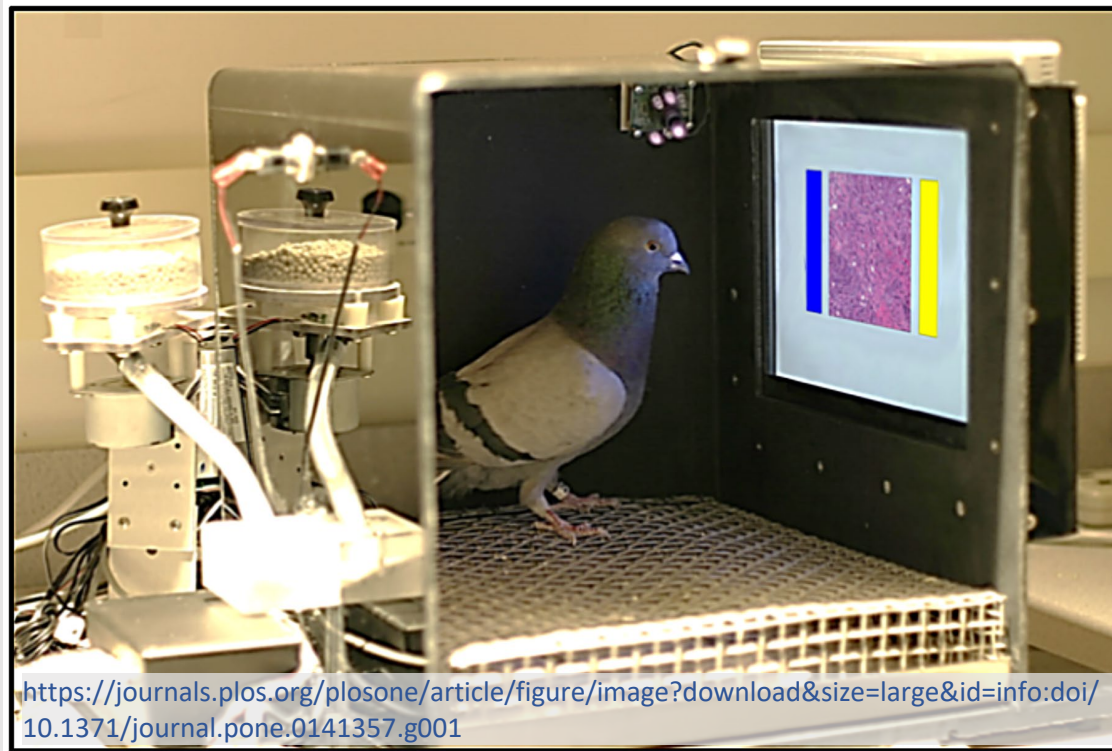


Lecture 7: Neural Networks



[Haiping Lu](#)

YouTube Playlist: <https://www.youtube.com/c/HaipingLu/>
[COM4059/6059: MLAI20@The University of Sheffield](#)

Week 7 Contents / Objectives

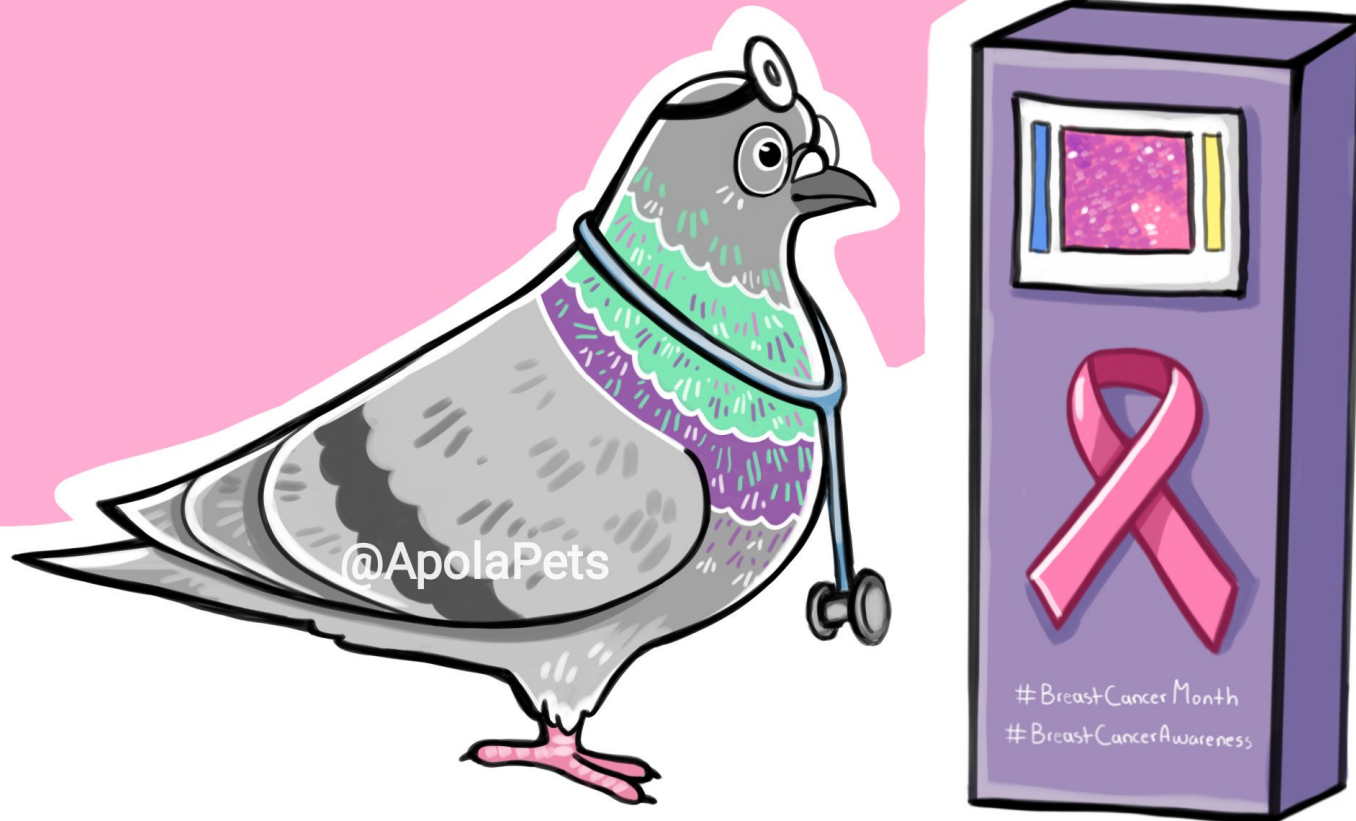
- Learning with Neurons
- Neural Networks (NNs)
- NN Decision Boundary & Features
- Convolutional NN Basics
- Convolutional NN Unboxing

Week 7 Contents / Objectives

- **Learning with Neurons**
- Neural Networks (NNs)
- NN Decision Boundary & Features
- Convolutional NN Basics
- Convolutional NN Unboxing

Did you know ?

Pigeons identify Breast Cancer



Just as well as radiologists

<https://pbs.twimg.com/media/DqczsWhX4AErP0h.jpg:large>

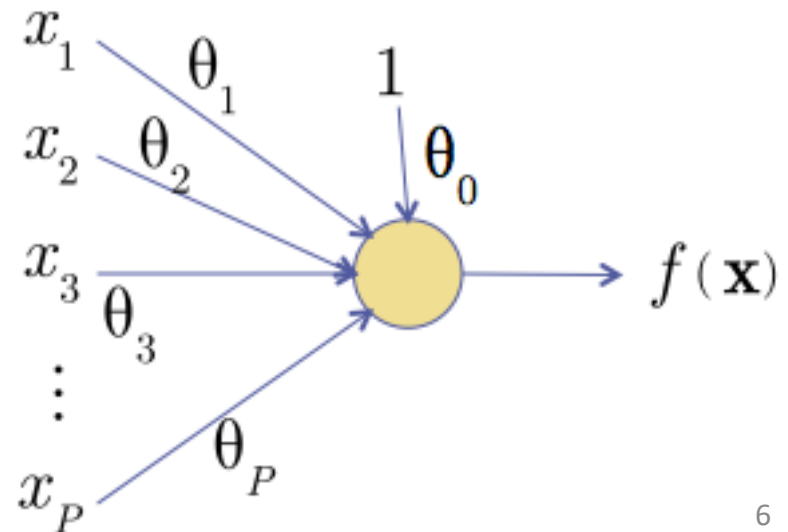
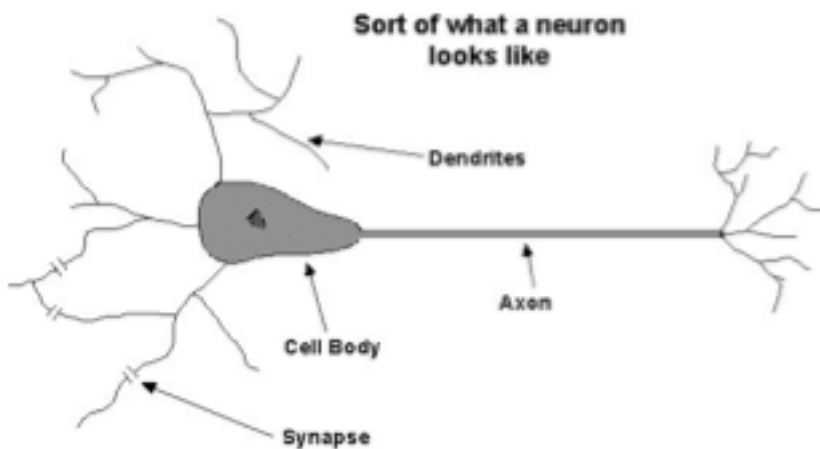


**Three correct trials with benign images follow
(the yellow button is correct).**

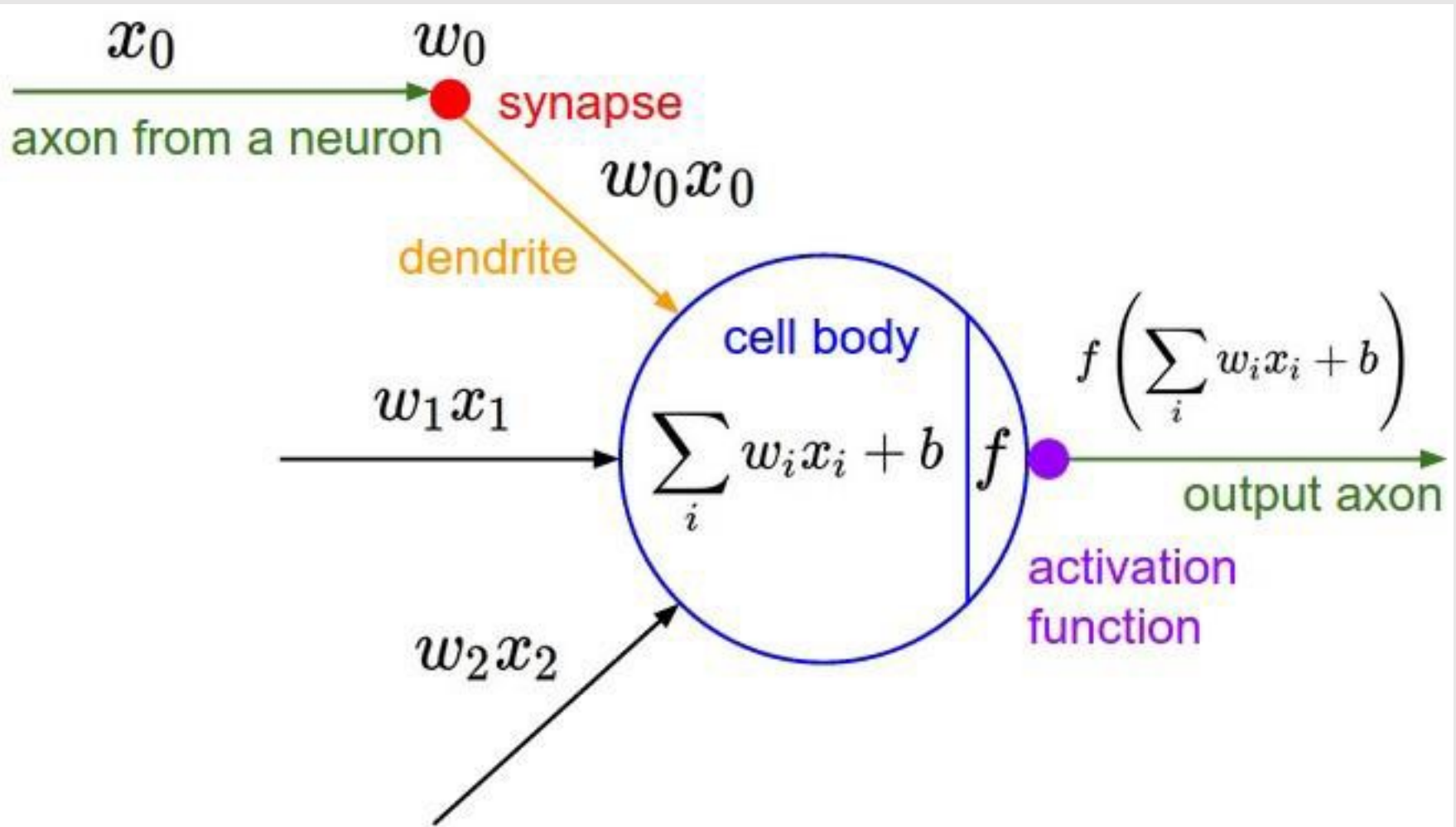
<https://www.youtube.com/watch?v=flzGjnJLyS0>

The Neuron Metaphor

- Neurons
 - Accept information from multiple inputs
 - Transmit information to other neurons
- Multiply inputs by weights along edges
- Apply some function to the inputs at each node

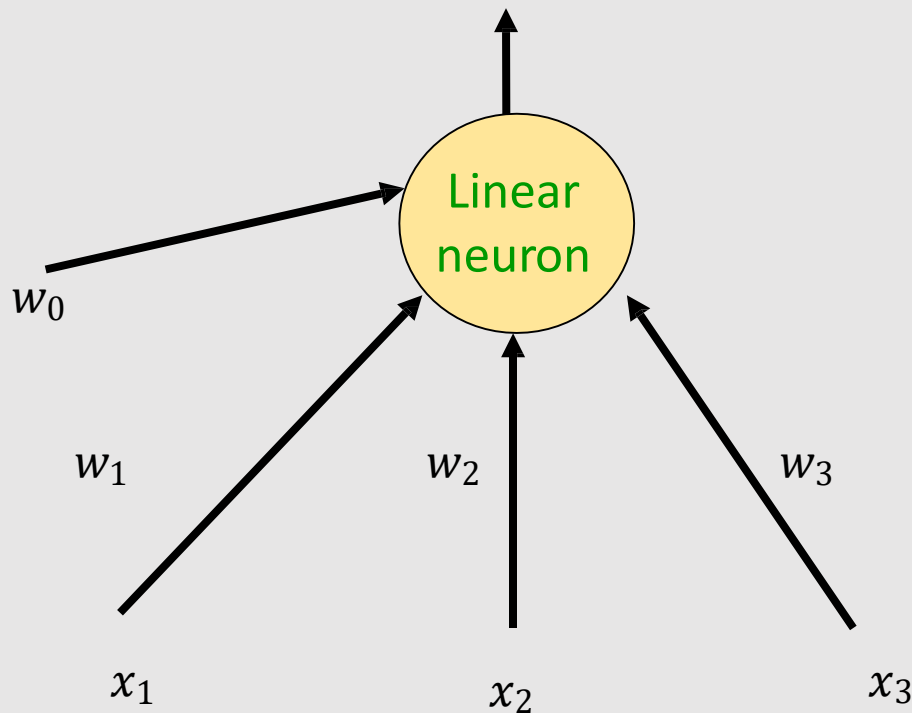


A Neuron Analogous to the Brain



Neural Network

- Network of neurons
- Linear neuron $w_0 + w_1x_1 + w_2x_2 + w_3x_3$



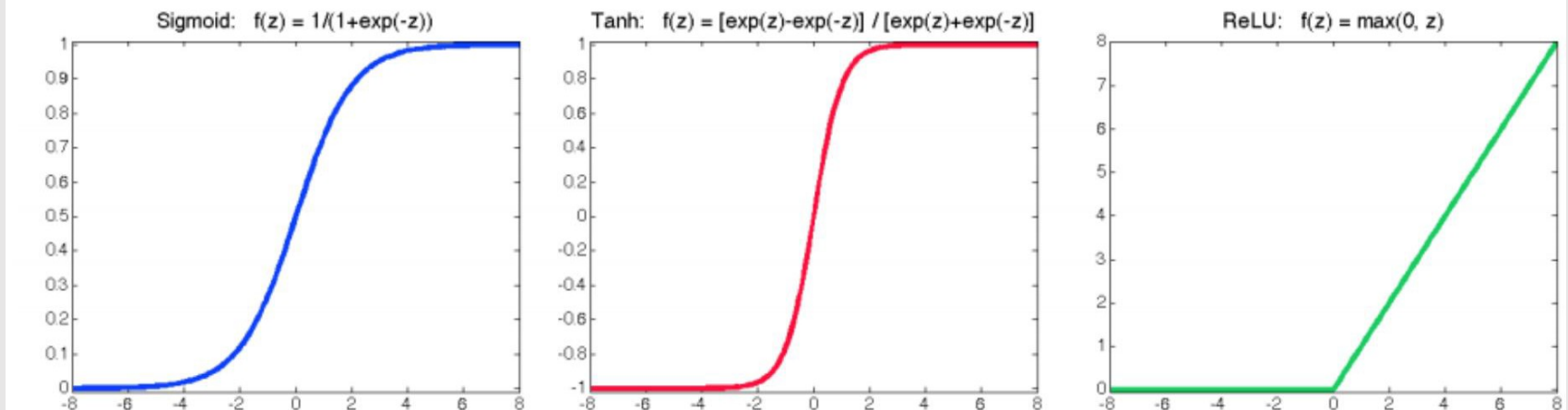
Activation Functions

- Commonly used activation functions

- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$

- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$

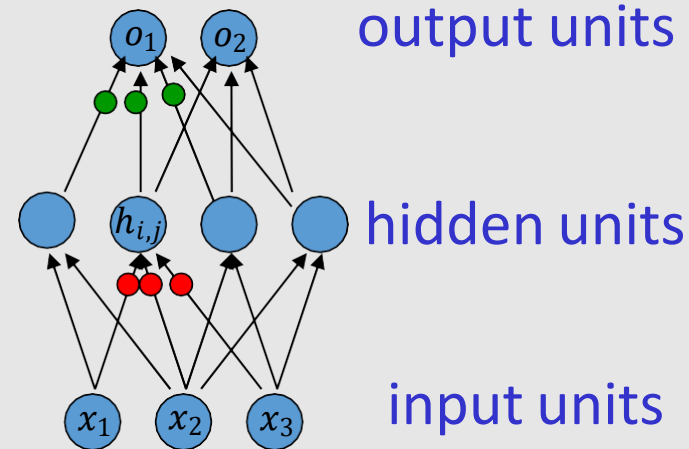
- ReLU (Rectified Linear Unit): $\text{ReLU}(z) = \max(0, z)$



Computation in Neural Networks

- Forward pass
 - Making **predictions (decisions)**
 - Plug in the input x , get the output y

$$\mathbf{o} = g \left((W^{(2)})^T \mathbf{h} + b^{(2)} \right)$$
$$\mathbf{h} = g \left((W^{(1)})^T \mathbf{x} + b^{(1)} \right)$$



- Backward pass (backpropagation for optimisation)
 - Compute the gradient of the **cost (loss/error)** function with respect to the weights to find good values for weights

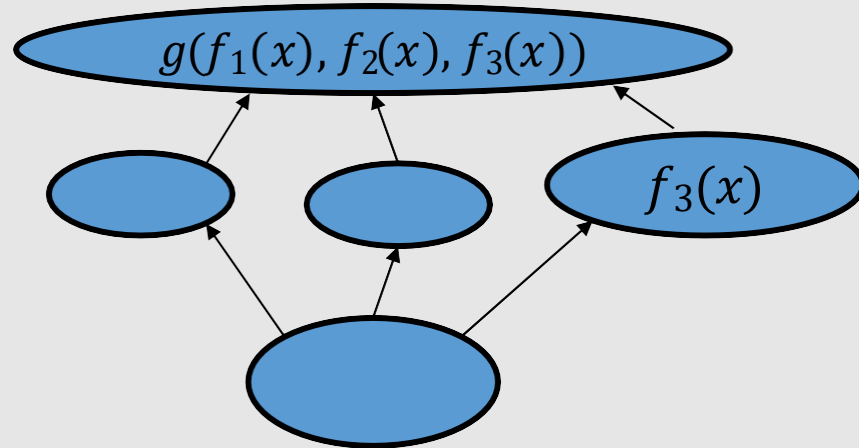
Autograd: Chain Rule

- Univariate chain rule

$$\frac{d}{dt} g(f(t)) = \frac{dg}{df} \cdot \frac{df}{dt}$$

- Multivariate chain rule

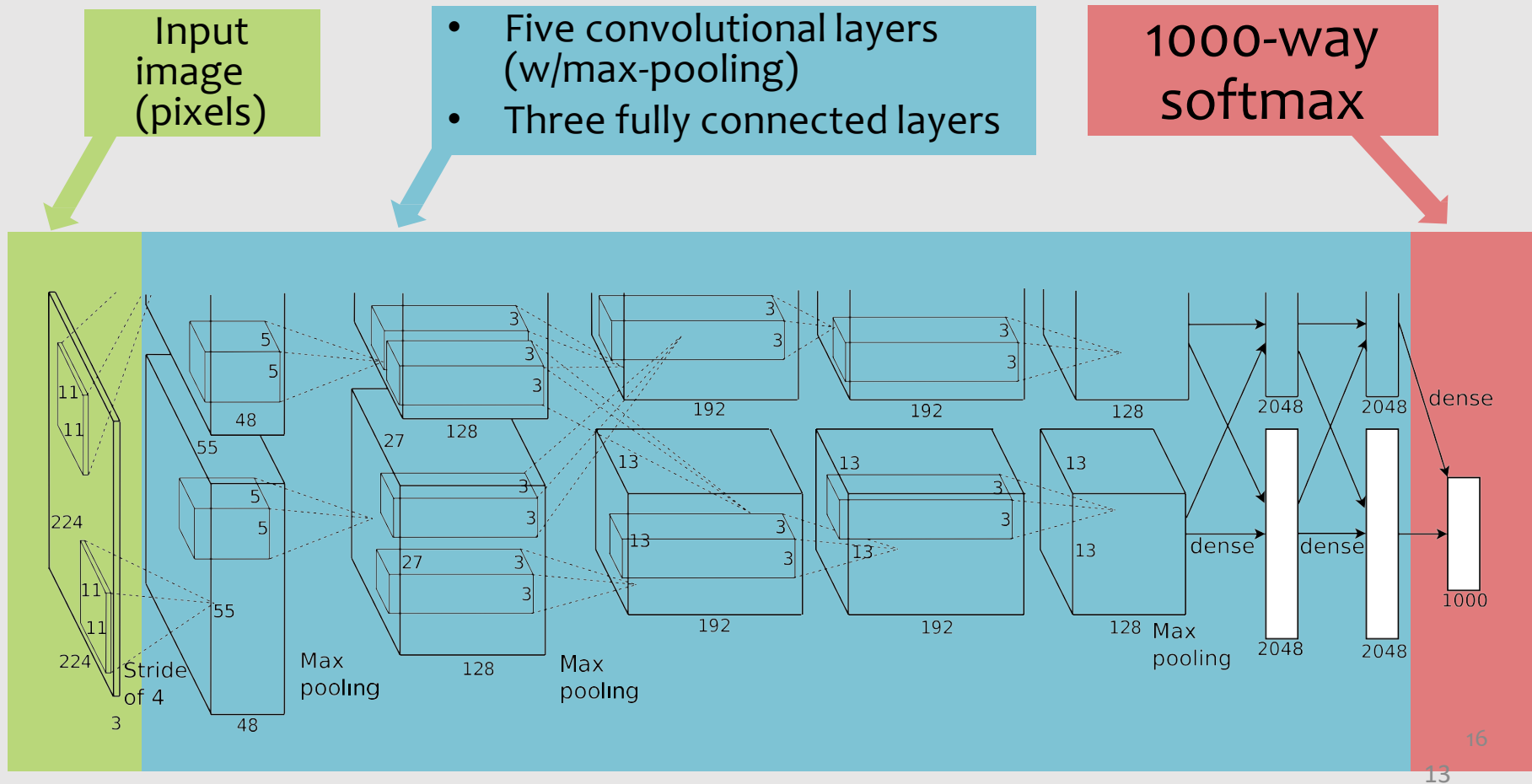
$$\frac{\partial g}{\partial x} = \sum \frac{\partial g}{\partial f_i} \frac{\partial f_i}{\partial x}$$



Week 7 Contents / Objectives

- Learning with Neurons
- **Neural Networks (NNs)**
- NN Decision Boundary & Features
- Convolutional NN Basics
- Convolutional NN Unboxing

AlexNet for ImageNet LSVRC-2010



Data, Model, Metric for Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

- Decision function/model

$$\hat{\mathbf{y}} = f_{\theta}(\mathbf{x}_i)$$

- Loss function/metric

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

Face



Face



Not a face



Examples: Linear regression, Logistic regression, Neural Network

Examples: Mean-squared error, Cross Entropy

Data, Model, Metric, Optimisation

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function/model

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

– Loss function/metric

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

– Objective function

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train/optimize with SGD: (take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

Data, Model, Metric, Optimisation

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function/model

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

– Loss function/metric


$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

– Objective function

Compute **gradients**
via *backpropagation*
Using *automatic*
differentiation

opposite the gradient)

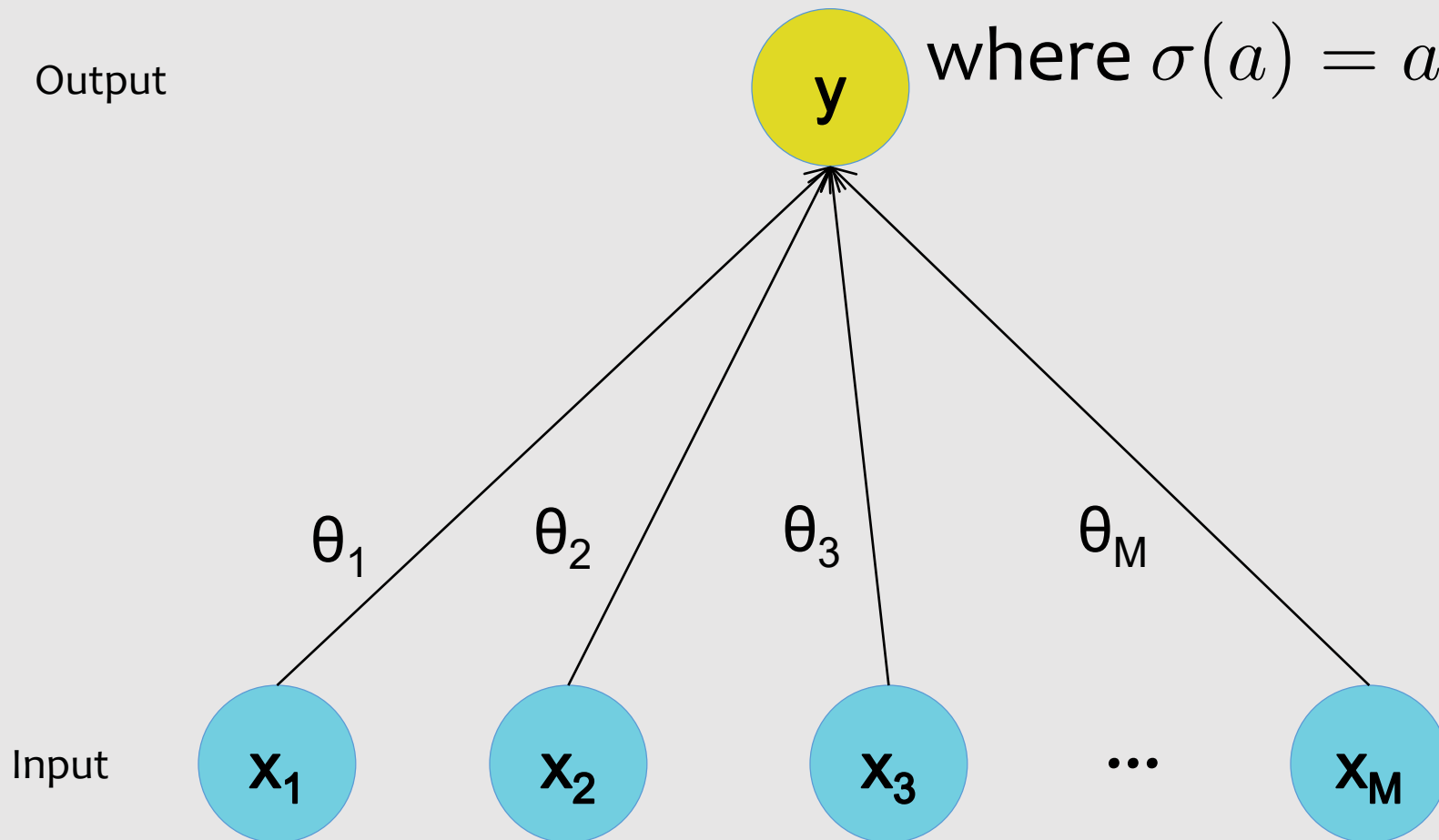

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

Linear Regression Model ($x \rightarrow y$)

$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

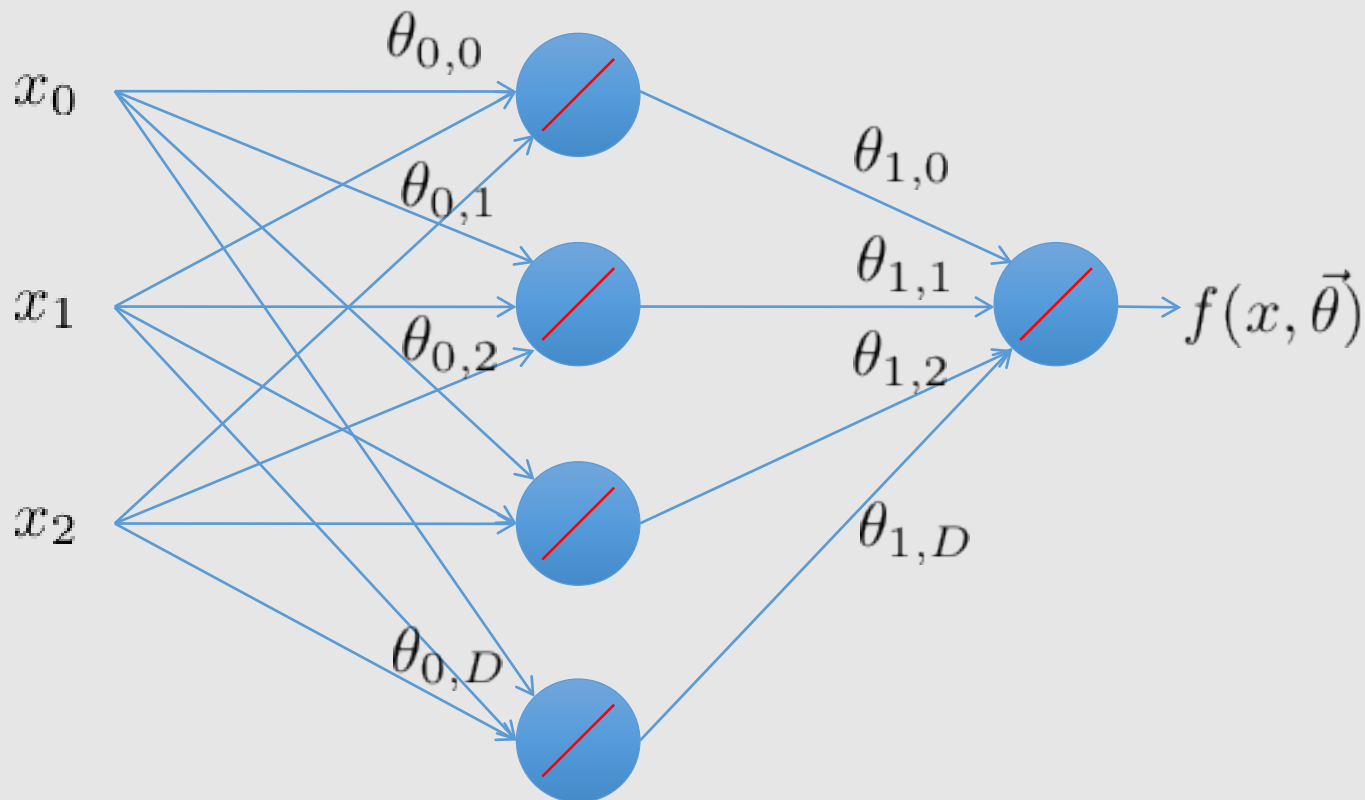
Output

where $\sigma(a) = a$



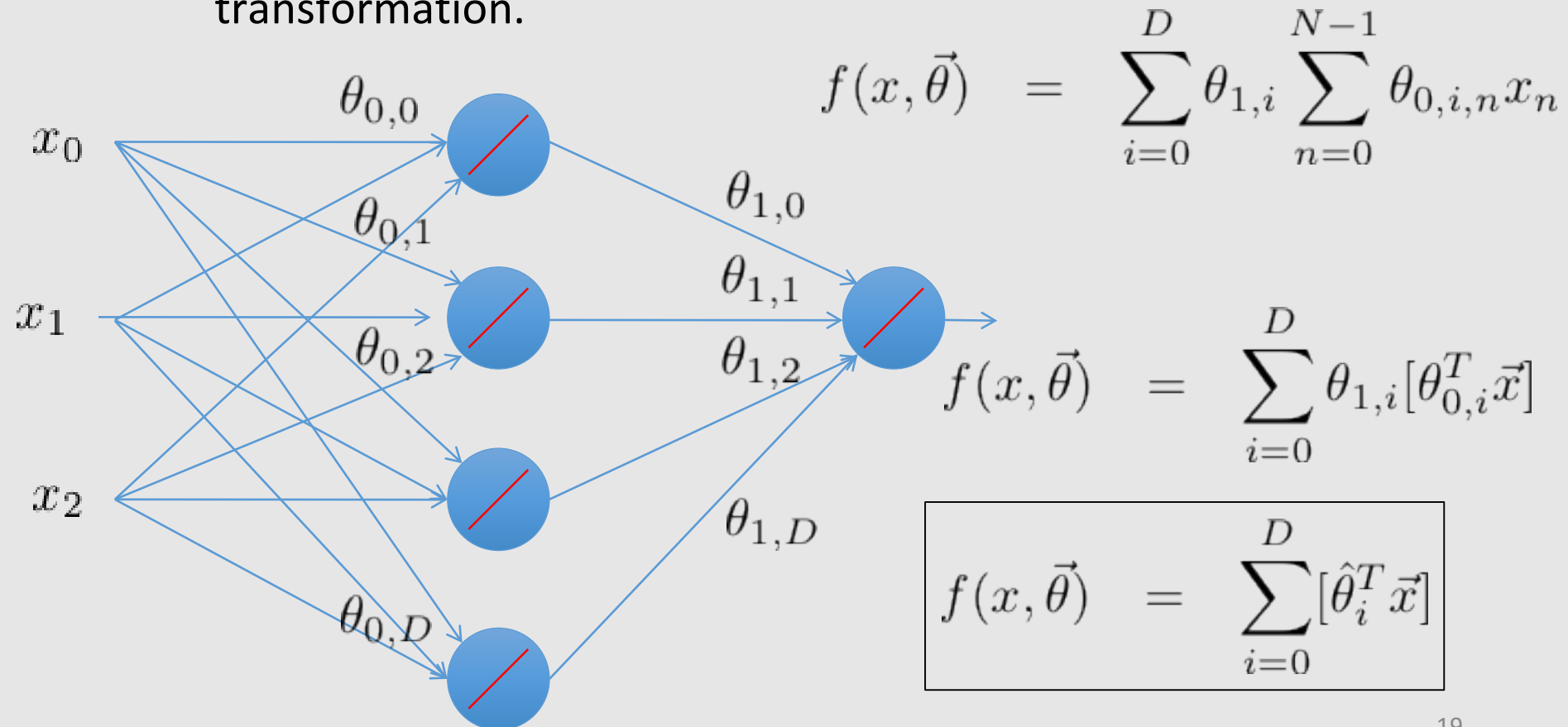
Linear Regression Neural Networks

- **Question:** What happens when we arrange **linear neurons** in a multilayer network?



Linear Regression Neural Networks

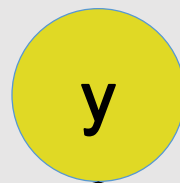
- Nothing special happens.
 - The product of two linear transformations is itself a linear transformation.



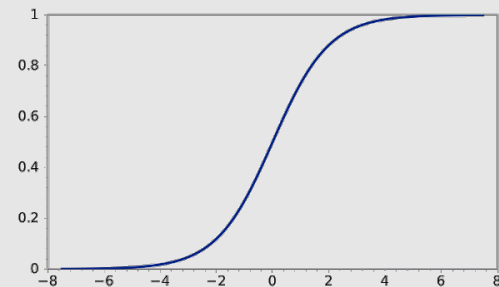
Logistic Regression Model ($x \rightarrow y$)

$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

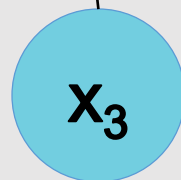
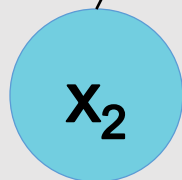
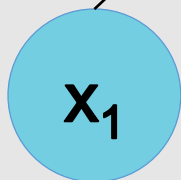
Output



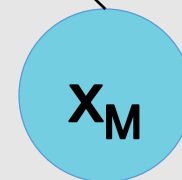
where $\sigma(a) = \frac{1}{1 + \exp(-a)}$



Input



...



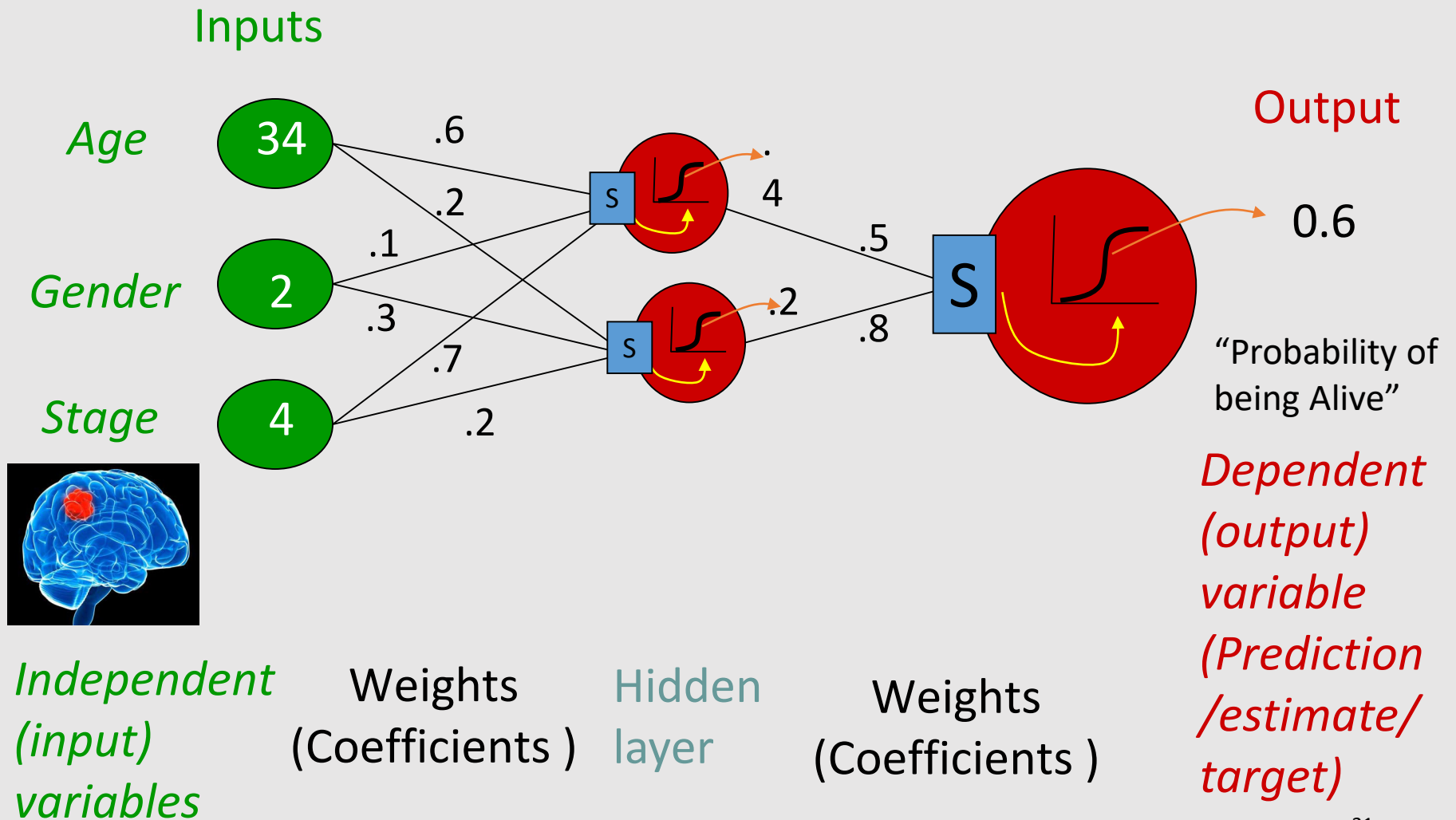
θ_1

θ_2

θ_3

θ_M

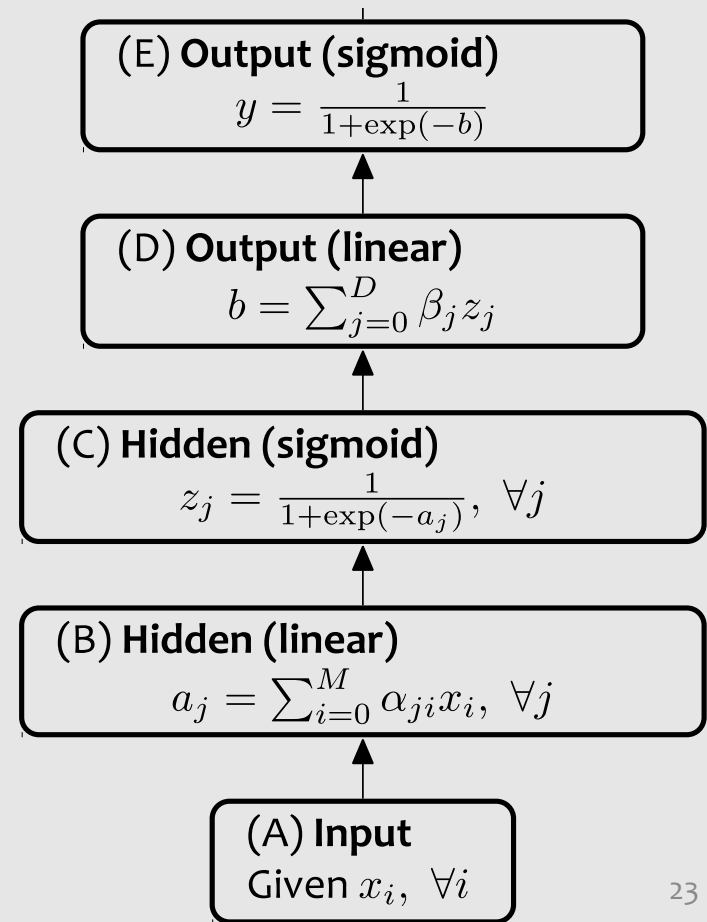
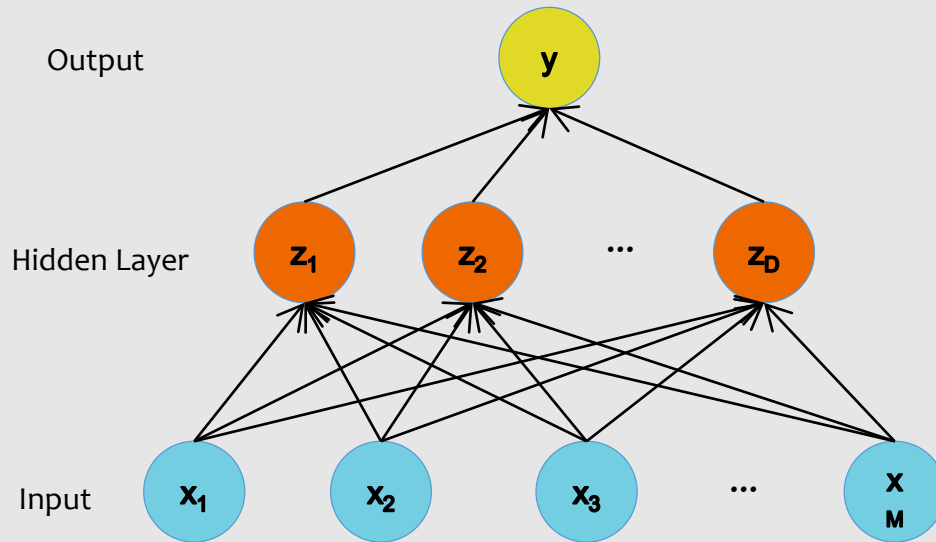
Logistic Regression Neural Networks



Week 7 Contents / Objectives

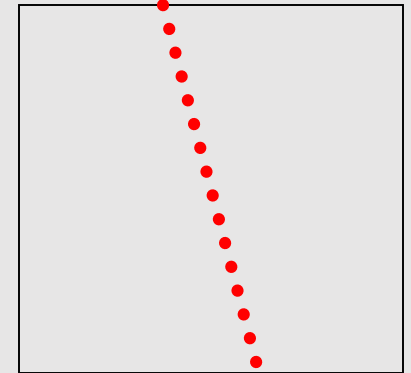
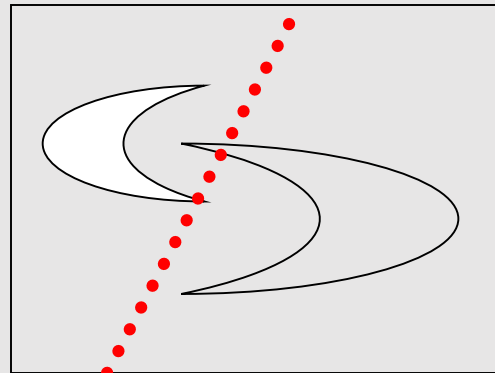
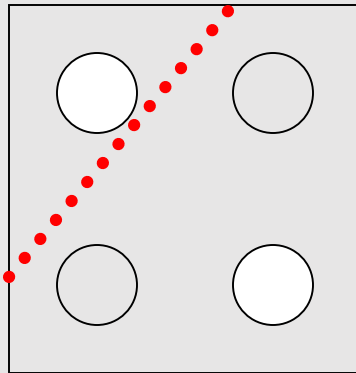
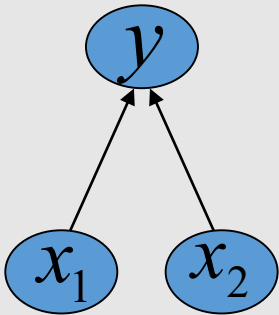
- Learning with Neurons
- Neural Networks (NNs)
- **NN Decision Boundary & Features**
- Convolutional NN Basics
- Convolutional NN Unboxing

Hidden Layers \rightarrow Decisions



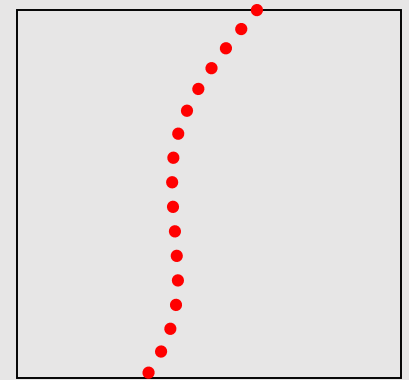
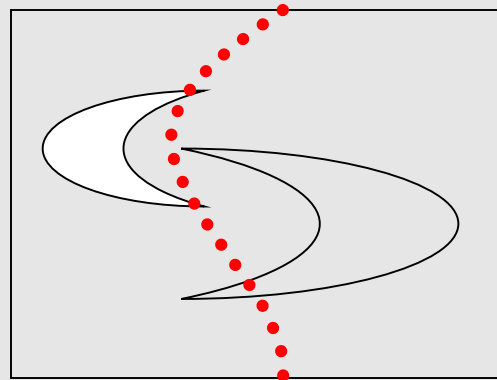
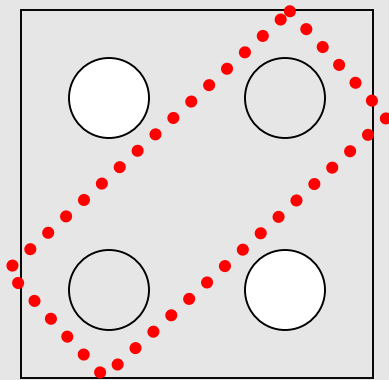
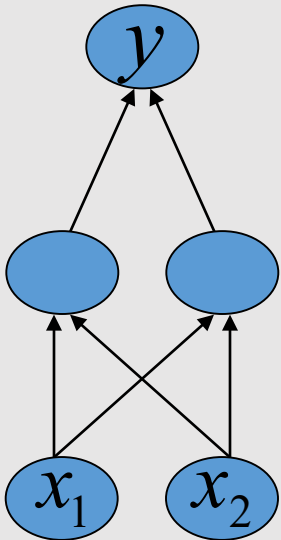
Decision Boundary

- 0 hidden layers: linear classifier
 - Hyperplanes

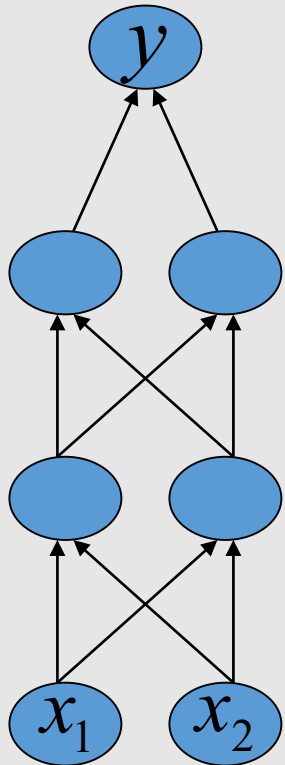


Decision Boundary

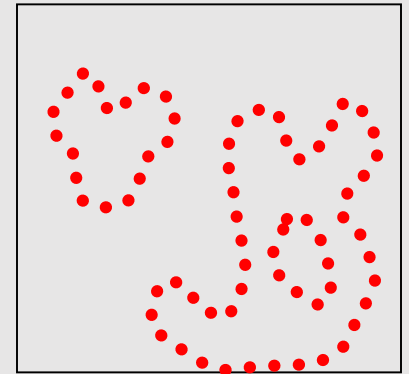
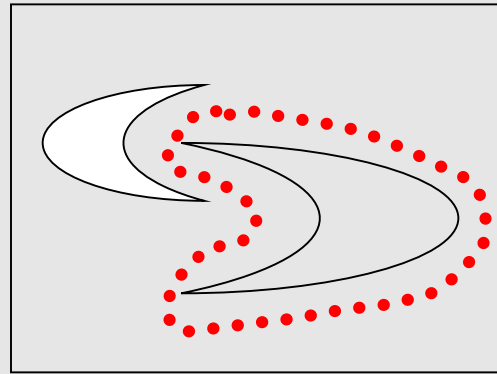
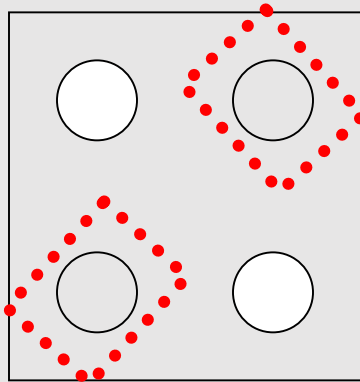
- 1 hidden layer
 - Boundary of **convex** region (open or closed)



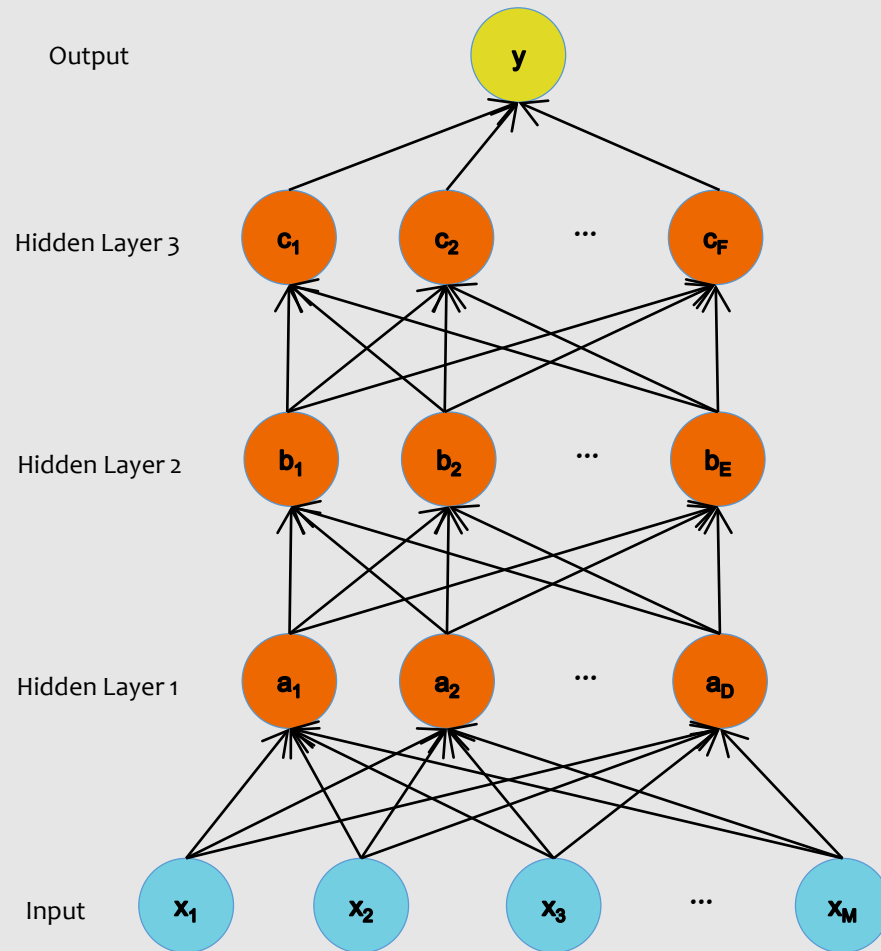
Decision Boundary



- 2 hidden layers
 - Combinations of convex regions

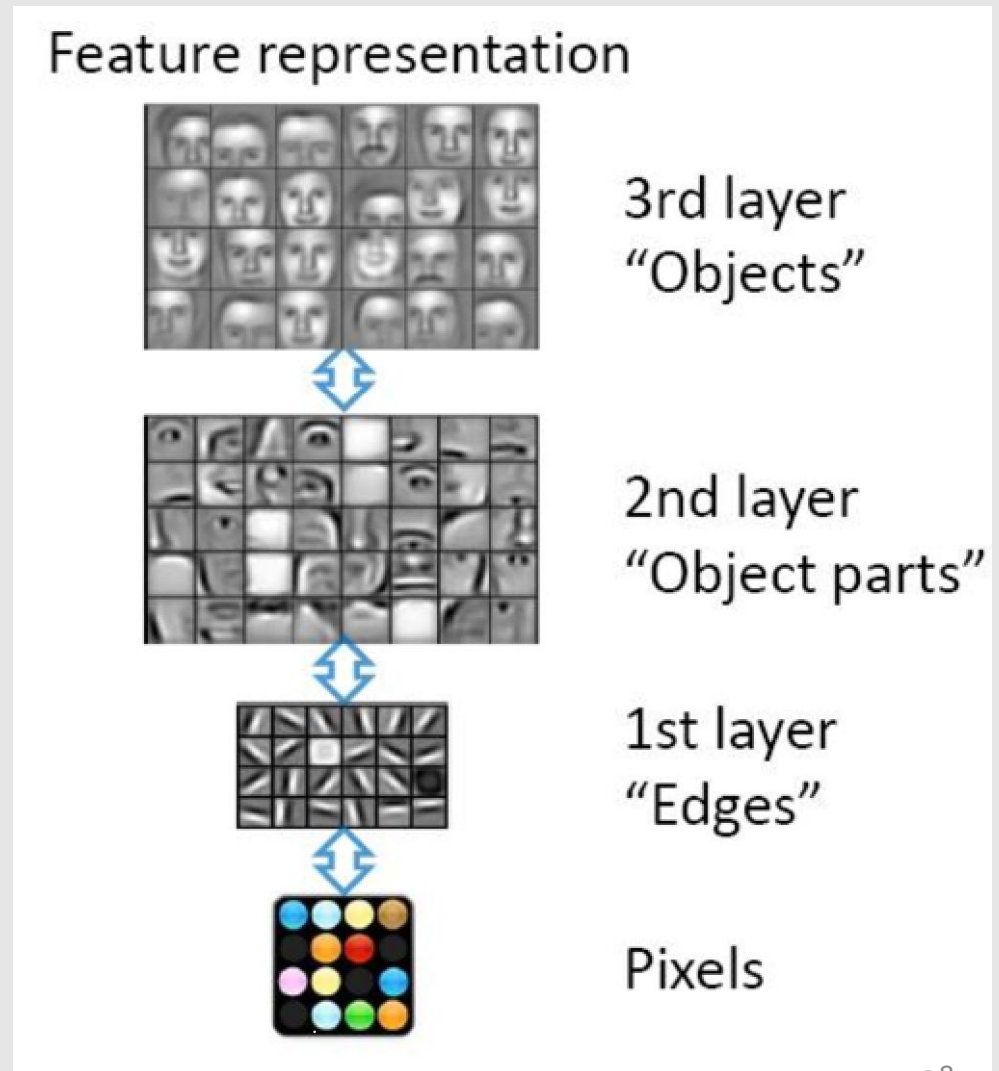


Deeper Networks



Different Levels of Abstraction

- We don't know the “right” levels of abstraction
- So let the model figure it out!



Different Levels of Abstraction

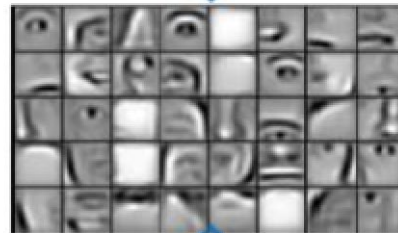
Face Recognition:

- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions

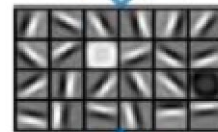
Feature representation



3rd layer
“Objects”



2nd layer
“Object parts”

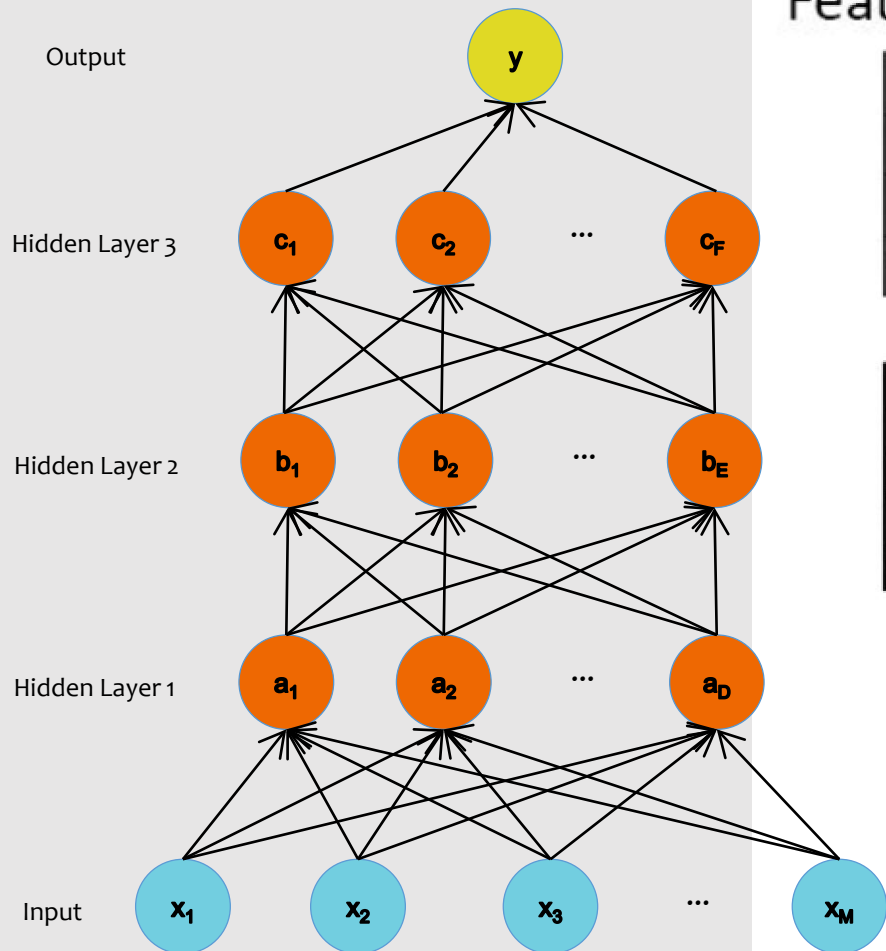


1st layer
“Edges”



Pixels

Different Levels of Abstraction



Feature representation



3rd layer
“Objects”



2nd layer
“Object parts”



1st layer
“Edges”

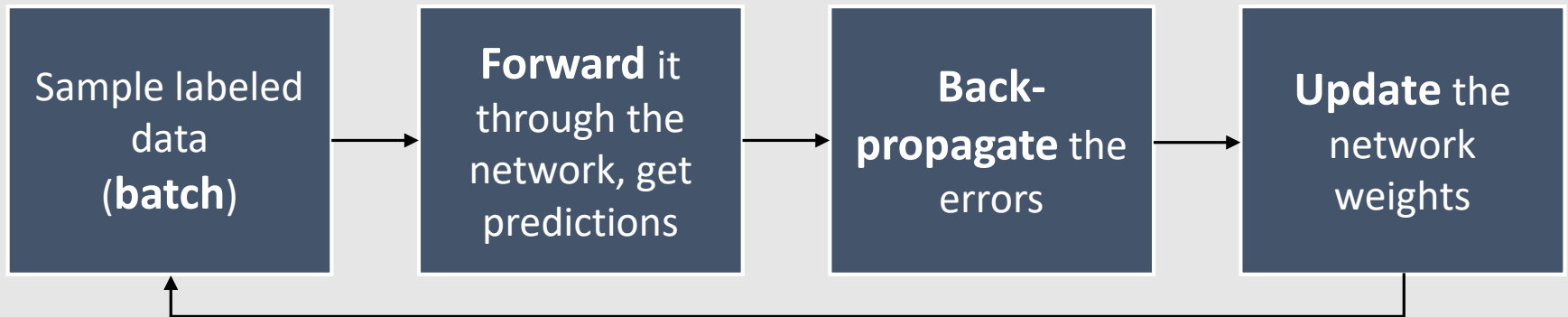


Pixels

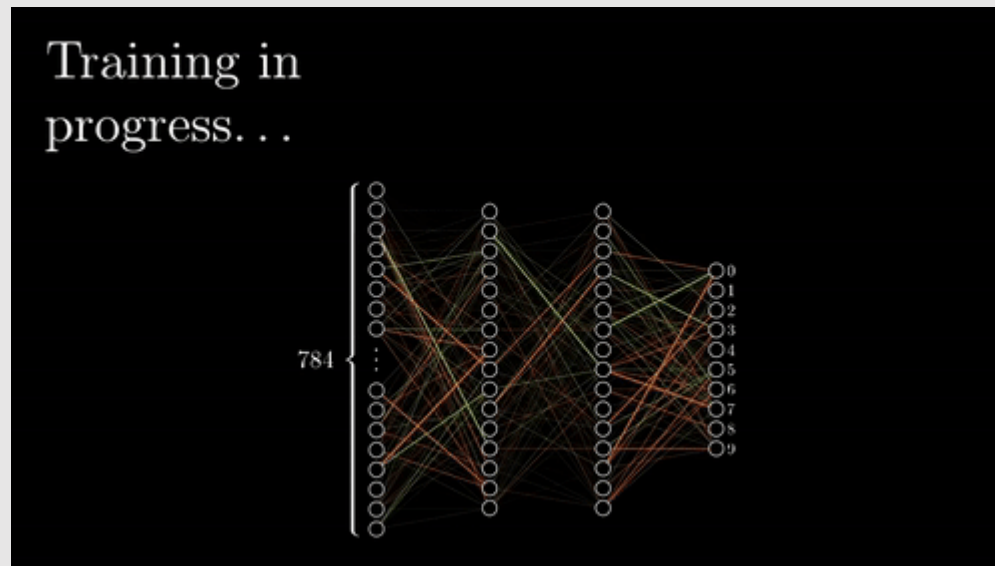
Machine Learning Ingredients

- **Data:** + pre-processing (& visualisation), e.g., $\mathcal{N}(0,1)$
- **Model**
 - Structure \sim Architecture \leftarrow expert knowledge
 - Must **specify** before ML, can optimise via cross validation (CV)
 - **Hyper-parameter**, e.g., prior, #degree, layer \leftarrow knowledge
 - Must **specify** (choices) and can optimise via CV (**tuning**)
 - Parameters (theta)
 - Compute/learn parameter, e.g., **weights**, bias \leftarrow optimisation alg.
- Evaluation **metric** (what's best): loss/error function
- **Optimisation:** (how to find the best) learnable parameters

Neural Network Training



Data → Model → Metric → Optimisation




Neural Network Ingredients

- **Data:** + pre-processing, e.g., $\mathcal{N}(0,1)$
- **Model**
 - Structure/Architecture: layered network
 - **Hyper-parameter:** layer specs, e.g. #layers, #neurons/units, activation function
 - Parameters (theta): layer weights & biases
- Evaluation metric (loss): max likelihood (min NLL), cross-entropy, etc.
- Optimisation: backpropagation (gradient-based)

Week 7 Contents / Objectives

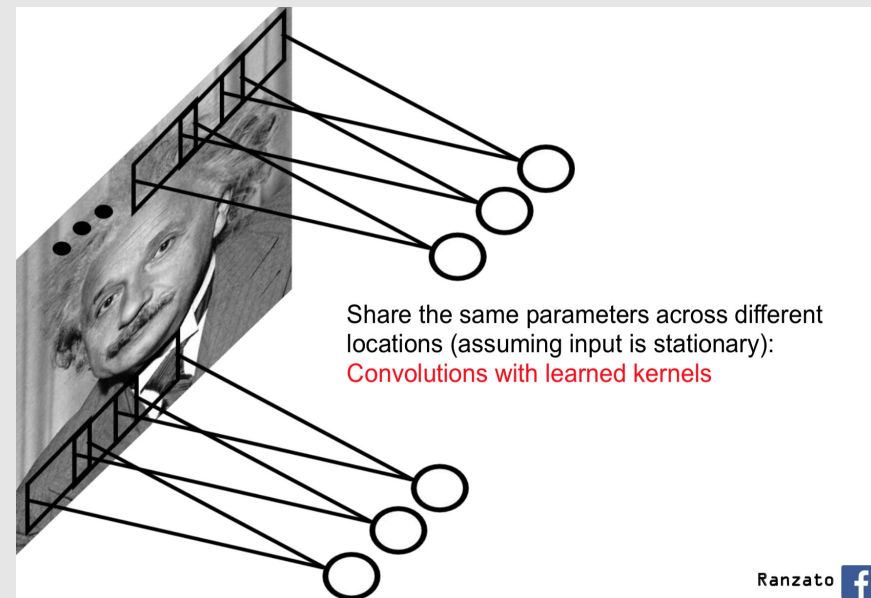
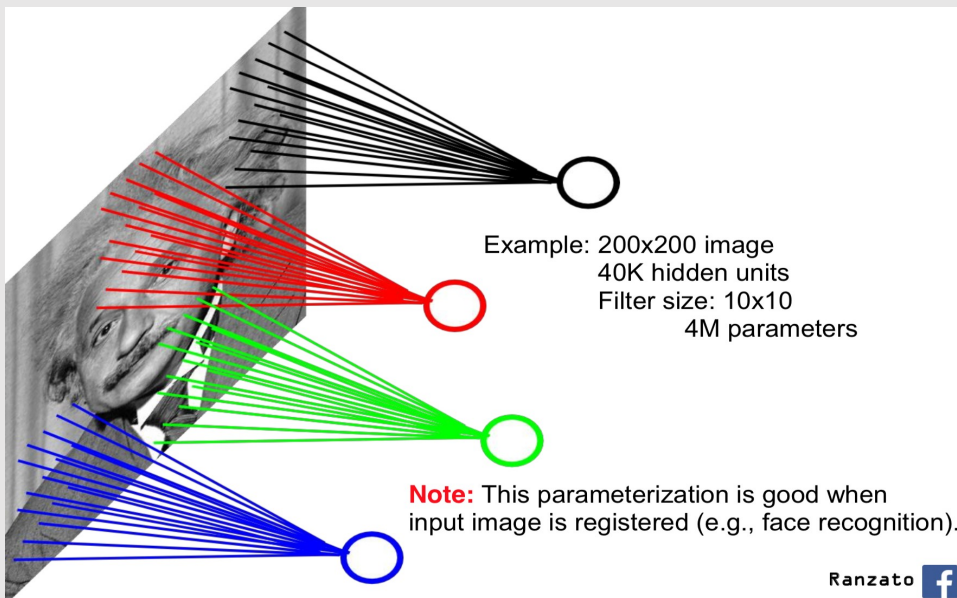
- Learning with Neurons
- Neural Networks (NNs)
- NN Decision Boundary & Features
- **Convolutional NN Basics**
- Convolutional NN Unboxing

Fully Connected (FC) Layer

- Linear layers such as linear/logistic regression
- What if our network is bigger?
 - Input image: 200×200 pixels, first hidden layer: 500 units
 - **Question:** How many weights for input \rightarrow 1st hidden?
 **20 million**
 - **Q:** Why is using an FC layer problematic for images?
 - Computing predictions (forward pass) will take a long time
 - A large number of weights requires a lot of training data to avoid overfitting
 - Small shift in image can result in large change in prediction
 - Not making use of the image geometry

Convolutional Neural Network

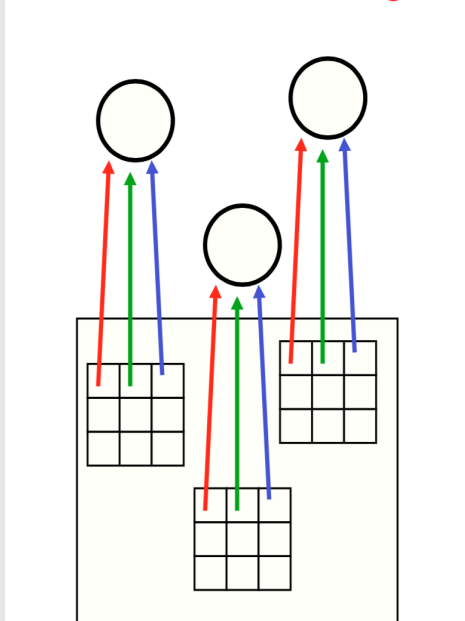
- Key ideas:
 - Locally-connected layers: look for local features in small regions of the image
 - Weight-sharing: detect the same local features across the entire image



Weight Sharing

- Each neuron on the higher layer detects the same feature, but in different locations on the lower layer

The red connections all have the same weight.



“Detecting” = the output (activation) is high if the feature is present

“Feature” = something in the image, like an edge, blob or shape

Forward Pass Example (Single Channel)

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

<https://developer.nvidia.com/sites/default/files/pictures/2018/convolution-2.gif>

- The kernel/filter (yellow) contains the trainable weights. In the above, the kernel size is 3×3 .
- The “*convolved features*” is another term for “convolution output”

Example of convolution

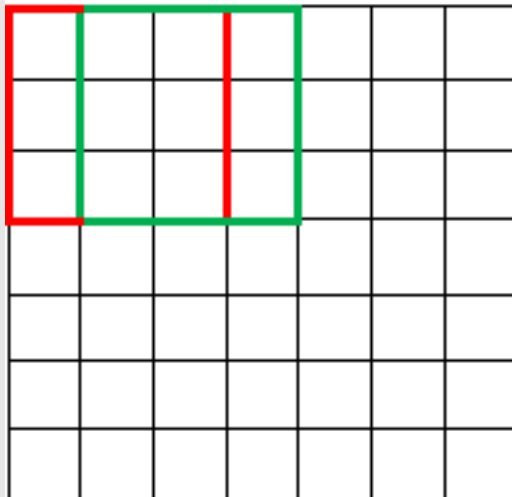
Greyscale input image: 7×7

Convolution **kernel**: 3×3

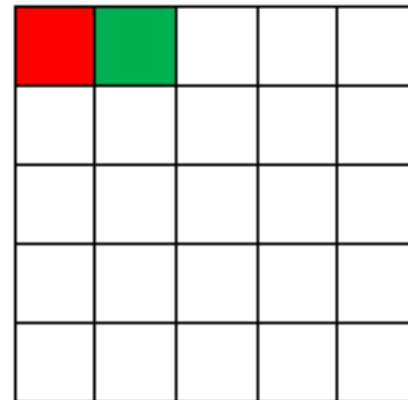
Questions:

- How many units are in the output?
- How many trainable weights are there?

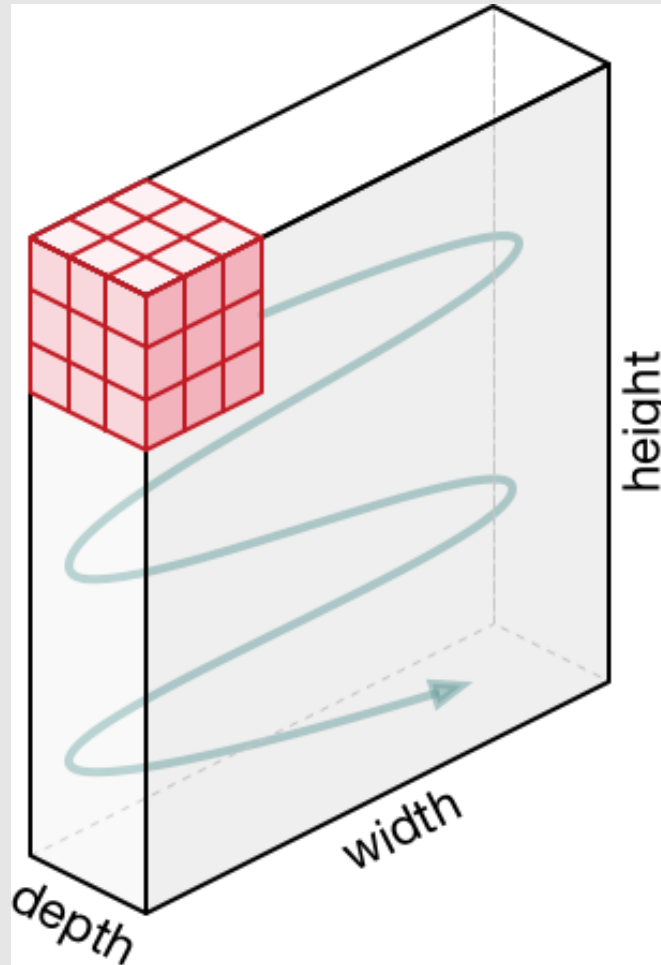
7 x 7 Input Volume



5 x 5 Output Volume



Convolution in RGB for colour images

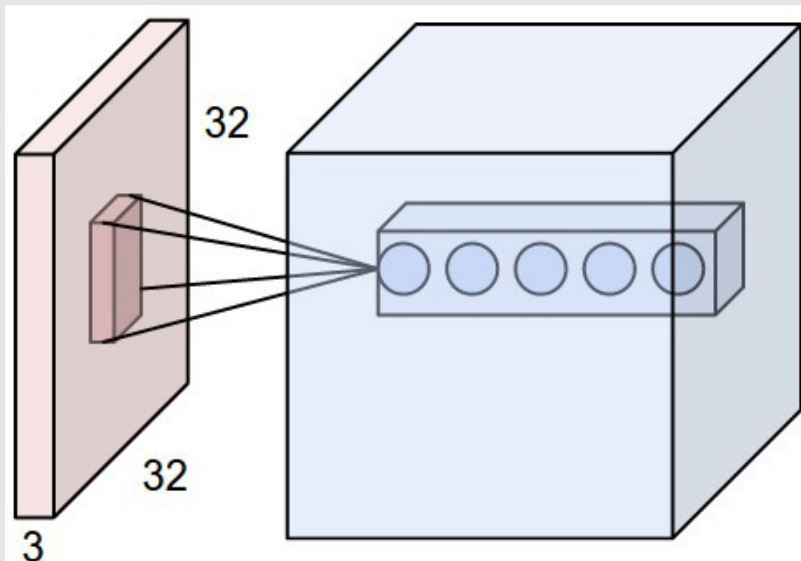


The kernel: a 3-D tensor! In this example, the kernel has size 3 × 3 × 3.

The first number 3: the number of **input channels** or **input feature maps**

Detecting Multiple Features

- **Q:** What if we want to detect many features of the input? (e.g. **both** horizontal edges and vertical edges, and maybe even other features?)
- **A:** Have many convolutional filters!

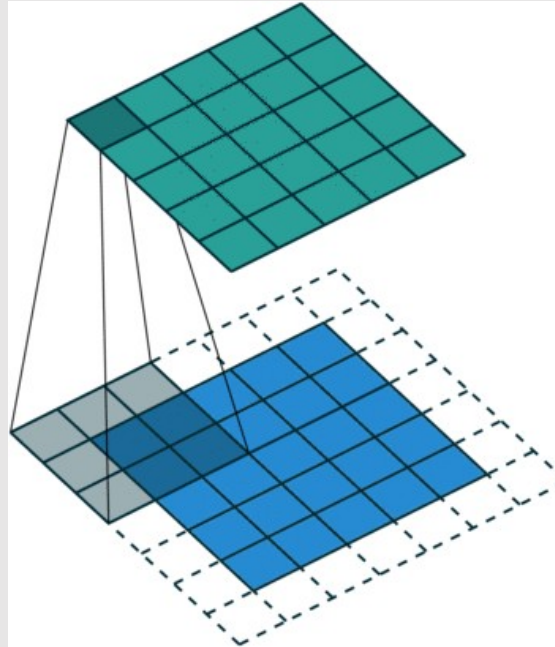


Input image size: $3 \times 32 \times 32$

Convolution kernel (4D): $\underline{3} \times 3 \times 3 \times \underline{5}$

- The number **3** is the number of **input channels** or **input feature maps**
- The number **5** is the number of **output channels** or **output feature maps**

Zero Padding

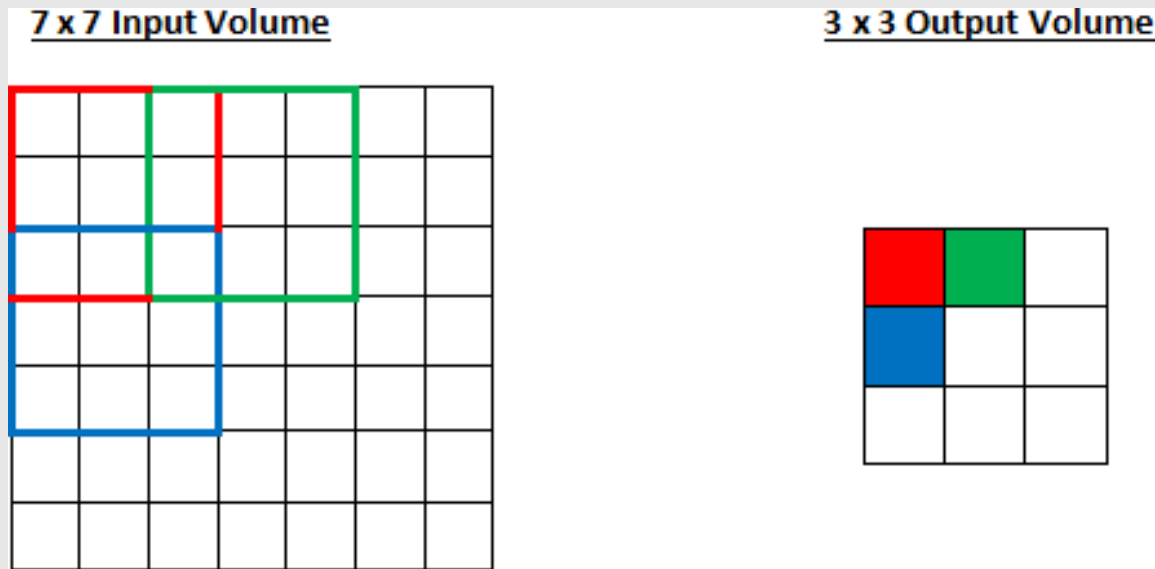


Add zeros around the border of the image (can add more than one pixel of zeros)

Question: Why might we want to add zero padding?

- Keep the next layer's width and height consistent with the previous
- Keep the information around the border of the image

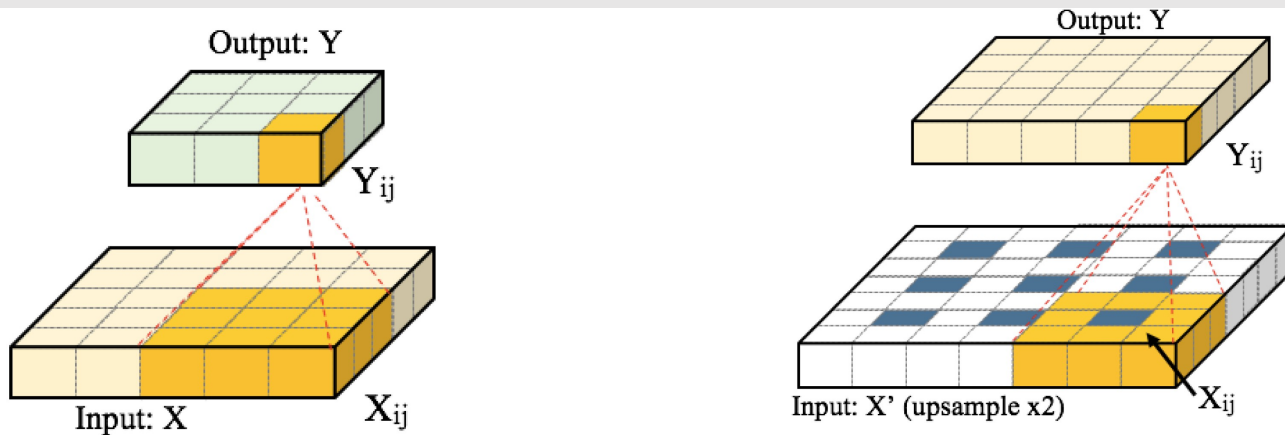
Strided Convolution



Shift the kernel by **2** (stride=2) when computing the next output feature.

Objective: to consolidate (summarise) information

Transpose Convolution Layer



(a) Convolutional layer: the input size is $W_1 = H_1 = 5$; the receptive field $F = 3$; the convolution is performed with stride $S = 1$ and no padding ($P = 0$). The output Y is of size $W_2 = H_2 = 3$.

(b) Transposed convolutional layer: input size $W_1 = H_1 = 3$; transposed convolution with stride $S = 2$; padding with $P = 1$; and a receptive field of $F = 3$. The output Y is of size $W_2 = H_2 = 5$.

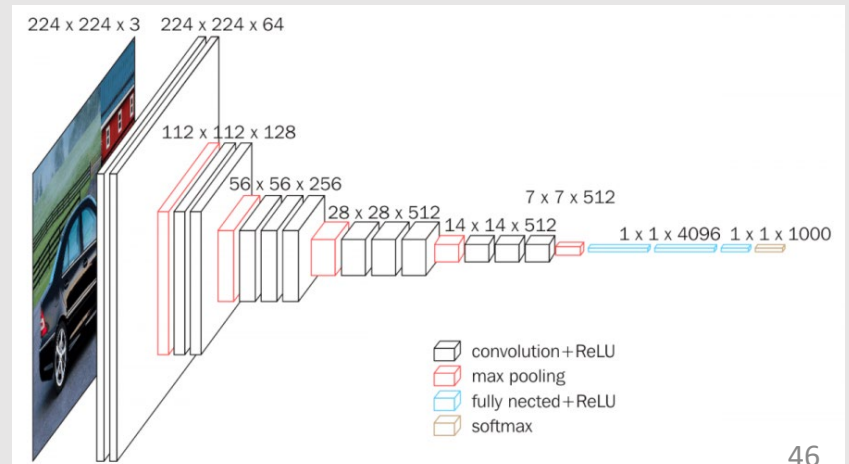
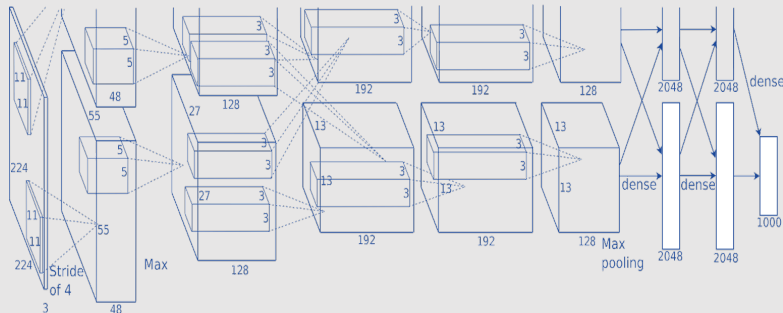
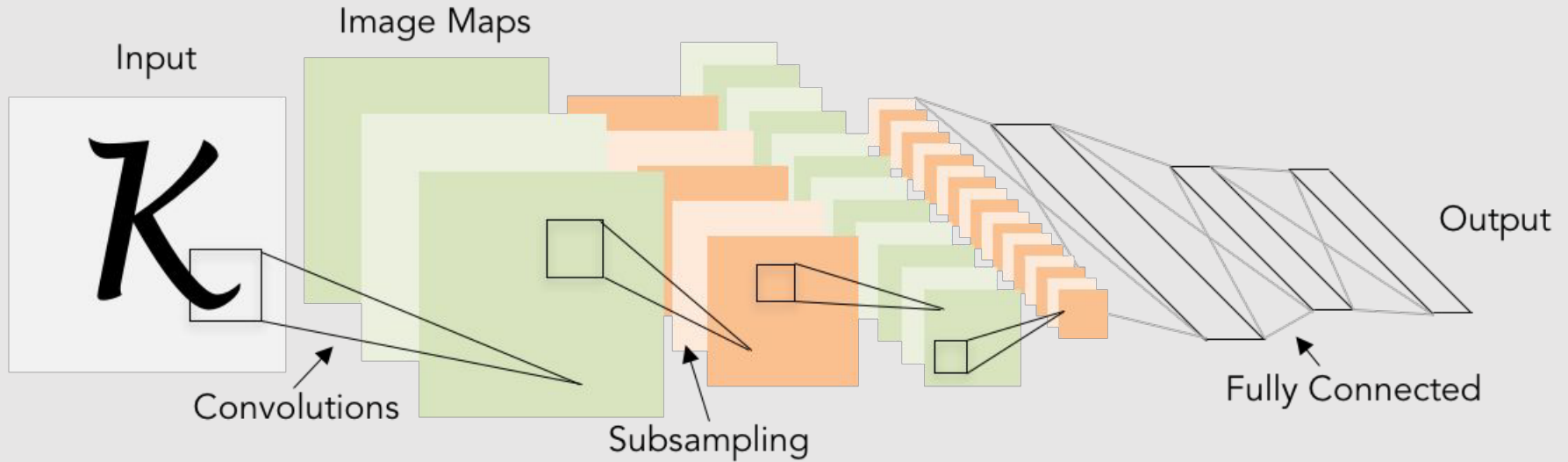
<https://www.mdpi.com/2072-4292/9/6/522/htm>

More at https://github.com/vdumoulin/conv_arithmetic

Week 7 Contents / Objectives

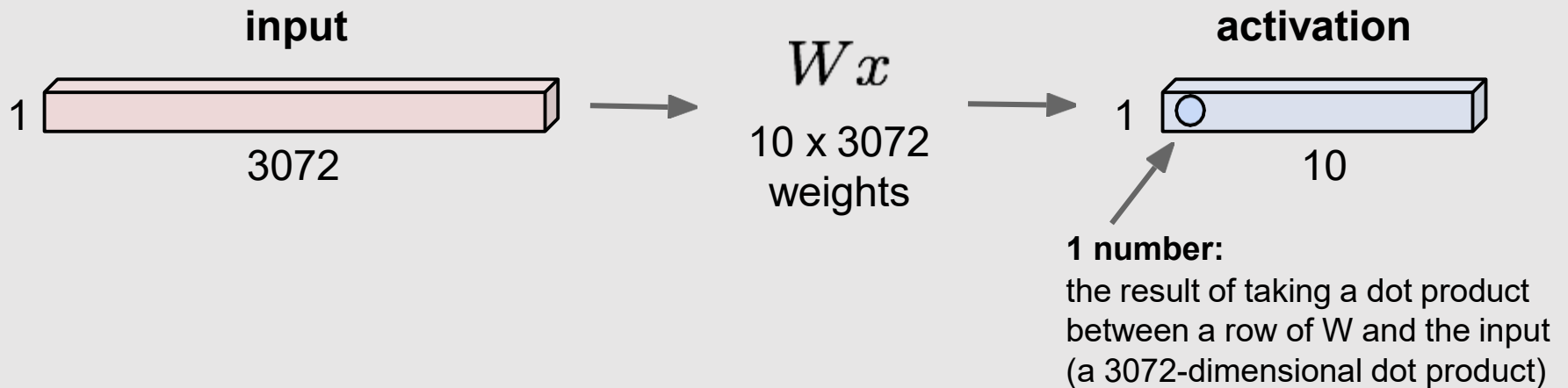
- Learning with Neurons
- Neural Networks (NNs)
- NN Decision Boundary & Features
- Convolutional NN Basics
- **Convolutional NN Unboxing**

Convolutional Neural Networks



Fully Connected Layer

32x32x3 image \rightarrow stretch to 3072 x 1

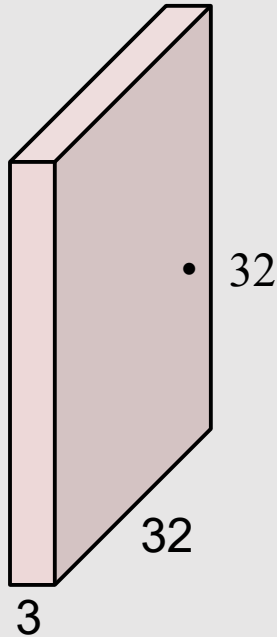


Convolution Layer

Tensor: Preserve spatial structure

Filters always extend the full depth of the input volume

- 32x32x3 image

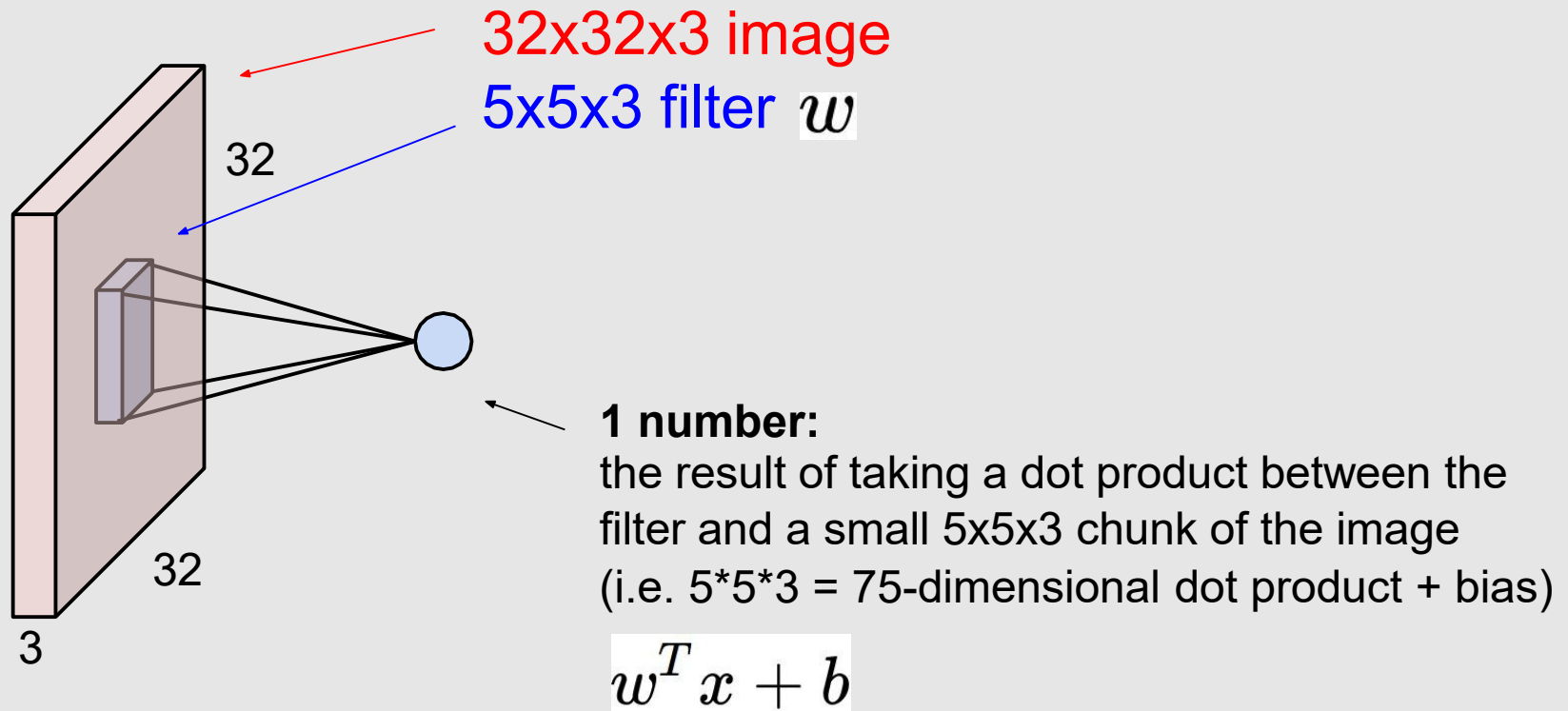


- 5x5x3 filter

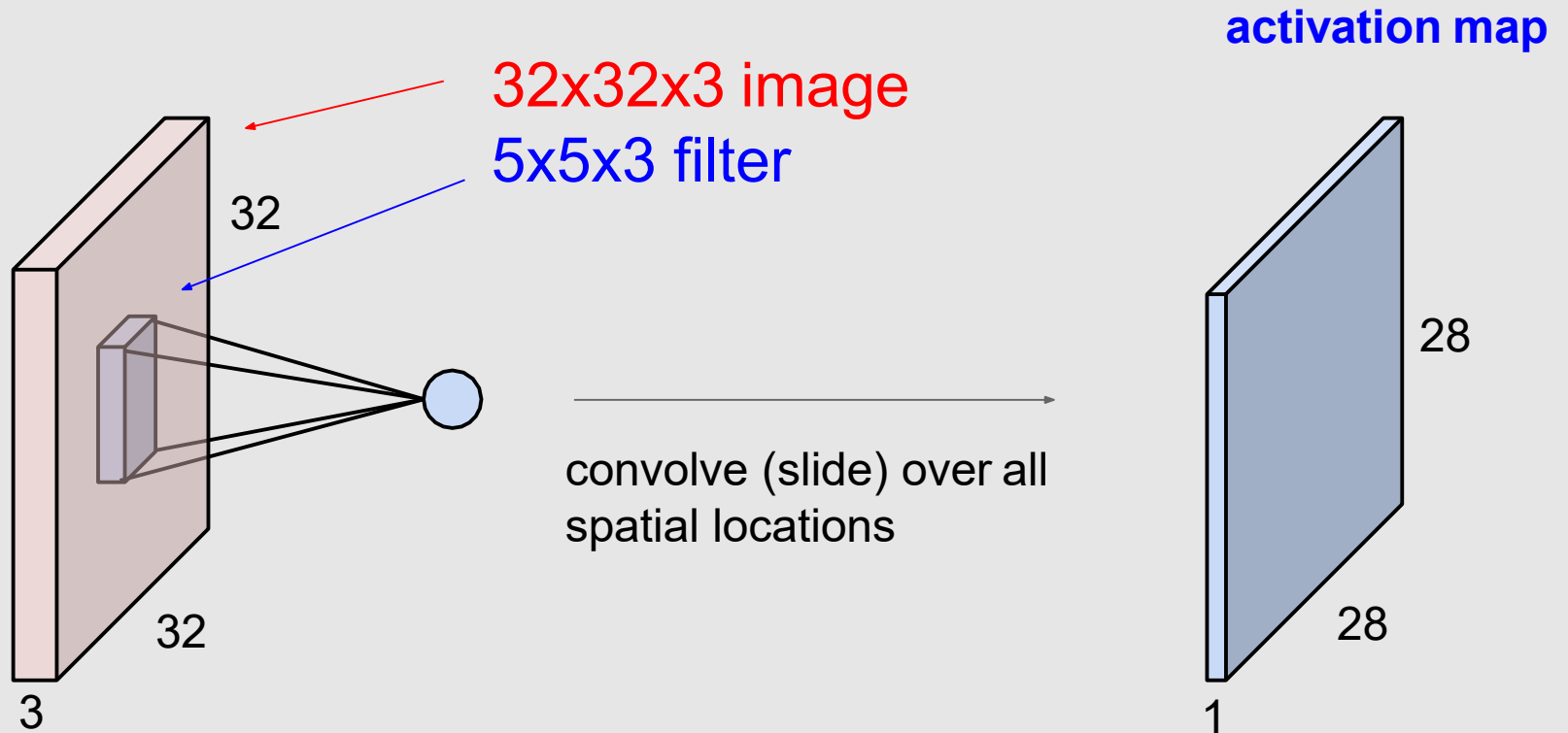


- **Convolve** the filter with the image
- i.e. “slide over the image spatially, computing dot products”

Convolution Layer

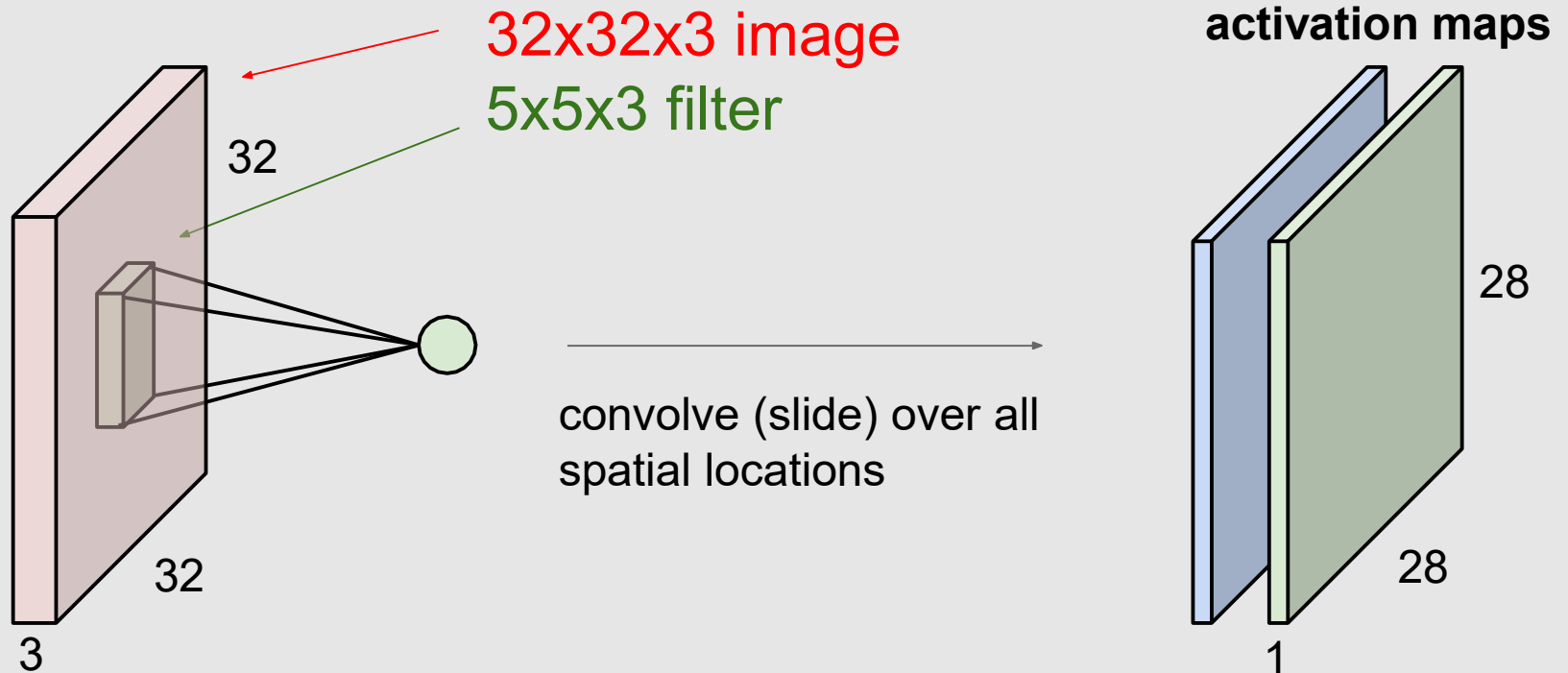


Convolution Layer



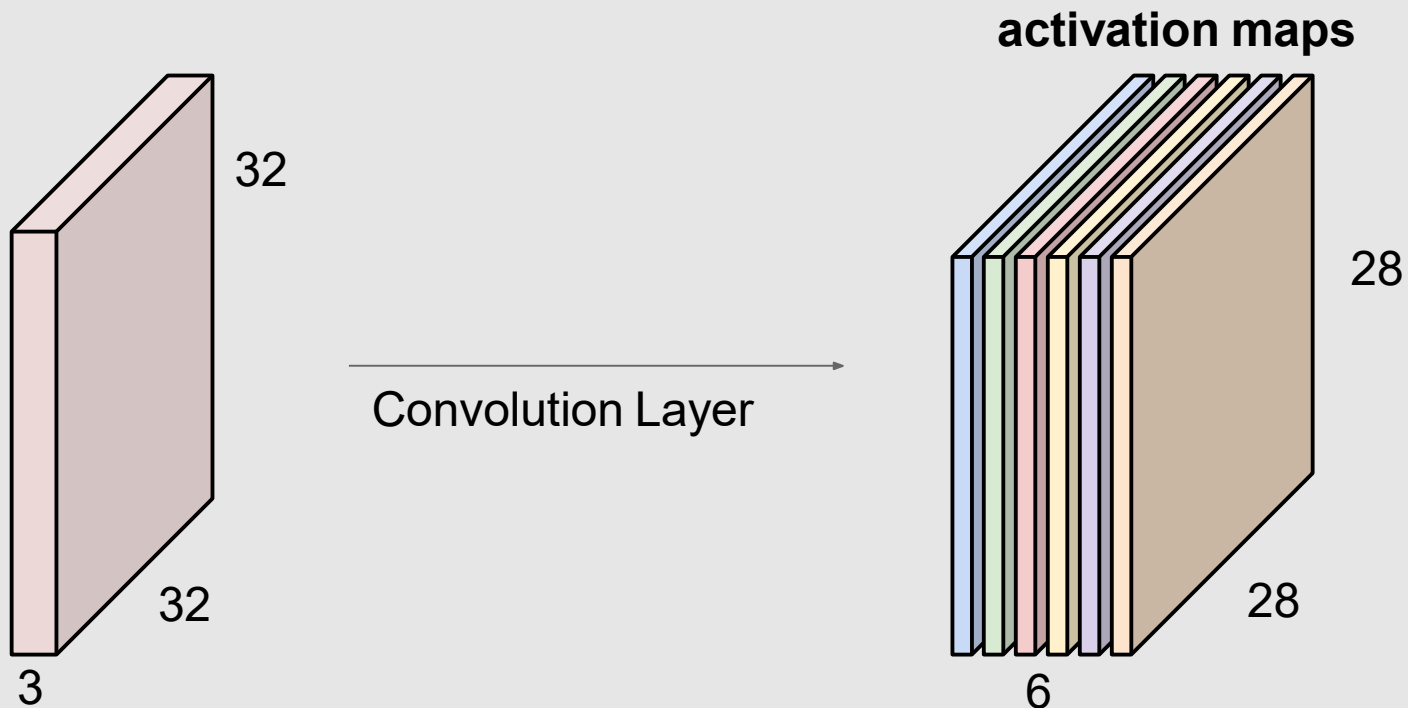
Convolution Layer

consider a second, **green** filter



Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size 28x28x6!

Convolution Operation

7

7

7x7 input (spatially)
assume 3x3 filter

Convolution Operation

7

7

7x7 input (spatially)
assume 3x3 filter

Convolution Operation

7

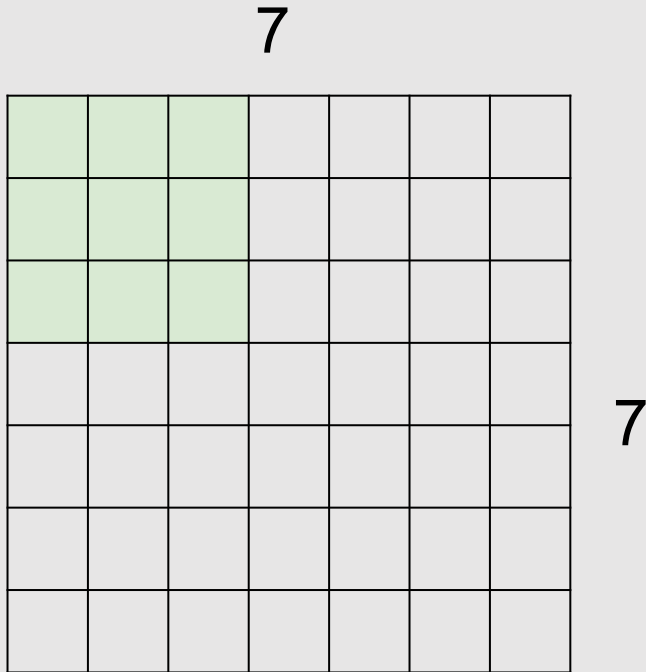
7

7x7 input (spatially)
assume 3x3 filter

→ **5x5**
output

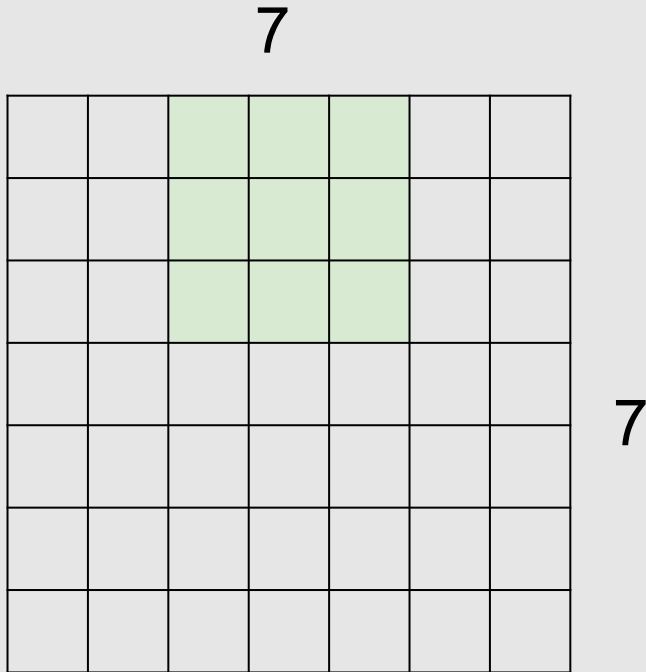
After three more sliding and dot products

Convolution with **Stride**



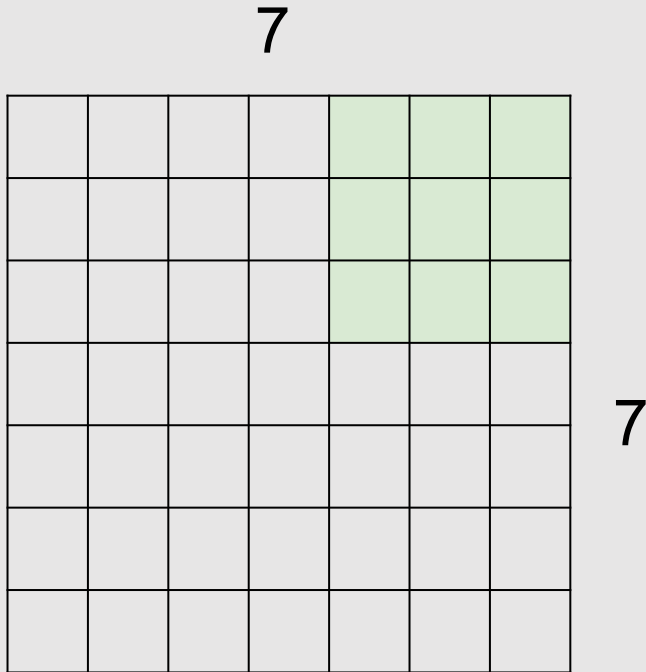
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution with Stride



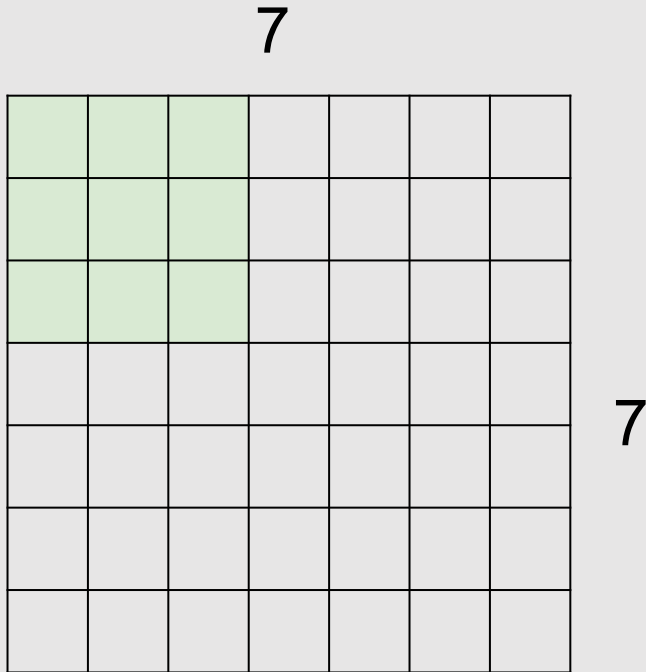
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution with **Stride**



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
→ **3x3 output!**

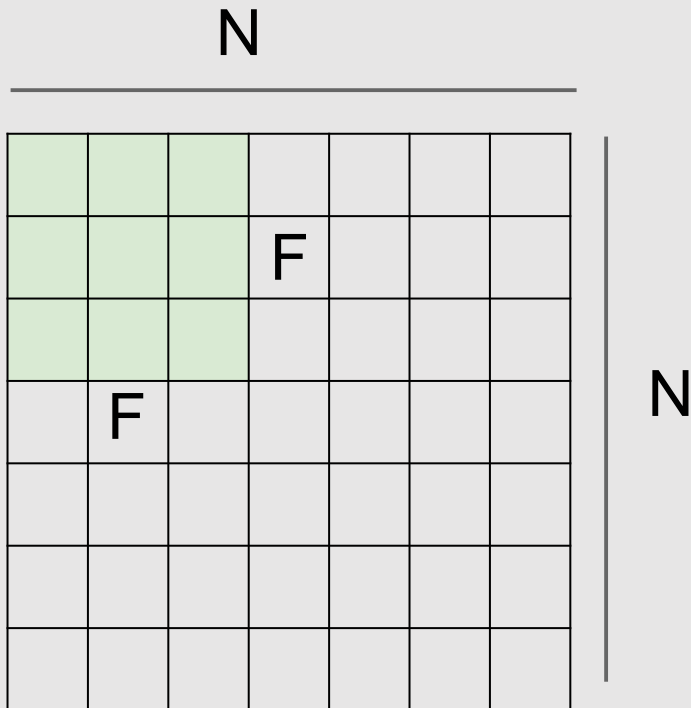
Convolution with Stride



Question: 7x7 input
(spatially) assume 3x3 filter
applied **with stride 3**?

doesn't fit!
cannot apply 3x3 filter
on 7x7 input with stride
3 (unless ignoring parts).

Convolution – Size of output



Output size:

$$(N - F) / \text{stride} + 1$$

e.g. $N = 7$, $F = 3$:

$$\text{stride } 1 \Rightarrow (7 - 3) / 1 + 1 = 5$$

$$\text{stride } 2 \Rightarrow (7 - 3) / 2 + 1 = 3$$

$$\text{stride } 3 \Rightarrow (7 - 3) / 3 + 1 = 2.33$$

Zero Padding

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

in general, common to see CONV layers with stride 1, filters of size $F \times F$, and zero-padding with $(F-1)/2$. (will preserve size spatially)

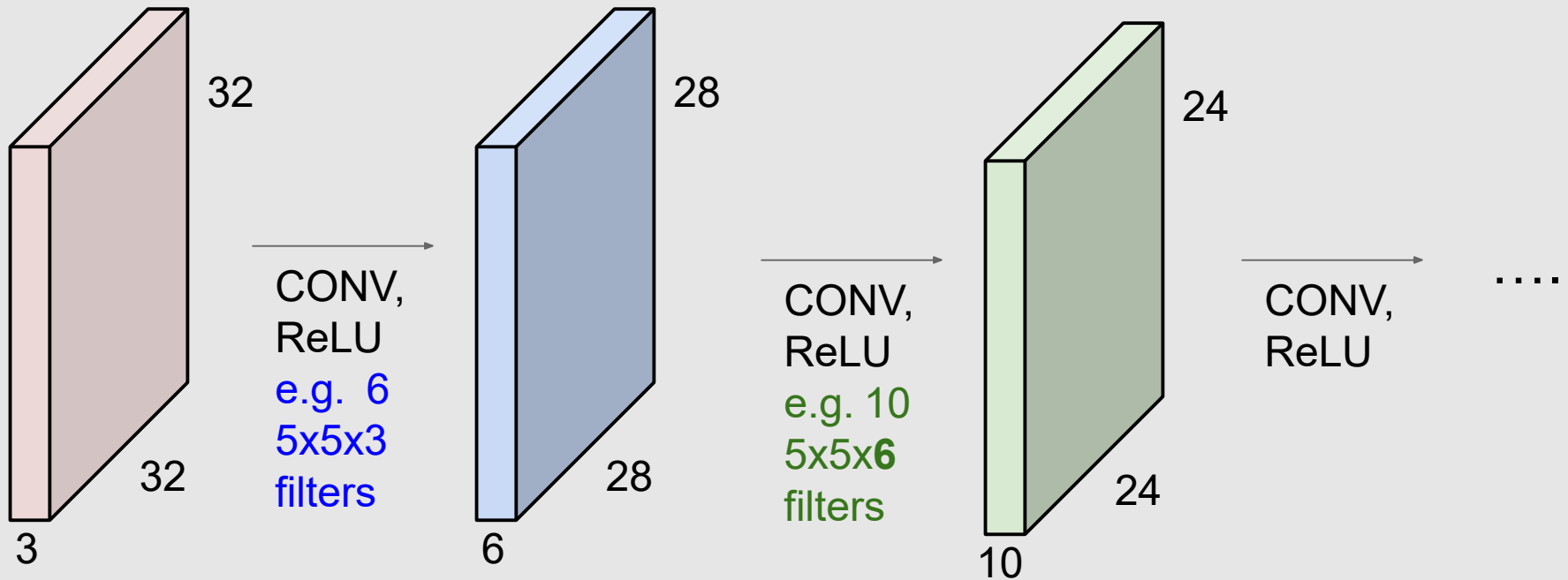
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2



$F = 7 \Rightarrow$ zero pad with 3

Convolution Shrinks

Example: 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! ($32 \rightarrow 28 \rightarrow 24 \dots$).

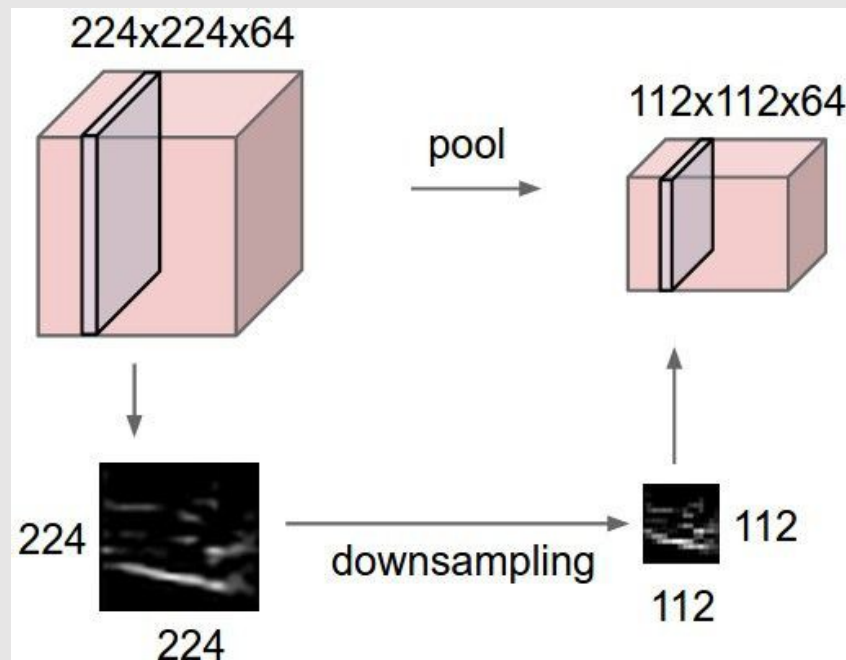


Exercises

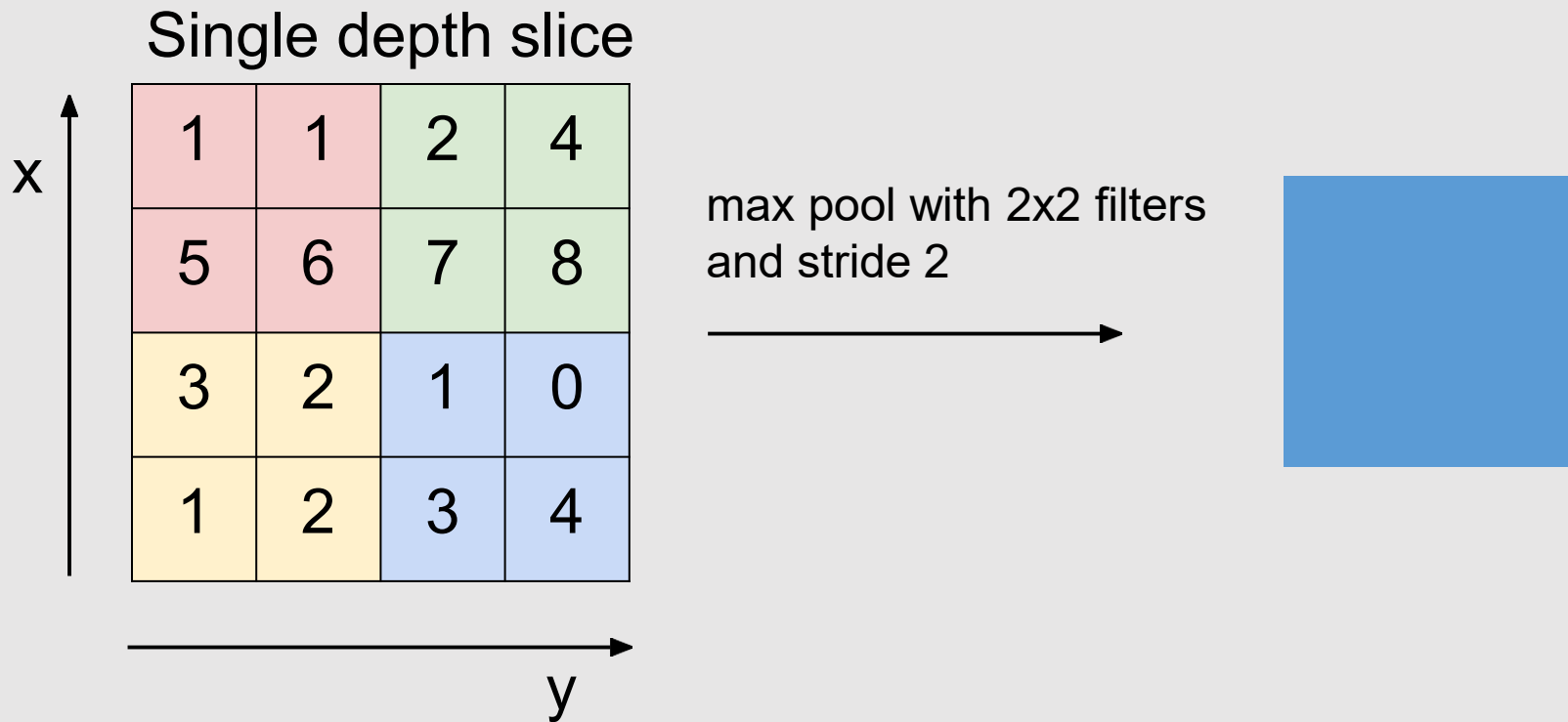
- Input volume: **32x32x3**; **10** **5x5** filters with stride **1**, pad **2**
 - Output volume size?
 - 
 - Number of parameters for this layer?
 - 

Pooling Layer: Downsampling

- Make the representations smaller and more manageable → dimensionality reduction
- Operate over each activation map independently:



Max Pooling



Lab 7 CNN (See notebook for details)

```
__init__(self):
    super(CNN, self).__init__()
    self.conv1 = nn.Conv2d(3, 6, 5) #3: #
    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)

    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
```

- Initial Image Size: $3 \times 32 \times 32$
- After conv1: $6 \times 28 \times 28$ (32)
- After Pooling: $6 \times 14 \times 14$ (ima
- After conv2: $16 \times 10 \times 10$ (14
- After Pooling: $16 \times 5 \times 5$ (halve
- After fc1: 120
- After fc2: 84
- After fc3: 10 (= number of cla



Acknowledgement

- The slides used materials from:
Matt Gormley, Eric Xing, Nina Balcan, Fei-Fei Li & Justin Johnson & Serena Yeung, Lisa Zhang, Michael Guerzhoy, Svetlana Lazebnik, Ismini Lourentzou, Dekai Wu

A large orange shape, resembling a quarter-circle or a thick arc, is positioned on the left side of the slide.

Recommended Reading

- [CS231n: Convolutional Neural Networks for Visual Recognition from Stanford](#) (Fei-Fei Li et al.)
- [The Deep Learning Book with a free official **html** version provided by the authors](#) (Ian Goodfellow et al.)
- [Convolution arithmetic](#)
- PyTorch documentations
- The lab notebook and references

Next



Lab notebooks



Feedback (if any)
@end of the week