A Gentle Introduction to Machine Learning

Second Lecture

Part I

Olov Andersson, AIICS

Slides online:

LINKÖPINGS UNIVERSITET

Outline of Machine Learning Lectures

First we will talk about **Supervised Learning**

- Definition
- Main Concepts
- General Approaches & Applications
- **Neural Networks and Deep Learning**
- **Pitfalls & Limitations**

Then finish with a short introduction to **Reinforcement Learning**

The idea is that you will be informed enough to select between and apply machine learning solutions if the need arises.

What About Deep Learning for Robot Behavior?

- Deep learning can require millions of examples
- As seen in pancake flipping video, learning with robots quickly gets tedious if you need to baby-sit it
- If you have an **automated teacher** however, like a numerical optimal control algorithm in a simulator, you can also deep learn robot behavior
- Mordatch et al, https://www.youtube.com/watch?v=IxrnT0JOs4o
- Another technique w/ real nano-quadcopters (deep ANN on-board the microcontroller)
- This is still supervised learning since it requires examples. Reinforcement learning is a more general approach

Training Advanced Representations

Learning these advanced representations raises two issues:

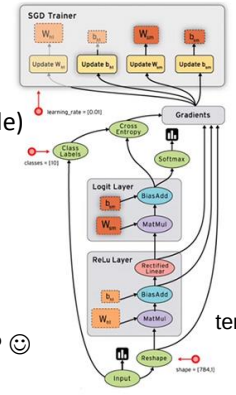
- Deep learning, in particular CNNs and RNNs, often result in very large networks with very large (”Big Data”) training sets.
- Advanced representations require modifications to backpropagation

Reverse-mode Automatic Differentiation is a technique that generalizes backpropagation to differentiate arbitrary scalar (loss) functions

Recent data flow languages like Tensorflow (Google) and Theano let you define **arbitrary models** from primitive mathematical operations and optimize them on one or more **GPUs**

Can be orders of magnitude faster!

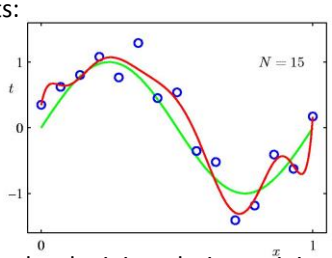
Will we ever have to manually differentiate again? ☺



tensorflow.org

Machine Learning Pitfall - Overfitting

- Models can overfit if you have **too many parameters in relation to the training set size**.
- Example: 9th degree polynomial regression model (10 parameters) on 15 data points:



Green: True function (unknown)
Blue: Training examples (noisy!)
Red: Trained model

(Bishop, 2006)

- This is **not** a local minima during training, it **is** the best fit possible on the given training examples!
- The trained model captured "**noise**" in data, variations independent of f(**x**)

Overfitting – Where Does the Noise Come From?

- Noise is small variations in the data due to **ignored** or **unknown variables,** that cannot be predicted via chosen feature vector **x**
  - Example: Predict the temperature based on season and time-of-day. What about atmospheric changes like a cold front? As they are not included in the model, nor entirely captured by other input features, this variation will show up as **random noise** for the model!
- With few data points to go on, the model can mistake the effect that unmodeled variables has on y as coming from variables **x** that **are** included.
- Since this relationship is merely chance, the model will **not generalize** well to future situations

Model Selection – Choosing Between Models

- In conclusion, we want to avoid **unnecessarily complex** models
- This is a fairly general concept throughout science and is often referred to as **Ockham's Razor**:

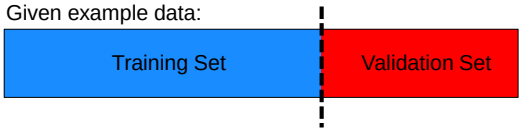
"*Pluralitas non est ponenda sine necessitate*"
-Willian of Ockham

"*Everything should be kept as simple as possible, but no simpler*."
-Albert Einstein (paraphrased)

- There are several mathematically principled ways to **penalize** model **complexity** during training, which we will not cover here.
- A simple approach is to use a separate **validation set** with examples that is **only** used for evaluating models of different complexity.

Model Selection – Hold-out Validation

- This is called a **hold-out validation set** as we keep the data away from the training phase
- Measuring performance on such a validation set is a **better metric of actual generalization** error to unseen examples
- With the validation set we can compare models of **different complexity** to select the one which generalizes best.
- Examples could be polynomial models of different order, the number of neurons or layers in an ANN etc.

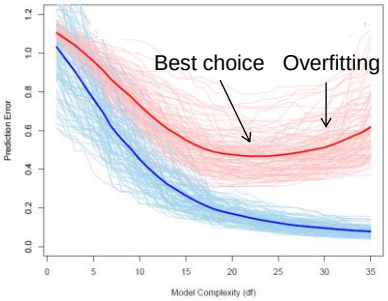


Model Selection – Selection Strategy

- As the number of parameters increases, the size of the hypothesis space also increases, allowing a **better fit to training data**
- However, at some point it is **sufficiently flexible** to capture the underlying patterns. Any more will just capture noise, leading to **worse generalization to new examples!**

Example: Prediction error vs. model complexity over many (simulated) data sets. (Hastie et al., 2009)

Red: Validation set (generalization) error
Blue: Training set error

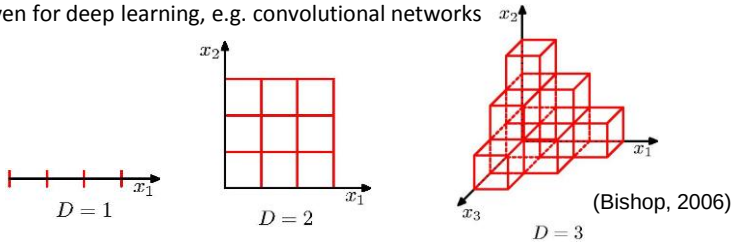


Limitations of Supervised Learning

- We noted earlier that the first phase of learning is usually to select the "**features**" to use as input vector **x** to the algorithm
- In the spam classification example we restricted ourselves to **a set of relevant words** (bag-of-words), but even that could be thousands
- Even for such binary features we would have needed $O(2^{\#features})$ examples to cover all **possible** combinations
- In a continuous feature space, there might be a difficult non-linear case where we need a grid with 10 examples along each feature dimension, requiring $O(10^{\#features})$ examples.

The Curse of Dimensionality

- This is known as the **curse of dimensionality** and also applies to reinforcement learning as we shall see later
  - However, this is a **worst case** scenario.
  - **The true amount of data needed for supervised learning depends on the model and the complexity of the function we are trying to learn**
  - **Deep learning** can be resistant due being able to capture abstractions
- Usually, learning works rather well even for many features
  - However, being able to construct a small set of informative features **can** be the difference between success and failure
  - Even for deep learning, e.g. convolutional networks



(Bishop, 2006)

Some Application Examples of Dimensionality

Computer Vision – Object Recognition

- One HD image can be 1920x1080 = 2 **million** pixels
- If **each pixel** is naively treated as one dimension, learning to classify images (or objects in them) can be a **million-dimensional problem**.
- Much of computer vision involves clever ways to extract a small set of descriptive features from images (edges, contrasts)
  - Recently **deep convolutional networks** dominate most benchmarks

Data Mining – Product models, shopping patterns etc

- Can be anything from a few key features to millions
- Can often get away with using **linear models**, for the very high-dimensional cases there are few easy alternatives

Some Application Examples of Dimensionality II

Robotics – Learning The Dynamics of Motion

- Realistic problems are often **non-linear**
- Ground robots have at least a handful dimensions
- Air vehicles (UAVs) have at least a dozen dimensions
- Humanoid robots have at least 30-60 dimensions
- The human body is said to have over 600 *muscles*

Multi-Agent Problems – Learning Behaviors

- If there are multiple agents, the naive way of supercomposing them into one problem will scale the dimensionality accordingly
- Decomposition into learning behaviours for individual agents plus cooperation can scale better

A Gentle Introduction to Machine Learning

Part II - Reinforcement Learning

Olov Andersson

Artificial Intelligence and Integrated Computer Systems
Department of Computer and Information Science
Linköping University

Introduction to Reinforcement Learning

- Remember:
  - In Supervised Learning agents learn to act given **examples** of correct choices.
- What if an agent is given **rewards** instead?
- Examples:
  - In a game of chess, the agent may be rewarded when it wins.
  - A soccer playing agent may be rewarded when it scores a goal.
  - A helicopter acrobatics agent may be rewarded if it performs a loop.
  - A pet agent may be given a reward if it fetches its masters slippers.
- These are all examples of **Reinforcement Learning**, where the agent itself figures out *how* to solve the task.

Defining the domain

- How do we formally define this problem?
- An agent is given a sensory **input** consisting of:

  State $s \in \mathcal{S}$
  Reward $R(s) \in \mathbb{R}$

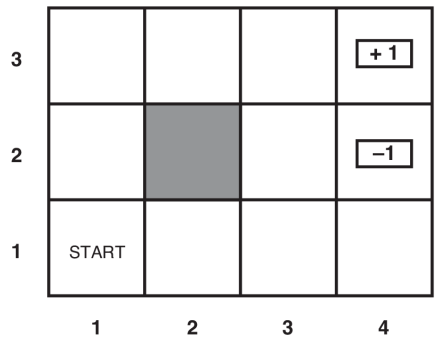
- It should pick an **output**

  Action $a \in \mathcal{A}$

- It wants to learn the "best" action for each state.

What do we need to solve?

- An example domain...
- $\mathcal{S}$ = {squares}
- $\mathcal{A}$ = {N,W,S,E}
- $R(s)$ = 0 except for the two terminal states on the right



- Considerations:
  - It may not know the effect of actions yet $p(s'|s,a)$
  - It may not know the rewards $R(s)$ in all states yet
  - Reward will be zero for all actions in all states not adjacent to the two terminal states.
  - **Need to consider all future reward!**

Rewards and Utility

- We define the reward for reaching a state s_i as $R(s_i)$
- To enable the agent to think ahead it must look at a sum of rewards over a **sequence** of states $R(s_{i+1}), R(s_{i+2}), R(s_{i+2}), \ldots$
- This can be formalized as the **utility** U for the sequence

$$U = \sum_{t=0}^{\infty} \gamma^t R(s_t), \text{ where } 0 < \gamma < 1 \quad (1)$$

- Where $\gamma < 1$ is the **discount factor** making the utility finite even for infinite sequences.
- A low γ makes the agent very short-sighted and greedy, while a gamma close to one makes it very patient.

The Policy Function

- We now have a utility function for a sequence of states
- ...but the sequence of states depends on the actions taken!
- We need one last concept, a **policy** function $\pi(s)$ decides which action to take in **each** state

$$a = \pi(s) \quad (2)$$

- Clearly, a good policy function is what we set out to find

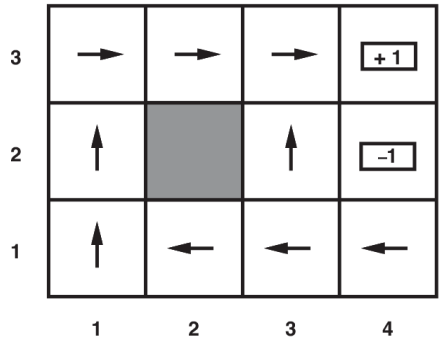
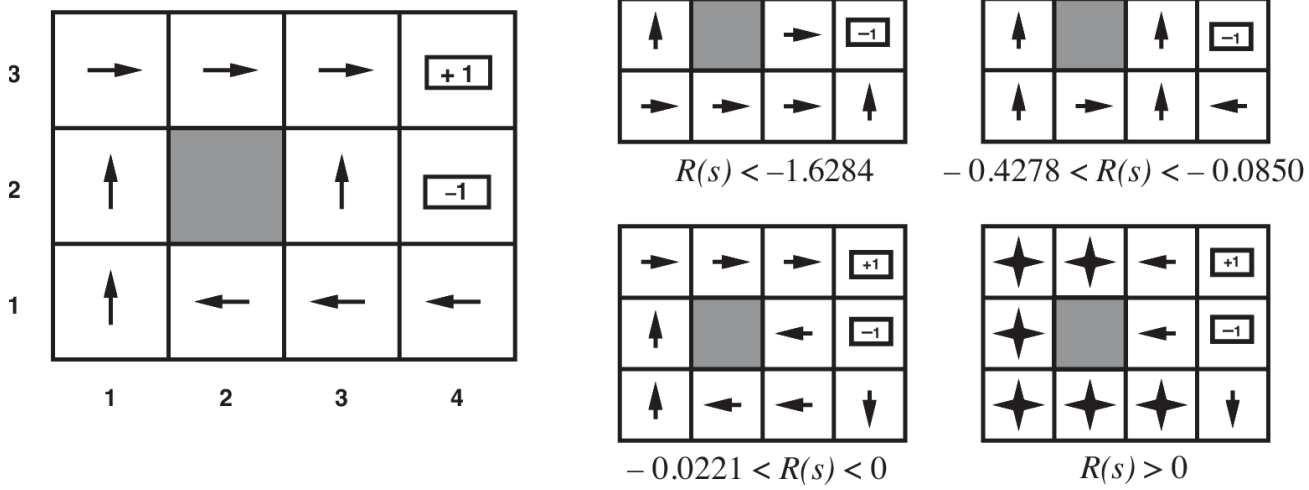
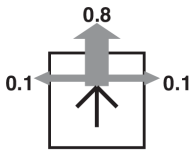


Figure : A policy function maps states to actions (arrows)

Examples of optimal policies for different R(s)



Assuming transition function (for each direction):



How to find such an optimal policy?

- There are two different philosophies for solving these problems
- Model-based reinforcement learning
  - Learn $R(s)$ and $f(s,a) = s'$ using supervised learning.
  - Solve a (probabilistic) planning problem using an algorithm like value iteration (not included in this course).
- Model-free reinforcement learning
  - Use an iterative algorithm that *implicitly* both adapts to the environment and solves the planning problem.
  - Q-learning is one such algorithm that has a very simple implementation.

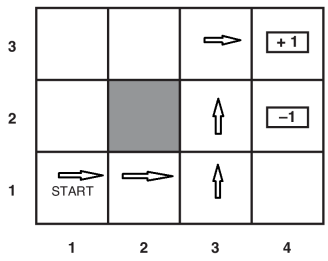
Q-Learning

- In Q-learning, all we need to keep track of is the "Q-table" $Q(s,a)$, a table of estimated utilities for taking action a in state s.
- Each time an agent moves the Q-values can be updated by nudging them towards the *observed* reward and the Q-value of the *observed* next state.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{\alpha' \in \mathcal{A}} Q(s',a') - Q(s,a)) \qquad (3)$$

  where α is the **learning rate** and γ the discount factor.
- This both exploits a recursive definition of utility and samples the transition function $p(s'|s,a)$ so we don't have to learn it. This means that the Q-function can **only** be updated when the agent actually **interacts** with the environment.
- If the transitions are **deterministic**, $\alpha = 1$ is optimal, but approximating continuous state by discrete ones can make actions appear non-deterministic (lab5).

The Q-table Update - An Example



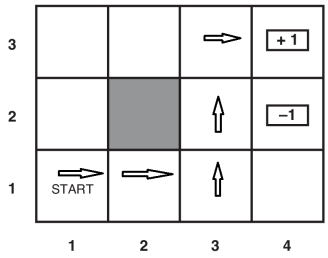
Where actions are N,E,S,W (North = up) and $\lambda = 0.9$. For simplicity the agent *repeatedly* executes the actions above, ending each episode in the terminal +1 state and restarting. Transitions are deterministic so we use learning rate $\alpha = 1$.

Begin by initializing all terminal $Q(s_T, *) =$ reward, all other $Q(s,a) = 0$
For each step the agent updates Q(s,a) for the *previous* state/action:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{\alpha' \in \mathcal{A}} Q(s',a') - Q(s,a))$$

After a while the Q-values will converge to the true utility

The Q-Learning Update - An Example



$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a' \in \mathcal{A}} Q(s',a') - Q(s,a))$$

First run (clarified): $Q(s_{3,3}, E) = 0 + 1 \cdot (0 + 0.9 \cdot \max(1,1,1,1) - 0) = 0.9$.
(Remember, all action Q-vals for terminal $s_{4,4}$ initialized to +1)
Second run: $Q(s_{3,2}, N) = 0 + 1 \cdot (0 + 0.9 \max(0, 0.9, 0, 0) - 0) = 0.81$,
$Q(s_{3,3}, E) = 0.9$ (unchanged due to learning rate $\alpha = 1$)
Third run: $Q(s_{3,1}, N) = 0 + 1 \cdot (0 + 0.9 \max(0.81, 0, 0, 0) - 0) = 0.729$,
$Q(s_{3,2}, N) = 0.81$, $Q(s_{3,3}, E) = 0.9$ (both unchanged). And so on...

Action selection while learning: Exploration

- That was assuming **fixed** actions. The agent should ideally pick the action with **highest utility**.
- However, always taking the highest estimated utility action while still learning will get the agent stuck in a sub-optimal policy.
- In the previous example, once the Q-table has been updated all the way to the start position, following that path will always be the only non-zero (and therefore best) choice.
- The agent needs to balance taking the best actions with exploration!
- Simple ϵ-greedy strategies do random movements $\epsilon\%$ of the time.

Curse of Dimensionality for Q-Learning

- Need to discretize continuous state and action spaces.
- The Q-table will grow exponentially with their dimension!
- Workaround: Approximate Q-table by supervised learning.
  - "Fitted" Q-iteration.
- If approximation **generalizes** well, we get large gains in scalability.
- Use deep learning $\rightarrow$ **deep reinforcement learning**
  - Deep ANN was used for the video game example (plus some tricks)
  - Google's Go champion combines several approaches, deep convolutional nets for approximating the game board, a tree-search planning approach for updating utilities and more...
- Caveat: Non-linear approximations may impede convergence.

Q-Learning - Final Words

- Implementation is very simple, having no model of the environment.
  - It only needs a table of Q(s,a) values!
- Once the Q(s,a) function has converged, the optimal policy $\pi^*(s)$ is simply the action with highest utility in the table for each s
- Technically the learning rate α actually needs to decrease over time for perfect convergence.
- Q-learning must also be combined with exploration
- Q-learning requires very little computational overhead per step
- The curse of dimensionality: The Q-table grows exponentially with dimension. A good approximation can avoid this.
- Model-free methods may require more interactions with the world than model-based, and much more than a human.

What we expect you to know about Machine Learning

- *See reading instructions on course page!*
- Definitions of supervised and reinforcement learning.
- The main concepts of both, no advanced deep learning.
- Worked examples on Q-learning.
- We will provide you with formulas if we ask you to calculate anything.