CSCI 599: Deep Learning and its Applications

Lecture 9

Spring 2019
Joseph J. Lim

Disclaimer

- This course is taught for the 2nd time @ USC. This course is 599, and thus an **experimental** course.
- The syllabus, course policy, and grading details **may change** over the semester (**check website!**)
- If you prefer a well-structured course, this is **NOT** a course for you, and I encourage you to take the course next year. We really mean this.
- But, it will be **fun** and **challenging**!

Speakers visiting USC

- **Behnam Neyshabur (NYU) on March 6th**
  - Why Do Neural Networks Learn?
- **Sida Wang (Princeton) on March 6th**
  - Learning Adaptive Language Interfaces Through Interaction
- **Eunsol Choi (Univ. of Washington) on March 7th**
- Yuke Zhu (Stanford) on April 8th
- Jeff Clune (Uber) on TBD

Important Dates

Subject to change!

- ~~Entrance exam: 1/15~~
- ~~Assignment 1: 2/20~~
- ~~Midterm: 2/26~~
- **Project meeting with Instructor #1: 3/6 — 3/8**
- Assignment 2: 3/26
- Project meeting with Instructor #2: 4/1 — 4/3
- Project meeting with TA: 2 times (arranged later)
- Final presentation: 4/23 5-9:00pm **4 hours**

Course Project

Subject to change!

- Computational resource (**be conser**
  $150 Google Cloud credit per student
  $125 Amazon AWS credit per student
- Tentative Schedule for Project
  - ~~Week 5 (2/5): Course Project Team~~
  - **Week 9 (3/5): Course Project Proposal**
  - Week 13 (4/2): Mid-report
  - Week 16 (4/23): Project Presentation (5-9pm) + Report

Proposal

- 1 page write-up (refer to our Piaza post)
- Google slide

Proposal (write-up)

- 1 page write-up (refer to our Piaza post)
  - Goal
  - Motivation
  - Problem formulation (e.g., input/output)
  - Milestones
  - Expected approach + results

Proposal (google slide)

- 1 Google slide (1-3 slides)
  - 70 seconds spotlight (during Lecture 9)
  - 60 teams => 70mins
  - Brief motivation
  - Problem formulation (input & output)
  - Expect approach

Midterm

- You should have all received midterm grades by now.
- You can appeal for midterm questions and answers until next Tuesday (during OHs)
  - Which question + what answers + why
- After we gather all appeals, we will re-grade and release your scores once again (if there is any change)

Assignment 2

- We released the assignment 2.
- There are some corrections made — check Piazza.
- Search before asking questions!

Assignment 1 grading

- We waited for late submissions.
- Currently grading. We will return within 2 weeks.

Cloud credit

- Google cloud and AWS cloud service credits are shared on Piazza
- $125 Amazon per student
- $150 Google per student

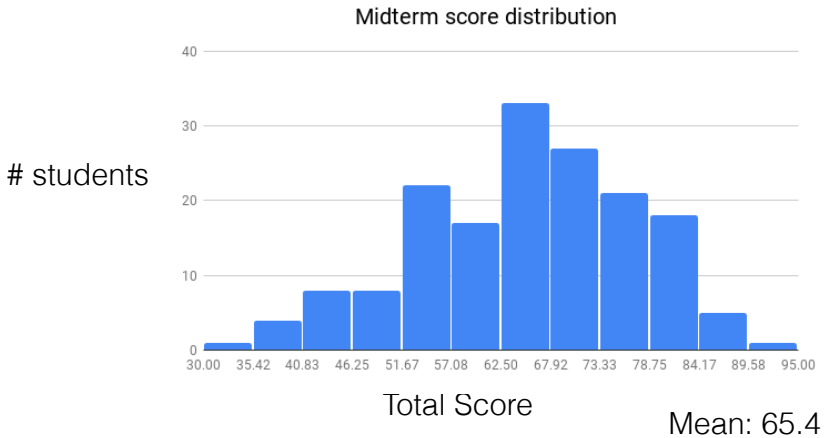
Today's agenda

- Part 1: Midterm Review
- Part 2: Variational Autoencoders
- Part 3: PixelRNN and PixelCNN
- Part 4: Reinforcement Learning

Today's agenda

- Part 1: Midterm Review
- Part 2: Variational Autoencoders
- Part 3: PixelRNN and PixelCNN
- Part 4: Reinforcement Learning

Score Distribution



Mean: 65.4
STD: 12.08

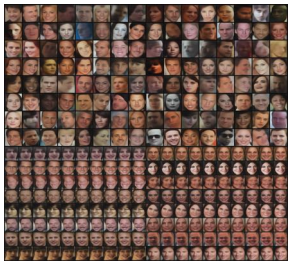
Break Time

See you in 10 mins!

Today's agenda

- Part 1: Midterm Review
- Part 2: Variational Autoencoders
- Part 3: PixelRNN and PixelCNN
- Part 4: Reinforcement Learning

Variational Autoencoder

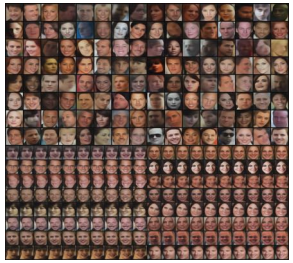


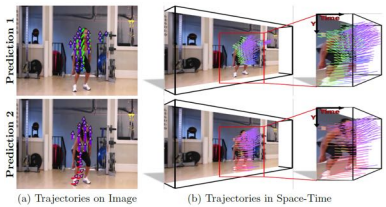
https://github.com/davidsandberg/facenet/wiki/Variational-autoencoder

Walker et. al. The Pose Knows: Video Forecasting by Generating Pose Futures. ICCV 2017

Walker et. al. An Uncertain Future: Forecasting from Static Images using Variational Autoencoders. ECCV 2016

Variational Autoencoder



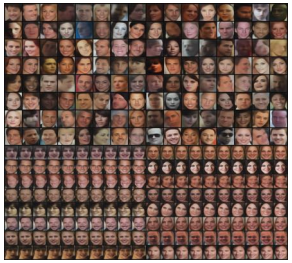


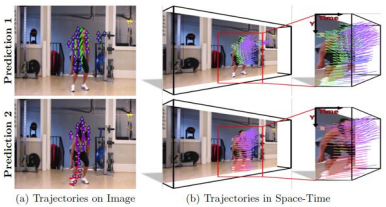
https://github.com/davidsandberg/facenet/wiki/Variational-autoencoder

Walker et. al. The Pose Knows: Video Forecasting by Generating Pose Futures. ICCV 2017

Walker et. al. An Uncertain Future: Forecasting from Static Images using Variational Autoencoders. ECCV 2016

Variational Autoencoder



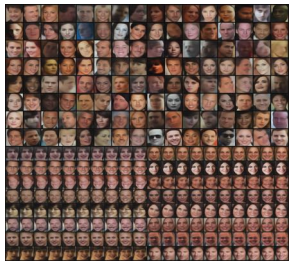


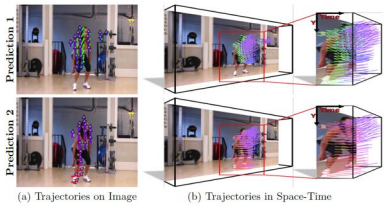
https://github.com/davidsandberg/facenet/wiki/Variational-autoencoder

Walker et. al. The Pose Knows: Video Forecasting by Generating Pose Futures. ICCV 2017

Walker et. al. An Uncertain Future: Forecasting from Static Images using Variational Autoencoders. ECCV 2016

Variational Autoencoder



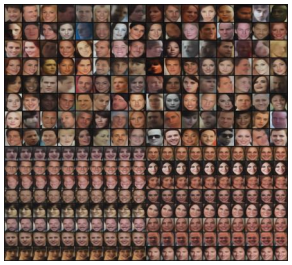


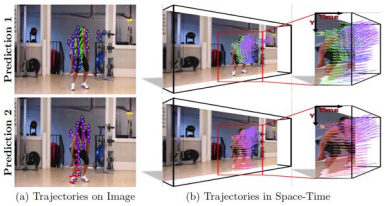
https://github.com/davidsandberg/facenet/wiki/Variational-autoencoder

Walker et. al. The Pose Knows: Video Forecasting by Generating Pose Futures. ICCV 2017

Walker et. al. An Uncertain Future: Forecasting from Static Images using Variational Autoencoders. ECCV 2016

Variational Autoencoder



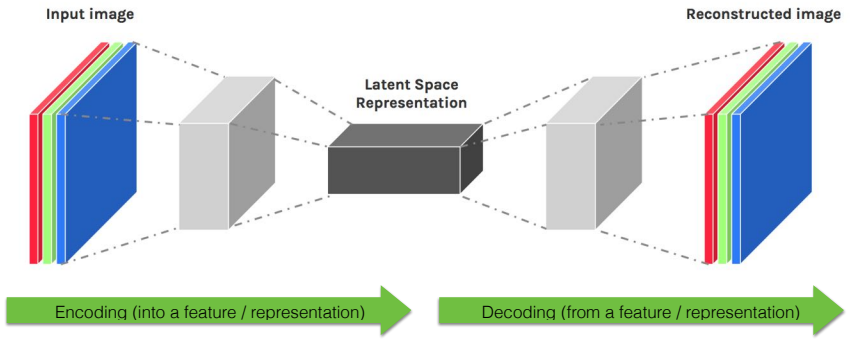


https://github.com/davidsandberg/facenet/wiki/Variational-autoencoder

Walker et. al. The Pose Knows: Video Forecasting by Generating Pose Futures. ICCV 2017

Walker et. al. An Uncertain Future: Forecasting from Static Images using Variational Autoencoders. ECCV 2016

Autoencoder



https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694

Autoencoder

- **Unsupervised** approach
- Learn a lower-dimensional **feature representation** from unlabeled training data

Autoencoder

- Encoder
  - "**Encode**" the information

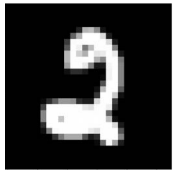
Autoencoder

- Encoder
  - "**Encode**" the information

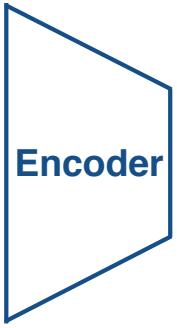
Input Data x

Autoencoder

- Encoder
  - "**Encode**" the information



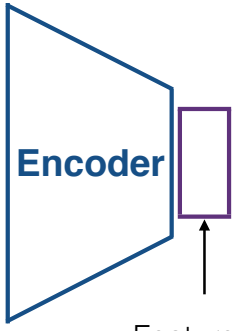
Input Data x



Autoencoder

- Encoder
  - "**Encode**" the information

Input Data x



Features $\mathbf{z}$

Autoencoder

- Encoder
  - Architecture: Multilayer Perceptrons

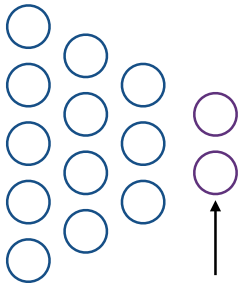
Input Data x

Features $\mathbf{z}$

Autoencoder

- Encoder
  - Architecture: Multilayer Perceptrons

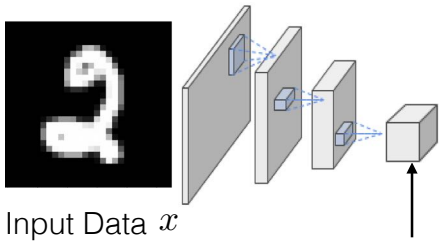
Input Data x



Features $\mathbf{z}$

Autoencoder

- Encoder
  - Architecture: Convolutional layers

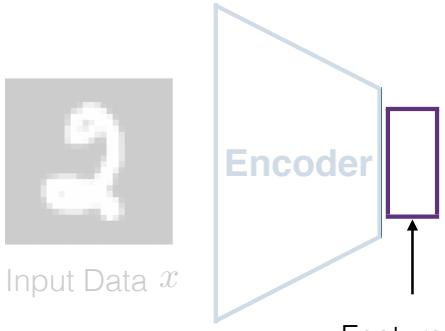


Input Data x

Features $\mathbf{z}$

Autoencoder

- Decoder
  - "**Decode**" the codes

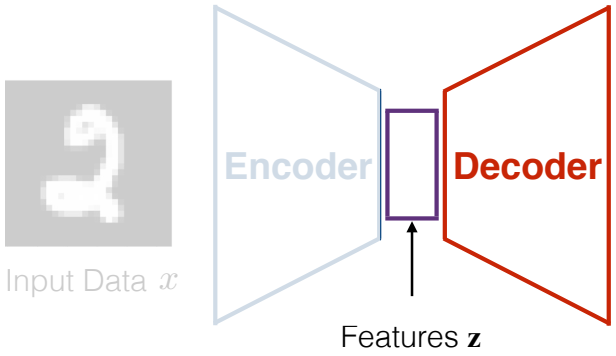


Input Data x

Features $\mathbf{z}$

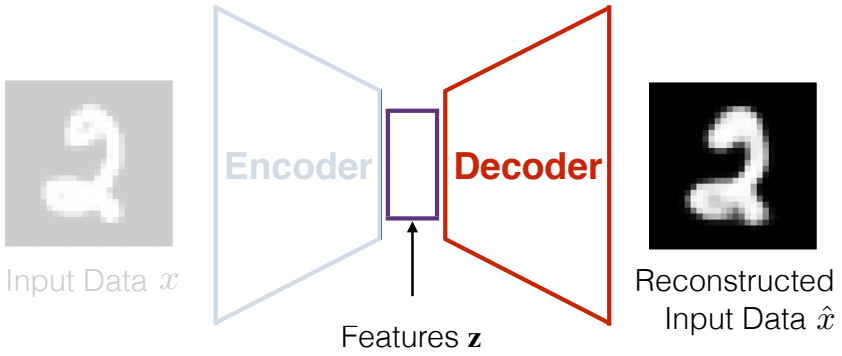
Autoencoder

- Decoder
  - "**Decode**" the codes



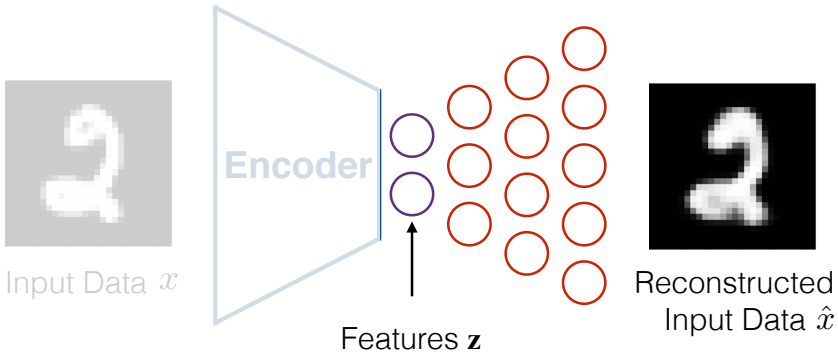
Autoencoder

- Decoder
  - "**Decode**" the codes



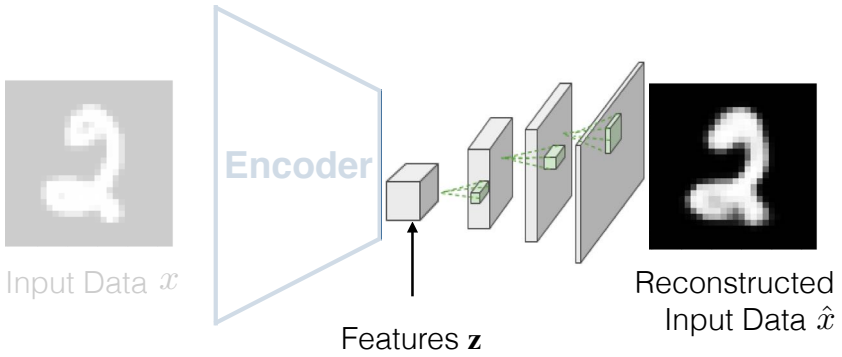
Autoencoder

- Decoder
  - Architecture: Multilayer Perceptrons



Autoencoder

- Decoder
  - Architecture: Deconvolutional layers



Autoencoder

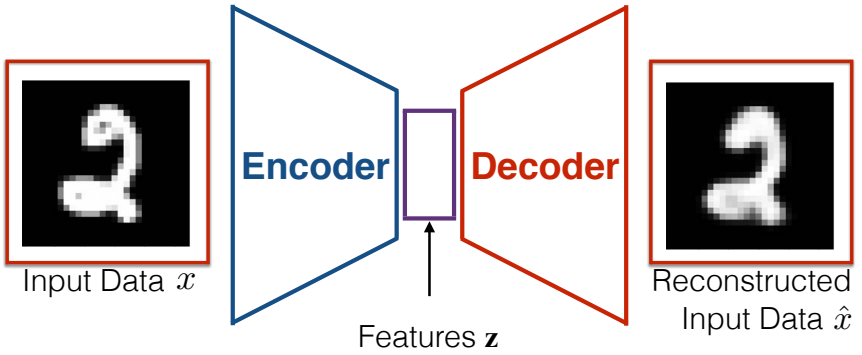
How do we learn the parameters?

- Encoder(x; θ_d)
- Decoder(z; θ_e)

Autoencoder

Jointly train the encoder and the decoder

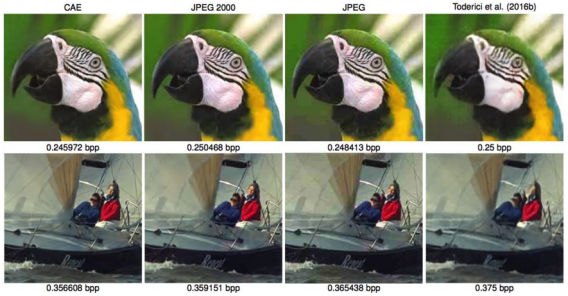
- Reconstruction loss: $distance(x, \hat{x})$



Autoencoder

Applications

- Compression

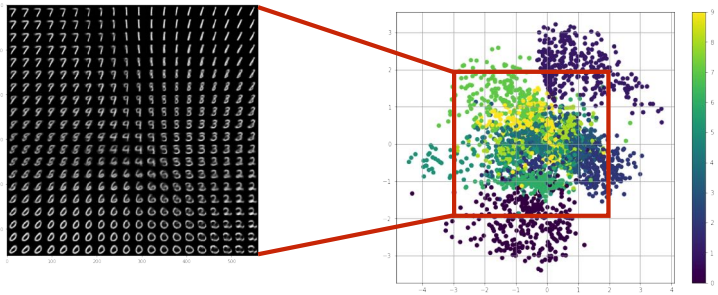


Theis et. al., Lossy Image Compression with Compressive Autoencoders, ICLR 2017

Autoencoder

Applications

- Capture meaningful feature representations

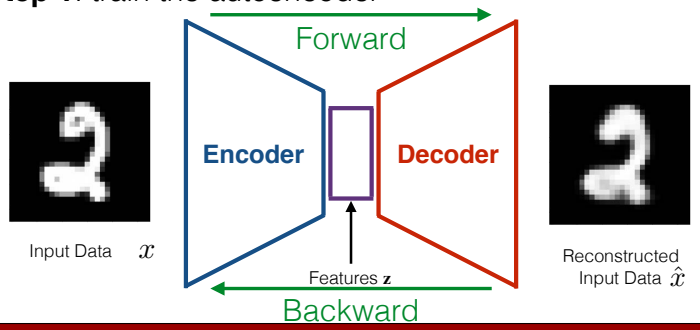


Autoencoder

Applications

- Capture meaningful feature representations

Step 1: train the autoencoder

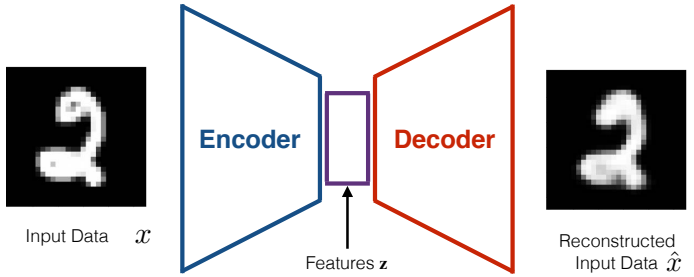


Autoencoder

Applications

- Capture meaningful feature representations

Step 2: throw away the decoder

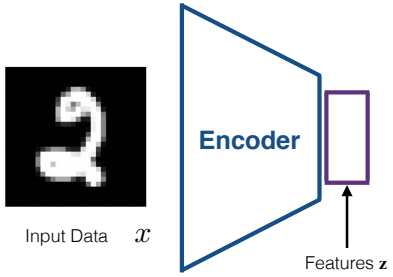


Autoencoder

Applications

- Capture meaningful feature representations

Step 2: throw away the decoder

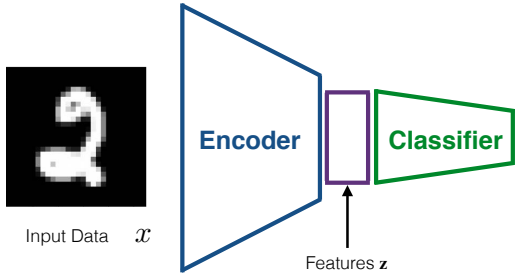


Autoencoder

Applications

- Capture meaningful feature representations

Step 3: add additional layers

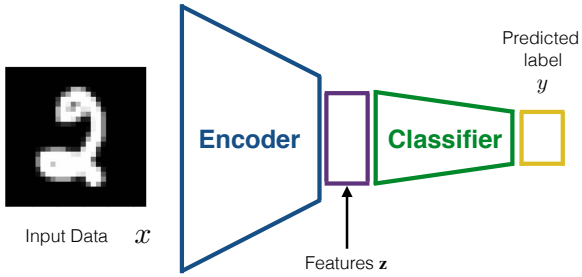


Autoencoder

Applications

- Capture meaningful feature representations

Step 3: add additional layers

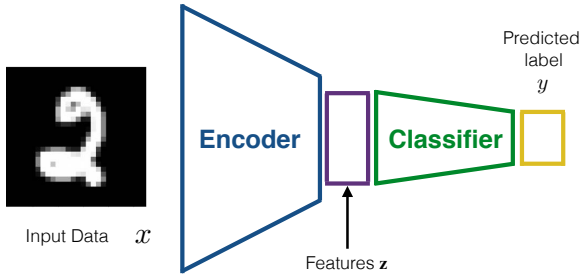


Autoencoder

Applications

- Capture meaningful feature representations

Step 4: jointly train the classifier and fine-tune the encoder

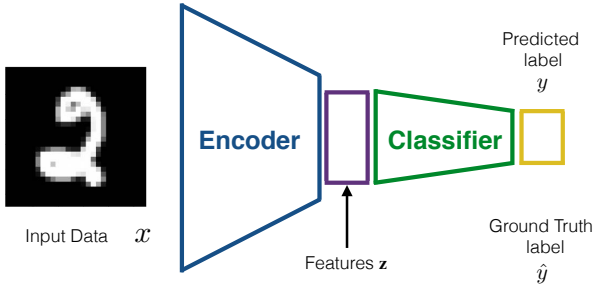


Autoencoder

Applications

- Capture meaningful feature representations

Step 4: jointly train the classifier and fine-tune the encoder

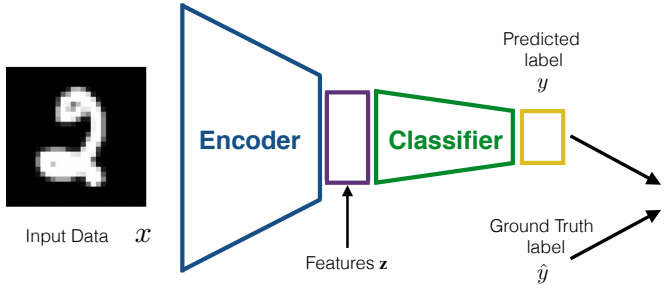


Autoencoder

Applications

- Capture meaningful feature representations

Step 4: jointly train the classifier and fine-tune the encoder

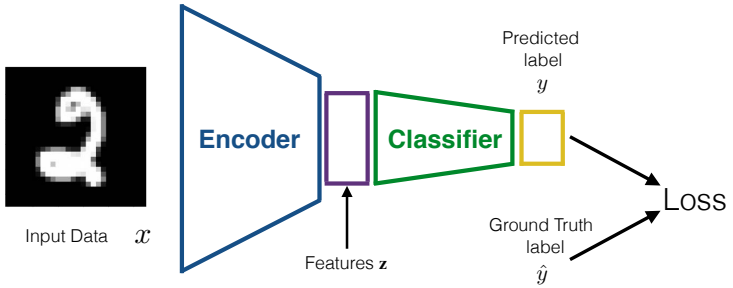


Autoencoder

Applications

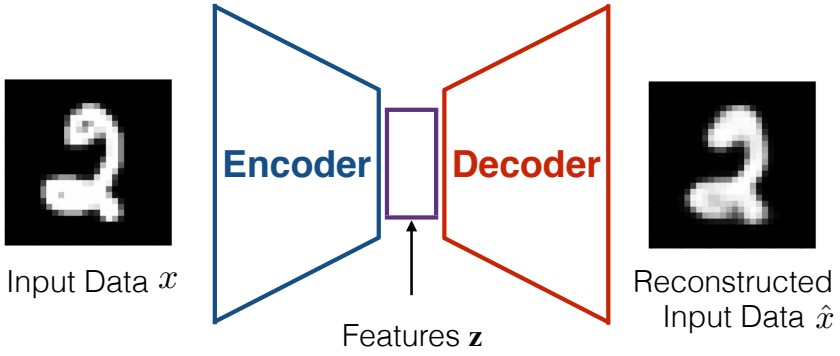
- Capture meaningful feature representations

Step 4: jointly train the classifier and fine-tune the encoder



Autoencoder

Looks pretty good! Doesn't it?



Autoencoder

What about generation?

Autoencoder

What about generation?

Images in a dataset

Autoencoder

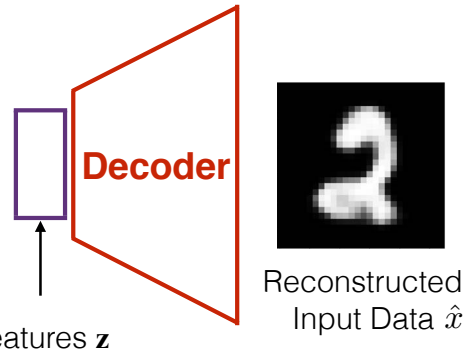
What about generation?

Images in a dataset

Generated images

Autoencoder

What about generation?



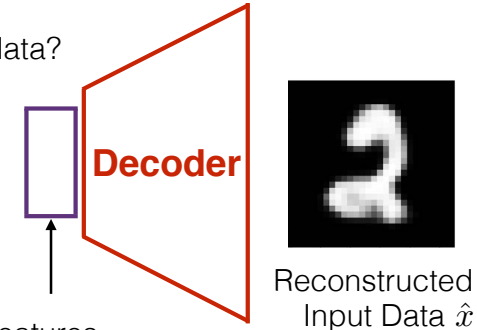
Features $\mathbf{z}$

Reconstructed Input Data $\hat{x}$

Autoencoder

What about generation?

- Given feature vectors
- Can we generate data?



Features $\mathbf{z}$

Reconstructed Input Data $\hat{x}$

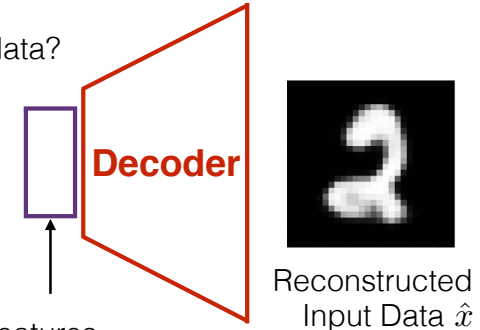
Autoencoder

What about generation?

- Given feature vectors
- Can we generate data?

No!

- What are right feature vectors?



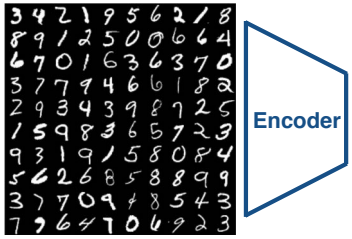
Features $\mathbf{z}$

Reconstructed Input Data $\hat{x}$

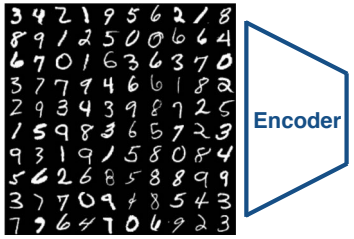
Autoencoder

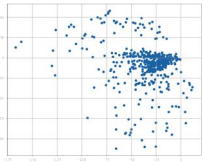
Autoencoder

Autoencoder

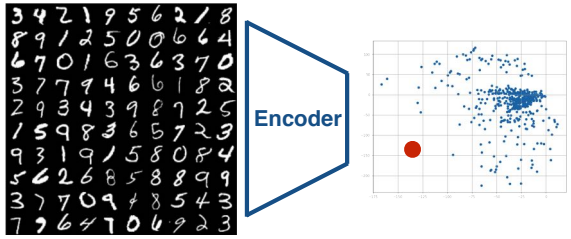


Autoencoder

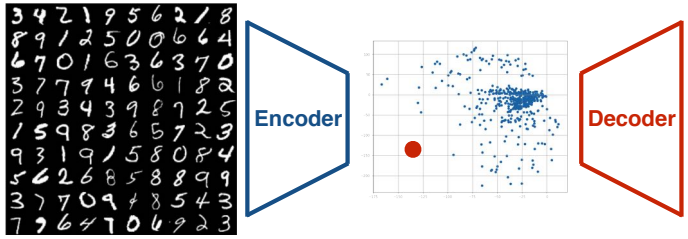




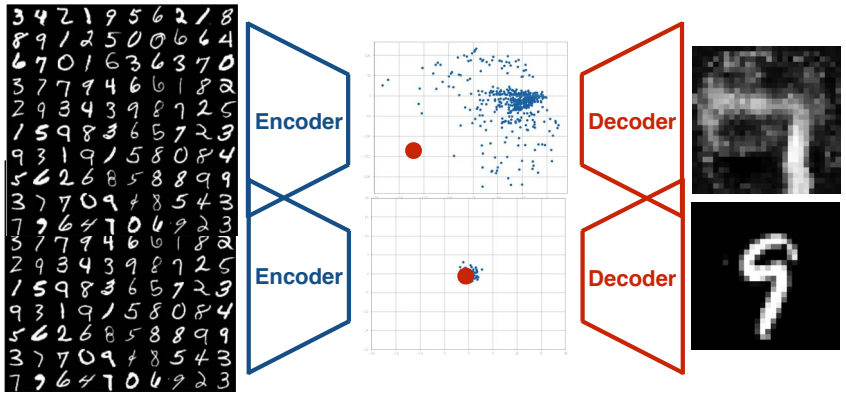
Autoencoder



Autoencoder

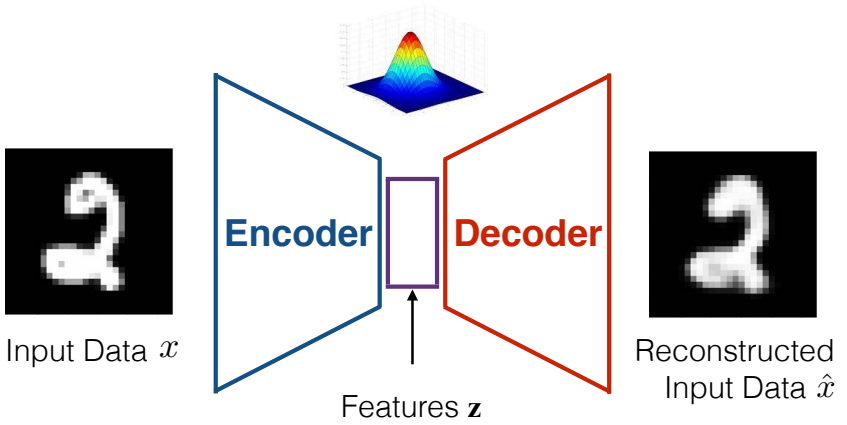


Autoencoder



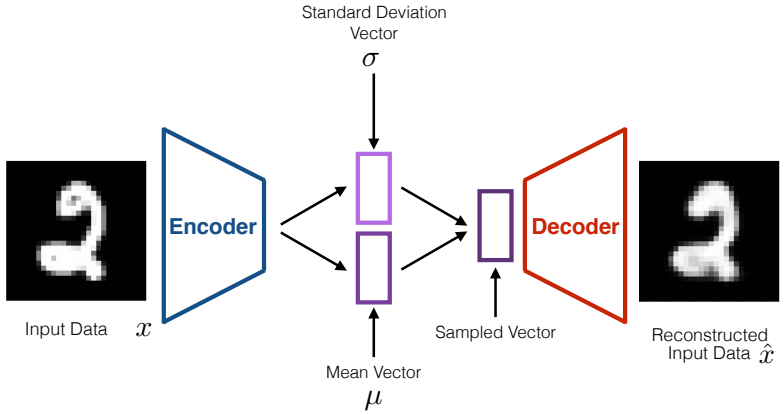
Variational Autoencoder

The trick: constrain the latent distribution



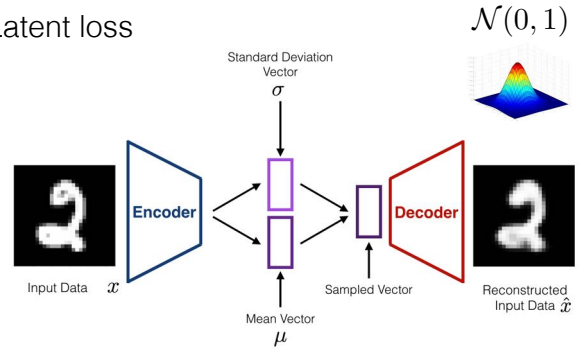
Variational Autoencoder

- Constrain the latent distribution: a Gaussian distribution



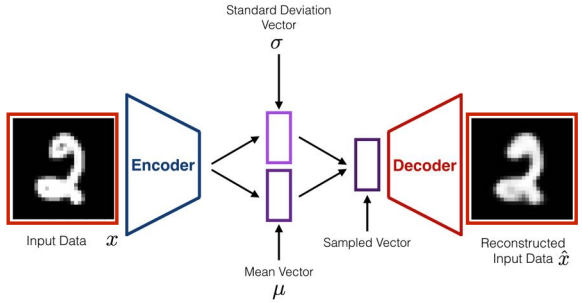
Variational Autoencoder

- Loss
  - Reconstruction loss
  - Latent loss



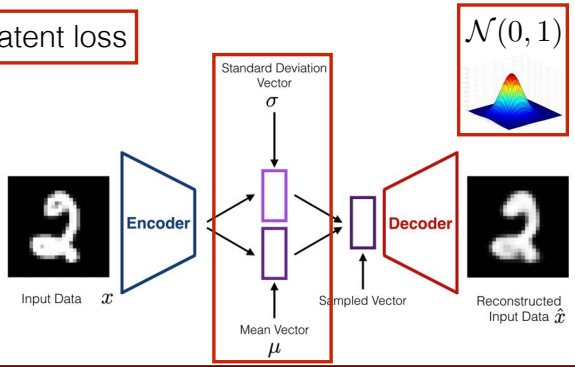
Variational Autoencoder

- Loss
  - Reconstruction loss
  - Latent loss



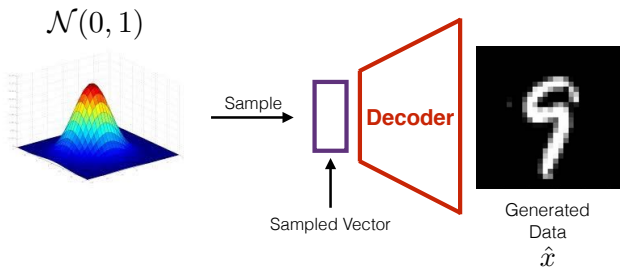
Variational Autoencoder

- Loss
  - Reconstruction loss
  - Latent loss



Variational Autoencoder

- Generation



Autoencoder

Results: Generated from a VAE trained with IAF

Images in the dataset

Generated images

Kingma et. al., Improving Variational Inference with Inverse Autoregressive Flow, NIPS 2016

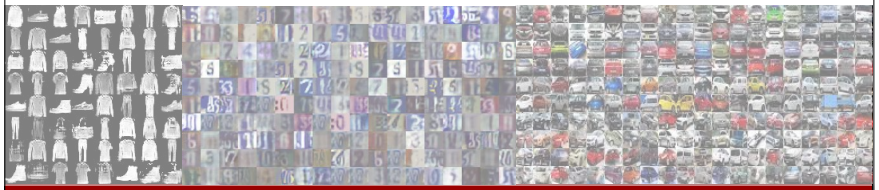
Variational Autoencoder

Seriously, how does a VAE work?

Following couples of slides: Kingma and Welling, Auto-Encoding Variational Bayes, ICLR 2014

Variational Autoencoder

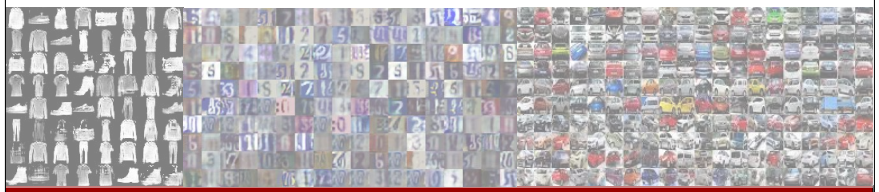
Before being able to generate new data…



Variational Autoencoder

Let's first talk about

how **real data** are generated…



Variational Autoencoder

A dataset: $X = \{x^{(i)}\}_{i=1}^{N}$ is

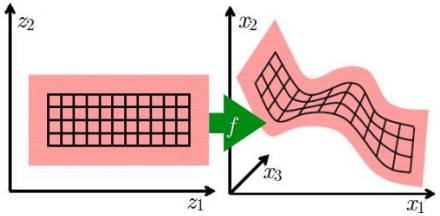
generated from an unobserved random variable z

Variational Autoencoder

A dataset: $X = \{x^{(i)}\}_{i=1}^{N}$ is

generated from an unobserved random variable z

Lower dimensional space

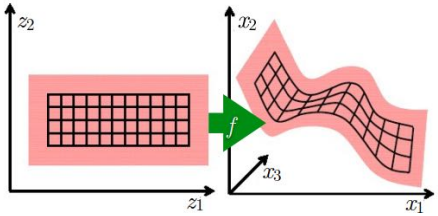


Higher dimensional space

Variational Autoencoder

A dataset: $X = \{x^{(i)}\}_{i=1}^{N}$ is

generated from an unobserved random variable z

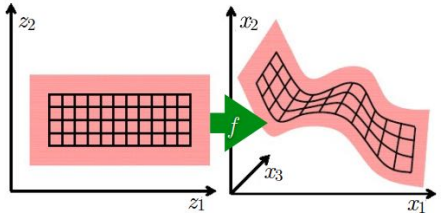


$p_\theta(z)$: prior distribution

Variational Autoencoder

A dataset: $X = \{x^{(i)}\}_{i=1}^{N}$ is

generated from an unobserved random variable z



$p_\theta(z)$: prior distribution $\sim p(z;\phi)$

Variational Autoencoder

A dataset: $X = \{x^{(i)}\}_{i=1}^{N}$ is

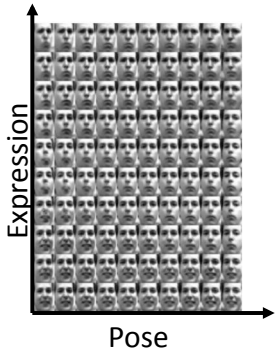
generated from an unobserved random variable z

Example:

A datapoint $\mathbf{x_i}$: 

The latent variable $\mathbf{z_i}$:

{pose: right, expression: sour look}



Variational Autoencoder

The generation process

Variational Autoencoder

The generation process

Step 1: sample from the prior distribution $p_\theta(z)$ to get z

Variational Autoencoder

The generation process

Step 1: sample from the prior distribution $p_\theta(z)$ to get z

Prior distribution $\xrightarrow{p_\theta(z)}$ z

{pose: right,
expression: sour look}

Variational Autoencoder

The generation process

Step 1: sample from the prior distribution $p_\theta(z)$ to get z

Step 2: generate x

from a conditional distribution $p_\theta(x|z)$

Prior distribution $\xrightarrow{p_\theta(z)}$ z

{pose: right,
expression: sour look}

Variational Autoencoder

The generation process

Step 1: sample from the prior distribution $p_\theta(z)$ to get z

Step 2: generate x

from a conditional distribution $p_\theta(x|z)$

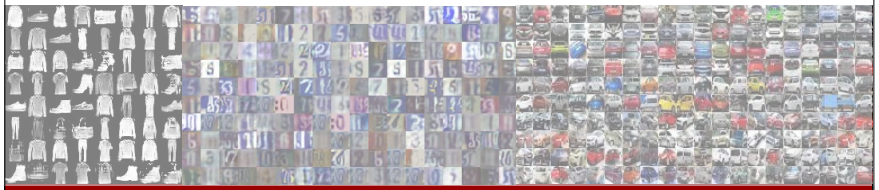
Prior distribution $\xrightarrow{p_\theta(z)}$ z $\xrightarrow{p_\theta(x|z)}$ x

{pose: right,
expression: sour look}

Variational Autoencoder

Explain **observed variables** x

in terms of **latent variables** z

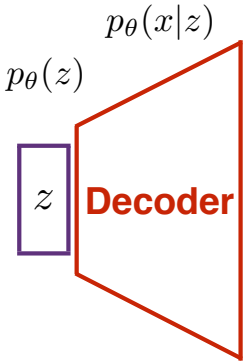


Variational Autoencoder

So, what if we want to generate more data?

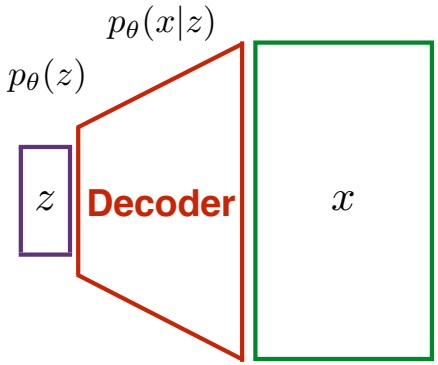
Variational Autoencoder

So, what if we want to generate more data?



Variational Autoencoder

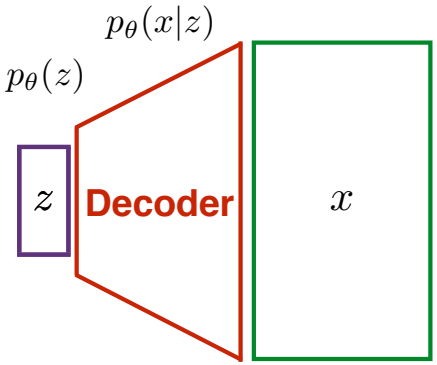
So, what if we want to generate more data?



Variational Autoencoder

So, what if we want to generate more data?

How do we **represent** this model?

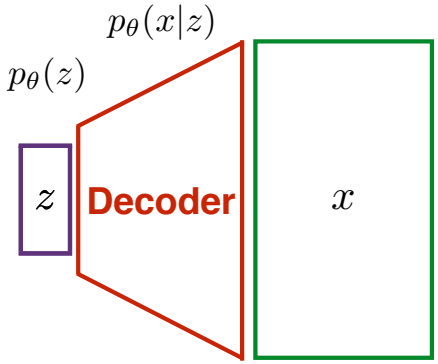


Variational Autoencoder

So, what if we want to generate more data?

How do we **represent** this model?

Choose **prior** as a known distribution, e.g. Gaussian



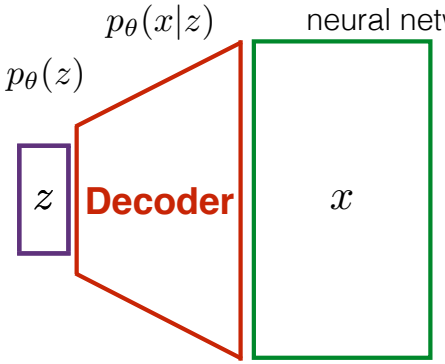
Variational Autoencoder

So, what if we want to generate more data?

How do we **represent** this model?

Represent **likelihood** as a neural network

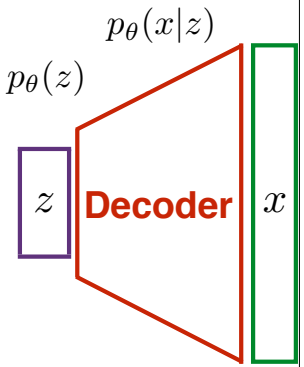
Choose **prior** as a known distribution, e.g. Gaussian



Variational Autoencoder

So, what if we want to generate more data?

How do we **train** this model?

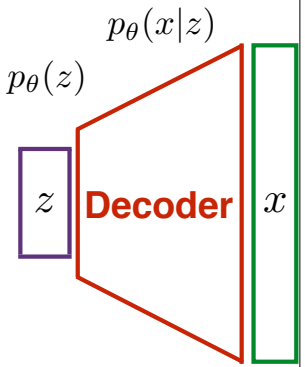


Variational Autoencoder

So, what if we want to generate more data?

How do we **train** this model?

What do we want?



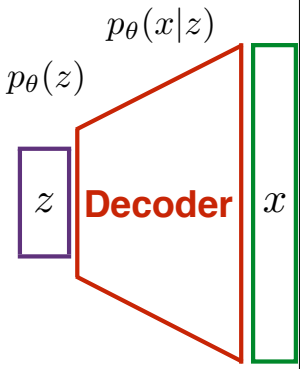
Variational Autoencoder

So, what if we want to generate more data?

How do we **train** this model?

What do we want?

Maximize **the marginal likelihood**:



Variational Autoencoder

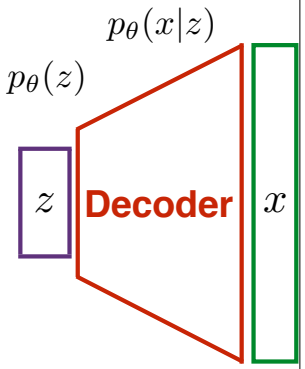
So, what if we want to generate more data?

How do we **train** this model?

What do we want?

Maximize **the marginal likelihood**:

$p_\theta(x)$



Variational Autoencoder

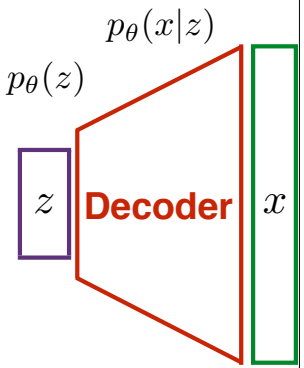
So, what if we want to generate more data?

How do we **train** this model?

What do we want?

Maximize **the marginal likelihood**:

$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$



Variational Autoencoder

The marginal likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

Variational Autoencoder

The marginal likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

Intractable:
enumerate every z

Variational Autoencoder

The marginal likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

Intractable:
enumerate every z

Intractable:
posterior

Variational Autoencoder

The marginal likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

Intractable:
enumerate every z

Intractable:
posterior

The intractability: $p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$

Variational Autoencoder

The marginal likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

Intractable:
enumerate every z

Intractable:
posterior

The intractability: $p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$

The recognition network: $q_\phi(z|x)$

an approximation of the true posterior

Variational Autoencoder

The marginal likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

Intractable: enumerate every z

Intractable: posterior

The intractability: $p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}$

The recognition network: $q_\phi(z|x) \sim q(z|x;\phi)$

an approximation of the true posterior

Variational Autoencoder

Autoencoder

Variational Autoencoder

Autoencoder

x

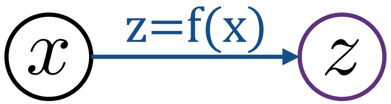
Variational Autoencoder

Autoencoder

x z=f(x)

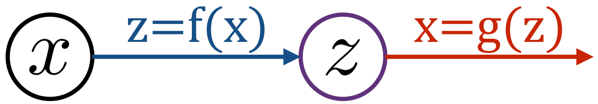
Variational Autoencoder

Autoencoder



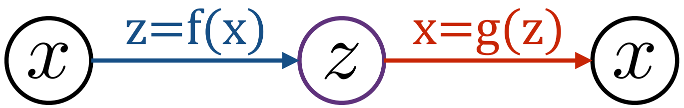
Variational Autoencoder

Autoencoder



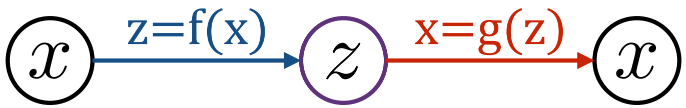
Variational Autoencoder

Autoencoder



Variational Autoencoder

Autoencoder



We learn **deterministic** mappings!

- An encoder f
- A decoder g

Variational Autoencoder

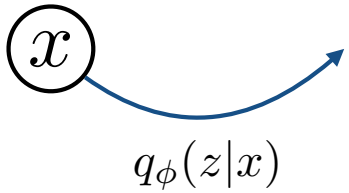
Variational Autoencoder

Variational Autoencoder

Variational Autoencoder

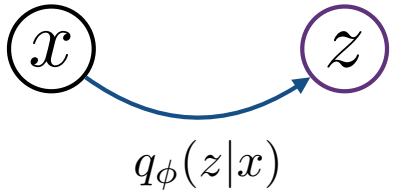
Variational Autoencoder

Variational Autoencoder



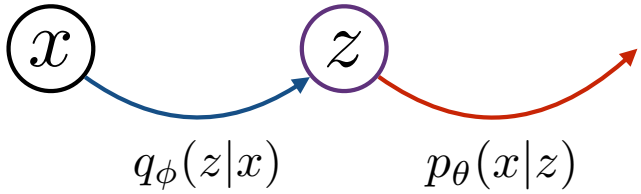
Variational Autoencoder

Variational Autoencoder



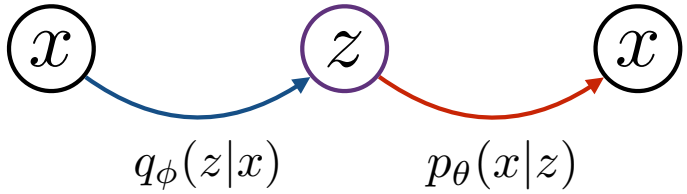
Variational Autoencoder

Variational Autoencoder



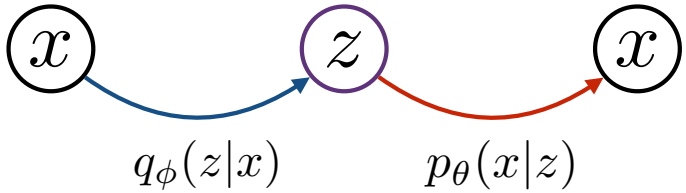
Variational Autoencoder

Variational Autoencoder



Variational Autoencoder

Variational Autoencoder



We learn **probabilistic** mappings!

- A probabilistic encoder $q_\phi(z|x)$
- A probabilistic decoder $p_\theta(x|z)$

Variational Autoencoder

Now we have an encoder and a decoder…

- A probabilistic encoder $q_\phi(z|x)$
- A probabilistic decoder $p_\theta(x|z)$

Variational Autoencoder

Goal

Maximize **the marginal likelihood**

$$p_\theta(x)$$

which is a sum over individual datapoints

$$\log p_\theta(x^{(1)}, ..., x^{(N)}) = \Sigma_{i=1}^{N} \log p_\theta(x^{(i)})$$

Variational Autoencoder

Assumptions

Variational Autoencoder

Assumptions

- We **don't know** $\theta, \emptyset$

Variational Autoencoder

Assumptions

- We **don't know** $\theta, \emptyset$
- We **don't know** $p_\theta(z|x)$

Variational Autoencoder

Assumptions

- We **don't know** $\theta, \emptyset$
- We **don't know** $p_\theta(z|x)$
- Given $\theta, \emptyset$, we **know** the **distributions**

$$p_\theta(z), p_\theta(x|z), q_\phi(z|x)$$

Variational Autoencoder

Assumptions

- We **don't know** $\theta, \emptyset$
- We **don't know** $p_\theta(z|x)$
- Given $\theta, \emptyset$, we **know** the **distributions**

$$p_\theta(z), p_\theta(x|z), q_\phi(z|x)$$

- Each latent variable z_i is generated from $p_\theta(z)$

Variational Autoencoder

Assumptions

- We **don't know** $\theta, \emptyset$
- We **don't know** $p_\theta(z|x)$
- Given $\theta, \emptyset$, we **know** the **distributions**

$$p_\theta(z), p_\theta(x|z), q_\phi(z|x)$$

- Each latent variable z_i is generated from $p_\theta(z)$
- Each data point x_i is generated from $p_\theta(x|z_i)$

Variational Autoencoder

Goal

Maximize **the marginal likelihood**

$$p_\theta(x)$$

which is a sum over individual datapoints

$$\log p_\theta(x^{(1)}, ..., x^{(N)}) = \Sigma_{i=1}^{N} \log p_\theta(x^{(i)})$$

Variational Autoencoder

Now we can work on the marginal likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

Now we can work on the marginal likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

Now we can work on the marginal likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

Latent variable

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

Now we can work on the marginal likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

Now we can work on the marginal likelihood:

$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}
\end{aligned}$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

Now we can work on the marginal likelihood:

$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}
\end{aligned}$$

KL divergence: $D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\log\frac{P(i)}{Q(i)}$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

Now we can work on the marginal likelihood:

$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}$$

KL divergence: $D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\log\frac{P(i)}{Q(i)}$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

Now we can work on the marginal likelihood:

$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}$$

Let's take a look at this

KL divergence: $D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\log\frac{P(i)}{Q(i)}$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) =$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) =$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

- Given the parameters of the decoder, we can compute this through sampling

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) =$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

- Given the parameters of the decoder, we can compute this through sampling
- Given the parameters of the encoder and the Gaussian assumption we made, we can compute this

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) =$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

- Given the parameters of the decoder, we can compute this through sampling
- Given the parameters of the encoder and the Gaussian assumption we made, we can compute this
- We cannot compute this term since $p_\theta(z|x^{(i)})$ is intractable. At least, we know this term is greater than or equal to zeros. (the property of the KL divergence)

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) =$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) =$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

$$\mathcal{L}(x^{(i)}, \theta, \phi)$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) =$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

$$\mathcal{L}(x^{(i)}, \theta, \phi)$$

- **Tractable** ELBO (evidence lower bound)

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) =$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

$$\mathcal{L}(x^{(i)}, \theta, \phi)$$

- **Tractable** ELBO (evidence lower bound)
- We can optimize it using gradient descent
  (both the decoder and the KL term are differentiable)

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) =$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

$$\mathcal{L}(x^{(i)}, \theta, \phi)$$

- **Tractable** ELBO (evidence lower bound)
- We can optimize it using gradient descent (both the decoder and the KL term are differentiable)
- The goal of the training: maximize ELBO to maximize the marginal likelihood

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) \geqq \text{ELBO}$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) \geqq \text{ELBO}$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))$$

- Reconstruct the input data

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

$$\log p_\theta(x^{(i)}) \geqq \text{ELBO}$$

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))$$

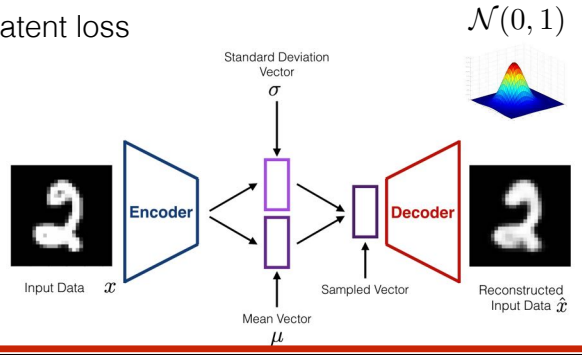
- Reconstruct the input data
- Bring the recognition network closer to the prior

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

- Loss
  - Reconstruction loss
  - Latent loss

Remember This?



Variational Autoencoder

ELBO

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))$$

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

ELBO

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))$$

Reconstruction loss

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

ELBO

$$\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))$$

Reconstruction loss Latent loss

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Variational Autoencoder

Goal

Maximize **the marginal likelihood**

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

Variational Autoencoder

Goal

Maximize **the marginal likelihood**

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

We can now maximize the likelihood
by maximizing

ELBO

$$\mathbf{E}_z \left[\log p_\theta(x^{(i)} \mid z) \right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))$$

Variational Autoencoder

Alternative

Variational Autoencoder

Again, we want to maximize: $p(x) = \int p(x|z) p(z) dz$

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

Again, we want to maximize: $p(x) = \int p(x|z)p(z)dz$

Intractable: enumerate every z

Intractable: posterior

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

Again, we want to maximize: $p(x) = \int p(x|z)p(z)dz$

Intractable: enumerate every z

Intractable: posterior

Instead, we can find a lower bound of it

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

Again, we want to maximize: $p(x) = \int p(x|z)p(z)dz$

Intractable: enumerate every z

Intractable: posterior

Instead, we can find a lower bound of it

$$p(x) = \int p(x|z)p(z)dz \quad \geq \quad a\ lower\ bound$$

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

Again, we want to maximize: $p(x) = \int p(x|z)p(z)dz$

Intractable: enumerate every z

Intractable: posterior

Instead, we can find a lower bound of it

$$p(x) = \int p(x|z)p(z)dz \quad \geq \quad a\ lower\ bound$$

By maximizing the lower bound,

we can maximize the likelihood

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

Again, we want to maximize: $p(x) = \int p(x|z)p(z)dz$

Intractable: enumerate every z

Intractable: posterior

Instead, we can find a lower bound of it

$p(x) = \int p(x|z)p(z)dz \geq$ *a lower bound*

By maximizing the lower bound,

we can maximize the likelihood

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

Again, we want to maximize: $p(x) = \int p(x|z)p(z)dz$

Intractable: enumerate every z

Intractable: posterior

Instead, we can find a lower bound of it

$p(x) = \int p(x|z)p(z)dz \geq$ *a lower bound*

By maximizing the lower bound,

we can maximize the likelihood

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Joseph J. Lim CSCI 599 @ USC Lecture 9

Variational Autoencoder

Again, we want to maximize: $p(x) = \int p(x|z)p(z)dz$

Now we do not care about

how to decompose the likelihood.

We only want to find **a lower bound** of it.

By maximizing the lower bound,

we can maximize the likelihood

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Joseph J. Lim CSCI 599 @ USC Lecture 9

Variational Autoencoder

The marginal likelihood: $p(x) = \int p(x|z)p(z)dz$

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Joseph J. Lim CSCI 599 @ USC Lecture 9

Variational Autoencoder

The marginal likelihood: $p(x) = \int p(x|z)p(z)dz$

Proposal $= \int p(x|z)p(z)\frac{q(z)}{q(z)}dz$

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

The marginal likelihood: $p(x) = \int p(x|z)p(z)dz$

Proposal $= \int p(x|z)p(z)\frac{q(z)}{q(z)}dz$

Importance Weight $= \int p(x|z)\frac{p(z)}{q(z)}q(z)dz$

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

The marginal likelihood: $p(x) = \int p(x|z)p(z)dz$

Proposal $= \int p(x|z)p(z)\frac{q(z)}{q(z)}dz$

Importance Weight $= \int p(x|z)\frac{p(z)}{q(z)}q(z)dz$

Jensen's inequality

$\log \int p(x)g(x)dx \geq \int p(x)\log g(x)dx$

$\log p(x) \geq \int q(z)\log\left(p(x|z)\frac{p(z)}{q(z)}\right)dz$

$= \int q(z)\log p(x|z) - \int q(z)\log\frac{q(z)}{p(z)}dz$

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

The marginal likelihood: $p(x) = \int p(x|z)p(z)dz$

Proposal $= \int p(x|z)p(z)\frac{q(z)}{q(z)}dz$

Importance Weight $= \int p(x|z)\frac{p(z)}{q(z)}q(z)dz$

Jensen's inequality

$\log \int p(x)g(x)dx \geq \int p(x)\log g(x)dx$

$\log p(x) \geq \int q(z)\log\left(p(x|z)\frac{p(z)}{q(z)}\right)dz$

$= \int q(z)\log p(x|z) - \int q(z)\log\frac{q(z)}{p(z)}dz$

$\mathbb{E}_{q(z)}[\log p(x|z)] - D_{KL}(q(z)||p(z))$

Tractable ELBO (evidence lower bound)

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

The marginal likelihood: $p(x) = \int p(x|z)p(z)dz$

Proposal $= \int p(x|z)p(z)\frac{q(z)}{q(z)}dz$

Importance Weight $= \int p(x|z)\frac{p(z)}{q(z)}q(z)dz$

Jensen's inequality $\log p(x) \geq \int q(z) \log \left(p(x|z)\frac{p(z)}{q(z)}\right)dz$

$\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$

$= \int q(z) \log p(x|z) - \int q(z) \log \frac{q(z)}{p(z)}dz$

$\mathbb{E}_{q(z)}[\log p(x|z)] - D_{KL}(q(z)||p(z))$

Tractable ELBO (evidence lower bound)

Shakir Mohamed's talk at the Deep Learning Summer School 2016

Variational Autoencoder

Pros

- There is a clear and recognized way to evaluate the quality of the model (log-likelihood)
- Trained recognition networks can learn to produce useful feature

Cons

- Tend to generate blurry results (compared to GANs)

Break Time

See you in 10 mins!

Today's agenda

- Part 1: Midterm Review
- Part 2: Variational Autoencoders
- Part 3: PixelRNN and PixelCNN
- Part 4: Reinforcement Learning

The Marginal Likelihood

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

The Marginal Likelihood

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- **Variational Autoencoders** look pretty promising!

The Marginal Likelihood

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- **Variational Autoencoders** look pretty promising!
- But… we do not directly maximize the marginal likelihood

The Marginal Likelihood

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

- **Variational Autoencoders** look pretty promising!
- But… we do not directly maximize the marginal likelihood
- Instead, we maximize the variational lower bound (ELBO)

The Marginal Likelihood

$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

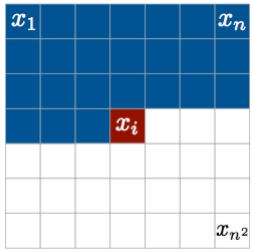
-
-
-

Why don't we directly maximize the marginal likelihood?

(ELBO)

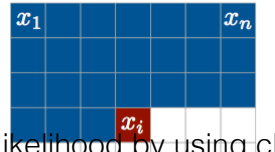
Decompose An Image

Imagine the process of generating images as a **sequential** process.



Following couples of slides: van den Oord et. al., Pixel Recurrent Neural Networks, ICML 2016

Decompose An Image



Decompose the likelihood by using chain rule

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

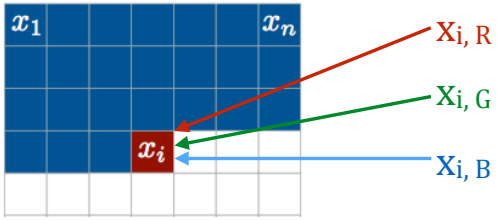
The product of **conditional distributions** over the pixels

Decompose An Image

How do we deal with RGB values?

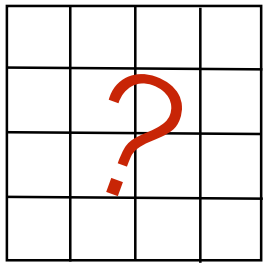
Every value is conditioned on the **other channels** as well as on all the **previously generated pixels**

$$p(x_{i,R}|\mathbf{x}_{<i}) p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R}) p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G})$$



PixelRNN

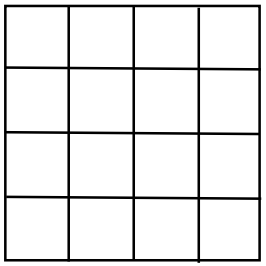
How do we decide the order of pixels?



PixelRNN

How do we decide the order of pixels?

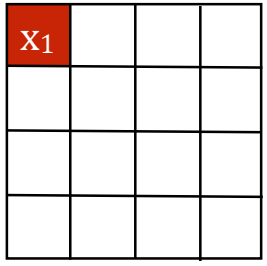
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

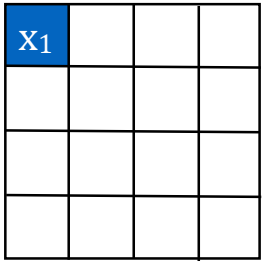
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

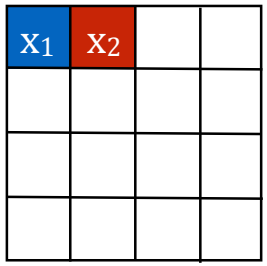
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

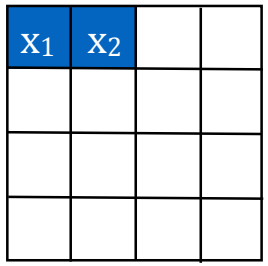
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

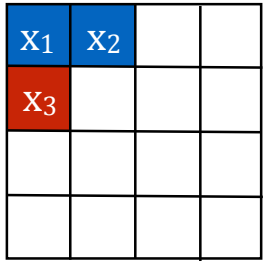
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

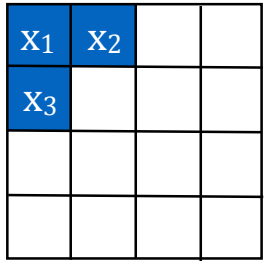
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

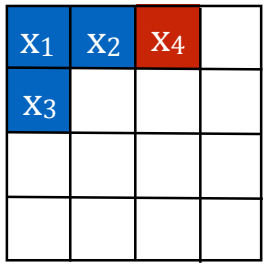
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

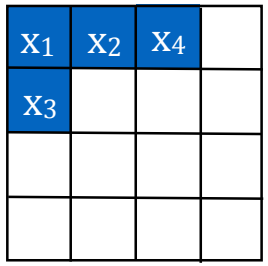
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

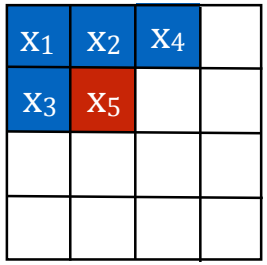
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

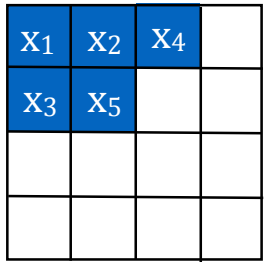
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

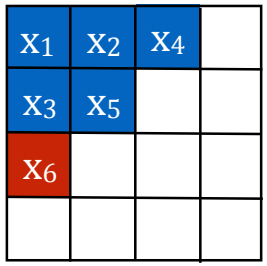
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

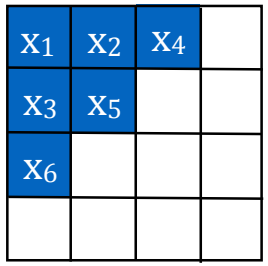
Start from the top-left corner



PixelRNN

How do we decide the order of pixels?

Start from the top-left corner

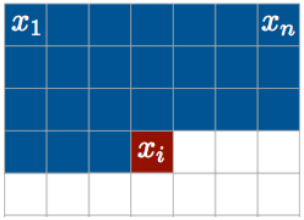


PixelRNN

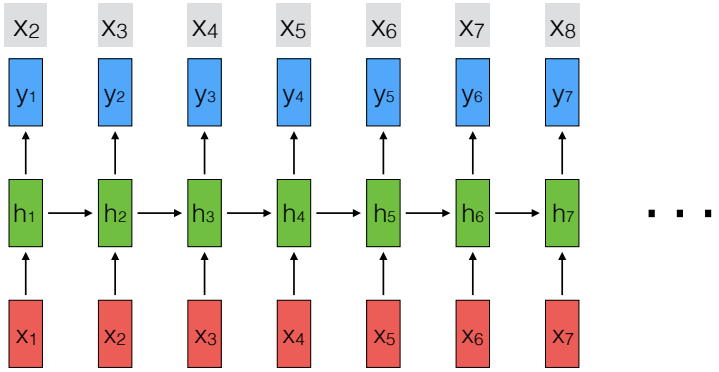
How do we model the pixel value **dependency**?

Remember the architectures we just learned?

RNN (LSTM) !



PixelRNN

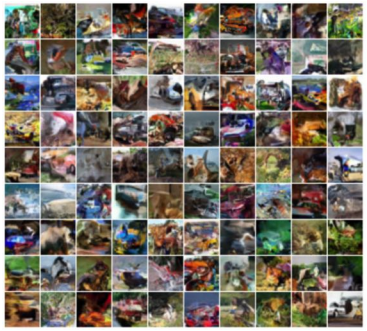


PixelRNN

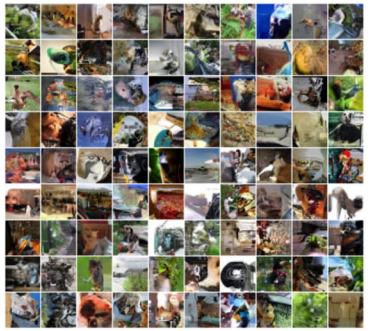
Results: Image completion

PixelRNN

Results: Image generation



Cifar10



ImageNet (32x32)

PixelRNN

Pros

- Explicitly maximize the likelihood
- There is a clear and recognized way to evaluate the quality of the model (log-likelihood)
- Generate sharper images

Cons

- The sequential generation is slow

PixelRNN

Pros

- Explicitly maximize the likelihood
- There is a clear and recognized way to evaluate the quality of the model (log-likelihood)
- Generate sharper images

Cons

- The sequential generation is slow

PixelCNN [van den Oord et. al. ICML 2016]

Summary

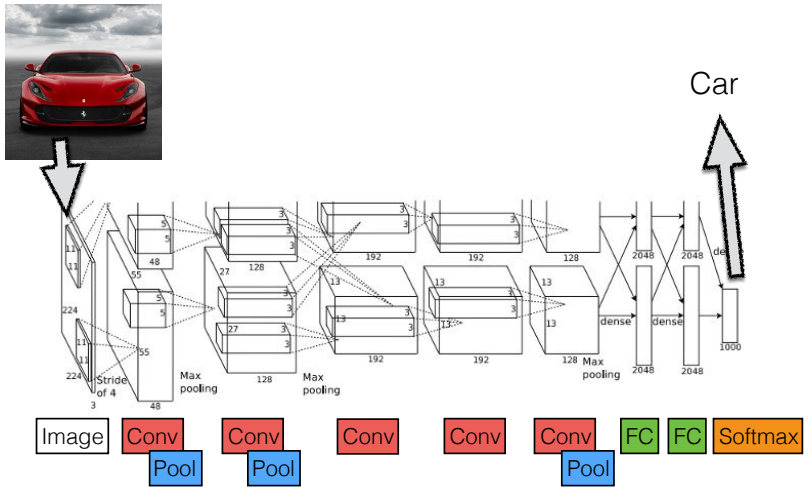
Break Time

See you in 10 mins!

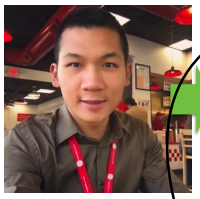
Today's agenda

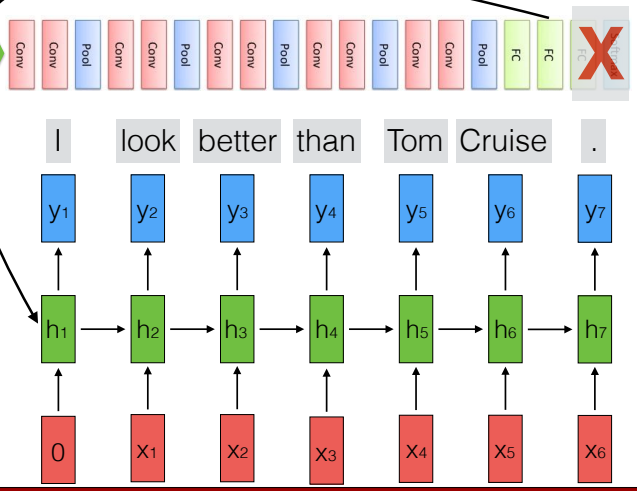
- Part 1: Midterm Review
- Part 2: Variational Autoencoders
- Part 3: PixelRNN and PixelCNN
- Part 4: Reinforcement Learning

Recap: CNNs

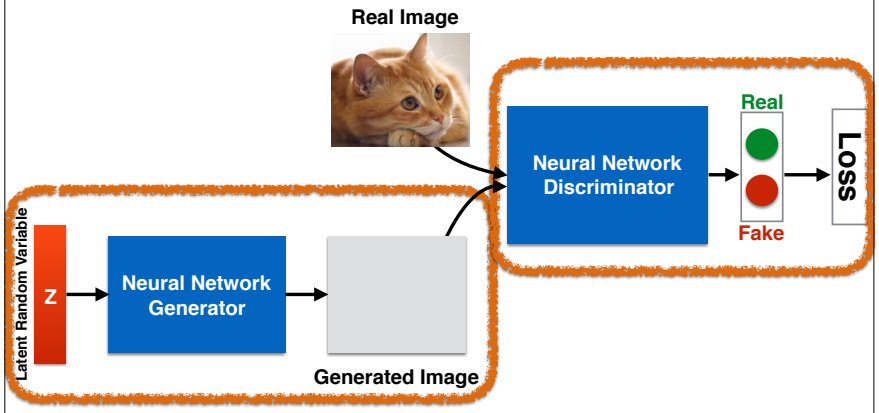


Recap: RNNs



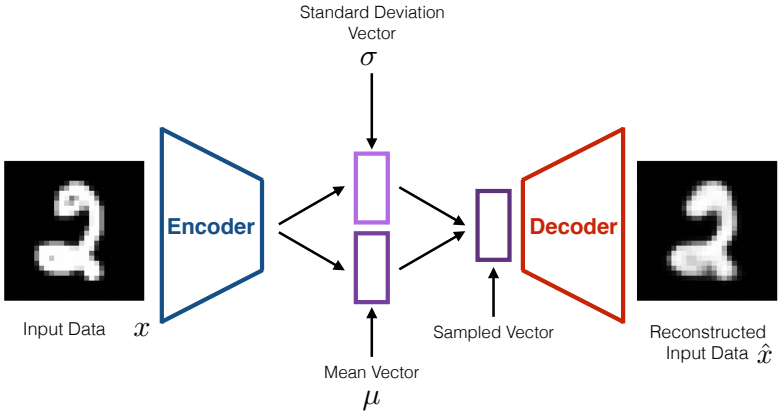


Recap: GANs



Inspired by https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Recap: AE & VAE



So far…

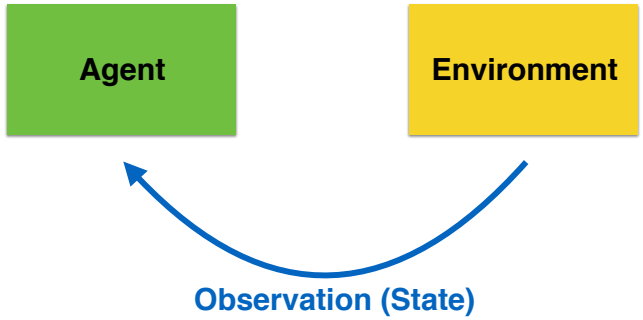
- Mostly,
  - Supervised Learning
  - Unsupervised Learning

Reinforcement Learning

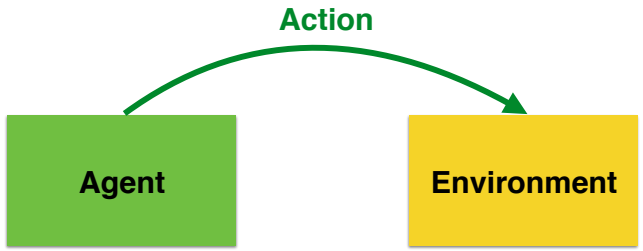
Environment

Reinforcement Learning

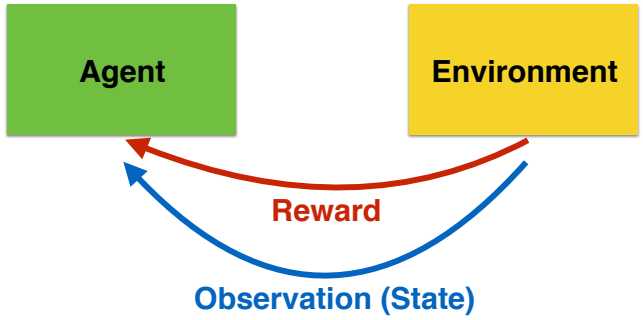
Reinforcement Learning



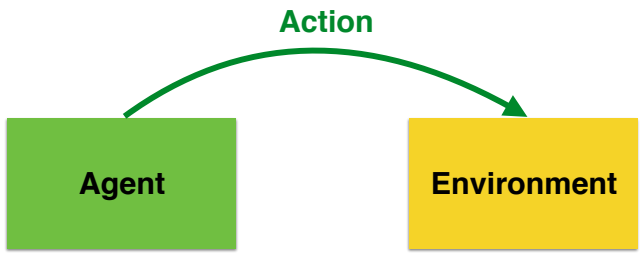
Reinforcement Learning



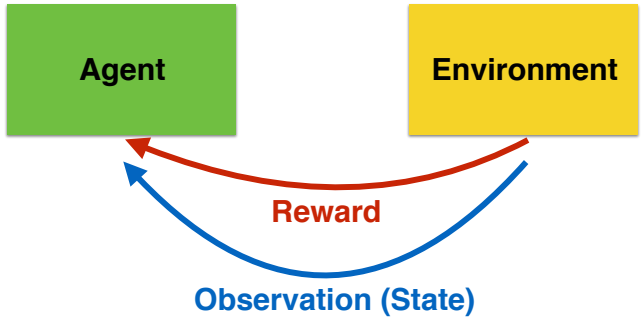
Reinforcement Learning



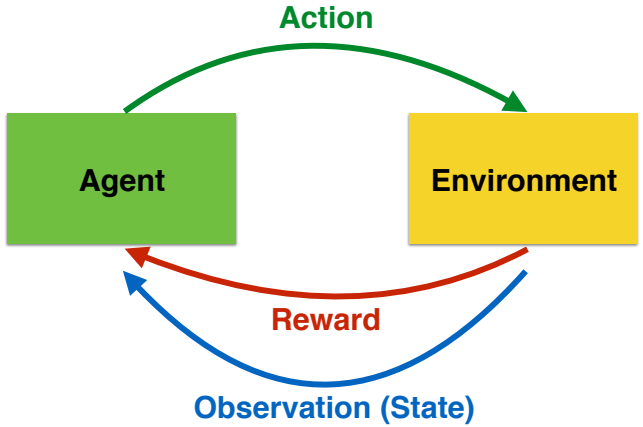
Reinforcement Learning



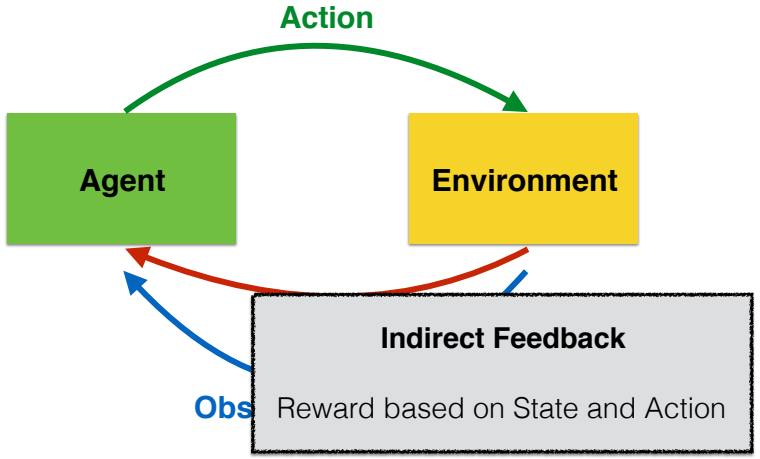
Reinforcement Learning



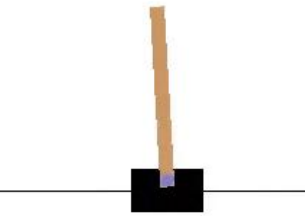
Reinforcement Learning



Reinforcement Learning



Cart Pole Problem



Left Right

Cart Pole Problem

RL training over time

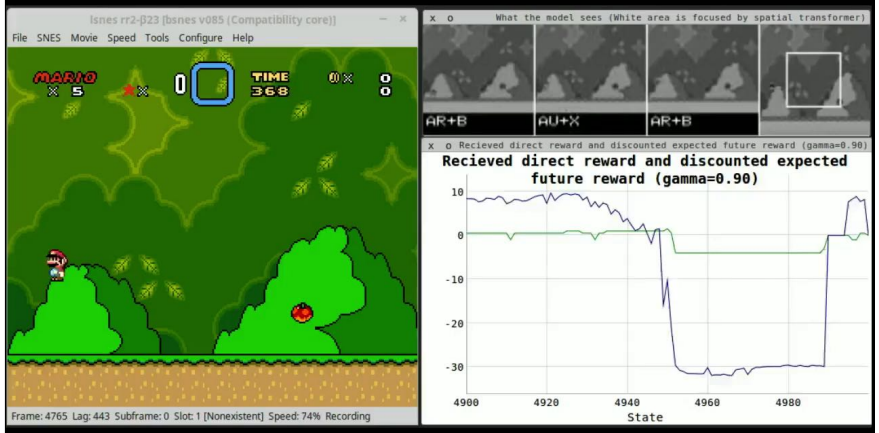
Google Deepmind DQN playing
Atari Breakout

Setup:
NVIDIA GTX 690
i7-3770K - 16 GB RAM
Ubuntu 16.04 LTS
Google Deepmind DQN

Robot Grasping

Levine et. al., Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. 2016.

Mario AI



https://github.com/aleju/mario-ai

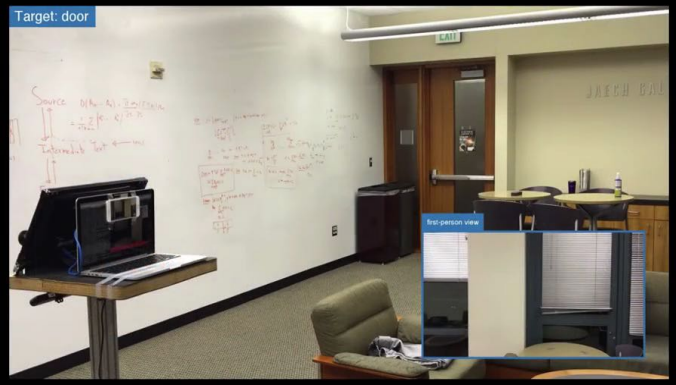
Locomotion

Emergence of Locomotion Behaviours in Rich Environments

Emergence of Locomotion Behaviours in Rich Environments [Heess et al]

Navigation

Model trained with simulation + real images: Go to Door



Zhu et. al. Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. ICRA 2017.

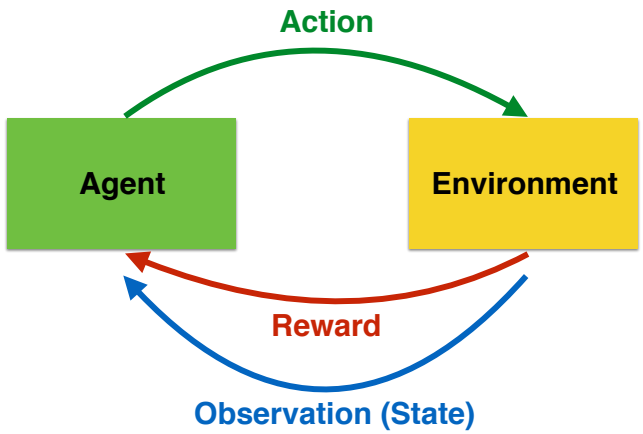
RL Applications

- Cart-Pole Problem
- Go
- Poker
- Atari
- Robot Locomotion (e.g. Walking, Sumo)
- Other games + robotics

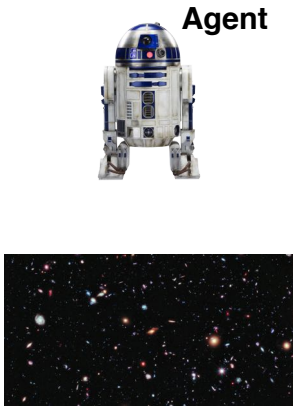
Why do we care about RL?

The world is full of interactive feedback

Reinforcement Learning

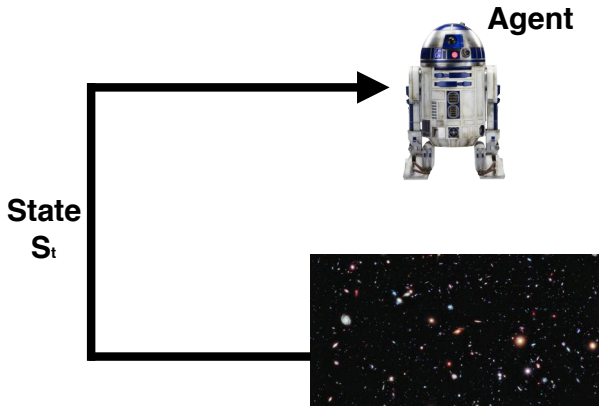


Markov Decision Process



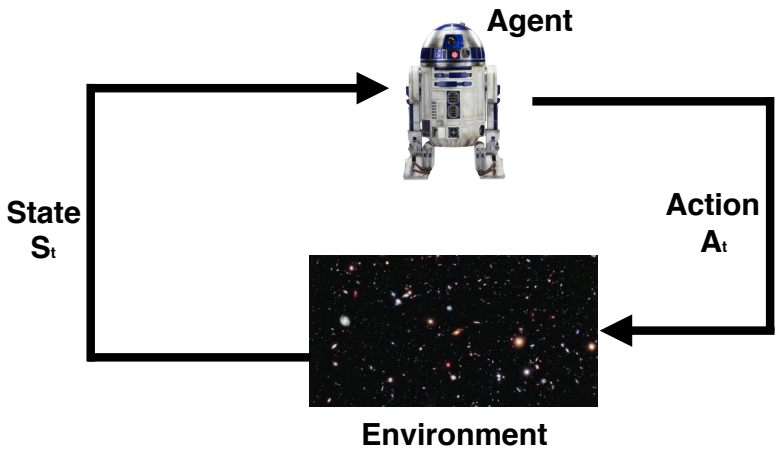
Inspired by UCL Course on RL (David Silver)

Markov Decision Process



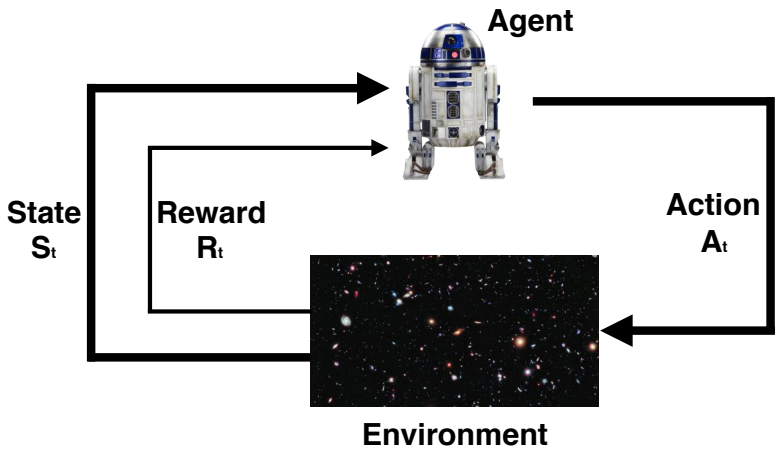
Inspired by UCL Course on RL (David Silver)

Markov Decision Process



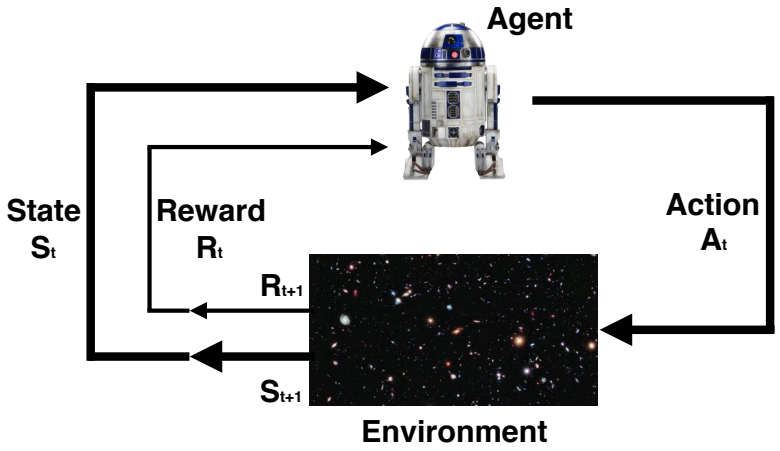
Inspired by UCL Course on RL (David Silver)

Markov Decision Process



Inspired by UCL Course on RL (David Silver)

Markov Decision Process



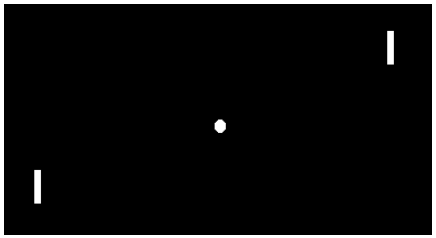
Inspired by UCL Course on RL (David Silver)

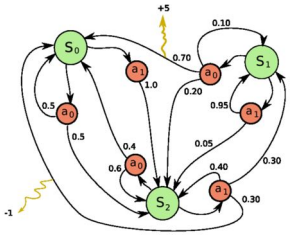
Markov Decision Process

- **Markov Property:** Current state completely characteristics the state of the world
- **Defined by:** $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$
- $\mathcal{S}$: set of possible states
  $\mathcal{A}$: set of possible actions
  $\mathcal{R}$: distribution of reward given (state, action) pair
  $\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
  γ : discount factor

Inspired by UCL Course on RL (David Silver)

Markov Decision Process





Pong Game can be played with MDP!

http://karpathy.github.io/2016/05/31/rl/

Example - Atari Games

- **Objective**: Win the game (with scores as high as possible)
- **State**: Raw image pixel of current game state
- **Action**: Game controls (e.g. U, D, L, R, Shoot)
- **Reward**: Positive/Negative score obtained at time step = t

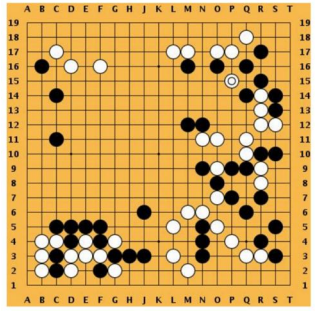
Inspired by Stanford CS231N

Example - Dota Games

- **Objective**: Win the game!
- **State**: Raw image pixel of current game state, Champion Status (Health point, magic point, gold)
- **Action**: Game controls (e.g. Mouse control, keyboard spells)
- **Reward**: Gold earned, enemies slain, special objects obtained

Inspired by Stanford CS231N

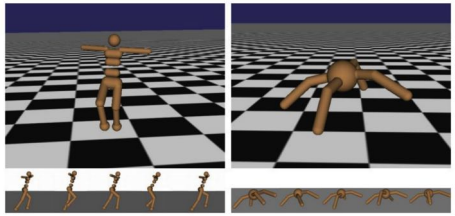
Example - Go



- **Objective**: Win the game!
- **State**: Position of all the go pieces
- **Action**: The position to put the next go
- **Reward**: 1 if eventually win, 0 if lose

Inspired by Stanford CS231N

Example - Robotics



- **Objective**: Robot learns to move!
- **State**: Position, angles of the joints
- **Action**: Torques applied on joints
- **Reward**: 1 at each time step if upright and forward movement

Inspired by Stanford CS231N

Markov Decision Process

- **Markov Property:** Current state completely characteristics the state of the world
- **Defined by:** $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$
- $\mathcal{S}$: set of possible states
  $\mathcal{A}$: set of possible actions
  $\mathcal{R}$: distribution of reward given (state, action) pair
  $\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
  γ : discount factor

Inspired by Stanford CS231N

Markov Decision Process

- At time step t = 0, the environment sample initial state

Inspired by Stanford CS231N

Markov Decision Process

- At time step t = 0, the environment sample initial state
- For t = 0 until done:
  - Agent take an action $\mathbf{a}_t$

Inspired by Stanford CS231N

Markov Decision Process

- At time step t = 0, the environment sample initial state
- For t = 0 until done:
  - Agent take an action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$

Inspired by Stanford CS231N

Markov Decision Process

- At time step t = 0, the environment sample initial state
- For t = 0 until done:
  - Agent take an action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

Inspired by Stanford CS231N

Joseph J. Lim CSCI 599 @ USC Lecture 9

Markov Decision Process

- At time step t = 0, the environment sample initial state
- For t = 0 until done:
  - Agent take an action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}
- Policy π is a function from S to A that tells what action to take in each state s

Inspired by Stanford CS231N

Joseph J. Lim CSCI 599 @ USC Lecture 9

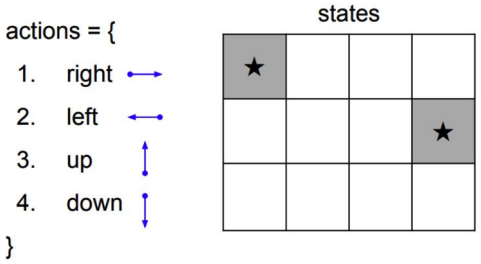
Markov Decision Process

- At time step t = 0, the environment sample initial state
- For t = 0 until done:
  - Agent take an action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}
- Policy π is a function from S to A that tells what action to take in each state s
- **Find** π^* that maximized the cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

Inspired by Stanford CS231N

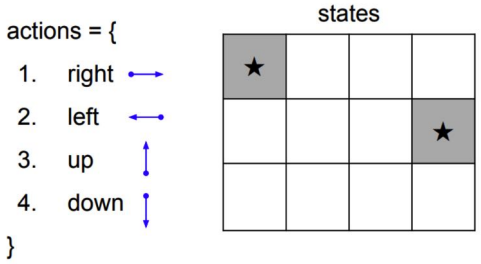
Joseph J. Lim CSCI 599 @ USC Lecture 9

A simple MDP



Inspired by Stanford CS231N

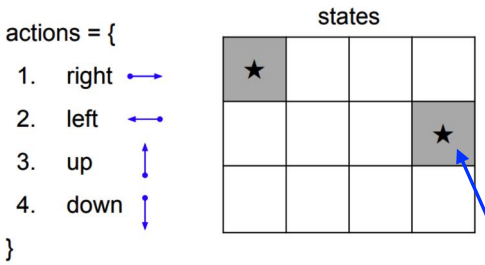
A simple MDP



Objective: Reach stars (either one) with least number of actions

Inspired by Stanford CS231N

A simple MDP



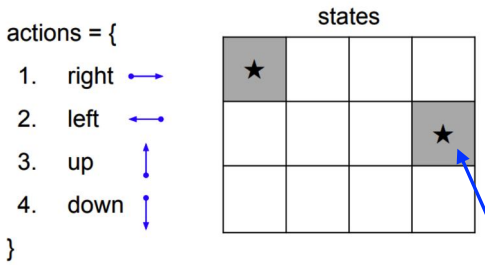
Set positive reward= 10 when reaching a star

Objective: Reach stars (either one) with least number of actions

Inspired by Stanford CS231N

A simple MDP

Set negative reward= -1 after taking each action

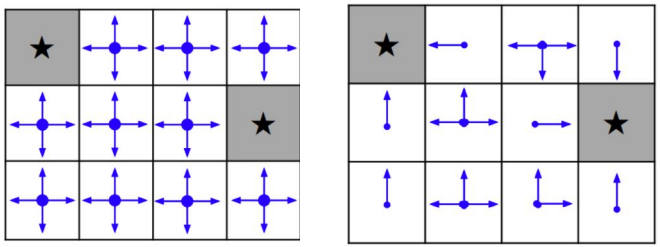


Set positive reward= 10 when reaching a star

Objective: Reach stars (either one) with least number of actions

Inspired by Stanford CS231N

A simple MDP



Random Policy

Optimal Policy

Inspired by Stanford CS231N

Optimal policy

- **Want**: Optimal policy π^* maximizing the expected value of total rewards

Inspired by Stanford CS231N

Optimal policy

- **Do:** Maximize the ***expected*** total rewards!
- **Define**: $\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$
- **With:** $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

Inspired by Stanford CS231N

Value function & Q-value function

- Sampled trajectories: s_0, a_0, r_0, s_1, a_1, r_1, …
- How do we **evaluate a state**?

Inspired by Stanford CS231N

Value function & Q-value function

- Sampled trajectories: s_0, a_0, r_0, s_1, a_1, r_1, …
- Value Function: $V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi\right]$

The expected cumulative reward if following the policy at state s

Inspired by Stanford CS231N

Value function & Q-value function

- Sampled trajectories: s_0, a_0, r_0, s_1, a_1, r_1, …
- How do we **evaluate a state-action pair**?

Inspired by Stanford CS231N

Value function & Q-value function

- Sampled trajectories: s_0, a_0, r_0, s_1, a_1, r_1, …
- Q-value Function: $Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$

The expected cumulative reward if following the policy at state s, taking the action a

Inspired by Stanford CS231N

Bellman equation

The Q-value function can be decomposed into two parts

Bellman equation

The Q-value function can be decomposed into two parts

- Immediate reward

Bellman equation

The Q-value function can be decomposed into two parts

- Immediate reward
- Discounted value of successor state

Bellman equation

The Q-value function can be decomposed into two parts

- Immediate reward
- Discounted value of successor state

s, a

Bellman equation

The Q-value function can be decomposed into two parts

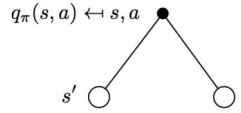
- Immediate reward
- Discounted value of successor state

$q_\pi(s, a) \mapsto s, a$

Bellman equation

The Q-value function can be decomposed into two parts

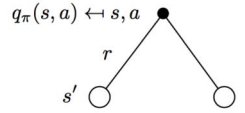
- Immediate reward
- Discounted value of successor state



Bellman equation

The Q-value function can be decomposed into two parts

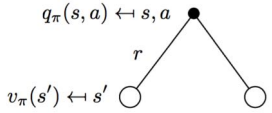
- Immediate reward
- Discounted value of successor state



Bellman equation

The Q-value function can be decomposed into two parts

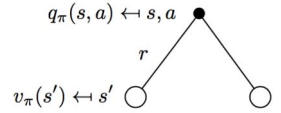
- Immediate reward
- Discounted value of successor state



Bellman equation

The Q-value function can be decomposed into two parts

- Immediate reward
- Discounted value of successor state

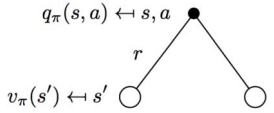


$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Bellman equation

The Q-value function can be decomposed into two parts

- Immediate reward
- Discounted value of successor state



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Note: Q-value function depends on a given policy

Bellman equation

The **optimal** Q-value function Q*

Bellman equation

The **optimal** Q-value function Q*

gives the **maximum** possible expected return

Bellman equation

The **optimal** Q-value function Q*

gives the **maximum** possible expected return

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Bellman equation

The **optimal** Q-value function Q*

gives the **maximum** possible expected return

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

It satisfies **Bellman equation**:

Bellman equation

The **optimal** Q-value function Q*

gives the **maximum** possible expected return

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

It satisfies **Bellman equation**:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

Bellman equation

The **optimal** Q-value function Q*

gives the **maximum** possible expected return

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

It satisfies **Bellman equation**:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

Optimal strategy

take the action maximizing the expected return

Bellman equation

The **optimal** Q-value function Q*

gives the **maximum** possible expected return

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

It satisfies **Bellman equation**:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

Optimal strategy → **Optimal policy**

take the action maximizing the expected return

Solving for the optimal policy

How do we find the optimal policy π^*?

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Solving for the optimal policy

How do we find the optimal policy π^*?

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Value iteration algorithm

Value Iteration Algorithm

Using Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Value Iteration Algorithm

Intuition

Update the Q-**value** function **iteratively**

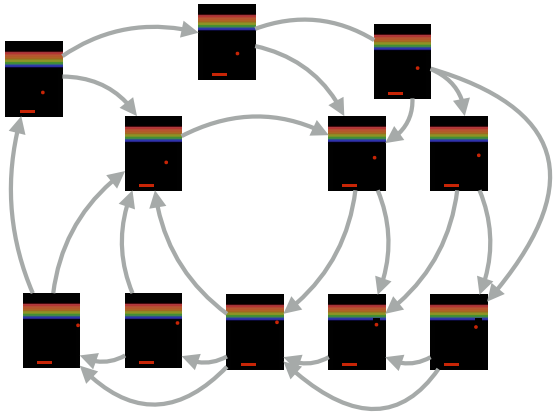
$$Q_{i+1}(s,a) = \mathbb{E} \left[r + \gamma \max_{a'} Q_i(s', a') | s, a \right]$$

Using Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

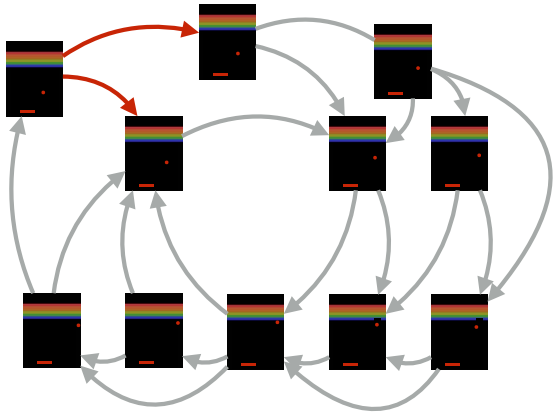
Value Iteration Algorithm

Q_0



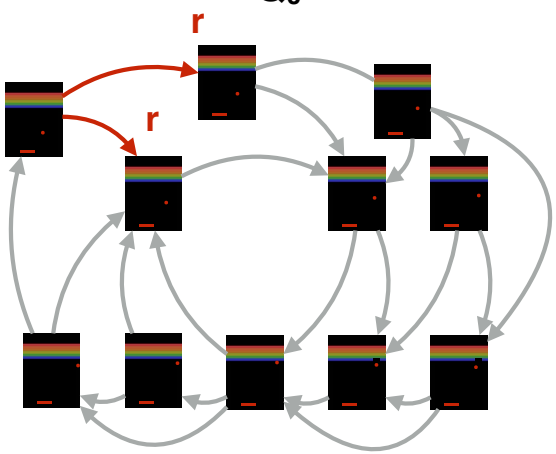
Value Iteration Algorithm

Q_0



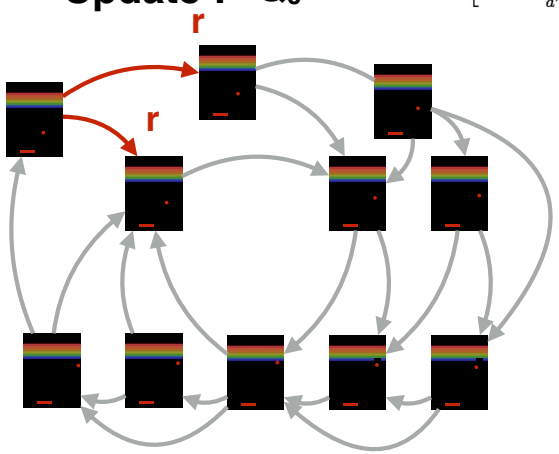
Value Iteration Algorithm

Q_0



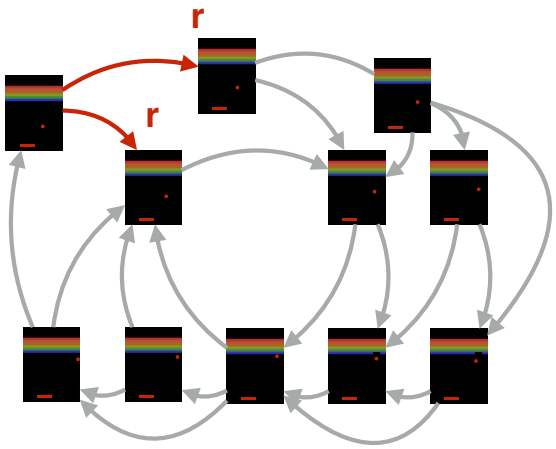
Value Iteration Algorithm

Update : Q_0 $Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$



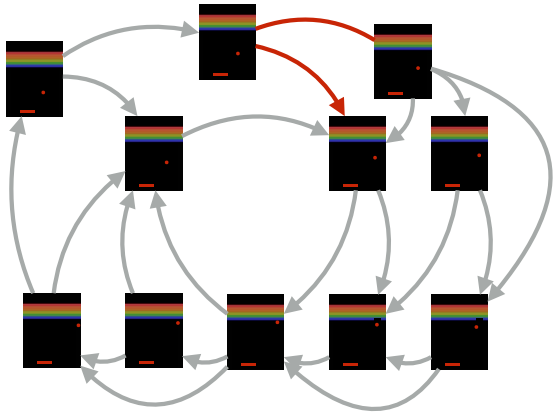
Value Iteration Algorithm

Q_1



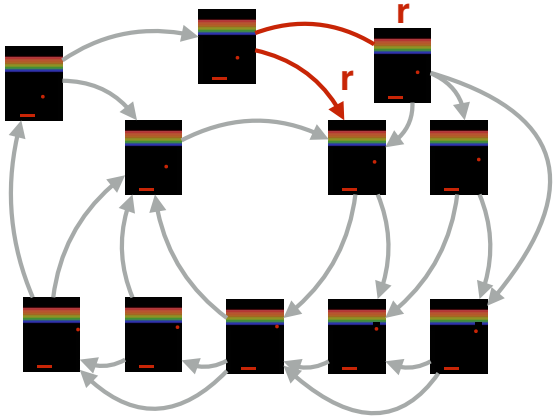
Value Iteration Algorithm

Q_1



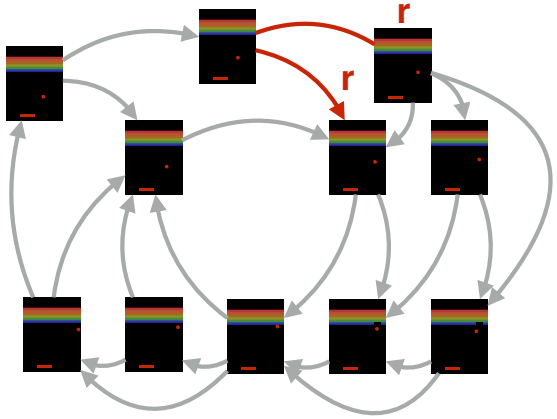
Value Iteration Algorithm

Q_1



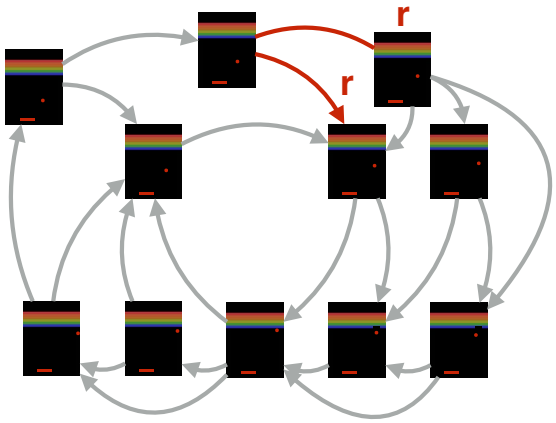
Value Iteration Algorithm

Update : Q_1 $Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$



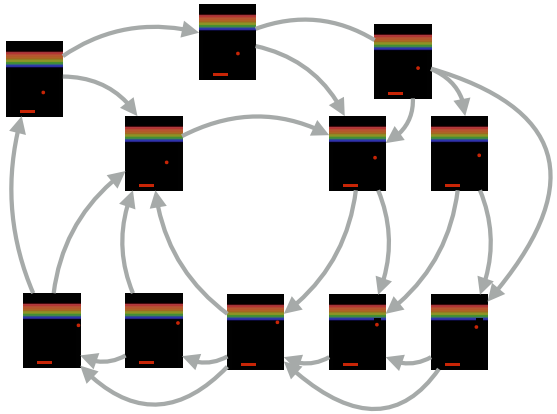
Value Iteration Algorithm

Q_2



Value Iteration Algorithm

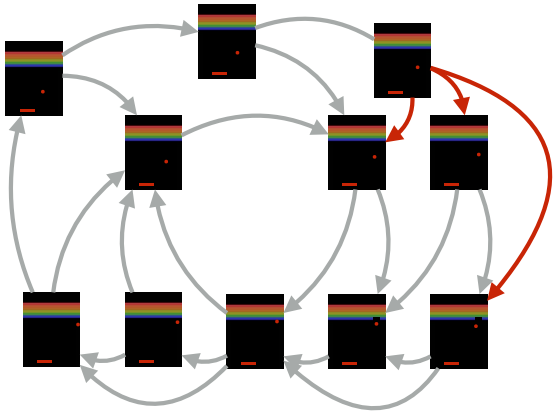
Q_2



Joseph J. Lim CSCI 599 @ USC Lecture 9

Value Iteration Algorithm

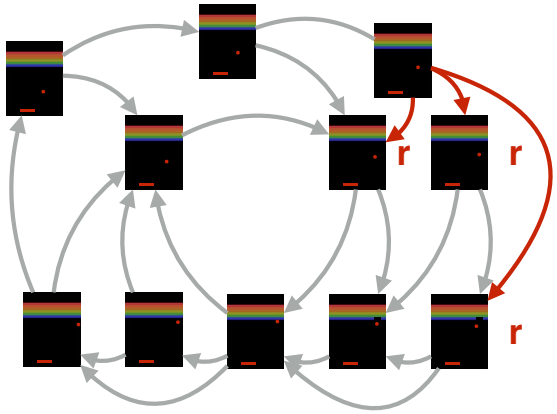
Q_2



Joseph J. Lim CSCI 599 @ USC Lecture 9

Value Iteration Algorithm

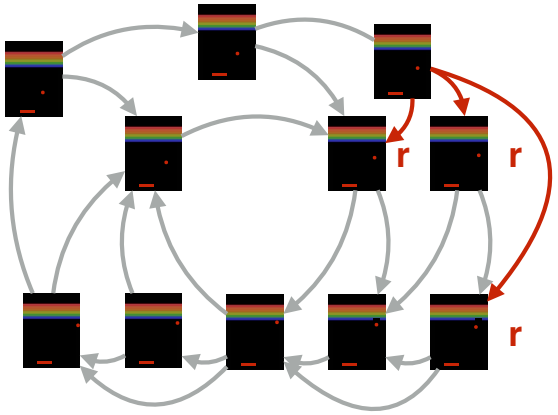
Q_2



Joseph J. Lim CSCI 599 @ USC Lecture 9

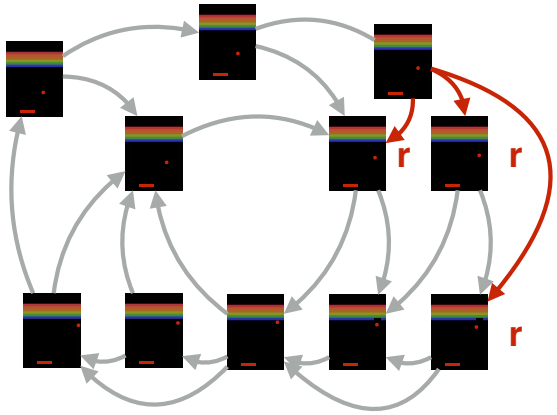
Value Iteration Algorithm

Update : Q_2 $Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$



Value Iteration Algorithm

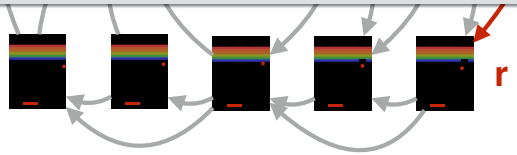
Q_3



Value Iteration Algorithm

Q_3

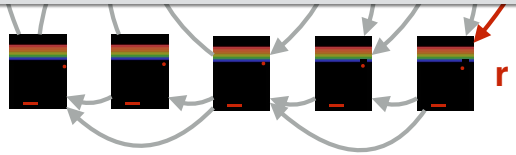
The Q-value function will converge to Q*



Value Iteration Algorithm

Q_3

But… there's a problem



Q-Learning

We need Q(s, a) for every state-action pair

Q-Learning

We need Q(s, a) for every state-action pair

Instead of memorizing Q*(s, a)

Q-Learning

We need Q(s, a) for every state-action pair

Instead of memorizing Q*(s, a)

We **approximate** the Q-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Q-Learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Q-Learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

What would be a good **function approximator**?

Q-Learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

What would be a good **function approximator**?

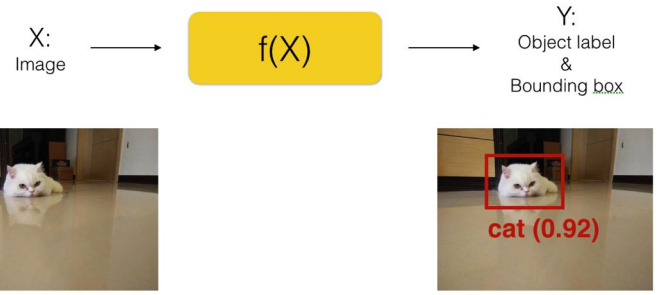
Yes! A neural network!

Q-Learning

Do you remember…?

It's matter of one function (lecture 3)

Object detection

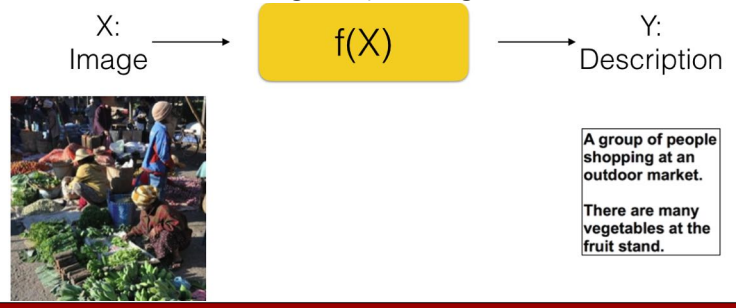


Q-Learning

Do you remember…?

It's matter of one function (lecture 3)

Image captioning



Q-Learning

Do you remember…?

It's matter of one function (lecture 3)

Text to image

X: Description → f(X) → Y: Image

This flower has a lot of small round pink petals.

Q-Learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

What would be a good **function approximator**?

Yes! A neural network!

Q-Learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

What would be a good **function approximator**?

Yes! A neural network!

X: (State, action)

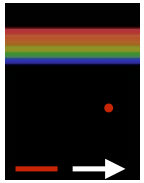
Q-Learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

What would be a good **function approximator**?

Yes! A neural network!

X: (State, action) →



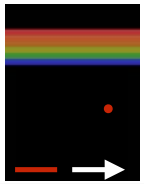
Q-Learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

What would be a good **function approximator**?

Yes! A neural network!

X:
(State, action)



f(X)

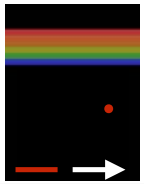
Q-Learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

What would be a good **function approximator**?

Yes! A neural network!

X:
(State, action)



f(X)

Q-Learning

$$Q(s, a; \theta) \approx Q^*(s, a)$$

What would be a good **function approximator**?

Yes! A neural network!

X:
(State, action)

f(X)

Y:
Q-value

0.87

Q-Learning

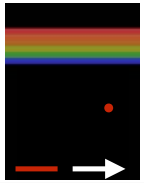
The parameters of the network

$$Q(s, a; \theta) \approx Q^*(s, a)$$

What would be a good **function approximator**?

Yes! A neural network!

X:
(State, action)



f(X)

Y:
Q-value

0.87

Joseph J. Lim CSCI 599 @ USC Lecture 9

Q-Learning

The parameters of the network

$Q(s, a; \theta) \approx Q^*(s, a)$

What would be a good **function approximator**?

Yes! A neural network!

How do we train it?

Q-Learning

Our goal: $Q(s, a; \theta) \approx Q^*(s, a)$

Q-Learning

Our goal: $Q(s, a; \theta) \approx Q^*(s, a)$

Optimal Q-value functions satisfy Bellman equation $Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$

Q-Learning

Our goal: $Q(s, a; \theta) \approx Q^*(s, a)$

Optimal Q-value functions satisfy Bellman equation $Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$

The loss function

$$L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

Q-Learning

Our goal: $Q(s, a; \theta) \approx Q^*(s, a)$

Optimal Q-value functions satisfy Bellman equation $Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$

The loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

$$\mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Q-Learning

Forward Pass

The loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

$$\mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Backward Pass

Q-Learning

Forward Pass

The loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

$$\mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Backward Pass

Update θ

Q-Learning

Forward Pass

The loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

$$\mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Backward Pass

Update θ

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

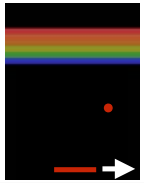
We have the optimal policy π^*

Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

We have the optimal policy π^*

X:
(State, RIGHT)

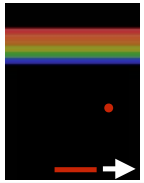


Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

We have the optimal policy π^*

X:
(State, RIGHT) →

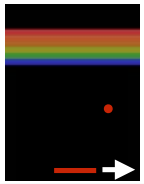


Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

We have the optimal policy π^*

X:
(State, RIGHT)



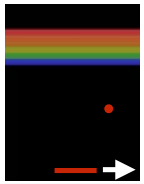
f(X)

Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

We have the optimal policy π^*

X:
(State, RIGHT)



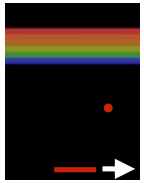
f(X)

Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

We have the optimal policy π^*

X:
(State, RIGHT)



f(X)

Y:
Q-value

0.87

Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

We have the optimal policy π^*

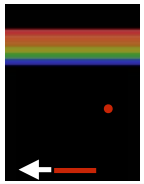
X:
(State, LEFT)

Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

We have the optimal policy π^*

X:
(State, LEFT) →

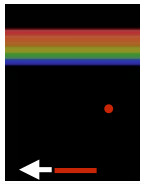


Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

We have the optimal policy π^*

X:
(State, LEFT) → f(X)

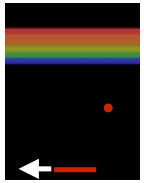


Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

We have the optimal policy π^*

X:
(State, LEFT) → f(X) →



Q-Learning

If we have $Q(s, a; \theta) = Q^*(s, a)$

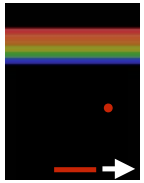
We have the optimal policy π^*

X:
(State, LEFT) → f(X) → Y:
Q-value

0.94

Q-Learning

X:
(State, RIGHT)

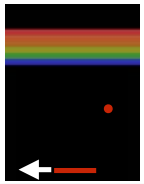


f(X)

Y:
Q-value

0.87

X:
(State, LEFT)



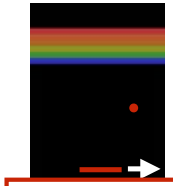
f(X)

Y:
Q-value

0.94

Q-Learning

X:
(State, RIGHT)

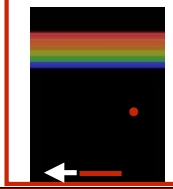


f(X)

Y:
Q-value

0.87

X:
(State, LEFT)



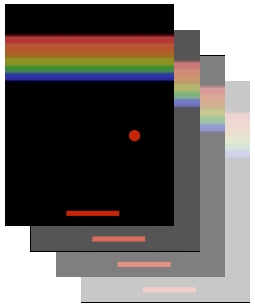
f(X)

Y:
Q-value

0.94

Case study: Atari

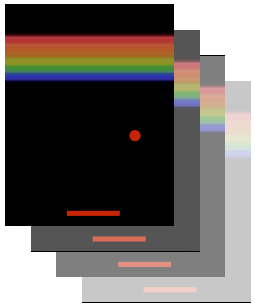
A Stack of States



Why not just a single state?

Case study: Atari

A Stack of States

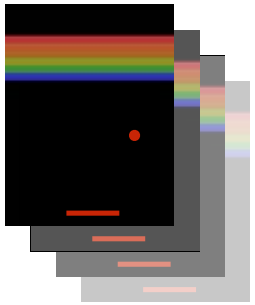


Why not just a single state?

Temporal dependency

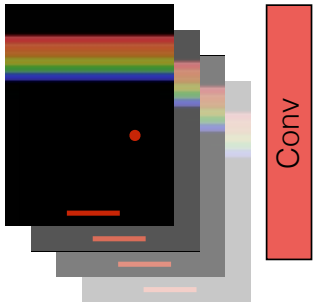
Case study: Atari

A Stack of States



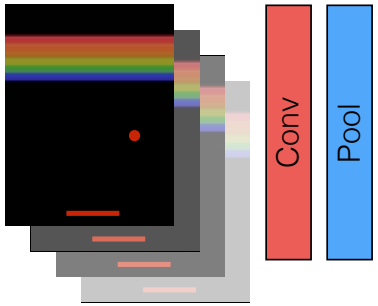
Case study: Atari

A Stack of States



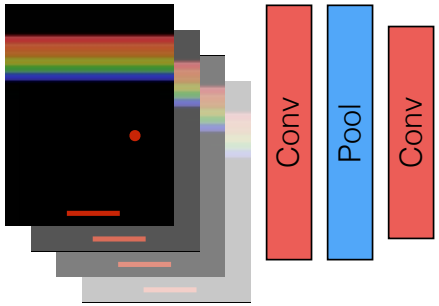
Case study: Atari

A Stack of States



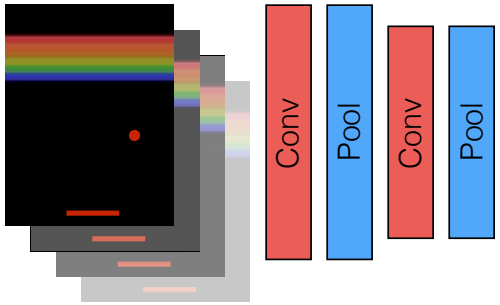
Case study: Atari

A Stack of States



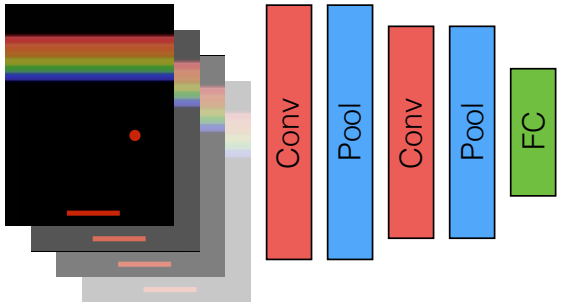
Case study: Atari

A Stack of States



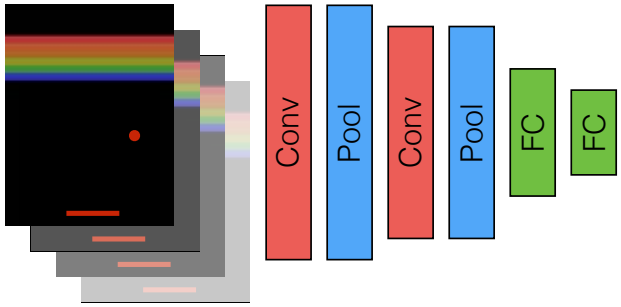
Case study: Atari

A Stack of States



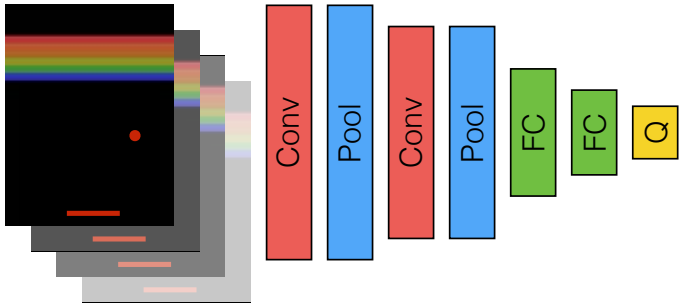
Case study: Atari

A Stack of States



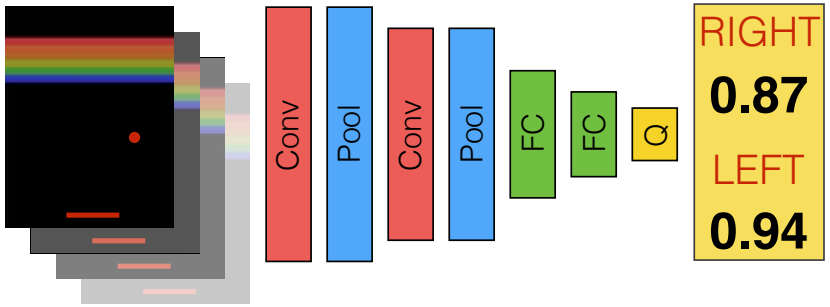
Case study: Atari

A Stack of States

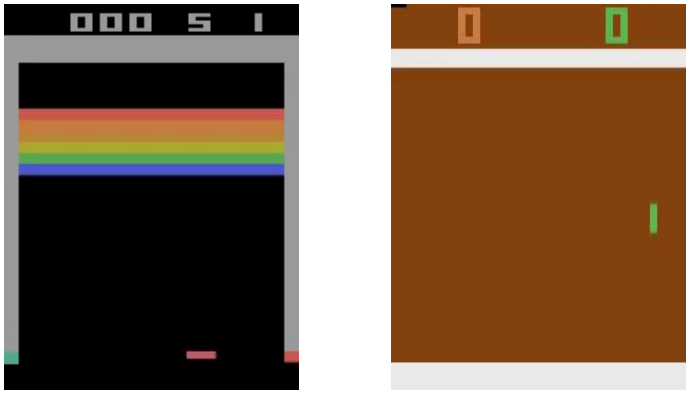


Case study: Atari

A Stack of States



Deep Q Learning

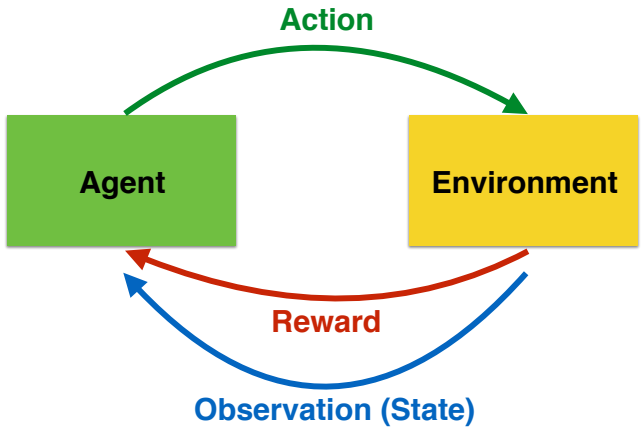


Mnih, et al., Human-level control through deep reinforcement learning. Nature 2015.

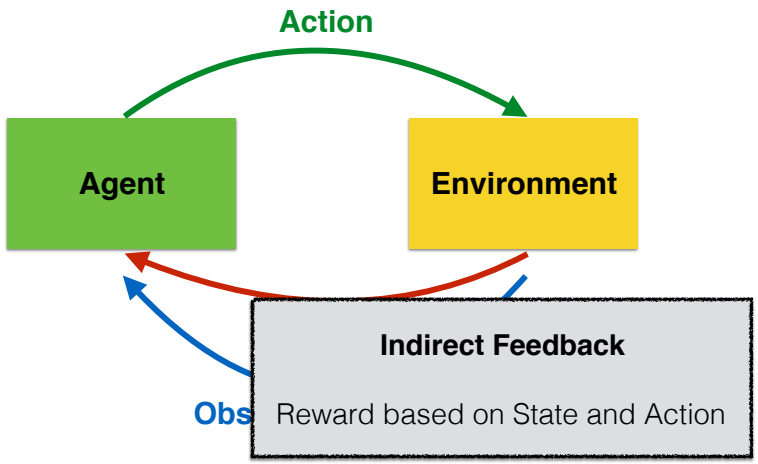
Today's agenda

- Part 1
  - Deep Reinforcement Learning
- Part 2
  - VAE in details
- Part 3
  - Research Highlight

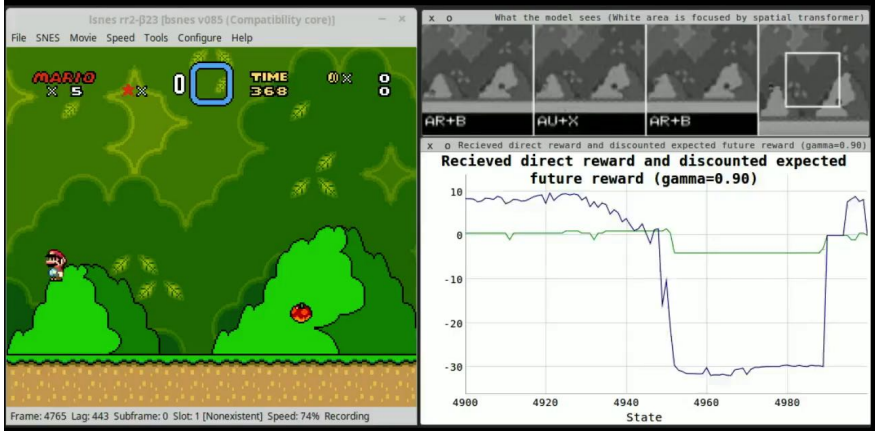
Reinforcement Learning



Reinforcement Learning



Mario AI



https://github.com/aleju/mario-ai

Locomotion

Emergence of Locomotion Behaviours in Rich Environments

Emergence of Locomotion Behaviours in Rich Environments [Heess et al]

RL Applications

- Games (e.g., go, poker, Atari, and Starcraft)
- Robotics (e.g., locomotion)
- Business operation (e.g., financial market)
- Other learning related problems

What is the goal of RL?

- Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

What is the goal of RL?

- Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

$\pi(a|s)$ tells which action is likely to take.

What is the goal of RL?

- Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

$\pi(a|s)$ tells which action is likely to take.

- **Want**: Optimal policy π^* maximizing the expected value of total rewards

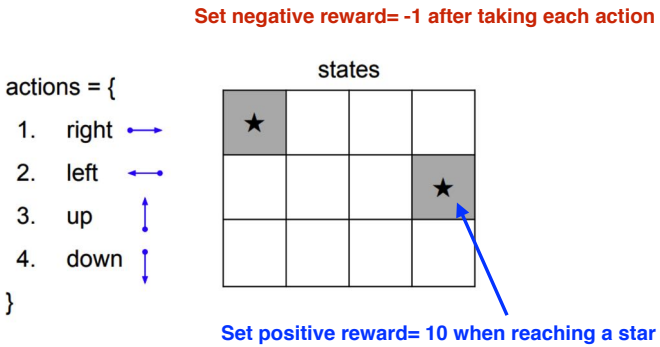
Optimal policy

- **Do:** Maximize the ***expected*** total rewards!
- **Define:** $\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$

$$s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

Inspired by Stanford CS231N

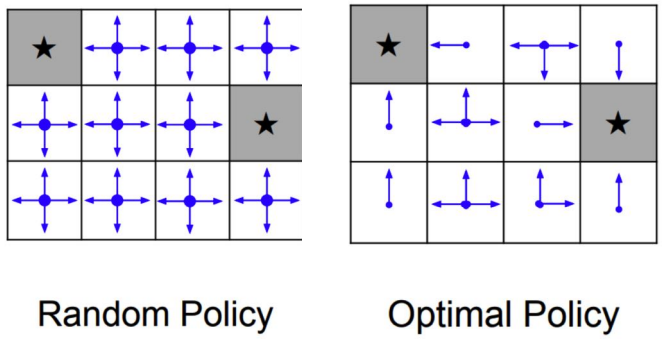
A simple MDP



Objective: Reach stars (either one) with least number of actions

Inspired by Stanford CS231N

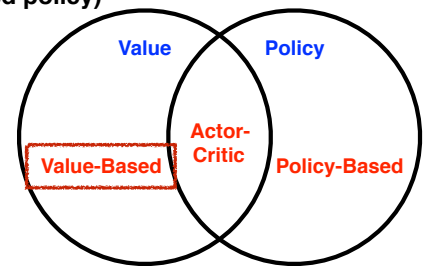
A simple MDP



Inspired by Stanford CS231N

Approaches for finding the best policy

- **Value Based (e.g. DQN)**
  - Learn Value Function
  - Implicit Policy (e.g. epsilon-greedy)
- **Policy Based (e.g. guided policy)**
  - No Value Function
  - Directly Learn Policy
- **Actor-Critic**
  - Learn Value Function
  - Learn Policy



Inspired by UCL Course on RL (David Silver)

Value-based DRL

Valued-based DRL

- Our goal is simple:

  We want a function that measures how valuable an action (a) is given a state (s).

Indirect!

State-Action Evaluation

- Sampled trajectories: s_0, a_0, r_0, s_1, a_1, r_1, …
- Q-value Function: $Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$

The best expected cumulative reward at state s, taking the action a

State-Action Evaluation

- Sampled trajectories: s_0, a_0, r_0, s_1, a_1, r_1, …
- Q-value Function: $Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$

The best expected cumulative reward at state s, taking the action a

With this Q-value function, an agent can pick the next action.

Bellman equation

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

It satisfies **Bellman equation**:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

Value Iteration Algorithm

Intuition

Update the Q-**value** function **iteratively**

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

Using Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

State-Action Evaluation

- Sampled trajectories: s_0, a_0, r_0, s_1, a_1, r_1, …
- Q-value Function: $Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$

The best expected cumulative reward at state s, taking the action a

With this Q-value function, an agent can pick the next action.

However, getting an exact optimal Q function is tricky.

Q-Learning

Instead of memorizing Q*(s, a),

our goal is to **approximate.**

$$Q(s,a;\theta) \approx Q^*(s,a)$$

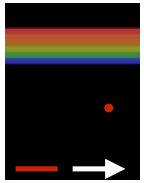
Q-Learning

$$Q(s,a;\theta) \approx Q^*(s,a)$$

What would be a good **function approximator**?

Yes! A neural network!

X: (State, action) → f(X) → Y: Q-value



0.87

Q-Learning

Our goal: $Q(s, a; \theta) \approx Q^*(s, a)$

Optimal Q-value functions satisfy Bellman equation $Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$

The loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

$$\mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Q-Learning

Forward Pass

The loss function

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$$

$$\mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$$

Backward Pass

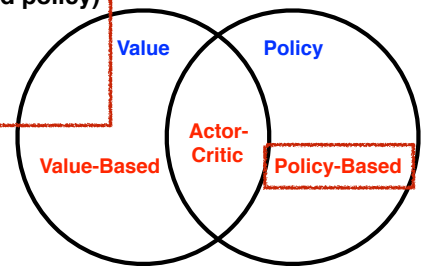
Update θ

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Policy-based DRL

Approaches for finding the best policy

- **Value Based (e.g. DQN)**
  - Learn Value Function
  - Implicit Policy (e.g. epsilon-greedy)
- **Policy Based (e.g. guided policy)**
  - No Value Function
  - Directly Learn Policy
- **Actor-Critic**
  - Learn Value Function
  - Learn Policy



Inspired by UCL Course on RL (David Silver)

Policy-based DRL

- Our goal is simple:

  We want a function that measures the probably of which action is to give the best outcome.

Direct!

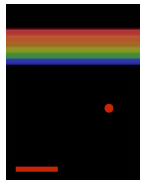
Valued-based v.s. Policy-based

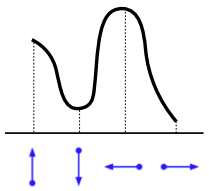
- Valued-based: a function that measures how valuable an action (a) is given a state (s). **Indirect!**
- Policy-based: a function that measures the probability of which action (a) is to give the best outcome in a given state (s). **Direct!**

Policy-based DRL

- Learn policy directly

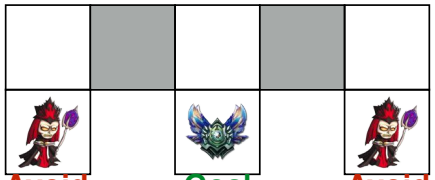
X: State → f(X) → Y: Policy $\pi(\cdot|s_t)$





Example - Aliased GridWorld


Agent



- The agent can't differentiate the grey state
- Action = {**U**p, **D**own, **L**eft, **R**ight}
- Consider features: $\phi(s,a) = 1(wall\ to\ U, a = move\ to\ R)$

Inspired by UCL Course on RL (David Silver)

Image credit: https://starforsaken101.deviantart.com/art/Karthus-the-Deathsinger-304665462
Image credit: https://www.lol-smurfs.com/blog/how-to-get-to-diamond-in-lol/

Example - Aliased GridWorld

- An optimal **deterministic** policy will either:
  - move to L in grey states, or
  - move to R in grey states
- The agent got stuck!
- **Value-Based RL learns an almost *deterministic* policy!**

Inspired by UCL Course on RL (David Silver)

Image credit: https://starforsaken101.deviantart.com/art/Karthus-the-Deathsinger-304665462
Image credit: https://www.lol-smurfs.com/blog/how-to-get-to-diamond-in-lol/

Example - Aliased GridWorld

- **Stochastic** policy is better!

$$\pi_\theta(\text{wall to U and D, move to R}) = 0.5$$
$$\pi_\theta(\text{wall to U and D, move to L}) = 0.5$$

- Higher probability to get the diamond!
- **Policy-Based RL learns optimal *stochastic* policy!**

Inspired by UCL Course on RL (David Silver)

Image credit: https://starforsaken101.deviantart.com/art/Karthus-the-Deathsinger-304665462
Image credit: https://www.lol-smurfs.com/blog/how-to-get-to-diamond-in-lol/

Policy-based DRL

- Q function is difficult to achieve
  - High dimensional or continuous action space
  - Stochastic policies

Inspired by Stanford CS231N

Policy-based DRL

- Q function is difficult to achieve
  - High dimensional or continuous action space
  - Stochastic policies

Policy-based RL can achieve

Inspired by Stanford CS231N

Value v.s. Policy

Q-Value	7	7	8

Value v.s. Policy

Q-Value	7	7	8	**Exploitable**
Policy	0.33	0.33	0.34	**Stochastic**

Policy Gradient Algorithm

- Find an ***optimal policy*** that ***maximize*** expected return

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, J(\theta)$$

$$J(\theta) = \mathbb{E}_{\pi(\theta)}\Big[\underbrace{\sum \gamma^t r_t}_{\text{Return}}\Big]$$

Policy Gradient Algorithm

- Find an ***optimal policy*** that ***maximize*** expected return

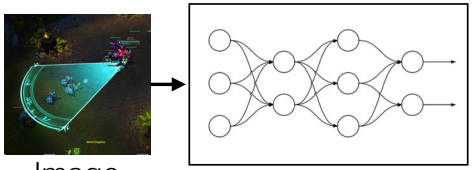
$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, J(\theta)$$

$$J(\theta) = \mathbb{E}_{\pi(\theta)}\Big[\underbrace{\sum \gamma^t r_t}_{\text{Return}}\Big]$$

By gradient ascent w.r.t $\nabla_\theta J(\theta)$

Supervised Learning v.s. Policy Gradients

Supervised Learning

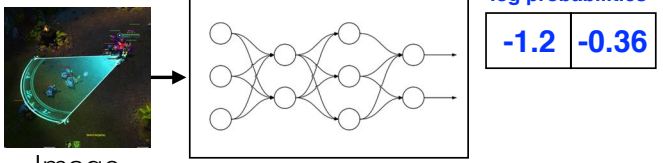


http://karpathy.github.io/2016/05/31/rl/

Supervised Learning v.s. Policy Gradients

Supervised Learning

Forward Pass

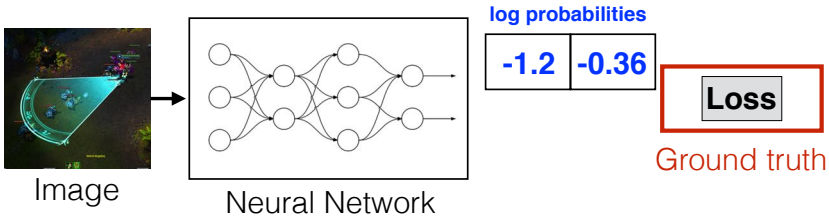


http://karpathy.github.io/2016/05/31/rl/

Supervised Learning v.s. Policy Gradients

Supervised Learning

Forward Pass

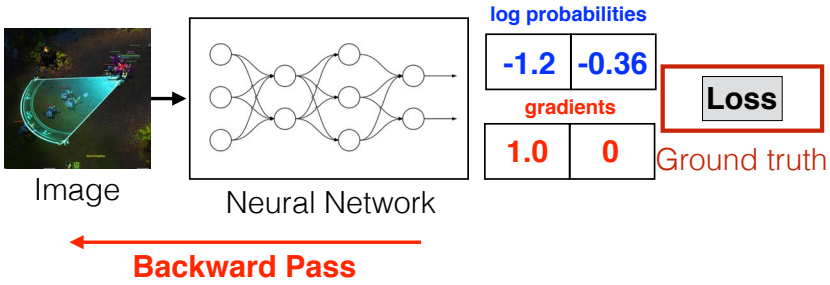


http://karpathy.github.io/2016/05/31/rl/

Supervised Learning v.s. Policy Gradients

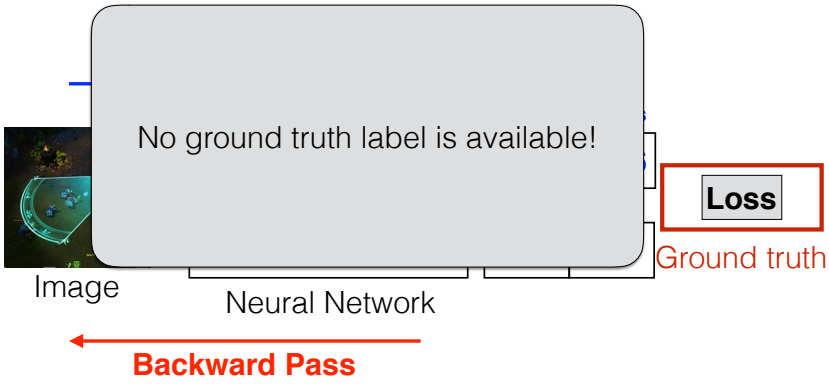
Supervised Learning

Forward Pass



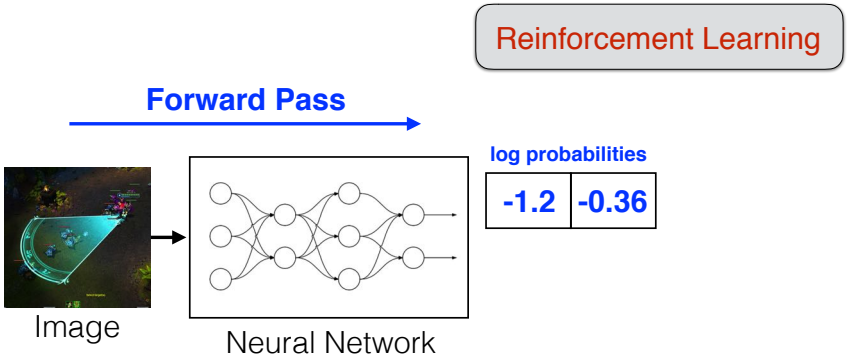
http://karpathy.github.io/2016/05/31/rl/

Supervised Learning v.s. Policy Gradients



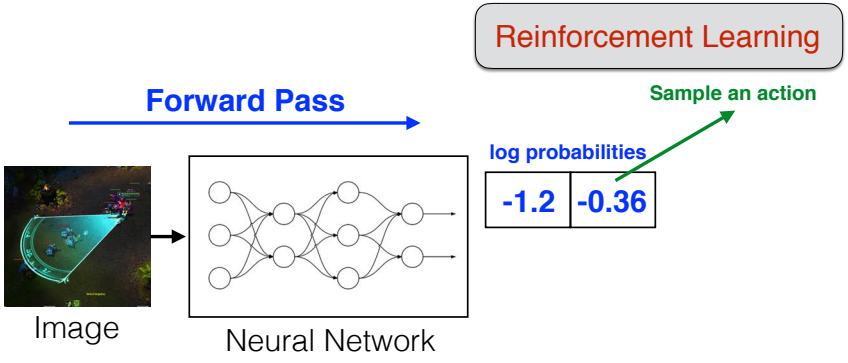
http://karpathy.github.io/2016/05/31/rl/

Supervised Learning v.s. Policy Gradients



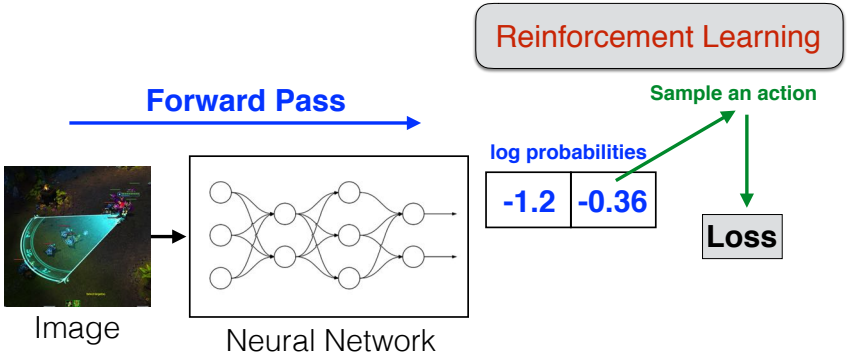
http://karpathy.github.io/2016/05/31/rl/

Supervised Learning v.s. Policy Gradients



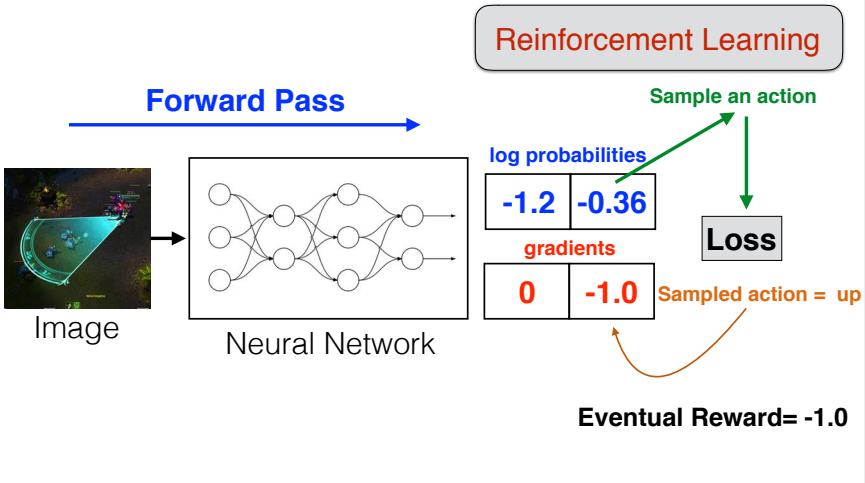
http://karpathy.github.io/2016/05/31/rl/

Supervised Learning v.s. Policy Gradients

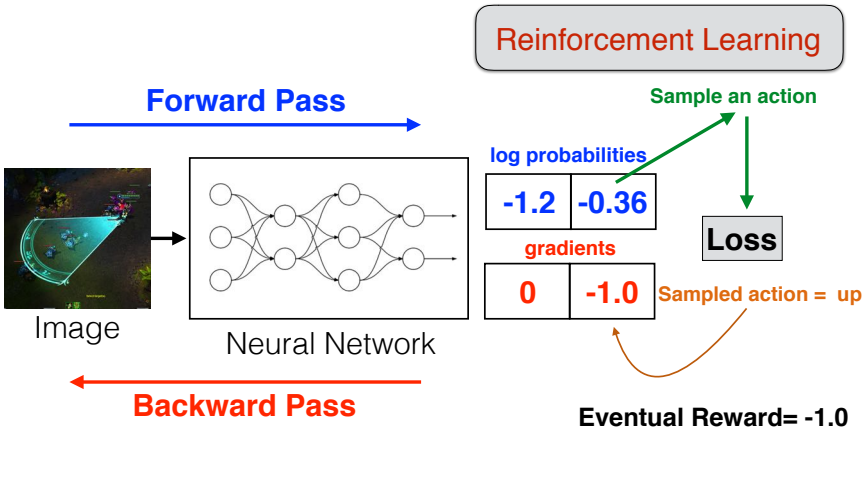


http://karpathy.github.io/2016/05/31/rl/

Supervised Learning v.s. Policy Gradients

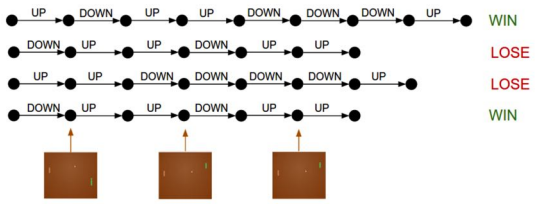


Supervised Learning v.s. Policy Gradients



Policy Gradients

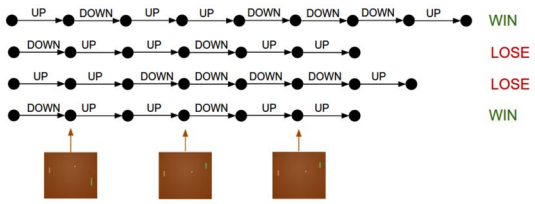
1. Run a policy for several episodes
2. See what actions lead to higher rewards, *increase their probabilities*



Policy Gradients

1. Run a policy for several episodes
2. See what actions lead to higher rewards, *increase their probabilities*

Policy gradients



How to compute policy gradients?

- Our objective (expected reward)

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\big[r(\tau)\big]$$

Inspired by UCL Course on RL (David Silver)

How to compute policy gradients?

- Our objective (expected reward)

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\big[r(\tau)\big]$$

Trajectory $\tau = (s_0, a_0, r_0, s_1, ...)$

Inspired by UCL Course on RL (David Silver)

How to compute policy gradients?

- Policy Gradient Theorem

$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\big[r(\tau)\big]$
$\tau = (s_0, a_0, r_0, s_1, ...)$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_\tau p(\tau;\theta) r(\tau) && \text{definition of expectation} \\
&= \sum_\tau r(\tau) \nabla_\theta p(\tau;\theta) \\
&= \sum_\tau r(\tau) p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} \\
&= \sum_\tau r(\tau) p(\tau;\theta) \nabla_\theta \log p(\tau;\theta) && \nabla \log(x) = \nabla x / x \\
&= \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau) \nabla_\theta \log p(\tau;\theta)] && \text{definition of expectation}
\end{aligned}$$

How to compute policy gradients?

- Gradients

$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\big[r(\tau)\big]$
$\tau = (s_0, a_0, r_0, s_1, ...)$

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau p(\tau;\theta) r(\tau) \qquad \text{definition of expectation}$$

We can compute policy gradients by sampling

$$= \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau) \nabla_\theta \log p(\tau;\theta)] \qquad \text{definition of expectation}$$

Policy Gradients

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} \boxed{r(\tau)} \boxed{\nabla_\theta \log p(\tau;\theta)}$$

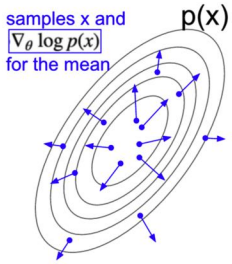
Learning step

Gradient direction (Score function)

Score Function

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} \boxed{r(\tau)} \boxed{\nabla_\theta \log p(\tau;\theta)}$$

Learning step **Score function**



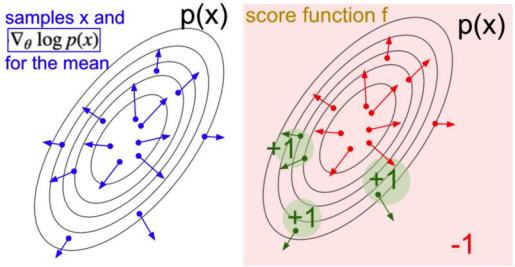
Gaussian distribution of few samples Arrow: The direction to increase the prob

http://karpathy.github.io/2016/05/31/rl/

Score Function

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} \boxed{r(\tau)} \boxed{\nabla_\theta \log p(\tau;\theta)}$$

Learning step **Score function**



Gaussian distribution of few samples Arrow: The direction to increase the prob

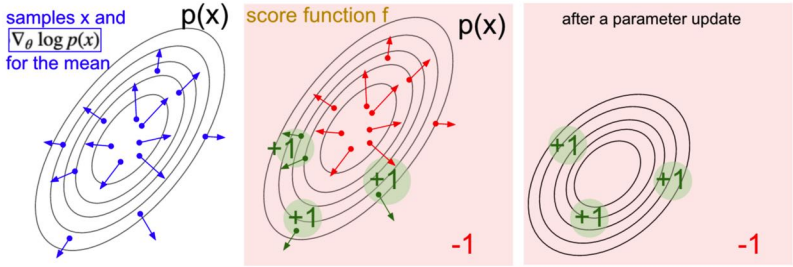
Overlay with Score Function

http://karpathy.github.io/2016/05/31/rl/

Score Function

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} \boxed{r(\tau)} \boxed{\nabla_\theta \log p(\tau;\theta)}$$

Learning step **Score function**



Gaussian distribution of few samples Arrow: The direction to increase the prob

Overlay with Score Function

After an update: samples will now have higher expected scores

http://karpathy.github.io/2016/05/31/rl/

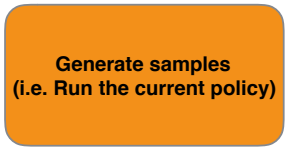
How to train the network?

- Monte-Carlo Policy Gradient (REINFORCE)

Inspired by CS294 at UCB

REINFORCE algorithm

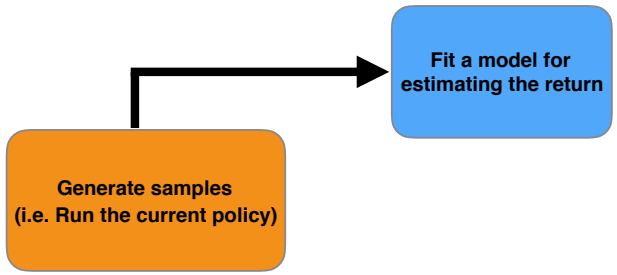
- Monte-Carlo Policy Gradient (REINFORCE)



Inspired by CS294 at UCB

REINFORCE algorithm

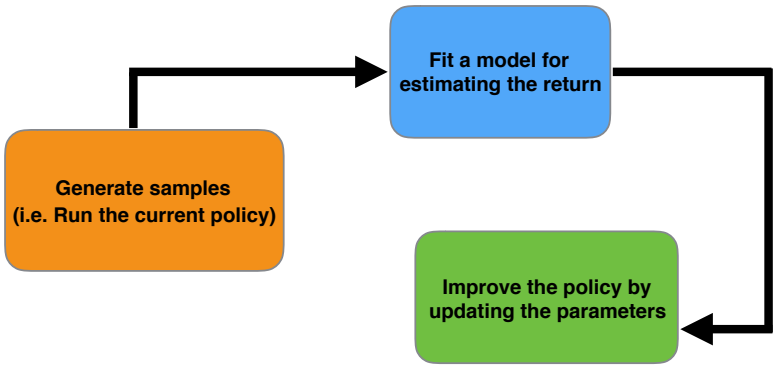
- Monte-Carlo Policy Gradient (REINFORCE)



Inspired by CS294 at UCB

REINFORCE algorithm

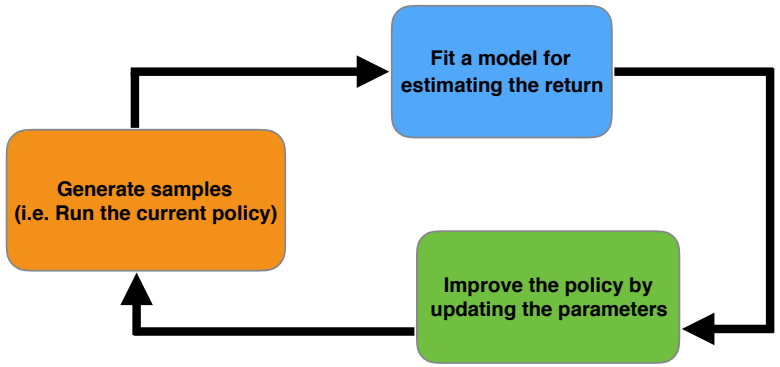
- Monte-Carlo Policy Gradient (REINFORCE)



Inspired by CS294 at UCB

REINFORCE algorithm

- Monte-Carlo Policy Gradient (REINFORCE)



Inspired by CS294 at UCB

REINFORCE algorithm

- We have: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)]$
$= \int_\tau r(\tau) p(\tau;\theta) \mathrm{d}\tau$
- Where r(τ) is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

Inspired by Stanford CS231N

REINFORCE algorithm

- We have: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau)]$
$= \int_\tau r(\tau) p(\tau;\theta) \mathrm{d}\tau$
- To compute the gradients, we differentiate

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

- But $\nabla_\theta J(\theta)$ is intractable, since p depends on θ

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

- But $\nabla_\theta J(\theta)$ is intractable, since p depends on θ
- We can use the following trick:

$$\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

- But $\nabla_\theta J(\theta)$ is intractable, since p depends on θ
- We can use the following trick:

$$\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$$

- Substitute back we get:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_\tau \left(r(\tau) \nabla_\theta \log p(\tau;\theta) \right) p(\tau;\theta) \mathrm{d}\tau \\ &= \mathbb{E}_{\tau \sim p(\tau;\theta)} \left[r(\tau) \nabla_\theta \log p(\tau;\theta) \right] \end{aligned}$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

- But $\nabla_\theta J(\theta)$ is intractable, since p depends on θ
- We can use the following trick:

$$\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$$

- Substitute back we get:

Can be estimated with Monte Carlo Sampling

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_\tau \left(r(\tau) \nabla_\theta \log p(\tau;\theta) \right) p(\tau;\theta) \mathrm{d}\tau \\ &= \boxed{\mathbb{E}_{\tau \sim p(\tau;\theta)} \left[r(\tau) \nabla_\theta \log p(\tau;\theta) \right]} \end{aligned}$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) \mathrm{d}\tau$$

- But $\nabla_\theta J(\theta)$ is intractable, since p depends on θ
- We can use the following trick:

$$\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta)$$

- Substitute back we get:

Can be estimated with Monte Carlo Sampling

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_\tau \left(r(\tau) \nabla_\theta \log p(\tau;\theta) \right) p(\tau;\theta) \mathrm{d}\tau \\ &= \boxed{\mathbb{E}_{\tau \sim p(\tau;\theta)} \left[r(\tau) \nabla_\theta \log p(\tau;\theta) \right]} \end{aligned}$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

- Without knowing the transition probabilities:

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

- Without knowing the transition probabilities:

$$p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$$

- We have:

$$\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t|s_t)$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

$$\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t|s_t)$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

$$\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t|s_t)$$

- And we differentiate:

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Inspired by Stanford CS231N

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

$$\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t|s_t)$$

- And we differentiate:

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- When sampling a trajectory $\boldsymbol{\tau}$, we have:

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Not depending on the transition probabilities!

Inspired by Stanford CS231N

REINFORCE algorithm

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G_t \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$

https://leimao.github.io/article/REINFORCE-Policy-Gradient/

Policy Gradients

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Learning step — Gradient direction

- If $r(\tau)$ is high, push up the prob. of the seen actions
- If $r(\tau)$ is low, push down the prob. of the seen actions

Policy Gradients

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Reward can have high variance

- If $r(\tau)$ is high, push up the prob. of the seen actions
- If $r(\tau)$ is low, push down the prob. of the seen actions

Variance reduction

Gradient Estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$

- Idea 1: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$
- Idea 2: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$

Inspired by Stanford CS231N

Variance reduction

Gradient Estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$

- Idea 1: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$

Push up the seen actions probabilities, by only the cumulative future reward from that state

- Idea 2: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$

Use discount factor to ignore the delayed effects

Inspired by Stanford CS231N

Variance reduction

Baseline

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Whether a reward is better/worse than expectation

Inspired by Stanford CS231N

How to choose the baseline?

- A simple baseline:
  - Constant moving average of rewards experienced so far from all trajectories

Inspired by Stanford CS231N

How to choose the baseline?

- A *better* baseline:
  - **Want**: Push up the prob. of an action from a state if such action is better than the **expected value of could get from that state**

Inspired by Stanford CS231N

How to choose the baseline?

- A *better* baseline:
  - **Want**: Push up the prob. of an action from a state if such action is better than the **expected value of could get from that state**
  - **Do:** Use **Q-function** and **value function**!

    If $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large

    If $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is small

Inspired by Stanford CS231N

How to choose the baseline?

- A *better* baseline:
  - **Want**: Push up the prob. of an action from a state if such action is better than the **expected value of could get from that state**
  - **Do:** Use **Q-function** and **value function**!

    If $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large 😊

    If $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is small 😒

Inspired by Stanford CS231N

How to choose the baseline?

- A *better* baseline:
  - Use **Q-function** and **value function**!

If $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large 😊
If $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is small 😐

The estimator:

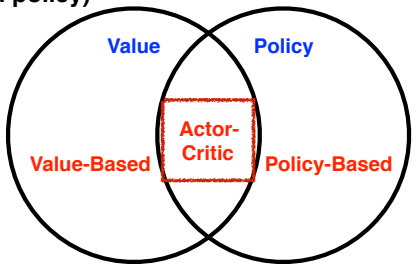
$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Inspired by Stanford CS231N

Actor-Critic

Approaches for finding the best policy

- **Value Based (e.g. DQN)**
  - Learn Value Function
  - Implicit Policy (e.g. epsilon-greedy)
- **Policy Based (e.g. guided policy)**
  - No Value Function
  - Directly Learn Policy
- **Actor-Critic**
  - Learn Value Function
  - Learn Policy



Inspired by UCL Course on RL (David Silver)

Actor-Critic Algorithm

- Episode has to finish before computing policy gradients

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Inspired by Stanford CS231N

Actor-Critic Algorithm

- Episode has to finish before computing policy gradients

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Data inefficient!

Inspired by Stanford CS231N

Actor-Critic Algorithm

- Episode has to finish before computing policy gradients

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

$Q(s_t, a_t)$ or $Q(s_t, a_t) - V(s_t)$

Using value estimation!

Inspired by Stanford CS231N

Actor-Critic Algorithm

- Can we learn Q and Value functions?
  - Q-Learning + Policy Gradient methods!
  - Train both an **Actor** (policy) and a **Critic** (Q-function)

Inspired by Stanford CS231N

Actor-Critic Algorithm

- **Actor**: Which action to take?

Inspired by Stanford CS231N

Actor-Critic Algorithm

- **Actor**: Which action to take?
- **Critic**: How good the action was and how to adjust?

Inspired by Stanford CS231N

Actor-Critic Algorithm

- **Actor**: Which action to take?
- **Critic**: How good the action was and how to adjust?
- **Advantage Function**: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$

By how much an action is better than expected?
Reduce the variance of policy gradient

Inspired by Stanford CS231N

Actor-Critic Algorithm

- **Actor**: Which action to take?
- **Critic**: How good the action was and how to adjust?
- **Advantage Function**: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$

By how much an action is better than expected?
Reduce the variance of policy gradient

- **Advantage Actor-Critic (A2C)**:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Inspired by Stanford CS231N

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ϕ
For iteration=1, 2 … **do**
  Sample m trajectories under the current policy
  $\Delta\theta \leftarrow 0$
  For i=1, …, m **do**
    For t=1, … , T **do**
      $A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$
      $\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i|s_t^i)$
  $\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$
  $\theta \leftarrow \alpha\Delta\theta$
  $\phi \leftarrow \beta\Delta\phi$
End for

Inspired by Stanford CS231N

Challenges of RL

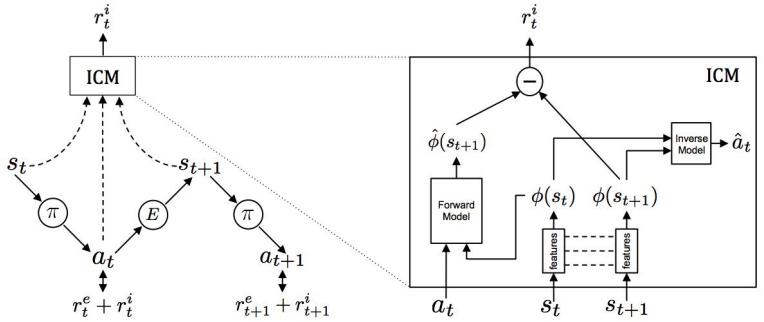
What are challenges of RL?

What are challenges of RL?

- # of iterations
- Reward sparsity
- Reward imbalance
- Hierarchical nature of tasks

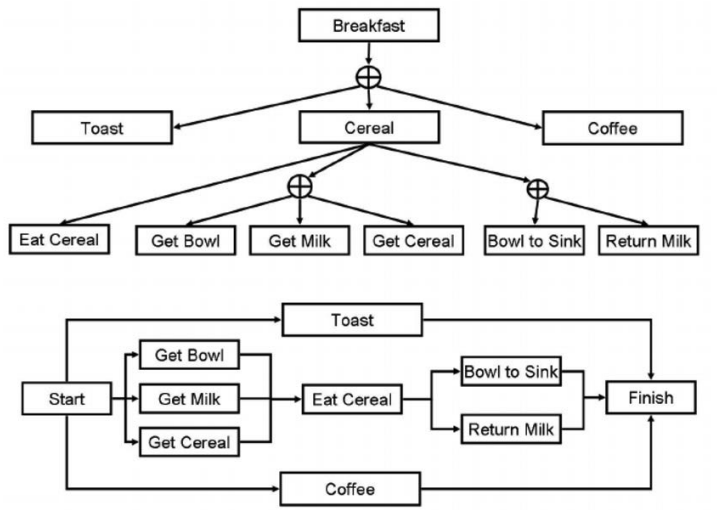
Reward sparsity

- Intrinsic motivation



Pathak, et. al. Curiosity-driven Exploration by Self-supervised Prediction. 2017.

Hierarchical nature of tasks



Hierarchical nature of tasks

- Kulkarni et al. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. NIPS 2016.
- Vezhnevets et al. FeUdal Networks for Hierarchical Reinforcement Learning. ICML 2017.
- Peng et al. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. SIGGRAPH 2017.
- Dilokthanakul et al. Feature Control as Intrinsic Motivation for Hierarchical Reinforcement Learning. arXiv 2017.
- Tessler et al. A Deep Hierarchical Approach to Lifelong Learning in Minecraft. AAAI 2017.

Summary

- RL is a method learning with an indirect feedback.
- **Q-learning:** does not always work but when it works, usually more sample-efficient. **Challenge:** exploration
- **Policy gradients:** very general but suffer from high variance so requires a lot of samples. **Challenge:** sample-efficiency.
- Guarantees:
  - Policy Gradients: Converges to a local minima, often good enough
  - Q-learning: Zero guarantees

Other reading material

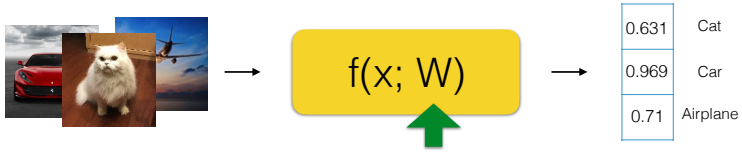
- http://joschu.net/docs/2016-bayareadlschool.pdf
- Levine & Koltun (2013). Guided policy search: deep RL with importance sampled policy gradient.
- Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size.
- Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms: deep RL with importance sampled policy gradient.

Questions?

We covered deep learning basics

- Loss functions & Optimization
- Neural Networks
- Training Neural Networks

Loss functions

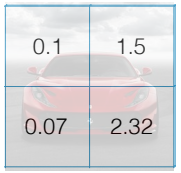


L measures how well learned W can map X to Y.

- Hinge Loss
- L_1, L_2 Loss
- Cross-Entropy

Optimization

Gradient Descent



x

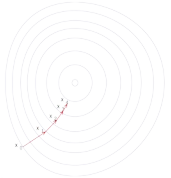
?	?	?	?
?	?	?	?
?	?	?	?

W

0
1
0

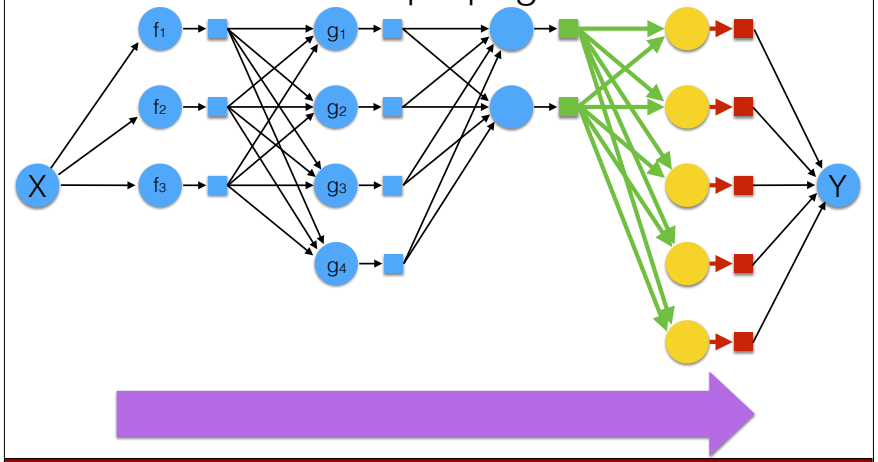
y* (ideal output)

$L(W) = || f(x; W) - y^* ||_2$



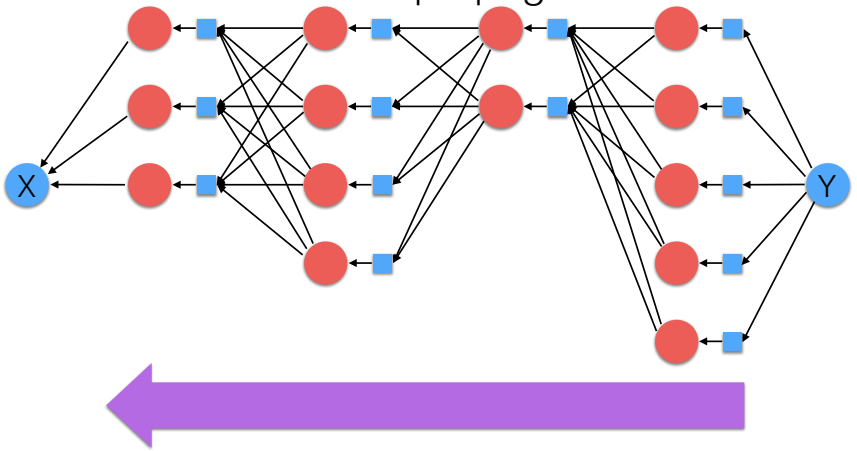
Neural Networks

Forward propagation



Neural Networks

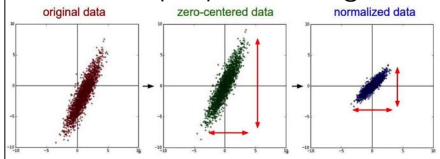
Backward propagation



Training Neural Networks

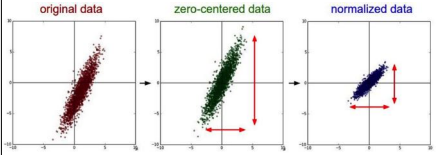
Training Neural Networks

Data preprocessing

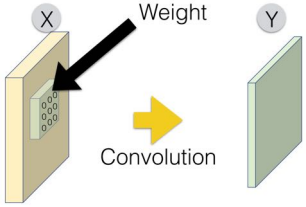


Training Neural Networks

Data preprocessing

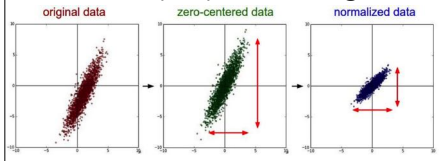


Weight Initialization



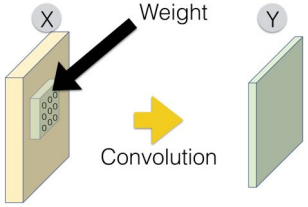
Training Neural Networks

Data preprocessing



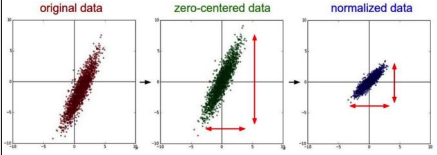
Learning Rate & Batch Size

Weight Initialization



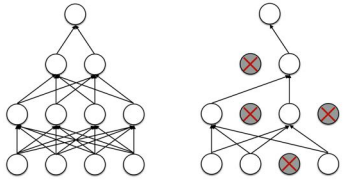
Training Neural Networks

Data preprocessing

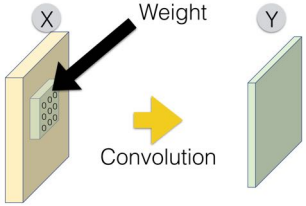


Learning Rate & Batch Size

Dropout

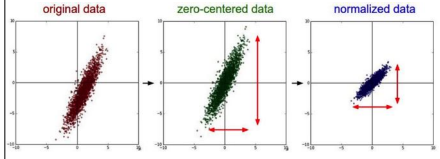


Weight Initialization



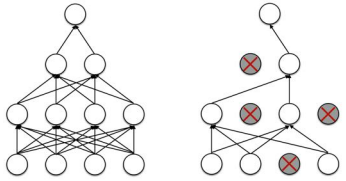
Training Neural Networks

Data preprocessing

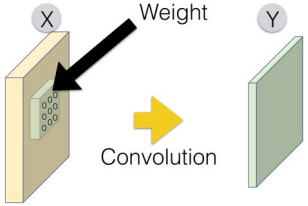


Learning Rate & Batch Size

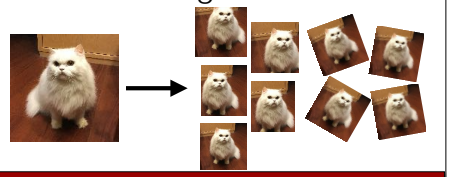
Dropout



Weight Initialization



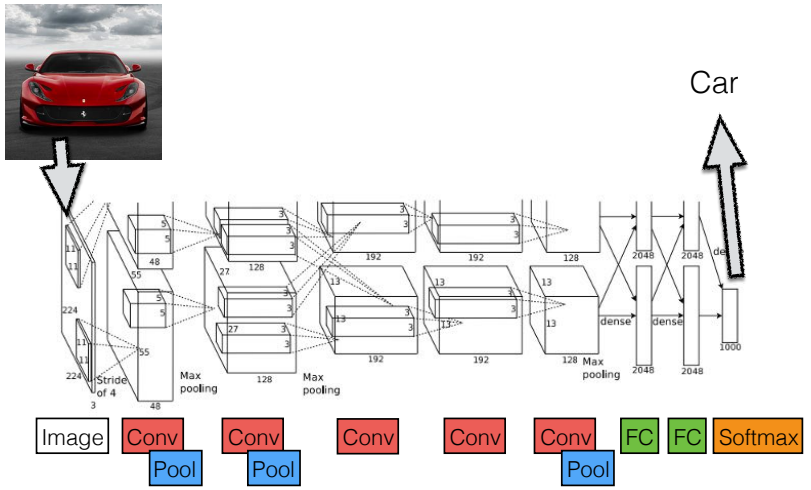
Data Augmentation



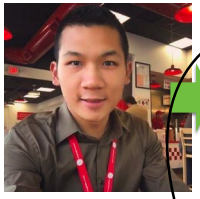
We covered three main threads

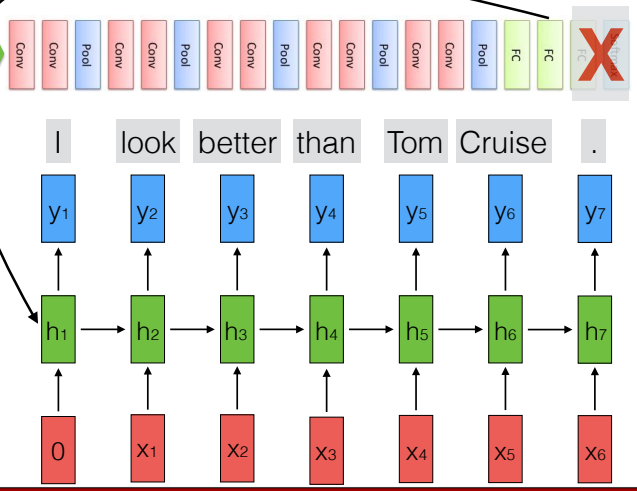
- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

CNNs

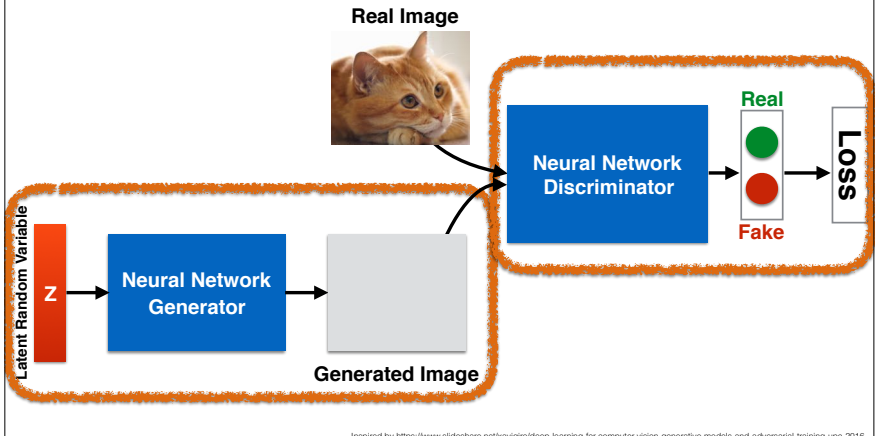


RNNs



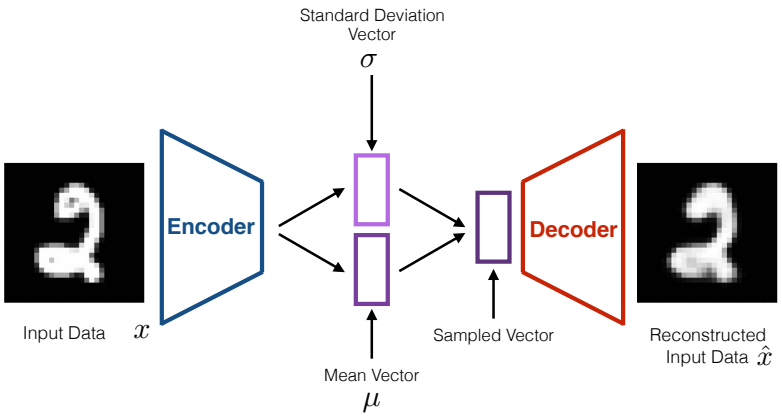


GANs

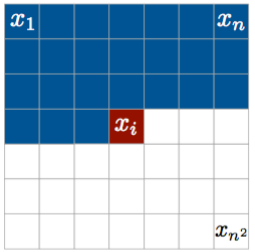


Inspired by https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

AE & VAE



Pixel RNN



Decompose the likelihood by using chain rule

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

The product of **conditional distributions** over the pixels

Deep Reinforcement Learning

Todo

- Projects! :)

Questions?