INF5860 - Maskinlæring for bildeanalyse

Reinforcement Learning
<table>
<thead>
<tr>
<th>Themes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual attention</td>
</tr>
<tr>
<td>Reinforcement learning</td>
</tr>
<tr>
<td>Policy learning</td>
</tr>
<tr>
<td>Q-Learning</td>
</tr>
<tr>
<td>Actor-Critic model</td>
</tr>
</tbody>
</table>
Material

Must learn

cs231n Lecture 13: https://youtu.be/UFnO-ADC-k0?t=38m
(Attention models from 38 mins out)

Karpathy blog: http://karpathy.github.io/2016/05/31/rl/
(Reinforcement learning/Policy learning)

Value-learning lecture: https://youtu.be/KHZVXao4gXs

Policy-learning lecture (also actor-critic):
https://youtu.be/UoPei5o4fps

Blog series on Reinforcement learning:
https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df (Q-learning, Policy learning, Actor-Critic model)
Remember image captioning from last time

- First step in the recurrent network we input the feature vector from a convolutional network
- Iterate to generate text
- Inputting the image in every timestep actually lowers accuracy.

https://github.com/tensorflow/models/tree/master/im2txt
Remember image captioning from last time

- First step in the recurrent network we input the feature vector from a convolutional network
- Iterate to generate text
- Inputting the image in every timestep actually lowers accuracy.

\[
\text{before: } h = \tanh(W_{xh} \ast x + W_{hh} \ast h)
\]

\[
\text{now: } h = \tanh(W_{xh} \ast x + W_{hh} \ast h + W_{ih} \ast v)
\]
Remember image captioning from last time
Integrating image attention with word generation

- Taking the features at an earlier level, where we have spatial information
- Output weighting of input features
Integrating image attention with word generation

- Input to next step: the weighted input features and the first word

Integrating image attention with word generation

- Input to next step: the weighted input features and the first word
- Output new weighing of input features and second word
Integrating image attention with word generation

- Input to next step: the weighted input features and the first word
- Output new weighing of input features and second word
- Iterate till you end of caption

What does the attention look like?

- White indicate the weighted part of the image
- Bottom row use “hard attention”
Attention can give interpretable results

A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background.

A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.
Hard attention

Soft attention:
- Weighting the input with floating numbers

Hard attention:
- Pick one region and ignore the rest. Weighting with 1’s and 0’s
Remember spatial transformer networks

- Can work out of the box, but you can face problems with optimization
- Regions outside of the cropped box is not affection the gradients

\[
\begin{pmatrix}
x_i^s \\
y_i^s
\end{pmatrix} = \begin{bmatrix}
\theta_{11} & \theta_{12} & \theta_{13} \\
\theta_{21} & \theta_{22} & \theta_{23}
\end{bmatrix} \begin{pmatrix}
x_i^t \\
y_i^t \\
1
\end{pmatrix}
\]
Why would you want to use hard attention?

- For “show, attend and tell” it worked best
- Can save memory and computation
- You may be able to utilize high resolution

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>BLEU-1</th>
<th>BLEU-2</th>
<th>BLEU-3</th>
<th>BLEU-4</th>
<th>METEOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr8k</td>
<td>Google NIC (Vinyals et al., 2014)</td>
<td>63</td>
<td>41</td>
<td>27</td>
<td>17.7</td>
<td>17.31</td>
</tr>
<tr>
<td></td>
<td>Log Bilinear (Kiros et al., 2014a)</td>
<td>65.6</td>
<td>42.4</td>
<td>27.7</td>
<td>17.7</td>
<td>18.93</td>
</tr>
<tr>
<td></td>
<td>Soft-Attention</td>
<td>67</td>
<td>44.8</td>
<td>29.9</td>
<td>19.5</td>
<td>18.49</td>
</tr>
<tr>
<td></td>
<td>Hard-Attention</td>
<td>67</td>
<td>45.7</td>
<td>31.4</td>
<td>21.3</td>
<td>20.30</td>
</tr>
<tr>
<td>Flickr30k</td>
<td>Google NIC (Fang et al., 2014)</td>
<td>66.3</td>
<td>42.3</td>
<td>27.7</td>
<td>18.3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Log Bilinear</td>
<td>60.0</td>
<td>38</td>
<td>25.4</td>
<td>17.1</td>
<td>16.88</td>
</tr>
<tr>
<td></td>
<td>Soft-Attention</td>
<td>66.7</td>
<td>43.4</td>
<td>28.8</td>
<td>19.1</td>
<td>18.49</td>
</tr>
<tr>
<td></td>
<td>Hard-Attention</td>
<td>66.9</td>
<td>43.9</td>
<td>29.6</td>
<td>19.9</td>
<td>18.46</td>
</tr>
<tr>
<td>COCO</td>
<td>CMU/MS Research (Chen &amp; Zitnick, 2014)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>20.41</td>
</tr>
<tr>
<td></td>
<td>MS Research (Fang et al., 2014)</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>20.71</td>
</tr>
<tr>
<td></td>
<td>BRNN (Karpathy &amp; Li, 2014)</td>
<td>64.2</td>
<td>45.1</td>
<td>30.4</td>
<td>20.3</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Google NIC (Fang et al., 2014)</td>
<td>66.6</td>
<td>46.1</td>
<td>32.9</td>
<td>24.6</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Log Bilinear</td>
<td>70.8</td>
<td>48.9</td>
<td>34.4</td>
<td>24.3</td>
<td>20.03</td>
</tr>
<tr>
<td></td>
<td>Soft-Attention</td>
<td>70.7</td>
<td>49.2</td>
<td>34.4</td>
<td>24.3</td>
<td>23.90</td>
</tr>
<tr>
<td></td>
<td>Hard-Attention</td>
<td>71.8</td>
<td>50.4</td>
<td>35.7</td>
<td>25.0</td>
<td>23.04</td>
</tr>
</tbody>
</table>
Distinguishing images with small differences

- You have high resolution images
- You separate classes based on small, but characteristic differences
  - Birds
  - Tumors
  - Brands
Distinguishing images with small differences

- Pre-trained CNN features
- Attention network output confidence map
- Spatial softmax for finding probabilities of locations
- Crop and resize image features

Fully Convolutional Attention Networks for Fine-Grained Recognition
How can we train nondifferentiable functions?

Reinforcement learning:

- Use a distribution over actions
- Sample over actions
- Learn over time, across batches
- Punishment and rewards
Reinforcement learning
Reinforcement learning

Policy: A set of actions for a given set of states

Value: The reward at a certain state
Reinforcement learning

In reinforcement learning there are many different targets for your learning algorithms.

Policy learning: Learning the probability that an action is good.

Q learning: Learning the expected future reward for a given action.

Actor-Critic: Learn both a direct policy (action) and the expected future reward.
Policy learning

- Take some state as input (image)
- Output probability of being good action
- Choose an action
- Observe: reward (/punishment)
- Improve
Policy learning

- Take some state as input (image)
- Output **probability** of being good action
- Choose an action
- Observe: reward (/punishment)
- Improve
Policy learning vs supervised learning

Supervised learning:
- You choose the output with the highest probability
- You get an immediate reward

Policy learning:
- You sample an action with given probability
- Wait til you get a reward to backprop (May be many steps)
Policy learning - action output

- Softmax is a typical output function for a discrete action space. The score function is then

$$\phi(s, a) - \mathbb{E}_{\pi_\theta}[\phi(s, \cdot)]$$

A good alternative for continuous action space can be a gaussian policy. Sample an action from a gaussian distribution around your output. The score function is then:

$$\frac{(a - \mu(s)) \phi(s)}{\sigma^2}$$
Playing a game of Pong

A simple game with actions:

- UP
- DOWN

Reward for:

- 1 for winning
- -1 for losing

So is there any other reward function?
Playing a game of Pong

- So your games can look something like this
- You have a lot of actions and a final result
Assigning credit for winning

You get a result, WIN! Great, but how do you know which action, caused the victory?
Assigning credit for winning

You get a result, WIN! Great, but how do you know which action, caused the victory?

- Well… you don’t
Which action caused the final results?

- In a winning series there may be many non-optimal actions
- In a losing series there may be good actions

The **true** effect is found by averaging out the noise, as winnings series tend to have more good action and visa versa.
Which action caused the final results?

We give each “network” in a winning series a gradient of 1 and in a losing series a gradient of -1.
Which action caused the final results?

We give each “network” in a winning series a gradient of 1 and in a losing series a gradient of -1.
A chain of actions can cause large variations in performance.

If we change one action early in the network, we can easily move into unchartered territory.
A chain of actions can cause large variations in performance.

If we change one action early in the network, we can easily move into unchartered territory.

Imagine a self-driving car model that is used to driving on roads. If it happens to miss the road, it may have no idea what to do.
A chain of actions can cause large variations in performance.

If we change one action early in the network, we can easily move into unchartered territory.

Imagine a self-driving car model that is used to driving on roads. If it happens to miss the road, it may have no idea what to do.

If one action in the chain changes, other earlier actions may go from WIN, WIN, WIN to LOSE, LOSE, LOSE

This high variance gradients make learning slow
A Discount factor

In pong and most other applications, the final actions leading up to the win relate more to the final result than other actions.

A discount factor is commonly applied (e.g. 0.99). So earlier actions get diminishingly less of the reward.

\( \gamma \) is the discount factor.

\[
R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}
\]
A Discount factor

In pong and most other applications, the final actions leading up to the win relate more to the final result than other actions.

A discount factor is commonly applied (e.g. 0.99). So earlier actions get diminishingly less of the reward.

\( \gamma \) is the discount factor.

This create less variance, as you “dare” to change distant actions less.
You need a whole series to do any learning…

\[ R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \]
Policy gradients - variance
Policy gradients - variance
Variance - all choices get the reward
Variance - other possible paths
Variance - high probability to chose some other path
Variance - same actions for same state: now negative
Q learning - Learning the reward function

- Learn value of state and action
- Find the action with highest estimated value
Q learning - Learning the reward function

- You learn the expected value of the reward in a given state for each action
- With this approach you don’t have to calculate a whole serie to do learning
- In some cases a value function may be useful

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

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

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

- In Q-learning we only need pairs of states and actions
- More stable gradients
- More sample efficient learning
- Not guaranteed to converge
Q learning - Learning the reward function

- In the start of your training the value function at each state is independent of action policy
- When you learn actions you also change the value functions
- The value function becomes flatter as the car learns to adapt to different situations
- A the end you get very steep change, as you have little time to adapt
Less variance in Q-learning
Variance - same actions for same state: now negative
Q-learning vs Policy learning

**Policy learning:**
- More stable
- Policy can be simpler to represent
- Imagine pong:
  - It can be easy to find out that you have to move in one direction this step
  - It can be hard to estimate the actual reward for that step
- Effective
  - With large or continuous action spaces
  - You get action directly
- “Built-in” stochastic policies

**Q-Learning:**
- Can converge faster
- Can come closer to global optimum
- Can be more flexible as you only need state pair to learn
  - Easy to do *experience replay* etc.
Actor-Critic - Best of both worlds

With an Actor-Critic model you can optimize the policy directly and still learn off-policy.

You don’t get the same variance problem since you only learn transition between steps at a time.

Basic actor-critic method:

Start with state s, and sample action a

1. get reward r from critic for s and a
2. sample action a’ from actor
3. estimate new reward r’ from critic
4. update critic with difference between r and r’ (or real reward)
5. update actor based on estimated reward r’
6. set a <- a’, s <- s’

Repeat 1-6
Actor-Critic - Best of both worlds

Applying advantage to actor update.

Instead of updating with the $\text{probability of action} \times \text{estimated reward}$

Update with $\text{probability of action} \times \text{improvement of expected reward from current state to next state}$

Then you will actually discourage actions that are worse than average from that state. E.g. all actions may give high reward. Alternatively all actions give negative reward, and you want to cut your losses.
Integrating reinforcement learning

In many applications you have to deal with parts of your system being non-differentiable:

- Obviously interacting with real world environment
- Hard attention
- Using memory (has to use RF to get real capacity)
Integrating reinforcement learning

In many applications you have to deal with parts of your system being non-differentiable:

- Obviously interacting with real world environment
- Hard attention
- Using memory (has to use RF to get real capacity)
Cool image application! Guess what game

**Generator** generate questions based on input image and previous questions.

**Oracle** answers yes or not to questions, given an image and a question.

**Guesser** outputs object answer based on question/answer pairs and image.
Cool image application! Guess what game

- Independently train each model supervised
- Use reinforcement learning to make the models collaborate