DEEP LEARNING

A HUMBLE INTRODUCTION

Krishna Dusad
University of Illinois @ Urbana-Champaign

Contents

HISTORY & MOTIVATION

NEURAL NETS

APPLICATIONS

PROBLEMS

RESOURCES

What Is Machine Learning?

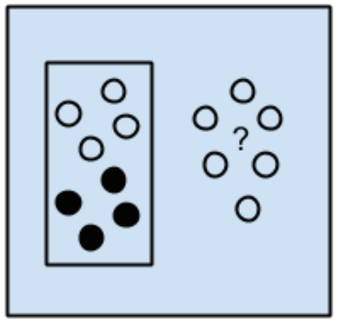
"Machine Learning is the subfield that gives computers the ability to learn without being explicitly programmed"

ML explores techniques that can learn from and make predictions on data – making data-driven predictions or decisions, through building a model from sample inputs

Machine-learning technology powers many aspects of modern society: web searches, content filtering on social networks, recommendations on e-commerce websites…

Supervised

Input data is has a known label and a model required to make predictions on unseen data.



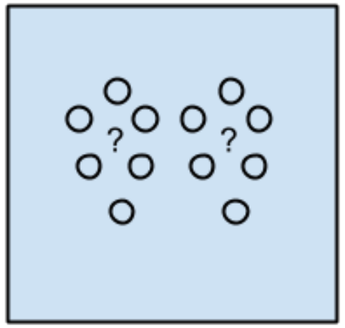
Supervised Learning Algorithms

Ex. Problems: classification and regression

Ex. Algorithms: Linear Regression, **SVM**

Unsupervised

Input data is not labeled and does not have a known result.



Unsupervised Learning Algorithms

Ex. Problems: clustering and dimensionality reduction

Ex. Algorithms: EM, LDA, k-Means, PCA

Big Problem

"Conventional machine-learning techniques were limited in their ability to process natural data in their raw form.

For decades, constructing a machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input."

bad features —> bad representations —> bad model

So, how was it solved?

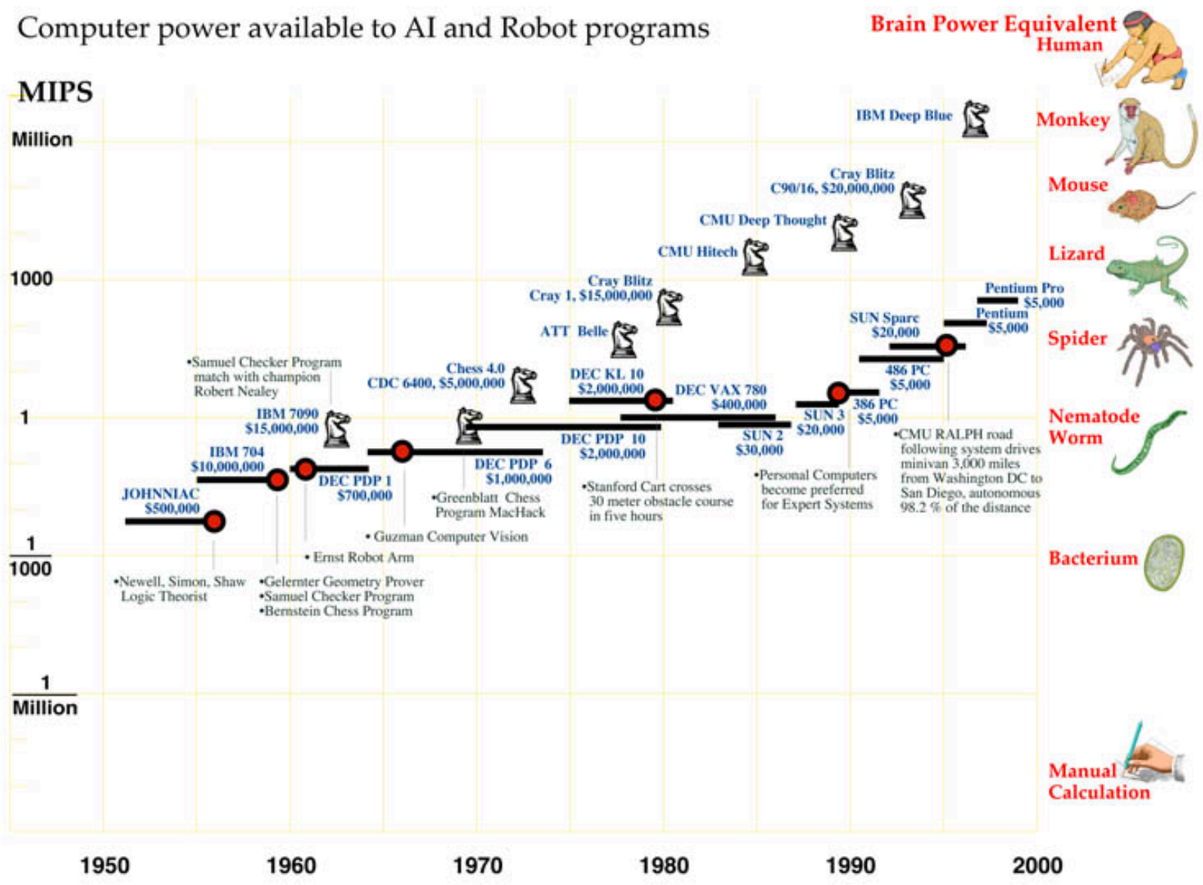
Before we look at how it was solved, let's consider what things were needed before we could do anything meaningful.

1. *Compute*

2. *Natural Data (Lots)*

3. *Algorithms*

Compute


Computer power available to AI and Robot programs
MIPS
Million
1000
1
1/1000
1/Million
1950
1960
1970
1980
1990
2000
Brain Power Equivalent
Human
Monkey
Mouse
Lizard
Spider
Nematode Worm
Bacterium
Manual Calculation
JOHNNIAC $500,000
IBM 704 $10,000,000
DEC PDP 1 $700,000
IBM 7090 $15,000,000
CDC 6400, $5,000,000
Chess 4.0
DEC PDP 6 $1,000,000
DEC PDP 10 $2,000,000
DEC KL 10 $2,000,000
ATT Belle
Cray 1, $15,000,000
Cray Blitz
CMU Hitech
CMU Deep Thought
DEC VAX 780 $400,000
Cray Blitz C90/16, $20,000,000
IBM Deep Blue
SUN 2 $30,000
SUN 3 $20,000
386 PC $5,000
486 PC $5,000
SUN Sparc $20,000
Pentium $5,000
Pentium Pro $5,000
•Newell, Simon, Shaw Logic Theorist
•Gelernter Geometry Prover
•Samuel Checker Program
•Bernstein Chess Program
•Samuel Checker Program match with champion Robert Nealey
• Ernst Robot Arm
• Guzman Computer Vision
•Greenblatt Chess Program MacHack
•Stanford Cart crosses 30 meter obstacle course in five hours
•Personal Computers become preferred for Expert Systems
•CMU RALPH road following system drives minivan 3,000 miles from Washington DC to San Diego, autonomous 98.2 % of the distance

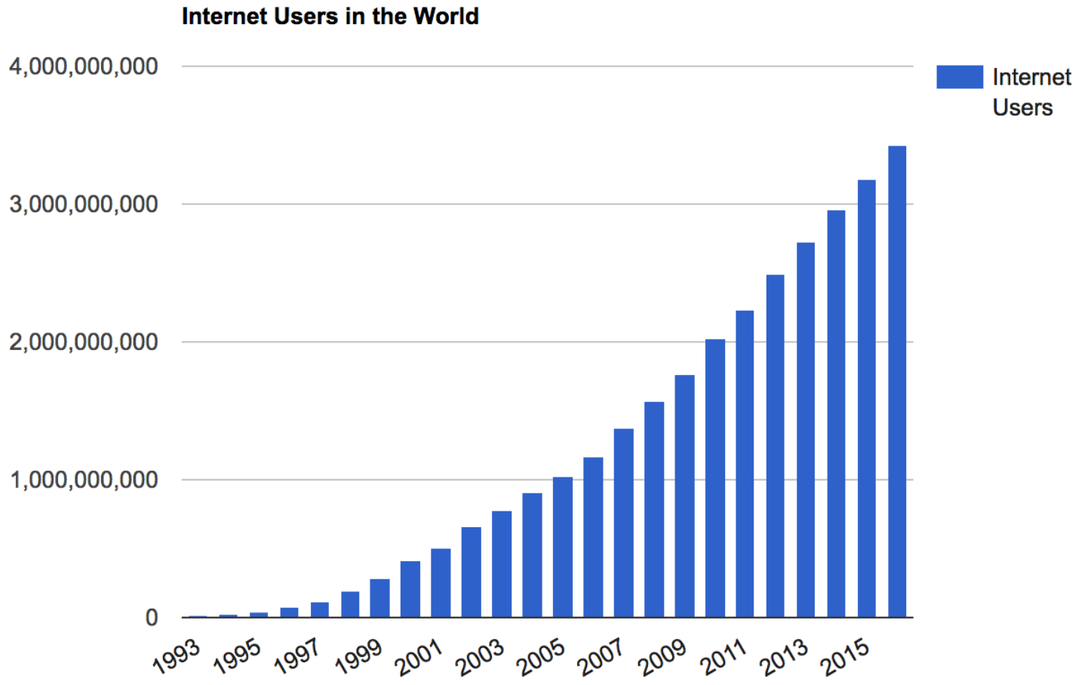
Data

Around **40%** of the world population has an internet connection today.

In 1995, it was less than **1%**.

The number of internet users has increased **tenfold** from 1999 to 2013.

The first billion was reached in 2005. The second billion in 2010. The **third billion** in 2014.



Data

Internet users in the world

3,610,454,579

All on this page, one by one

watch as they increase

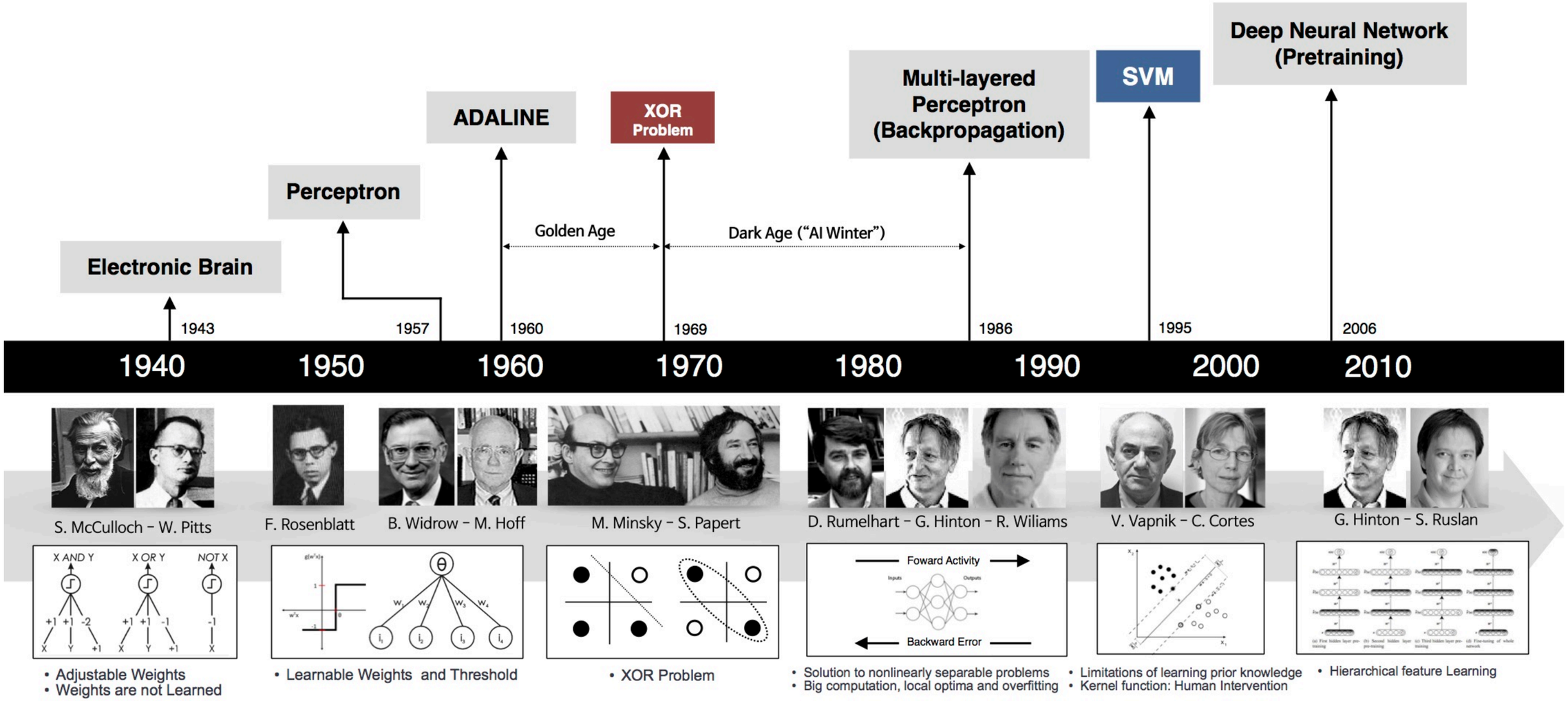
Algorithms

Neural Networks

Probabilistic Graphical Models

Not Much Progress Needed!

Brief Hist of NNs

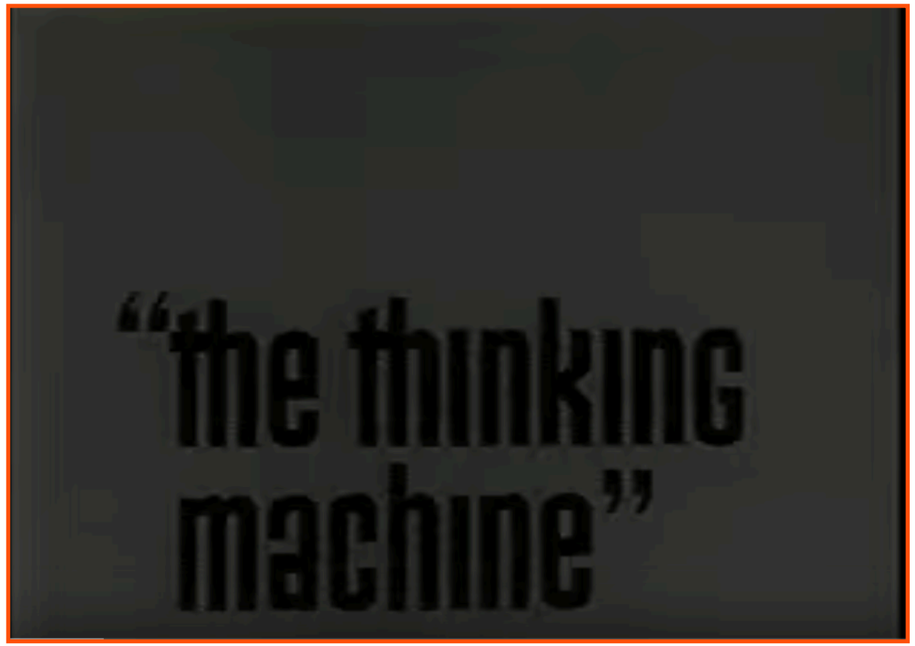


1940-1960s

1. Walter Pitts and Warren McCulloch developed a “thresholded logic unit” and was informed by the current working theories of the brain

Model that sums binary inputs and outputs a 1 if the sum exceeds a certain threshold value, and otherwise outputs a 0; can model the basic OR/AND/NOT functions.

Lacked a mechanism for learning, which was crucial for it to be usable for AI



Link

1940-1960s

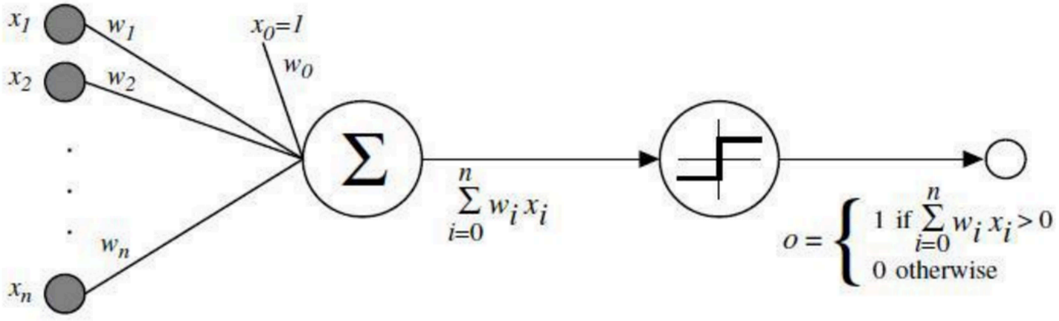
2. Frank Rosenblatt's "perceptron" was the first real precursor to NNs. It came with a learning procedure that would provably converge to the correct solution and could recognize letters and numbers.

NYT even remarked:

"[The perceptron is] the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

However...

1. Start off with a Perceptron having random weights and a training set
2. For the inputs of an example in the training set, compute the Perceptron's output
3. If the output of the Perceptron does not match the output that is known to be correct for the example:
   - If the output should have been 0 but was 1, decrease the weights that had an input of 1.
   - If the output should have been 1 but was 0, increase the weights that had an input of 1.
4. Go to the next example in the training set and repeat steps 2-4 until the Perceptron makes no more mistakes



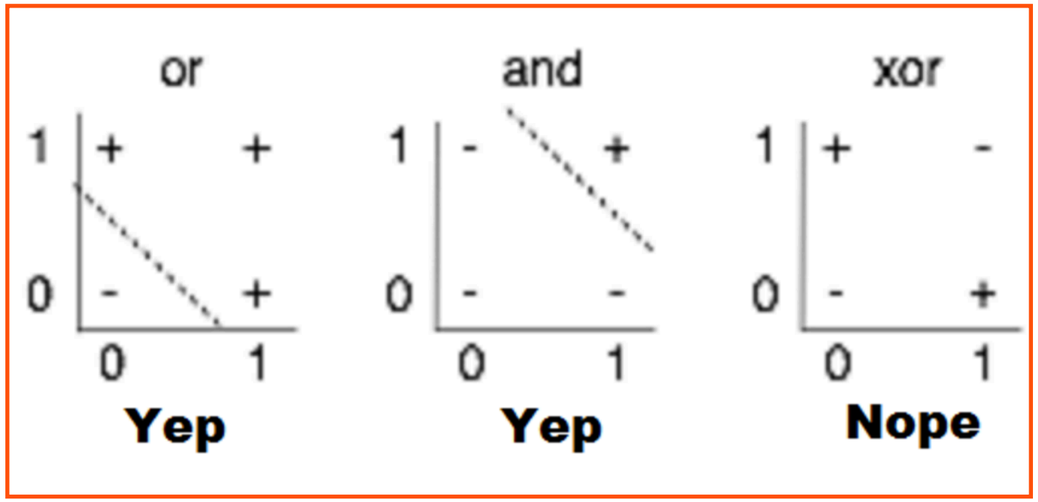
A diagram showing how the Perceptron works. (Source)

1960-1990s

3. Marvin Minsky, founder of the MIT AI Lab, were skeptical of the hype and published a rigorous analysis on of the limitations of Perceptrons in a seminal book aptly named Perceptrons

The most discussed element of this analysis was that Perceptrons for instance, could not learn the simple boolean function **XOR** because it was not **linearly separable**

This is widely believed to have helped usher in the first of the **AI Winters** - a period following massive hype for AI characterized by disillusionment that causes a freeze to funding.

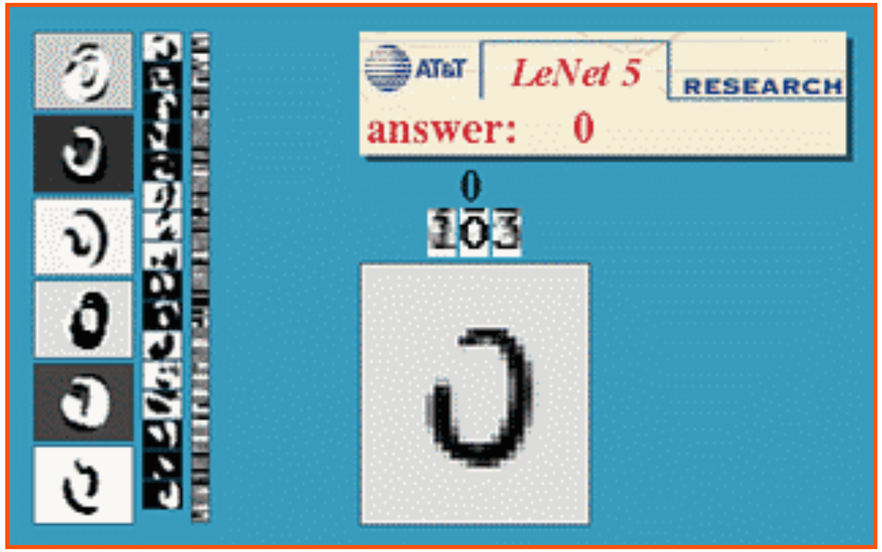


1990-2005

3. Backpropogation comes to practice. (Paul Werbos, Hinton, LeCun).

Support vector machines (Vapnik) and other, much simpler methods such as linear classifiers gradually overtook neural networks in machine learning popularity. Probabilistic graphical models also start gaining some popularity.

Deep Networks like ConvNets (LeCun), RNNs, LSTMs (Schmidhuber) proposed and actually show promise.



2006-2012

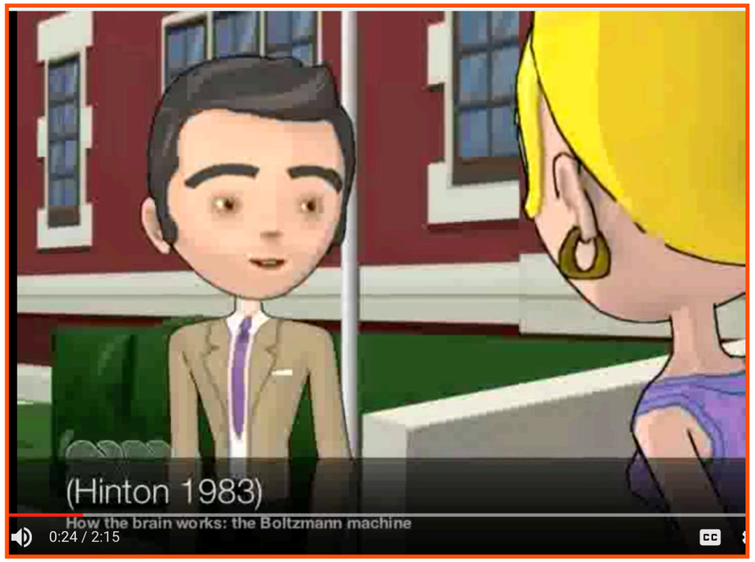
4. Neural Networks 2.0 a.k.a Deep Learning.

Improvements in GPU computing, clever tricks for optimization, regularization, and big datasets (challenges) start forming.

In 2010, ImageNet challenge with millions of high-res labeled images published by Stanford.

ML communities start getting excited about Neural Nets.

Andrew Ng starts 'Brain' team at Google with Jeff Dean._



The Deep Learning Saga

2012-*

4. AlexNet by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton entered a submission that halves the existing error rate to 16%.

Clever tricks such as dropout and batch-norm, and GPU usage suddenly ubiquitous. ReLUs and other activation functions come up. NNs now the most popular models in a variety of fields from vision, NLP, bioinformatics, finance, and even design.

All major companies, Facebook, Microsoft, Google, Pinterest, Twitter, Yahoo open AI Labs. YCombinator creates OpenAI.

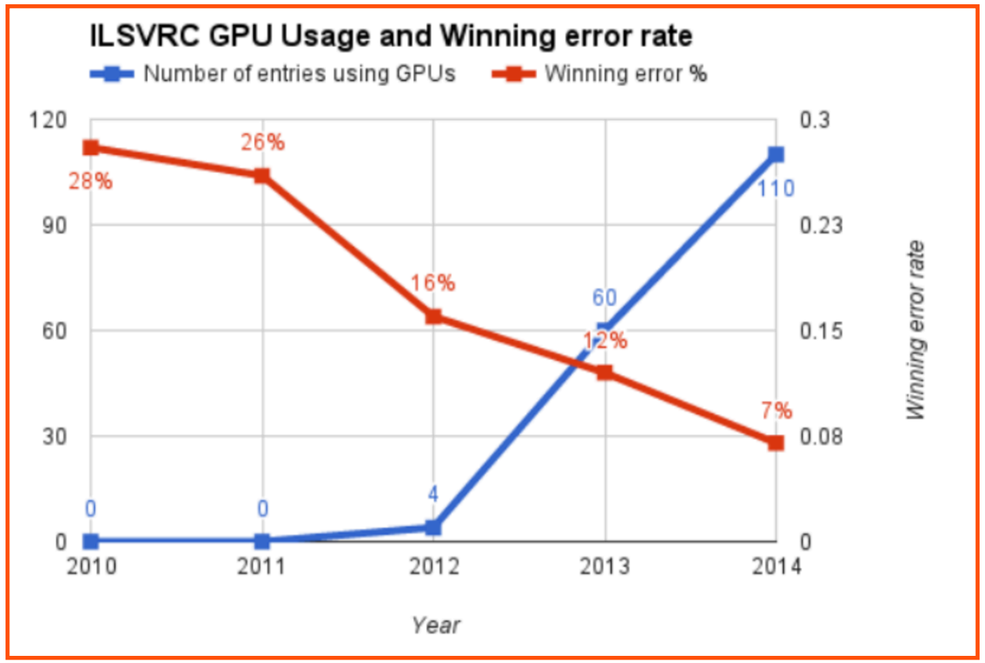
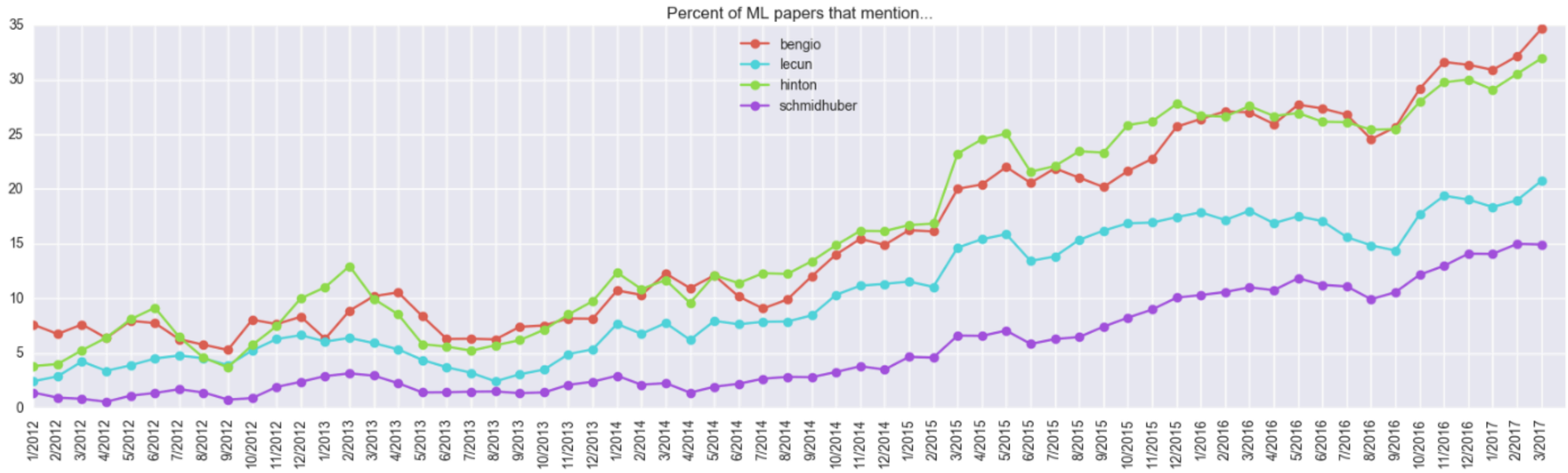
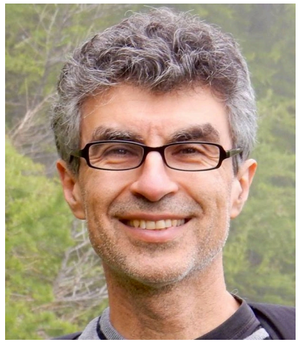


Image Net

2012-*


Percent of ML papers that mention...
bengio
lecun
hinton
schmidhuber
35
30
25
20
15
10
5
0
1/2012
2/2012
3/2012
4/2012
5/2012
6/2012
7/2012
8/2012
9/2012
10/2012
11/2012
12/2012
1/2013
2/2013
3/2013
4/2013
5/2013
6/2013
7/2013
8/2013
9/2013
10/2013
11/2013
12/2013
1/2014
2/2014
3/2014
4/2014
5/2014
6/2014
7/2014
8/2014
9/2014
10/2014
11/2014
12/2014
1/2015
2/2015
3/2015
4/2015
5/2015
6/2015
7/2015
8/2015
9/2015
10/2015
11/2015
12/2015
1/2016
2/2016
3/2016
4/2016
5/2016
6/2016
7/2016
8/2016
9/2016
10/2016
11/2016
12/2016
1/2017
2/2017
3/2017



NEURAL NETS

MATH

TRICKS

INTUITION

PROBLEMS

NOT this one.

Linear SVMs

Steps:

1. Goal: learn weights W to minimize 'loss' on misclassification.

2. Loss = Hinged Loss

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

3. Minimize some regularized version of L (using SGD), use the learned W as weights.

10	cat score
14	dog score
-12	car score
-2	horse score

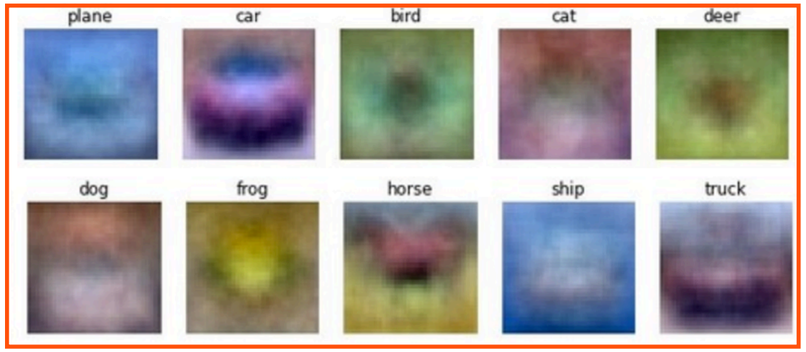
f (W, **x**)

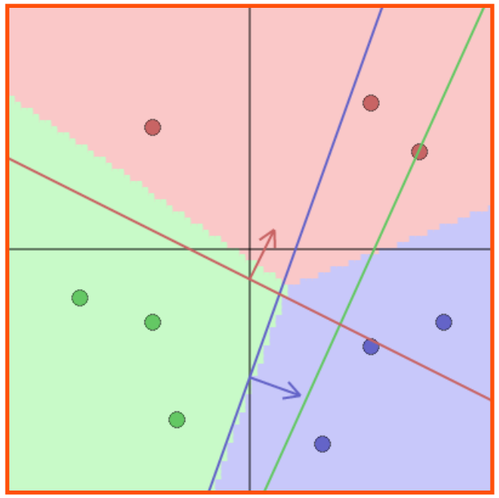
Linear SVMs

Note:

1. Can be thought of as finding widest street of separation.

2. ***Can be thought of as template matching.***





Gradients

We left our SVM at a point where the only step left was to find the value of W that minimizes L.

One technique to do that is gradient descent.

Gradients are slopes. They indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point.

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n,$$

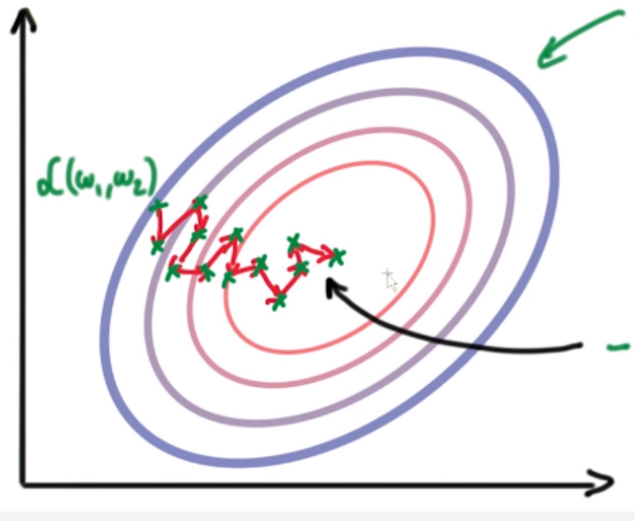
Stochastic Gradient Descent

True gradient descent is hard to perform.

Since the data is huge, other techniques like Newton's method are also not feasible

So, we stick to a stochastic approach which usually works.

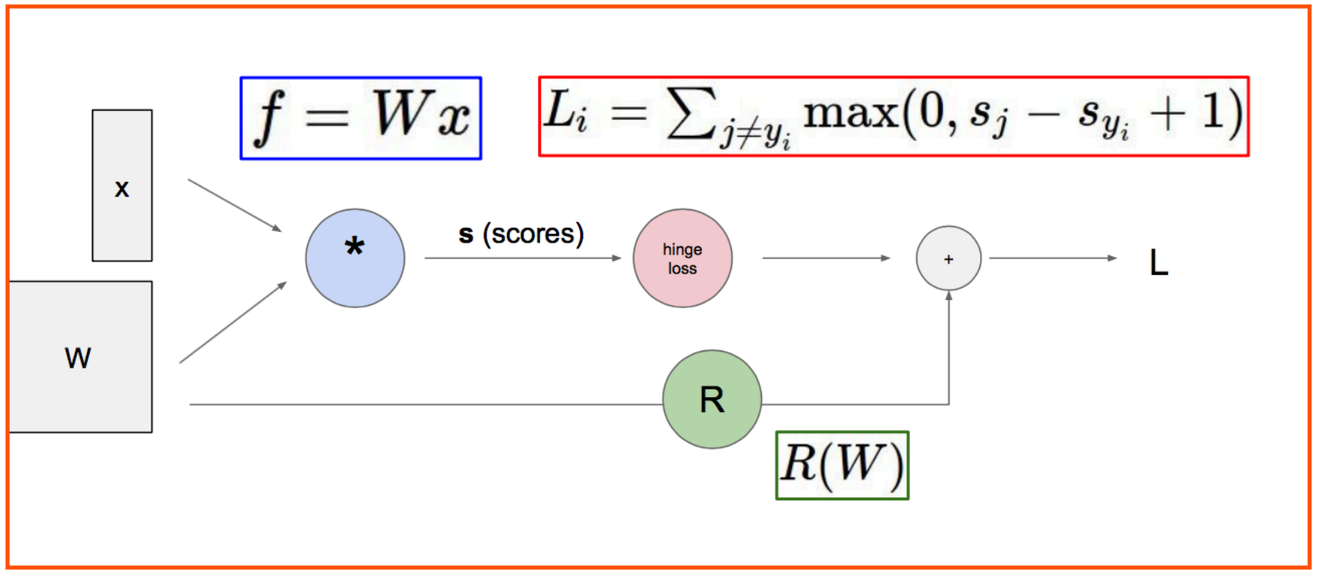
- Choose an initial vector of parameters w and learning rate η.
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta \nabla Q_i(w)$.



$$w := w - \eta \nabla Q_i(w)$$

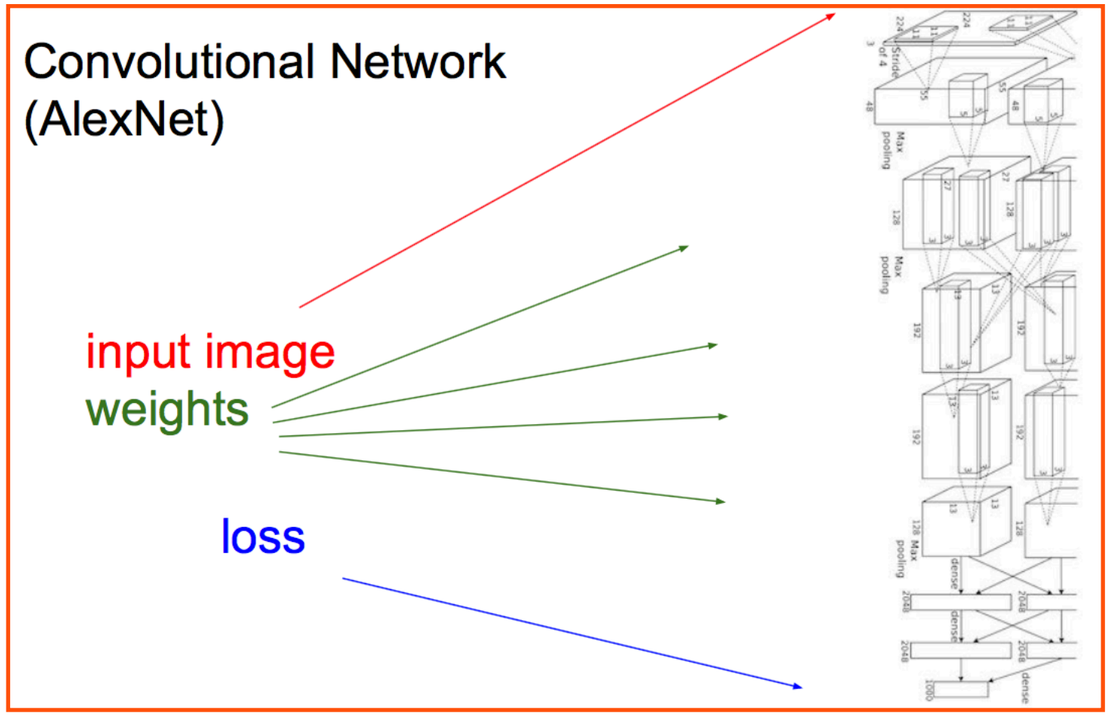
Computation Graphs

SVM



Computation Graphs

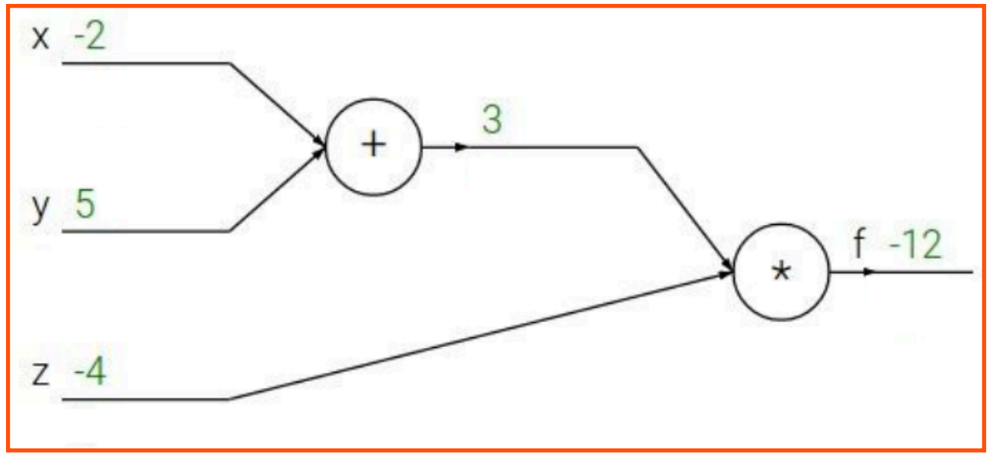
AlexNet



Chain Rule

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Chain Rule

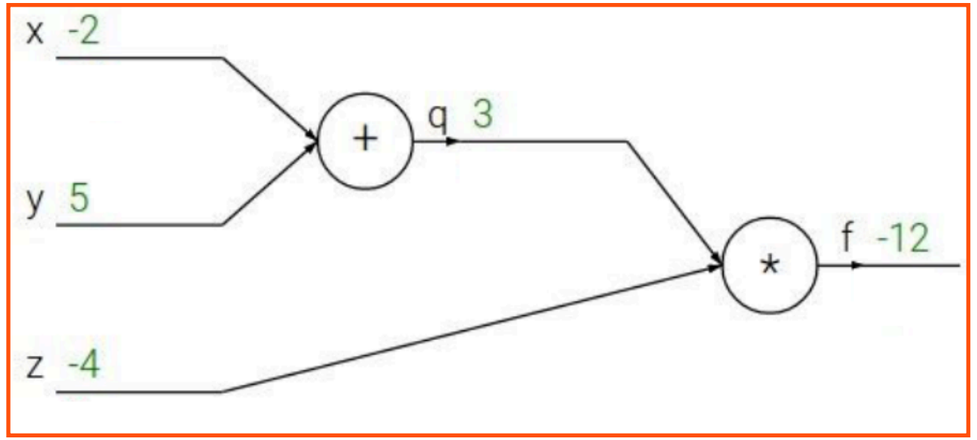
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain Rule

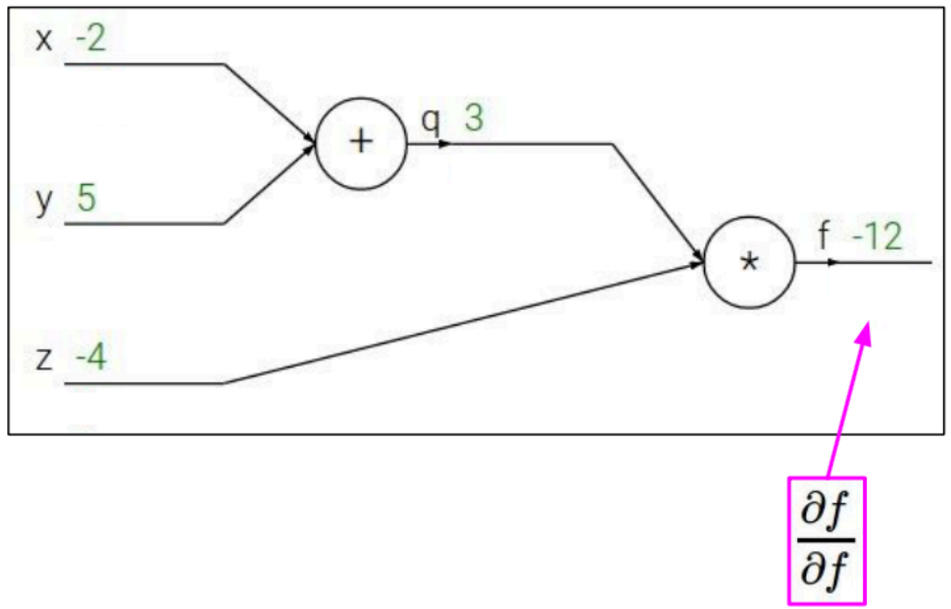
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain Rule

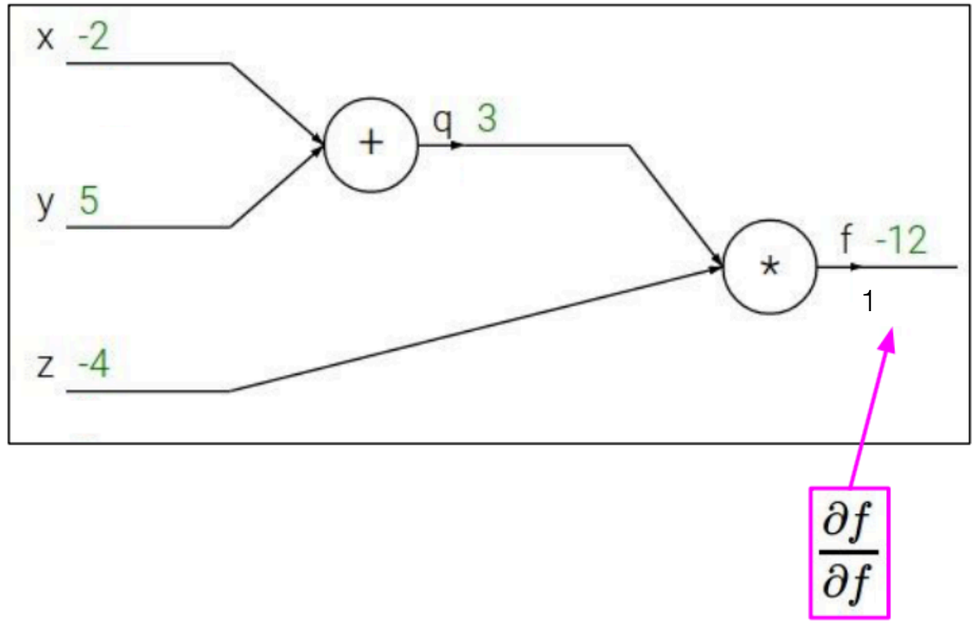
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain Rule

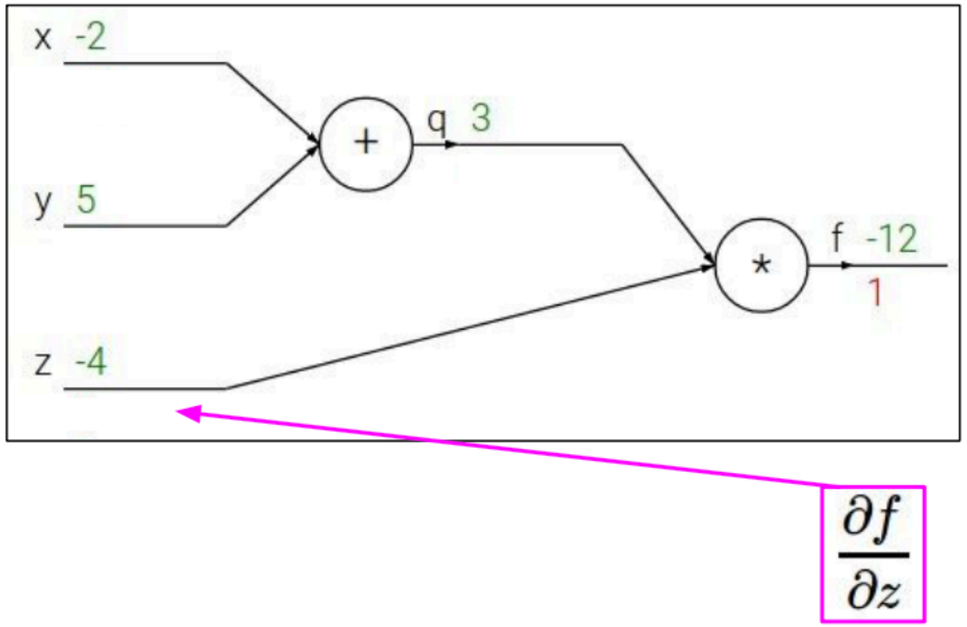
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain Rule

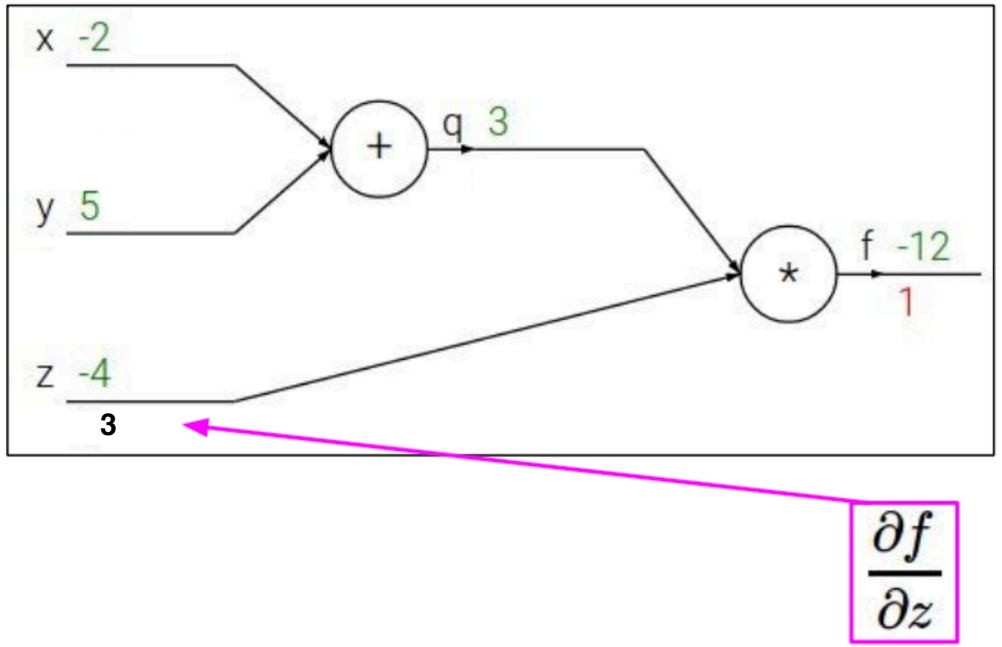
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial z}$$

Chain Rule

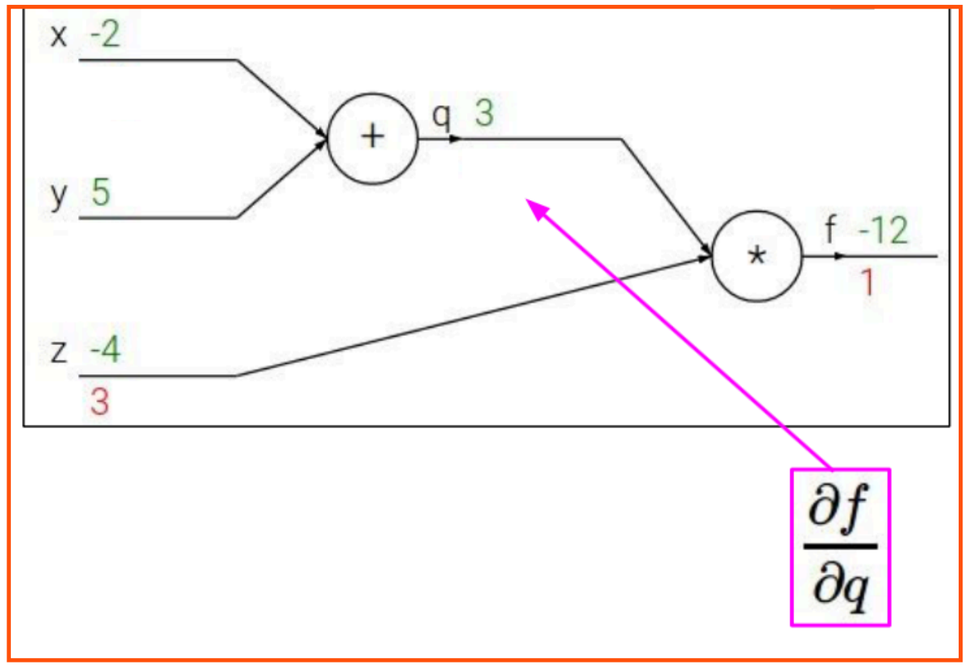
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain Rule

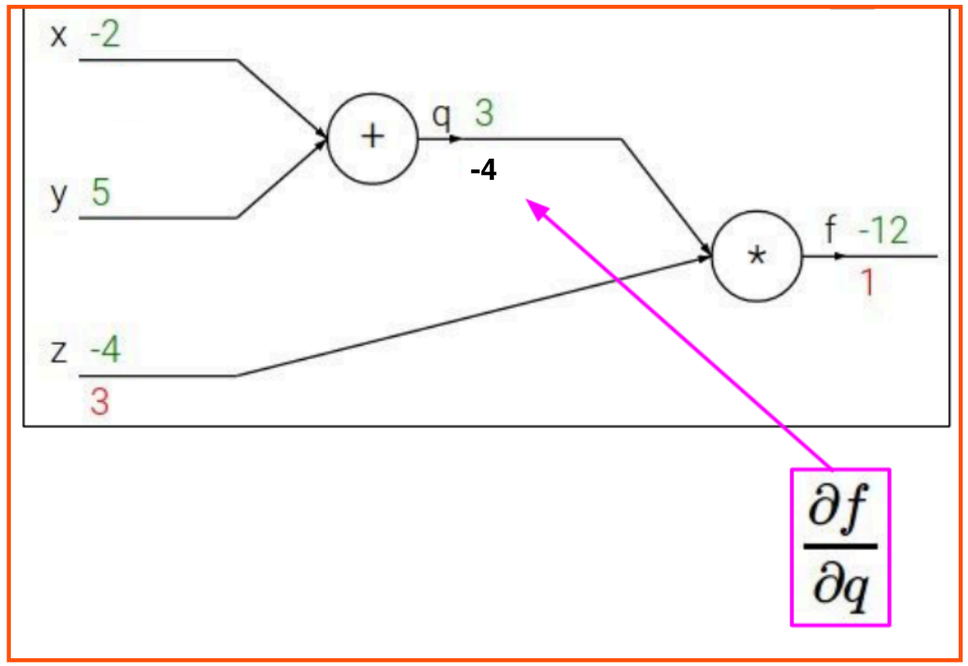
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain Rule

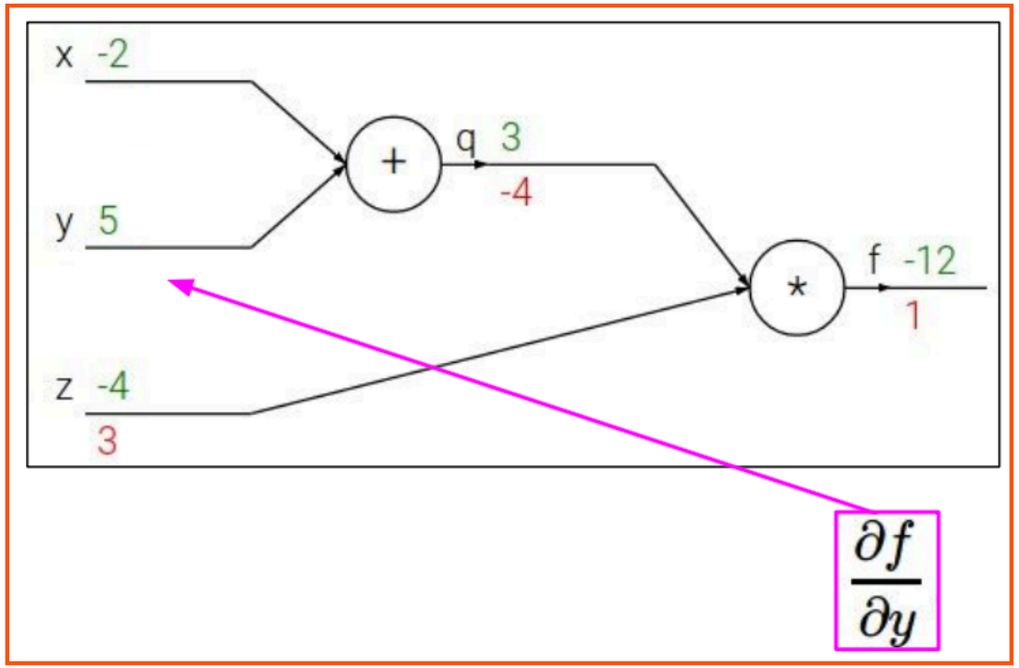
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain Rule

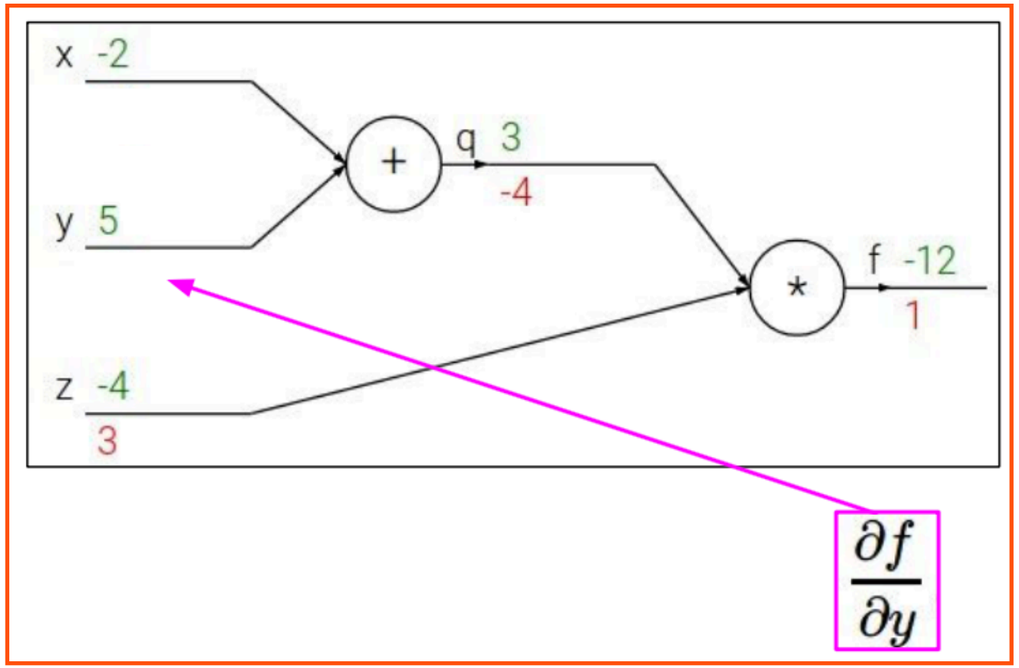
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain Rule

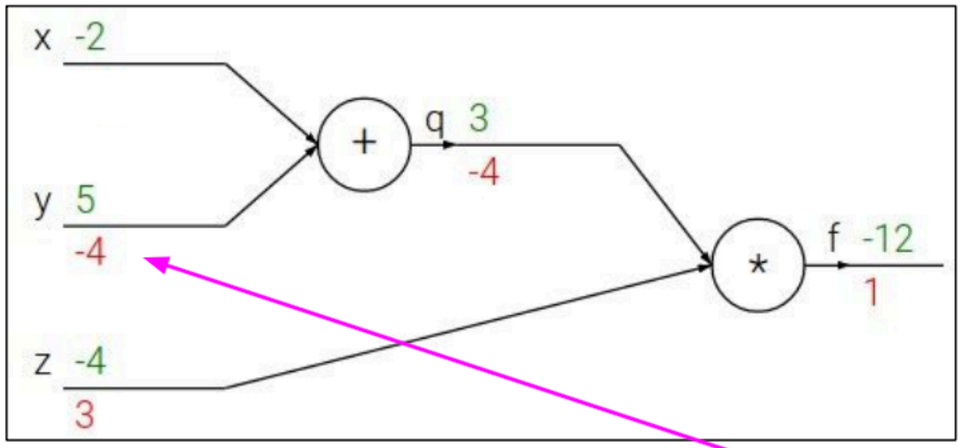
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Chain Rule = Backpropogation

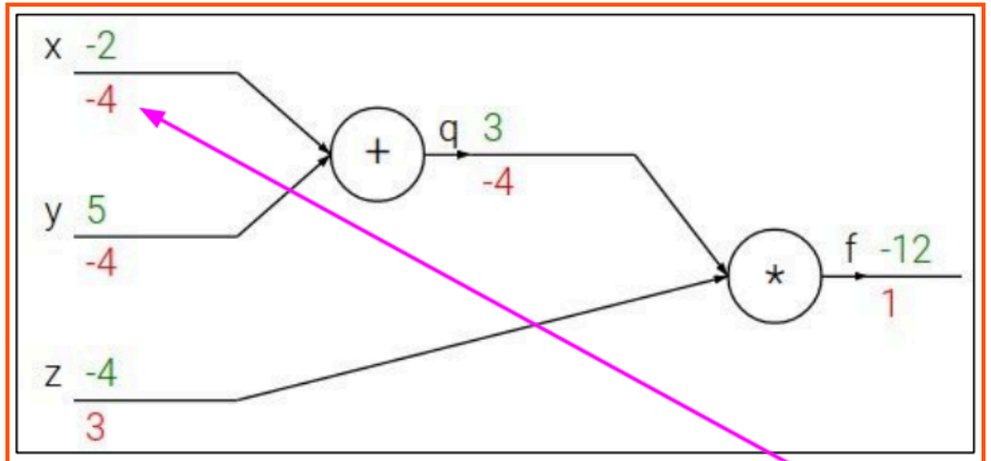
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

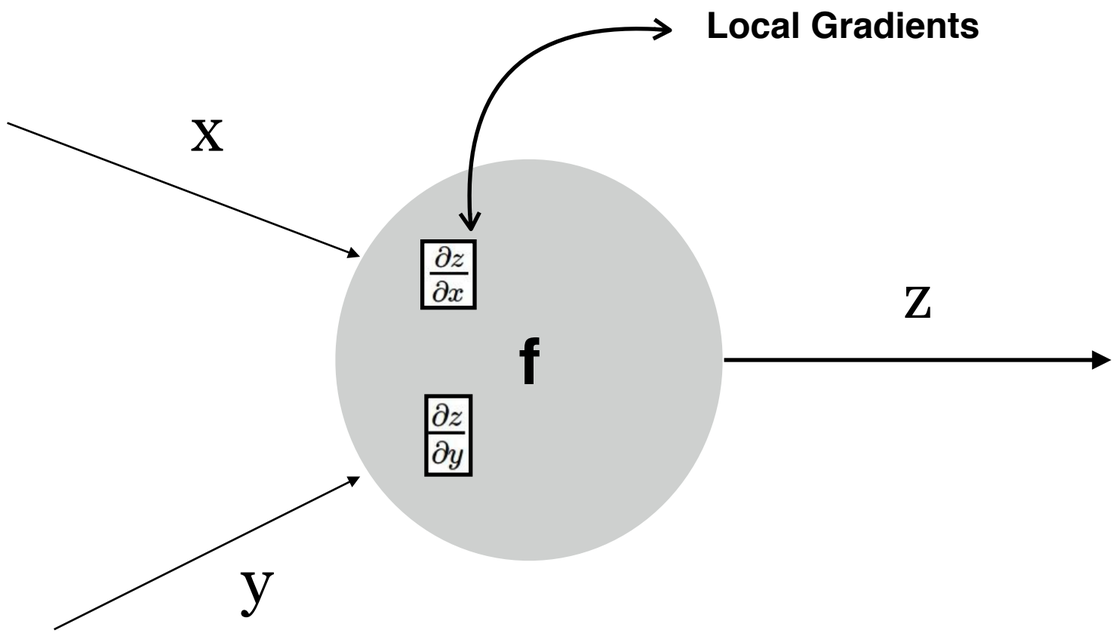


$$\frac{\partial f}{\partial x}$$

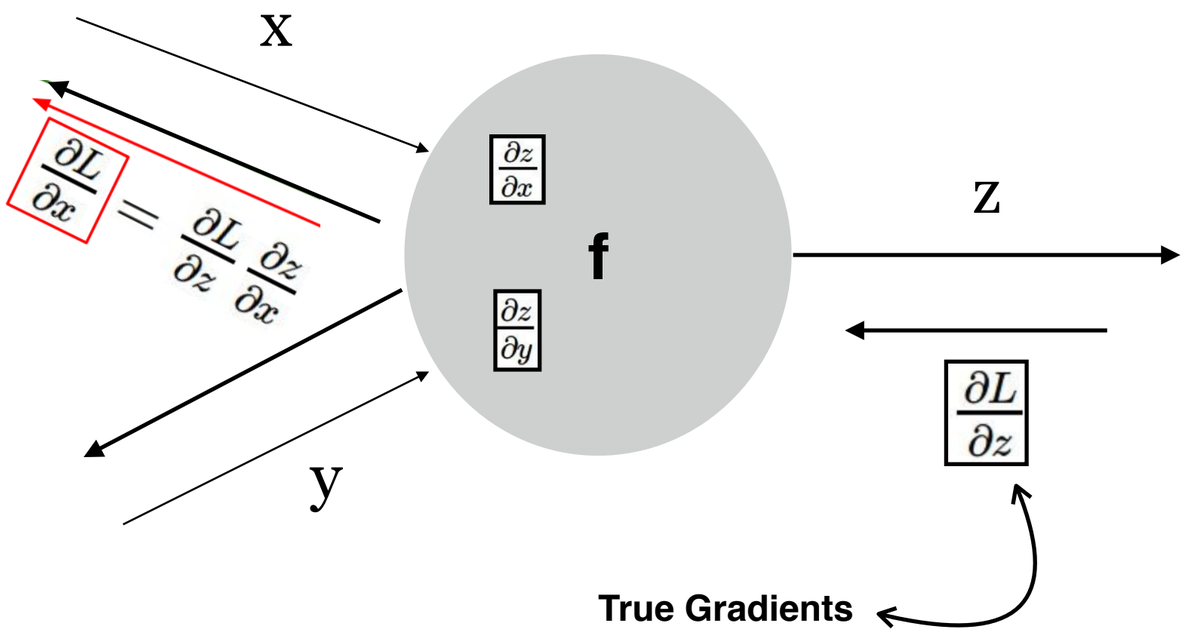
Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

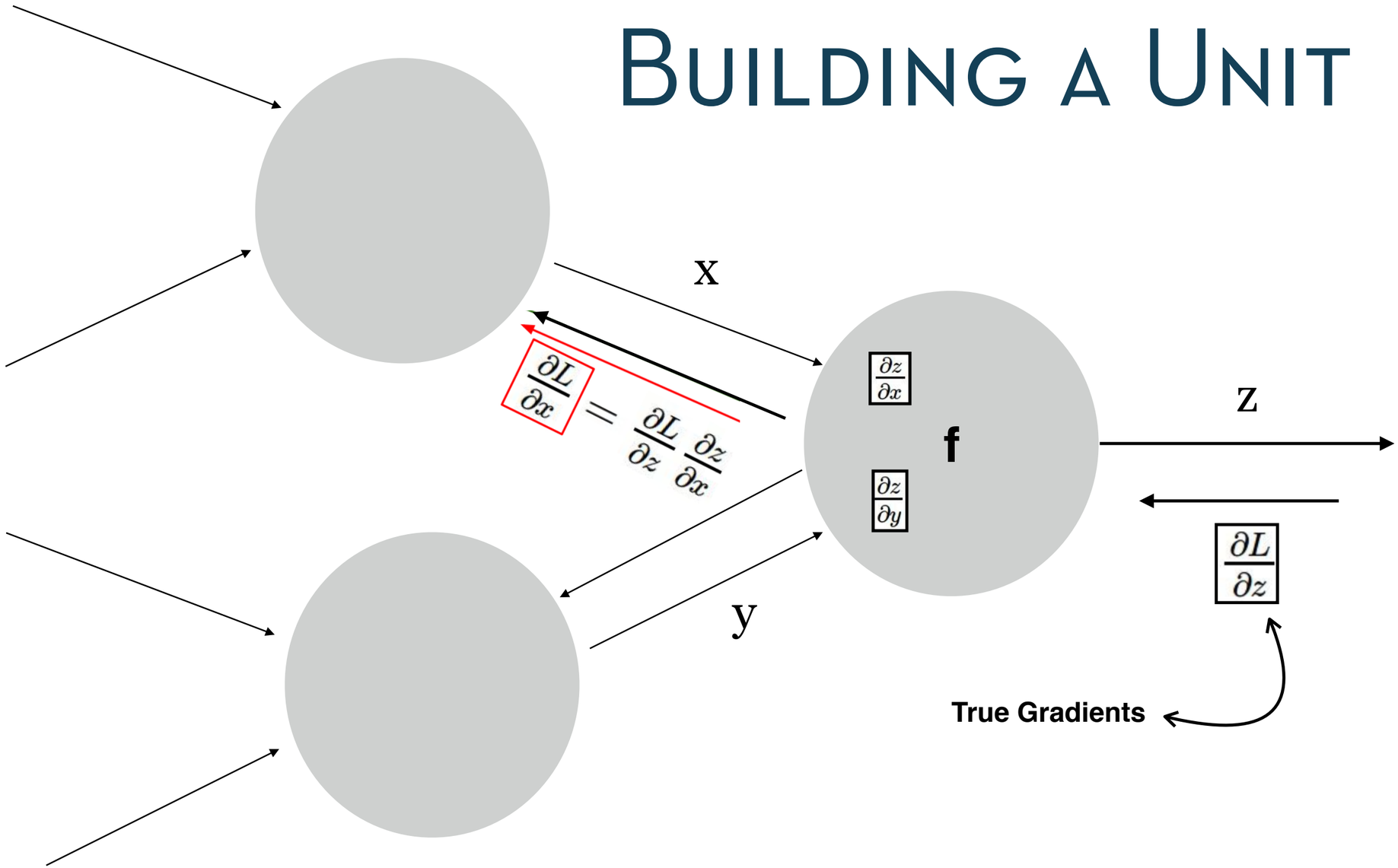
Building a Unit



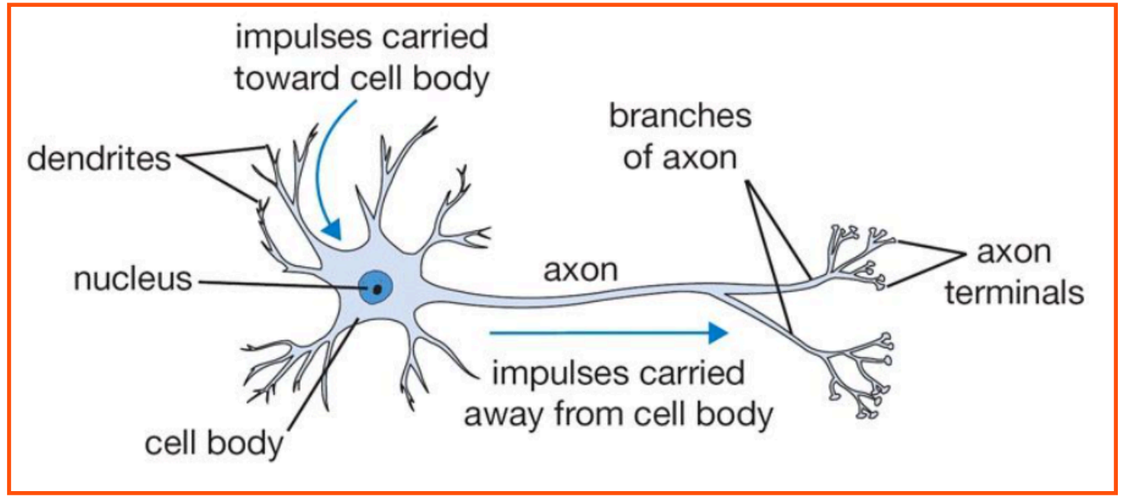
Building a Unit



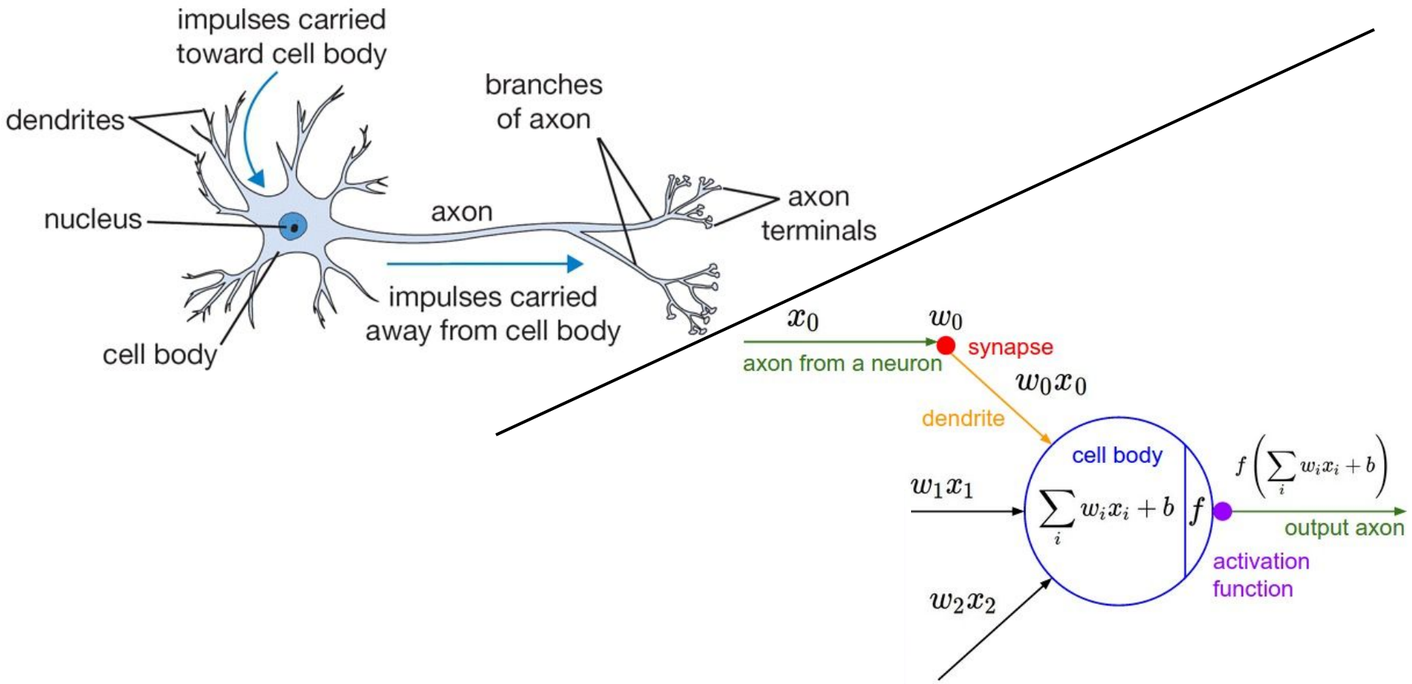
Building a Unit



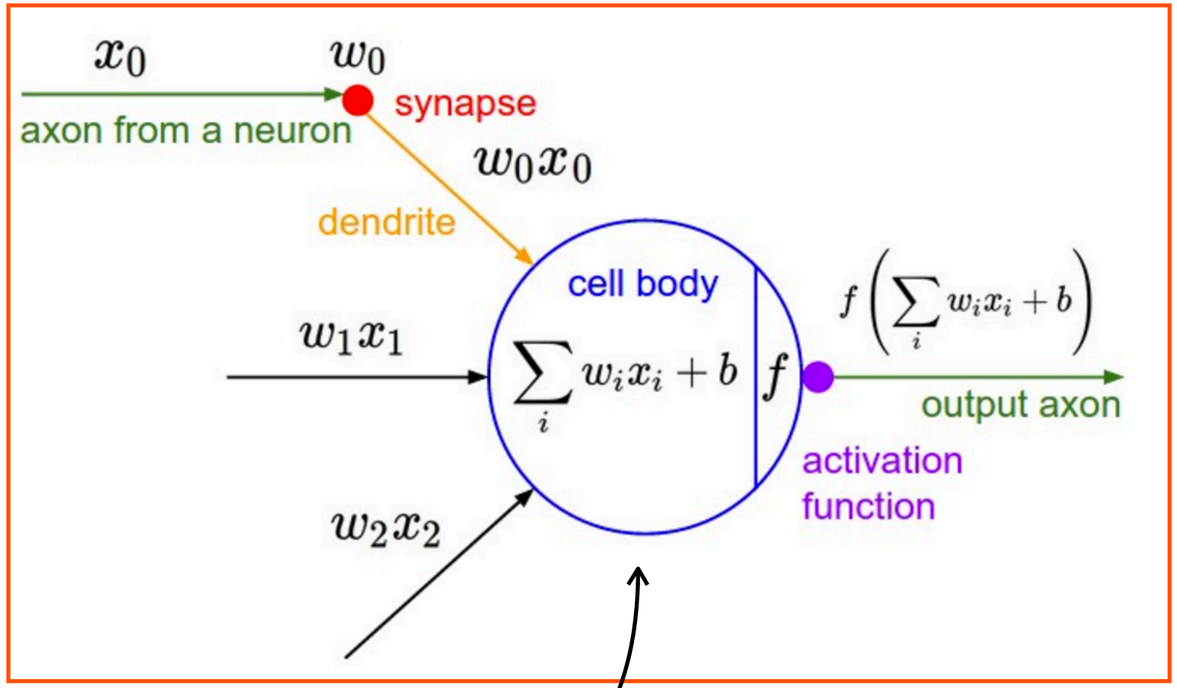
A Neuron


impulses carried
toward cell body
dendrites
branches
of axon
nucleus
axon
axon
terminals
impulses carried
away from cell body
cell body

Our Model


impulses carried
toward cell body
dendrites
nucleus
cell body
axon
branches
of axon
axon
terminals
impulses carried
away from cell body
x_0
w_0
synapse
axon from a neuron
$w_0 x_0$
dendrite
cell body
$w_1 x_1$
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation
function
$w_2 x_2$

Our Model



Weights, as in SVM

Activation Functions

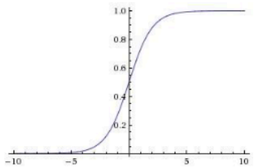
Activation functions are non-linearities which we attach at the end of each unit.

$f_{ReLU}(u) = Max(O, u)$

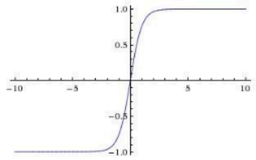
Where, $u = \mathbf{W.x}$

Sigmoid

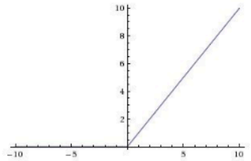
$$\sigma(x) = 1/(1 + e^{-x})$$



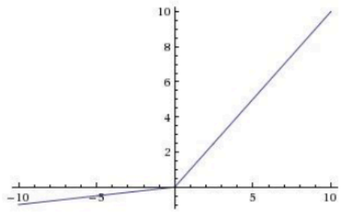
tanh tanh(x)



ReLU max(0,x)

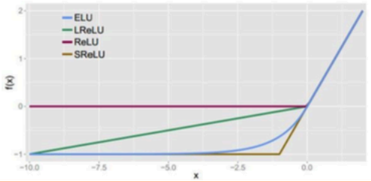


Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

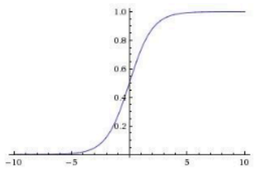
ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$



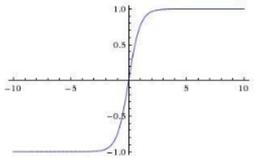
Activation Functions

Sigmoid

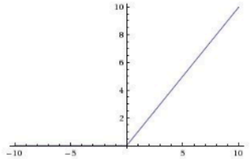
$$\sigma(x) = 1/(1 + e^{-x})$$



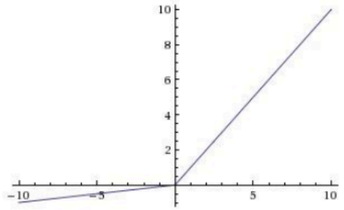
tanh tanh(x)



ReLU max(0,x)

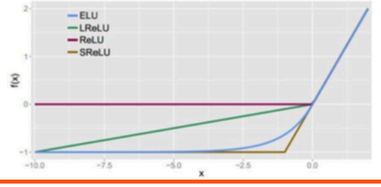


Leaky ReLU
max(0.1x, x)



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$



ReLU

ReLU. The Rectified Linear Unit has become very popular in the last few years. It computes the function $f(x) = \max(0, x)$. In other words, the activation is simply thresholded at zero (see image above on the left). There are several pros and cons to using the ReLUs:

- (+) It was found to greatly accelerate (e.g. a factor of 6 in Krizhevsky et al.) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
- (+) Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
- (-) Unfortunately, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as 40% of your network can be "dead" (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. With a proper setting of the learning rate this is less frequently an issue.

ReLU

ReLU. The Rectified Linear Unit has become very popular in the last few years. It computes the function $f(x) = \max(0, x)$. In other words, the activation is simply thresholded at zero (see image above on the left). There are several pros and cons to using the ReLUs:

- (+) It was found to greatly accelerate (e.g. a factor of 6 in Krizhevsky et al.) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
- (+) Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
- (-) Unfortunately, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as 40% of your network can be "dead" (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. With a proper setting of the learning rate this is less frequently an issue.

Sigmoid

Sigmoid. The sigmoid non-linearity has the mathematical form $\sigma(x) = 1/(1 + e^{-x})$ and is shown in the image above on the left. As alluded to in the previous section, it takes a real-valued number and "squashes" it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1). In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks:

- *Sigmoids saturate and kill gradients*. A very undesirable property of the sigmoid neuron is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Recall that during backpropagation, this (local) gradient will be multiplied to the gradient of this gate's output for the whole objective. Therefore, if the local gradient is very small, it will effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. Additionally, one must pay extra caution when initializing the weights of sigmoid neurons to prevent saturation. For example, if the initial weights are too large then most neurons would become saturated and the network will barely learn.
- *Sigmoid outputs are not zero-centered*. This is undesirable since neurons in later layers of processing in a Neural Network (more on this soon) would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive (e.g. $x > 0$ elementwise in $f = w^T x + b$)), then the gradient on the weights w will during backpropagation become either all be positive, or all negative (depending on the gradient of the whole expression f). This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights. However, notice that once these gradients are added up across a batch of data the final update for the weights can have variable signs, somewhat mitigating this issue. Therefore, this is an inconvenience but it has less severe consequences compared to the saturated activation problem above.

Sigmoid

Sigmoid. The sigmoid non-linearity has the mathematical form $\sigma(x) = 1/(1 + e^{-x})$ and is shown in the image above on the left. As alluded to in the previous section, it takes a real-valued number and "squashes" it into range between 0 and 1. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has seen frequent use historically since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1). In practice, the sigmoid non-linearity has recently fallen out of favor and it is rarely ever used. It has two major drawbacks:

- *Sigmoids saturate and kill gradients*. A very undesirable property of the sigmoid neuron is that when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero. Recall that during backpropagation, this (local) gradient will be multiplied to the gradient of this gate's output for the whole objective. Therefore, if the local gradient is very small, it will effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. Additionally, one must pay extra caution when initializing the weights of sigmoid neurons to prevent saturation. For example, if the initial weights are too large then most neurons would become saturated and the network will barely learn.
- *Sigmoid outputs are not zero-centered*. This is undesirable since neurons in later layers of processing in a Neural Network (more on this soon) would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive (e.g. $x > 0$ elementwise in $f = w^T x + b$)), then the gradient on the weights w will during backpropagation become either all be positive, or all negative (depending on the gradient of the whole expression f). This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights. However, notice that once these gradients are added up across a batch of data the final update for the weights can have variable signs, somewhat mitigating this issue. Therefore, this is an inconvenience but it has less severe consequences compared to the saturated activation problem above.

Complete Model

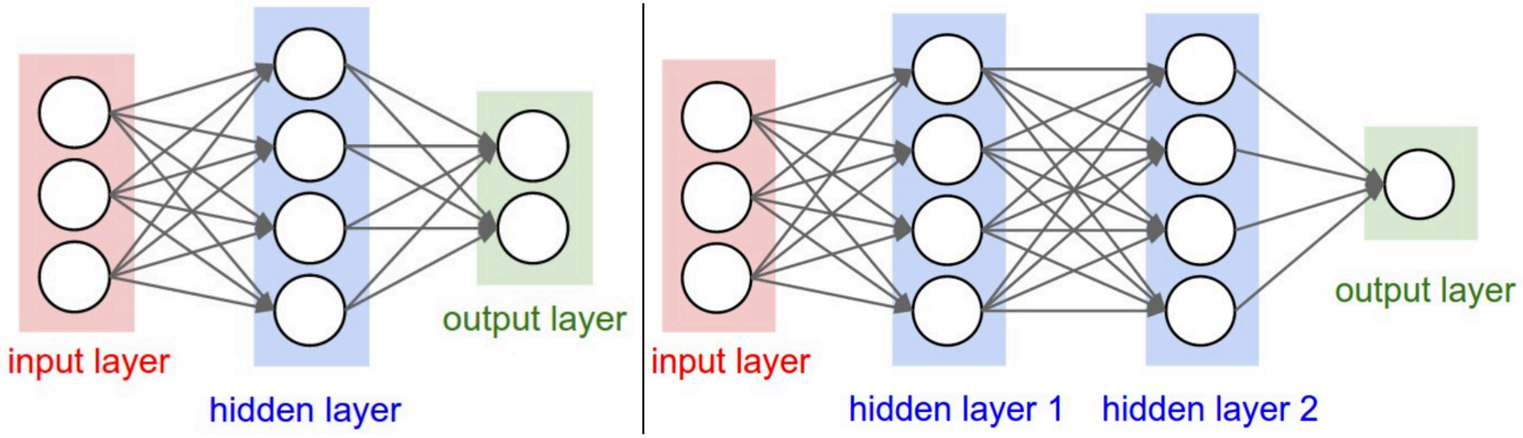
(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

or 3-layer Neural Network

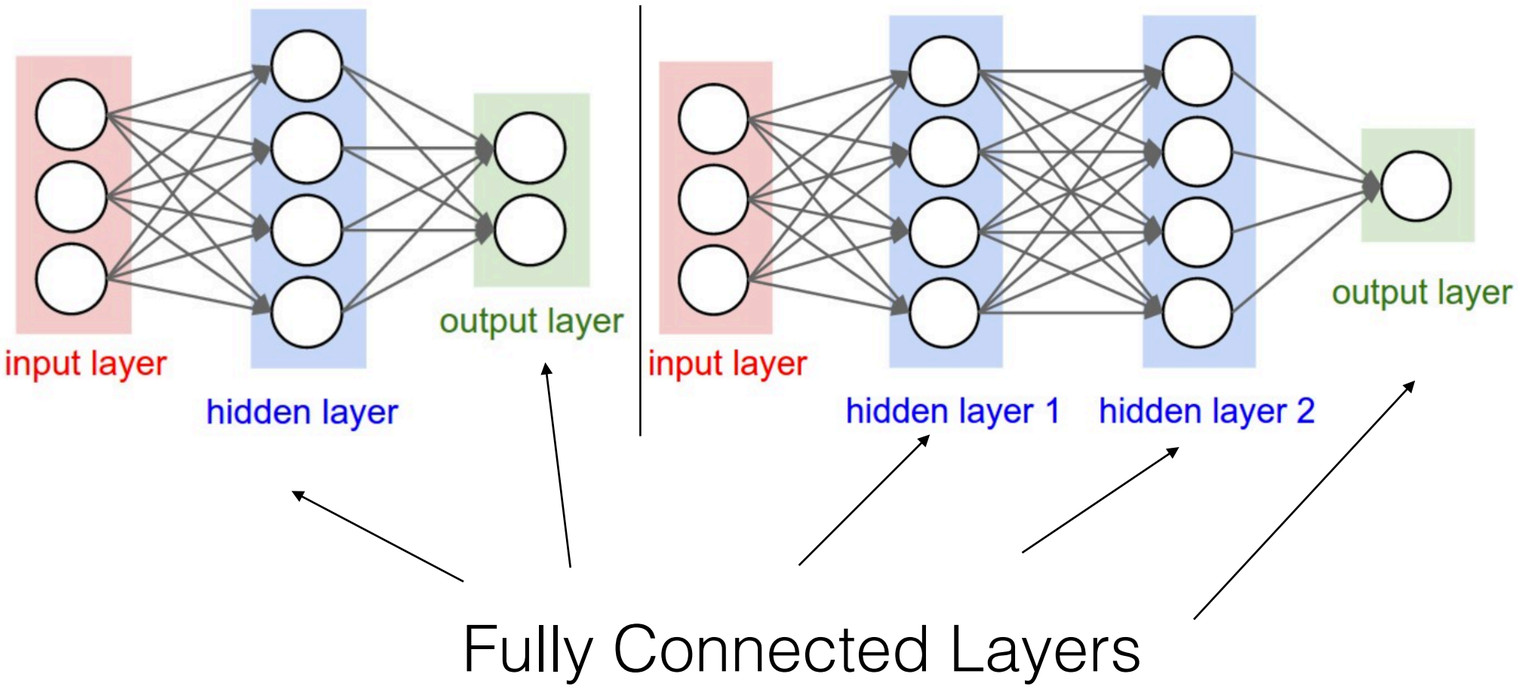
$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

A Neural **Network**

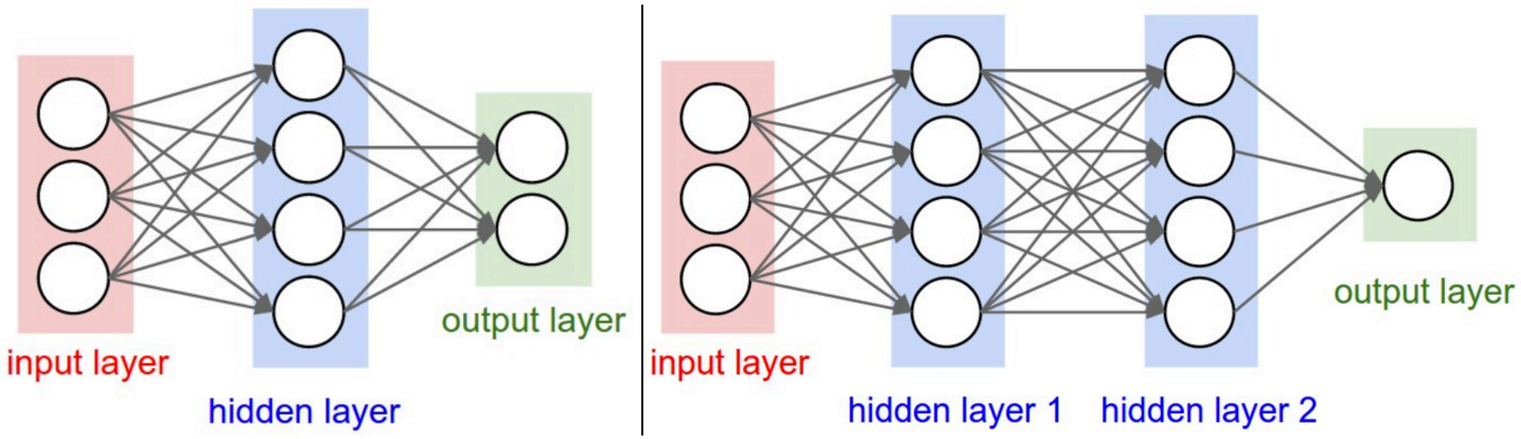


Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. **Right:** A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

A Neural **Network**



Size



Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. **Right:** A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

The first network (left) has 4 + 2 = 6 neurons (not counting the inputs), [3 x 4] + [4 x 2] = 20 weights and 4 + 2 = 6 biases, for a total of 26 learnable parameters.
The second network (right) has 4 + 4 + 1 = 9 neurons, [3 x 4] + [4 x 4] + [4 x 1] = 12 + 16 + 4 = 32 weights and 4 + 4 + 1 = 9 biases, for a total of 41 learnable parameters.

Training

Dropout: Randomly drop out neurons in a layer when training.

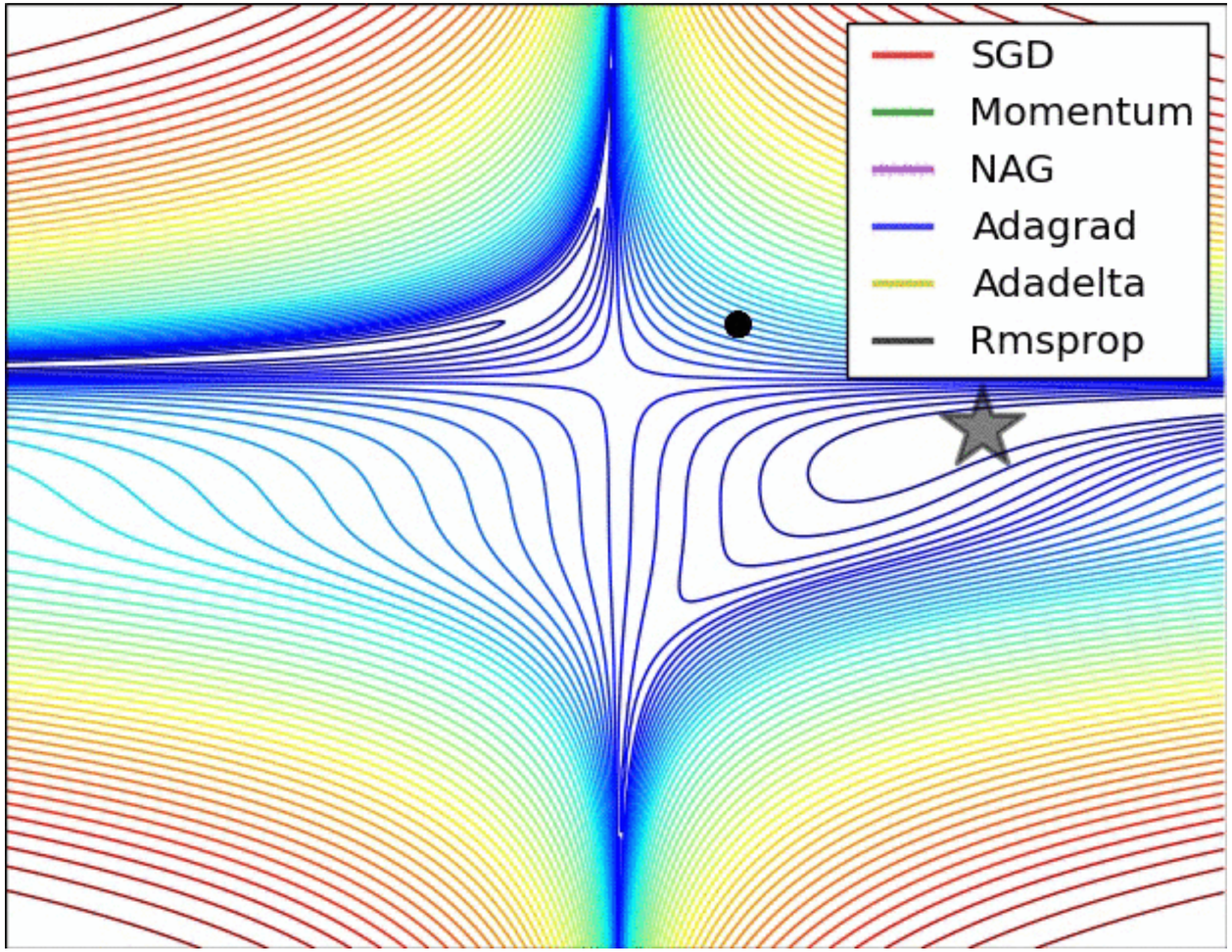
Mini-batch gradient descent: Choose batch size according to GPU. Choose variations/extensions of SGD like Momentum, Adam, Adagrad..

Initialization: Be careful while initializing so that you don't get dead neurons.

Activation Function: Use ReLU or some version of it. If you have to, use tanh.

Deeper —> Better

Training



What did we just build?

One way to look at Neural Networks with fully-connected layers is that they define a family of functions that are parameterized by the weights of the network.

A natural question that arises is: What is the representational power of this family of functions? In particular, are there functions that cannot be modeled with a Neural Network?

It turns out that Neural Networks with at least one hidden layer are ***universal approximators***.

Why go deeper?

The answer is that the fact that a two-layer NN is a universal approximator is, while mathematically cute, a relatively weak and useless statement in practice.

Neural Networks work well in practice because they compactly express nice, smooth functions that fit well with the statistical properties of data we encounter in practice, and are also easy to learn using our optimization algorithms (e.g. gradient descent).

Similarly, the fact that deeper networks (with multiple hidden layers) can work better than a single-hidden-layer networks is an empirical observation, despite the fact that their representational power is equal

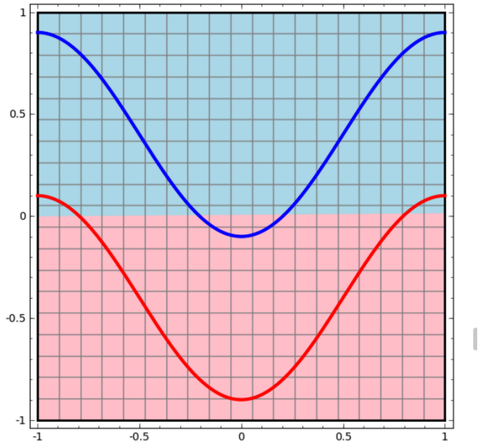
Topological Argument

So, while we do understand how to build these networks, and know that they have good properties, what is actually happening across the layers? And How do we understand this?

Let's look at how it classifies data.

1.

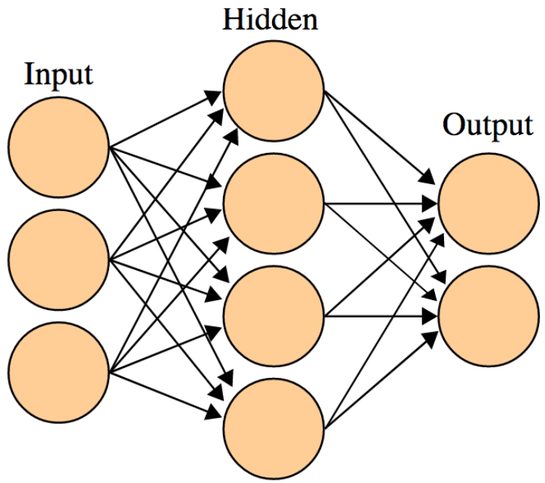
10	A
14	B
-12	C
-2	D

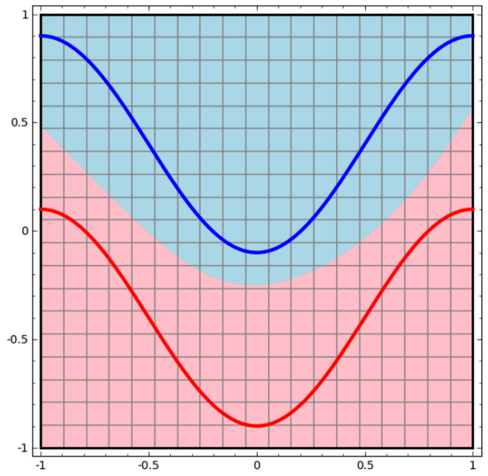


We'll start with the simplest possible class of neural network, one with only an input layer and an output layer.

This is not much more interesting than a SVM.

2.

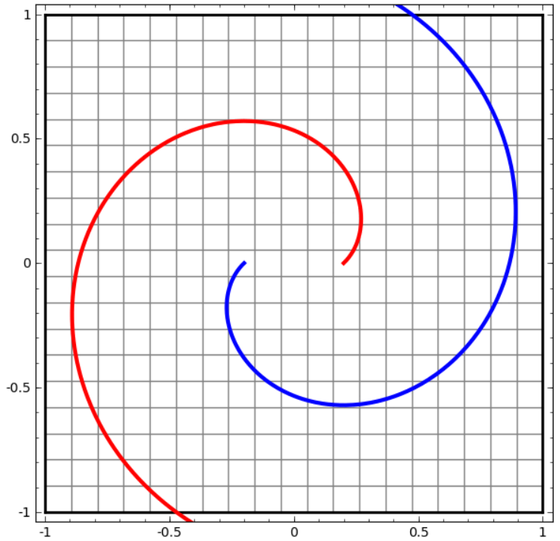




With each layer, the network transforms the data, creating a new ***representation***.

We can look at the data in each of these representations and how the network classifies them. **When we get to the final representation, the network will just draw a line through the data (or, in higher dimensions, a hyperplane).**

3.

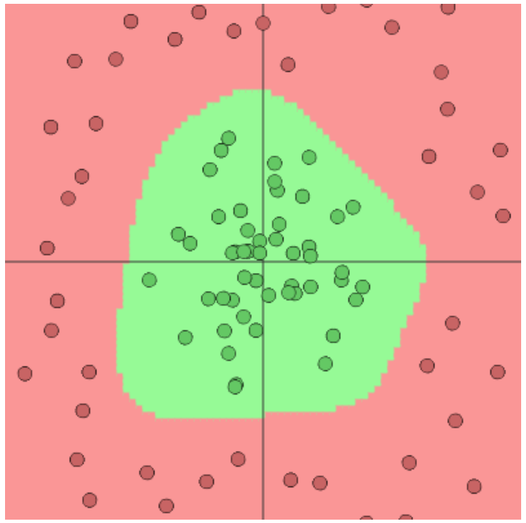


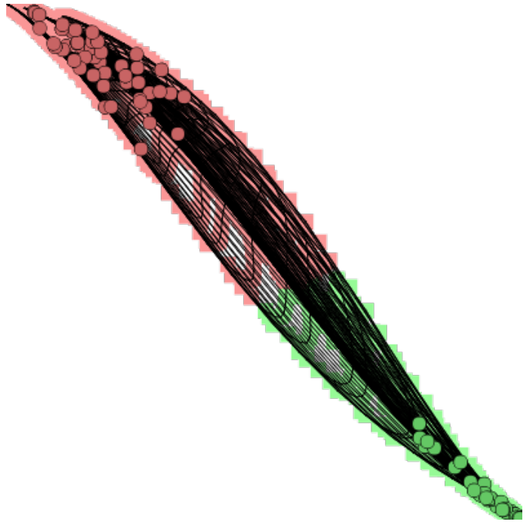
The story is much the same for other standard layers, consisting of an affine transformation followed by pointwise application of a monotone activation function.

We can apply this technique to understand more complicated networks. For example, the following network classifies two spirals that are slightly entangled, using **four hidden layers**.

4.

More Examples





Link

Applications

Anywhere where representations can not be explicitly written and there is a need to learn them.

There also needs to be enough data.

Majority of the advancements have taken place in:

- Computer Vision

- NLP

How does Deep Learning solve this?

and how did it become so popular?

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction.

Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer.

These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains...