# A Gentle Introduction to Machine Learning

**First Lecture**

Olov Andersson, AIICS

LINKÖPINGS UNIVERSITET

## What is Machine Learning about?

- To imbue the capacity to *learn* into machines
- 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

- Statistics (learn from **data**)
- Optimization (intertwined with both learning and decision making)
- Computer Science (efficient **algorithms**)

However, in this introduction we will focus more on **intuitions.**

It **overlaps** with multiple areas of engineering, e.g.

- Signal processing
- Computer vision
- Control
- Robotics

...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 a general learning agent!
  - but the algorithms we have so far have shown themselves to be useful in a wide range of applications!

## Some Application Aspects

- Not as versatile as human learning, but domain specific problems can often be processed **much faster**
- Computers work 24/7 and you can often **scale** performance by piling on more of them

**Data Mining, Recommender Systems**

- Companies collect ever more data and processing power is cheap
- Put it to use automatically analyzing the performance of products!
- Machine Learning is almost ubiquitous on the web: **Mail filters, search engines, product recommendations, customized content, ad serving**…
- "Big Data" – much hyped technology trend.

**Robotics**

- 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.



## Demo – Stanford Helicopter Acrobatics

…in narrow applications machine learning can even rival human performance

## Demo – Stanford Helicopter Acrobatics

…in narrow applications machine learning can even rival human performance

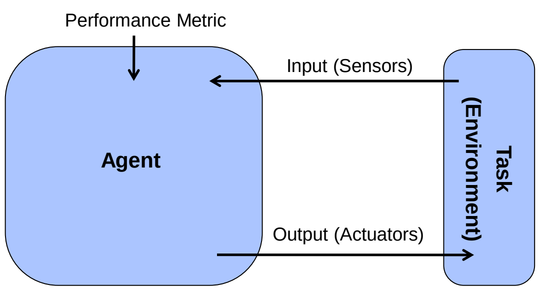


## To Define Machine Learning

*A machine learns with respect to a particular task T, performance metric P, and type of experience E, if the system reliably* ***improves*** *its performance P at task T, following experience E*

-Tom Mitchell

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

*A machine* ***learns*** *with respect to a particular task T, performance metric P, and type of experience E, if the system reliably* ***improves*** *its performance P at task T, following experience E*

-Tom Mitchell

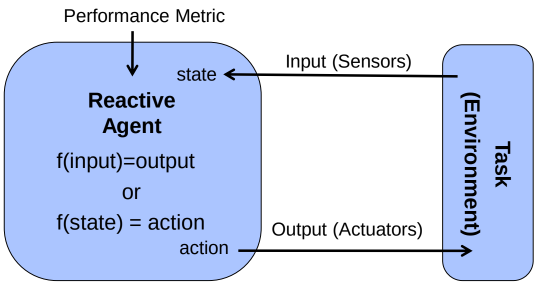
In Supervised Learning:

- Examples of ***correct*** behavior is ***given*** in a *training* phase.
- Experience: Tuples of correct (Input,Output) pairs.
- Performance metric: How well ***learned*** output prediction matches the **correct** output.

Mathematically, want to approximate an unknown function ***f(x) = y*** given examples of (x, y)

## Supervised Learning – Agent Perspective

Representation from agent perspective:




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

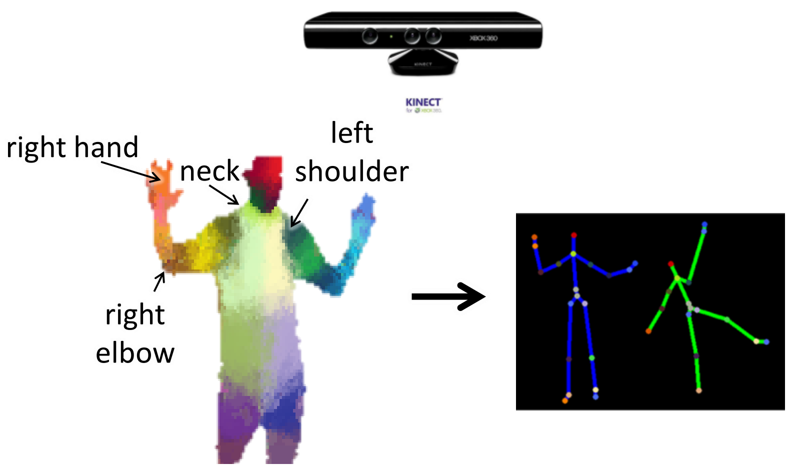
Supervised Learning is surprisingly powerful and ubiquitous

Some real world examples

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



## Body Part Classification on the Microsoft Kinect




Shotton et al @ MSR, CVPR 2011

## Reinforcement Learning at a Glance

*A machine **learns** with respect to a particular task T, performance metric P, and type of experience E, if the system reliably **improves** its performance P at task T, following experience E*

-Tom Mitchell

In Reinforcement Learning:

- A ***reward*** is given at each step instead of the correct input/output
- Instead of mimicking example behavior, the **agent plans actions** to maximize **reward over time**
- Experience E: history of inputs, chosen outputs and rewards
- Performance metric P is sum of **reward over time**

Inspired by early work in psychology, and how pets are trained

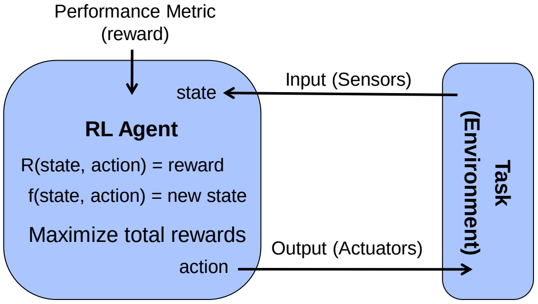
The agent can learn on its own if the reward signal can be mathematically defined.



## Reinforcement Learning at a Glance II

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

- Rewards of actions in different states are ***learned***
- Agent plans ahead to maximize reward over time




Real world examples – Robot Control, Game Playing (Checkers…)

## Demo - Robot Control

- Learning to flip pancakes.



## Unsupervised Learning at a Glance

*A machine **learns** with respect to a particular task T, performance metric P, and type of experience E, if the system reliably improves its performance P at task T, following experience E*

-Tom Mitchell

In Unsupervised Learning:

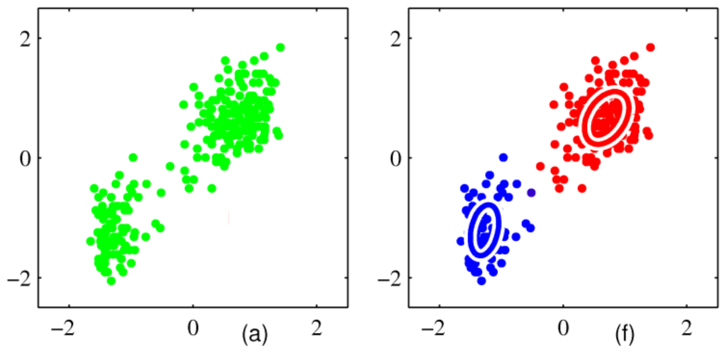
- The Task is to find some simpler structure or patterns in the data
- Neither a correct answer/output, nor a reward is given
- Experience E can be arbitrary input data
- P is some **reconstruction** error of patterns compared to the input data distribution

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

## Clustering – Continuous Data

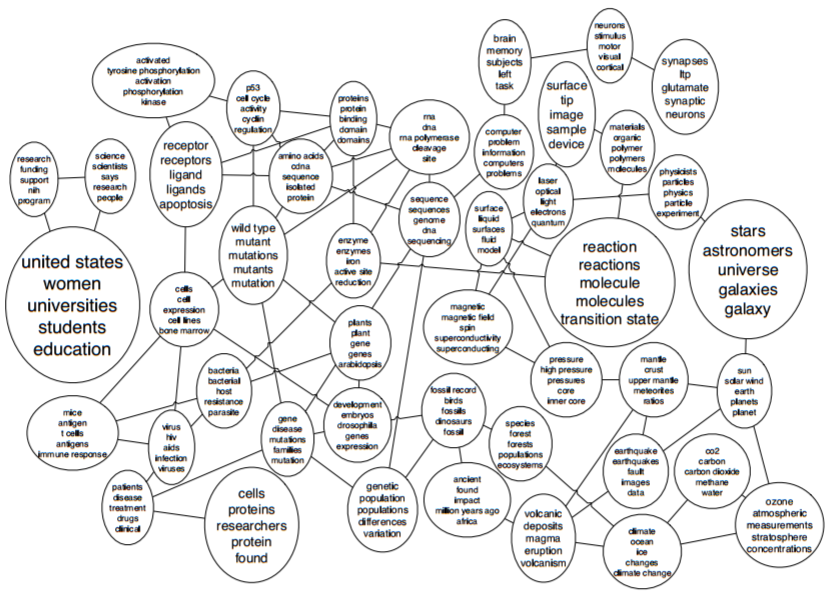



Two-dimensional continuous input

(Bishop, 2006)



## Topic Modeling – Science Articles (Blei et al, 2009)



## 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.

2016-09-26 20

## 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).

2016-09-26 21

## 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 **feature vector**.
   - Examples of such ($\mathbf{x}_i$, $y_i$) is the **training set.**
2. A model is selected and **trained** on the examples by **searching** for **parameters** (the hypothesis space) that yield a good approximation to the unknown true function.
3. Evaluate performance, (carefully) tweak algorithm or features.

2016-09-26 22

## Feature Vector Construction

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

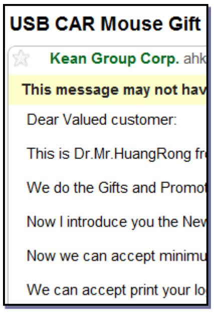
- Most standard algorithms work on **real number variables**
- If inputs ***x*** or outputs y contain categorical values like "book" or "car", we need to encode them with numbers
  - With only two classes we get y in {0,1}, called **binary classification**
  - Classification into multiple classes can 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

2016-09-26 23

## 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 **feature vector**.

$\mathbf{x}_i$ = 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 classifier!



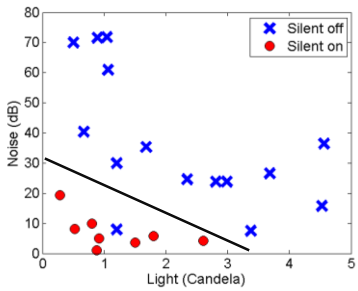
## Simple Linear Classification Example

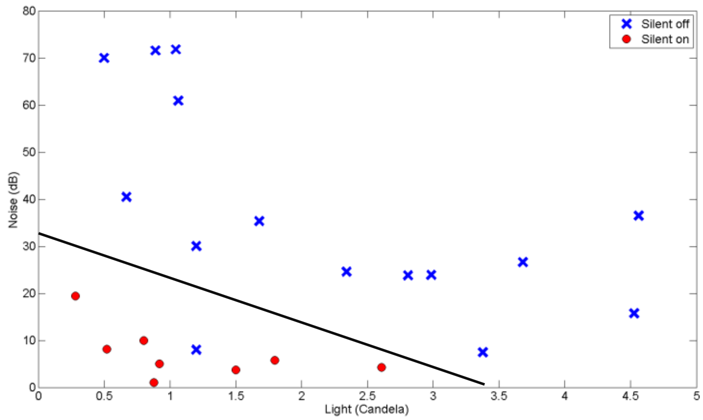
I. Construct a **feature vector $\mathbf{x}_i$** to be used with examples of $y_i$
II. Select algorithm and **train** on training data by searching for a good approximation to the unknown function

Fictional example: A learning smartphone app that determines if silent mode should be on or off based on background noise, light level and previous user choices.

Feature vector **$\mathbf{x}_i$ = (Light level, Noise)**, $y_i$ = {"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








## Simple Linear Regression Example

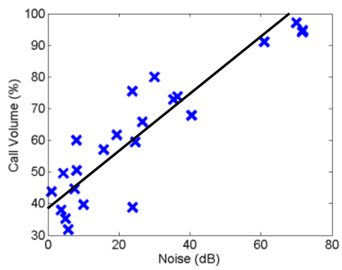
I. Construct a **feature vector $\mathbf{x}_i$** to be used with examples of $y_i$
II. Select algorithm and **train** on training set by searching for a good approximation to the unknown function

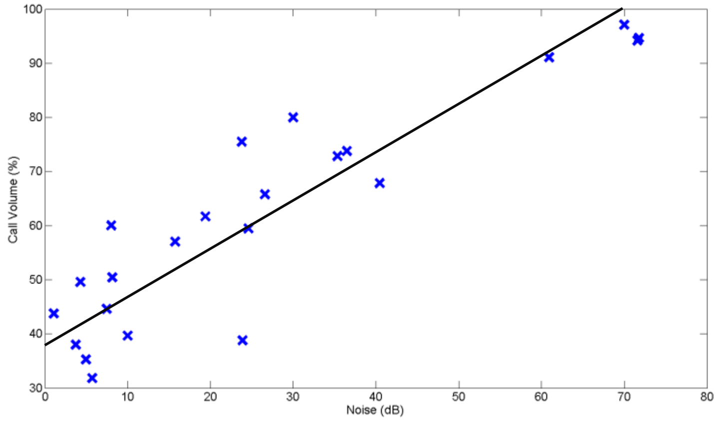
Fictional example: Same smartphone app but now we want to predict the ring volume based on noise level (only)

Feature vector $\mathbf{x}_i$ **=** (Noise), $y_i$ = (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?







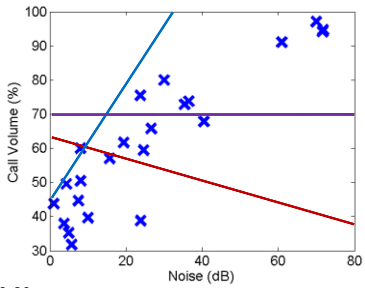

# Training a Learning Algorithm

Feature vector $\mathbf{x}_i$ = (Noise), 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 approximations
- Maximize some function of how well it **fits the data**
  - Equivalently: minimize deviation between approximation and data
  - **Loss functions** $L(f(x), h_{\mathbf{w}}(x))$
- 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**
  - (see Ch4.2 in course book AIMA)

2016-09-26 30

## Training a Learning Algorithm – Optimization

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

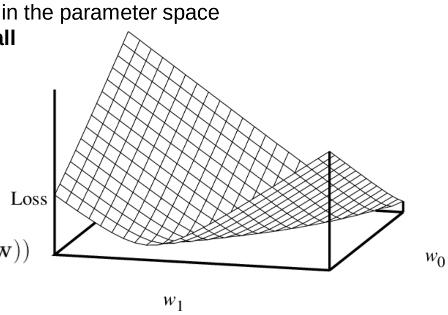
- Optimization approaches typically move in the **direction that locally decreases** the loss function
- Simple and popular approach: **gradient descent**

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:

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

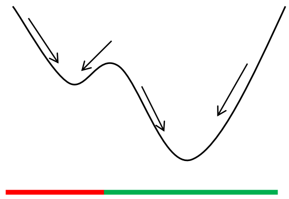



2016-09-26 31

## 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.



Start positions in red area will get stuck in a local minima!



## Training a Learning Algorithm – Loss Functions II

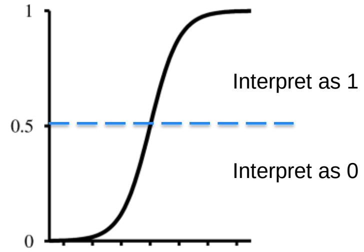
- What about classification?
  - Squared error does not make sense when target output in {0,1}
- Custom algorithms for classification
  - Minimize number of missclassifications (unsmooth w.r.t. parameter changes)
  - Maximize information gain (used in decision trees, see book)
- These require specialized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

Sigmoid functions allow us to use any regression method for binary classification.

Logistic function for binary classification:

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

Soft-max (see book) for multiple classes



## Linear Models in Summary

**Advantages**
- Linear algorithms are **simple and computationally efficient**
- 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

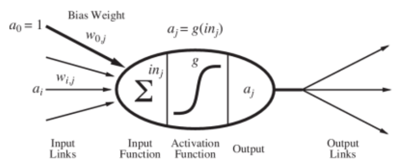
They are **widely used** in applications
- Recommender Systems – Initial Netflix Cinematch was a linear regression, before their **$1 million** competition to improve it
- At the core of many big internet services. Ad systems at Twitter, Facebook, Google etc...



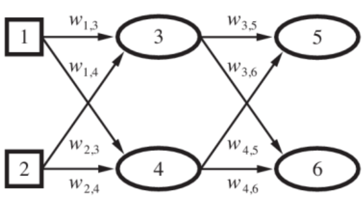
## Beyond Linear Models – Artificial Neural Networks

- 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:

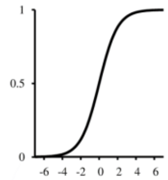
  $y_i = w_1 \cdot x_i + w_0$

- 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

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



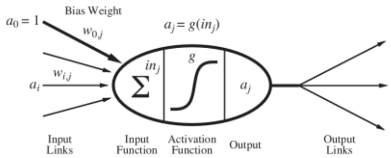

- ANNs generalize logistic linear regression!



## 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:

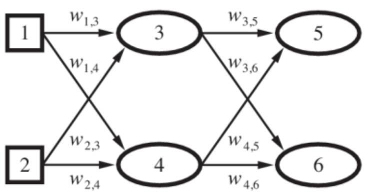
$$a_j = g(\sum_{i=1}^{k} w_{i,j} a_i + w_0)$$



## 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

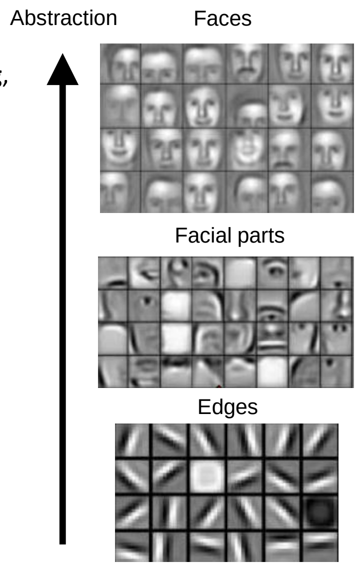
$$a_5 = g(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4)$$
$$= g(w_{0,5} + w_{3,5}\,g(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}\,g(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2))$$




2016-09-26 38

## Why Multi-layer Neural Networks?

- Recent surge of successes with **deep learning**, using multi-layer models like ANNs to better capture **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.



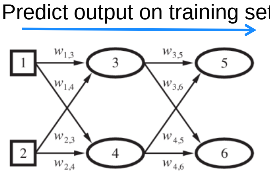

(Honglak Lee, 2009)

2016-09-26 39

## 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)

Predict output on training set




- 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

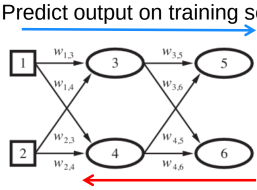
**Some intuitions:**

- 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 derivatives!
- Caching these terms gives rise to a famous O(p) gradient algorithm called **backpropagation**

Predict output on training set




Compute errors w.r.t. a loss function

Propagate backwards and compute derivatives of weights in all layers

# Artificial Neural Networks - Demo

- See interactive examples of ANN training
  http://playground.tensorflow.org/
- 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)



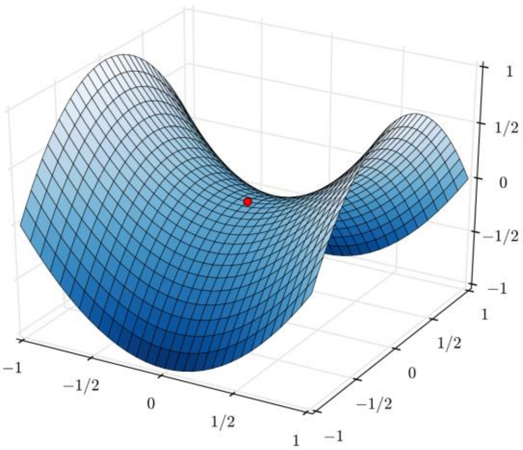
# Artificial Neural Networks – Summary

**Advantages**
- Very large hypothesis space, under some conditions it is a **universal approximator** to any function **f(x)**
- Some biological justification (real spiking dynamics more complicated)
- Can be layered to capture abstraction (**deep learning**)
  - Used for speech, object and text recognition at Google, MSFT etc.
  - Often using millions of neurons/parameters and GPU acceleration.
- Modern GPU-accelerated tools for large models and Big Data
  - Tensorflow (Google), Theano, Torch (Facebook) 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?

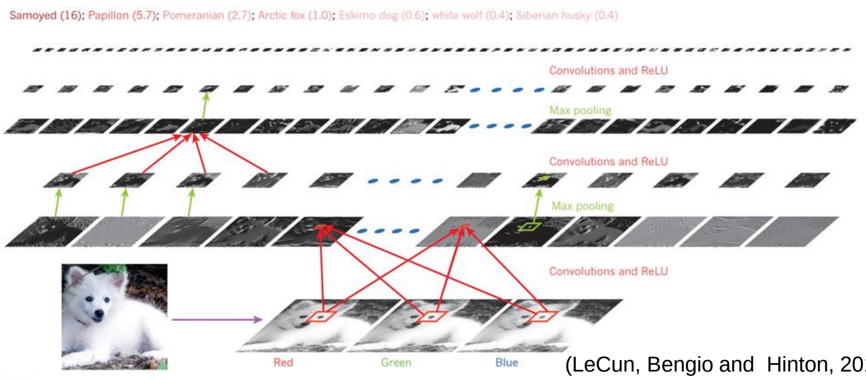
- Believed to be a more common problem than local minima for ANN




2016-09-26 44

## Learning Advanced Representations: Convolutional Neural Networks

- Images results in millions of inputs (pixels) for **fully-connected** ANN
- **Convolutional Neural Networks (CNNs)** better exploit structure in images
- They use "convolutional" layers with **local connections** to capture **translation invariance** and "pooling" layers to capture **scale invariance**

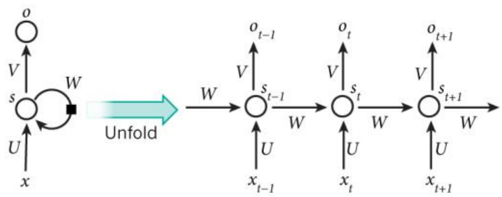



(LeCun, Bengio and Hinton, 2015)

2016-09-27 45

## Learning Advanced Representations: Sequential Dependence

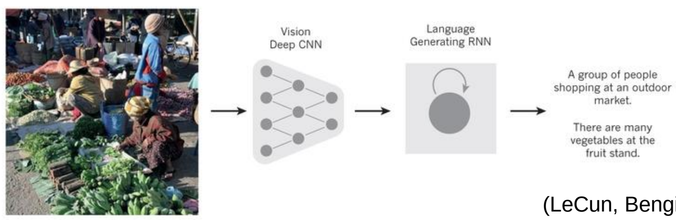
- We used the Bag of Words feature vector in the spam classification example. It's simple but it **discards the sequential structure** of the text.
- This is flawed since the **meaning** of a text strongly depends on the order of the words!
- **Recurrent Neural Networks (RNN)** depend not only on current input **x**, but also remembers **internal state s** representing previous inputs (e.g. words)
- **RNNs are difficult to train** (successful variant: "Long Short-Term Memory")



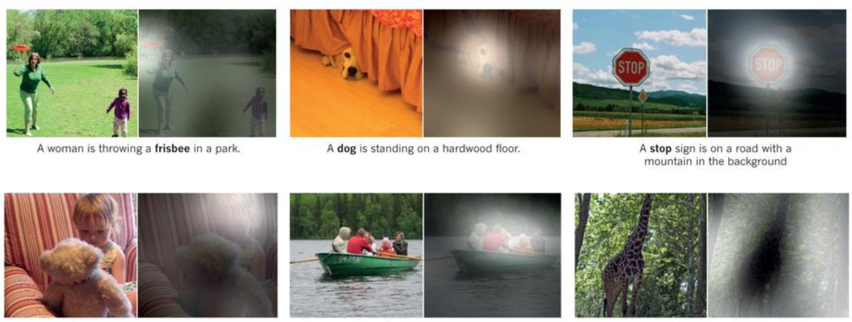

(LeCun, Bengio and Hinton, 2015)



## Combining Representations




(LeCun, Bengio and Hinton, 2015)



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.

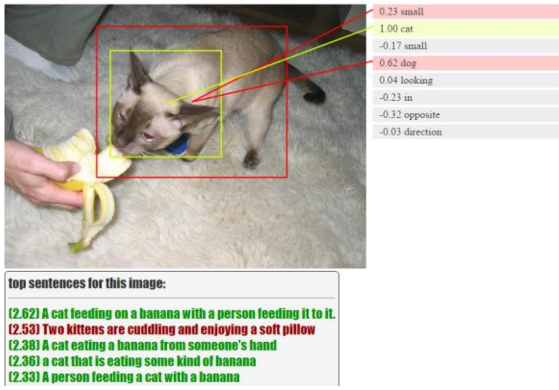
A group of **people** sitting on a boat in the water.

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

## Demo – Visual-Semantic Alignment

Paper: http://cs.stanford.edu/people/karpathy/deepimagesent/
Demo: http://cs.stanford.edu/people/karpathy/deepimagesent/rankingdemo/

Impressive, but not perfect. Some "weird" mistakes highlight on-going debate on what neural networks have really learned.






## Training Advanced Representations

Learning these advanced representations raises two issues:

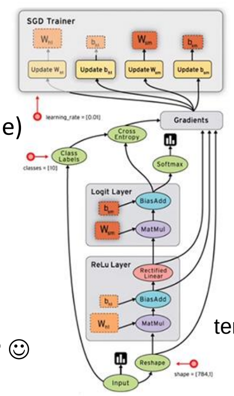
- Deep learning, in particular CNNs and RNNs, often result in very large networks with very large ("Big Data") training sets.
- Advanced representations require modifications to backpropagation

**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