

# Neural Networks - MLP

DSI 3 May 2017  
F. Burkholder

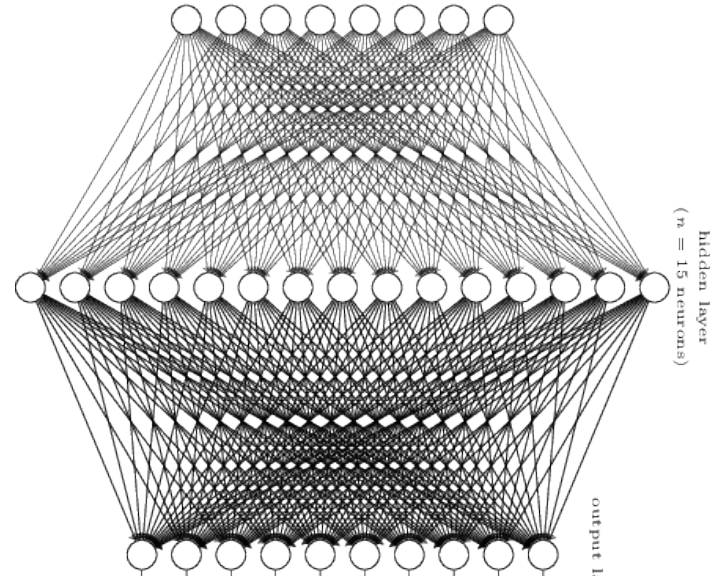
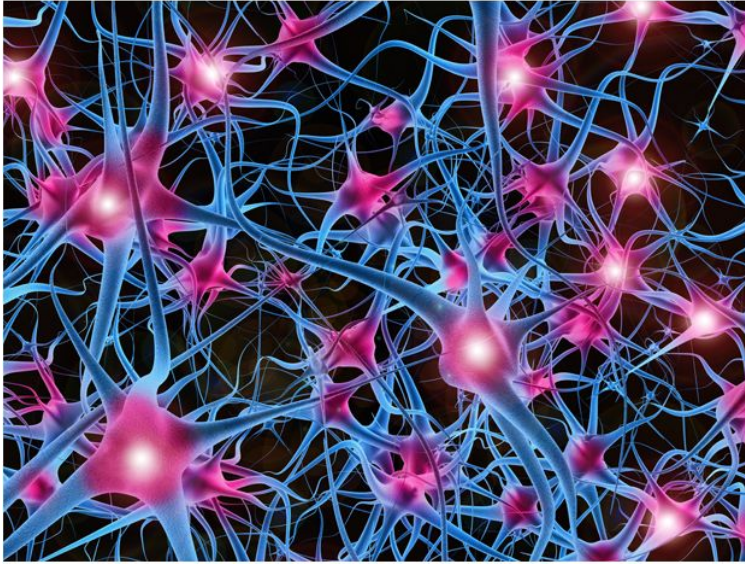
# Part 1 Objectives

Objectives of this lecture:

- Debrief on homework
- Neural net history, why they were pursued, break-throughs and set-backs
- Review feed-forward calculation and back-propagation
- Discuss full - scale networks (multilayer perceptrons)
- Hyperparameters and training
- Keras introduction
- References

# Neural Net motivation

# The ultimate parallel computer



Number of parameters of biggest artificial neural net: 160,000,000,000 (Digital Reasoning, 2015)

The number of neurons in your brain: 86,000,000,000

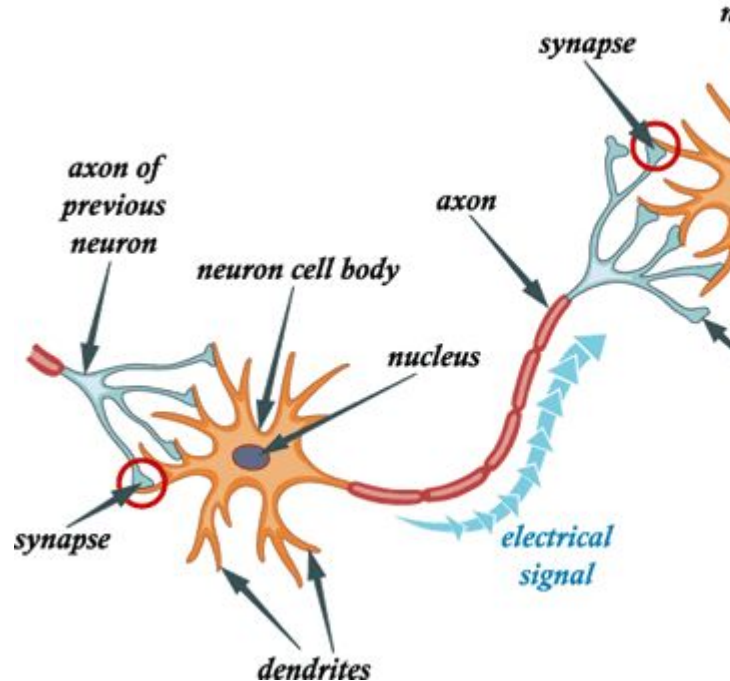
The number of synapses in your brain: 1,000,000,000,000,000

The average number of synapses per neuron: 10,000

# Neural Net history

# Biomimicry

McCulloch-Pitts  
first neuron model in 1943



BULLETIN OF  
MATHEMATICAL BIOPHYSICS  
VOLUME 5, 1943

## A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,  
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,  
AND THE UNIVERSITY OF CHICAGO

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

### *I. Introduction*

Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse.

# Improvements to MCP

- 1949, Donald Hebb, neuropsychologist

*"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."*

- 1957, Frank Rosenblatt invents the Perceptron

- Initializes the weights to random values (e.g. -1.0 to 1.0)
- Weights change during supervised learning according to the delta rule,  $\sim(y_i - y_p)$ . After a certain number of training passes through the whole training set (a.k.a. the number of *epochs*) stop changing the weights.
- Implements a learning rate that affects how quickly weights can change.
- Adds a bias to the activation function that shifts the location of the activation threshold and allows this location to be a learned quantity (by multiplying it by a weight).


## How to pronounce epoch time?

Elizabeth Goldberg, B.A. in English, I know at least eight words. I've always **pronounced** it EE-pok. Merriam-Webster's website provides three possible pronunciations: 'e-pək (EH-pik), 'e-,pāk (EH-pok), and 'ē-,pāk (EE-pok). The first way is probably the most correct, considering it's Latin and Greek origins.


# The XOR affair

- *Perceptrons, an introduction to computational geometry* (book by Minsky and Papert 1969)
  - From Wikipedia: “critics of the book state that the authors imply that, since a single artificial neuron is incapable of implementing some functions such as the [XOR](#) logical function, larger networks also have similar limitations, and therefore should be dropped.”
  - **Clarification: single layer perceptron networks are limited to being linear classifiers. Not true of deeper MLP (multi-layer perceptron) networks.**


AND

		A	
		0	1
B	0	0	0
	1	0	1

OR

		A	
		0	1
B	0	0	1
	1	1	1

XOR

		A	
		0	1
B	0	0	1
	1	1	0

?



# Back propagation

## LEARNING INTERNAL REPRESENTATIONS BY ERROR PROPAGATION

David E. Rumelhart, Geoffrey E. Hinton,  
and Ronald J. Williams

September 1985

ICS Report 8506



13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM Mar 85 to Sept 85	14. DATE OF REPORT (Year, Month, Day) September 1985	15. PAGE COUNT 34
16. SUPPLEMENTARY NOTATION To be published in J. L. McClelland, D. E. Rumelhart, & the PDP Research Group, <i>Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol 1. Foundations.</i> Cambridge, MA: Bradford Books/MIT Press.			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	learning; networks; perceptrons; adaptive systems; learning machines; back propagation
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
<p>This paper presents a generalization of the perceptron learning procedure for learning the correct sets of connections for arbitrary networks. The rule, called the generalized delta rule, is a simple scheme for implementing a gradient descent method for finding weights that minimize the sum squared error of the system's performance. The major theoretical contribution of the work is the procedure called error propagation, whereby the gradient can be determined by individual units of the network based only on locally available information. The major empirical contribution of the work is to show that the problem of local minima is not serious in this application of gradient descent.</p>			

# Back propagation

## LEARNING INTERNAL REPRESENTATIONS BY ERROR PROPAGATION

David E. Rumelhart, Geoffrey E. Hinton,  
and Ronald J. Williams

September 1985

ICS Report 8506



This paper presents a generalization of the perceptron learning procedure for learning the correct sets of connections for arbitrary networks. The rule, called the generalized delta rule, is a simple scheme for implementing a gradient descent method for finding weights that minimize the sum squared error of the system's performance. The major theoretical contribution of the work is the procedure called error propagation, whereby the gradient can be determined by individual units of the network based only on locally available information. The major empirical contribution of the work is to show that the problem of local minima is not serious in this application of gradient descent.

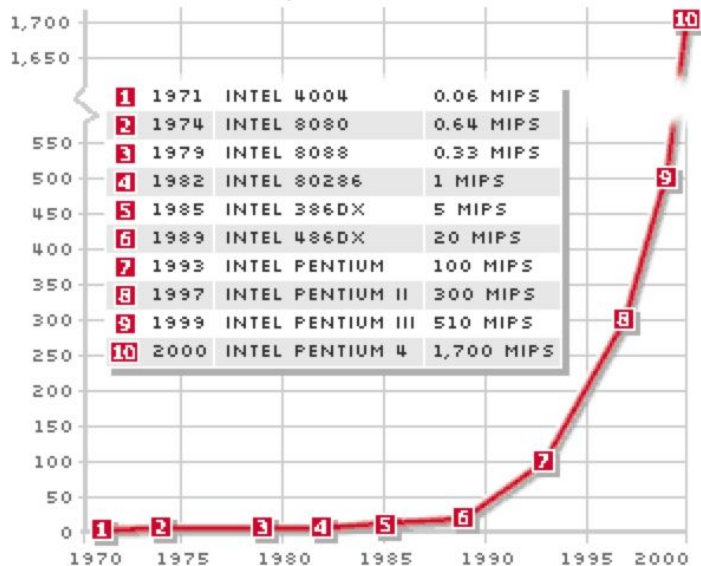
# Compute power

**BBC NEWS**

## In Pictures: PC Power

### How Intel processors have grown

millions of instructions per second



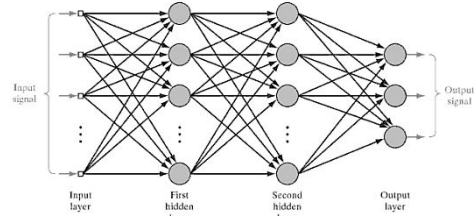
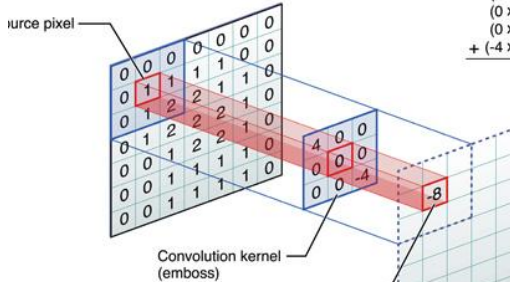
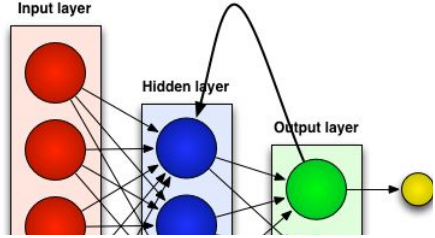
### Rise in power

Computer-processing power has risen exponentially in the past 50 years.

A general rule of thumb is that computer power doubles every 18-24 months. The number of transistors on a chip equates roughly to its processing power.

The result is incredible-quality video, extremely realistic and innovative games and ever-faster response times.

# Some types of ANNs (not exhaustive)

ANN	Description	Picture of architecture
<p>Multilayer perceptron (MLP)</p>	<p>Standard algorithm for supervised learning. Used for pattern, speech, image recognition.</p>	 <p>The diagram illustrates a Multilayer Perceptron (MLP) architecture. It consists of four layers of nodes: an input layer, two hidden layers (labeled 'First hidden layer' and 'Second hidden layer'), and an output layer. Arrows indicate the flow of information from the input layer through the hidden layers to the output layer. The input layer is labeled 'Input signal' and the output layer is labeled 'Output signal'.</p>
<p>Convolutional neural network</p>	<p>Variation of MLP where node connections inspired by visual cortex. Uses kernels to aggregate/transform information in network. Used for image, video recognition, natural lang. proc..</p>	 <p>The diagram illustrates a Convolutional Neural Network (CNN) architecture. It shows a 'source pixel' grid (a 5x5 grid of numbers) being processed by a 'Convolution kernel (emboss)' (a 3x3 grid of numbers). The resulting output is a 3x3 grid of numbers. The diagram also shows a 'source pixel' grid (a 5x5 grid of numbers) and a 'Convolution kernel (emboss)' (a 3x3 grid of numbers). The resulting output is a 3x3 grid of numbers. The diagram also shows a 'source pixel' grid (a 5x5 grid of numbers) and a 'Convolution kernel (emboss)' (a 3x3 grid of numbers). The resulting output is a 3x3 grid of numbers.</p>
<p>Recurrent neural network</p>	<p>Connections between nodes can be cyclical, which gives the network memory. Used for handwriting, speech recognition, time series.</p>	 <p>The diagram illustrates a Recurrent Neural Network (RNN) architecture. It shows an 'Input layer' with three red nodes, a 'Hidden layer' with two blue nodes, and an 'Output layer' with one green node. Arrows indicate the flow of information from the input layer to the hidden layer, and from the hidden layer to the output layer. A curved arrow indicates a recurrent connection between the hidden layer nodes, representing the network's memory.</p>

# Neural networks (ANNs)

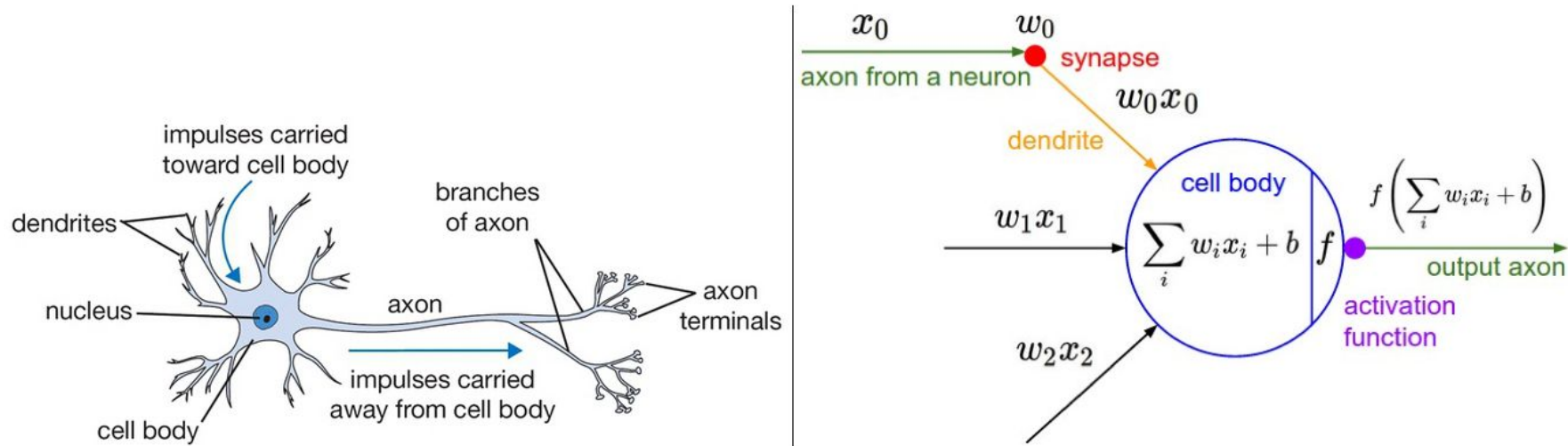
Paraphrased from Wikipedia:

*“Artificial neural networks are a family of models inspired by biological neural networks that estimate functions that can depend on a large number of inputs. Artificial neural networks are generally presented as systems of interconnected ‘neurons’ which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.*

*Like other machine learning methods – systems that learn from data – neural networks have been used to solve a wide variety of tasks, like computer vision and speech recognition, that are hard to solve using ordinary rule-based programming.”*

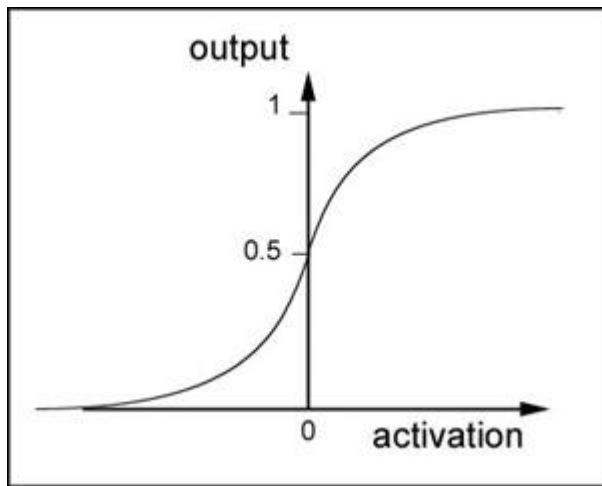
# How neural nets work

# The computational unit: neuron (neurode, node, unit)



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

# Sigmoid activation function, and its derivative



$$y = \frac{1}{1 + e^{-x}}$$

*quotient rule:*

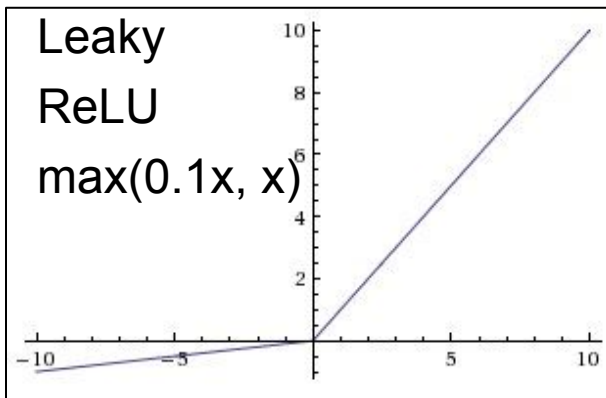
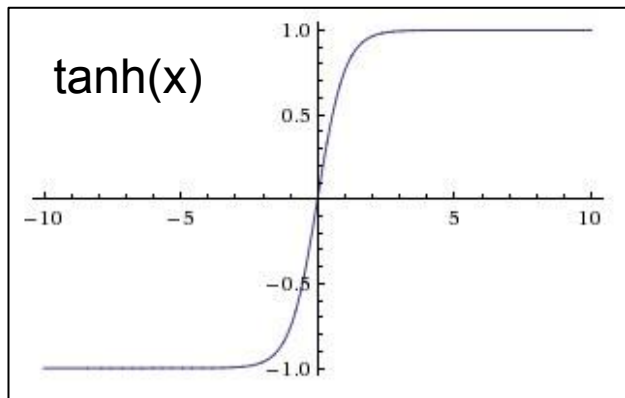
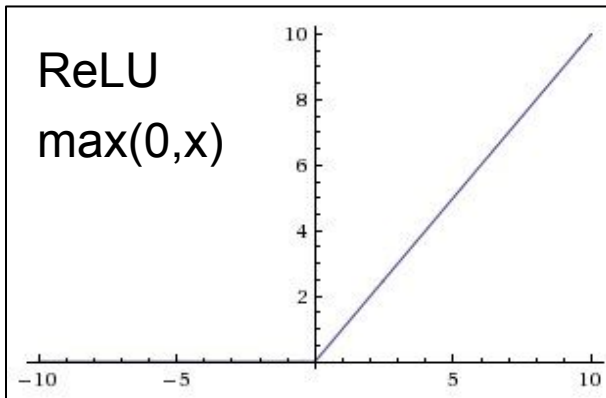
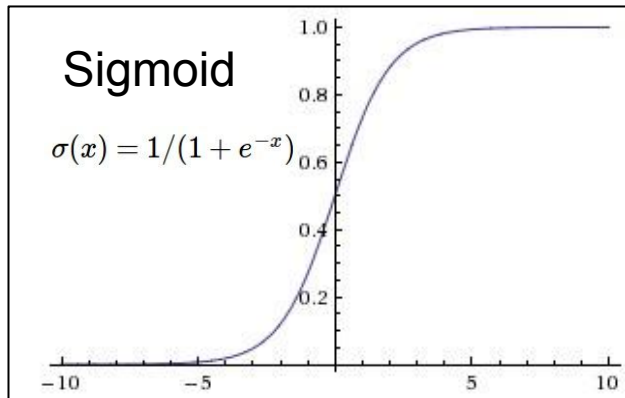
$$\text{derivate} \left( \frac{f}{g} \right) = \left( \frac{f'g - g'f}{g^2} \right)$$

$$\frac{dy}{dx} = \frac{0(1 + e^{-x}) - (-1 * e^{-x})}{(1 + e^{-x})^2}$$

$$\frac{dy}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = (1 - y)y$$



# Activation functions - a subset



Questions to consider when selecting:

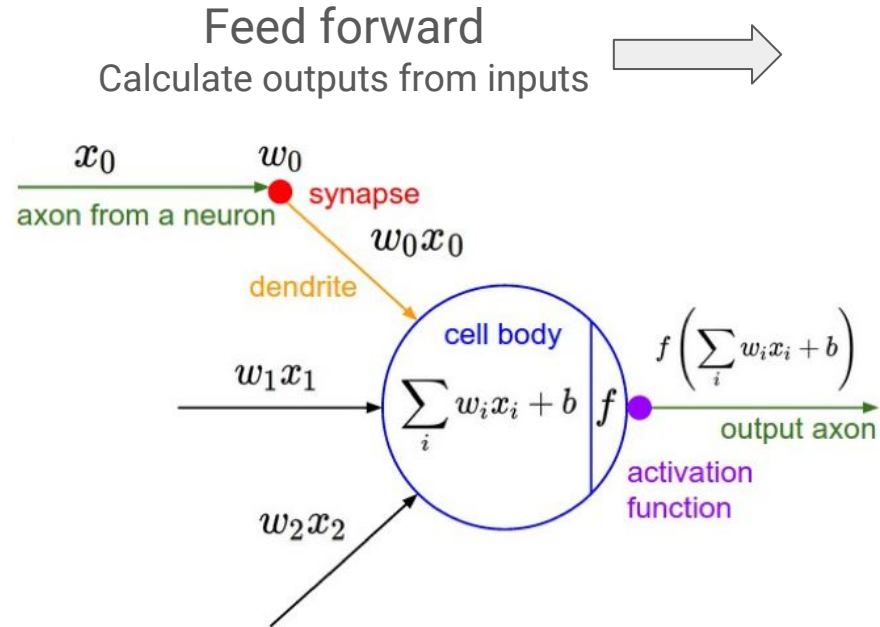
- 1) Benefit of zero mean?
- 2) Does derivative exist everywhere and does it give useful information?
- 3) Easy to compute?
- 4) Weight and bias initializations differ for these - take care!

Karpathy:

ReLU > Leaky ReLU > tanh  
(but be careful of ReLU learning rates)

and: **don't use sigmoid!**  
(exploding, vanishing gradient issues, slower to compute)

# Computations



Back propagation  
During **training**, change weights so that predicted output is closer to train output.

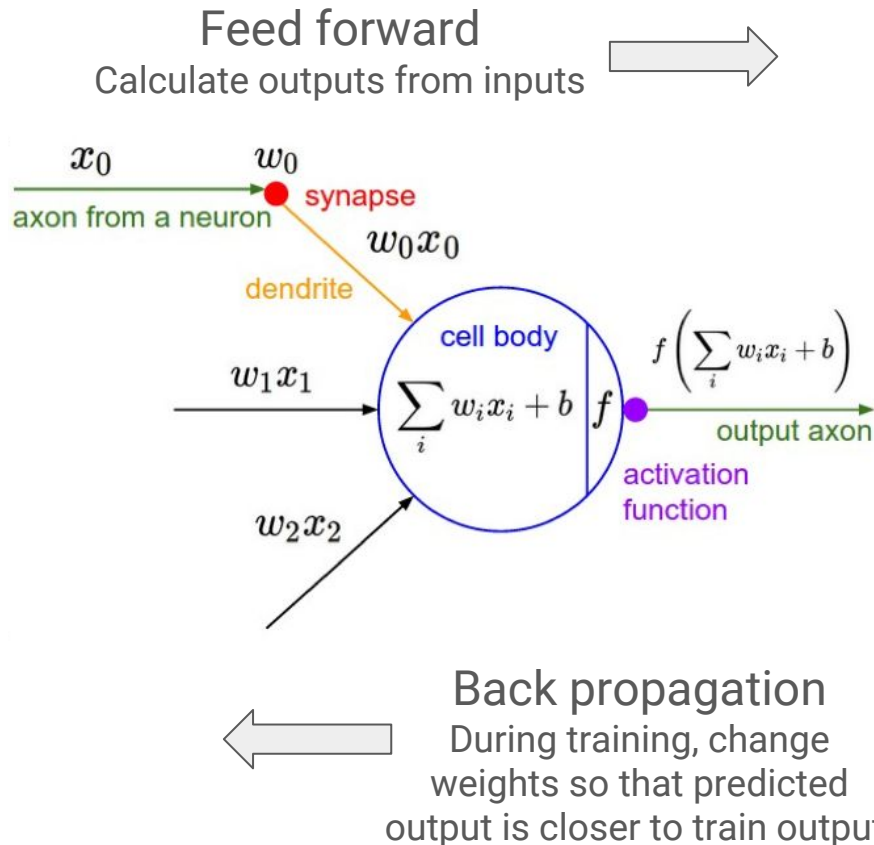
# Computations

**Goal:** Minimize the error or loss function - RSS (regression), misclassification rate (classification) - by changing the weights in the model.

**Back propagation**, the recursive application of the chain rule back along the computational network, allows the calculation of the local gradients, enabling....

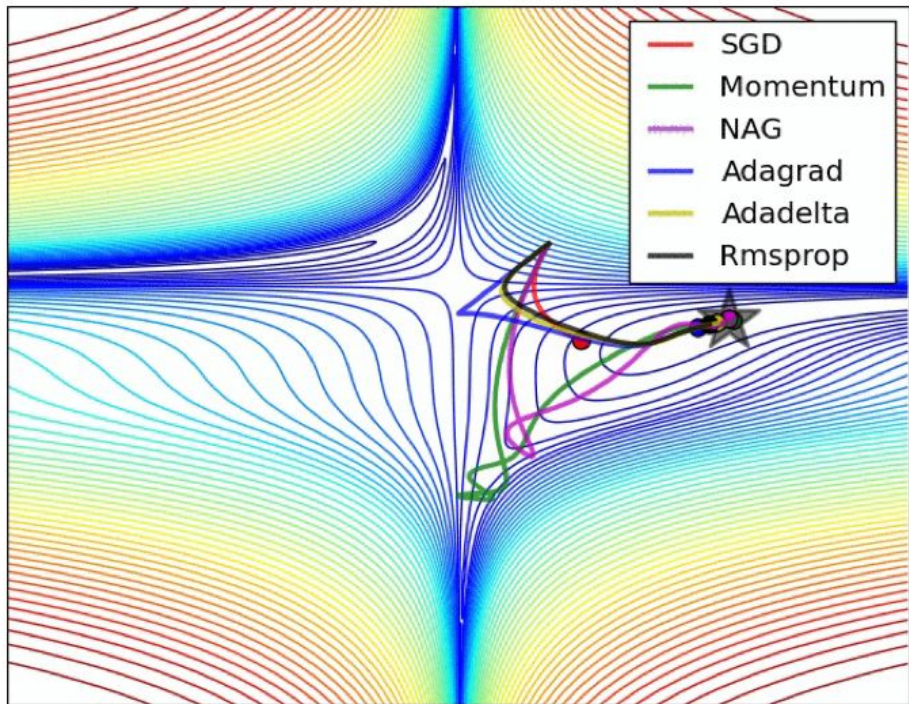
**Gradient descent** to be used to find the changes in the weights required to minimize the loss function.

**And you already did it in your homework!**



# Ok - gradient descent, but how?

Different gradient descent optimizers: <http://cs231n.github.io/neural-networks-3/>



See also:  
<https://keras.io/optimizers/>

# Batch, Mini-Batch, and SGD in pseudocode

In high-level pseudo-code, batch back-propagation is:

```
loop maxEpochs times
  for-each data item
    compute a gradient for each weight and bias
    accumulate gradient
  end-for
  use accumulated gradients to update each weight and bias
end-loop
```

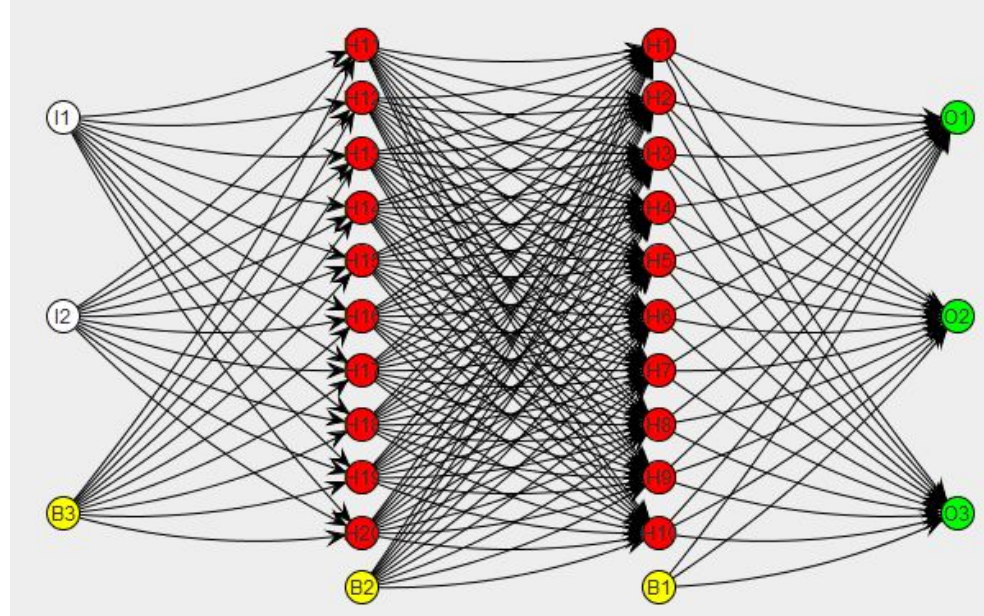
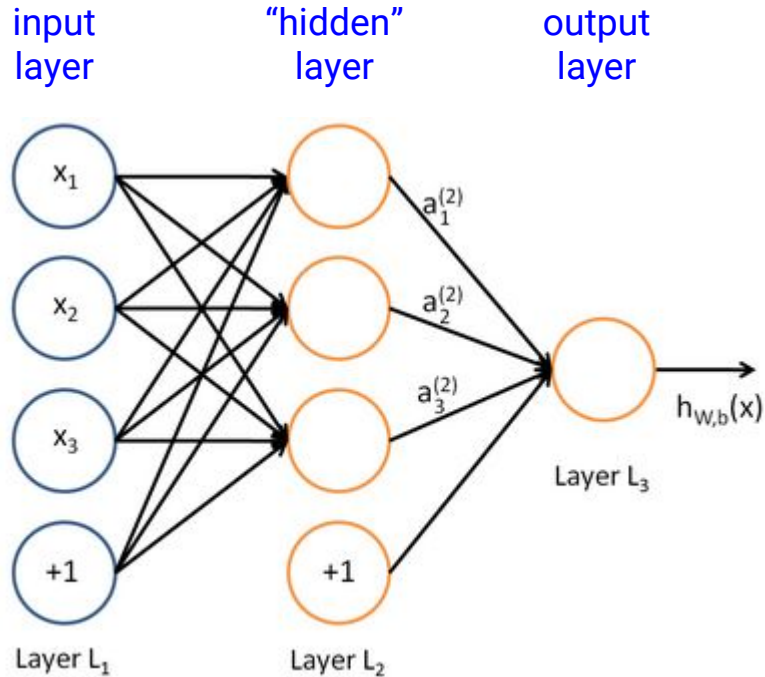
mini-batch

```
loop maxEpochs times
  loop until all data items used
    for-each batch of items
      compute a gradient for each weight and bias
      accumulate gradient
    end-batch
    use accumulated gradients to update each weight and bias
  end-loop all item
end-loop
```

In high-level pseudo-code, stochastic back-propagation is:

```
loop maxEpochs times
  for-each data item
    compute gradients for each weight and bias
    use gradients to update each weight and bias
  end-for
end-loop
```

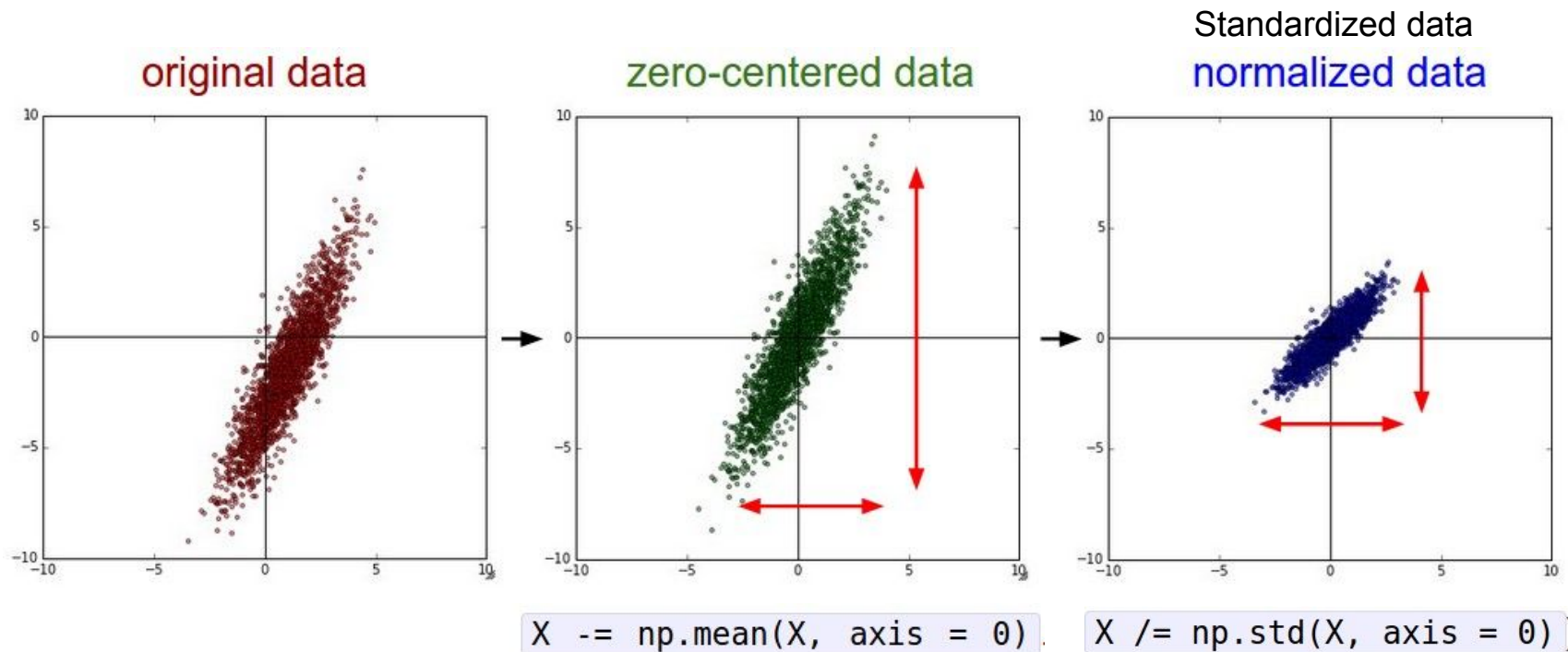
# Multilayer perceptron architecture



Training



# Pre-processing



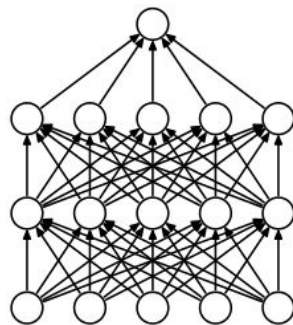
Karpathy advice for images: subtract the mean image only, can do it per channel



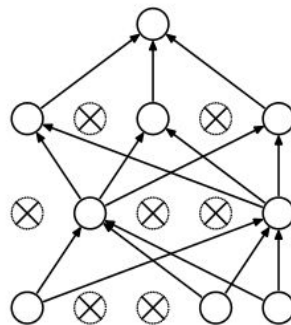
# Defining a model - available hyperparameters

- Most are particular to your application: read the literature and start with something that has been shown to work well.
- Structure: the number of hidden layers, the number of nodes in each layer
- Activation functions
- Weight and bias initialization (for weights Karpathy recommends Xavier init.)
- Training method: Loss function, learning rate, batch size, number of epochs
- Regularization: weight decay (loss function L1 and L2), dropout
- Random seed

How do you pick your hyperparameters?

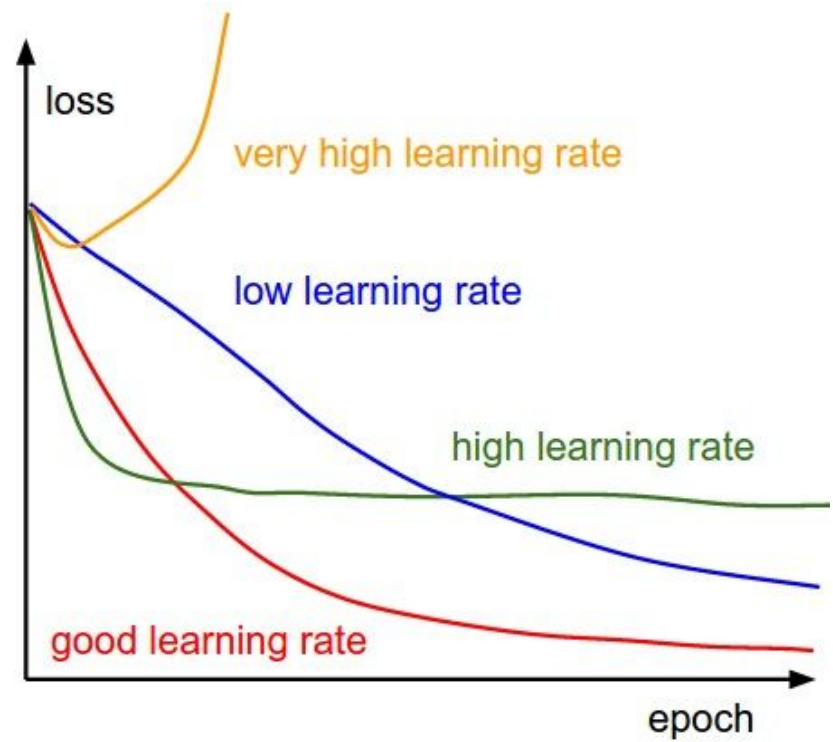
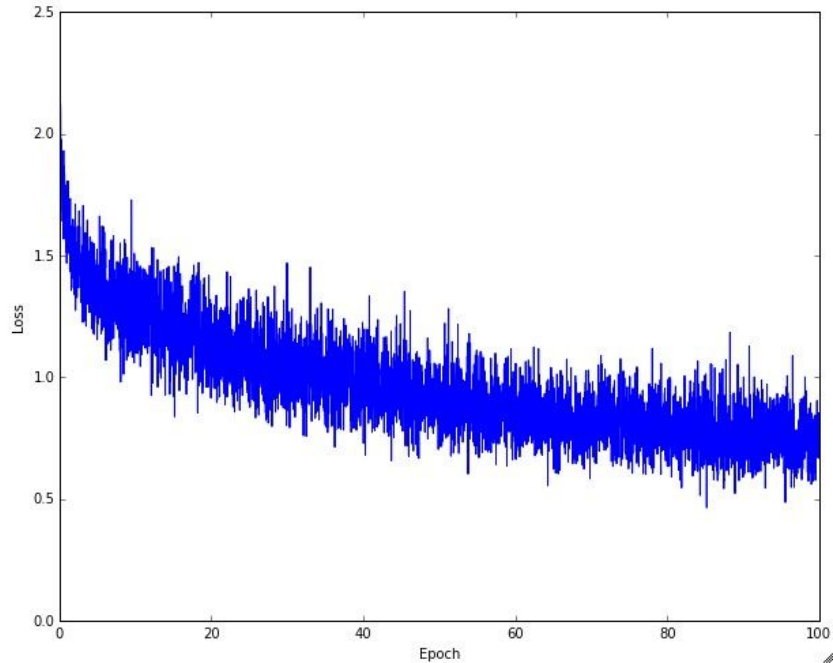


(a) Standard Neural Net



(b) After applying dropout.

# Monitor training process



# Potential problems

- Instability, difficulty visualizing the training process, interpretability
- The vanishing (or exploding) gradient problem during backpropagation:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

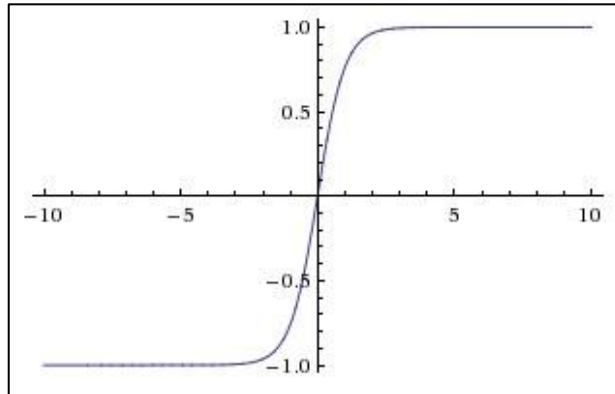
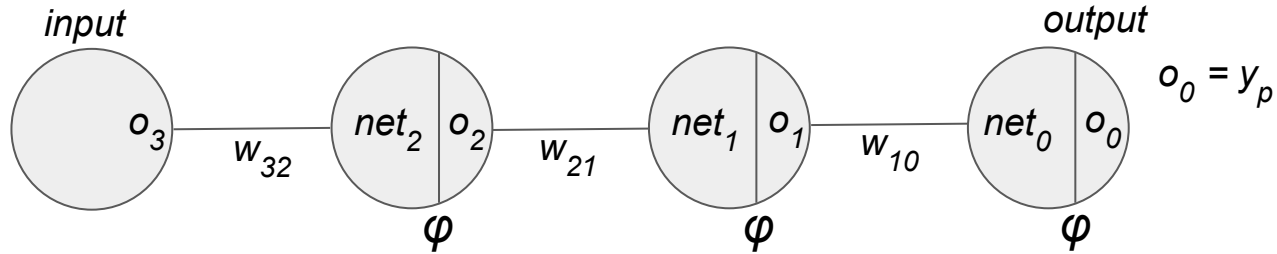
$$\Delta w_{ij} = -\alpha \left( \frac{\partial E}{\partial w_{ij}} \right)$$

$$\Delta w_{32} = -\alpha \left( \frac{\partial E}{\partial w_{32}} \right)$$

$$\frac{\partial E}{\partial w_{32}} \propto \varphi'(net_0) \cdot \varphi'(net_1) \cdot \varphi'(net_2)$$

---

The weights farthest away from the output can have very large, or very small, updates.



# Implementing neural network in Keras

# Keras

## Models

About Keras models

Sequential

Model (functional API)

## Layers

About Keras layers

Core Layers

Convolutional Layers

Recurrent Layers

Embedding Layers

Advanced Activations Layers

Normalization Layers

Noise layers

Layer wrappers

Writing your own Keras layers

## Preprocessing

Sequence Preprocessing

Text Preprocessing

Image Preprocessing

## Objectives

## Optimizers

## Activations



## Keras: Deep Learning library for Theano and TensorFlow

### You have just found Keras.

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either **TensorFlow** or **Theano**. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Use Keras if you need a deep learning library that:

- allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- supports both convolutional networks and recurrent networks, as well as combinations of the two.
- supports arbitrary connectivity schemes (including multi-input and multi-output training).
- runs seamlessly on CPU and GPU.

Read the documentation at [Keras](#)

Keras is compatible with: **Python**

### Examples

Here are a few examples to get you started!

In the examples folder, you will also find example models for real datasets:

- CIFAR10 small images classification: Convolutional Neural Network (CNN) with realtime data augmentation
- IMDB movie review sentiment classification: LSTM over sequences of words
- Reuters newswires topic classification: Multilayer Perceptron (MLP)
- MNIST handwritten digits classification: MLP & CNN
- Character-level text generation with LSTM

...and more.

Quick overview of example code...

# References

- <http://cs231n.stanford.edu/> - Andrej Karpathy class slides, lectures available on youtube
- <http://playground.tensorflow.org> Visualize neural network training online