Deep Learning

Dr. Omri Allouche

Y-Data Deep Learning Course

Machine Learning at a Glance

Data

Training

Model

Prediction / Inference

Classification
Regression

"Cat"
88.5%

Cats

Dogs

(Labels)

New (Unseen) Data

Traditional Programming

Machine Learning

IMAGENET Large Scale Visual Recognition Challenge

1,000 object classes
Over 1M images
(currently 14M)

High level category	# synset (subcategories)	Avg # images per synset	Total # images
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	188K
fungus	303	453	137K
furniture	187	1043	195K
geological formation	151	838	127K
invertebrate	728	573	417K
mammal	1138	821	934K
musical instrument	157	891	140K
plant	1666	600	999K
reptile	268	707	190K
sport	166	1207	200K
structure	1239	763	946K
tool	316	551	174K
tree	993	568	564K
utensil	86	912	78K
vegetable	176	764	135K
vehicle	481	778	374K
person	2035	468	952K

28x28
pixels
784 pixels
...
"neurons"
...
0
1
2
9

$$f\left(\sum_i w_i x_i + b\right)$$

28x28
pixels

$$f\left(\sum_i w_i x_i + b\right)$$

The Universal Approximation Theorem

- One hidden layer is sufficient to approximate any **continuous** function for inputs within a **specific range.**

The Universal Approximation Theorem

- The **universal approximation theorem** states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbf{R}^n$, under mild assumptions on the activation function
- Kurt Hornik showed in 1991 that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential of being universal approximators
- Although feed-forward networks with a single hidden layer are universal approximators, the width of such networks has to be exponentially large

Multi-Class Classification with NN and SoftMax Function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Softmax

- Softmax is a generalization of the logistic regression binary activation to multiple classes
- Generated values are in range [0, 1]
- It favors the max, in a "soft" way (in contrast to a "hard" max)

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax

$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector

e^{z_j} = standard exponential function for output vector

Input pixels, $\mathbf{x}$

Forward propagation

Feedforward output, $\mathbf{y}_i$

cat	dog	horse
5	4	2
4	2	8
4	4	1

Softmax function

Softmax output, $\mathbf{S}(\mathbf{y}_i)$

cat	dog	horse
0.71	0.26	0.04
0.02	0.00	0.98
0.49	0.49	0.02

Shape: (3, 32, 32)

Shape: (3,)

Shape: (3,)

Slides credit: Andrew Ng

Softmax + Cross-entropy

Slide credit: Andrew Ng

Cross Entropy (Log Likelihood Loss)

- For a binary classification problem:

$$-(y\log(p) + (1-y)\log(1-p))$$

- For a multiclass problem;

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

Note

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Gradient Descent

Suppose we have a **<u>scalar function</u>**

$$f(w): \Re \rightarrow \Re$$

We want to find a local minimum.

Assume our current weight is *w*

GRADIENT DESCENT RULE:

$$w_{t+1} \leftarrow w_t - \eta \frac{\partial}{\partial w} f(w_t)$$

η is the ***learning rate***.

A small positive number, e.g. η = 0.05

Gradient Descent Illustration

Code in Python can be obtained from: https://www.cs.toronto.edu/~frossard/post/linear_regression/

Optimizing Neural Networks Using Gradient Descent

WHAT ARE WE DOING TODAY, BRAIN?
THE SAME THING WE DO EVERY DAY, PINKY.
BACKPROPAGATE
memegenerator.net

Deep Learning Training Recipe

- Randomly choose the initial weights
- While error is too large
  - For each training pattern (presented in random order)

    Feedforward:

    - Apply the inputs to the network
    - Calculate the output for every neuron from the input layer, through the hidden layer(s), to the output layer

    Loss:

    - Calculate the error at the outputs
    - Use the output error to compute the **loss** – error signals for pre-output layers

    Backpropagate:

    - Use the loss to compute weight adjustments
    - Apply the weight adjustments
  - Periodically evaluate the network performance

Terms

- **Epoch** – a single pass through all examples in the training set
- Why do we need to use multiple epochs?
- **Batch size** – The number of examples used in each step of gradient descent
- **Iterations** – a step of weights update (forward pass + backward pass)

Gradient Descent with Batches

Large Minibatches

- A larger batch size requires more memory
- A minibatch with *m* examples requires *O(m) computation* and use *O(m) memory*, but reduces the amount of uncertainty in the gradient by a factor of only *O(sqrt(m))*.
- Computing gradient on the entire training set is also an approximation – why?
- Common batch sizes are often 32-1024 samples

28x28
pixels
784 pixels
...
"neurons"
...
0
1
2
9

Deep Networks Perform Better!

Fully Connected Layers

$$f\left(\sum_i w_i x_i + b\right)$$

How can we teach Neural Networks?

- There are complex NN architectures
- Using exact mathematical derivation is impossible in many cases

Training in progress. . .

The Basics of Learning

Toy Problem – The Cost of Riding a Taxi

- Taxi fares are affected by:
  - the distance passed
  - The time taken
  - Fixed cost

$$price = x_{distance} w_{distance} + x_{time} w_{time} + 1 * w_{fixed_cost}$$

- You want to learn the effect of each
- Your data – the final cost of rides you're making, and the "input" values

Taxi Ride Calculation

- It’s possible to compute things analytically with a pen & paper
- But you’re sitting in a taxi, without a pen and without a paper…
- What can you do?

An Idea: Let’s try an iterative method

- An iterative method allows us to solve non-linear problems with complex behavior
- We can handle cases with errors
- We don’t necessarily want the correct weights – we want weights that result in low error
- Neurons in the brain don’t compute things analytically – how do they do it?

Solution 1 – Randomly change weights

- Randomly perturb one weight and see if it improves performance
- If so, save the change
- Doesn’t scale well – each time we modify just a single weight
- Near the optimal solutions, the vast majority of changes will make things worse
- Changing multiple weights at once requires many samples, to separate the effect of each weight

Solution 2 – Randomly Change the Activations

- There are much fewer activations than weights
- Once we find the optimal activations, we can continue to find the optimal weights that generate them
- Still inefficient relative to back propagation

Solving the equations iteratively

- The cost of the ride is given by:

$$price = x_{distance} w_{distance} + x_{time} w_{time} + 1 * w_{fixed_cost}$$

- We wish to find

$$\mathbf{w} = (w_{distance}, w_{time}, w_{fixed_cost})$$

- Start with guesses for the weights and adapt iteratively according to the error.

The true weights used

Price of ride = 4.8 + 2.75 + 11.3 = 18.85 = target

Deriving the delta rule

- Define the error as the squared residuals summed over all training cases:

$$E = \frac{1}{2} \sum_{n \in training} (t^n - y^n)^2$$

- Now differentiate to get error derivatives for weights

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_n \frac{\partial E^n}{\partial w_i} = \sum_n \frac{\partial y^n}{dy^n} x_i^n (t^n - y^n)$$

- The batch delta rule changes the weights in proportion to their error derivatives summed over all training cases

$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i} = -\varepsilon \sum_n x_i^n (t^n - y^n)$$

Starting from random weights

Predicted price of ride =
6+1.1+10 = 17.1

- Residual error = 17.1-18.2 = -1.1
- The "delta-rule" for learning is:

$$\Delta w_i = \varepsilon\, x_i (t - y)$$

- With a learning rate ε of 0.1, the weight changes are +0.66, +.011, +1.1
- This gives new weights of 1.66, 0.111, 11.1
- Notice that the weight for kilometers got worse!

Notes

- Notice that the weight for kilometers got worse!
- Note that all weights update in the same direction (all positive or all negative)
- Correlations between feature make learning much slower
  - Since distance and time are highly correlated, it is difficult to find the separate effect of each
- In the real world, labels might be incorrect
- There might not be a perfect set of weights

Backpropagation

- We'll use the Chain Rule:

Slides credit: Stanford CS231N

"local gradient"
x
y
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z

x
y
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
"local gradient"
f
z
$\frac{\partial L}{\partial z}$
gradients

x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
y
z
$\frac{\partial L}{\partial z}$
gradients

x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Backpropagation: a simple example

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Chain rule:

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

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

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

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

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:

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

Another example

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$(\frac{-1}{1.37^2})(1.00) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$(1)(-0.53) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$(e^{-1})(-0.53) = -0.20$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

(-1) * (-0.20) = 0.20

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

[local gradient] x [upstream gradient]

[1] x [0.2] = 0.2

[1] x [0.2] = 0.2 (both inputs!)

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

[local gradient] x [upstream gradient]

x0: [2] x [0.2] = 0.4

w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

sigmoid function

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

sigmoid function

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

(0.73) * (1 - 0.73) = 0.2

Tree Age Prediction

Age	Width	Length
5	5	7
8	4	9
9	6	12
4	3	6
3	2	3

Input Layer
Hidden Layer
Output Layer
First Observation
$w_{1,2} = [0.5, 0.25];\ b_1 = 1$
Width
5
w_1
$(5 \cdot 0.5) + (7 \cdot 0.25) + 1 = 5.25$
w_5
$w_{5,6} = [1, 0];\ b_3 = 1$
Result
w_3
w_2
$w_{3,4} = [0.5, 0.25];\ b_2 = 0$
$(5.25 \cdot 1) + (4.25 \cdot 0) + 1$
6.25
Length
7
w_4
$(5 \cdot 0.5) + (7 \cdot 0.25) + 0 = 4.25$
w_6

Neural Networks Playground

http://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=gauss&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=5,2&seed=0.62520&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=true&xSquared=true&ySquared=true&cosX=false&sinX=true&cosY=false&sinY=true&collectStats=false&problem=classification&initZero=false&hideText=false

The Training Process

- When should we stop?

The Training Process

- When should we stop?

Convolutional Neural Network

A convolution-pooling layer in Convolutional Neural Network is a composite function decomposed into function elements $f^{(conv)}$, $f^{(sigm)}$, and $f^{(pool)}$.

Let x be the output from the previous layer. Sigmoid nonlinearity is optional.

$$\left(f^{(pool)} \circ f^{(sigm)} \circ f_w^{(conv)}\right)(x)$$

Forward propagation

Backward propagation

WHAT ARE WE DOING TODAY, BRAIN?
THE SAME THING WE DO EVERY DAY, PINKY.
BACKPROPAGATE
memegenerator.net

Deep Learning Training Recipe

- Randomly choose the initial weights
- While error is too large
  - For each training pattern (presented in random order)

    Feedforward:

    - Apply the inputs to the network
    - Calculate the output for every neuron from the input layer, through the hidden layer(s), to the output layer

    Loss:

    - Calculate the error at the outputs
    - Use the output error to compute the **loss** – error signals for pre-output layers

    Backpropagate:

    - Use the loss to compute weight adjustments
    - Apply the weight adjustments
  - Periodically evaluate the network performance

Terms

- **Learning rate** – a multiplier of the gradient that often reduces the magnitude of the update step taken
- **Iteration** – a step of weights update (forward pass + backward pass)
- **Batch size** – The number of examples used in each step of gradient descent. A larger batch size better estimates the gradient but requires more memory
- **Epoch** – a single pass through all examples in the training set
- Why do we need to use multiple epochs?

Neural Networks in PyTorch

- The network architecture is often defined as a class inheriting from nn.Module
  - Inits parameters in the constructor
  - Implements the *forward* method
- We then define the loss function and the optimizer:

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Training the Network is done in a loop

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html