Lecture 18
GANs and AlphaGo

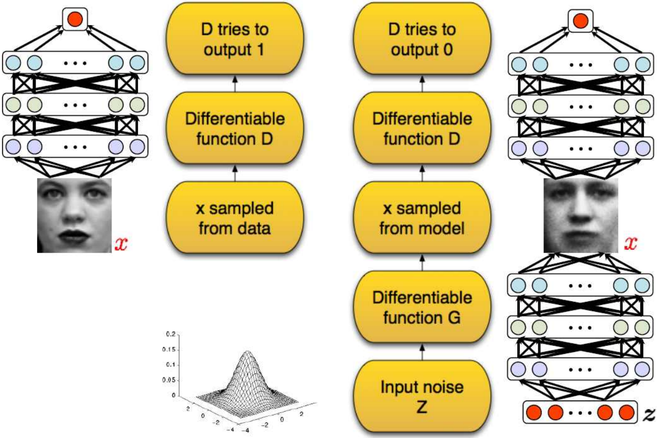
CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

May 31, 2017

Recap: Generative Adversarial Networks



Slide adapted from Ian Goodfellow

Recap: Generative Adversarial Networks

- Minimax value function

Generator: generate samples that D would classify as real

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Discriminator: Pushes up

Discriminator: Classify data as being real

Discriminator: Classify generator samples as being fake

Generator: Pushes down

- Optimal strategy for Discriminator is:

$$D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$$

Slide adapted from Ian Goodfellow

Recap: Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1-D(G(X)))]$$

Recap: Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1-D(G(X)))]$$

- For training we want to:

Recap: Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1-D(G(X)))]$$

- For training we want to:
  - Fix G, find D which **maximizes** $V(D,G)$

Recap: Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1-D(G(X)))]$$

- For training we want to:
  - Fix G, find D which **maximizes** $V(D,G)$
  - Fix D, find G which **minimizes** $V(D,G)$

Recap: Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1 - D(G(X)))]$$

- For training we want to:
  - Fix G, find D which **maximizes** $V(D,G)$
  - Fix D, find G which **minimizes** $V(D,G)$
- Alternate till convergence

Recap: Generative Adversarial Networks

- The value function:

$$V(D,G) = \mathbb{E}_{X\sim P_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim P_{\text{noise}}}[\log(1-D(G(X)))]$$

- For training we want to:
  - Fix G, find D which **maximizes** $V(D,G)$
  - Fix D, find G which **minimizes** $V(D,G)$
- Alternate till convergence
- This is good since we can use the machinery for neural networks

Divergences and Distances between distributions

1. KL

$$KL(P||Q) = \mathbb{E}_P \log \frac{P}{Q}$$

2. JS

$$JS(P||Q) = \frac{1}{2} KL(P||\frac{P+Q}{2}) + \frac{1}{2} KL(Q||\frac{P+Q}{2})$$

3. Wasserstein

$$W(P||Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||]$$

- $\Pi(P,Q)$ denotes the set of all joint distributions $\gamma(x,y)$ whose marginals are P and Q, respectively
- $\gamma(x,y)$ indicates a plan to transport "mass" from x to y, when deforming P into Q.
  The Wasserstein (or Earth-Mover) distance is then the "cost" of the **optimal** transport plan

Generative Adversarial Networks

- Let the real data distribution be P_r and the generator's distribution be P_g with $\mathbf{x} = G(\mathbf{z}), \mathbf{z} \sim P(\mathbf{z})$

Generative Adversarial Networks

- Let the real data distribution be P_r and the generator's distribution be P_g with $\mathbf{x} = G(\mathbf{z}), \mathbf{z} \sim P(\mathbf{z})$
- The optimization was:

$$\min_G \max_D V(D, G)$$

Generative Adversarial Networks

- Let the real data distribution be P_r and the generator's distribution be P_g with $\mathbf{x} = G(\mathbf{z}), \mathbf{z} \sim P(\mathbf{z})$
- The optimization was:

$$\min_G \max_D V(D, G)$$

- Discriminator:

$$-\mathbb{E}_{P_r}[\log D(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim P_g}[\log(1 - D(\mathbf{x}))]$$

Generative Adversarial Networks

- Let the real data distribution be P_r and the generator's distribution be P_g with $\mathbf{x} = G(\mathbf{z}), \mathbf{z} \sim P(\mathbf{z})$
- The optimization was:

$$\min_G \max_D V(D, G)$$

- Discriminator:

$$-\mathbb{E}_{P_r}[\log D(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim P_g}[\log(1 - D(\mathbf{x}))]$$

- Generator:

$$\mathbb{E}_{\mathbf{x}\sim P_g}[\log(1 - D(\mathbf{x}))] \text{ Method 1}$$

$$\mathbb{E}_{\mathbf{x}\sim P_g}[-D(\mathbf{x}))] \text{ Method 2}$$

Problems

- In practice $\mathbb{E}_{\mathbf{x}\sim P_g}[\log(1 - D(\mathbf{x}))]$ does not give sufficient gradient to work with, so we use $\mathbb{E}_{\mathbf{x}\sim P_g}[-D(\mathbf{x}))]$ instead

Problems

- In practice $\mathbb{E}_{\mathbf{x}\sim P_g}[\log(1 - D(\mathbf{x}))]$ does not give sufficient gradient to work with, so we use $\mathbb{E}_{\mathbf{x}\sim P_g}[-D(\mathbf{x}))]$ instead
- Sketch: For given $\mathbf{x}$, the optimal discriminator is

$$D^*(\mathbf{x}) = \frac{P_r(\mathbf{x})}{P_r(\mathbf{x}) + P_g(\mathbf{x})}$$

Problems

- In practice $\mathbb{E}_{\mathbf{x}\sim P_g}[\log(1 - D(\mathbf{x}))]$ does not give sufficient gradient to work with, so we use $\mathbb{E}_{\mathbf{x}\sim P_g}[-D(\mathbf{x}))]$ instead
- Sketch: For given $\mathbf{x}$, the optimal discriminator is

$$D^*(\mathbf{x}) = \frac{P_r(\mathbf{x})}{P_r(\mathbf{x}) + P_g(\mathbf{x})}$$

- Plugging into the generator loss:

$$\mathbb{E}_{P_r}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim P_g}[\log(1 - D(\mathbf{x}))]$$

makes the loss $2JS(P_r||P_g) - 2\log 2$

Problems

- If the supports of P_r and P_g have little overlap then $2JS(P_r||P_g) = \log 2$, and the gradient w.r.t P_g vanishes

Problems

- If the supports of P_r and P_g have little overlap then $2JS(P_r||P_g) = \log 2$, and the gradient w.r.t P_g vanishes
- The probability that the support of P_r and P_g have almost zero overlap is 1 (Arjovsky, 2017)

Problems

- If the supports of P_r and P_g have little overlap then $2JS(P_r||P_g) = \log 2$, and the gradient w.r.t P_g vanishes
- The probability that the support of P_r and P_g have almost zero overlap is 1 (Arjovsky, 2017)
- Using $\mathbb{E}_{\mathbf{x}\sim P_g}[-D(\mathbf{x}))]$ makes G collapse too many values of $\mathbf{z}$ to the same value of $\mathbf{x}$ (mode collapse)

Problems

- If the supports of P_r and P_g have little overlap then $2JS(P_r||P_g) = \log 2$, and the gradient w.r.t P_g vanishes
- The probability that the support of P_r and P_g have almost zero overlap is 1 (Arjovsky, 2017)
- Using $\mathbb{E}_{\mathbf{x} \sim P_g}[-D(\mathbf{x}))]$ makes G collapse too many values of $\mathbf{z}$ to the same value of $\mathbf{x}$ (mode collapse)
- This objective equals to optimize $KL(P_g||P_r) - 2JS(P_g||P_r)$

Problems

- If the supports of P_r and P_g have little overlap then $2JS(P_r||P_g) = \log 2$, and the gradient w.r.t P_g vanishes
- The probability that the support of P_r and P_g have almost zero overlap is 1 (Arjovsky, 2017)
- Using $\mathbb{E}_{\mathbf{x}\sim P_g}[-D(\mathbf{x}))]$ makes G collapse too many values of $\mathbf{z}$ to the same value of $\mathbf{x}$ (mode collapse)
- This objective equals to optimize $KL(P_g||P_r) - 2JS(P_g||P_r)$
- $KL(P_g||P_r)$ imposes a high cost to generating fake looking samples, but a low cost on mode dropping

Problems

- If the supports of P_r and P_g have little overlap then $2JS(P_r||P_g) = \log 2$, and the gradient w.r.t P_g vanishes
- The probability that the support of P_r and P_g have almost zero overlap is 1 (Arjovsky, 2017)
- Using $\mathbb{E}_{\mathbf{x}\sim P_g}[-D(\mathbf{x}))]$ makes G collapse too many values of $\mathbf{z}$ to the same value of $\mathbf{x}$ (mode collapse)
- This objective equals to optimize $KL(P_g||P_r) - 2JS(P_g||P_r)$
- $KL(P_g||P_r)$ imposes a high cost to generating fake looking samples, but a low cost on mode dropping
- $KL(P_r||P_g)$ imposes high cost to not covering parts of the data, and a low cost on fake looking samples

Wasserstein GAN

- When the supports of P_r and P_g have little overlap, then KL and JS give no meaningful gradient

Wasserstein GAN

- When the supports of P_r and P_g have little overlap, then KL and JS give no meaningful gradient
- The Wasserstein distance is always continuous and differentiable a.e. hence always sensible

Wasserstein GAN

- When the supports of P_r and P_g have little overlap, then KL and JS give no meaningful gradient
- The Wasserstein distance is always continuous and differentiable a.e. hence always sensible
- Problem: The inf is intractable

Wasserstein GAN

- When the supports of P_r and P_g have little overlap, then KL and JS give no meaningful gradient
- The Wasserstein distance is always continuous and differentiable a.e. hence always sensible
- Problem: The inf is intractable
- But, the Wasserstein distance has the duality form:

Wasserstein GAN

- When the supports of P_r and P_g have little overlap, then KL and JS give no meaningful gradient
- The Wasserstein distance is always continuous and differentiable a.e. hence always sensible
- Problem: The inf is intractable
- But, the Wasserstein distance has the duality form:

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x}\sim P_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim P_g}[f(\mathbf{x})]$$

$$W(P_r, P_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{\mathbf{x}\sim P_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim P_g}[f(\mathbf{x})]$$

Wasserstein GAN

- When the supports of P_r and P_g have little overlap, then KL and JS give no meaningful gradient
- The Wasserstein distance is always continuous and differentiable a.e. hence always sensible
- Problem: The inf is intractable
- But, the Wasserstein distance has the duality form:

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x}\sim P_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim P_g}[f(\mathbf{x})]$$

$$W(P_r, P_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{\mathbf{x}\sim P_r}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim P_g}[f(\mathbf{x})]$$

- Optimize over a parameterized family w of functions that are all K-Lipschitz

Vanilla GAN

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Wasserstein GAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Wasserstein GAN

- Main differences with vanilla GAN

Wasserstein GAN

- Main differences with vanilla GAN
  - The sigmoid in the last layer in D is removed

Wasserstein GAN

- Main differences with vanilla GAN
  - The sigmoid in the last layer in D is removed
  - The log in the loss for D and G is removed

Wasserstein GAN

- Main differences with vanilla GAN
  - The sigmoid in the last layer in D is removed
  - The log in the loss for D and G is removed
  - Clip the parameters of D in an interval centered at 0

Wasserstein GAN

- Main differences with vanilla GAN
  - The sigmoid in the last layer in D is removed
  - The log in the loss for D and G is removed
  - Clip the parameters of D in an interval centered at 0
  - Don't use momentum based optimization

AlphaGo

Motivation

- Most of the problem domains that we have seen so far are natural application areas for deep learning (vision, speech, language)

Motivation

- Most of the problem domains that we have seen so far are natural application areas for deep learning (vision, speech, language)
- Predictions are inherently ambiguous, need to find statistical structure

Motivation

- Most of the problem domains that we have seen so far are natural application areas for deep learning (vision, speech, language)
- Predictions are inherently ambiguous, need to find statistical structure
- Board games are a classic AI domain which relied heavily on sophisticated search techniques with a little bit of machine learning

Motivation

- Most of the problem domains that we have seen so far are natural application areas for deep learning (vision, speech, language)
- Predictions are inherently ambiguous, need to find statistical structure
- Board games are a classic AI domain which relied heavily on sophisticated search techniques with a little bit of machine learning
- Full observations, deterministic environment - why would we need uncertainty?

Overview

Some miltstones in computer game playing

- 1949: Claude Shannon proposes the idea of game tree search, explaining how games could be solved algorithmically, in principle

Overview

Some miltstones in computer game playing

- 1949: Claude Shannon proposes the idea of game tree search, explaining how games could be solved algorithmically, in principle
- 1951: Alan Turing writes a chess program that he executes by hand

Overview

Some miltstones in computer game playing

- 1949: Claude Shannon proposes the idea of game tree search, explaining how games could be solved algorithmically, in principle
- 1951: Alan Turing writes a chess program that he executes by hand
- 1956: Arthur Samuel writes a program that plays checker better than he does

Overview

Some miltstones in computer game playing

- 1949: Claude Shannon proposes the idea of game tree search, explaining how games could be solved algorithmically, in principle
- 1951: Alan Turing writes a chess program that he executes by hand
- 1956: Arthur Samuel writes a program that plays checker better than he does
- 1968: An algorithn defeats human novices at Go

Overview

Some miltstones in computer game playing

- 1949: Claude Shannon proposes the idea of game tree search, explaining how games could be solved algorithmically, in principle
- 1951: Alan Turing writes a chess program that he executes by hand
- 1956: Arthur Samuel writes a program that plays checker better than he does
- 1968: An algorithn defeats human novices at Go
- 1992: TD-Gammon plays backgammon competitively with best human players

Overview

Some miltstones in computer game playing

- 1949: Claude Shannon proposes the idea of game tree search, explaining how games could be solved algorithmically, in principle
- 1951: Alan Turing writes a chess program that he executes by hand
- 1956: Arthur Samuel writes a program that plays checker better than he does
- 1968: An algorithn defeats human novices at Go
- 1992: TD-Gammon plays backgammon competitively with best human players
- 1996: Chinook wins the US national checkers championship

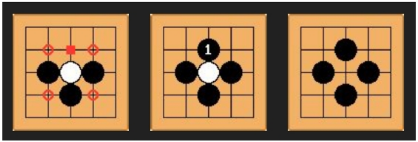
Overview

Some miltstones in computer game playing

- 1949: Claude Shannon proposes the idea of game tree search, explaining how games could be solved algorithmically, in principle
- 1951: Alan Turing writes a chess program that he executes by hand
- 1956: Arthur Samuel writes a program that plays checker better than he does
- 1968: An algorithn defeats human novices at Go
- 1992: TD-Gammon plays backgammon competitively with best human players
- 1996: Chinook wins the US national checkers championship
- 1997: DeepBlue defeats Garry Kasparov

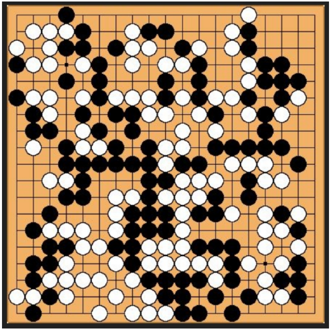
Go

- Played on a 19×19 board
- Two players, black and white, each place one stone per turn
- Capture opponent's stones by surrounding them



Go

- Goal is to surround as much territory as possible



Go

What makes Go challenging:

- Hundreds of legal moves from any position, many of which are plausible

Go

What makes Go challenging:

- Hundreds of legal moves from any position, many of which are plausible
- Games can last hundreds of moves

Go

What makes Go challenging:

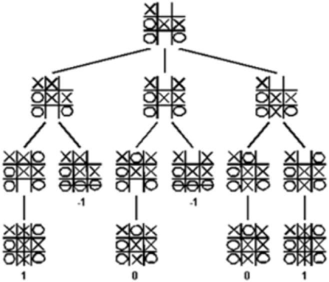
- Hundreds of legal moves from any position, many of which are plausible
- Games can last hundreds of moves
- Unlike in chess, endgames are too complicated to solve exactly

Go

What makes Go challenging:

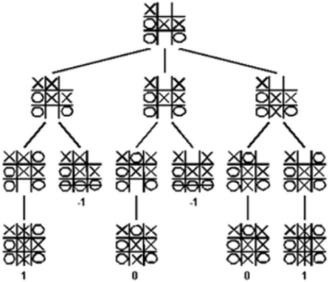
- Hundreds of legal moves from any position, many of which are plausible
- Games can last hundreds of moves
- Unlike in chess, endgames are too complicated to solve exactly
- Heavily dependent on pattern recognition

Game Trees



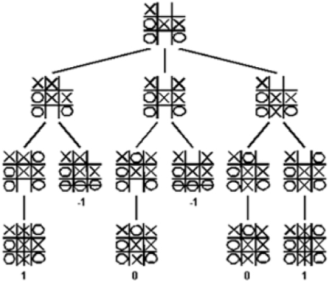
- Each node corresponds to a legal state in the game

Game Trees



- Each node corresponds to a legal state in the game
- Children of a node correspond to possible actions taken by a player

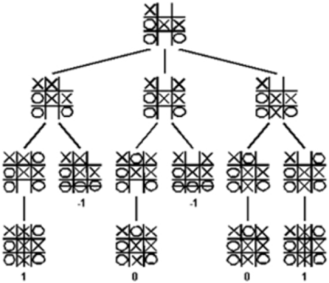
Game Trees



- Each node corresponds to a legal state in the game
- Children of a node correspond to possible actions taken by a player
- Leaf nodes are ones where we can compute the value since a win/draw condition was met

Figure: Russel and Norvig

Game Trees



- To label the internal nodes, take the max over the children is its player 1's turn, min over the children if its player 2's turn

Figure: Russel and Norvig

Game Trees

- As Shannon pointed out, for games with finite number of states, in principle you can solve them by drawing out the whole game tree.

Game Trees

- As Shannon pointed out, for games with finite number of states, in principle you can solve them by drawing out the whole game tree.
- Ways to deal with exponential blowup:
  - Search to some fixed depth, then estimate the value using an evaluation function

Game Trees

- As Shannon pointed out, for games with finite number of states, in principle you can solve them by drawing out the whole game tree.
- Ways to deal with exponential blowup:
  - Search to some fixed depth, then estimate the value using an evaluation function
  - Prioritize exploring the most promising actions for each player (according to the evaluation function)

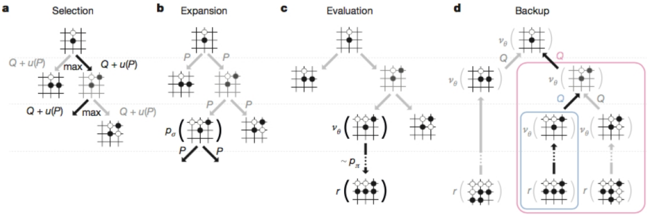
Game Trees

- As Shannon pointed out, for games with finite number of states, in principle you can solve them by drawing out the whole game tree.
- Ways to deal with exponential blowup:
  - Search to some fixed depth, then estimate the value using an evaluation function
  - Prioritize exploring the most promising actions for each player (according to the evaluation function)
- Having a good evaluation function is the key to good performance

Game Trees

- As Shannon pointed out, for games with finite number of states, in principle you can solve them by drawing out the whole game tree.
- Ways to deal with exponential blowup:
  - Search to some fixed depth, then estimate the value using an evaluation function
  - Prioritize exploring the most promising actions for each player (according to the evaluation function)
- Having a good evaluation function is the key to good performance
- Traditionally this was the main application of Machine Learning to game playing (in DeepBlue it was a learned linear function of hand desgined features)

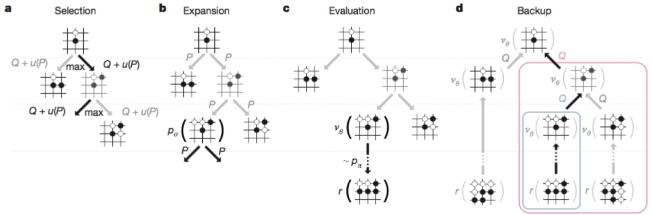
Monte Carlo Tree Search



Silver et al., 2016

- In 2006, Computer Go was revolutionized by MCTS

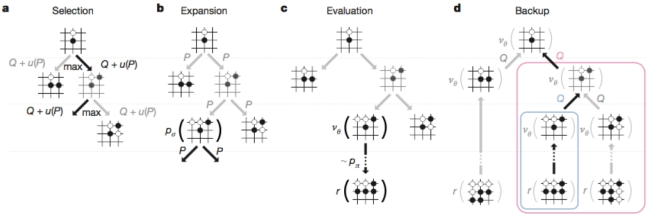
Monte Carlo Tree Search



Silver et al., 2016

- In 2006, Computer Go was revolutionized by MCTS
- Estimate the value of a position by simulating lots of rollouts (random game plays)

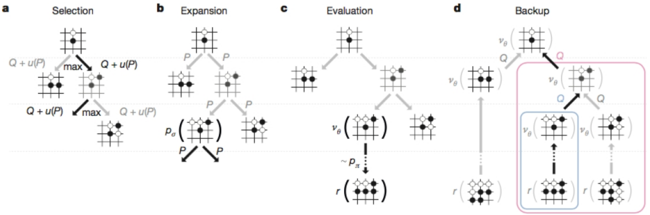
Monte Carlo Tree Search



Silver et al., 2016

- In 2006, Computer Go was revolutionized by MCTS
- Estimate the value of a position by simulating lots of rollouts (random game plays)
- Keep track of wins and losses for each node in the tree

Monte Carlo Tree Search



Silver et al., 2016

- In 2006, Computer Go was revolutionized by MCTS
- Estimate the value of a position by simulating lots of rollouts (random game plays)
- Keep track of wins and losses for each node in the tree
- How to select which parts of the tree to evaluate?

Monte Carlo Tree Search

- The selection step determines which part of the game tree to spend computational resources on simulating

Monte Carlo Tree Search

- The selection step determines which part of the game tree to spend computational resources on simulating
- Exploration-Exploitation tradeoff: Want to focus on good actions for the current player, but want to explore parts of the tree we are still uncertain about

Monte Carlo Tree Search

- The selection step determines which part of the game tree to spend computational resources on simulating
- Exploration-Exploitation tradeoff: Want to focus on good actions for the current player, but want to explore parts of the tree we are still uncertain about
- Common heuristic: Uniform confidence bound –

$$\mu_i + \sqrt{\frac{2 \log N}{N_i}}$$

Monte Carlo Tree Search

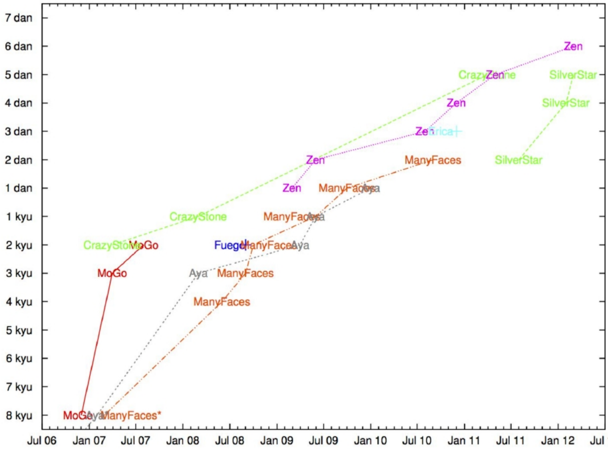
- The selection step determines which part of the game tree to spend computational resources on simulating
- Exploration-Exploitation tradeoff: Want to focus on good actions for the current player, but want to explore parts of the tree we are still uncertain about
- Common heuristic: Uniform confidence bound –

$$\mu_i + \sqrt{\frac{2 \log N}{N_i}}$$

- μ_i is the fraction of wins for action i, N_i number of times we've tried action i, N is the total number of times we have visited this node

Monte Carlo Tree Search

Improvement of computer Go since MCTS (plot is within the amateur range



AlphaGo: Predicting Expert Moves

- Can a computer play Go without any computer search?

AlphaGo: Predicting Expert Moves

- Can a computer play Go without any computer search?
- Argument: Should be possible to just use a ConvNet to identify good moves

AlphaGo: Predicting Expert Moves

- Can a computer play Go without any computer search?
- Argument: Should be possible to just use a ConvNet to identify good moves
- Input: a 19 by 19 ternary image

AlphaGo: Predicting Expert Moves

- Can a computer play Go without any computer search?
- Argument: Should be possible to just use a ConvNet to identify good moves
- Input: a 19 by 19 ternary image
- Prediction: A distribution over all legal next moves

AlphaGo: Predicting Expert Moves

- Can a computer play Go without any computer search?
- Argument: Should be possible to just use a ConvNet to identify good moves
- Input: a 19 by 19 ternary image
- Prediction: A distribution over all legal next moves
- Training data: KGS Go Server,consisting of 160,000 games and 29 million board/next-move pairs

AlphaGo: Predicting Expert Moves

- Can a computer play Go without any computer search?
- Argument: Should be possible to just use a ConvNet to identify good moves
- Input: a 19 by 19 ternary image
- Prediction: A distribution over all legal next moves
- Training data: KGS Go Server,consisting of 160,000 games and 29 million board/next-move pairs
- Architecture: 11 layer generic conv net

AlphaGo: Predicting Expert Moves

- Can a computer play Go without any computer search?
- Argument: Should be possible to just use a ConvNet to identify good moves
- Input: a 19 by 19 ternary image
- Prediction: A distribution over all legal next moves
- Training data: KGS Go Server,consisting of 160,000 games and 29 million board/next-move pairs
- Architecture: 11 layer generic conv net
- In real game play: Pick position with highest probability

AlphaGo: Predicting Expert Moves

- Can a computer play Go without any computer search?
- Argument: Should be possible to just use a ConvNet to identify good moves
- Input: a 19 by 19 ternary image
- Prediction: A distribution over all legal next moves
- Training data: KGS Go Server,consisting of 160,000 games and 29 million board/next-move pairs
- Architecture: 11 layer generic conv net
- In real game play: Pick position with highest probability
- Just a network that predicted expert moves could beat most of the previous Go programs that used search 97 % of the times

RL 101

- The basic Reinforcement Learning model consists of:
  - A set of environment and agent states S

RL 101

- The basic Reinforcement Learning model consists of:
  - A set of environment and agent states S
  - A set of actions A of the agent

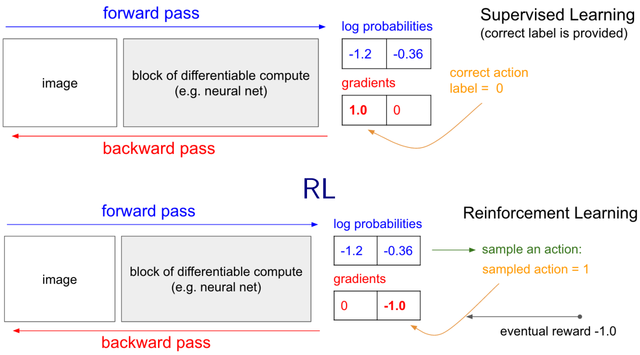
RL 101

- The basic Reinforcement Learning model consists of:
  - A set of environment and agent states S
  - A set of actions A of the agent
  - Policies of transitioning from states to actions

RL 101

- The basic Reinforcement Learning model consists of:
  - A set of environment and agent states S
  - A set of actions A of the agent
  - Policies of transitioning from states to actions
  - Rules that determine the immediate scalar reward of a transition
  - Rules that determine what the agent observes

RL 101


forward pass
log probabilities
Supervised Learning
(correct label is provided)
image
block of differentiable compute
(e.g. neural net)
-1.2
-0.36
gradients
correct action
label = 0
1.0
0
backward pass
RL
forward pass
log probabilities
Reinforcement Learning
-1.2
-0.36
sample an action:
sampled action = 1
image
block of differentiable compute
(e.g. neural net)
gradients
0
-1.0
eventual reward -1.0
backward pass

Self-Play and REINFORCE

- If θ denotes the parameters of the policy network, a_t is the action at time t and s_t is the state of the board, and z the rollout of the rest of the game using the current policy

Self-Play and REINFORCE

- If θ denotes the parameters of the policy network, a_t is the action at time t and s_t is the state of the board, and z the rollout of the rest of the game using the current policy

$$R = \mathbb{E}_{a_t \sim p_\theta(a_t|s_t)}[\mathbb{E}[r(z)|s_t, a_t]]$$

Self-Play and REINFORCE

- If θ denotes the parameters of the policy network, a_t is the action at time t and s_t is the state of the board, and z the rollout of the rest of the game using the current policy

$$R = \mathbb{E}_{a_t \sim p_\theta(a_t|s_t)}[\mathbb{E}[r(z)|s_t, a_t]]$$

- Gradient of expected reward:

Self-Play and REINFORCE

- If θ denotes the parameters of the policy network, a_t is the action at time t and s_t is the state of the board, and z the rollout of the rest of the game using the current policy

$$R = \mathbb{E}_{a_t \sim p_\theta(a_t|s_t)}[\mathbb{E}[r(z)|s_t, a_t]]$$

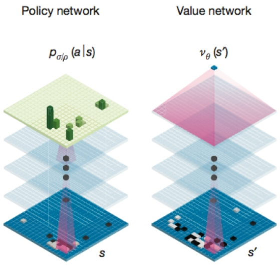
- Gradient of expected reward:

$$\begin{aligned}
\frac{\partial R}{\partial \boldsymbol{\theta}} &= \frac{\partial R}{\partial \boldsymbol{\theta}} \mathbb{E}_{a_t \sim p_{\boldsymbol{\theta}}(a_t \mid s_t)}[\mathbb{E}[r(z) \mid s_t, a_t]] \\
&= \frac{\partial}{\partial \boldsymbol{\theta}} \sum_{a_t} \sum_{z} p_\theta(a_t \mid s_t) p(z|s_t, a_t) R(z) \\
&= \sum_{a_t} \sum_{z} p(z) R(z) \frac{\partial}{\partial \boldsymbol{\theta}} p_\theta(a_t \mid s_t) \\
&= \sum_{a_t} \sum_{z} p(z \mid s_t, a_t) R(z) p_\theta(a_t \mid s_t) \frac{\partial}{\partial \boldsymbol{\theta}} \log p_\theta(a_t \mid s_t) \\
&= \mathbb{E}_{p_{\boldsymbol{\theta}}(a_t \mid s_t)} \left[\mathbb{E}_{p(z \mid s_t, a_t)} \left[R(z) \frac{\partial}{\partial \boldsymbol{\theta}} \log p_\theta(a_t \mid s_t) \right] \right]
\end{aligned}$$

Self-Play and REINFORCE

- In English: Sample action from the policy, then sample the rollout for the rest of the game. If you win, update the parameters to make the action more likely. If you lose, update them to make them less likely

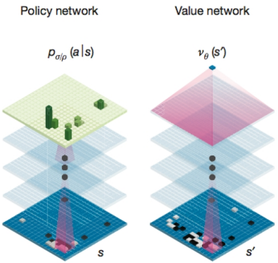
Policy and Value Network



Silver et al., 2016

- We have seen the policy and expert move networks, but AlphaGo has another network called the value network

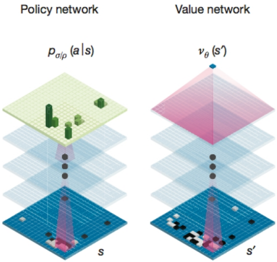
Policy and Value Network



Silver et al., 2016

- We have seen the policy and expert move networks, but AlphaGo has another network called the value network
- This network tries to predict, for a given position, which player has the advantage

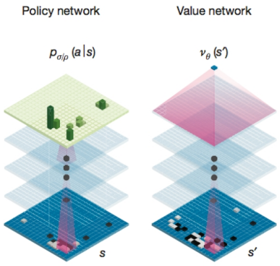
Policy and Value Network



Silver et al., 2016

- We have seen the policy and expert move networks, but AlphaGo has another network called the value network
- This network tries to predict, for a given position, which player has the advantage
- This is again, a conv net with a generic architecture trained with least squares regression

Policy and Value Network



Silver et al., 2016

- We have seen the policy and expert move networks, but AlphaGo has another network called the value network
- This network tries to predict, for a given position, which player has the advantage
- This is again, a conv net with a generic architecture trained with least squares regression
- Data comes from board positions and outcomes from self-play

Policy and Value Network

- AlphaGo combined the policy and value networks with Monte Carlo Tree Search

Policy and Value Network

- AlphaGo combined the policy and value networks with Monte Carlo Tree Search
- Policy network used to simulate rollouts

Policy and Value Network

- AlphaGo combined the policy and value networks with Monte Carlo Tree Search
- Policy network used to simulate rollouts
- Value networks to evaluate leaf positions

AlphaGo

- Most of the Go world expected AlphaGo to lose 5-0 to Lee Sedol

AlphaGo

- Most of the Go world expected AlphaGo to lose 5-0 to Lee Sedol
- It won 4-1, some of the moves seemed to be bizarre to experts, but turned out to be really good

AlphaGo

- Most of the Go world expected AlphaGo to lose 5-0 to Lee Sedol
- It won 4-1, some of the moves seemed to be bizarre to experts, but turned out to be really good
- Its one loss occurred when Lee Sedol played a key move unlike anything in the training data

AlphaGo

- Most of the Go world expected AlphaGo to lose 5-0 to Lee Sedol
- It won 4-1, some of the moves seemed to be bizarre to experts, but turned out to be really good
- Its one loss occurred when Lee Sedol played a key move unlike anything in the training data
- Last week AlphaGo defeated Ke Jie, arguably the best human Go player 3-0, and retired from competitive Go

End