Details on Exercises

- You need to come to the recitations (Übungen)
- You will mutually work through the exercises with your class-mates
- Tutor will clarify questions and work through solutions
- Hand it in at the end
Machine learning has three branches:
- supervised learning
- unsupervised learning
- reinforcement learning

Supervised learning: \( \{x, y\} \) pairs to function \( f(x) \mapsto y \)
Example: images and captions creates automatic image captioning function

DenseCap, Johnson et al 2015

Unsupervised learning: \( \{x\} \) to low-dimensional representation \( y: f(x) \mapsto y \)
Example: Take images of faces and find few descriptive variables: can also generate new images

Khan et al 2018
Machine learning has three branches:
- supervised learning
- unsupervised learning
- reinforcement learning

Reinforcement learning: System $S$: policy $\pi(s) \mapsto a$ (sensor to action)

Example: learning to locomote

Linear Regression: $f(x) = w^T x$

Cost function: Squared error: $\mathcal{L}(w) = \frac{1}{n} \sum_i (w^T x_i - y_i)^2$

To avoid overfitting:

Regularization: $\mathcal{L}(w) = \frac{1}{n} \sum_i (w^T x_i - y_i)^2 + \lambda \|w\|^2$
Supervised learning with Neural Networks
Imitation learning for behavior generation

Artificial Neural Networks – a short introduction

Inspired by biological neurons:

Math model:
Artificial Neural Networks – a short introduction

Inspired by biological neurons, but extremely simplified:

Simple artificial Neuron

\[
\hat{y}_i = \phi \left( \sum_{j=1}^{d} w_{ij} x_j \right)
\]

\[
\phi(z) = \frac{1}{1 + e^{-z}} \quad \text{sigmoid}
\]

Like in regression problems:

- can use squared error:

\[
\mathcal{L}(w) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2
\]

(plus regularization)

Delta Rule

Perform gradient descent on \(\mathcal{L}\):

\[
w^t = w^{t-1} - \epsilon \frac{\partial \mathcal{L}(w)}{\partial w}
\]

\[
\hat{y}_i = \phi \left( \sum_{j=1}^{d} w_{ij} x_j \right)
\]

\[
\mathcal{L}(w) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2
\]
Artificial Neural Networks – a short introduction

Multilayer Networks – backpropagation
Stack layers of neurons on top of each other.

\[
\hat{y} = \ldots \phi^2(W^2 \phi^1(W^1 x))
\]

\[
L(W) = \frac{1}{2} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2
\]

Stochastic gradient descent (SGD)

- Loss/Error is expected empirical error: sum over examples (batch)

- SGD: update parameters on every example:

\[
\Delta W = -\epsilon \sum_{i}^{N} \frac{\partial (\hat{y}_i - y_i)^2}{\partial W}
\]

- Minibatches: average gradient over a small # of examples

Advantages: many updates of parameters, noisier search helps to avoid flat regions
Momentum

Speed up gradient descent

- Momentum: add a virtual mass to the parameter-particle

\[ \Delta W_t = -\epsilon \frac{\partial L(x_t)}{\partial W} + \alpha \Delta W_{t-1} \]

Advantages: may avoid some local minima, faster on ragged surfaces
Disadvantages: another hyperparameter, may overshoot

Adam (2014)

Rescale gradient for each parameter to unit size:

\[ W_t = W_{t-1} - \epsilon \frac{\langle \nabla W \rangle_{\beta_2}}{\sqrt{(\langle \nabla W \rangle^2_{\beta_2} + \lambda)}} \]

with moving averages: \( \langle \cdot \rangle_{\beta} \)

Artificial Neural Networks – a short introduction
Training: old and new tricks

- Derivative of sigmoid vanished for large absolute input (saturation)
- For deep networks (many layers) \( \Rightarrow \) gradient vanishes

ReLU

Use a simpler non-linearity:

\[ \phi(z) = \max(0, z) \]

CRelu: concatenate positive and negative

\[ \phi(z) = (\max(0, z), -\max(0, -z)) \]

Unit-derivative everywhere

Try: http://playground.tensorflow.org/
Artificial Neural Networks – a short introduction

- Trainability and more computer power
  - larger and deeper networks (≥ 6 layers)
- Breakthrough in performance in many ML applications
  Vision, NLP, Speech, ...

Convolutionary Network (CNN) – for vision

[http://vision03.csail.mit.edu/cnn_art/data/single_layer.png]

Automatic differentiation (AutoDiv)

- need to compute derivatives to perform gradient descent
- not really complicated but tedious and error prone

Solution: Let the computer do the work for us! Given code:

\[ y = \tanh( \text{dot}(W, x) + b) \]

Treat as a symbolic definition, not an imperative command.

Create computational graph:
  - generate function to evaluate expression
  - generate function for derivative (know deriv. of individual terms, and apply chain rule etc)

Common frameworks: Tensorflow, PyTorch, Theano (Python) but exist also for Matlab, C++ etc.
Break

Imitation learning:
Supervised learning for decision making
Sequential decision making: state $s_t$ → observation $o_t$ → action: $a_t \sim \pi(a|o_t)$

if $o_t = s_t$: fully observable

Imitation Learning

Behavioral Cloning: after collecting data of \{o_t, a_t\} pairs: train policy function to “replicate behaviour”
How does it perform?

[Video: Bojarski et al. 16, NVIDIA] badly!

Why does it not work?

Data collected during training deviates quickly from training-data
How about this?

Why did that work?

Video: Bojarski et al. 16, NVIDIA

side cameras, more data and data augmentation
Expected trajectory should be inside of training data.
Options:
- use some stabilizing controller
- collect data in a different manner

\[ p_{\pi_\theta}(o_t) \]

\[ \pi_\theta(a_t | o_t) \]

Can we make \( p_{\text{data}}(o_t) = p_{\pi_\theta}(o_t) \)?

\[ p_{\text{data}}(o_t) = p_{\pi_\theta}(o_t) \]
Dataset Aggregation

want: \( p_{\text{data}}(o_t) = p_{\pi \theta}(o_t) \)

Instead of improving \( p_{\pi \theta}(o_t) \), improve \( p_{\text{data}}(o_t) \)!

**DAgger: Dataset Aggregation**

Goal: collect training data from \( p_{\pi \theta}(o_t) \) (instead of \( p_{\text{data}}(o_t) \))

How? We can run \( \pi_\theta(a_t|o_t) \), but we need the correct actions? (labels)!

Algorithm:

1. train \( \pi_\theta(a_t|o_t) \) from (human) data \( D = \{o_1, a_1, \ldots, o_N, a_N\} \)
2. run \( \pi_\theta(a_t|o_t) \) and collect new observations \( D_\pi = o_1, \ldots, o_M \)
3. Ask human/expert to provide actions \( a_t \) (labels) for \( D_\pi \)
4. Aggregate: \( D \leftarrow D \cup D_\pi \)

and iterate [Ross et al. ’11]

**Dataset Aggregation in action**

![Image of a humanoid experiment with a graph showing average reward over Num of Rollouts]
Why is that not the ultimate solution?

- train $\pi_\theta(a_t|o_t)$ from (human) data $D = \{o_1, a_1, \ldots, o_N, a_N\}$
- run $\pi_\theta(a_t|o_t)$ and collect new observations $D_\pi = o_1, \ldots, o_M$
- Ask human/expert to provide actions $a_t$ (labels) for $D_\pi$
- Aggregate: $D \leftarrow D \cup D_\pi$

- Human is expensive and sometimes also does not know the right control command (see Humanoid)
- If we have already a machine-expert what is the point?
- $\Rightarrow$ distilling a faster policy

Summary: Imitation Learning with supervised learning from Humans

- Typically insufficient by itself
  - Distribution mismatch between training and test data
  - It is simple and can sometimes work well
    - Hacks (left and right images)
    - Samples from a stable trajectory distribution
    - Add more on-policy data, (DAgger)
Can we do DAgger without human labeling?

Yes, but only if we know something about the system (can automatically label data).

1. Train \( \pi_\theta(a_t|o_t) \) from (human) data \( D = \{o_1, a_1, \ldots, o_N, a_N\} \)
2. Run \( \pi_\theta(a_t|o_t) \) and collect new observations \( D_\pi = o_1, \ldots, o_M \)
3. Ask human/expert to provide actions \( a_t \) (labels) for \( D_\pi \)
4. Run computer to provide actions \( a_t \) (labels) for \( D_\pi \)
5. Aggregate: \( D \leftarrow D \cup D_\pi \)

But there is still a problem:

1. Train \( \pi_\theta(a_t|o_t) \) from (human) data \( D = \{o_1, a_1, \ldots, o_N, a_N\} \)
2. Run \( \pi_\theta(a_t|o_t) \) and collect new observations \( D_\pi = o_1, \ldots, o_M \)
3. Ask computer to provide actions \( a_t \) (labels) for \( D_\pi \)
4. Aggregate: \( D \leftarrow D \cup D_\pi \)
train $\pi_\theta(a_t|o_t)$ from (human) data $D = \{o_1, a_1, \ldots, o_N, a_N\}$
run $\pi_\theta(a_t|o_t)$ and collect new observations $D_\pi = \{o_1, \ldots, o_M\}$ run $\hat{\pi}(a_t|o_t)$ and collect new observations $D_{\hat{\pi}} = \{o_1, \ldots, o_M\}$
Ask computer to provide actions $a_t$ (labels) for $D_{\pi}$
Aggregate: $D \leftarrow D \cup D_{\pi}$

$\hat{\pi}(s_t) = N K_t s_t + k_t$, $\Sigma_{a_t}$ is a stochastic policy that can control robot

$$\hat{\pi}(a_t|s_t) = \arg \min_{\hat{\pi}} \sum_{t'=t}^{T} c(s_{t'}, a_{t'}) + \lambda D_{KL}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t))$$

distance to learned policy

replanning = Model predictive Control (later lecture)
Need to get the state $s$ (not only the camera) while training.
PLATO: Policy Learning with Adaptive Trajectory Optimization

\[
\hat{\pi}(a_t|s_t) = \arg \min_{\tilde{\pi}} \sum_{t'=t}^T c(s_{t'}, a_{t'}) + \lambda D_{KL}(\hat{\pi}(a_t|s_t)||\pi_\theta(a_t|o_t)) 
\]

\(\text{cost}\) \hspace{1cm} \text{distance to learned policy}

Input substitution trick
need state at training time but not at test time!

Objective: fly through forest at 2m/s
Main sensor: 1d laser

Flight in forest
PLATO is a way to perform policy learning from known controller
- Avoids unsafe rollouts
- Don’t need full state-space at test-time (because policy uses camera images)