Introduction to Neural Networks

Motivation

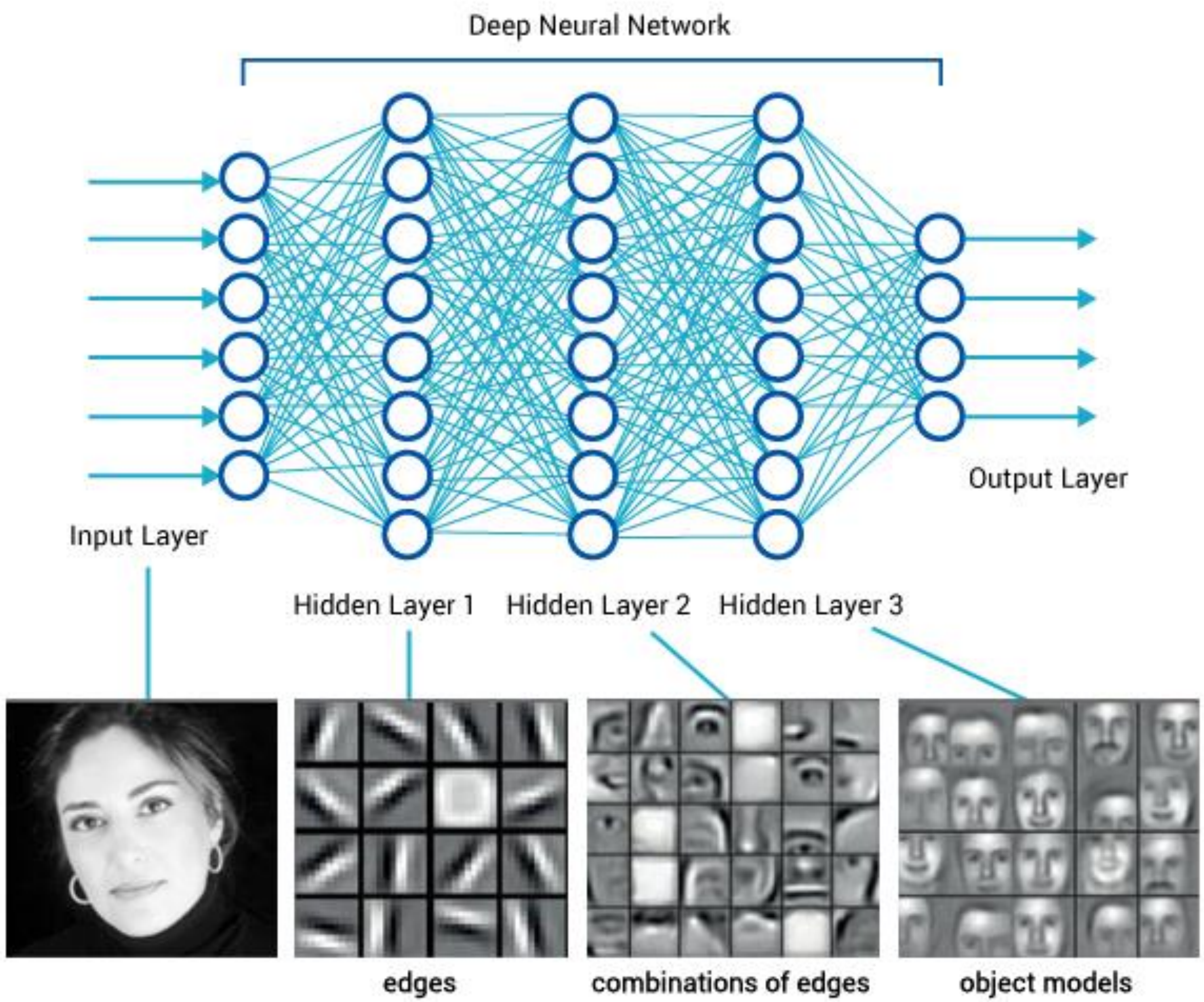
Limitations of linear models

- Logistic regression and other linear models cannot handle nonlinear decision boundaries
  - Must use non-linear feature transformations
  - Up to designer to specify which one

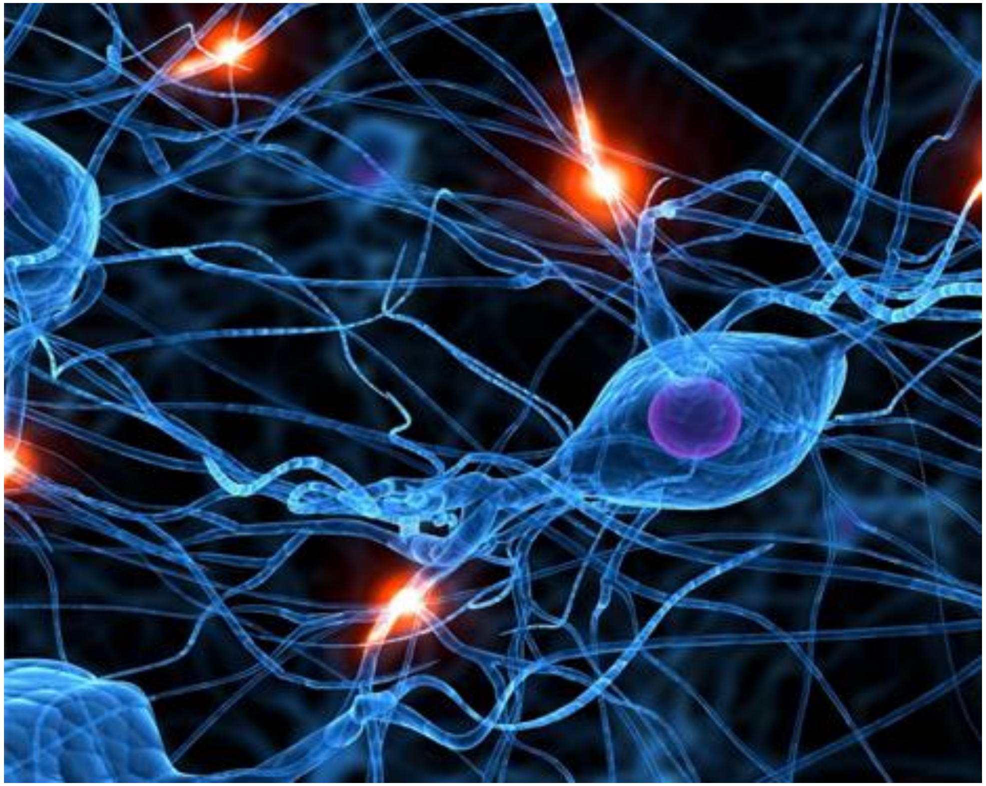
- Can we instead learn the transformation?
  - Yes, this is what neural networks do!

- A Neural network chains together many layers of “neurons” such as logistic units (logistic regression functions)

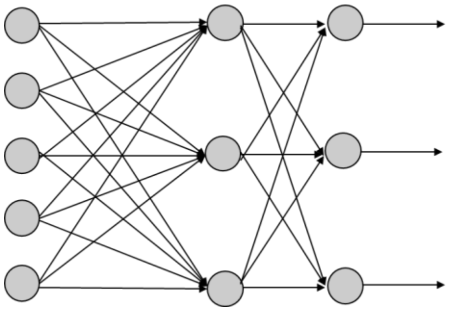
Neural Networks learn features



Neurons in the Brain

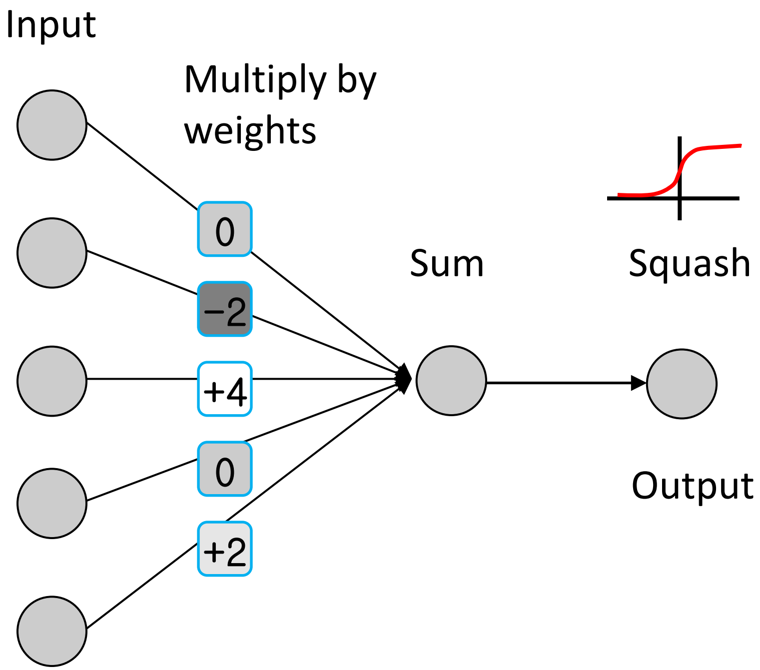


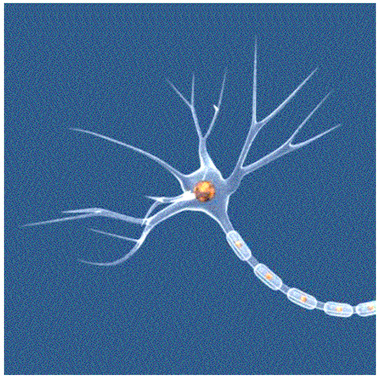
Inspired "Artificial Neural Networks"



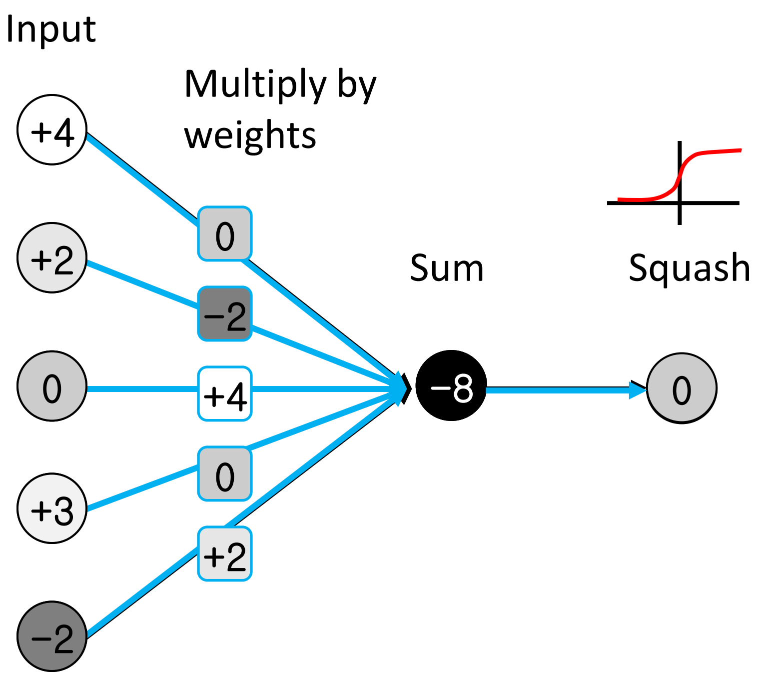
Neurons are cells that process chemical and electrical signals and transmit these signals to neurons and other types of cells

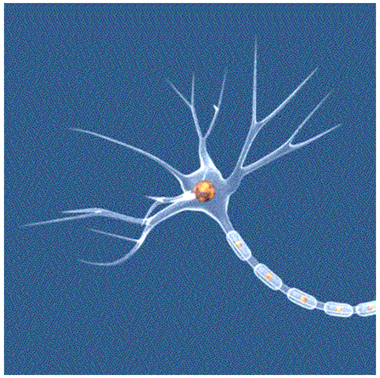
Logistic Unit as Artificial Neuron



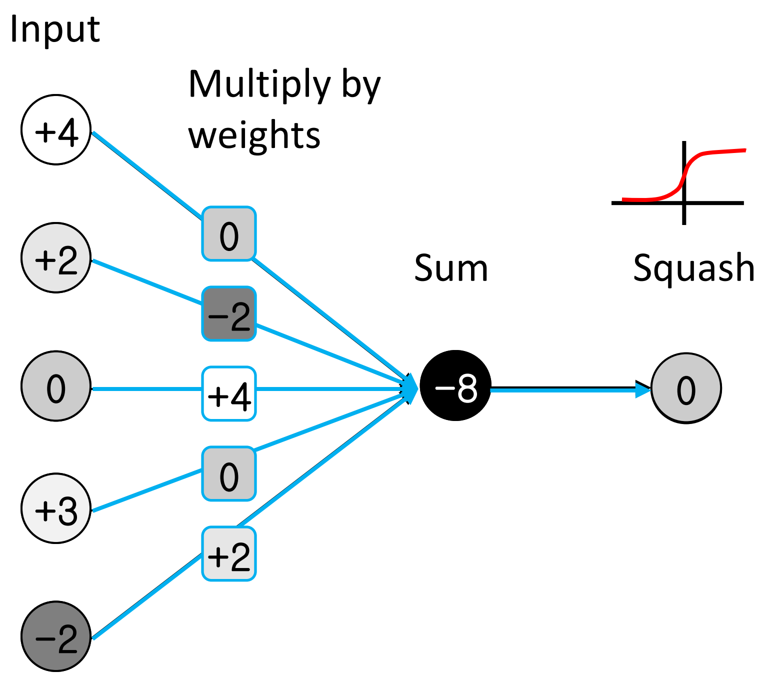


Logistic Unit as Artificial Neuron

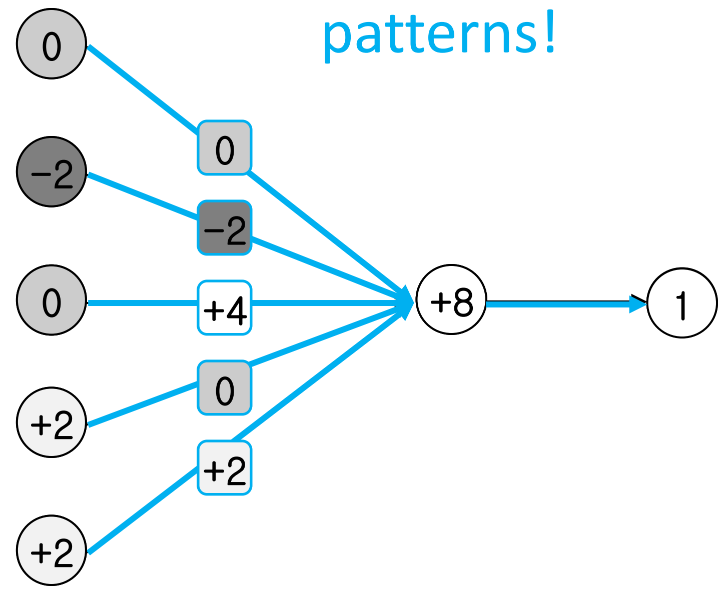




Logistic Unit as Artificial Neuron

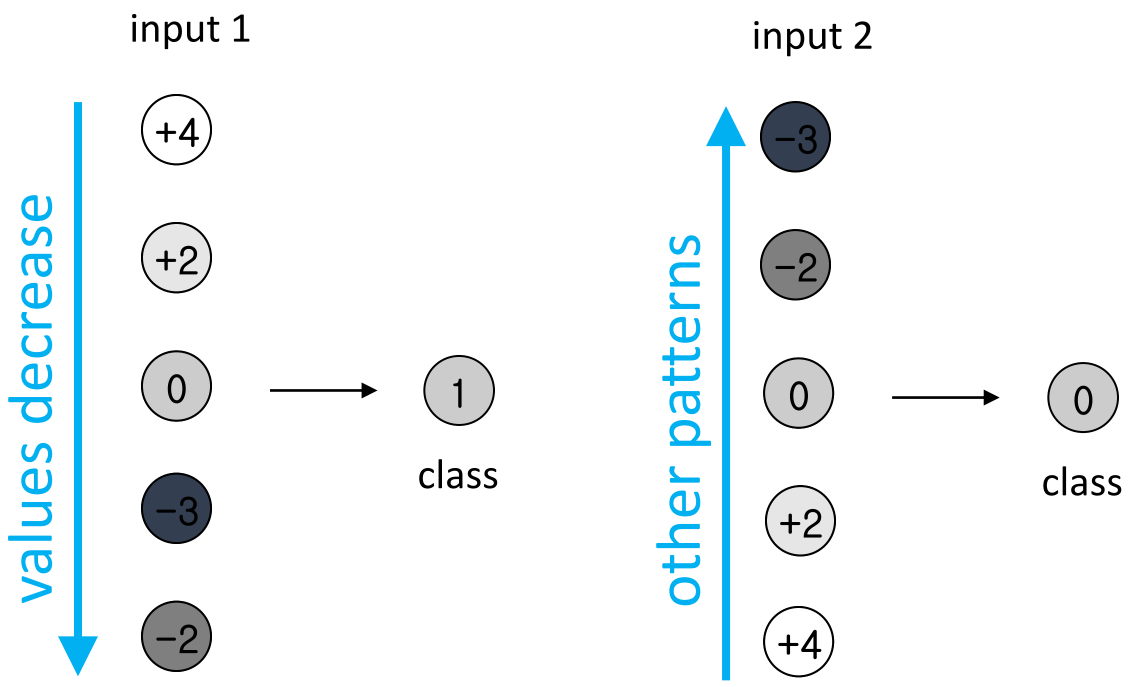


Neurons learn patterns!



Artificial Neuron Learns Patterns

- Classify input into class 0 or 1
- Teach neuron to predict correct class label
- Detect presence of a simple “feature”

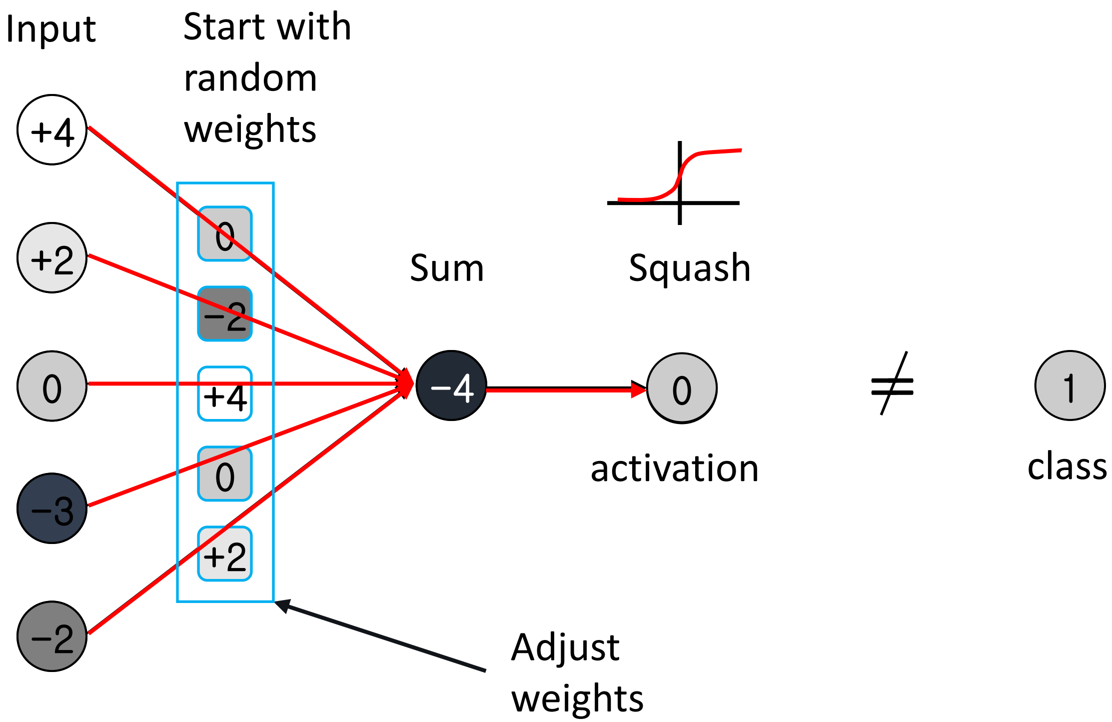


Example

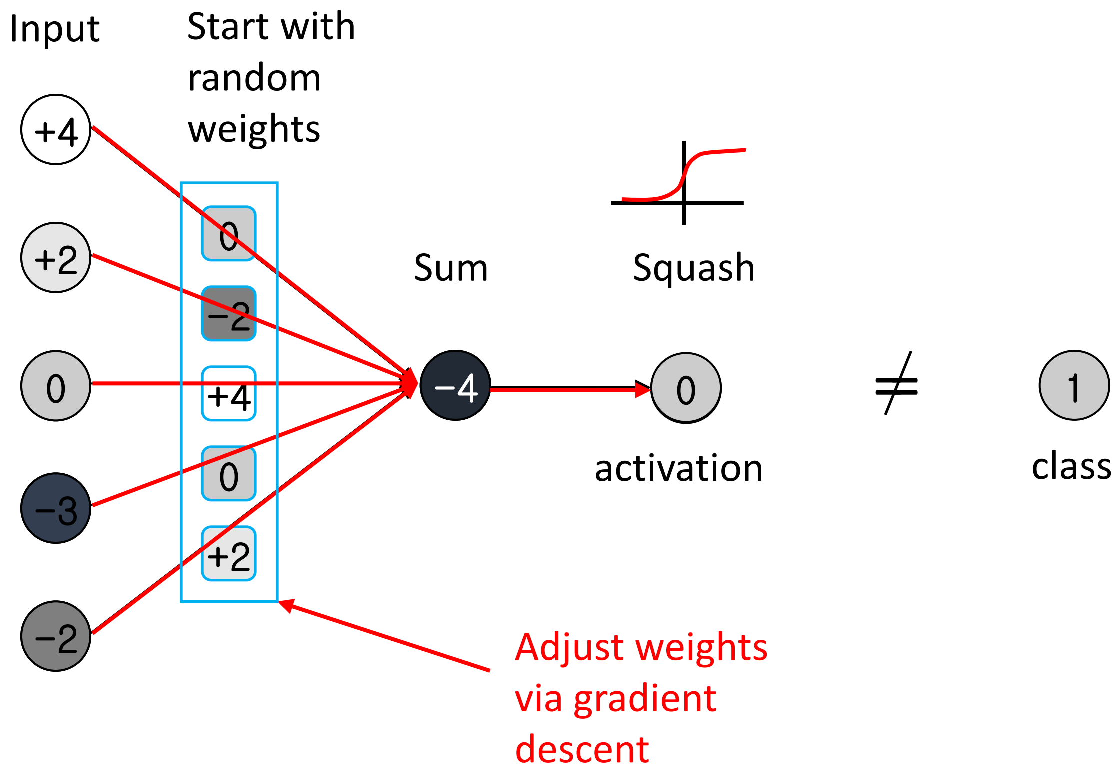
Neural Networks: Learning

Intuition

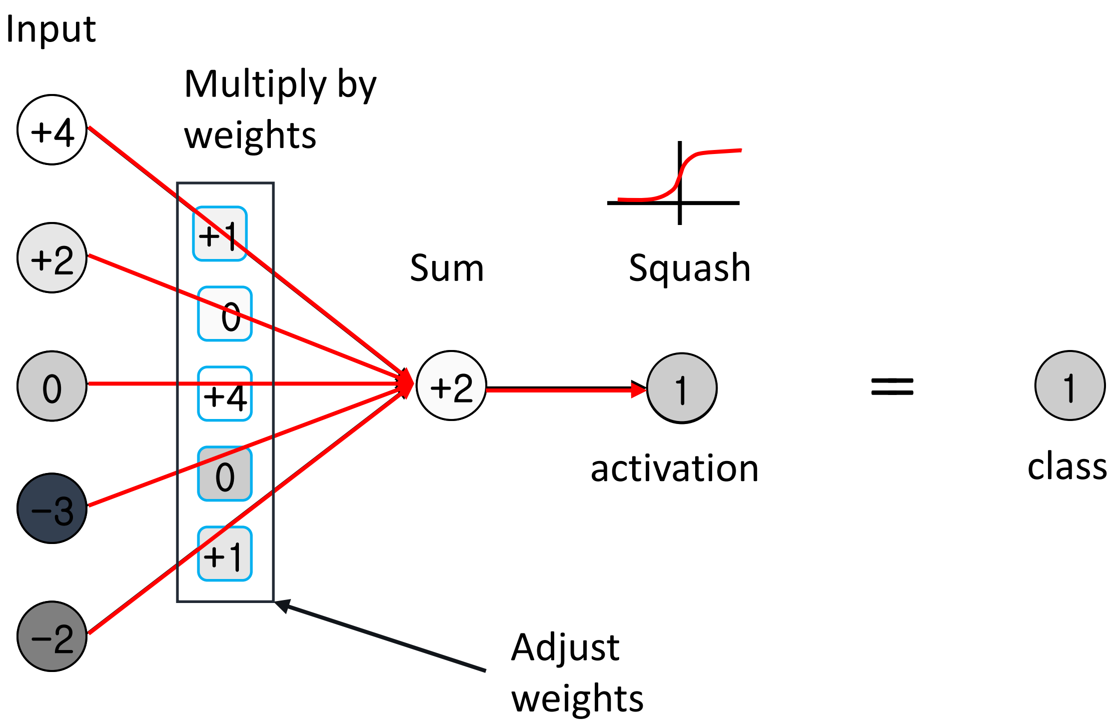
Artificial Neuron: Learning



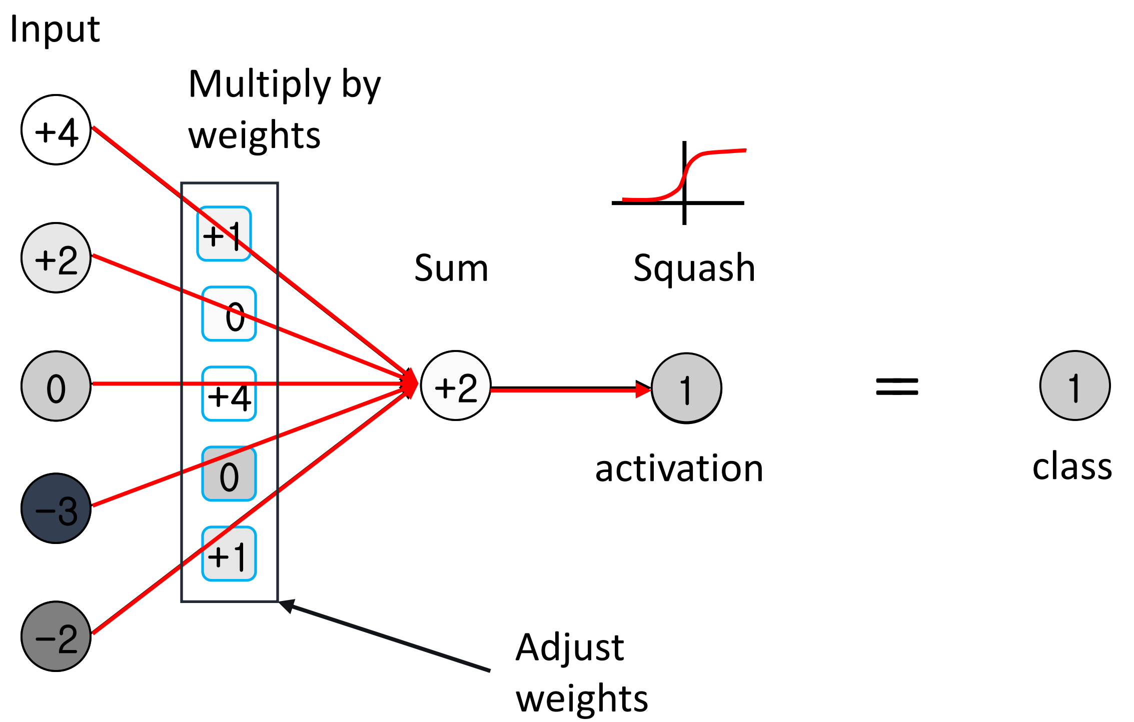
Artificial Neuron: Learning



Artificial Neuron: Learning



Artificial Neuron: Learning

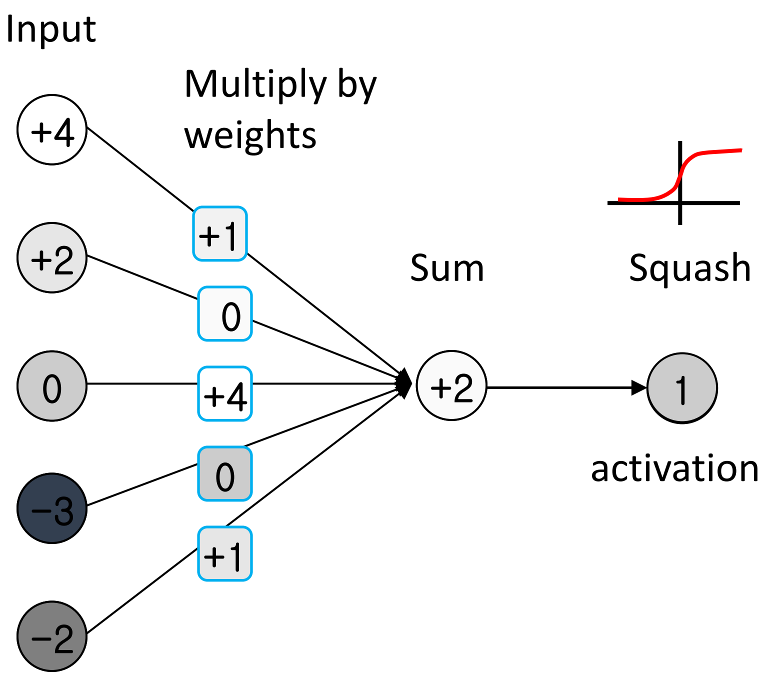


Forward propagation of information through a neuron

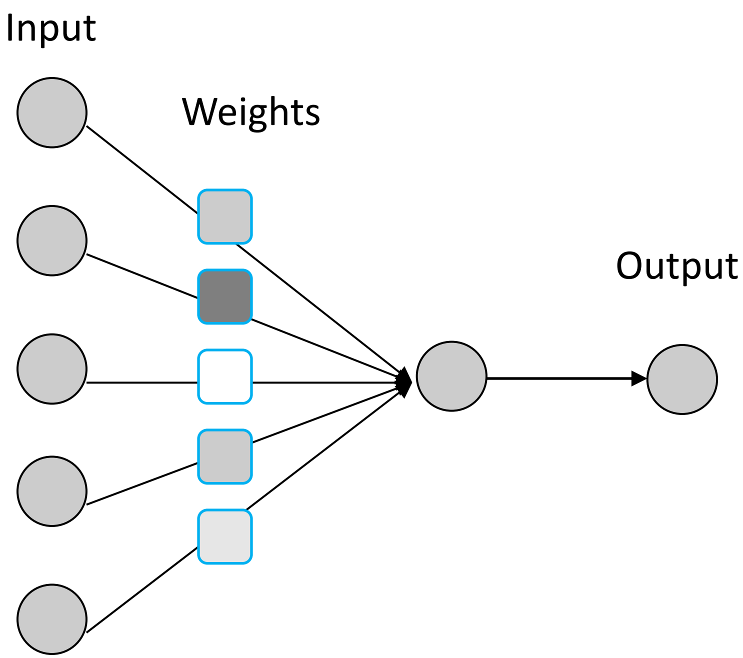
Neural Networks: Learning

Multi-layer network

Artificial Neuron: simplify

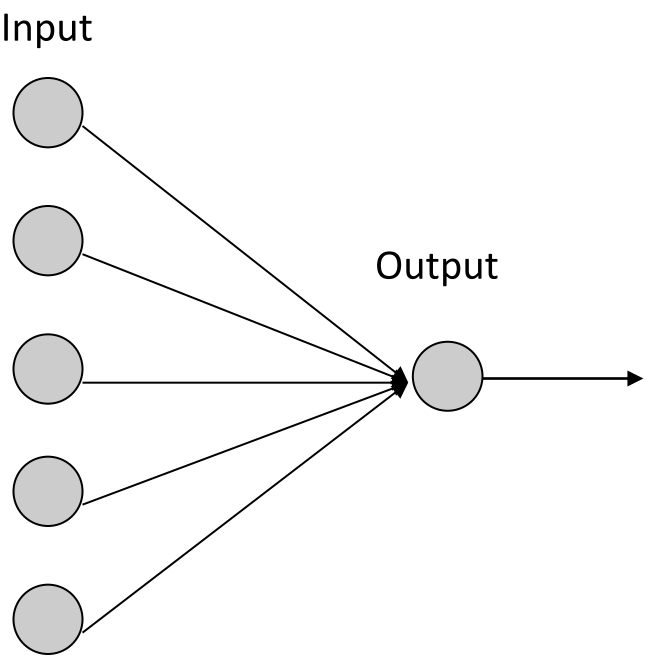


Artificial Neuron: simplify

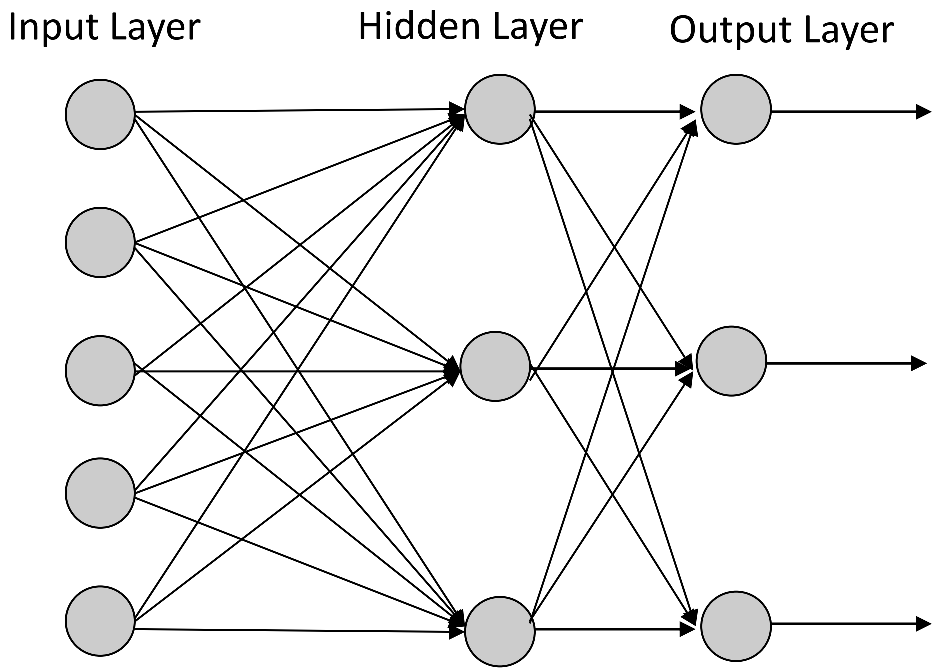


A single neuron is also called a perceptron

Artificial Neural Network



Single Neuron

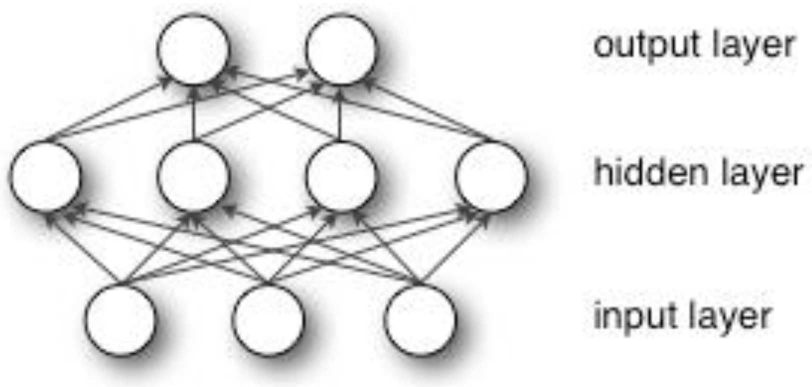


Neural Network (fully connected)

Deep Network: many hidden layers

Multi-layer perceptron (MLP)

- Just another name for a feed-forward neural network
- Logistic regression is a special case of the MLP with no hidden layer and sigmoid output.

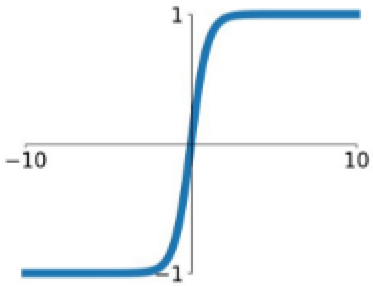


Other Non-linearities

- Also called activation functions

tanh

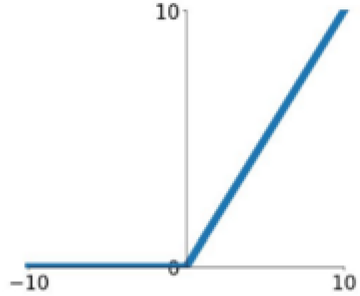
$\tanh(x)$



$$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

ReLU

$\max(0, x)$

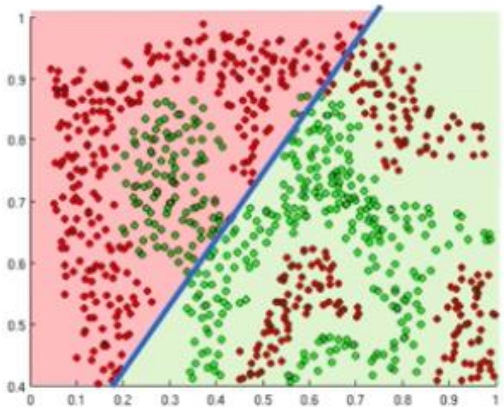


$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$

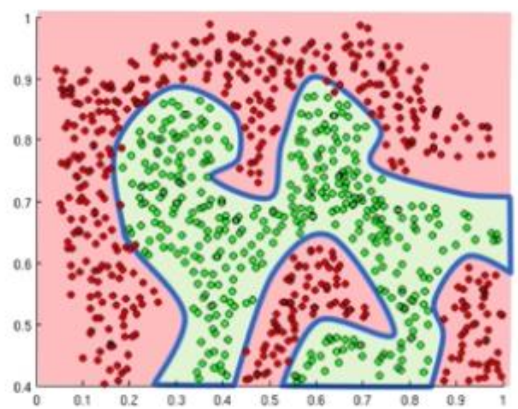
https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

Importance of Non-linearities

The purpose of activation functions is to ***introduce non-linearities*** *into the network*



Linear activation functions produce linear decisions no matter the network size



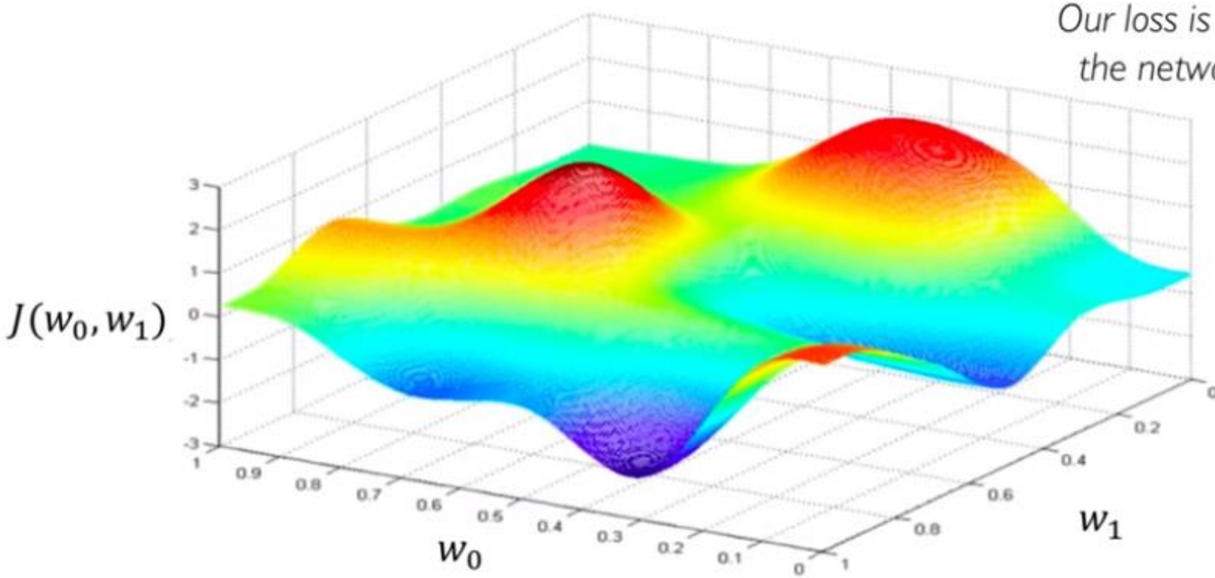
Non-linearities allow us to approximate arbitrarily complex functions

Loss Optimization

- A network learns the task defined in the Loss $\boldsymbol{J}(\boldsymbol{W})$.
- Neural network parameters are often referred to as weights $\boldsymbol{W}$.

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Remember:
Our loss is a function of the network weights!



Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$?
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$ ← ***Backpropagation***
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Not feasible to compute over all dataset

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Compute over a mini-batch

Gradient Descent

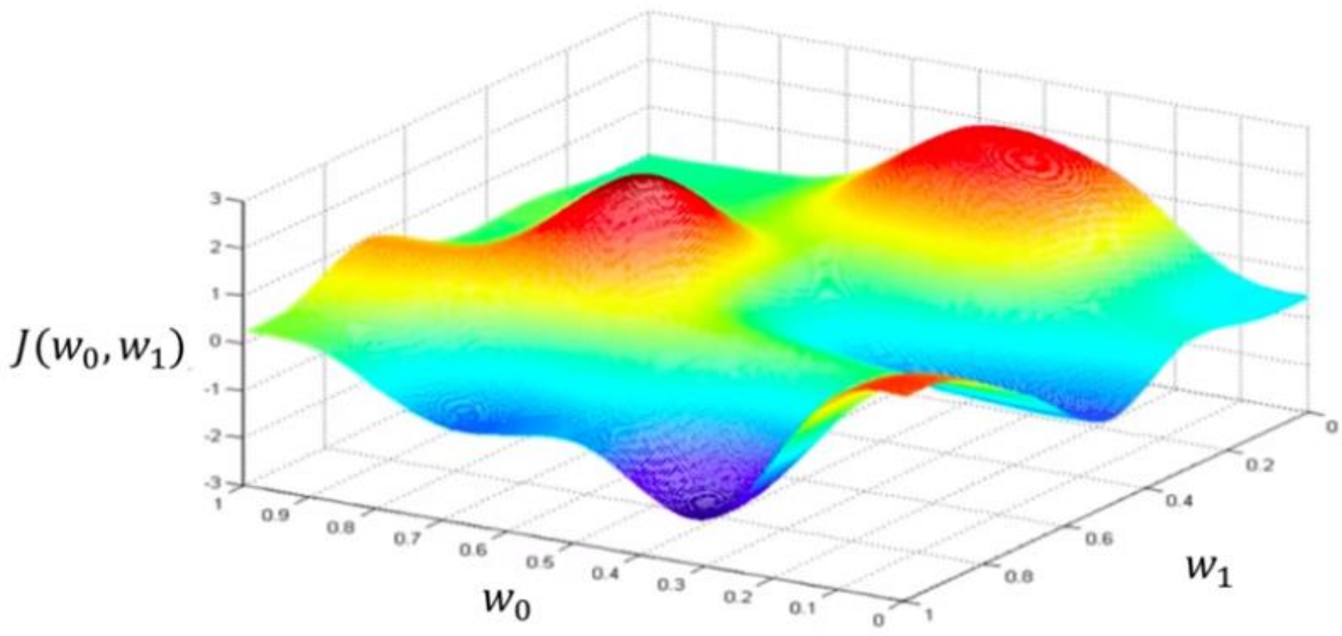
Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

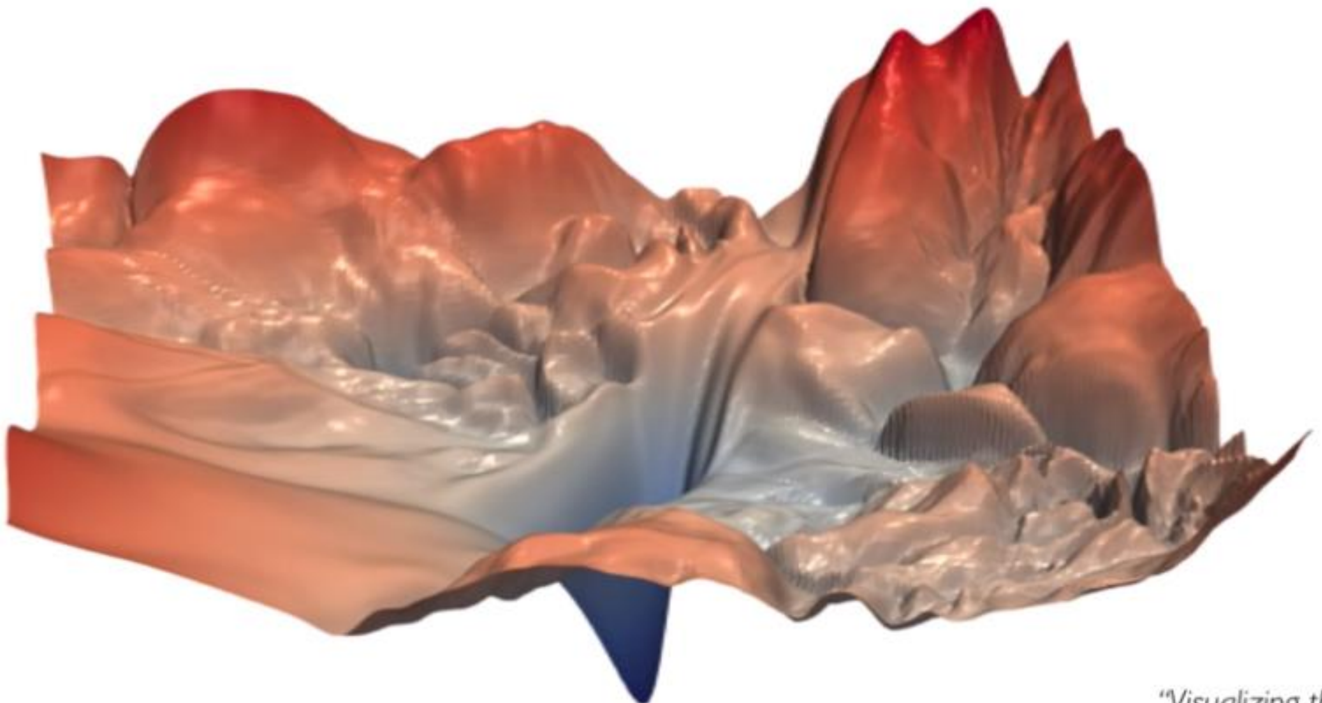
Compute over a mini-batch

Parallelization: Batches can be split onto multiple GPUs

Loss/Cost Function

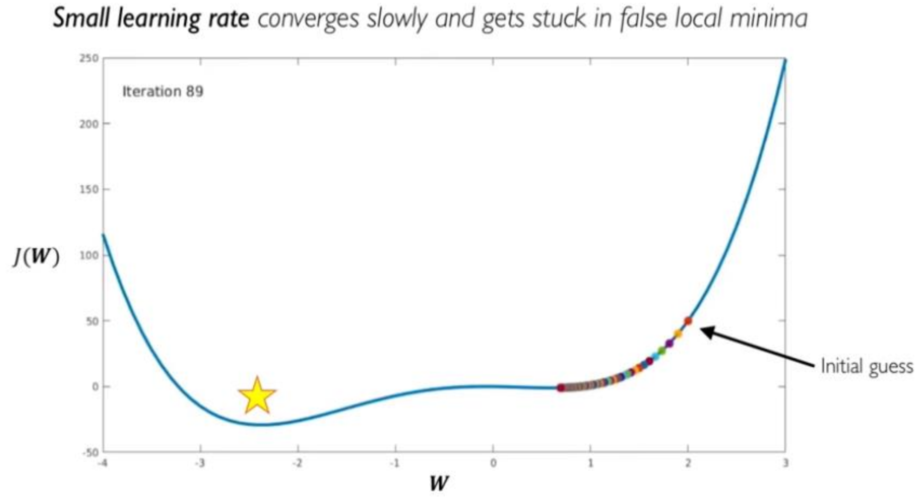


Landscape Visualization



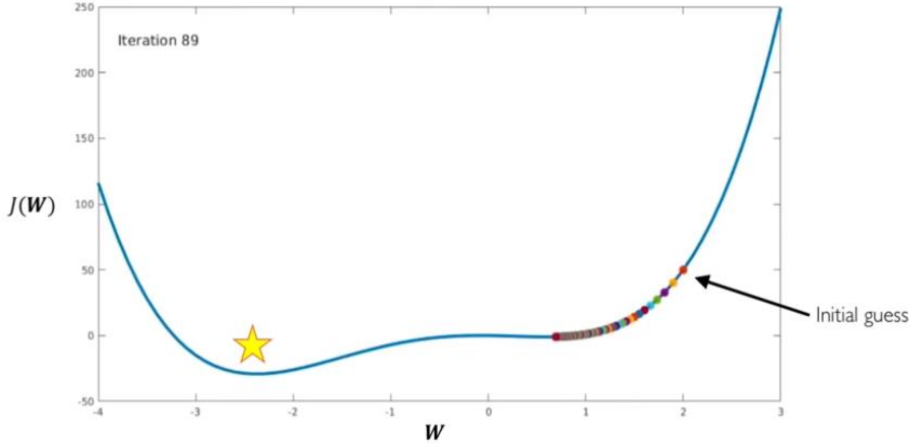
"Visualizing the loss landscape of neural nets". Dec 2017.

Setting the Learning Rate

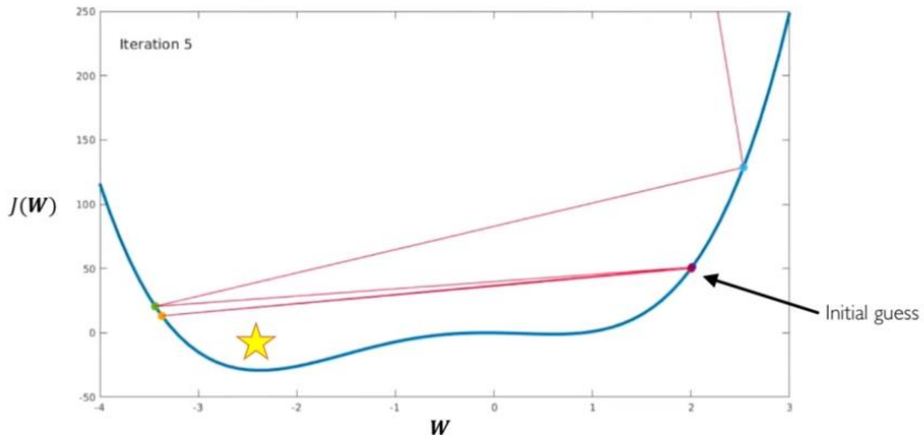


Setting the Learning Rate

Small learning rate *converges slowly and gets stuck in false local minima*



Large learning rates *overshoot, become unstable and diverge*



Setting the Learning Rate

- How to select the learning Rate?
  - Try several, and see which works best
  - Start with a learning rate, and change it ***adaptively*** as the model trains
  - Many are implemented in Neural Network Tools

Cost function

Neural network: $h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$

training error

$$J(\Theta) = \boxed{-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]}$$

$$\boxed{+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2}$$

regularization

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1-y_k^{(i)}) \log(1-h_\theta(x^{(i)})_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

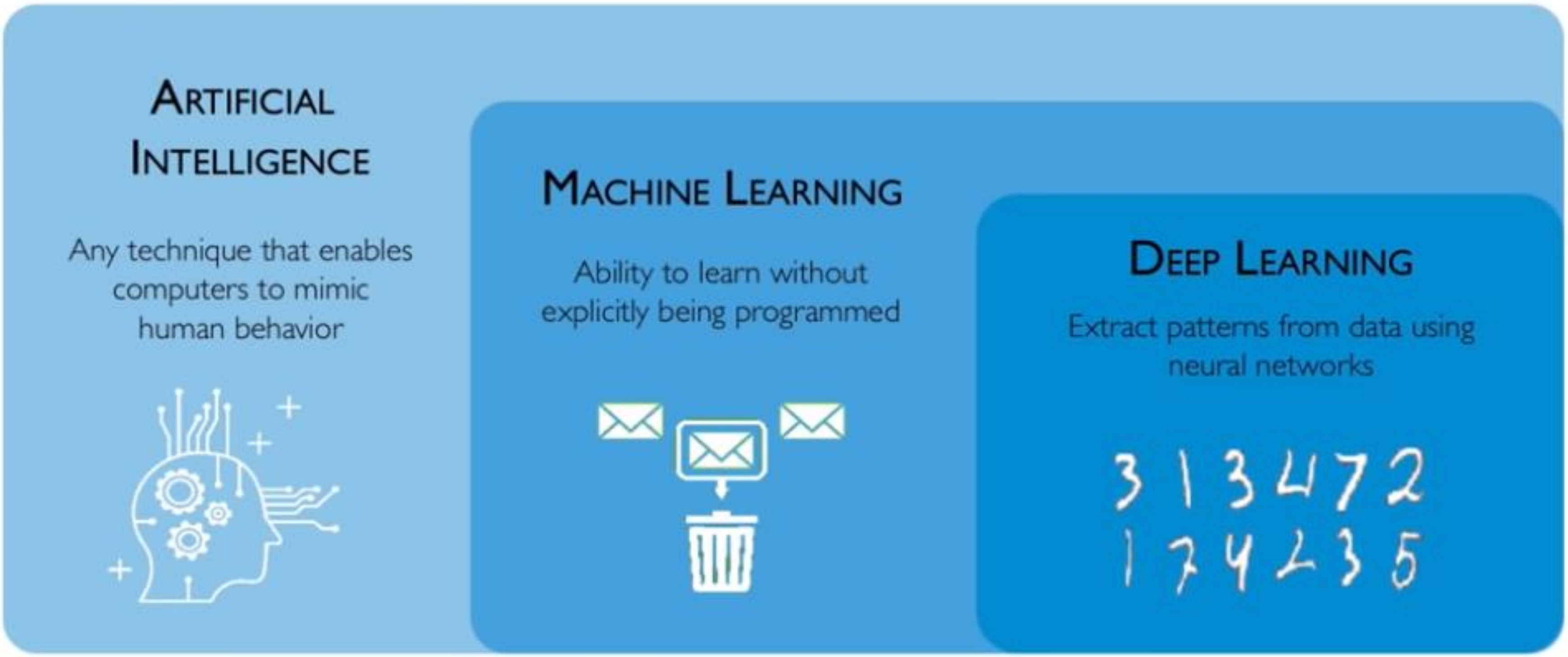
Need code to compute:

- $J(\Theta)$
- $\boxed{\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta)}$ *Backpropagation*

Deep Learning

Architectures

What is Deep Learning?

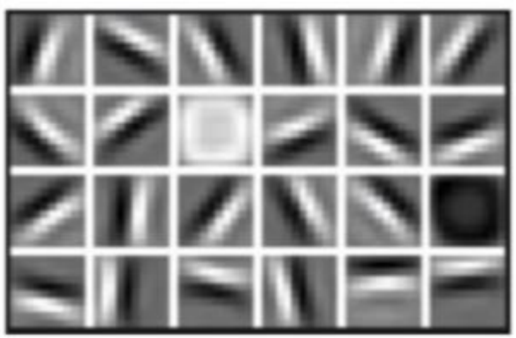


Why Deep Learning?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features

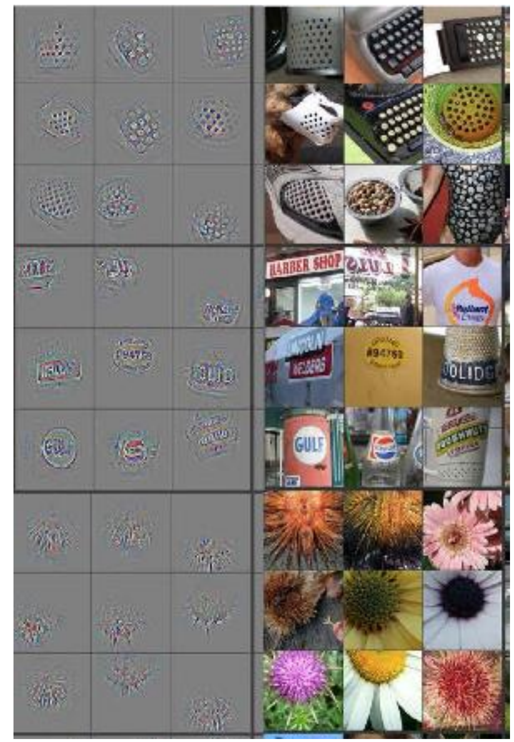
Eyes & Nose & Ears

High Level Features

Facial Structure

Why Deep Learning?
The Unreasonable Effectiveness of Deep Features

Maximal activations of pool_5 units [R-CNN]



conv_5 DeConv visualization
[Zeiler-Fergus]

Rich visual structure of features deep in hierarchy.

Why Now?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

⋮

Neural Networks date back decades, so why the resurgence?

1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

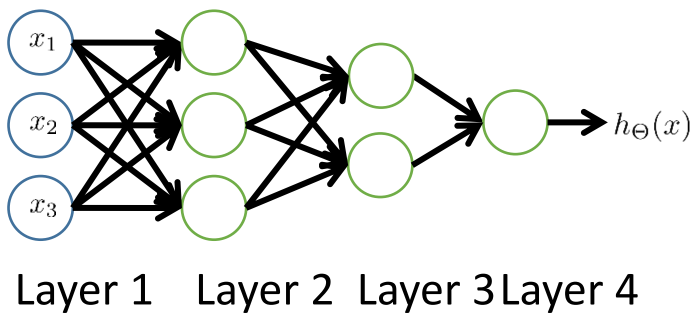
3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

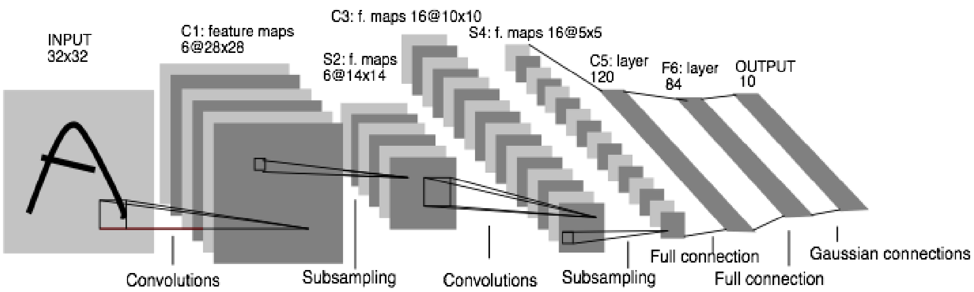
Network architectures

Feed-forward

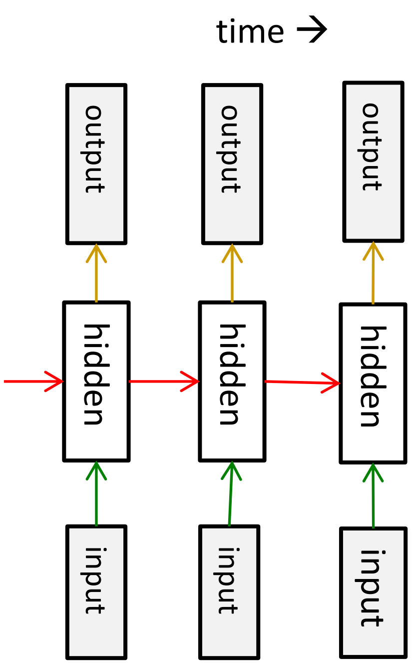
Fully connected



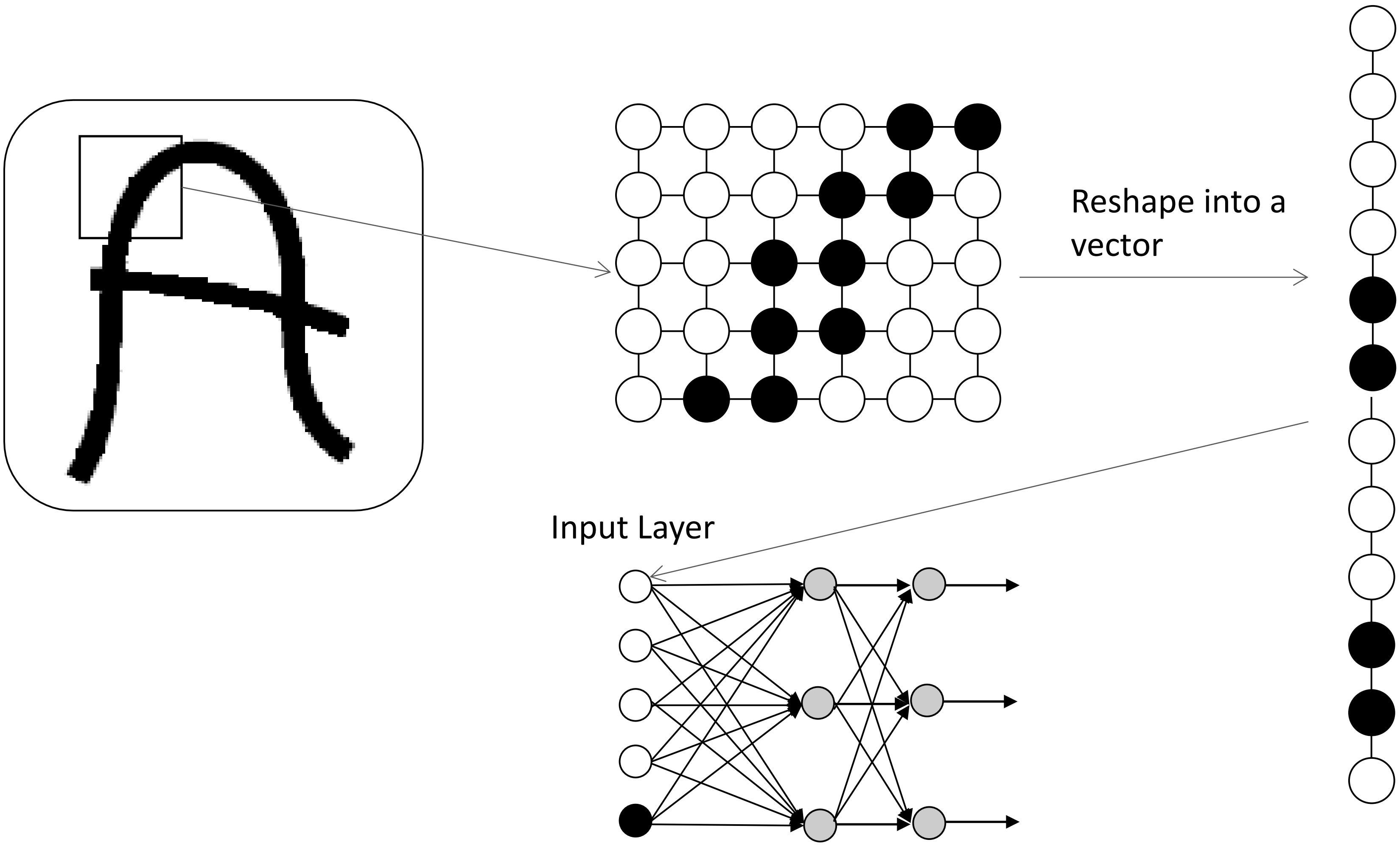
Convolutional



Recurrent



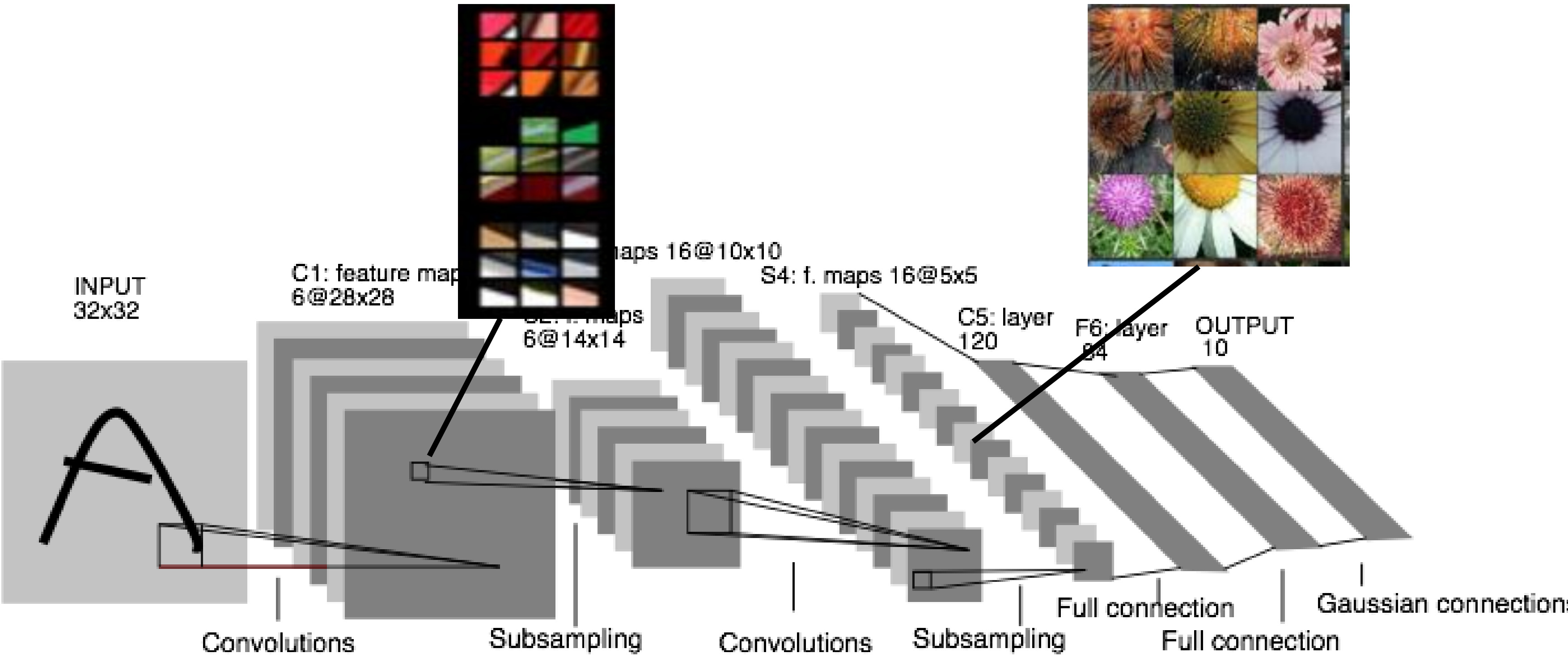
Fully Connected



Not ideal for representing images

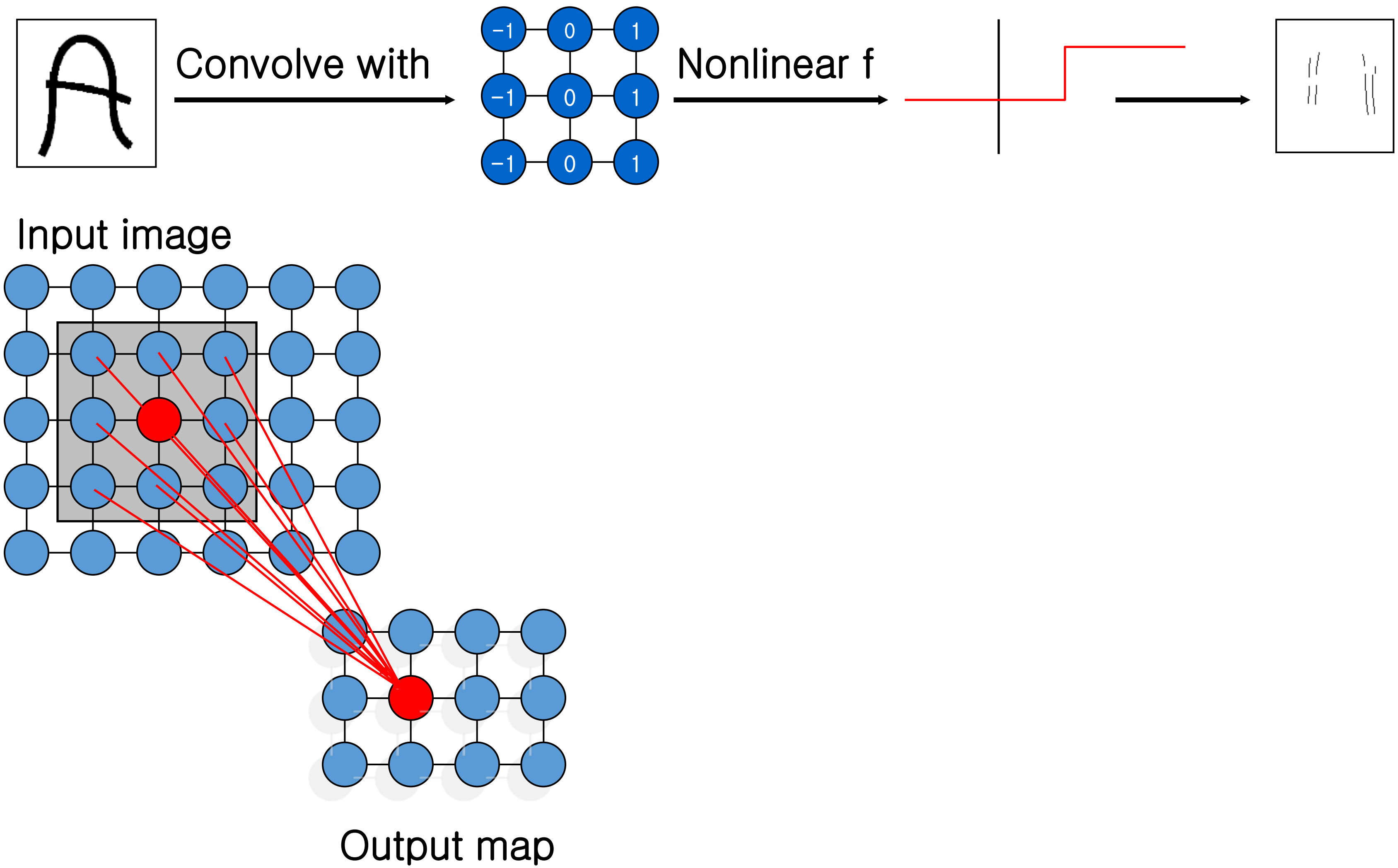
Convolutional Neural Network

A better architecture for 2d signals

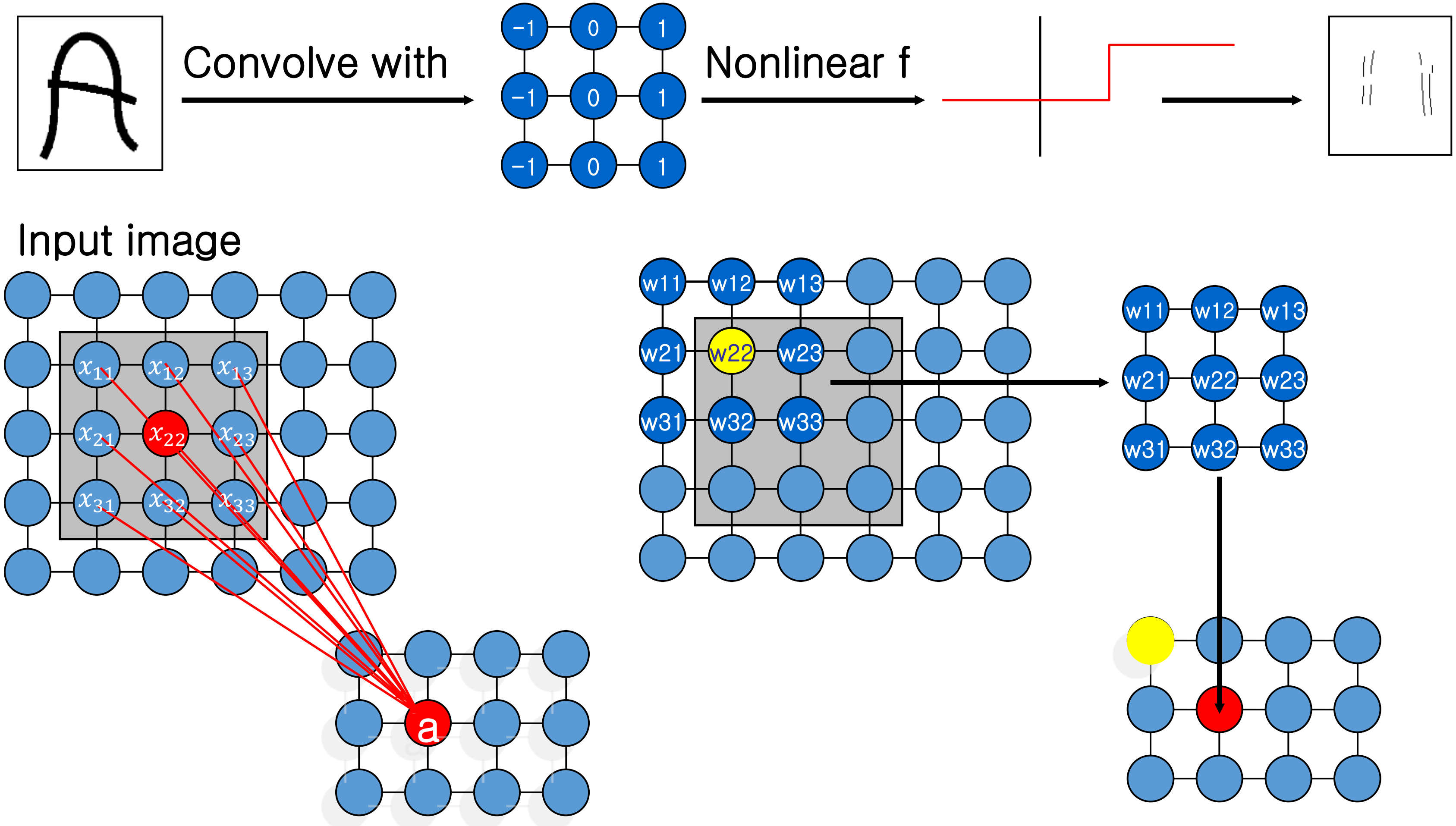


LeNet

Convolution layer in 2D



Convolution layer in 2D



Output map

Output looks like an image

$$a = f(w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + \cdots w_{33}x_{33})$$

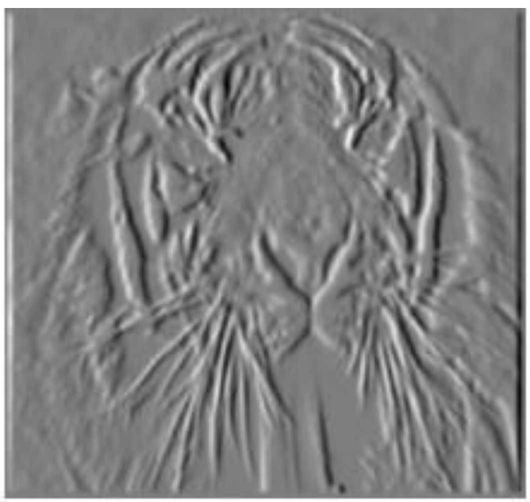
What weights correspond to these output maps?

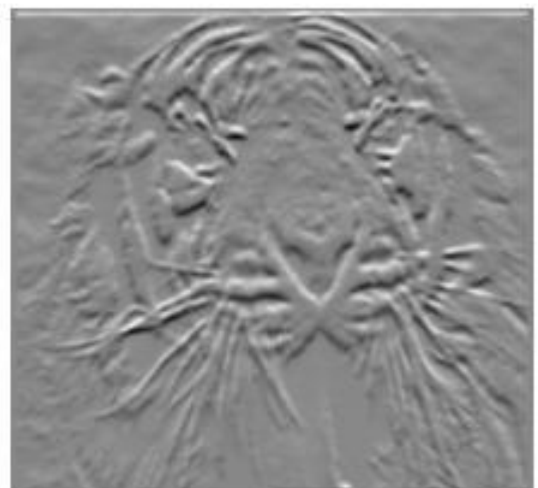
These are output maps before thresholding

Hint: filters look like the input they fire on

$$\frac{\partial f(x,y)}{\partial x}$$

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$



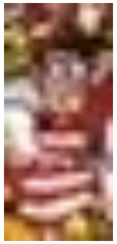


$$\frac{\partial f(x,y)}{\partial y}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Where is Waldo?

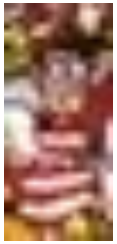
Input



filter

What will the output map look like?

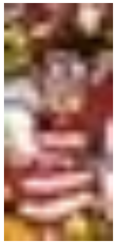
Input



filter

What will the output map look like?

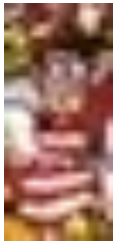
Output



filter

Here is Waldo

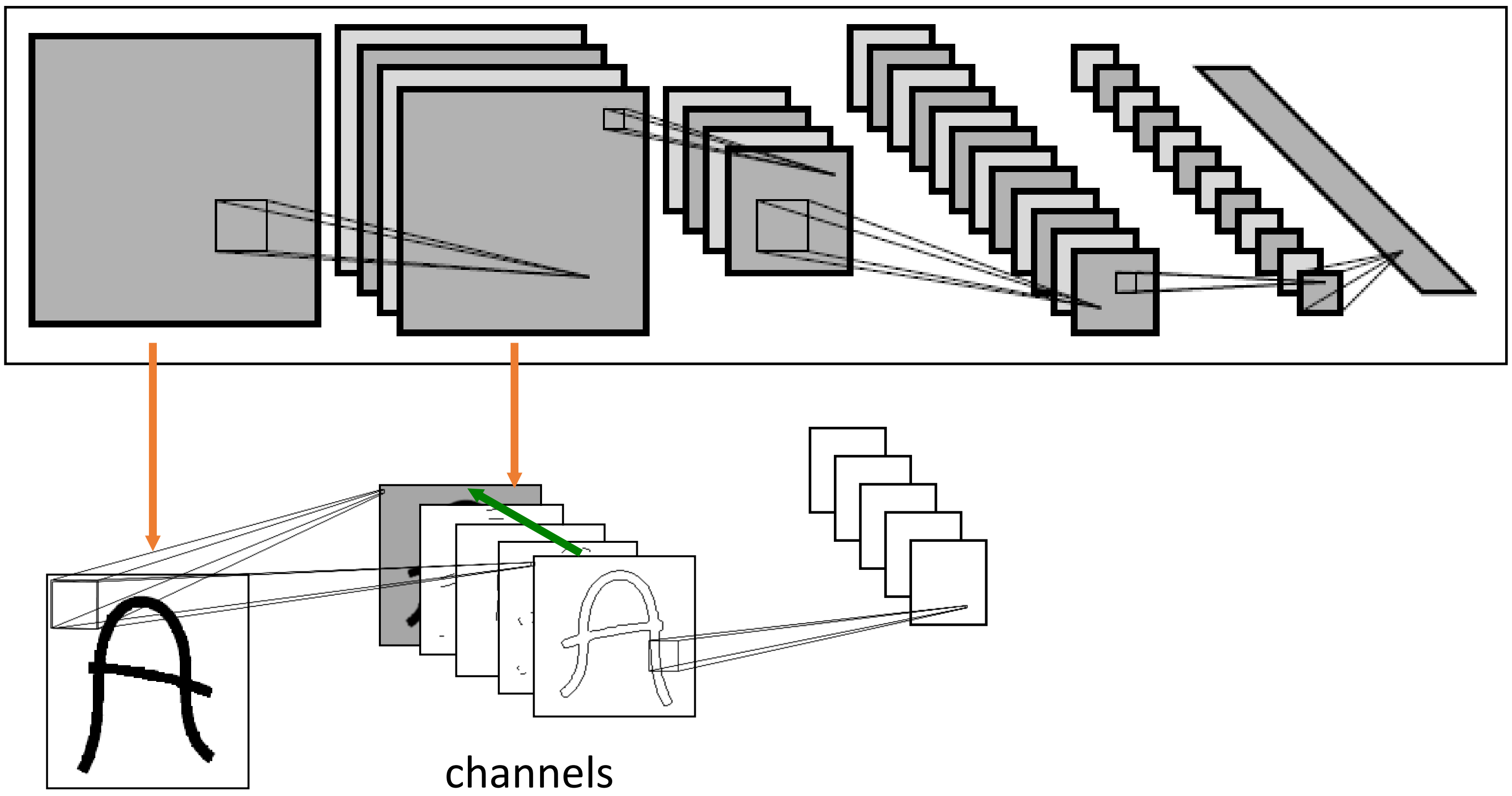
Input



filter

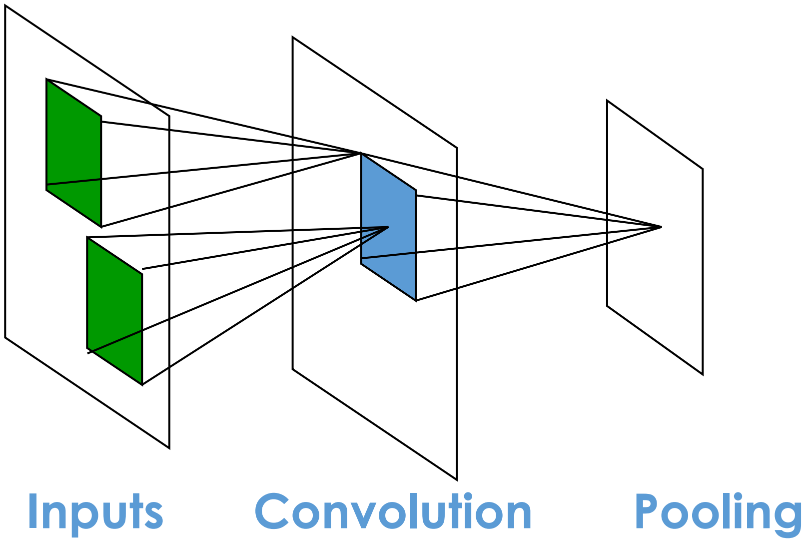
Stacking convolutional layers

- Each layer outputs multi-channel feature maps (like images)
- Next layer learns filters on previous layer's feature maps



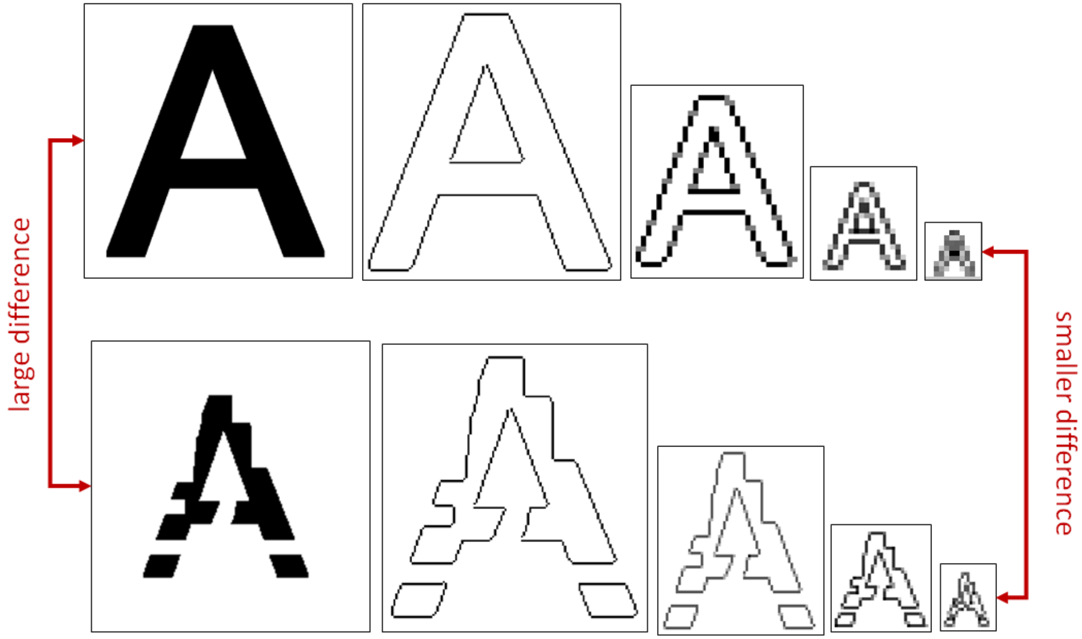
Pooling layers

- Convolution with stride > 1 reduces the size of the input
- Another way to downsize the feature map is with pooling
- A pooling layer subsamples the input in each sub-window
  - max-pooling: chose the max in a window
  - mean-pooling: take the average



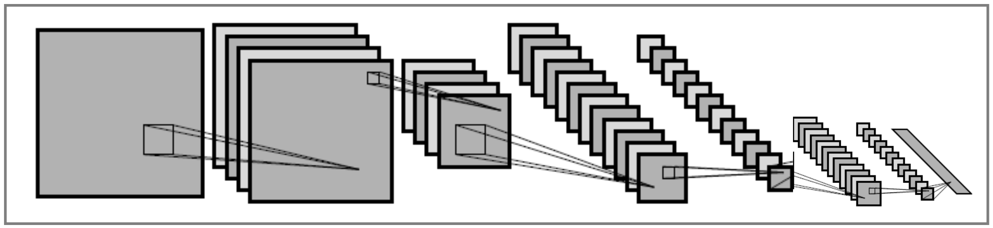
Pooling layer

- the pooling layers reduce the spatial resolution of each feature map
- Goal is to get a certain degree of shift and distortion invariance



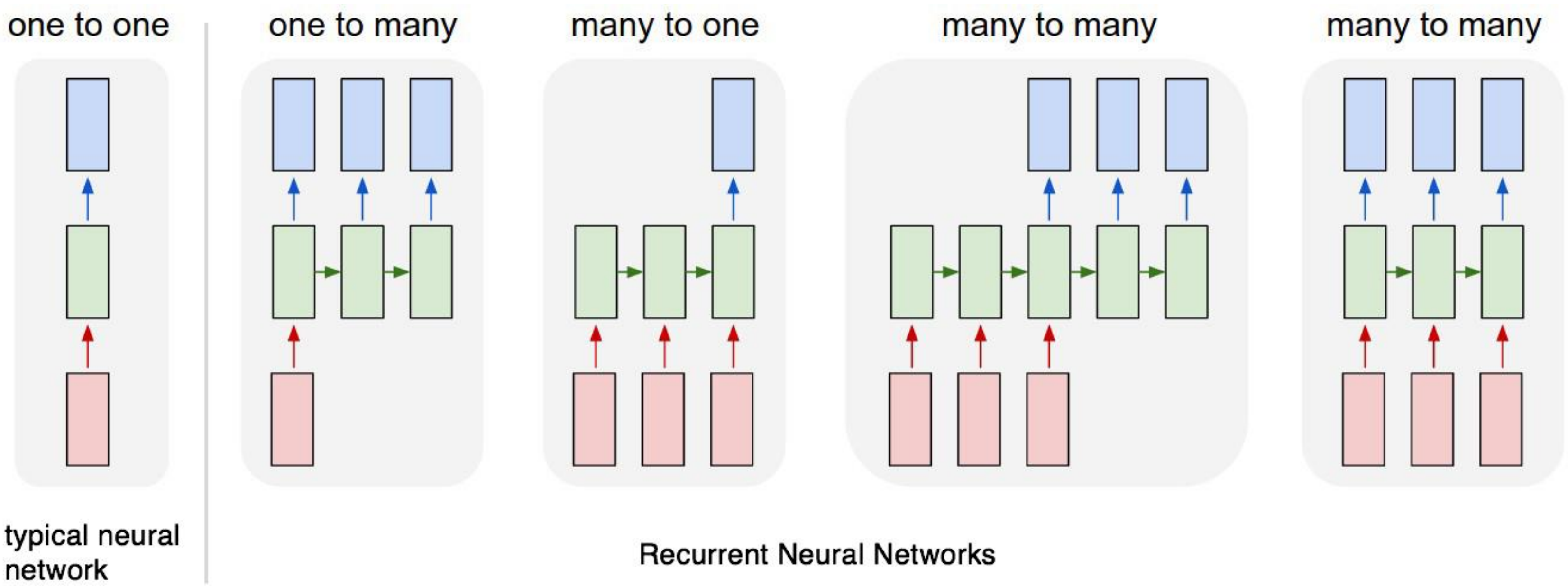
Testing the network

- Show top three most likely classes



http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Recurrent Neural Networks



https://towardsdatascience.com/language-translation-with-rnns-d84d43b40571

Recurrent Neural Networks

A young boy holding a baseball bat

A man riding a horse next to a building

Fei-Fei Li

- Ted Talk: https://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures?language=en
- Professor, Computer Science, Stanford University
- Co-Director of Stanford's Human-Centered AI Institute
- Previously Vice President at Google and Chief Scientist of AI/ML at Google Cloud
- Co-founder and chairperson of the national non-profit AI4ALL
- Online Deep Learning Course
- **"First, we teach them see, then they help us to see better."**