A Gentle Introduction to Machine Learning

First Lecture

Olov Andersson, AIICS

2018

Outline of Machine Learning Lectures

- Introduction to machine learning (two lectures)
  - Supervised learning
  - Reinforcement learning (lab)
- Recent Advances: Deep learning (one lecture)
  - Applied to both SL and RL above
  - Code examples

What is Machine Learning about?

- To enable machines to *learn and adapt* skills without programming them
- Our only frame of reference for learning is from biology
  - …but brains are hideously complex, the result of ages of evolution
- Like much of AI, Machine Learning mainly takes an **engineering approach**[1]
  - Remember, humanity didn't master flight by just imitating birds!

1. Although there is occasional biological inspiration

Theoretical Foundations

Hint: Lots of math...

- Statistics (theories of how to **learn from data**)
- Optimization (how to **solve** such learning problems)
- Computer Science (efficient **algorithms** for this)

This intro will focus more on **intuitions** than mathematical details

ML also **overlaps** with multiple areas of engineering, e.g.

- Computer vision
- Natural language processing (e.g. machine translation)
- Robotics, signal processing and control theory

...but traditionally differs by focusing more on **data-driven** models and AI

Why Machine Learning

- Difficulty in **manually programming** agents for every possible situation
- The world is ever **changing**, if an agent cannot adapt, it will fail
- Many argue learning is required for Artificial **General** Intelligence (AGI)
- We are still far from human-level general learning ability…
  - …but the algorithms we have so far have shown themselves to be useful in a wide range of applications!

Some Application Aspects

- **Not as data-efficient** as human learning, but once an AI is “good enough”, it can be cheaply duplicated
- Computers work **24/7** and you can usually **scale** throughput by piling on more of them

Software Agents (Apps and web services)

- Companies collect ever more data and processing power is cheap (“*Big data*”)
- Can **let an AI learn how to improve business**, e.g. smarter product recommendations, search engine results, or ad serving
- Can **sell services that traditionally required human work**, e.g. translation, image categorization, mail filtering, content generation…?

Hardware Agents (Robotics)

- Although data is more expensive, many capabilities that humans take for granted like **locomotion, grasping, recognizing objects**, **speech** have turned out to be ridiculously **difficult to manually construct rules** for.

Example – Google Deepmind's Go Agent

…in **narrow applications** machine learning can even rival or beat human performance

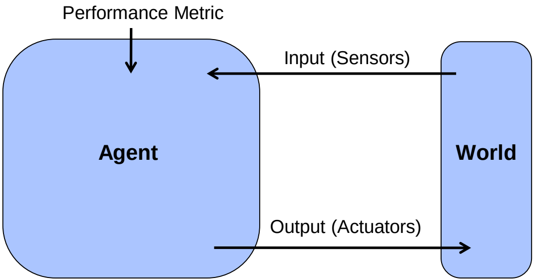
Example – Stanford Helicopter Acrobatics

…in **narrow applications** machine learning can even rival human performance

To Define Machine Learning

Given a task, mathematically encoded via some performance metric, a machine can improve its performance by learning from experience (data)

From the agent perspective:



The Three Main Types of Machine Learning

Machine learning is a young science that is still changing, but traditionally algorithms are divided into three types depending on their purpose.

- **Supervised Learning**
- **Reinforcement Learning**
- **Unsupervised Learning**

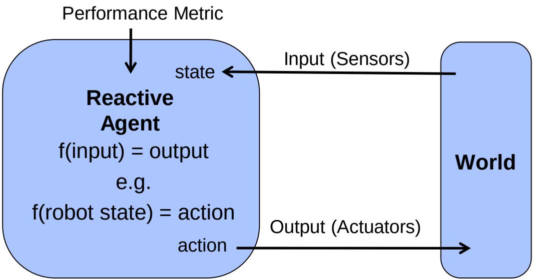
Supervised Learning at a Glance

In **supervised learning**

- Agent has to learn from **examples of *correct*** behavior
- Formally, **learn an unknown function f*(x) = y*** given examples of (x, y)
- Performance metric: **Loss** (difference) between learned function and correct examples

Supervised Learning – Agent Perspective

Representation from agent perspective:



…but it can also be used as a component in other architectures

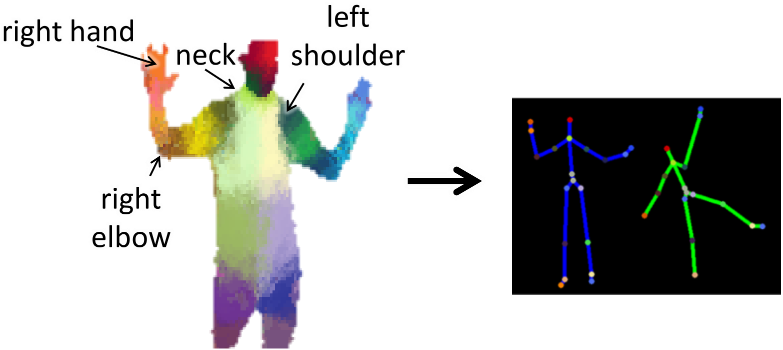
Supervised Learning is surprisingly powerful and ubiquitous

Some real world examples

- **Spam Filter**: f(mail) = spam?
- **Microsoft Kinect**: f(pixels, distance) = body part

Supervised Learning of Body Part Classification

- Learn y=f(x) from examples (x,y),...
  - x = ”depth image”, y = ”body part”
  - Given new depth image below, predict body part per pixel:



Used in Microsoft Kinect SDK (Shotton et al, CVPR 2011)

Supervised Learning of ”Super Resolution”

- Learn y=f(x) from examples (x,y),...
  - x = ”low-res image”, y = ”high-res image” (real numbers)
  - Given new low-res image x’ below, predict y’:

Reinforcement Learning at a Glance

In **reinforcement learning**

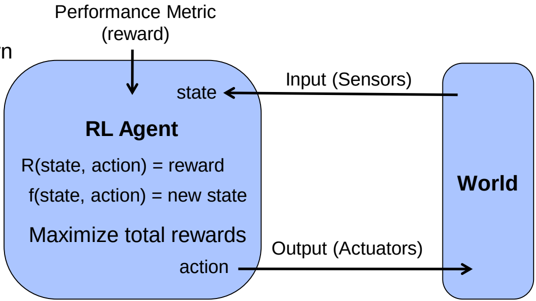
- World may have **state** (e.g. position in maze) and be **unknown** (how does an action change the state)
- In each step the agent is only given current **state** and **reward** instead of examples of correct behavior
- Performance metric is sum of **rewards over time**
- Combines learning with a **planning** problem
  - Agent has to **plan a sequence of actions** for good performance
- The agent can **even learn on its own** if the reward signal can be mathematically defined

Reinforcement Learning at a Glance II

RL is based on a utility (reward) maximizing agent framework

- Outcomes (next state,reward) of actions in different states are ***learned***
- Agent plans ahead to maximize reward over time

Do we need to learn world models?



Real world examples – Robot Behavior, Game Playing (AlphaGo…)

Demo – Learning Robot Behavior

- Learning to flip pancakes, ”supervised” and reinforcement learning.

Unsupervised Learning at a Glance

In **unsupervised learning**

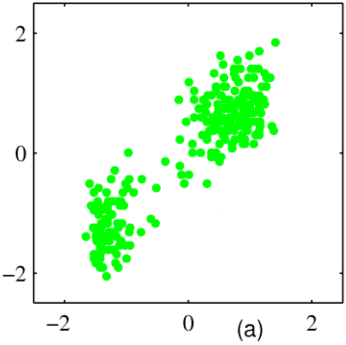
- Neither a correct answer/output, nor a reward is given
- Task is to **find some structure** in the data
- Performance metric is some **reconstruction** error of patterns compared to the input data distribution

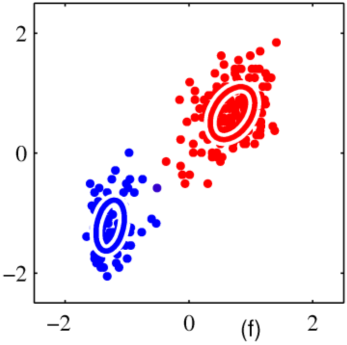
Examples:

- **Clustering** – When the data distribution is confined to lie in a small number of “clusters” we can find these and use them instead of the original representation
- **Dimensionality Reduction** – Finding a suitable lower dimensional representation while preserving as much information as possible

Recent trend: Found structure can be used to **generate new examples!**

Clustering – Continuous Data

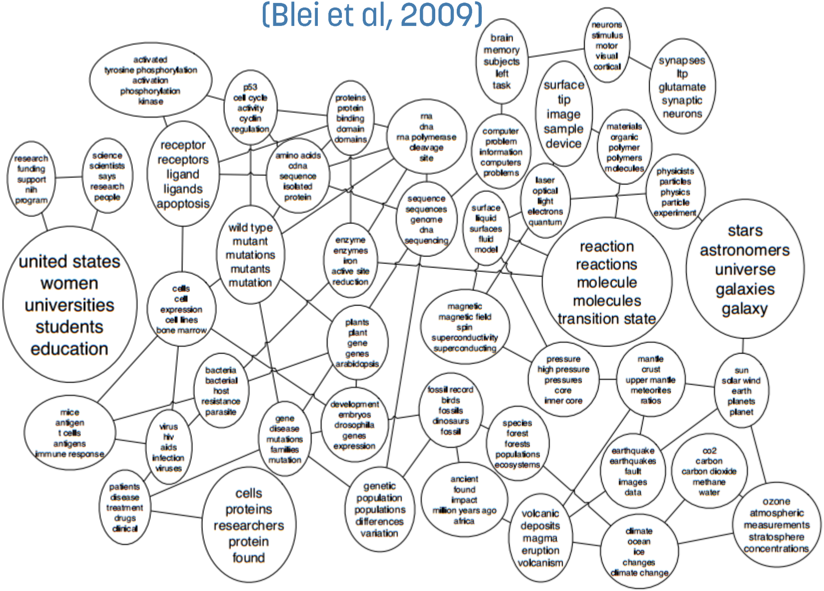




Two-dimensional continuous input

(Bishop, 2006)

Unsupervised Learning of Topics in Science Articles (Blei et al, 2009)



(Deep) Unsupervised Learning – Do AI's dream? ☺

- Generative model ("Dream up" new data) fed e.g. images...
- Can we use them to e.g. fill in scenery in a movie scene?

(Karras et al, 2018)
https://youtu.be/G06dEcZ-QTg

Outline of This Lecture

Today we will talk about **Supervised Learning**

- Definition
- Main Concepts
- General Approaches & Applications
- Trend: Neural Networks and Deep Learning

Formalizing Supervised Learning

Remember, in Supervised Learning:

- Given tuples of ***training data*** consisting of ($\mathbf{x}$,y) pairs
- The objective is to learn to **predict** the **output** y' for a new input **x'**

Formalized as **searching** for approximation to **unknown function** y = ***f(x)***, given N examples of **x** and y: ($\mathbf{x}_1$,y_1), … ,($\mathbf{x}_n$,y_n)

- A candidate approximation is sometimes called a **hypothesis** (book)

Two major classes of supervised learning

- **Classification** – Output are **discrete** category labels
  - Example: Detecting disease, y = "healthy" or "ill"
- **Regression** – Output are **numeric** values
  - Example: Predicting temperature, y = 15.3 degrees

In either case, input data $\mathbf{x}_i$ could be **vector valued** and **discrete**, **continuous** or **mixed**. Example: $\mathbf{x}_1$ = (12.5, "cloud free", true).

Supervised Learning in Practice

Can be seen as **searching** for an approximation to unknown function y = ***f(x)*** given N examples of **x** and y: ($\mathbf{x}_1$,y_1), … ,($\mathbf{x}_n$,y_n)

Want the algorithm to **generalize** from **training** examples to new inputs **x’,** so that y’=f(**x’**) is “close” to the correct answer.

1. First construct an **input vector** $\mathbf{x}_i$ of examples by encoding relevant problem data. This is often called the **<u>feature vector</u>**.
   - Examples of such ($\mathbf{x}_i$, y_i) is the **training set.**
2. A model is selected and **trained** on the examples by **<u>searching</u>** for **parameters** (the hypothesis space) that yield a good approximation to the unknown true function.
3. Evaluate performance, (carefully) tweak algorithm or features.

Feature Vector Construction

Want to learn ***f(x)* = *y*** given N examples of **x** and y: ($\mathbf{x}_1$,y_1), … ,($\mathbf{x}_n$,y_n)

Most standard algorithms work on **real number variables**

- If inputs ***x*** or outputs y contain categorical values like "book" or "car", we need to encode them with numbers
  - With only two classes we get y in {0,1}, called **binary classification**
  - Classification into multiple classes can be reduced to a sequence of binary one-vs-all classifiers
- The variables may also be structured like in **text**, **graphs**, **audio**, **image** or **video** data

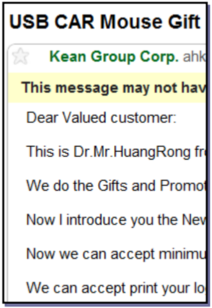
Finding a suitable feature representation **can be non-trivial**, but there are standard approaches for the common domains

- With sufficient data it can also be learned via *deep learning* (later…)

Example of Feature Vector- Bag of Words

One of the early successes was **learning spam filters**

Spam classification example:



Each mail is an input, some mails are flagged as spam or not spam to create training examples.

Bag of Words Feature Vector:

Encode the existence of a fixed set of relevant **key words** in each mail as the **<u>feature vector</u>**.

$\mathbf{x_i}$ **=** words$_i$ **=**

Feature	Exists?
"Customer"	1 (Yes)
"Dollar"	0 (No)
"Nigeria"	0
"Accept"	1
"Bank"	0
....	...

y_i = 1 (spam) or 0 (not spam)

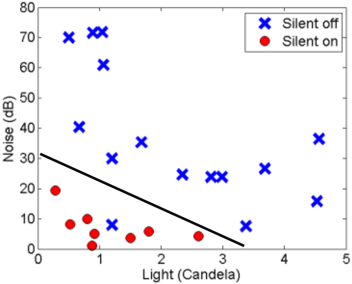
Simply learn f(x)=y using suitable <u>classifier</u>!

Example: Simple Linear **Classification**

I. Construct a **feature vector $\mathbf{x_i}$** to be used with examples of y_i
II. Select algorithm and **train** on training data by searching for a good approximation to the unknown function

Fictional example: A learning smartphone app that determines if silent mode should be on/off at different levels of **background noise** and **light** based previous user choices.
Feature vector $\mathbf{x_i}$ **= (Noise, Light level)**, y_i = {"silent on", "silent off"}

- Select the familiy of **linear** discriminant functions
- Train the algorithm by searching for a line that **separates the classes well**
- New cases will be classified according to which side they fall



Example: Simple Linear Regression

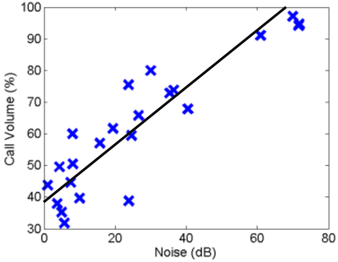
I. Construct a **feature vector $\mathbf{x}_i$** to be used with examples of y_i
II. Select algorithm and **train** on training set by searching for a good approximation to the unknown function

Fictional example: Same smartphone app but now we want to predict the ring volume based on background noise level (only)

Feature vector $\mathbf{x_i}$ **=** (Noise dB), y_i = (Volume %**)**

- Select the familiy of **linear** functions
- **Train** the algorithm by searching for a line that **fits the data well**

…but how does "training" really work?



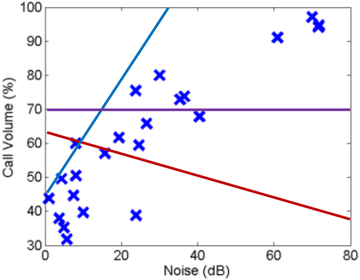
Training a Learning Algorithm

Feature vector $\mathbf{x}_i$ = (Noise in dB), outputs y_i = (Volume %)

- Want to find approximation *h*(x) to the unknown function f(x)
- As an example we select the hypothesis space to be the family of polynomials of degree one, that is **linear** functions:

$$y_i = w_1 \cdot x_i + w_0$$

- The hypothesis space of $h_{\mathbf{w}}(x)$ has two **parameters** $\mathbf{w} = (w_1, w_0)$
- How do we find parameters that result in a **good** approximation h?



Three (poor) linear hypotheses

Training a Learning Algorithm – Loss Functions

How do we find parameters **w** that result in a **good** approximation $h_{\mathbf{w}}(x)$?

- Need a performance metric for function approximations of unknown f(x)
  - **Loss functions** $L(f(x), h_{\mathbf{w}}(x))$

- Minimize deviation against the N example data points
  - For **regression** one common choice is a **sum square loss** function:

$$L(f(x), h_{\mathbf{w}}(x)) = (f(x) - h_{\mathbf{w}}(x))^2 = \sum_{i=1}^{N} (y_i - h_{\mathbf{w}}(x_i))^2$$

- Search in continuous domains like **w** is known as **optimization**
  - (if unfamiliar, see Ch4.2 in course book AIMA)

Training a Learning Algorithm – Optimization

How do we find parameters **w** that minimize the loss?

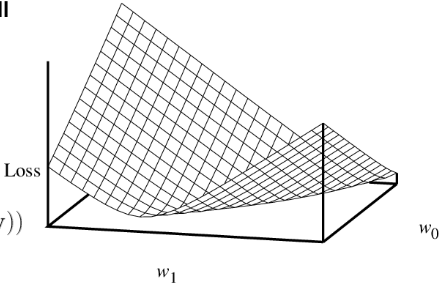
- Optimization approaches typically move in the **direction that locally decreases** the loss function
- Simple and popular approach: **gradient descent**

Initialize **w** to some random point in the parameter space
loop until decrease in loss is **small**
    for each w_j in **w do**

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} L(\mathbf{w})$$

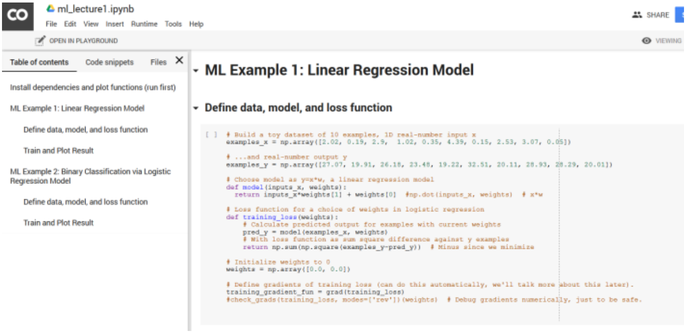
Note:

$$Gradient = (\frac{\partial}{\partial w_0} L(\mathbf{w}), \frac{\partial}{\partial w_1} L(\mathbf{w}))$$



Worked Example – Linear Regression

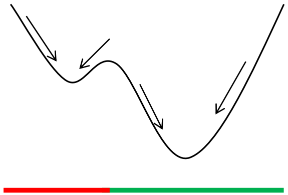
- Google Colab at: https://goo.gl/94VGye
- NOTE: May need to be signed in to a Google account and **save** or download workbook. Playground mode seems to complain about security issues.



Training a Learning Algorithm – Limitations

Limitations

- Locally greedy – Gets stuck in local minima unless the loss function is **convex** w.r.t. **w**, i.e. there is **only one minima**.
- **Linear models are convex**, however most more advanced models are **vulnerable** to getting stuck in local minina.
- Care should be taken when training such models by using for example **random restarts** and picking the least bad minima.



If we happen to start in red area, optimization will get stuck in a bad local minima!

Training a Learning Algorithm – Loss Functions II

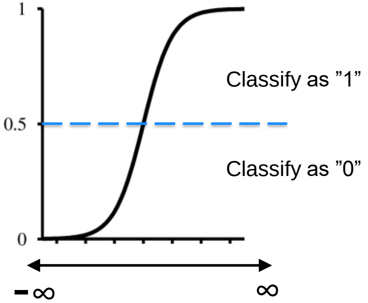
- What about classification?
  - Squared error does not make sense when target output discrete set {0,1}
- Custom loss functions for classification
  - Minimize number of missclassifications (unsmooth w.r.t. parameter changes)
  - Maximize information gain (used in decision trees, see book)
- These require specialized parameter search methods
- Alternative: Make outputs **probabilities [0,1]** by **squashing** predicted **numeric outputs** via sigmoid (”S”)

Sigmoid functions allow us to do use **any** regression model with binary classification by def. Pr(y=”1”|X) = g(model(x))

Where g is **”logistic” sigmoid** :

$$g(x) = \frac{1}{1+e^{-x}}$$

For >2 classes, use **soft-max** (see book)



Worked Example – Binary Classification via Linear Logistic Regression

- Google Colab at: https://goo.gl/94VGye
- NOTE: May need to be signed in to a Google account and **save** or download workbook. Playground mode seems to complain about security issues.
- Scroll down to ML Example 2: Binary Classification

Linear Models in Summary

Advantages

- Linear algorithms are **simple and computationally efficient**
  - For both regression and classification
- Training them is a **convex** optimization problem, i.e. one is guaranteed to find the **best** hypothesis in the space of linear hypothesis
- Can be extended by non-linear feature transformations

Disadvantages

- The hypothesis space is very restricted, it cannot handle non-linear relations well

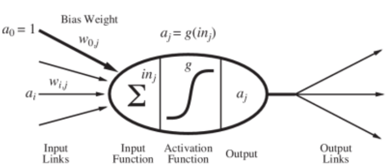
Still **widely used** in applications

- Recommender Systems – Initial Netflix Cinematch was a linear regression, before their **$1 million** competition to improve it
- At the core of many big internet services. Ad systems at Twitter, Facebook, Google etc...

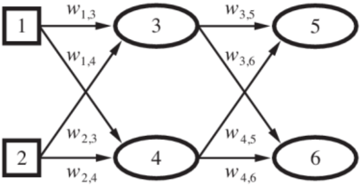
Beyond Linear Models – Artificial Neural Networks

- One **non-linear model** that has captivated people for decades is **Artificial Neural Networks (ANNs)**

- These draw upon inspiration from the physical structure of the brain as an interconnected network of "**neurons**", emitting electrical "spikes" when excited by inputs (represented by non-linear "**activation functions**")



The Neuron



The Network

Artificial Neural Networks – The Neuron

- In (one input) linear regression we used the model:

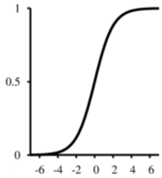
$$y_i = w_1 \cdot x_i + w_0$$

- Each **neuron** in an ANN is a linear model of **all** the inputs passed through a **non-linear** *activation function* g, representing the ”spiking” behavior.

$$y = g(\sum_{i=1}^{k} w_i x_i + w_0)$$

- The activation function is traditionally a **sigmoid**, but other options exist

$$g(x) = \frac{1}{1 + e^{-x}}$$

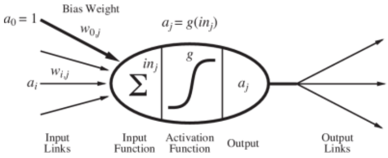


- ANNs generalize logistic linear regression!

Artificial Neural Networks – The Neuron II

- However, there is not just one neuron, but a **network** of neurons!
- Each neuron gets inputs from all neurons in the previous layer.
- We rewrite our neuron definition using a_i for the input, a_j for the output and $w_{i,j}$ for the weight parameters:

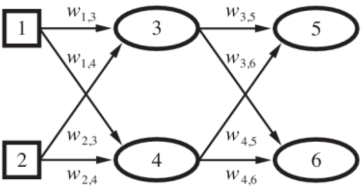
$$a_j = g(\sum_{i=1}^{k} w_{i,j} a_i + w_0)$$



Artificial Neural Networks – The Network

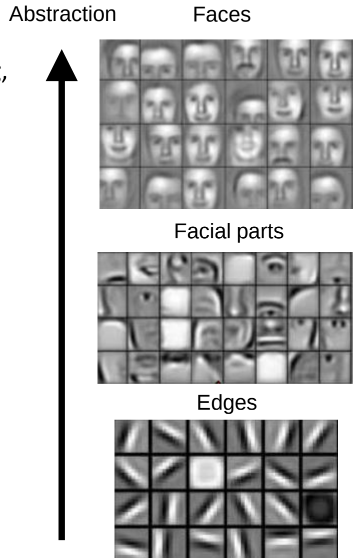
- The networks are composed into **layers**
- In a traditional **feed-forward** and **fully-connected** ANN, all neurons in a layer are connected to all neurons in the next layer, but not to each other
- Expanding the output of a second layer neuron (5) we get

$$
\begin{aligned}
a_5 &= g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \\
&= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2)))
\end{aligned}
$$



Why Multi-layer Neural Networks?

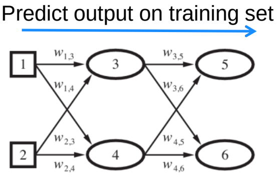
- Recent surge of successes with **deep learning**, using multi-layer models like ANNs to better capture **layers of abstractions** in data.
- Some tasks are uniquely suited to this like vision, text and speech recognition, where they hold state-of-the-art results.
- Already used by Google, MSFT etc.
- These require **large amounts of data and computation** to train, although unsupervised techniques can reduce need for data.
- More on this later.



(Honglak Lee, 2009)

Artificial Neural Networks – Training

- How do we train an ANN to find the best parameters $w_{i,j}$ for each layer?
- Like before, by **optimization**, minimizing a **loss function**
- What is the **computational complexity** of ANN gradients?
- Just evaluting network prediction for ANN with p parameters is O(p)



- Naive symbolic/numerical differentiation needs O(p) evaluations
  - This means computational complexity of $O(p^2)$!
- Deep learning networks often have >1M parameters. Can we do better?

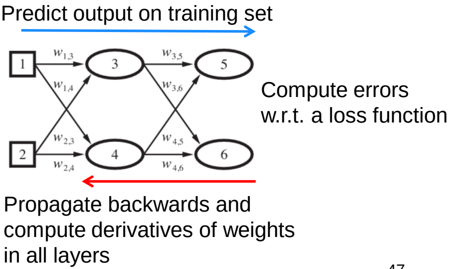
Artificial Neural Networks – Backpropagation

Some intuitions:

- Consider the chain rule of differentiation
  - E.g assume f(x) = g(h(i(x))), then f(x)' = g'(h(i(x)))h'(i(x))i'(x)
- ANN layers are just compositions of sums and non-linear functions g()

$$
\begin{aligned}
a_5 &= g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \\
&= g(w_{0,5} + w_{3,5}g(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2)))
\end{aligned}
$$

- ANN derivatives can be computed layerwise backwards, and terms are shared across parameter derivatives!
- Caching these terms gives rise to a famous **O(p)** gradient algorithm called **backpropagation**



Artificial Neural Networks - Demo

- See interactive examples of ANN training
  http://playground.tensorflow.org/
  - 2D input x -> 1D y (binary classification or regression)

- You can try playing with
  - Different data sets vs. network size
  - Deeper neurons can capture more complex patterns
  - Classification vs. Regression
  - Learning rate (Scaling of gradient descent step)

Artificial Neural Networks – Summary

Advantages

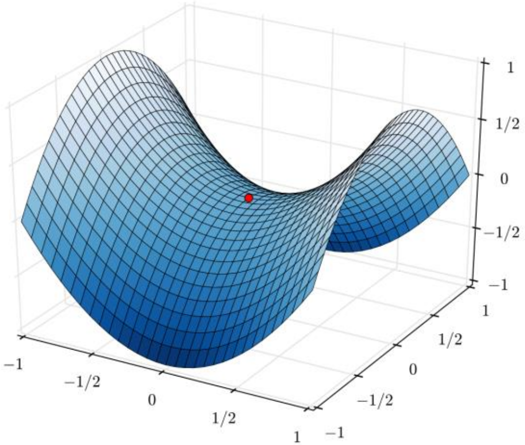
- Very large hypothesis space, under some conditions it is a **universal approximator** to any function **f(x)**
- Some biological justification (real NNs more complex)
- Can be layered to capture abstraction (**deep learning**)
  - Used for speech, object and text recognition at Google, MSFT etc.
  - For best results use <u>architectures tailored to input type</u> (see DL lecture)
  - Often using millions of neurons/parameters and GPU acceleration.
- Modern **GPU-accelerated tools** for large models and Big Data
  - Tensorflow (Google), PyTorch (Facebook), Theano etc.

Disadvantages

- Training is a **non-convex** problem with **saddle points** and **local minima**
- Has **many tuning parameters** to twiddle with (number of neurons, layers, starting weights, gradient scaling...)
- **Difficult to interpret** or debug weights in the network

What Was a Saddle Point Again?

- Believed to be a more common problem than local minima for ANN



Thank you for listening!