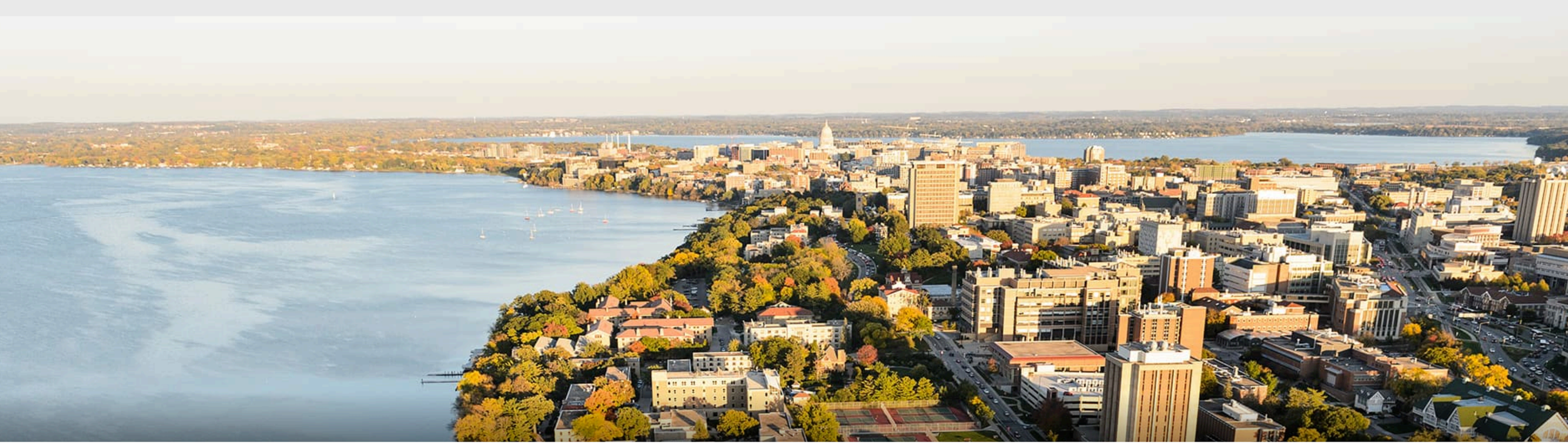


CS540 Introduction to Artificial Intelligence
(Deep) Neural Networks Summary

Sharon Yixuan Li
University of Wisconsin-Madison

March 30, 2021

How to classify

Cats vs. dogs?

Single-layer Perceptron

↓

Multi-layer Perceptron

↓

Training of neural networks

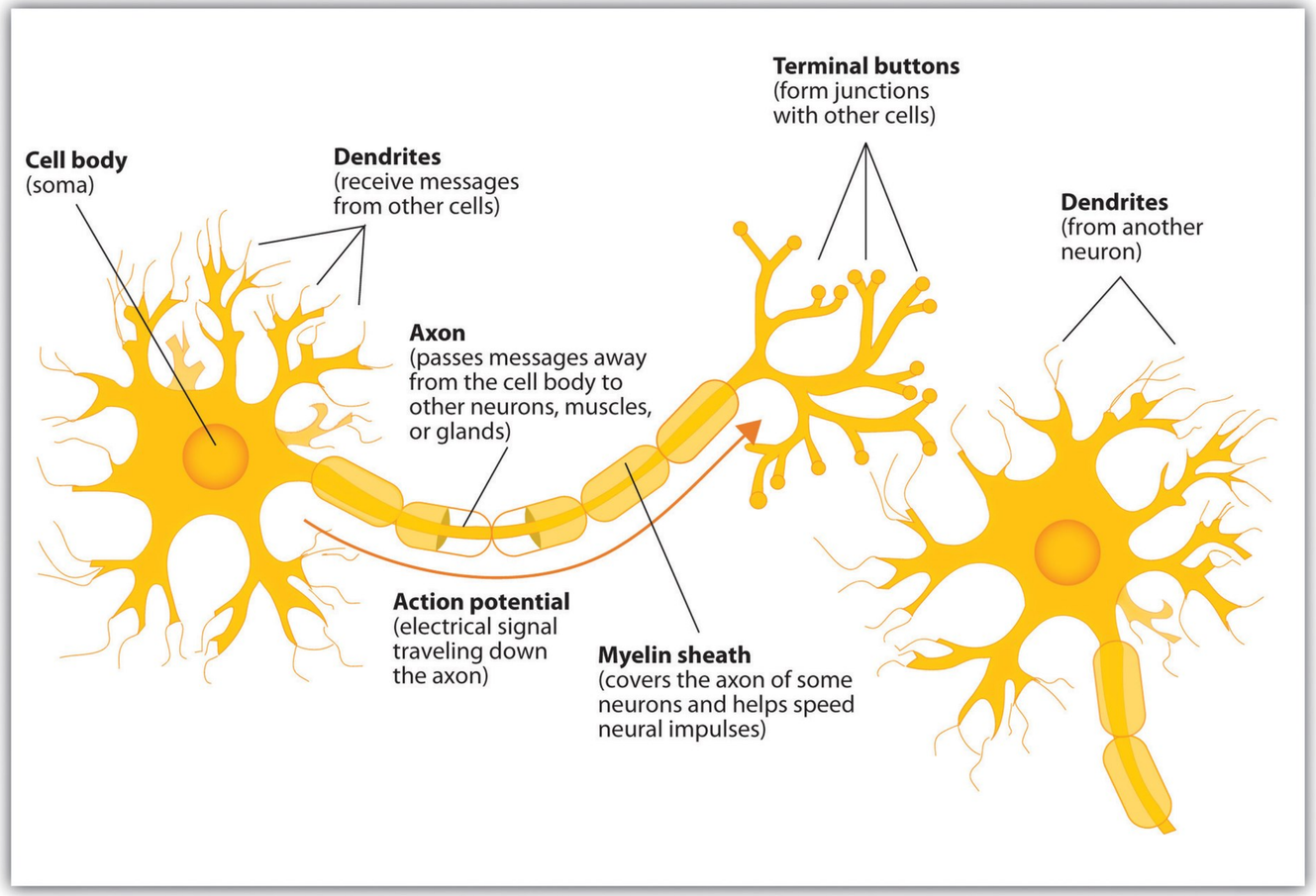
↓

Convolutional neural networks

Inspiration from neuroscience

- Inspirations from human brains
- Networks of simple and homogenous units (a.k.a **neuron**)

(wikipedia)



Perceptron

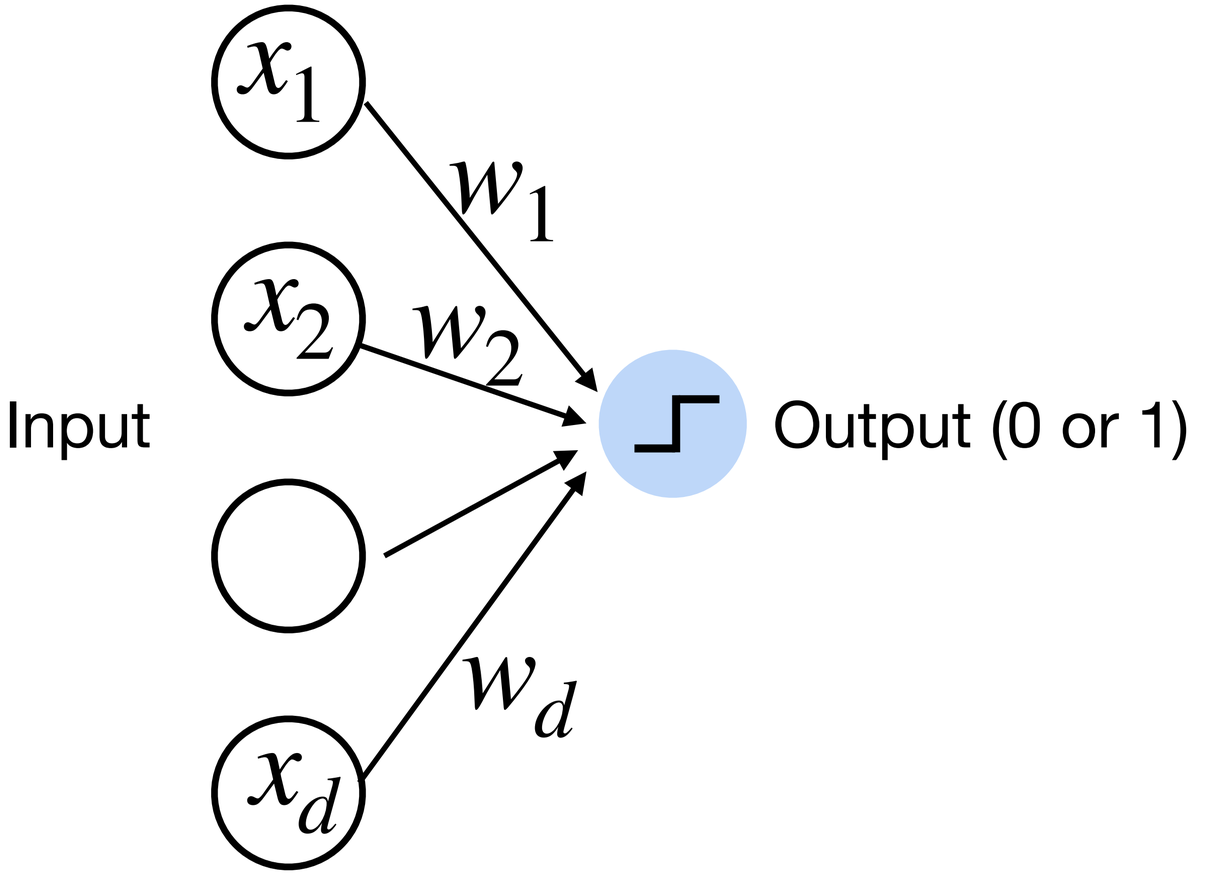
- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$o = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Activation function

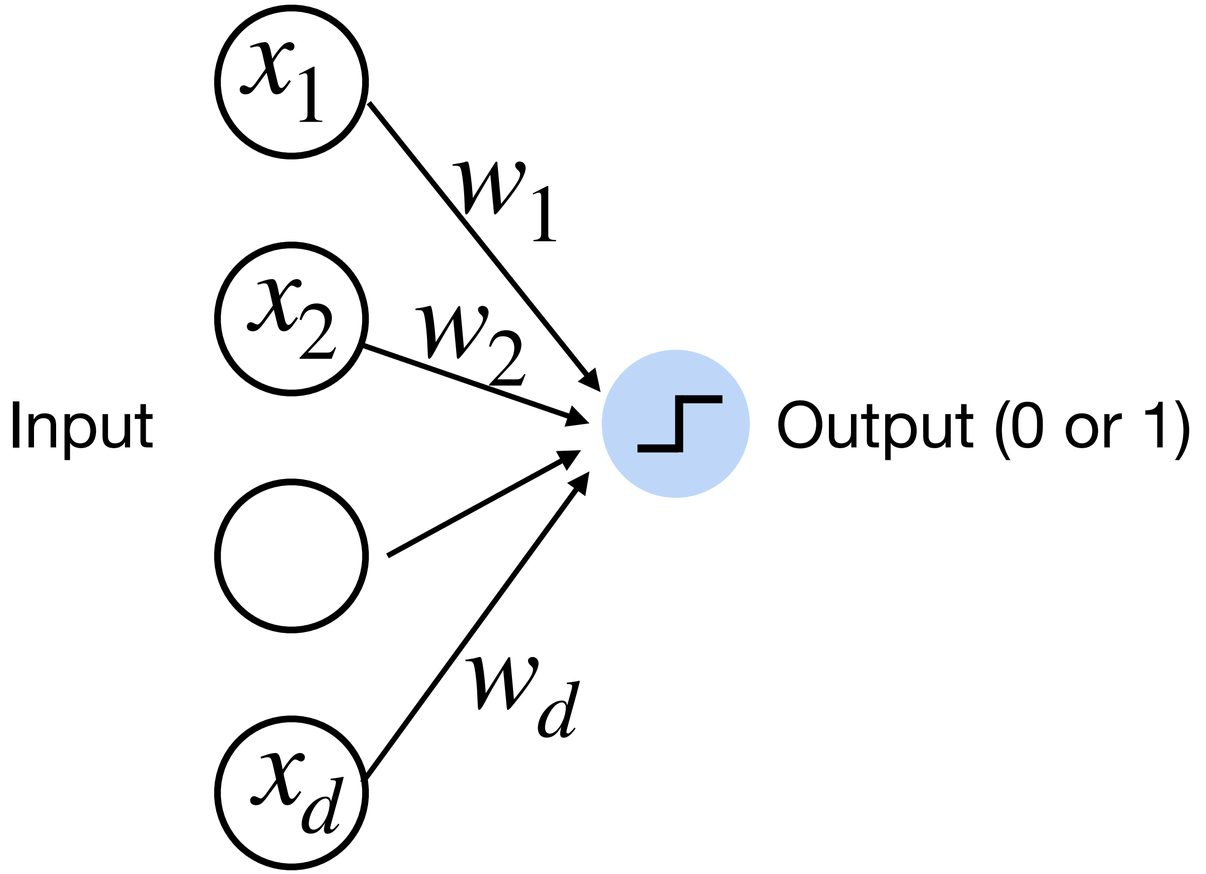
Cats vs. dogs?



Perceptron

- Goal: learn parameters $\mathbf{w} = \{w_1, w_2, \ldots, w_d\}$ and b to minimize the classification error

Cats vs. dogs?



Example 2: Predict whether a user likes a song or not

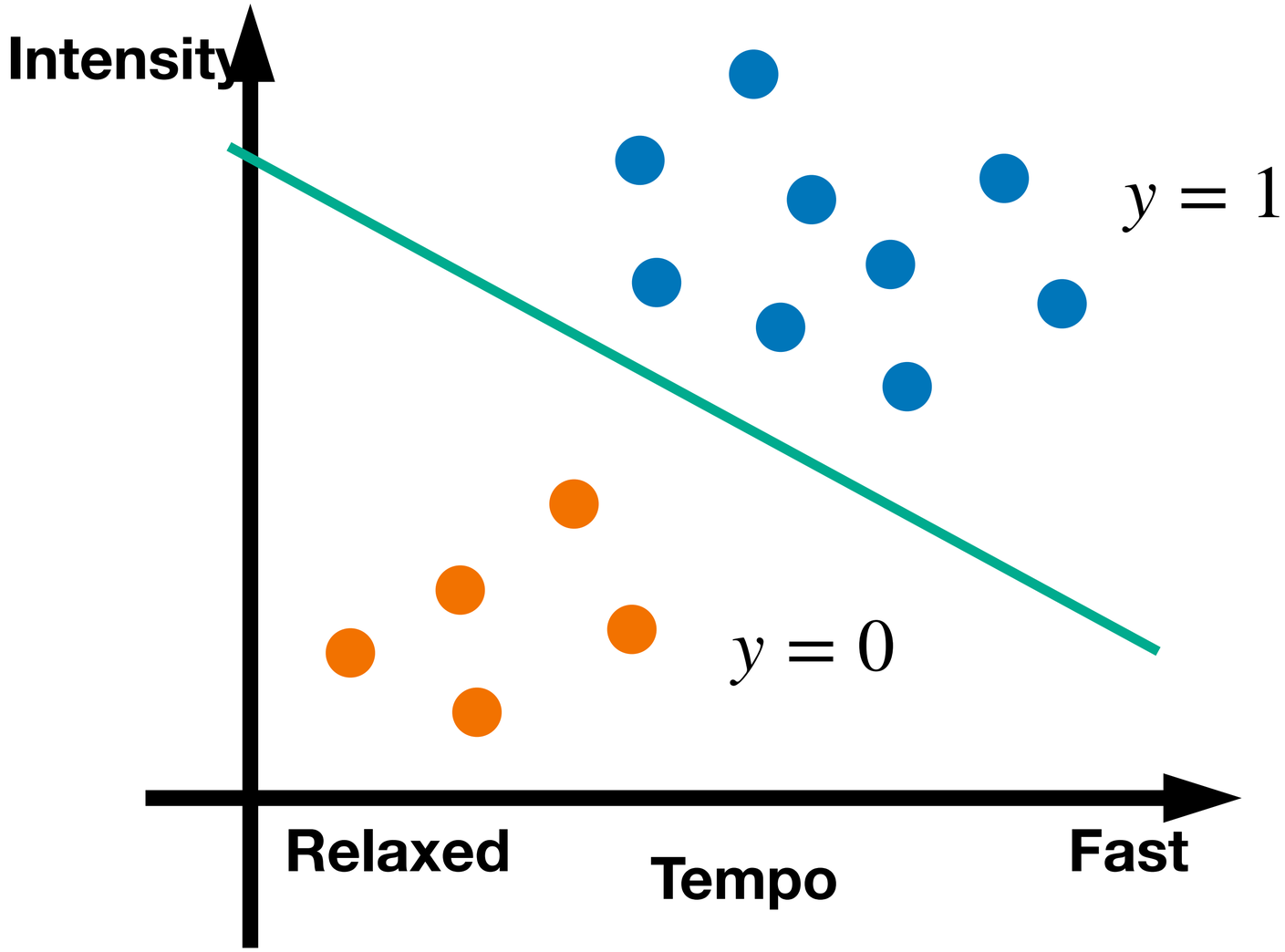
model

Example 2: Predict whether a user likes a song or not Using Perceptron

User Sharon

DisLike

Like



Learning logic functions using perceptron

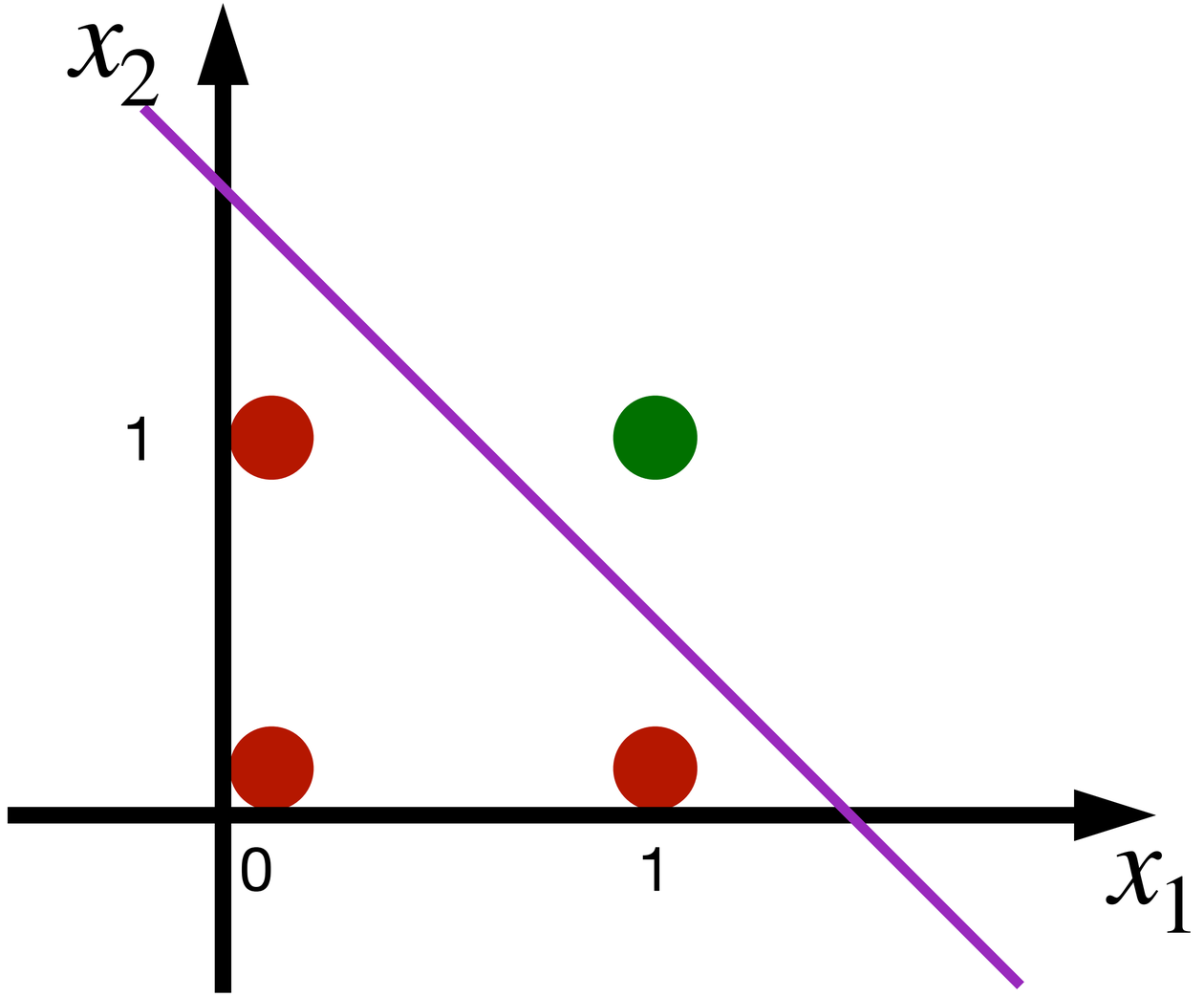
The perceptron can learn an AND function

$x_1 = 1, x_2 = 1, y = 1$

$x_1 = 1, x_2 = 0, y = 0$

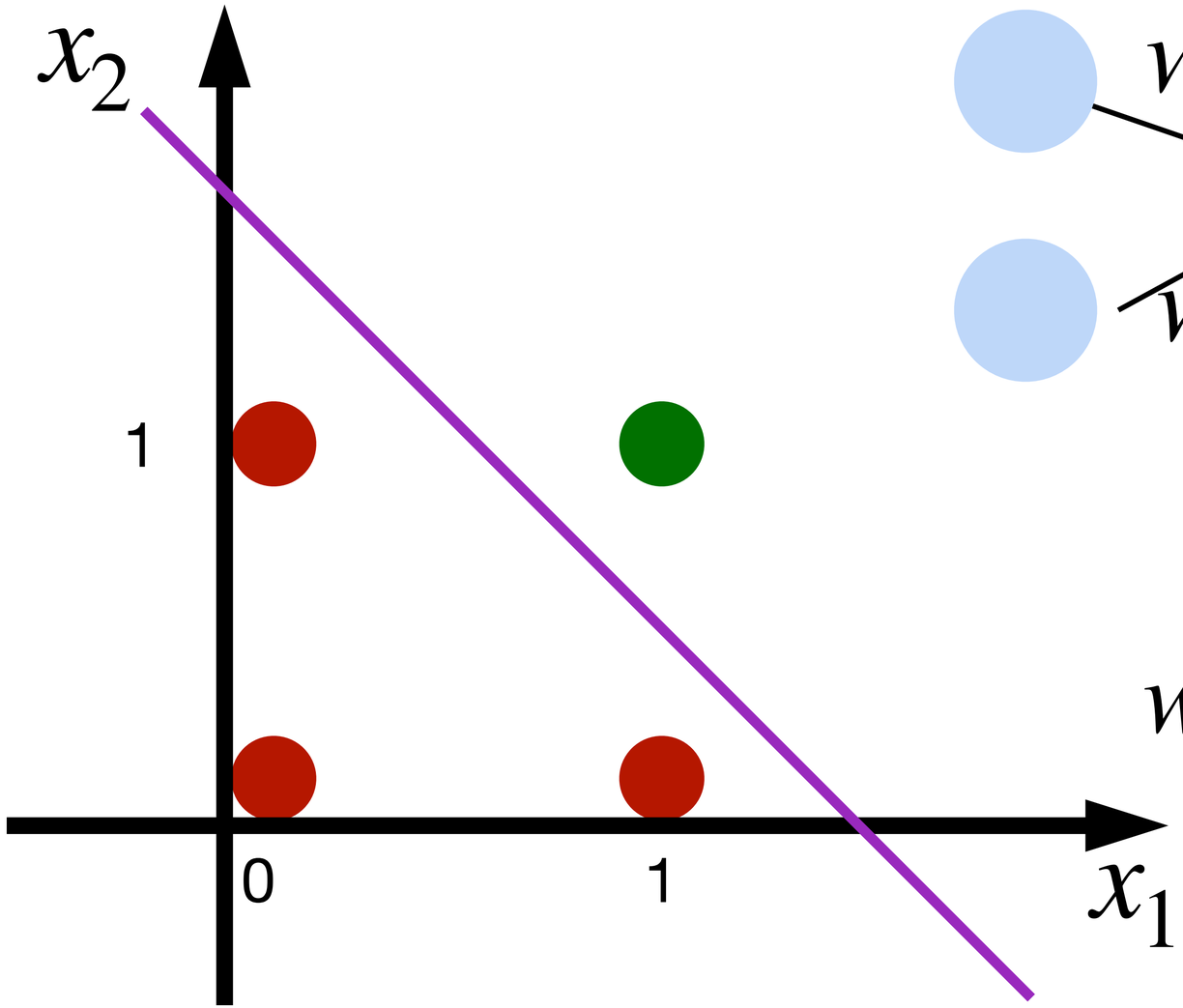
$x_1 = 0, x_2 = 1, y = 0$

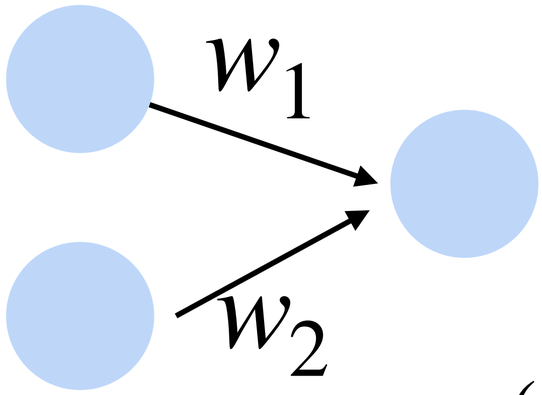
$x_1 = 0, x_2 = 0, y = 0$



Learning logic functions using perceptron

The perceptron can learn an AND function





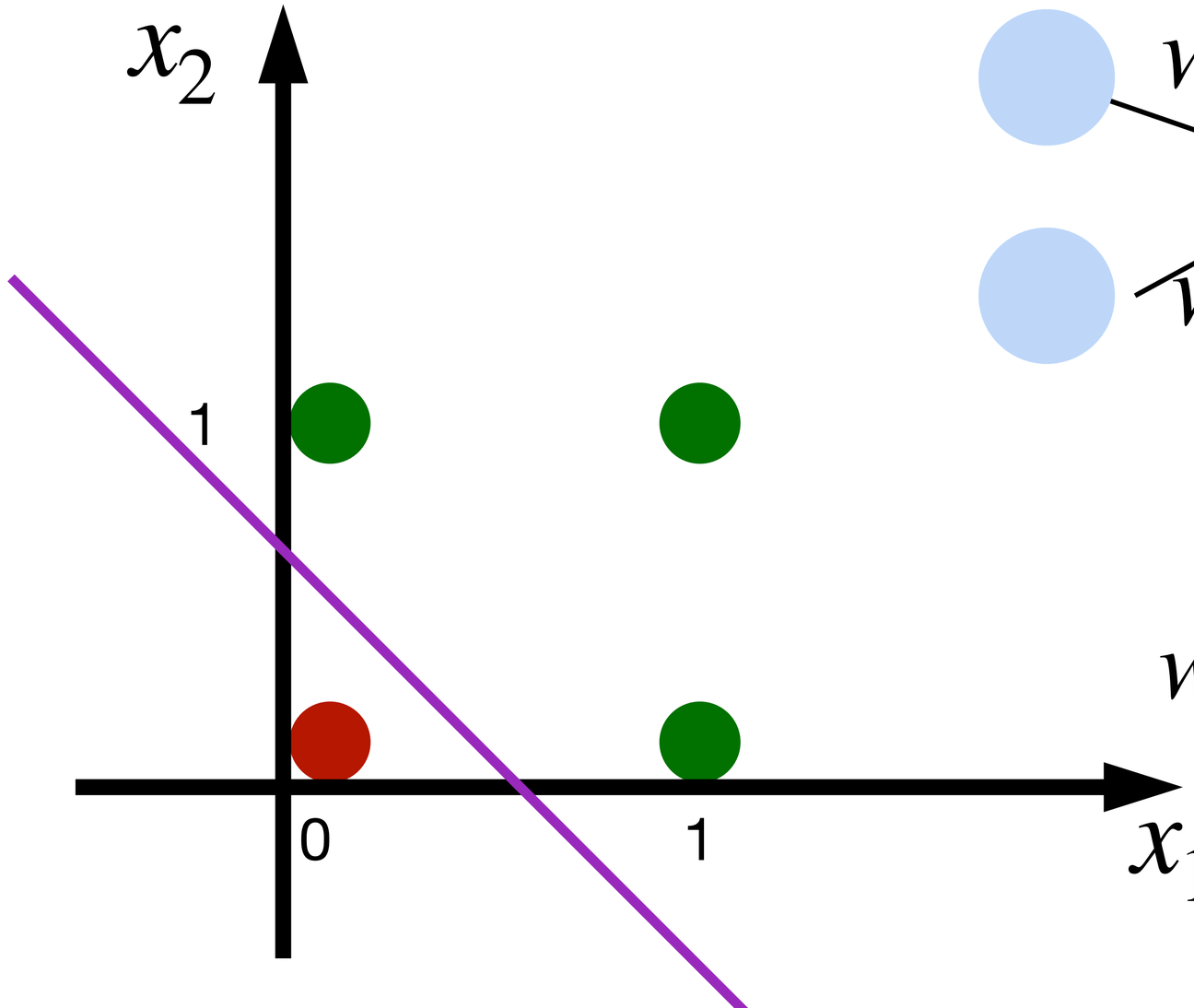
Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

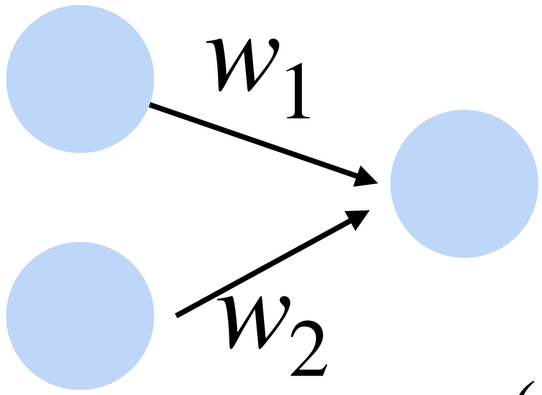
$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -1.5$$

Learning OR function using perceptron

The perceptron can learn an OR function





Output $\sigma(x_1 w_1 + x_2 w_2 + b)$

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$w_1 = 1, w_2 = 1, b = -0.5$$

XOR Problem (Minsky & Papert, 1969)

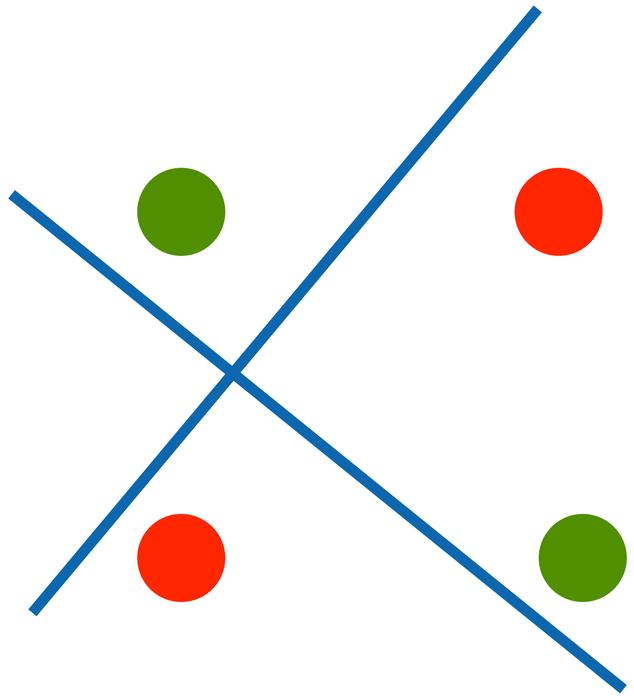
The perceptron cannot learn an XOR function
(neurons can only generate linear separators)

$$x_1 = 1, x_2 = 1, y = 0$$

$$x_1 = 1, x_2 = 0, y = 1$$

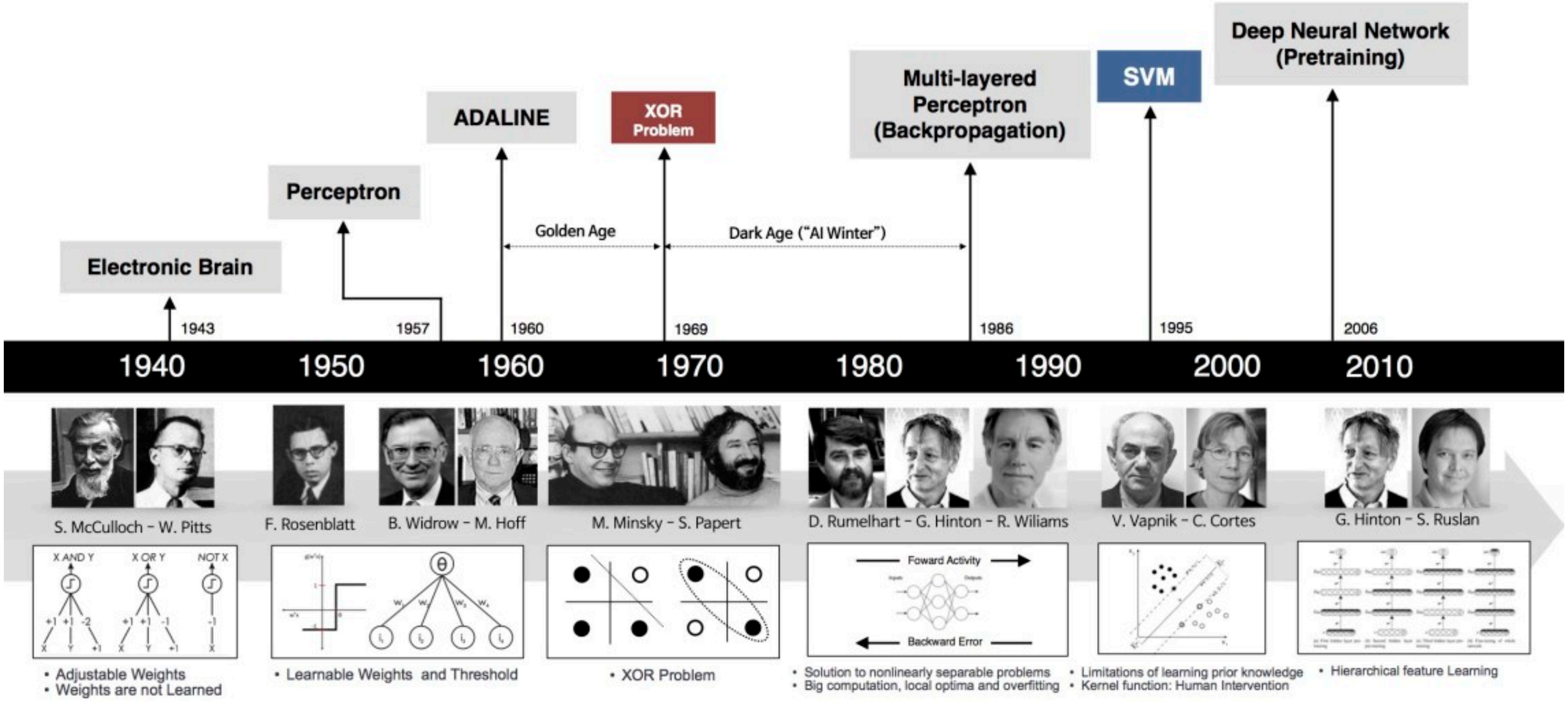
$$x_1 = 0, x_2 = 1, y = 1$$

$$x_1 = 0, x_2 = 0, y = 0$$

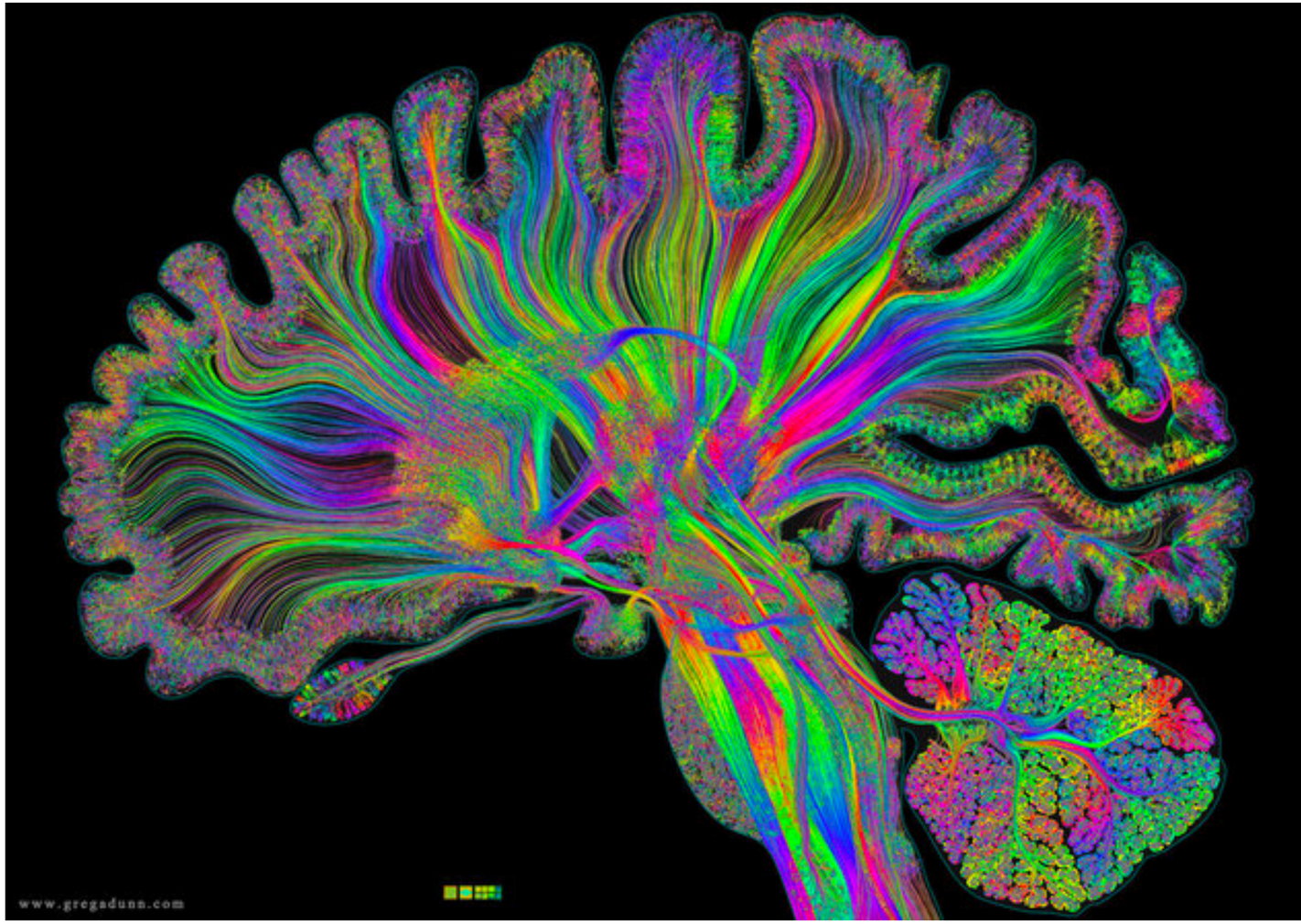


This contributed to the first AI winter

Brief history of neural networks



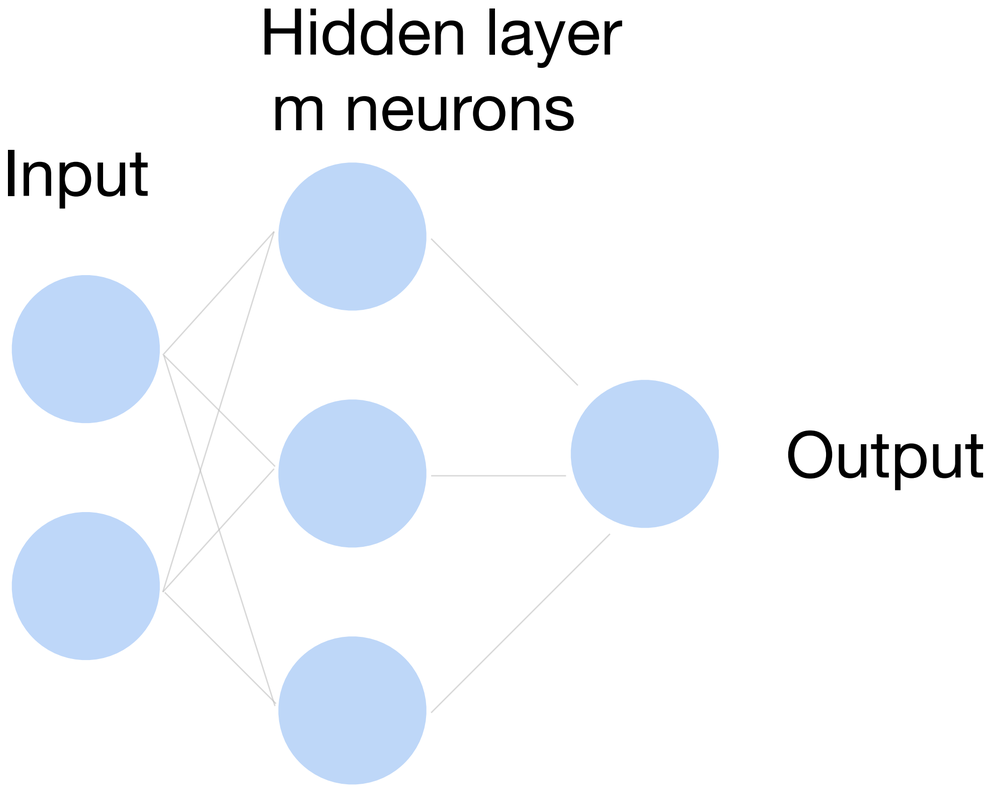
Multilayer Perceptron



Single Hidden Layer

How to classify

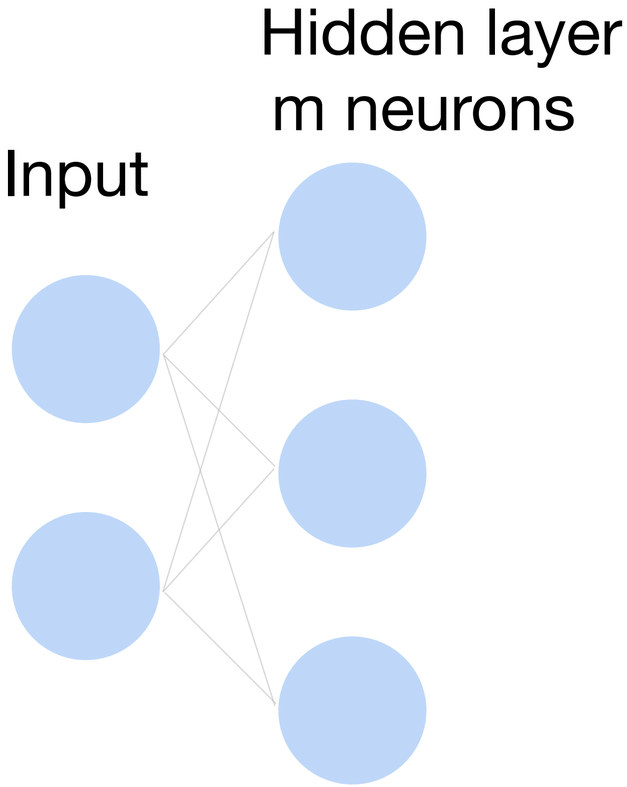
Cats vs. dogs?



Single Hidden Layer

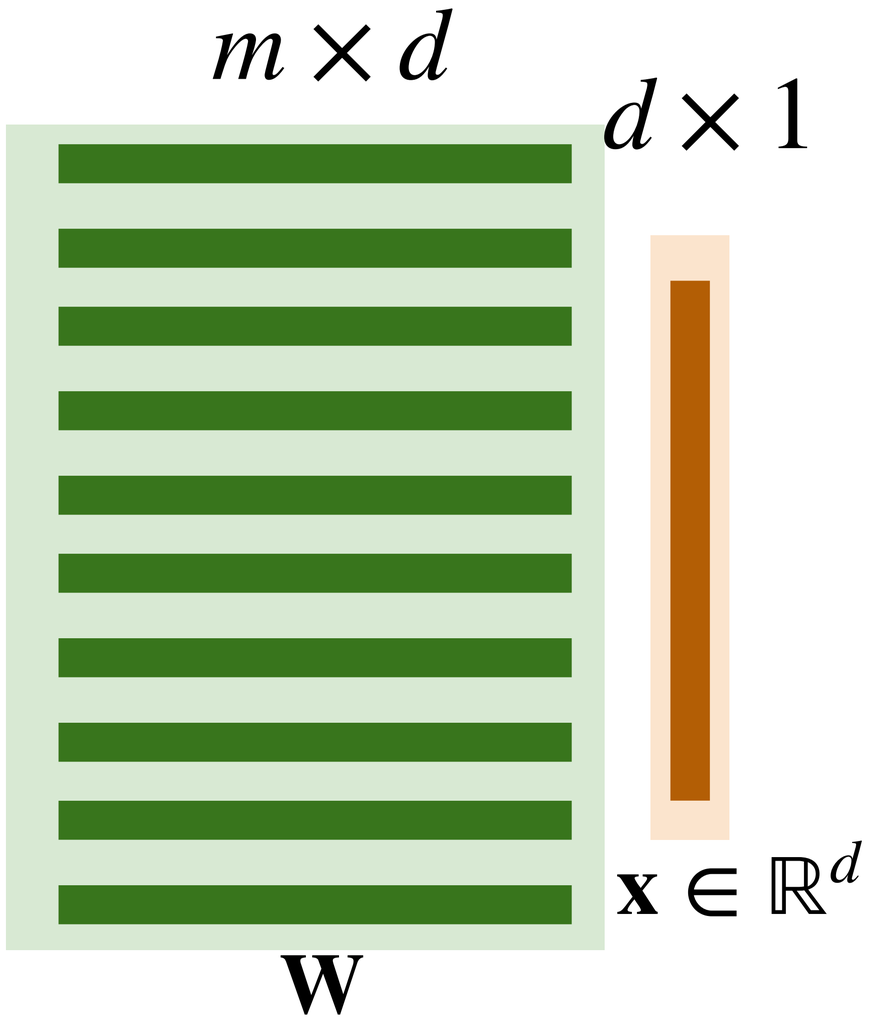
- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$



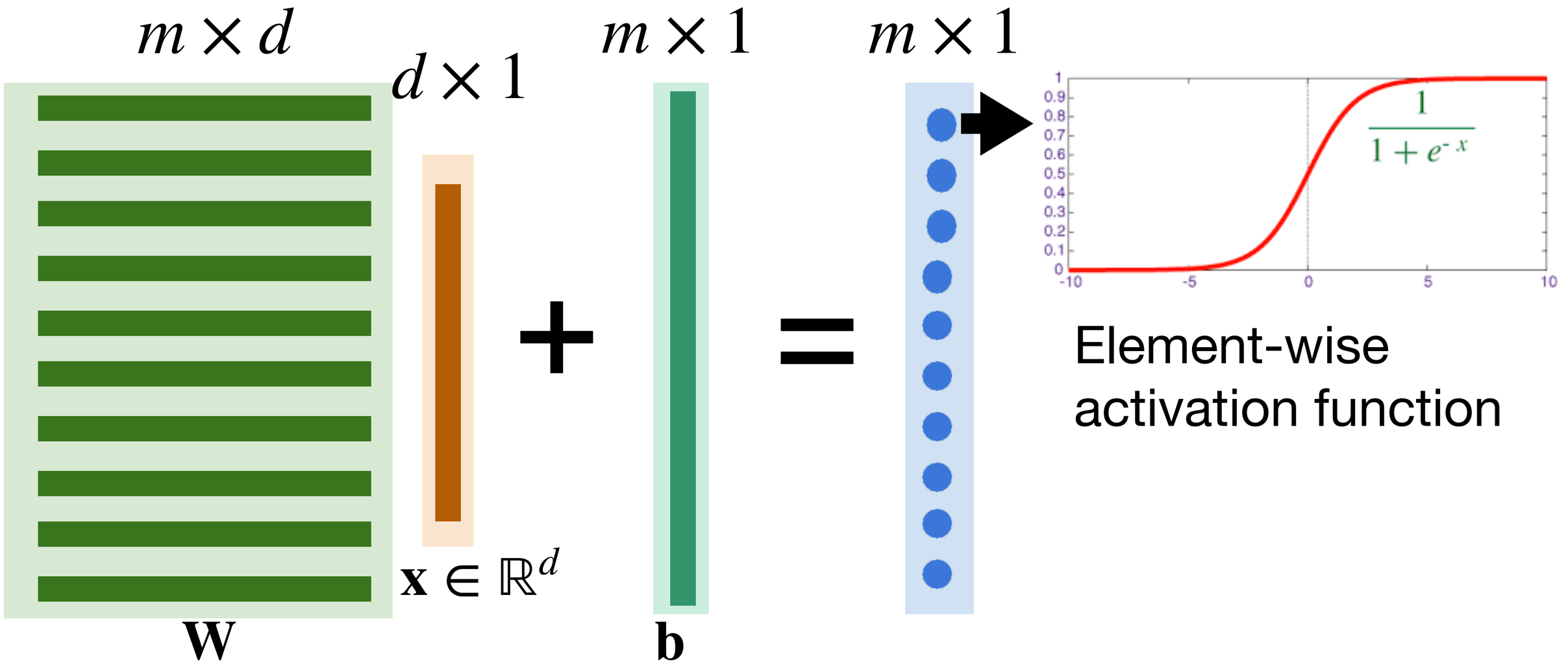
σ is an element-wise activation function

Neural networks with one hidden layer



Neural networks with one hidden layer

Key elements: linear operations + Nonlinear activations

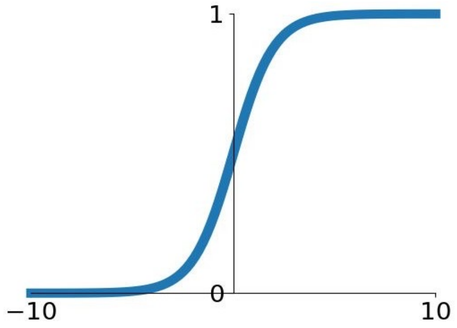


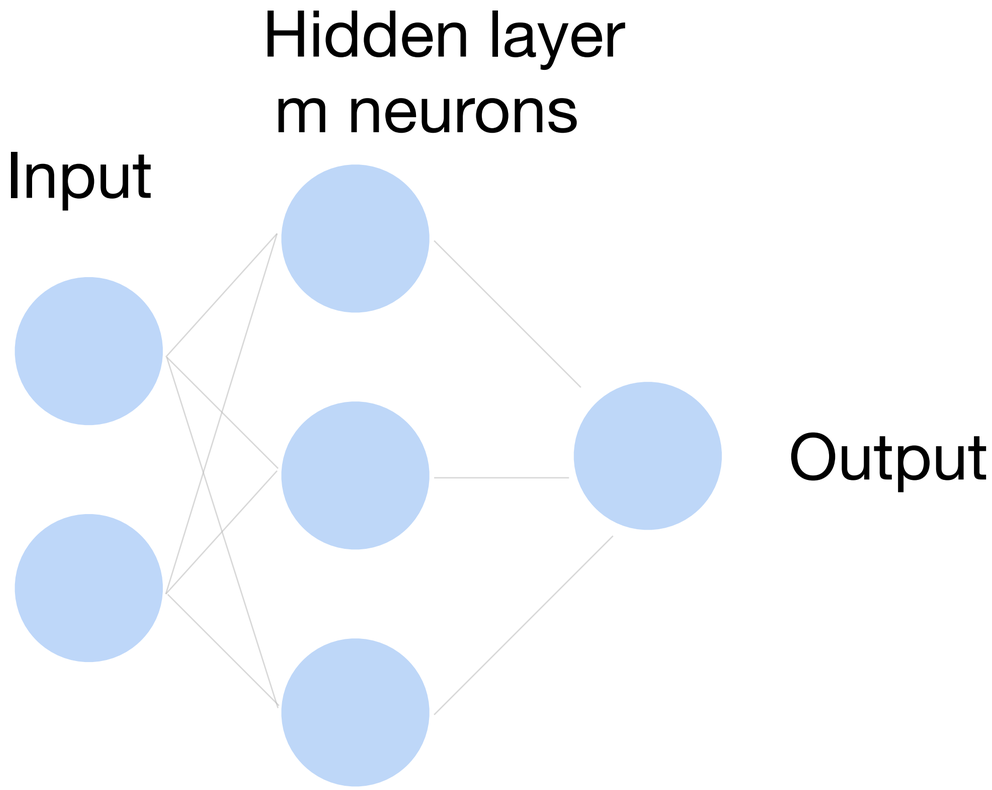
Single Hidden Layer

- Output $f = \mathbf{w}_2^{\top}\mathbf{h} + b_2$
- Normalize the output into probability using sigmoid

$$p(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-f}}$$

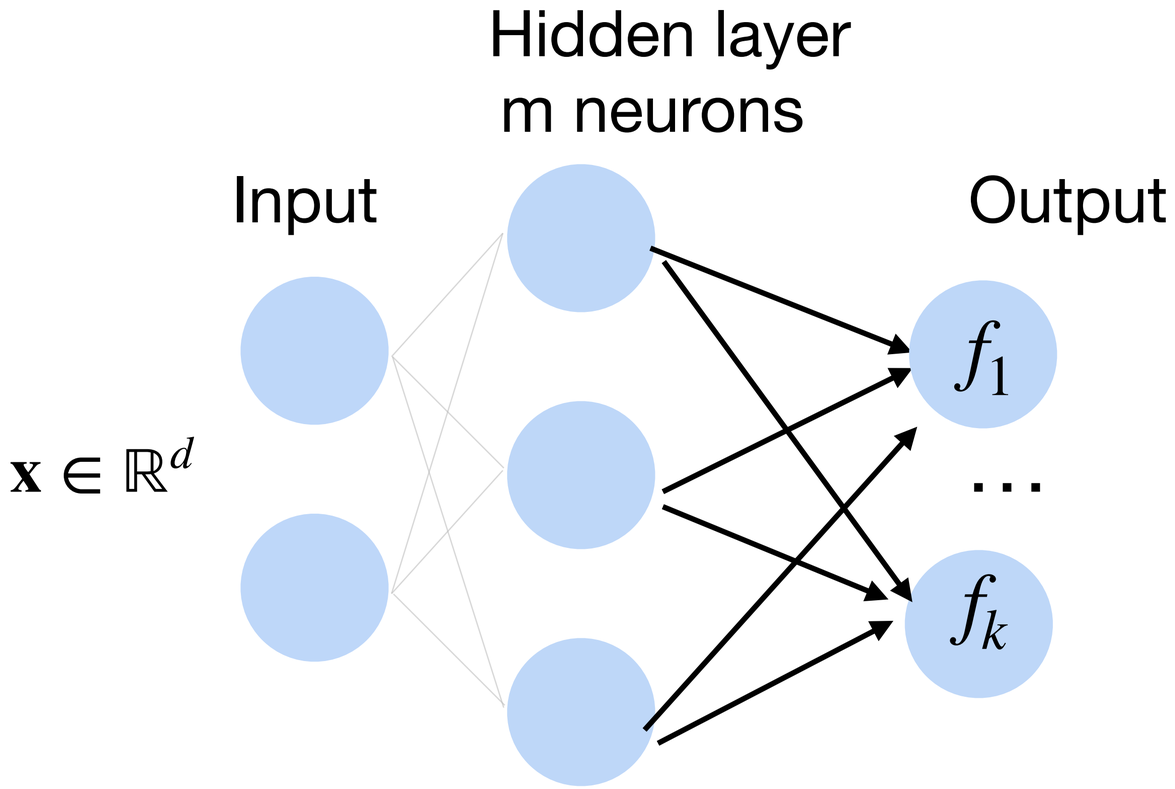
Sigmoid





Multi-class classification

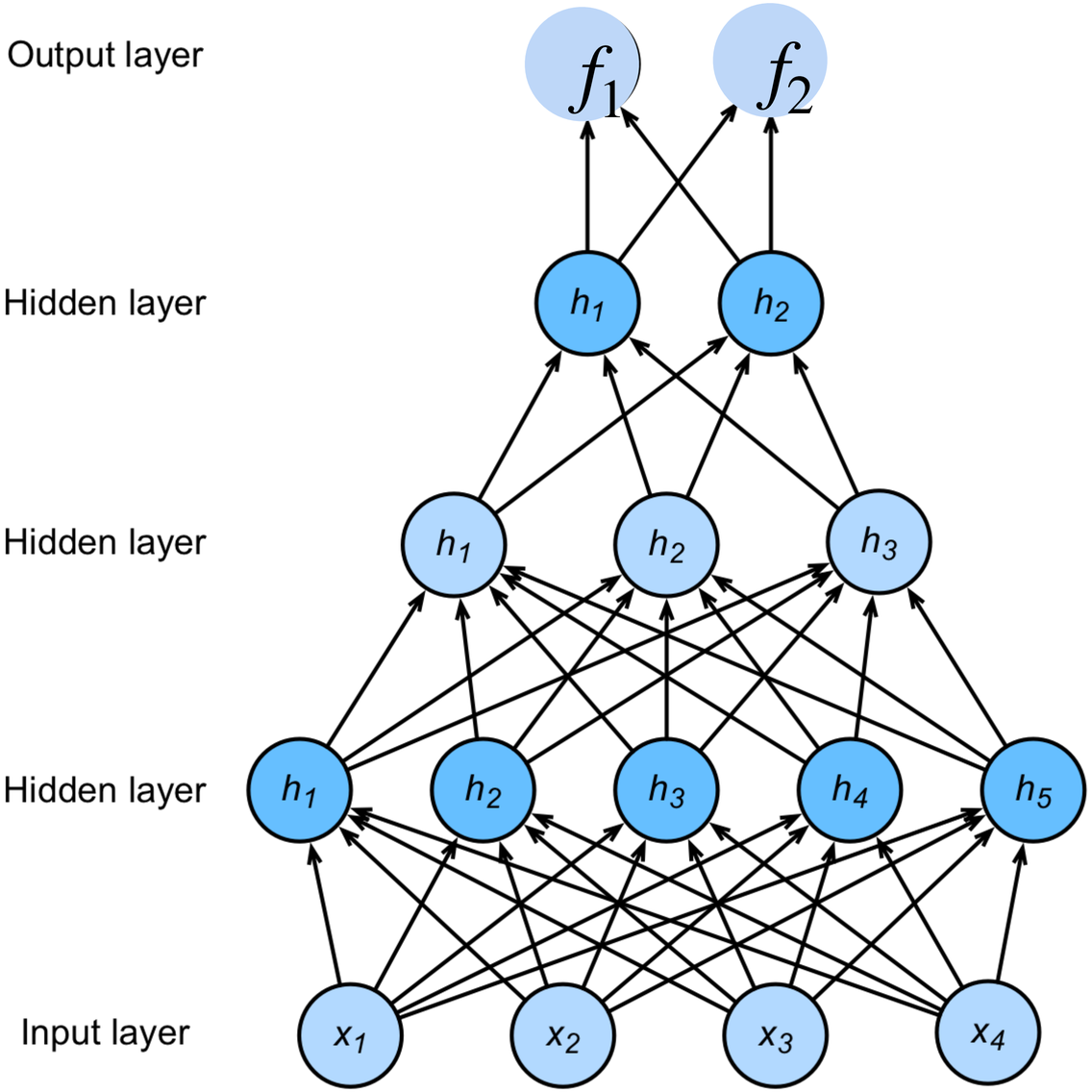
Turns outputs f into k probabilities (sum up to 1 across k classes)



$$p(y \mid \mathbf{x}) = \text{softmax}(\mathbf{f})$$

$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

Deep neural networks (DNNs)



$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$

$$\mathbf{f} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$

$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

NNs are composition of nonlinear functions

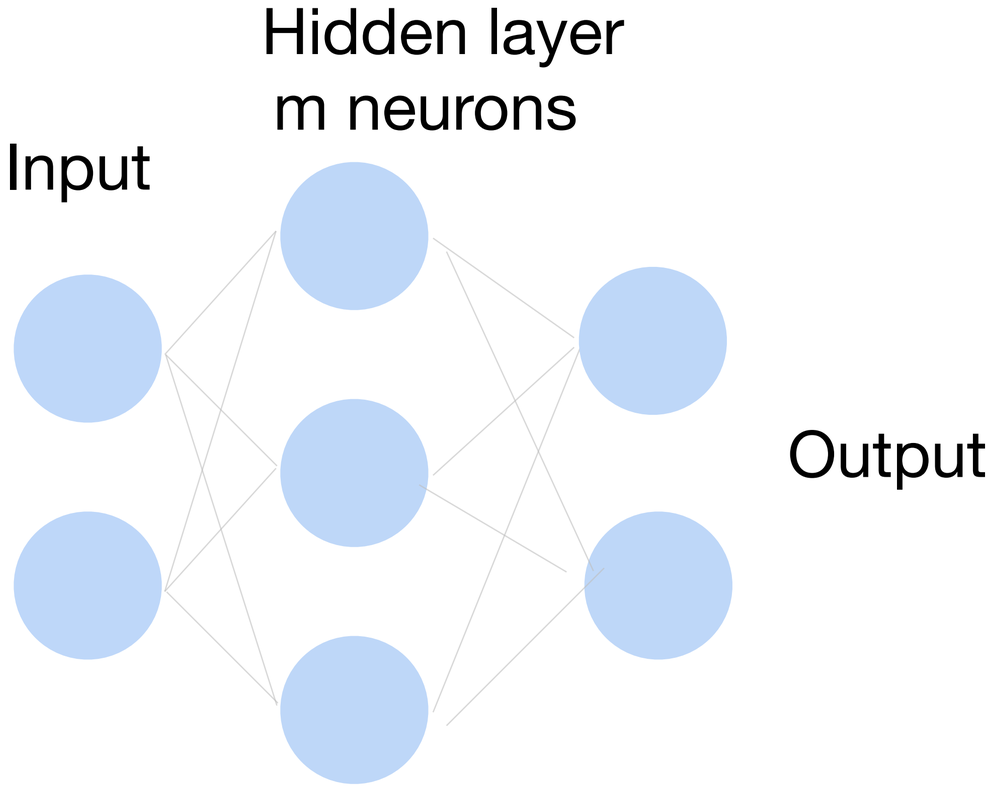
How to train a neural network?

Loss function: $$\frac{1}{|D|}\sum_i \ell(\mathbf{x}_i, y_i)$$

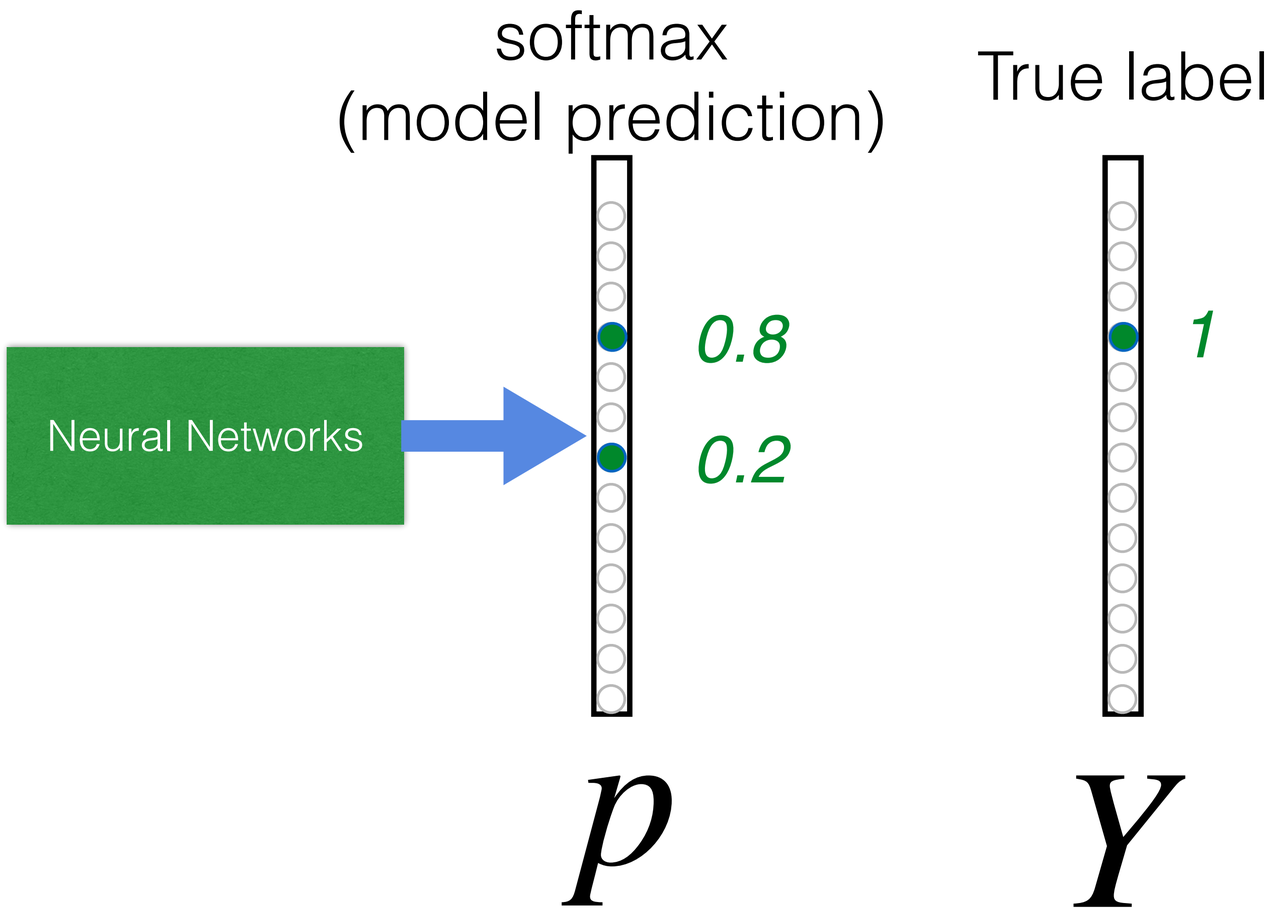
Per-sample loss:

$$\ell(\mathbf{x}, y) = \sum_{j=1}^{K} -y_j \log p_j$$

Also known as cross-entropy loss or softmax loss



Cross-Entropy Loss



$$L_{CE} = \sum_{j} - y_j \log(p_j)$$

$$= - \log(0.8)$$

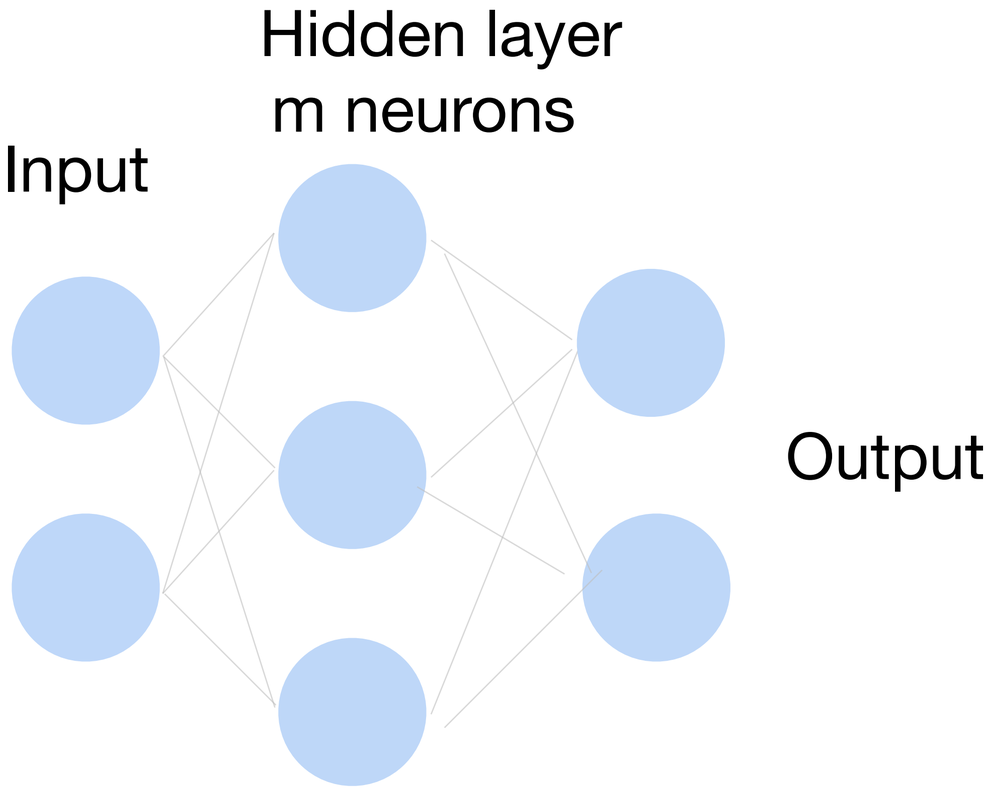
Goal: push **p** and **Y** to be identical

How to train a neural network?

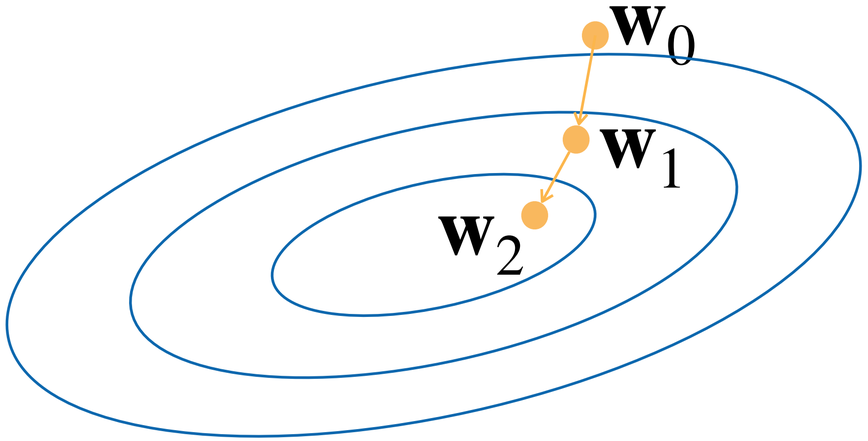
Update the weights W to minimize the loss function

$$L = \frac{1}{|D|} \sum_i \ell(\mathbf{x}_i, y_i)$$

Use gradient descent!



Gradient Descent



- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For t =1,2,…
  - Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{\partial L}{\partial \mathbf{w}_{t-1}}$$

D can be very large. Expensive

$$= \mathbf{w}_{t-1} - \alpha \frac{1}{|D|} \sum_{\mathbf{x} \in D} \frac{\partial \ell(\mathbf{x}_i, y_i)}{\partial \mathbf{w}_{t-1}}$$

- Repeat until converges

Minibatch Stochastic Gradient Descent

- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For t =1,2,…
  - **Randomly sample a subset (mini-batch)** $B \subset D$
    Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{1}{|B|} \sum_{\mathbf{x} \in B} \frac{\partial \ell(\mathbf{x}_i, y_i)}{\partial \mathbf{w}_{t-1}}$$

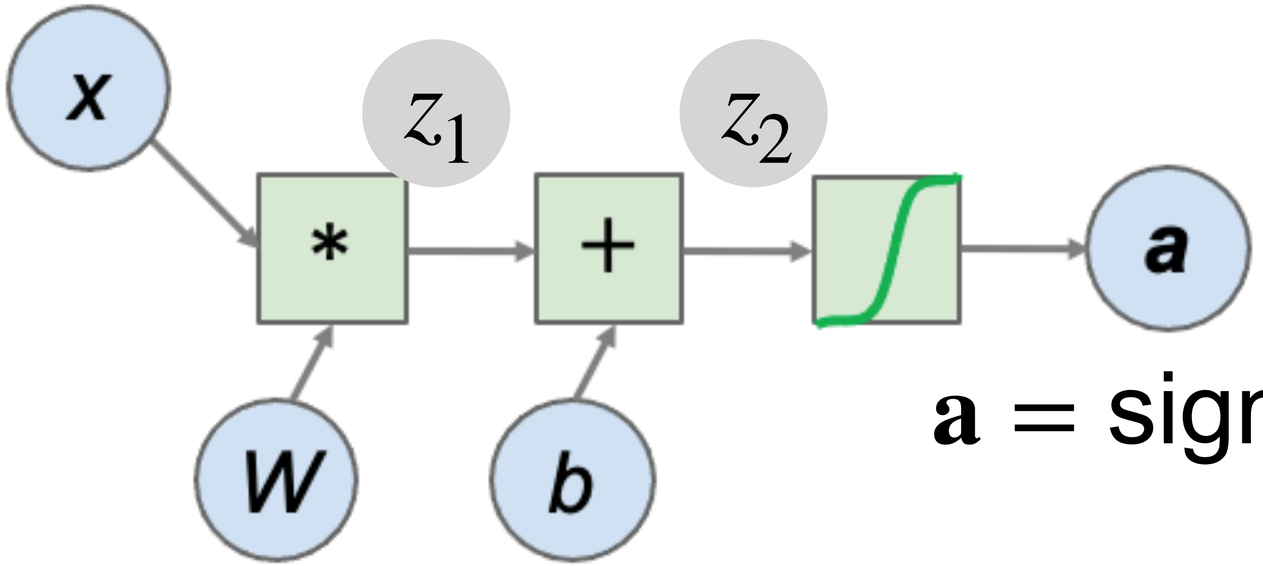
- Repeat

Calculate gradient: backpropagation with chain rule

- Define a loss function L
- Gradient to a variable =

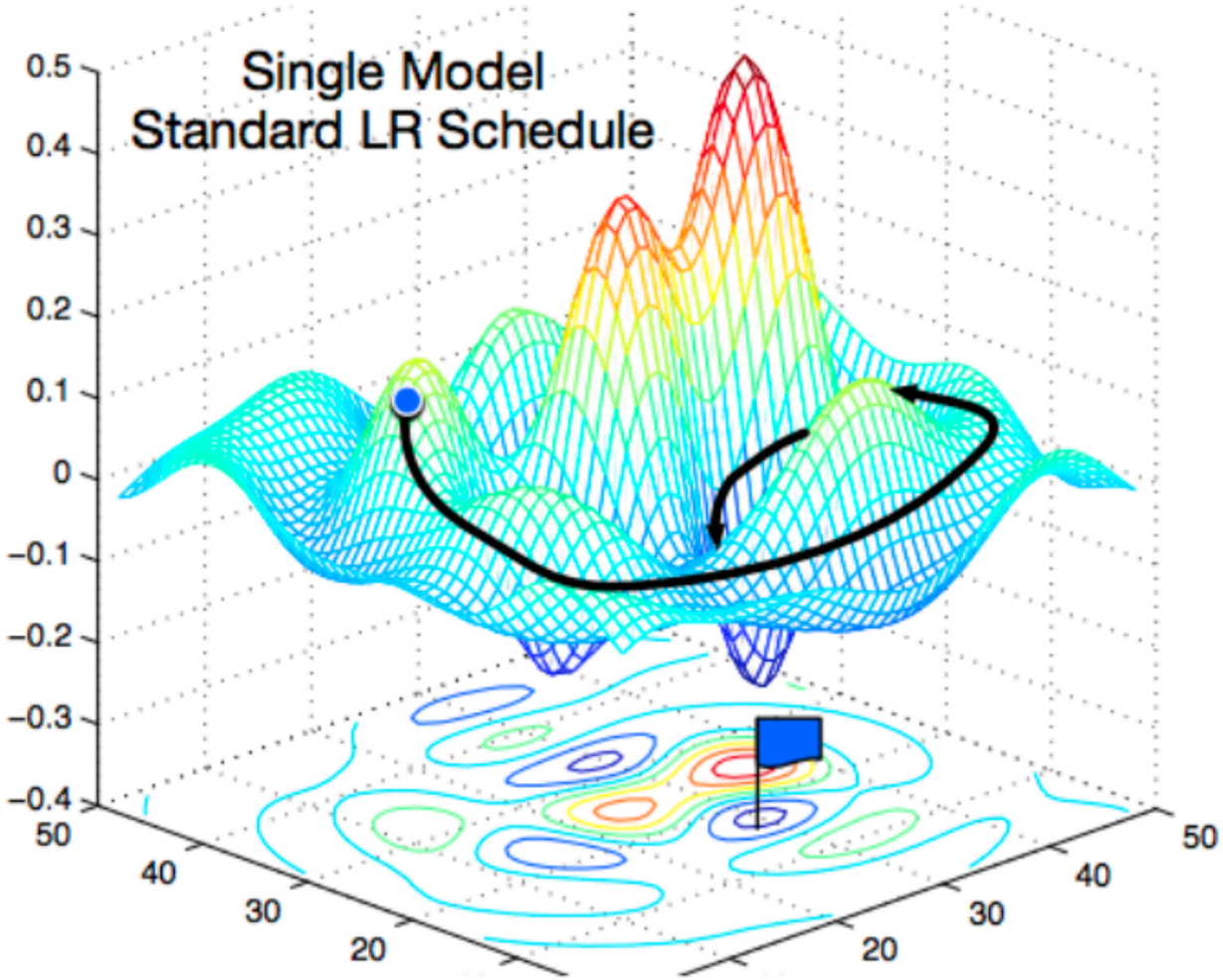
gradient on the top x gradient from the current operation

$$\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \mathbf{z_1}} \frac{\partial \mathbf{z_1}}{\partial \boldsymbol{W}}$$



$$\mathbf{a} = \text{sigmoid}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

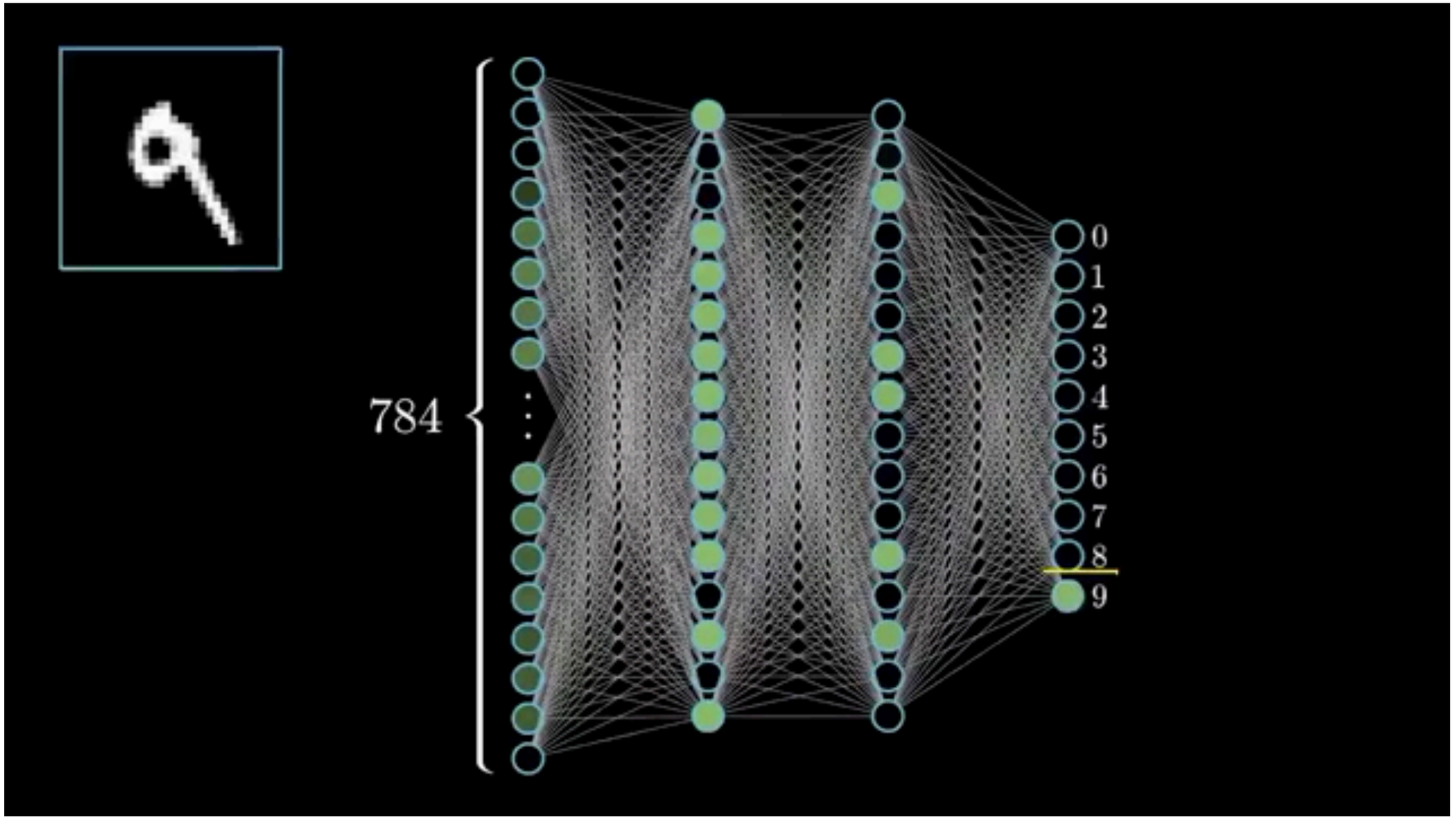
Non-convex Optimization



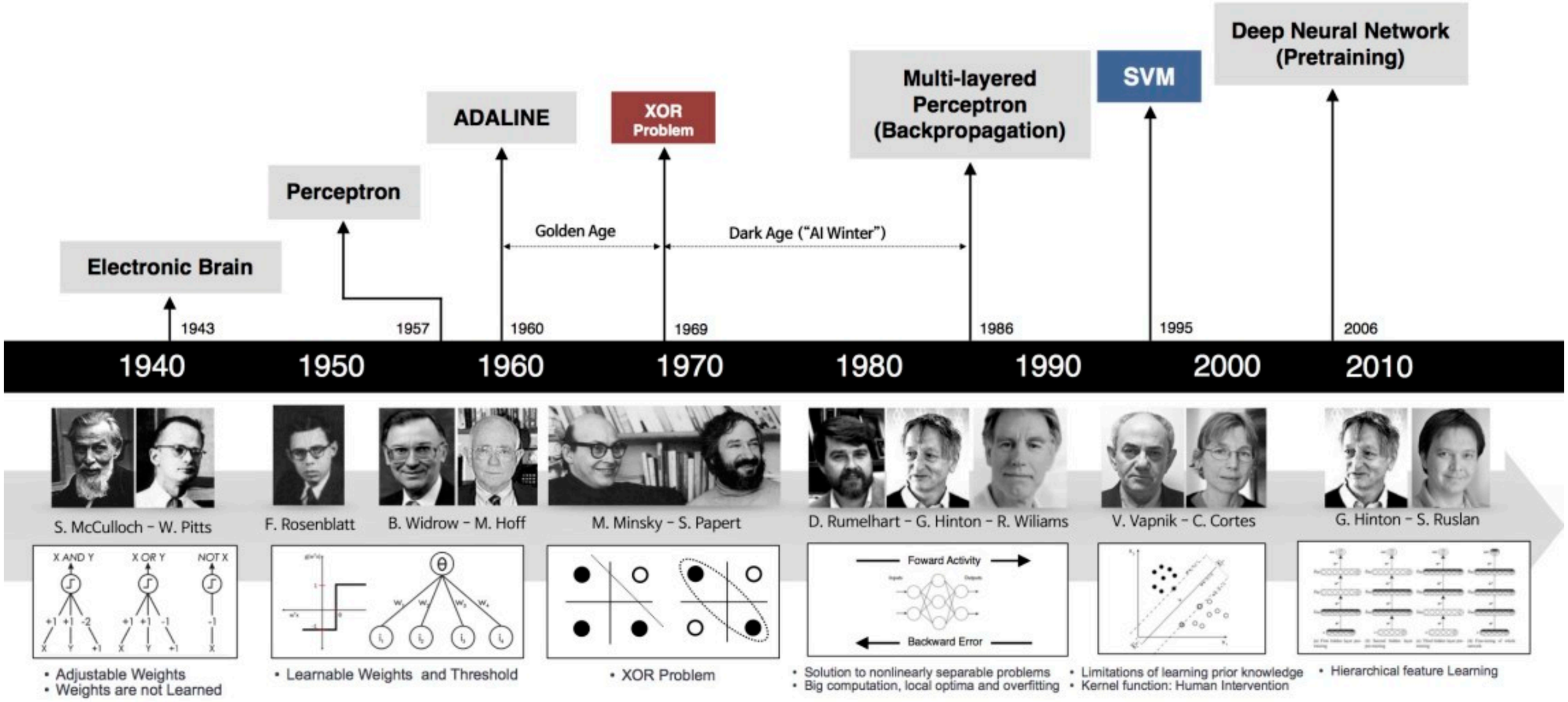
[Gao and Li et al., 2018]

Using SGD in PyTorch (code demo)

Classify MNIST handwritten digits (HW6)



Brief history of neural networks



How to classify

Cats vs. dogs?

Dual

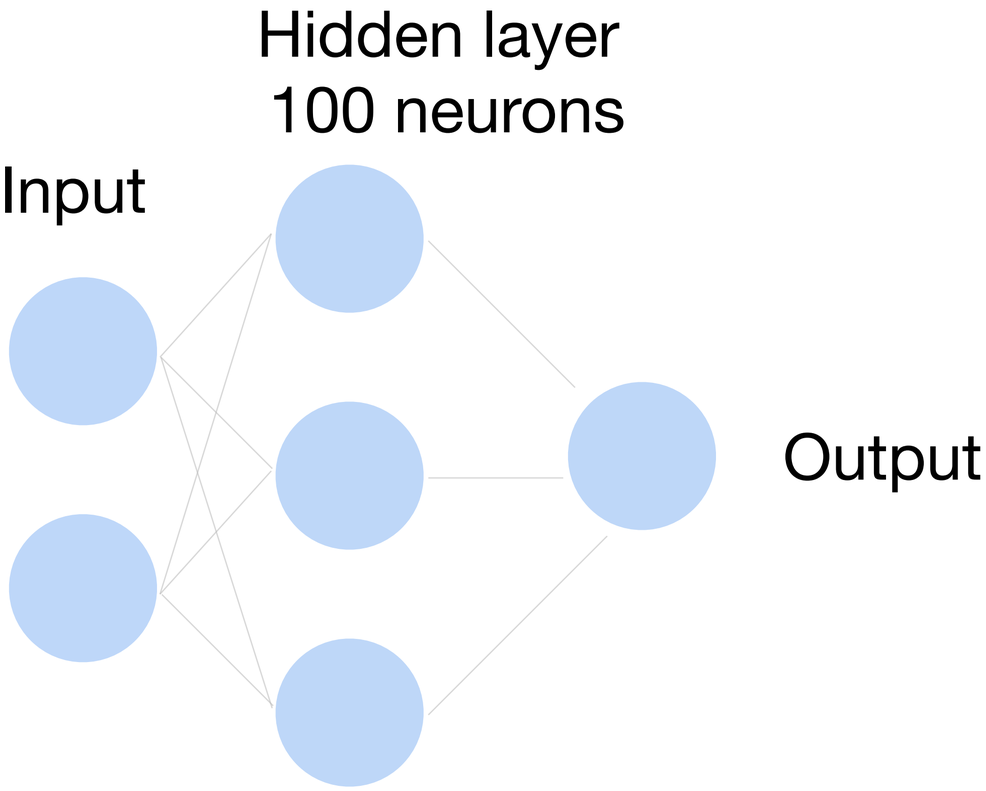
12MP

wide-angle and
telephoto cameras

36M floats in a RGB image!

Fully Connected Networks

Cats vs. dogs?



~ 36M elements x 100 = ~**3.6B** parameters!

Convolutions come to rescue!

Where is Waldo?

Why Convolution?

- Translation Invariance
- Locality

2-D Convolution

Input

0	1	2
3	4	5
6	7	8

*

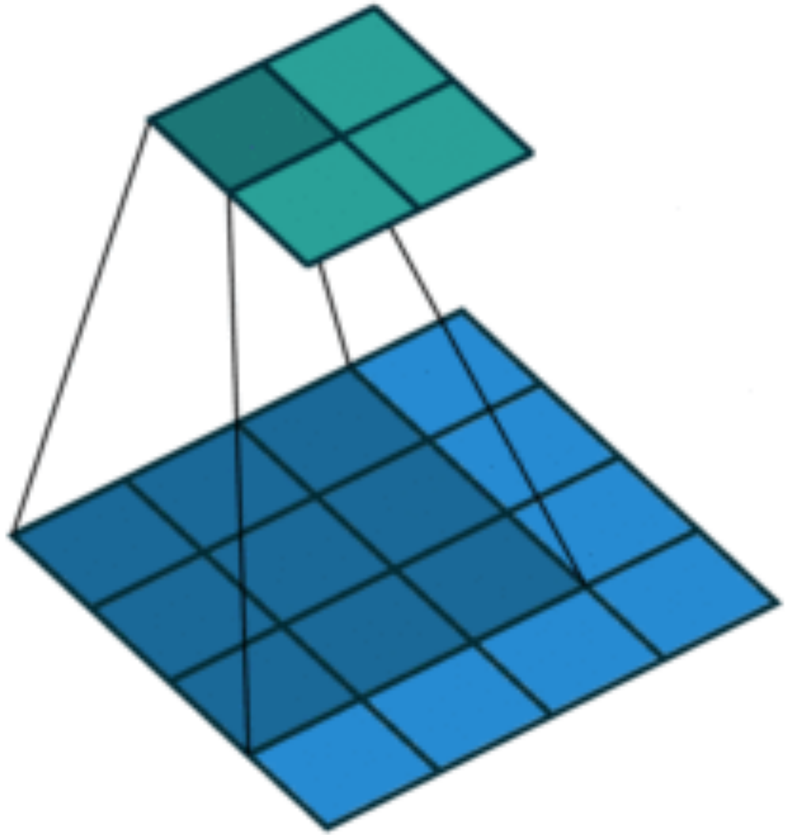
Kernel

0	1
2	3

=

Output

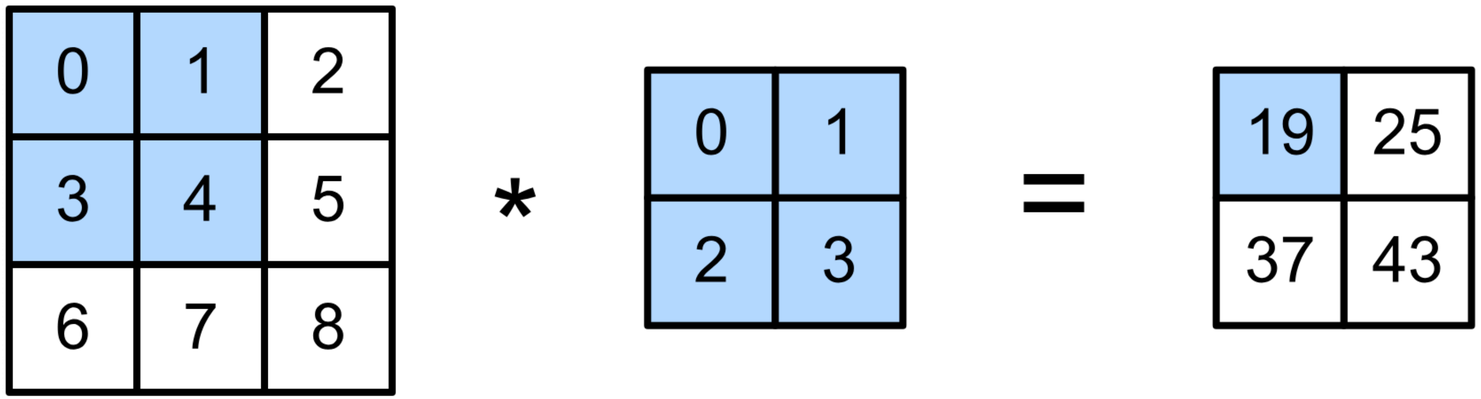
19	25
37	43



(vdumoulin@ Github)

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

2-D Convolution Layer



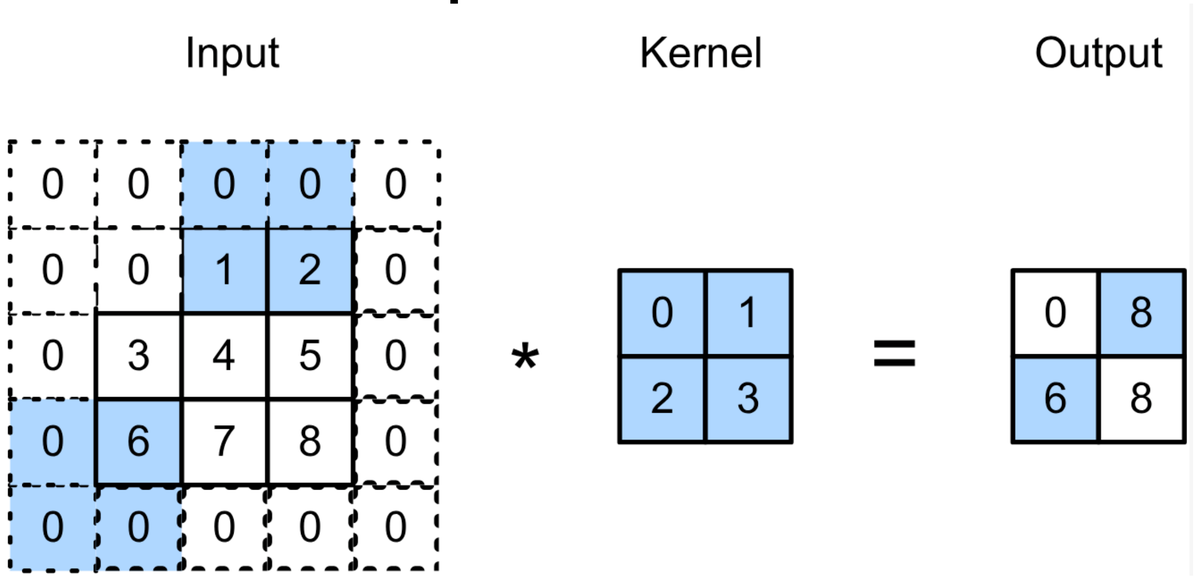
- $\mathbf{X} : n_h \times n_w$ input matrix
- $\mathbf{W} : k_h \times k_w$ kernel matrix
- b: scalar bias
- $\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

- **W** and *b* are learnable parameters

2-D Convolution Layer with Stride and Padding

- Stride is the #rows/#columns per slide
- Padding adds rows/columns around input
- Output shape



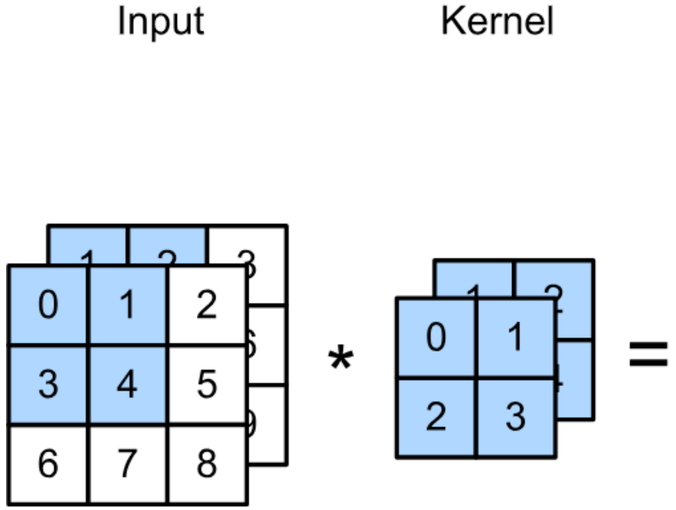
Kernel/filter size

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

Input size **Pad** **Stride**

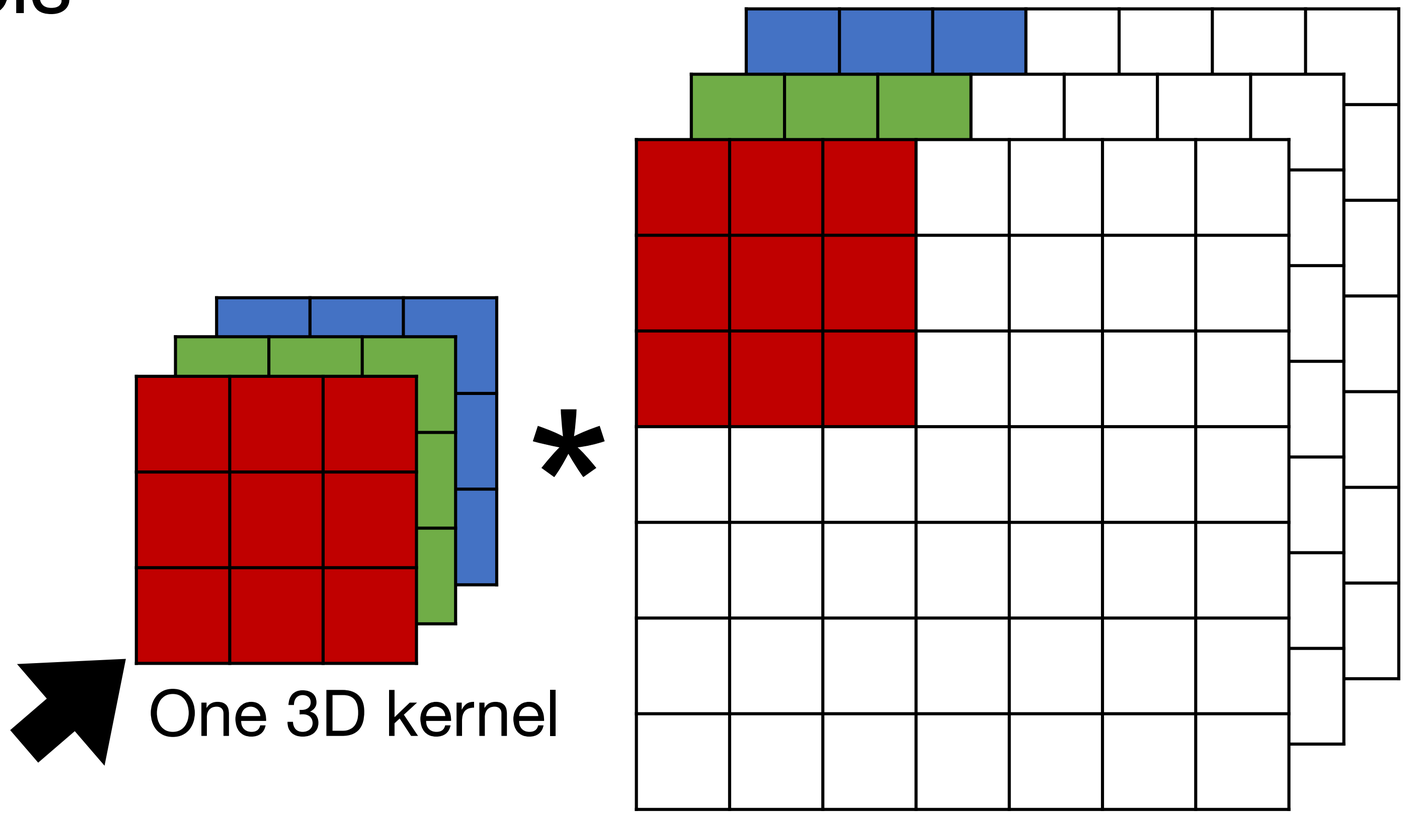
Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a kernel for each channel, and then sum results over channels



Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Have a 2D kernel for each channel, and then sum results over channels



Multiple Input Channels

- Input and kernel can be 3D, e.g., an RGB image have 3 channels
- Also call each 3D kernel a "**filter**", which produce only **one** output channel (due to summation over channels)

One filter
(3 channels)

*

RGB (3 input channels)

Multiple filters (in one layer)

- Apply multiple filters on the input
- Each filter may learn different features about the input
- Each filter (3D kernel) produces one output channel

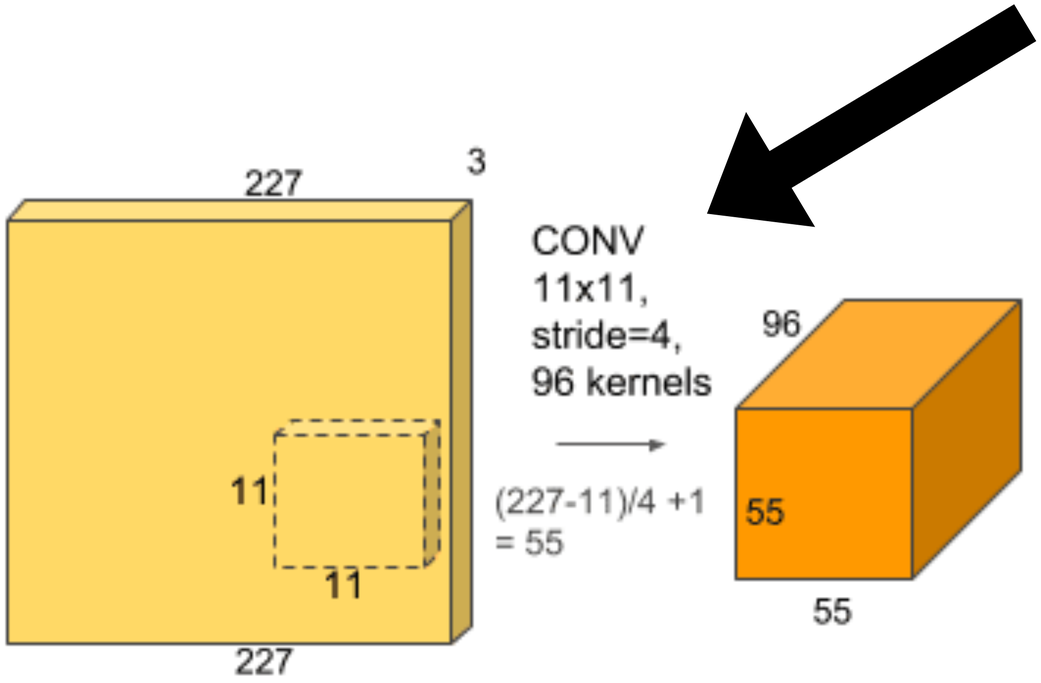
A different filter

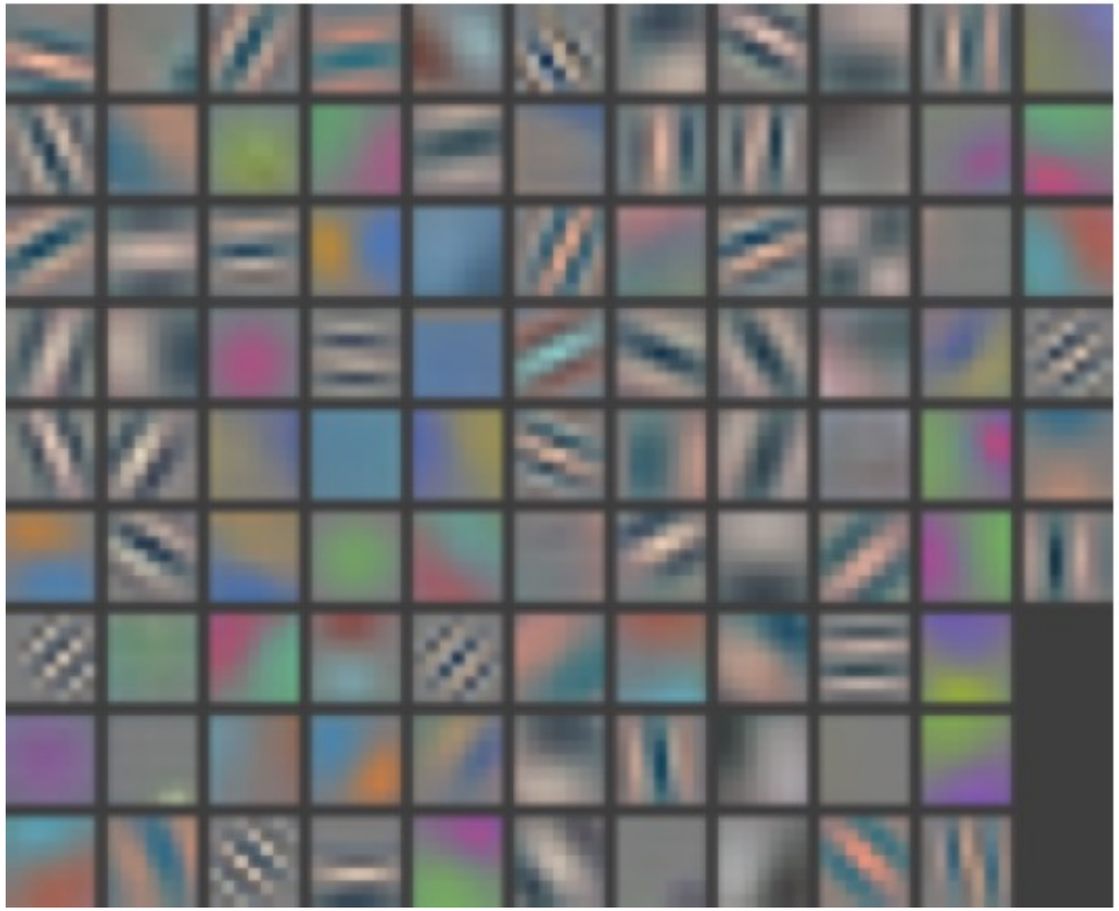
*

RGB (3 input channels)

Conv1 Filters in AlexNet

- 96 filters (each of size 11x11x3)
- Gabor filters





Figures from Visualizing and Understanding Convolutional Networks by *M. Zeiler and R. Fergus*

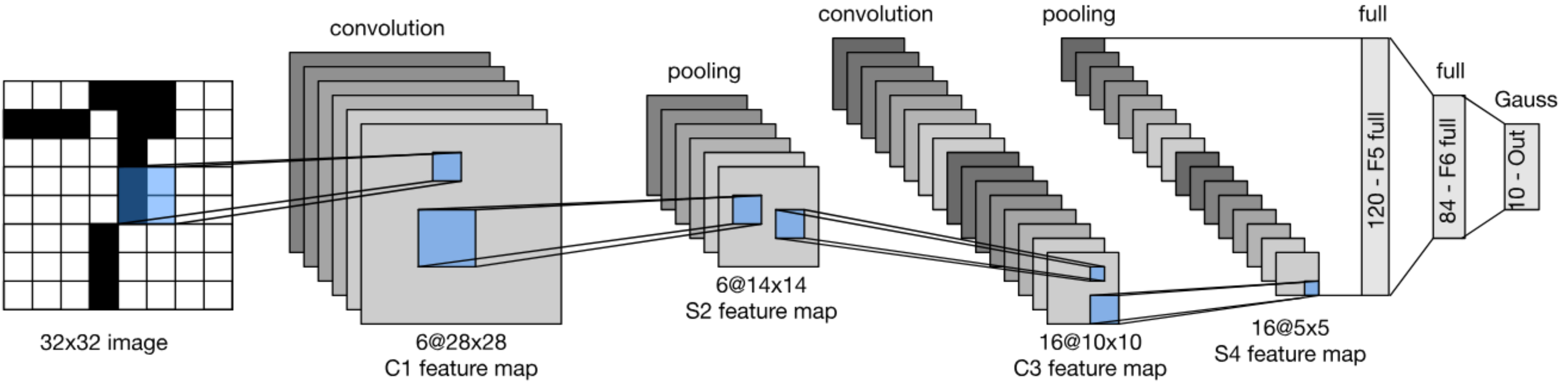
Multiple Output Channels

- The # of output channels = # of filters
- Input $\mathbf{X}: c_i \times n_h \times n_w$
- Kernel $\mathbf{W}: c_o \times c_i \times k_h \times k_w$
- Output $\mathbf{Y}: c_o \times m_h \times m_w$

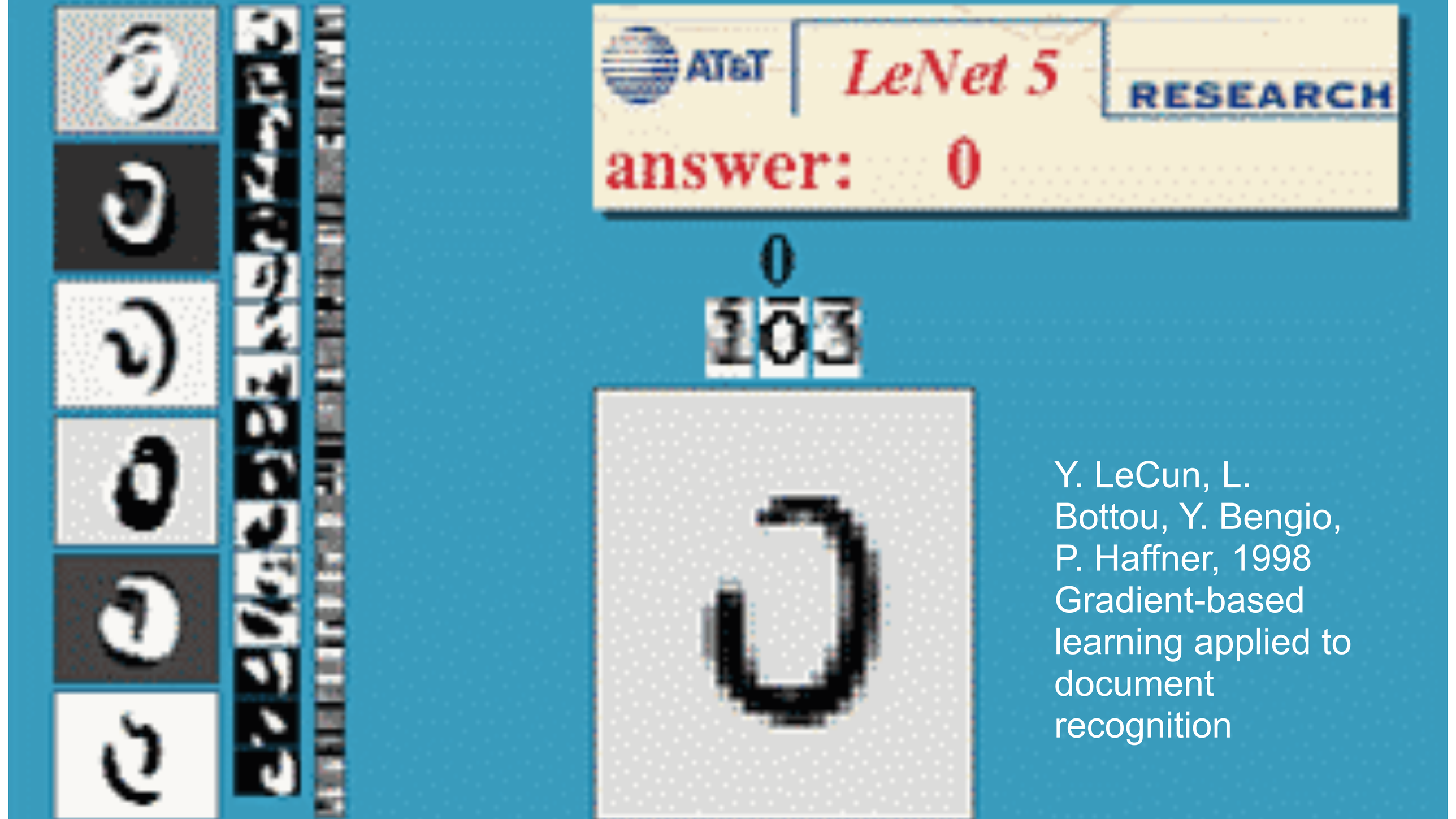
$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

for $i = 1, \ldots, c_o$

LeNet Architecture



gluon-cv.mxnet.io



Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998
Gradient-based learning applied to document recognition

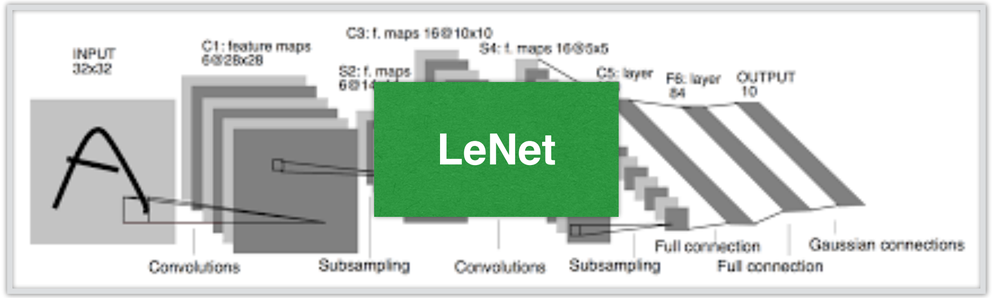
LeNet in Pytorch (HW7)

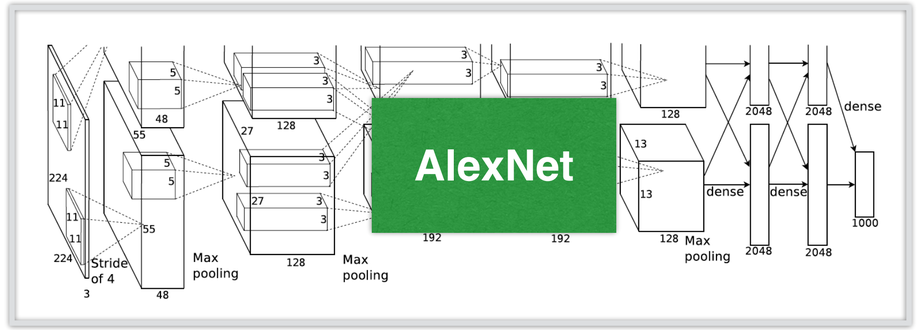
Connect theory and practice

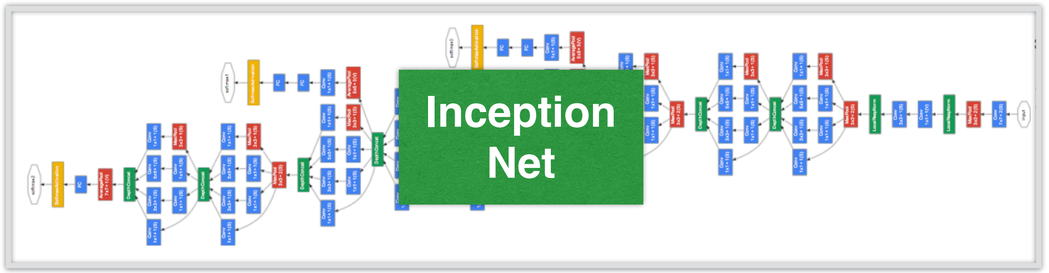
```
def __init__(self):
    super(LeNet5, self).__init__()
    # Convolution (In LeNet-5, 32x32 images are given as input. Hence padding of 2 is done below)
    self.conv1 = torch.nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1, padding=2, bias=True)
    # Max-pooling
    self.max_pool_1 = torch.nn.MaxPool2d(kernel_size=2)
    # Convolution
    self.conv2 = torch.nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1, padding=0, bias=True)
    # Max-pooling
    self.max_pool_2 = torch.nn.MaxPool2d(kernel_size=2)
    # Fully connected layer
    self.fc1 = torch.nn.Linear(16*5*5, 120)   # convert matrix with 16*5*5 (= 400) features to a matrix of 120 features (col
    self.fc2 = torch.nn.Linear(120, 84)       # convert matrix with 120 features to a matrix of 84 features (columns)
    self.fc3 = torch.nn.Linear(84, 10)        # convert matrix with 84 features to a matrix of 10 features (columns)
```

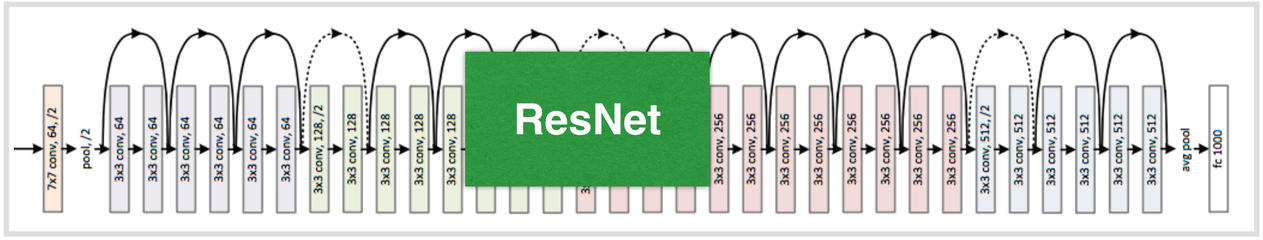
Convolutional Neural Networks

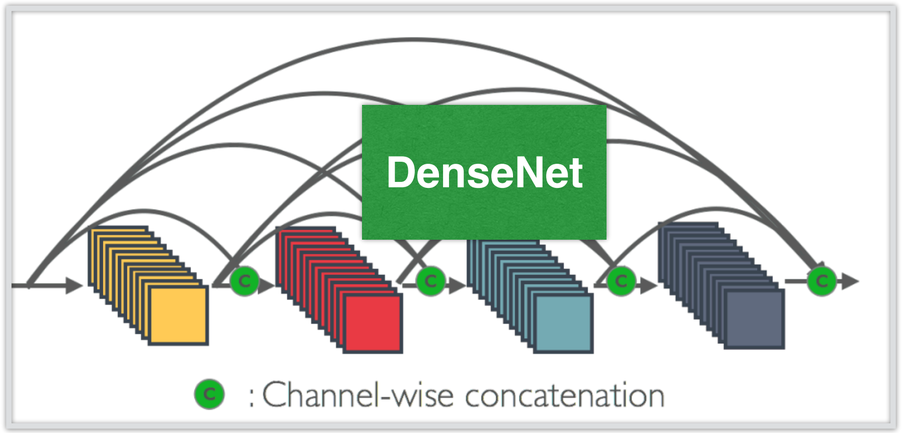
Evolution of neural net architectures

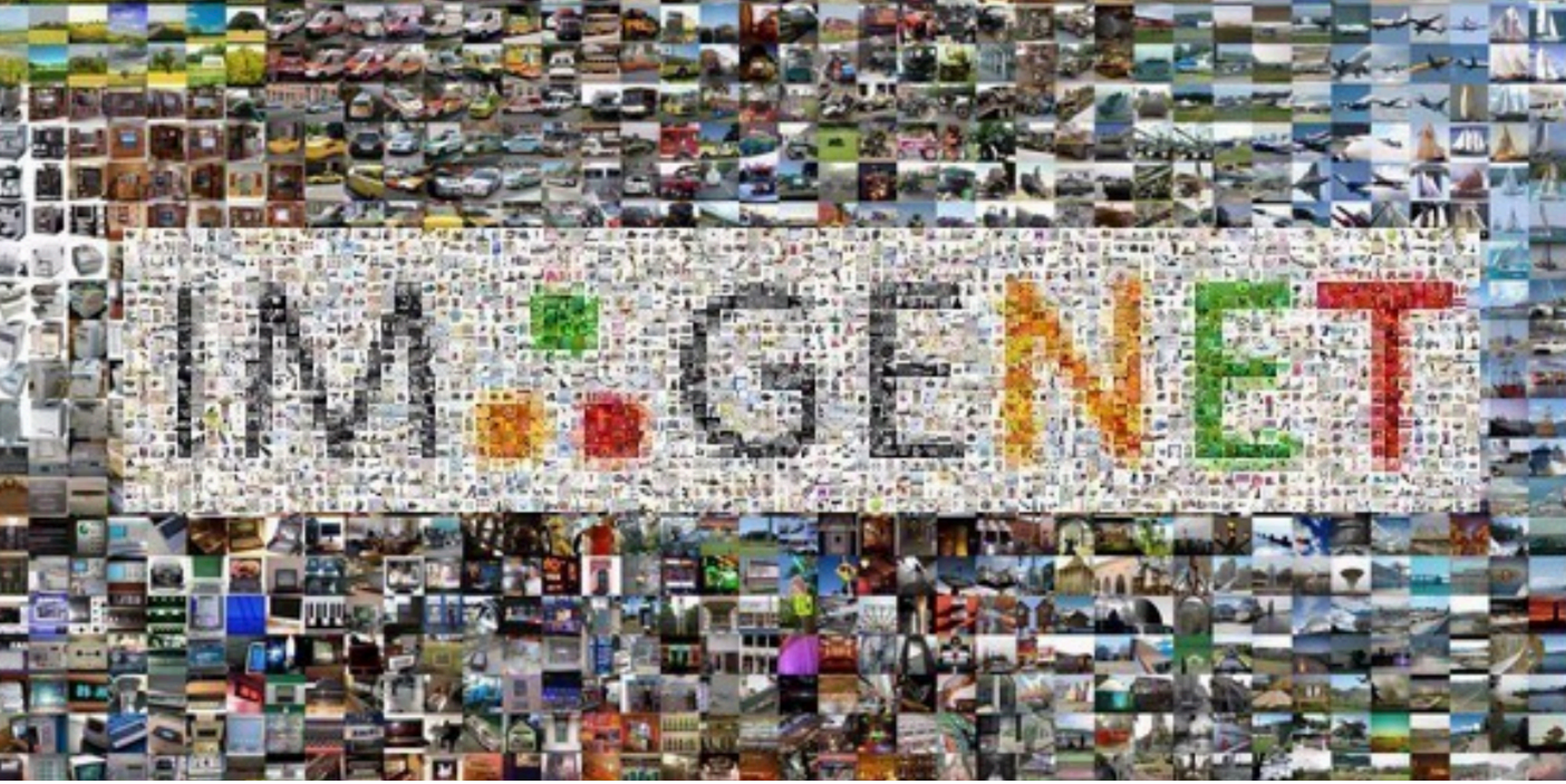






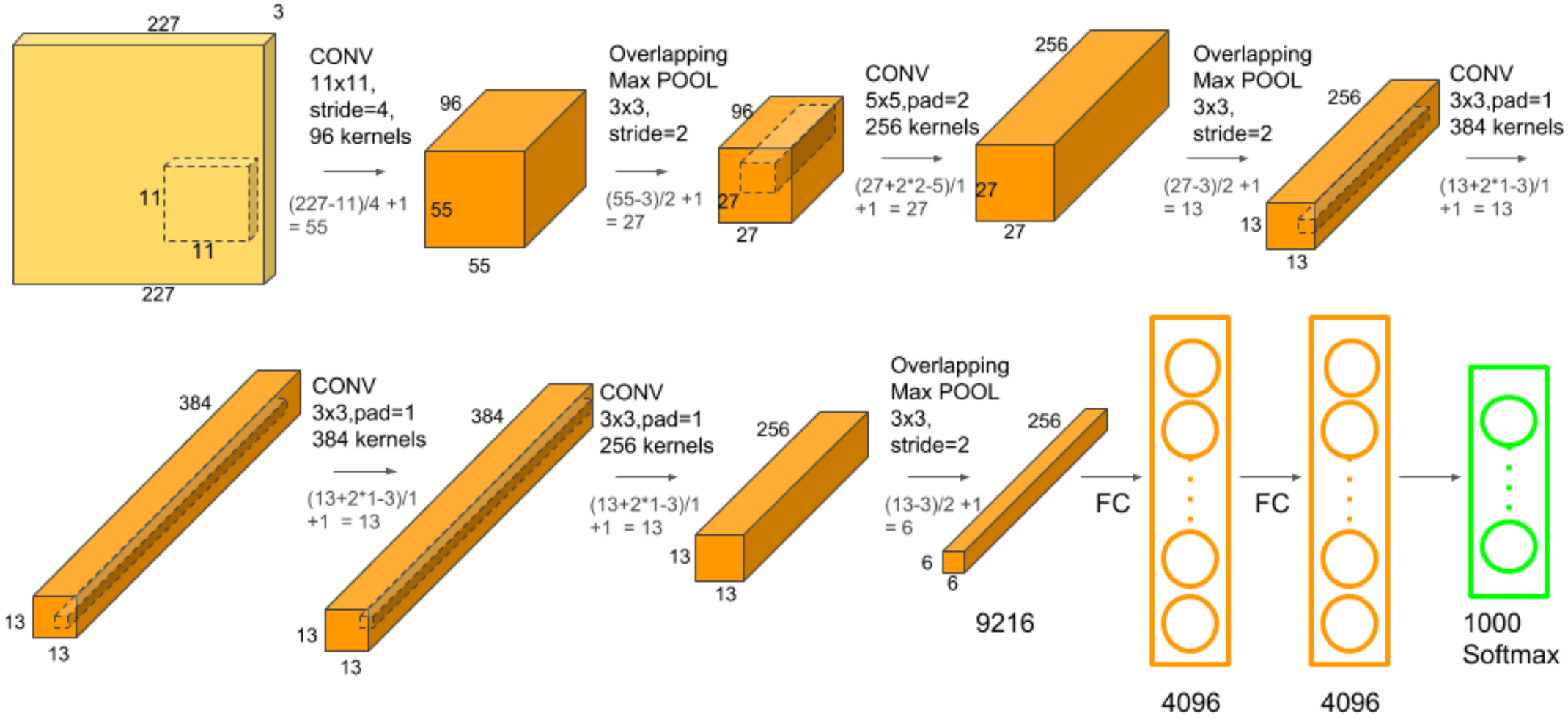






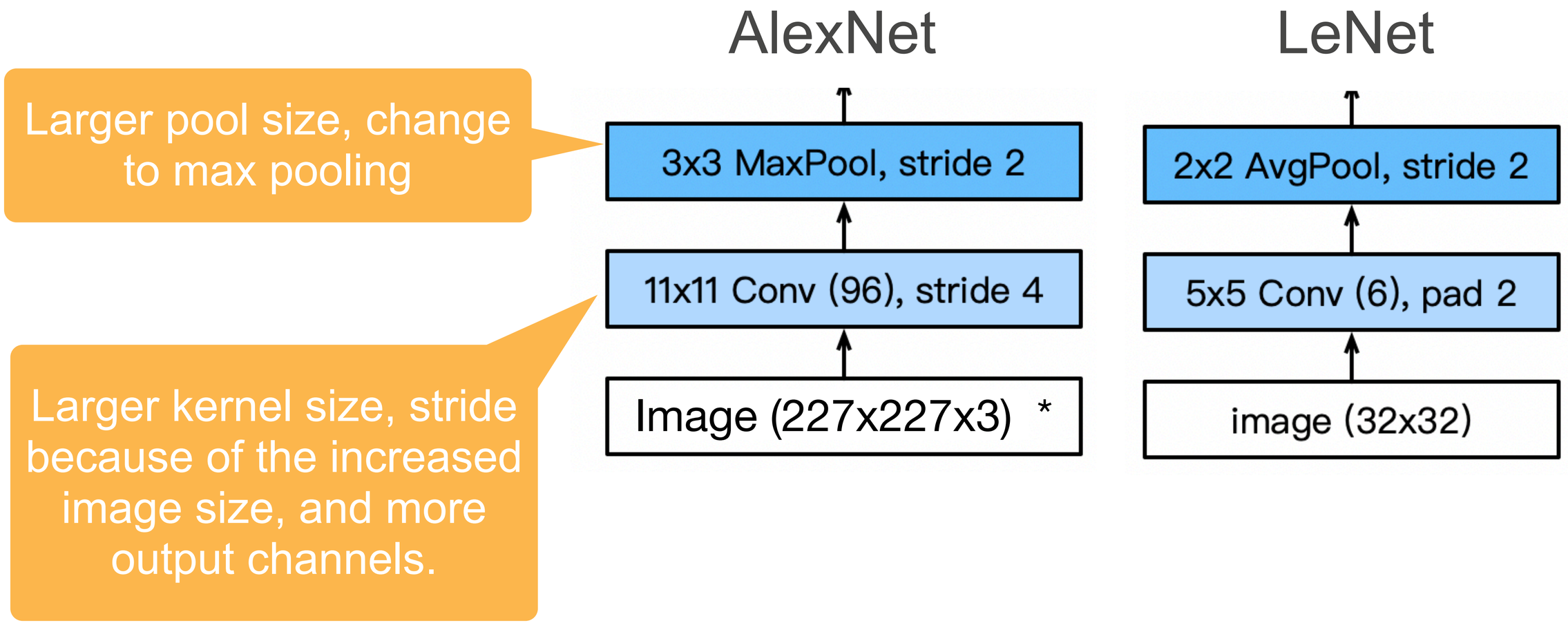
Deng et al. 2009

AlexNet



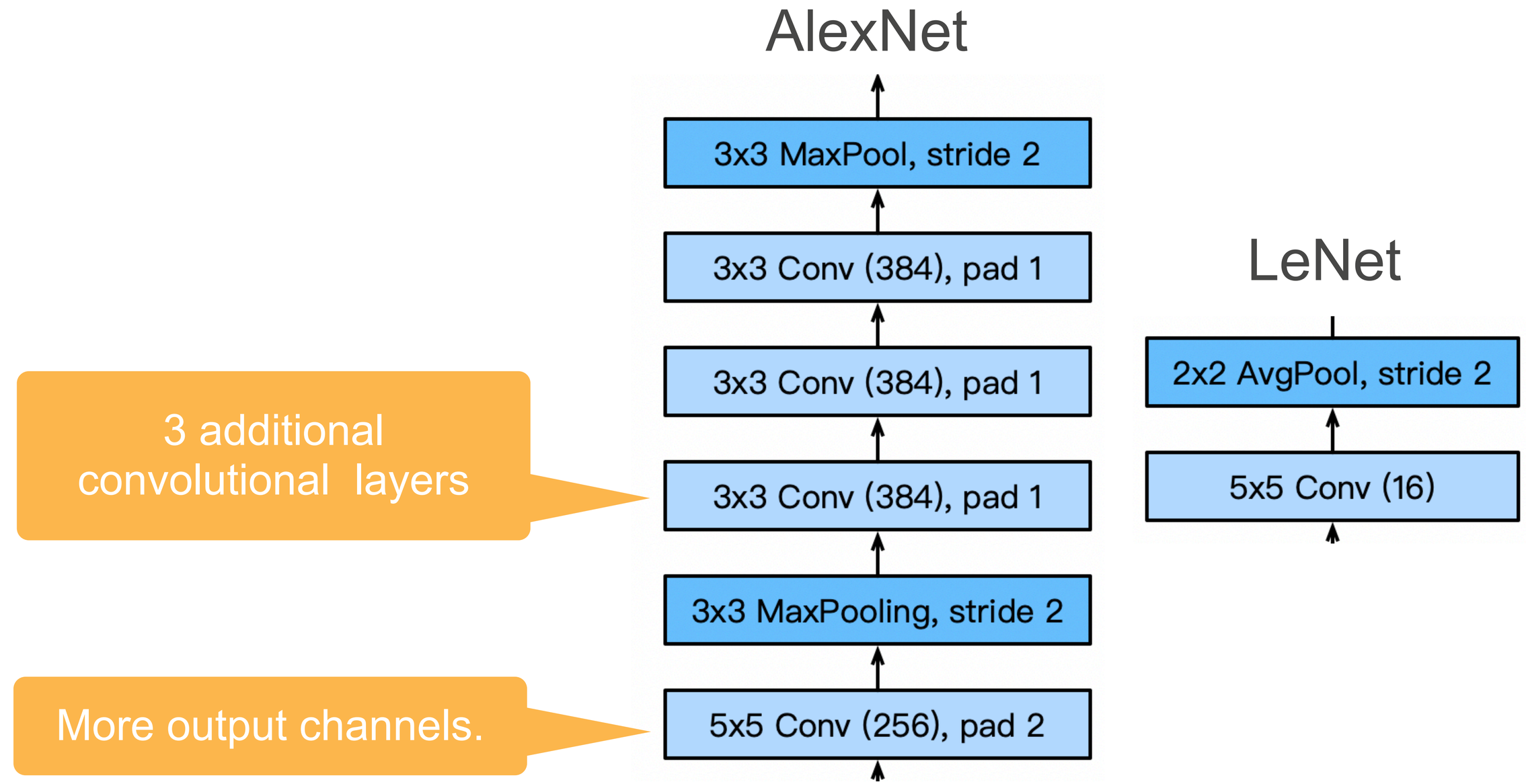
[Krizhevsky et al. 2012]

AlexNet vs LeNet Architecture

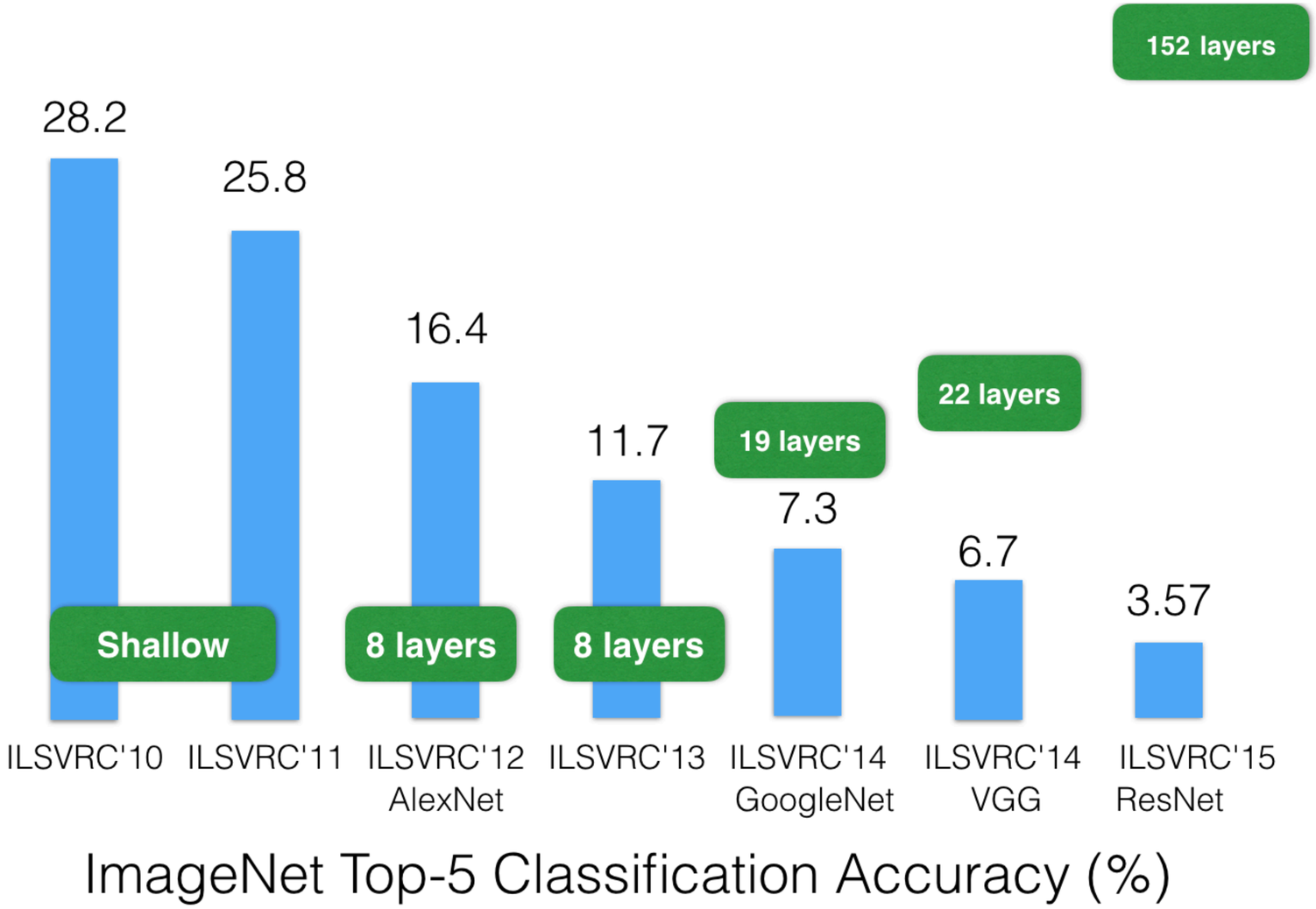


*Note that the original paper used 224x224x3, which was incorrect

AlexNet Architecture



ResNet: Going deeper in depth



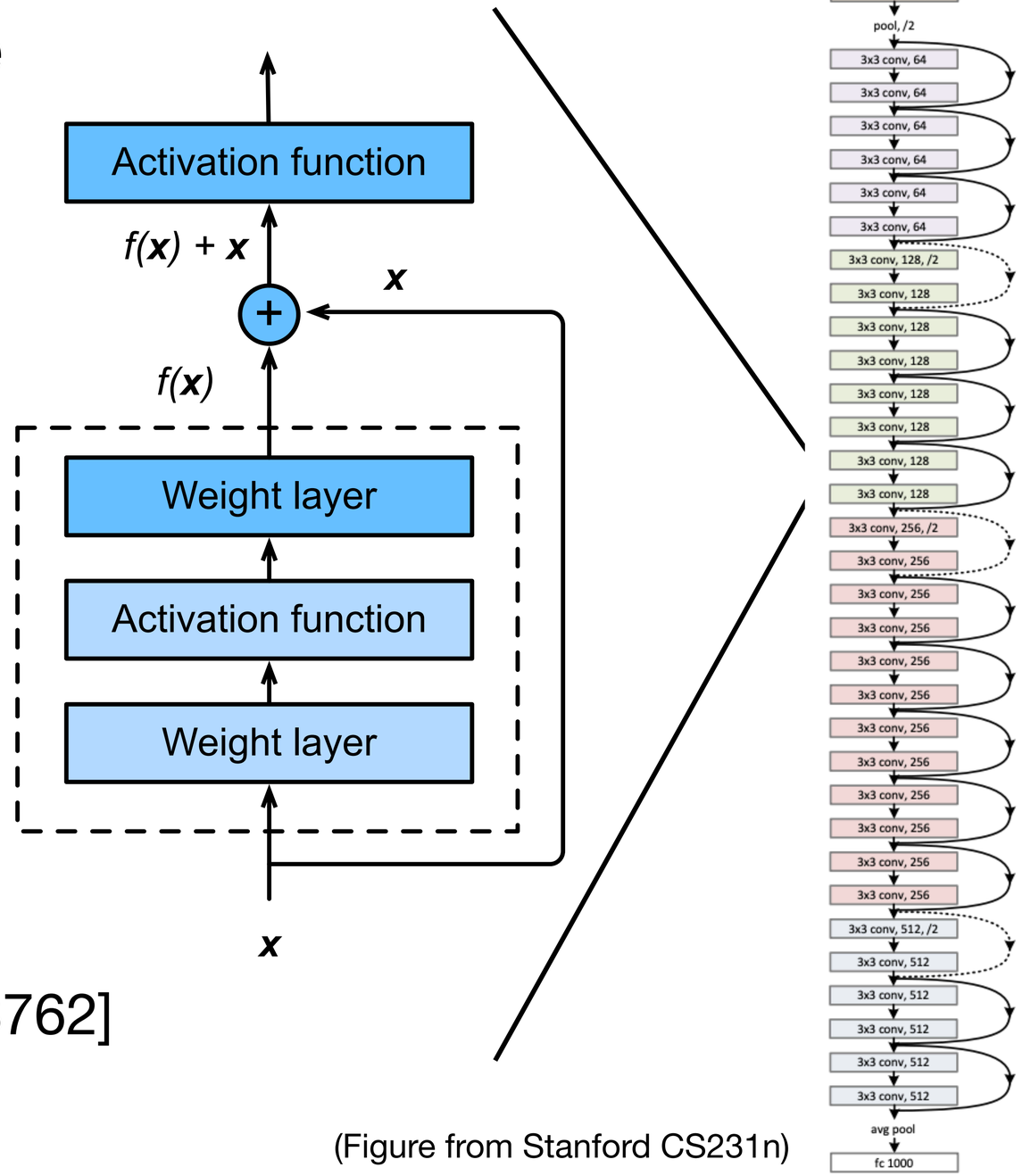
[He et al. 2015]

Full ResNet Architecture

[He et al. 2015]

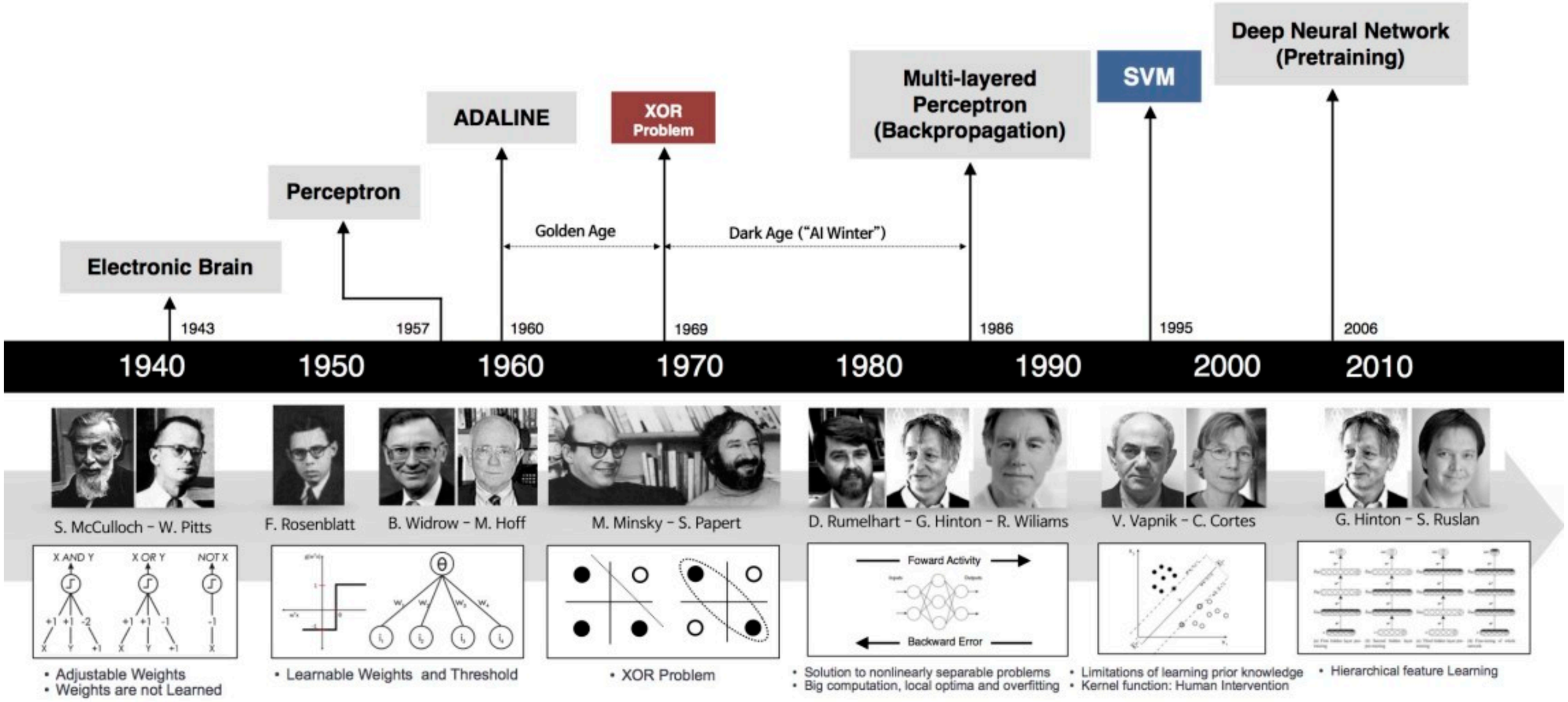
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride of 2 (/2 in each dimension)

[More advanced topics covered in CS762]



(Figure from Stanford CS231n)

Brief history of neural networks



What we've learned today…

- Modeling a single neuron
  - Linear perceptron
  - Limited power of a single neuron
- Multi-layer perceptron
- Training of neural networks
  - Loss function (cross entropy)
  - Backpropagation and SGD
- Convolutional neural networks
  - Convolution, pooling, stride, padding
  - Basic architectures (LeNet etc.)
  - More advanced architectures (AlexNet, ResNet etc)

Thank you!

Some of the slides in these lectures have been adapted from materials developed by Alex Smola and Mu Li:
https://courses.d2l.ai/berkeley-stat-157/index.html