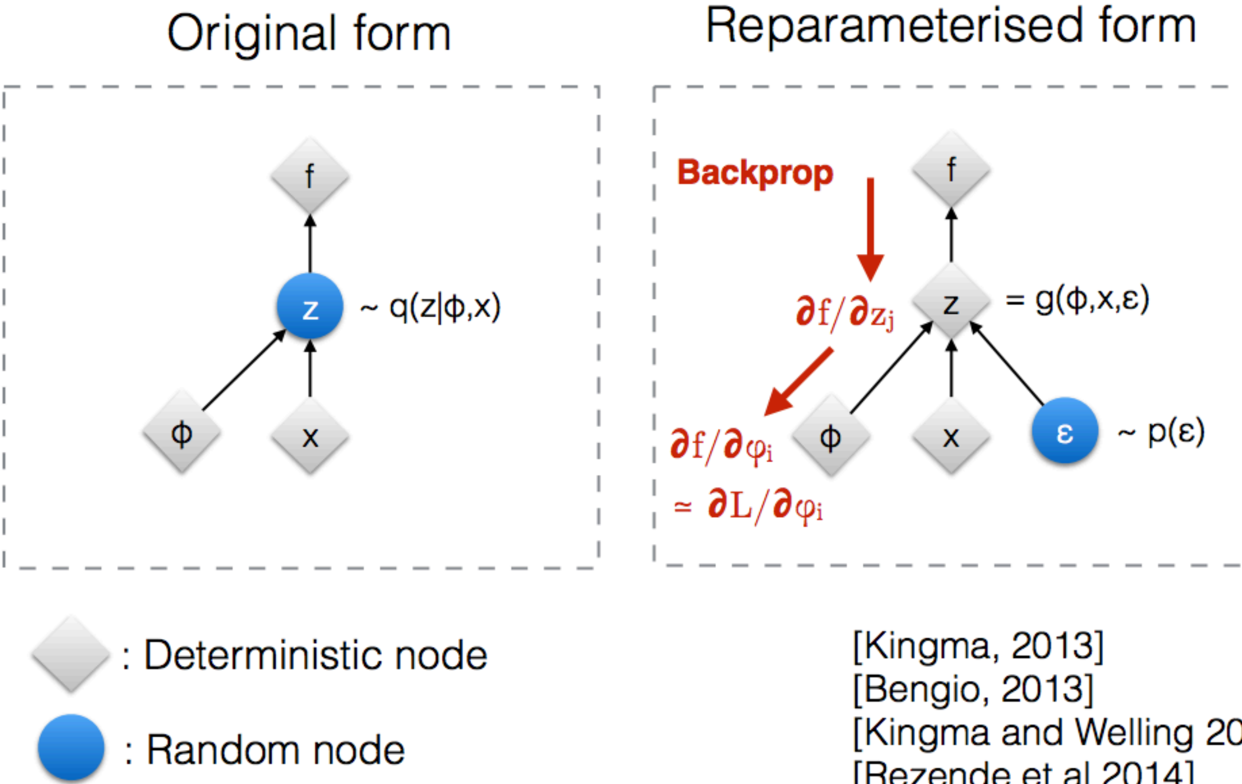
$$\overline{\mu_i} = \overline{z_i} \qquad \overline{\sigma_i} = \overline{z_i} \epsilon_i.$$

- This is exactly analogous to how we derived the backprop rules for dropout

Reparameterization Trick



Reparameterization Trick



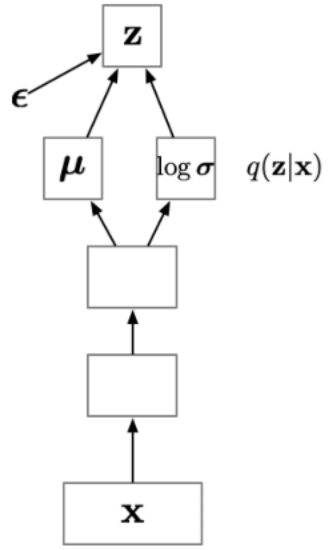
[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Amortization

- This suggests one strategy for learning the decoder. For each training example,
  1. Fit q to approximate the posterior for the current $\mathbf{x}$ by doing many steps of gradient ascent on $\mathcal{F}$.
  2. Update the decoder parameters $\boldsymbol{\theta}$ with gradient ascent on $\mathcal{F}$.
- **Problem:** this requires an expensive iterative procedure for every training example, so it will take a long time to process the whole training set.

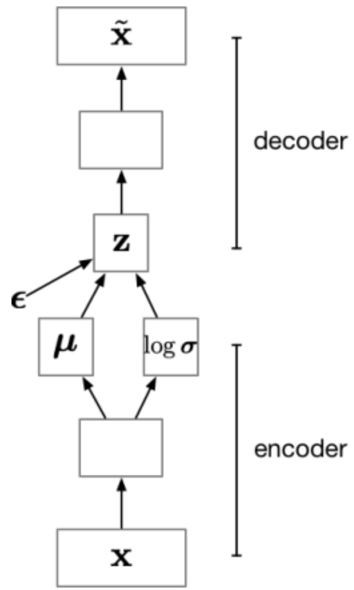
Amortization

- **Idea:** amortize the cost of inference by learning an inference network which predicts $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ as a function of $\mathbf{x}$.
- The outputs of the inference net are $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}$. (The log representation ensures $\boldsymbol{\sigma} > 0$.)
- If $\boldsymbol{\sigma} \approx \mathbf{0}$, then this network essentially computes $\mathbf{z}$ deterministically, by way of $\boldsymbol{\mu}$.
  - But the KL term encourages $\boldsymbol{\sigma} > 0$, so in general $\mathbf{z}$ will be noisy.
- The notation $q(\mathbf{z}|\mathbf{x})$ emphasizes that q depends on $\mathbf{x}$, even though it's not actually a conditional distribution.

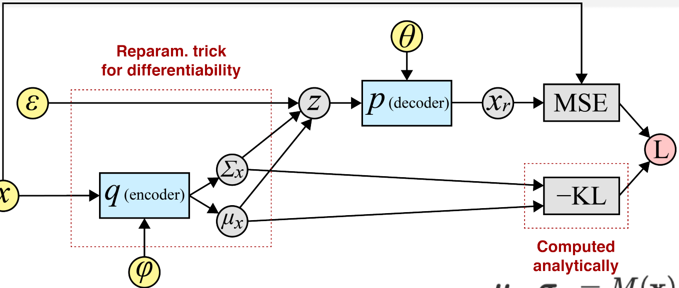


Amortization

- Combining this with the decoder network, we see the structure closely resembles an ordinary autoencoder. The inference net is like an encoder.
- Hence, this architecture is known as a variational autoencoder (VAE).
- The parameters of both the encoder and decoder networks are updated using a single pass of ordinary backprop.
  - The reconstruction term corresponds to squared error $\|\mathbf{x} - \tilde{\mathbf{x}}\|^2$, like in an ordinary VAE.
  - The KL term regularizes the representation by encouraging $\mathbf{z}$ to be more stochastic.



VAE - Summary



$\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x = M(\mathbf{x}), \Sigma(\mathbf{x})$	Push $\mathbf{x}$ through encoder
$\boldsymbol{\epsilon} \sim \mathcal{N}(0,1)$	Sample noise
$\mathbf{z} = \boldsymbol{\epsilon}\boldsymbol{\sigma}_x + \boldsymbol{\mu}_x$	Reparameterize
$\mathbf{x}_r = p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$	Push $\mathbf{z}$ through decoder
$\text{recon. loss} = \text{MSE}(\mathbf{x}, \mathbf{x}_r)$	Compute reconstruction loss
$\text{var. loss} = -\text{KL}[\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x) \| \mathcal{N}(0, I)]$	Compute variational loss
$\text{L} = \text{recon. loss} + \text{var. loss}$	Combine losses

VAEs vs. Other Generative Models

- In short, a VAE is like an autoencoder, except that it's also a generative model (defines a distribution $p(\mathbf{x})$).
- Unlike autoregressive models, generation only requires one forward pass.
- Unlike reversible models, we can fit a low-dimensional latent representation. We'll see we can do interesting things with this...

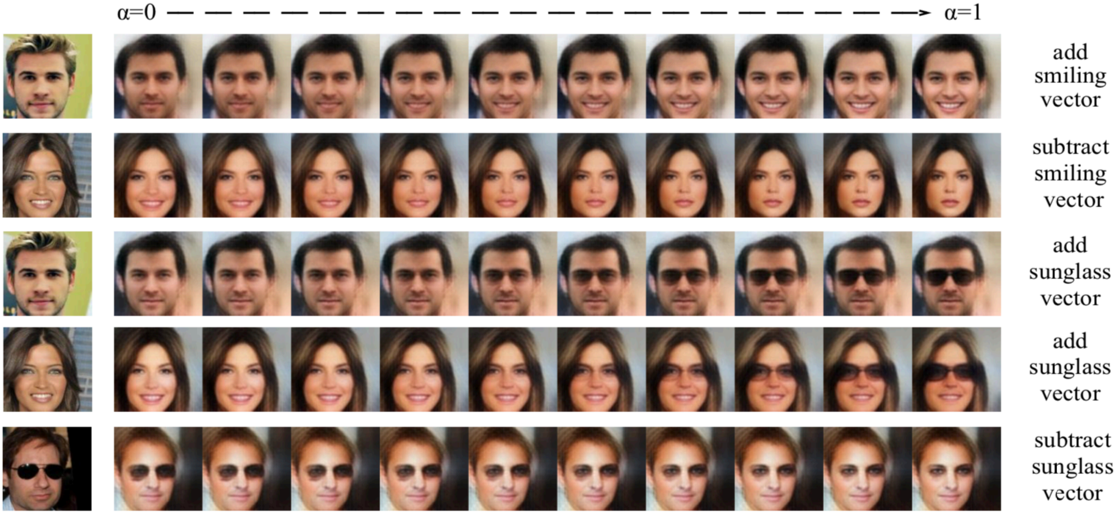
Latent Space Interpolations

- You can often get interesting results by interpolating between two vectors in the latent space:

Ha and Eck, "A neural representation of sketch drawings"

Latent Space Interpolations

- You can often get interesting results by interpolating between two vectors in the latent space:



https://arxiv.org/pdf/1610.00291.pdf

Latent Space Interpolations

- Latent space interpolation of music: https://magenta.tensorflow.org/music-vae

After the break

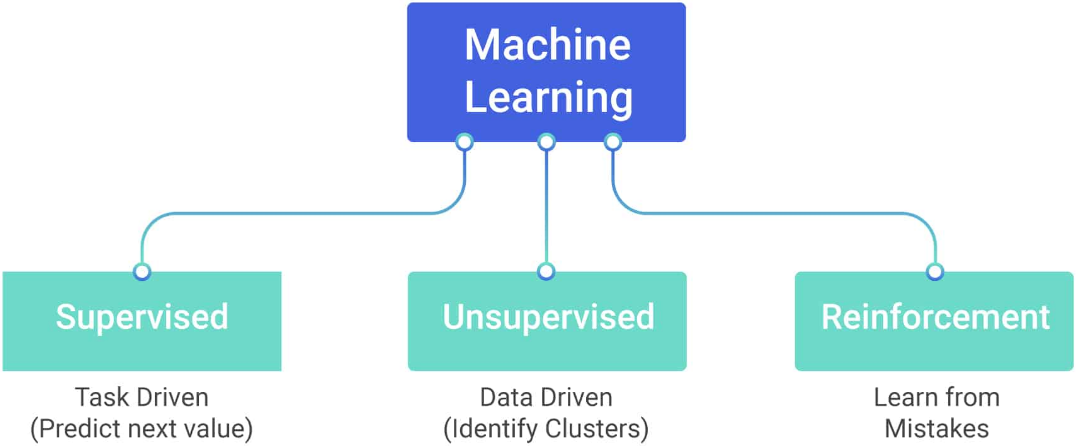
After the break: **Reinforcement Learning: Policy Gradient**

Alpha-Go Trailer --- This is it, folks!

Overview

- Most of this course was about supervised learning, plus a little unsupervised learning.
- Reinforcement learning:
  - Middle ground between supervised and unsupervised learning
  - An agent acts in an environment and receives a reward signal.
- Today: policy gradient (directly do SGD over a stochastic policy using trial-and-error)
- Next lecture: combine policies and Q-learning

Overview



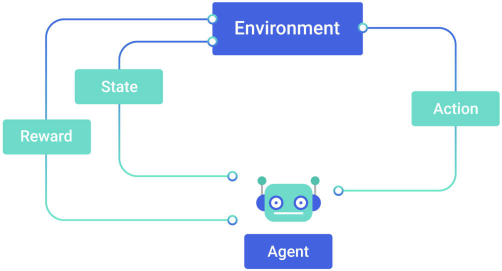
Source: https://perfectial.com/blog/reinforcement-learning-applications/

Overview

How does AI take over the world? Three Steps!

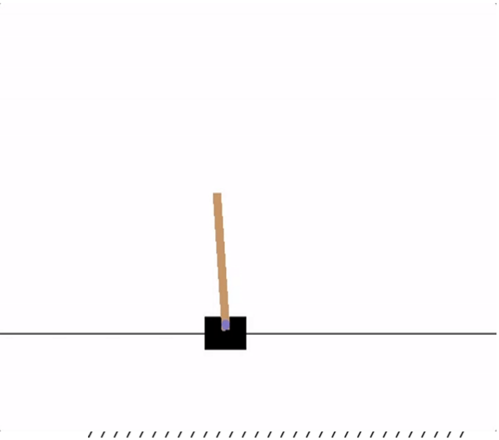
SUPERVISED LEARNING | **UNSUPERVISED LEARNING** | **REINFORCEMENT LEARNING**

Reinforcement learning



- An **agent** interacts with an **environment** (e.g. game of Breakout)
- In each time step t,
  - the agent receives **observations** (e.g. pixels) which give it information about the **state** $\mathbf{s}_t$ (e.g. positions of the ball and paddle)
  - the agent picks an **action** $\mathbf{a}_t$ (e.g. keystrokes) which affects the state
- The agent periodically receives a **reward** $r(\mathbf{s}_t, \mathbf{a}_t)$, which depends on the state and action (e.g. points)
- The agent wants to learn a **policy** $\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$
  - Distribution over actions depending on the current state and parameters $\boldsymbol{\theta}$

Cart-Pole Problem



Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

Source: Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

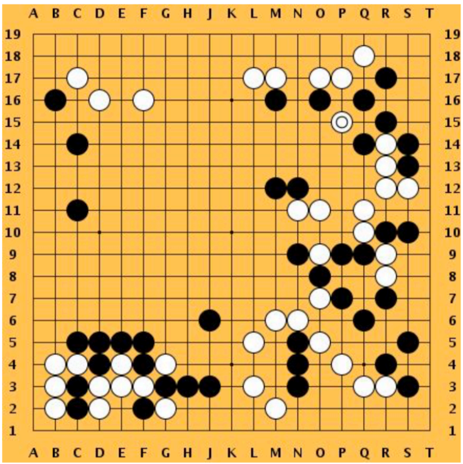
Robot Locomotion

Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

Source: Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Go



Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

Source: Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

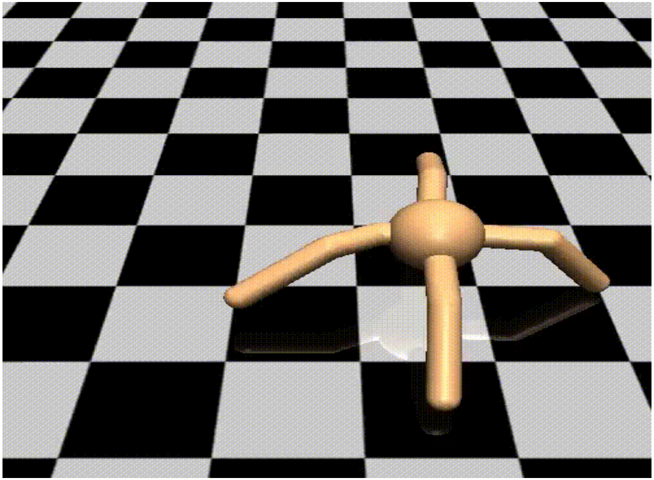
Markov Decision Processes

- The environment is represented as a Markov decision process $\mathcal{M}$.
- Markov assumption: all relevant information is encapsulated in the current state; i.e. the policy, reward, and transitions are all independent of past states given the current state
- Components of an MDP:
  - initial state distribution $p(\mathbf{s}_0)$
  - policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)$
  - transition distribution $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$
  - reward function $r(\mathbf{s}_t, \mathbf{a}_t)$
- Assume a fully observable environment, i.e. $\mathbf{s}_t$ can be observed directly
- Rollout, or trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)$
- Probability of a rollout

$$p(\tau) = p(\mathbf{s}_0)\, \pi_{\boldsymbol{\theta}}(\mathbf{a}_0 \mid \mathbf{s}_0)\, p(\mathbf{s}_1 \mid \mathbf{s}_0, \mathbf{a}_0) \cdots p(\mathbf{s}_T \mid \mathbf{s}_{T-1}, \mathbf{a}_{T-1})\, \pi_{\boldsymbol{\theta}}(\mathbf{a}_T \mid \mathbf{s}_T)$$

Markov Decision Processes

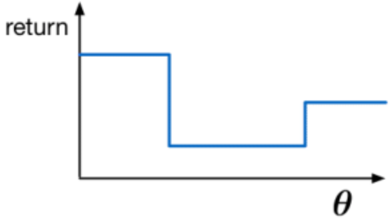
Continuous control in simulation, e.g. teaching an ant to walk



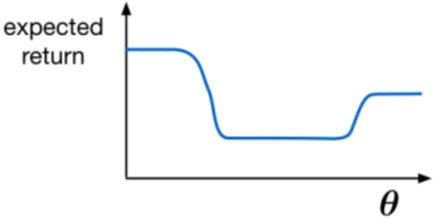
- State: positions, angles, and velocities of the joints
- Actions: apply forces to the joints
- Reward: distance from starting point
- Policy: output of an ordinary MLP, using the state as input
- More environments: https://gym.openai.com/envs/#mujoco

Markov Decision Processes

- Return for a rollout: $r(\tau) = \sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$
  - Note: we're considering a finite horizon T, or number of time steps; we'll consider the infinite horizon case later.
- Goal: maximize the expected return, $R = \mathbb{E}_{p(\tau)}[r(\tau)]$
- The expectation is over both the environment's dynamics and the policy, but we only have control over the policy.
- The stochastic policy is important, since it makes R a continuous function of the policy parameters.
  - Reward functions are often discontinuous, as are the dynamics (e.g. collisions)



deterministic policies



stochastic policies

REINFORCE (Policy Gradient)

- REINFORCE is an elegant algorithm for maximizing the expected return $R = \mathbb{E}_{p(\tau)}[r(\tau)]$.
- Intuition: trial and error
  - Sample a rollout τ. If you get a high reward, try to make it more likely. If you get a low reward, try to make it less likely.
- Interestingly, this can be seen as stochastic gradient ascent on R.

REINFORCE

- Recall the derivative formula for log:

Important Trick!

$$\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\tau) = \frac{\frac{\partial}{\partial \boldsymbol{\theta}} p(\tau)}{p(\tau)} \quad \Longrightarrow \quad \boxed{\frac{\partial}{\partial \boldsymbol{\theta}} p(\tau) = p(\tau) \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\tau)}$$

- Gradient of the expected return:

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\theta}} \mathbb{E}_{p(\tau)}\left[r(\tau)\right] &= \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{\tau} r(\tau) p(\tau) \\
&= \sum_{\tau} r(\tau) \frac{\partial}{\partial \boldsymbol{\theta}} p(\tau) \\
&= \sum_{\tau} r(\tau) p(\tau) \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\tau) \\
&= \mathbb{E}_{p(\tau)}\left[r(\tau) \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\tau)\right]
\end{aligned}$$

- Compute stochastic estimates of this expectation by sampling rollouts.

REINFORCE

- For reference:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathbb{E}_{p(\tau)}\left[r(\tau)\right] = \mathbb{E}_{p(\tau)}\left[r(\tau) \frac{\partial}{\partial \boldsymbol{\theta}} \log p(\tau)\right]$$

- If you get a large reward, make the rollout more likely. If you get a small reward, make it less likely.
- Unpacking the REINFORCE gradient:

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\tau) &= \frac{\partial}{\partial \boldsymbol{\theta}} \log \left[p(\mathbf{s}_0) \prod_{t=0}^{T} \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) \prod_{t=1}^{T} p(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1})\right] \\
&= \frac{\partial}{\partial \boldsymbol{\theta}} \log \prod_{t=0}^{T} \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) \\
&= \sum_{t=0}^{T} \frac{\partial}{\partial \boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)
\end{aligned}$$

- Hence, it tries to make *all* the actions more likely or less likely, depending on the reward. I.e., it doesn't do credit assignment.
  - This is a topic for next lecture.

REINFORCE

Repeat forever:

Sample a rollout $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T)$
$r(\tau) \leftarrow \sum_{k=0}^{T} r(\mathbf{s}_k, \mathbf{a}_k)$
For $t = 0, \dots, T$:

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha r(\tau) \frac{\partial}{\partial \boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)$

- Observation: actions should only be reinforced based on future rewards, since they can't possibly influence past rewards.
- You can show that this still gives unbiased gradient estimates.

Repeat forever:

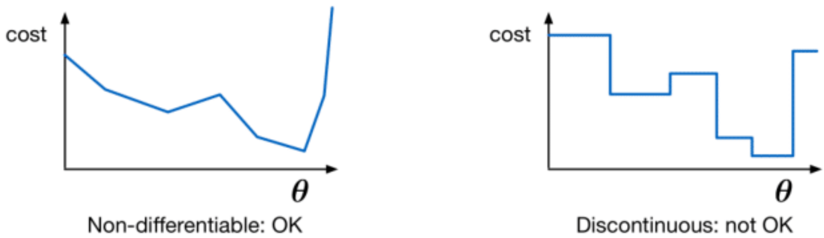
Sample a rollout $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T)$
For $t = 0, \dots, T$:

$r_t(\tau) \leftarrow \sum_{k=t}^{T} r(\mathbf{s}_k, \mathbf{a}_k)$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha r_t(\tau) \frac{\partial}{\partial \boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)$

RL for Classification

- A classification task under RL formulation
  - one time step
  - state $\mathbf{x}$: an image
  - action $\mathbf{a}$: a digit class
  - reward $r(\mathbf{x}, \mathbf{a})$: 1 if correct, 0 if wrong
  - policy $\pi(\mathbf{a} \mid \mathbf{x})$: a distribution over categories
    - Compute using an MLP with softmax outputs – this is a policy network

RL for Classification



- Original solution: use a surrogate loss function, e.g. logistic-cross-entropy
- RL formulation: in each episode, the agent is shown an image, guesses a digit class, and receives a reward of 1 if it's right or 0 if it's wrong
- We'd never actually do it this way, but it will give us an interesting comparison with backprop.

RL for Classification

- Let z_k denote the logits, y_k denote the softmax output, t the integer target, and t_k the target one-hot representation.
- To apply REINFORCE, we sample $\mathbf{a} \sim \pi_{\boldsymbol{\theta}}(\cdot \,|\, \mathbf{x})$ and apply:

$$\begin{aligned}
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha r(\mathbf{a}, \mathbf{t}) \frac{\partial}{\partial \boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a} \,|\, \mathbf{x}) \\
&= \boldsymbol{\theta} + \alpha r(\mathbf{a}, \mathbf{t}) \frac{\partial}{\partial \boldsymbol{\theta}} \log y_a \\
&= \boldsymbol{\theta} + \alpha r(\mathbf{a}, \mathbf{t}) \sum_k (a_k - y_k) \frac{\partial}{\partial \boldsymbol{\theta}} z_k
\end{aligned}$$

- Compare with the logistic regression SGD update:

$$\begin{aligned}
\boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha \frac{\partial}{\partial \boldsymbol{\theta}} \log y_t \\
&\leftarrow \boldsymbol{\theta} + \alpha \sum_k (t_k - y_k) \frac{\partial}{\partial \boldsymbol{\theta}} z_k
\end{aligned}$$

Reward Baselines

- For reference:
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha r(\mathbf{a}, \mathbf{t}) \frac{\partial}{\partial \boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a} \mid \mathbf{x})$$
- Clearly, we can add a constant offset to the reward, and we get an equivalent optimization problem.
- Behavior if $r = 0$ for wrong answers and $r = 1$ for correct answers
  - wrong: do nothing
  - correct: make the action more likely
- If $r = 10$ for wrong answers and $r = 11$ for correct answers
  - wrong: make the action more likely
  - correct: make the action more likely (slightly stronger)
- If $r = -10$ for wrong answers and $r = -9$ for correct answers
  - wrong: make the action less likely
  - correct: make the action less likely (slightly weaker)

Reward Baselines

- Problem: the REINFORCE update depends on arbitrary constant factors added to the reward.
- Observation: we can subtract a baseline b from the reward without biasing the gradient.

$$\begin{aligned}
\mathbb{E}_{p(\tau)}\left[(r(\tau)-b)\frac{\partial}{\partial\boldsymbol{\theta}}\log p(\tau)\right] &= \mathbb{E}_{p(\tau)}\left[r(\tau)\frac{\partial}{\partial\boldsymbol{\theta}}\log p(\tau)\right] - b\mathbb{E}_{p(\tau)}\left[\frac{\partial}{\partial\boldsymbol{\theta}}\log p(\tau)\right] \\
&= \mathbb{E}_{p(\tau)}\left[r(\tau)\frac{\partial}{\partial\boldsymbol{\theta}}\log p(\tau)\right] - b\sum_{\tau} p(\tau)\frac{\partial}{\partial\boldsymbol{\theta}}\log p(\tau) \\
&= \mathbb{E}_{p(\tau)}\left[r(\tau)\frac{\partial}{\partial\boldsymbol{\theta}}\log p(\tau)\right] - b\sum_{\tau}\frac{\partial}{\partial\boldsymbol{\theta}}p(\tau) \\
&= \mathbb{E}_{p(\tau)}\left[r(\tau)\frac{\partial}{\partial\boldsymbol{\theta}}\log p(\tau)\right] - 0
\end{aligned}$$

Hint: $\sum_{\tau} p_\theta(\tau) = 1$

Heads-up: You will show how b can be selected to reduce variance in HW4.

- We'd like to pick a baseline such that good rewards are positive and bad ones are negative.
- $\mathbb{E}[r(\tau)]$ is a good choice of baseline, but we can't always compute it easily. There's lots of research on trying to approximate it.

More Tricks

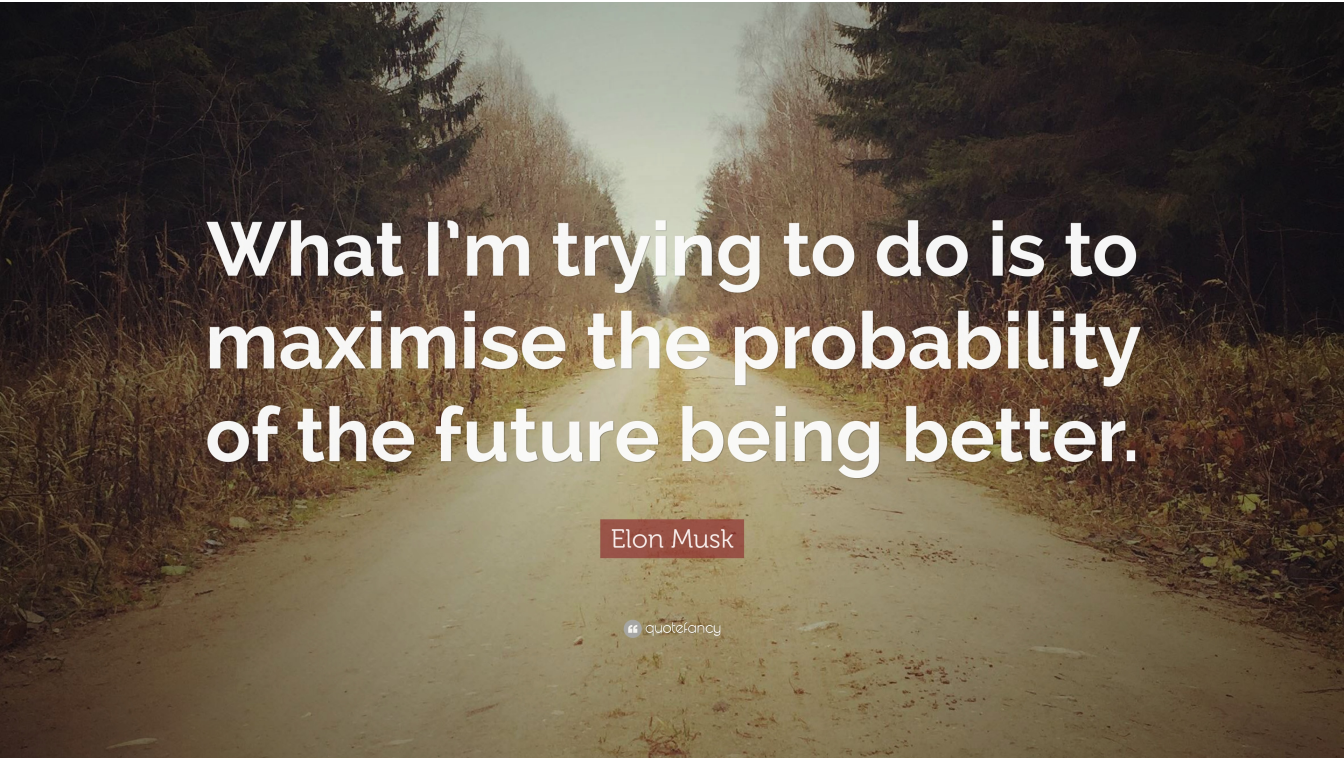
- We left out some more tricks that can make policy gradients work a lot better.
  - Natural policy gradient corrects for the geometry of the space of policies, preventing the policy from changing too quickly.
  - Rather than use the actual return, evaluate actions based on estimates of future returns. This is a class of methods known as actor-critic, which we'll touch upon next lecture.
- Trust region policy optimization (TRPO) and proximal policy optimization (PPO) are modern policy gradient algorithms which are very effective for continuous control problems.

Discussion

- What's so great about backprop and gradient descent?
  - Backprop does credit assignment – it tells you exactly which activations and parameters should be adjusted upwards or downwards to decrease the loss on some training example.
  - REINFORCE doesn't do credit assignment. If a rollout happens to be good, all the actions get reinforced, even if some of them were bad.
  - Reinforcing all the actions as a group leads to random walk behavior.

Discussion

- Why policy gradient?
  - Can handle discontinuous cost functions
  - Don't need an explicit model of the environment, i.e. rewards and dynamics are treated as black boxes
    - Policy gradient is an example of model-free reinforcement learning, since the agent doesn't try to fit a model of the environment
    - Almost everyone thinks model-based approaches are needed for AI, but nobody has a clue how to get it to work


What I'm trying to do is to maximise the probability of the future being better.
Elon Musk
quotefancy