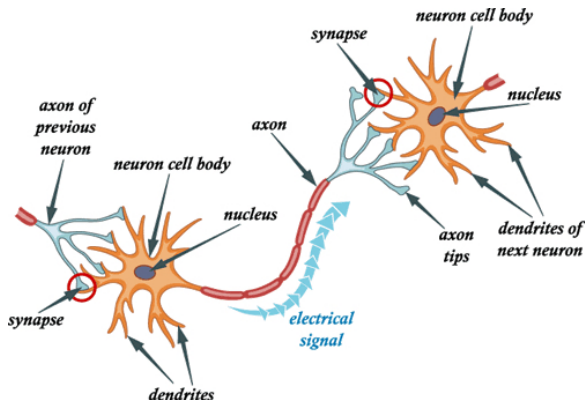


# Neural Networks <sup>1</sup>

---

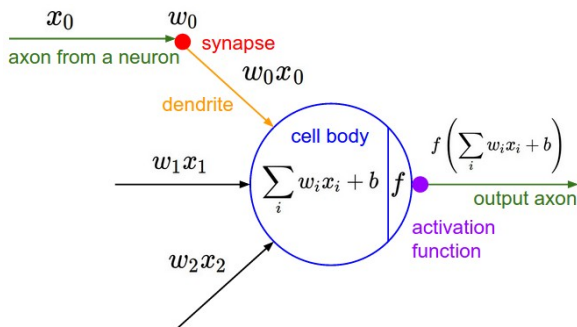
<sup>1</sup>Based on materials in PRML Ch 5, <http://cs231n.github.io/> and <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

- **Neural Networks (NN)** are a family of functions:  $y = g(x)$ .
- NN is extremely flexible: it can approximate any continuous function arbitrarily well.
- NN can be used for **regression** and **classification** tasks.
- Historically, NN was inspired by modeling biological neural networks.
- Deep Learning is NN combined with modern learning techniques



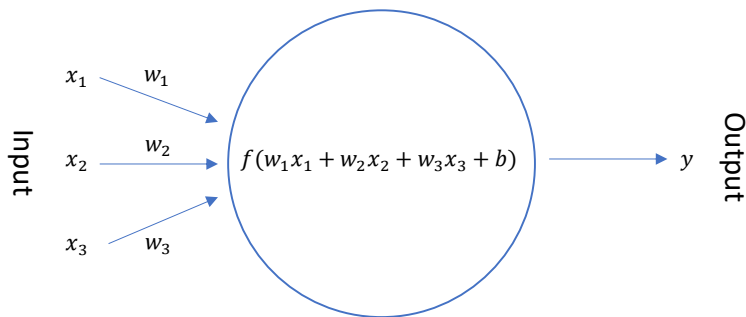
- Our neural system has  $\sim 86$  billion neurons and they are connected with  $10^{14}$ - $10^{15}$  synapses
- Neuron receives **input** signals from its dendrites and produces **output** signals along its axon
- The axon eventually branches out and connects via synapses to dendrites of other neurons.

## Simplistic mathematical modeling of biological neural networks



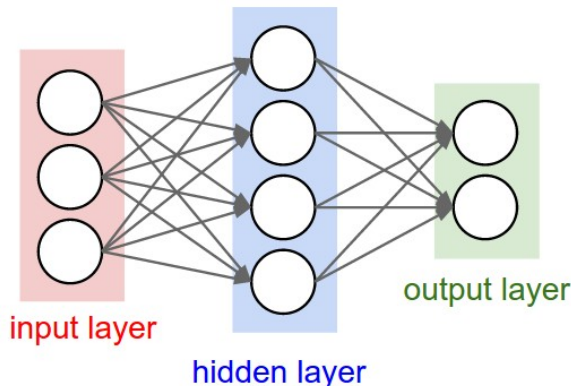
- $x_0, x_1, x_2$ : input signals
- $w_0, w_1, w_2$ : weights or synaptic strengths. Excitatory (positive weight) or inhibitory (negative weight).
- If the weighted sum of the input signals is above certain threshold, the neuron fires and sends a spike along its axon.
- $f(\cdot)$ : activation function which outputs the frequency of neuron firing.

## A Single Neuron



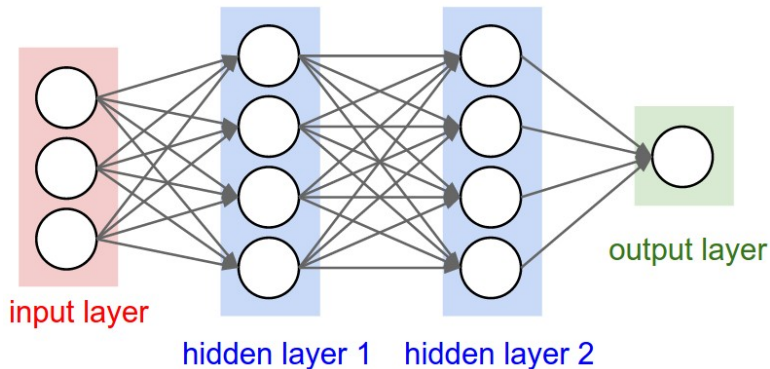
- Neuron is also known as a **node** or **unit**.
- $x_1, x_2, x_3$  are **inputs**.
- $w_1, w_2, w_3$  are **weights**.  $b$  is **bias**.
- $f(\cdot)$  is an **activation function**.
- $y$  is an **output**.
- A neuron is simply a transformation of its inputs.

A **feed-forward NN (also known as multilayer perceptron, MLP)** is a collection of neurons that are connected in a graph. In other words, the outputs of some neurons can become inputs to other neurons.



- We call this a **two-layer** NN. Notice, for naming purpose, we didn't count the input layer.

## Three-layer NN



## Example: One-layer NN



## Example: Two-layer NN

## Example: Another Two-layer NN

## Notations:

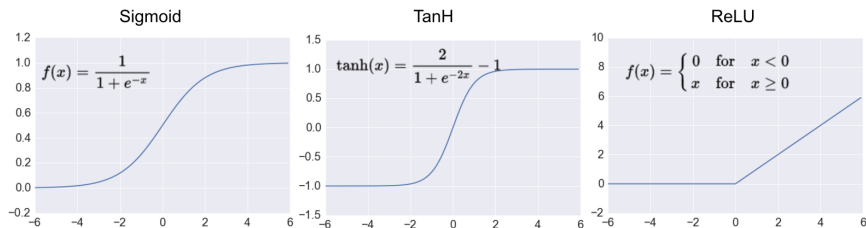
- $L$  layers ( $L - 1$  hidden layers)
- $p_\ell$ : number of nodes in layer  $\ell = 0, \dots, L$  where  $p_0$  is the number of input variables and  $p_L$  is the number of output variables. In this course, we've only considered  $p_L = 1$ , i.e. univariate response variable.
- $h^{(\ell)}$  is a  $p_\ell$ -dim vector of nodes in layer  $\ell$ .  $h^{(0)} = (x_1, \dots, x_{p_0})^T$  is the set of inputs and  $h^{(L)} = (y_1, \dots, y_{p_L})^T$  is the set of outputs.
- $b^{(\ell)}$  is a  $p_\ell$ -dim vector of biases
- $W^{(\ell)}$  is a  $p_{\ell-1} \times p_\ell$  weight matrix

Then a feed-forward NN is defined recursively for  $\ell = 1, \dots, L$

$$h^{(\ell)} = f \left( W^{(\ell)T} h^{(\ell-1)} + b^{(\ell)} \right)$$

where the activation function  $f(\cdot)$  is applied element by element. This is also known as **forward-propagation**.

# Popular activation functions



- **Sigmoid**: takes a real-valued input and squashes it to range between 0 and 1
- **TanH**: takes a real-valued input and squashes it to the range  $[-1, 1]$
- **ReLU (most popular)**: ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero).
- We only consider **nonlinear** activation functions because linear activation degenerates (linear transformation of linear transformation is still linear transformation).

- **Regression**: find weights which minimize the **squared error loss**

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

where  $\hat{y}_i = g(\mathbf{x}_i, \mathbf{w})$  is the output from the NN with input  $\mathbf{x}_i$  and weights  $\mathbf{w}$ .

- **Binary classification**: find weights which minimize the **cross-entropy loss**

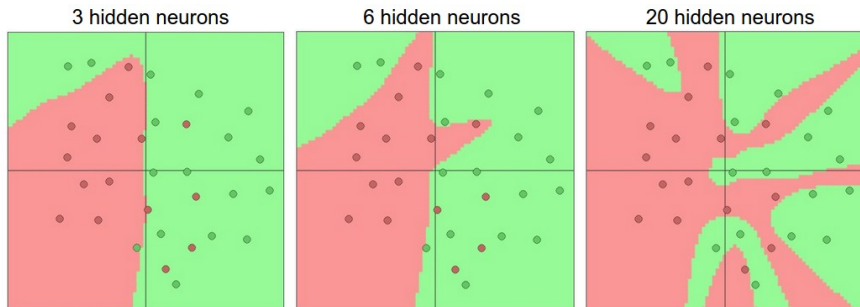
$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} - \sum_{i=1}^n \{y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\}$$

where  $0 \leq \hat{y}_i \leq 1$  is the output from NN with sigmoid activation function in the last layer.  $\hat{y}_i$  can be interpreted as the probability of  $y_i = 1$ . Multi-class classification can be trained in a similar fashion (see PRML 5.2).

- To find  $\hat{\mathbf{w}}$ , we use **back-propagation**: (stochastic) gradient descent + chain rule.

## How to choose number of layers and number of hidden units?

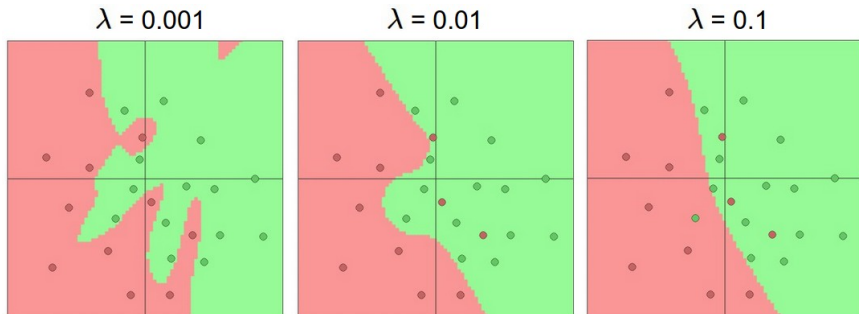
- More layers and hidden units increase the flexibility of the NN but tends to overfit.



- The model with one hidden layer and 20 hidden units fits all the training data but at the cost of segmenting the space into many disjoint red and green decision regions.
- **Rule of thumb:** choose 2 or 3 hidden layers and moderate to large number of hidden units and use regularization (e.g.  $\ell_1/\ell_2$  regularization, dropout) to prevent overfitting.

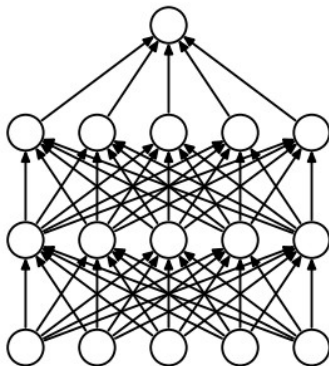
## $\ell_2$ regularization (weight decay)

- Add  $\frac{1}{2}\lambda w^2$  to the loss/objective function for each weight  $w$ .  $\lambda$  is a tuning parameter that controls the strength of the regularization.

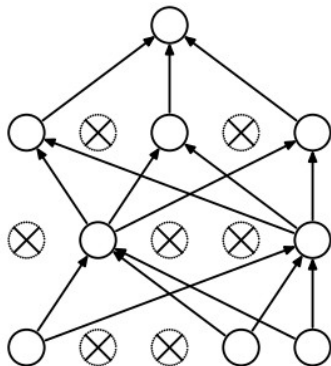


- Each neural network above has 20 hidden units, but changing the regularization strength makes its final decision regions smoother with a higher regularization.
- In practice,  $\lambda$  is chosen using cross-validation.

# Dropout



(a) Standard Neural Net



(b) After applying dropout.

- While training, dropout is implemented by only keeping a neuron active with some probability  $p$
- During testing, there is no dropout applied with each weight is multiply by  $p$ .
- In practice,  $p = 50\%$  works well.



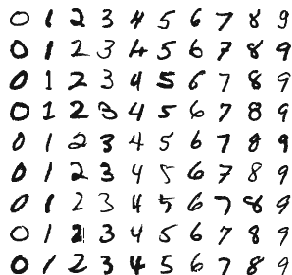
- **Data preprocessing**

- Normalization: standardization or scale each variable to  $(-1, 1)$
- Whitening: principal component analysis (will discuss later in this course)

- **Weight initialization**

- **DON'T** set all the initial weights to zero. If every neuron in the network computes the same output, then they will also all compute the same gradients during backpropagation and undergo the exact same parameter updates. In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same.
- Instead, for a ReLU neuron with  $m$  inputs, draw  $w_i \sim N(0, 2/m)$  for  $i = 1, \dots, m$ . This ensures that the input and the output have the same variance (also the same distribution). As a consequence, all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence.

- **Convolutional Neural Networks (CNN or ConvNet)** is a NN specifically designed for image inputs.
- Very popular in computer vision and imaging analysis.
- Most common task is to classify images.



airplane

automobile

bird

cat

deer

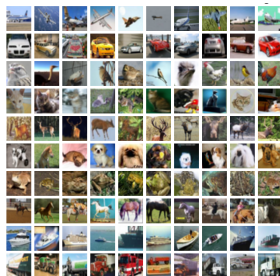
dog

frog

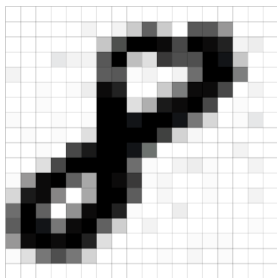
horse

ship

truck



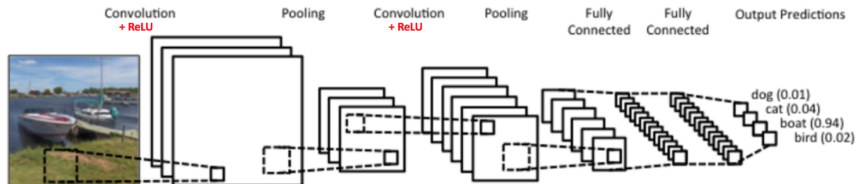
## Why not use regular NN?



```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 12 0 11 39 137 37 0 152 147 84 0 0 0
0 0 1 0 0 0 41 160 250 255 235 162 255 238 206 11 13 0
0 0 0 16 9 9 158 251 45 21 184 159 154 255 233 40 0 0
10 0 0 0 0 0 145 146 3 10 0 11 124 253 255 187 0 0
0 0 3 0 4 15 236 216 0 0 38 109 247 240 163 0 11 0
1 0 2 0 0 0 253 253 23 62 224 241 255 164 0 5 0 0
6 0 0 4 0 3 252 250 228 255 255 234 112 28 0 2 17 0
0 2 1 4 0 21 255 253 251 255 172 31 8 0 1 0 0 0
0 0 4 0 163 225 251 255 229 120 0 0 0 0 0 11 0 0
0 0 21 162 255 255 254 255 126 6 0 10 14 6 0 0 9 0
3 79 242 255 141 66 255 245 189 7 8 0 0 5 0 0 0 0
26 221 237 98 0 67 251 255 144 0 8 0 0 7 0 0 11 0
125 255 141 0 87 244 255 208 3 0 0 13 0 1 0 1 0 0
145 248 228 116 235 255 141 34 0 11 0 1 0 0 0 1 3 0
85 237 253 246 255 210 21 1 0 1 0 0 6 2 4 0 0 0
6 23 112 157 114 32 0 0 0 0 2 0 8 0 7 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

- A 2D color image is a 3 dimensional array (width  $\times$  height  $\times$  depth) of pixel values. Depth=3 corresponding to 3 color channels (red, green, blue).
- A grayscale image is a matrix (width  $\times$  height) of pixel values.
- Each pixel value is an input.
- A small  $200 \times 200$  color image would lead to  $200 \times 200 \times 3 = 120,000$  weights for each hidden unit.
- We may need hundreds of hidden units.
- The full connectivity of a regular NN is wasteful and the huge number of parameters would quickly lead to overfitting.

# ConvNet Architecture



Four building blocks

- 1 Convolution
- 2 ReLU
- 3 Pooling
- 4 Fully Connected Layer (MLP)

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

$5 \times 5$  image

1	0	1
0	1	0
1	0	1

$3 \times 3$  filter



## What's going on the last slide








We slide the orange matrix over our original image (green) by 1 pixel (also called **stride**) and for every position, we compute element wise multiplication (between the two matrices) and add the multiplication outputs to get the final integer which forms a single element of the output matrix (pink). Note that the  $3 \times 3$  matrix “sees” only a part of the input image in each stride.

In ConvNet terminology, the  $3 \times 3$  matrix is called a **filter** or kernel or feature detector and the matrix formed by sliding the filter over the image and computing the dot product is called the Convolved Feature or Activation Map or the **Feature Map**. It is important to note that filters acts as feature detectors from the original input image.

## Different filter effects

Different filters can detect different features from an image, for example edges, curves, blobs of color etc.

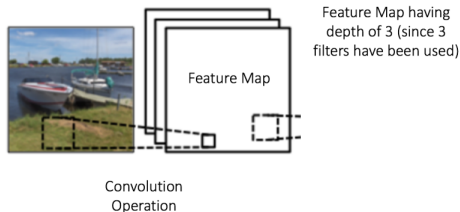
In practice, a ConvNet learns the values of these filters on its own during the training process

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	



The size of the Feature Map is controlled by three parameters

- **Depth**: the number of filters used for the convolution operation.

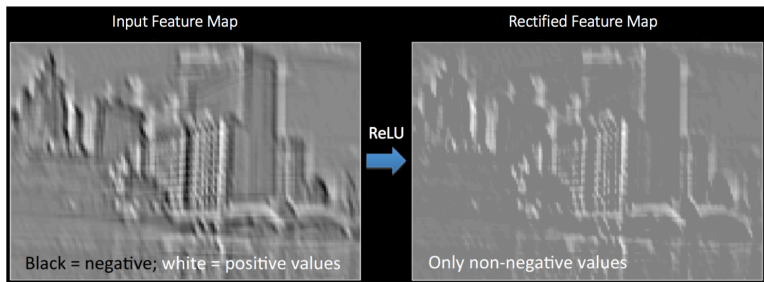


- **Stride**: the number of pixels by which we slide our filter matrix over the input matrix. In practice, we rarely use stride greater than 2.
- **Zero-padding**: pad the input matrix with zeros (i.e. black margins) so that it allows us to control the size of the feature maps

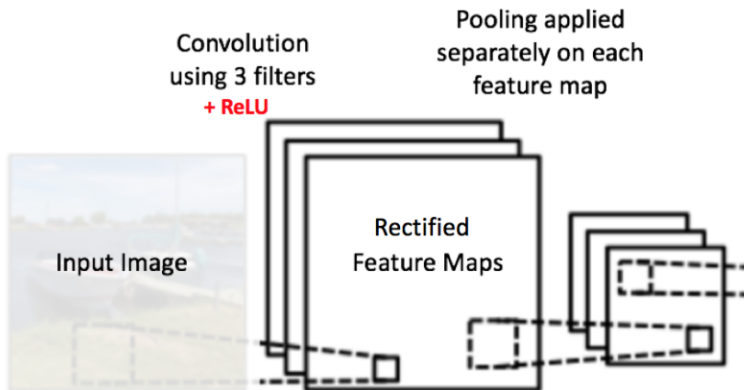


- Each of the 96 filters shown here is of size  $11 \times 11 \times 3$ .

- Apply ReLU (element wise) to the feature map to introduce non-linearity.



- Max pooling progressively reduce the spatial size of each feature map while keeping the most important information. It reduces the amount of parameters and computation in the network, and hence to also control overfitting.



- The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2. Every MAX operation would in this case be taking a max over 4 numbers.

Single depth slice

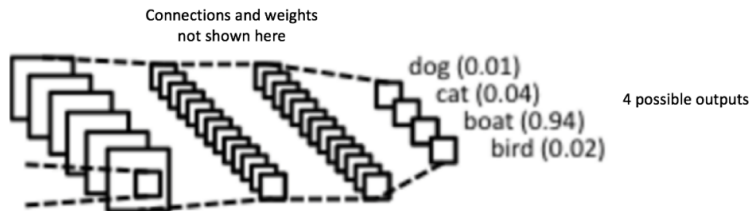
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters  
and stride 2



6	8
3	4

- The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.





<http://scs.ryerson.ca/~aharley/vis/conv/flat.html>