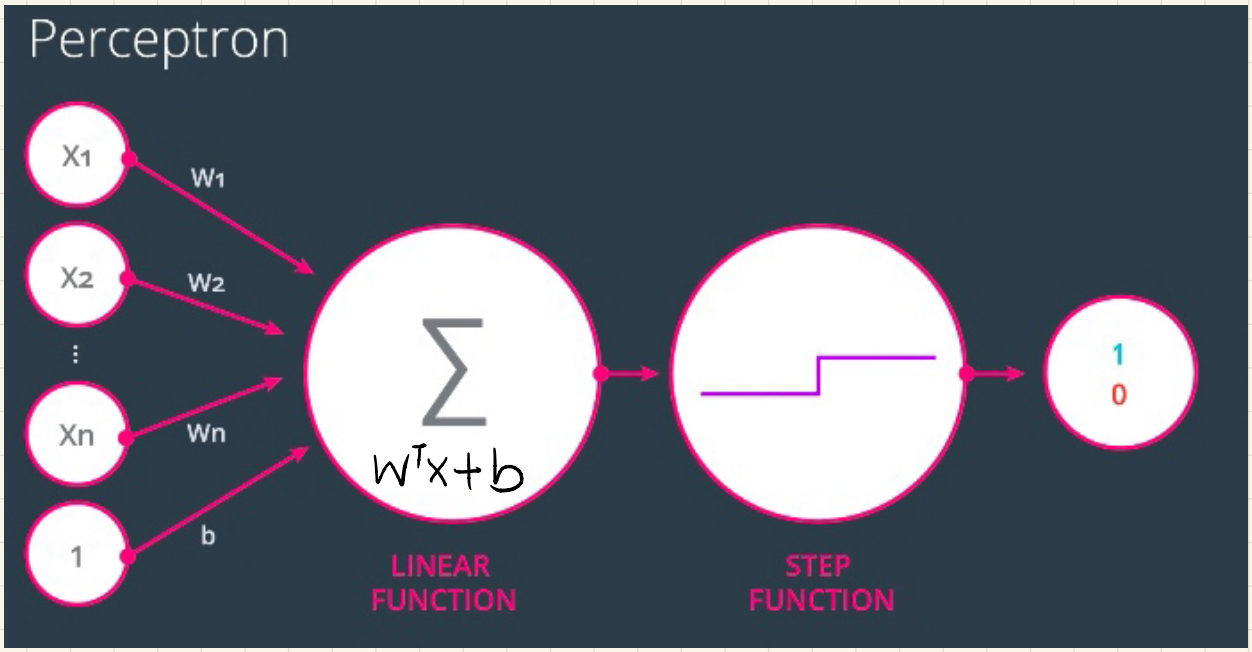
Neural Networks

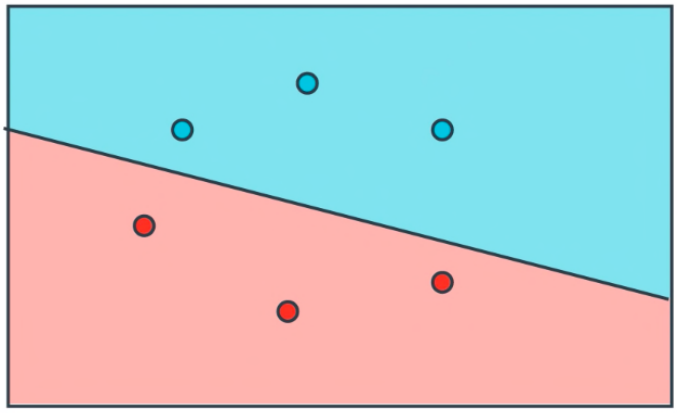
Lesson 2 Concepts

- Preceptron
- Neural Network
- Error Function
- One-hot coding
- Cross entropy
- Maximum likelihood
- Gradient Descent
- Feed forward
- Back Propagation
- Model Complexity Graph
- Addressing Overfitting
  - Dropout
  - Regularization
- Local Minima & Random Initiation
- Vanishing gradients
- Batch normalization v.s. Stochastic GD
- Learning Rate Decay
- Momentum

Preceptron



Perceptron Algorithm



1. Start with random weights: $w_1, \ldots, w_n, b$

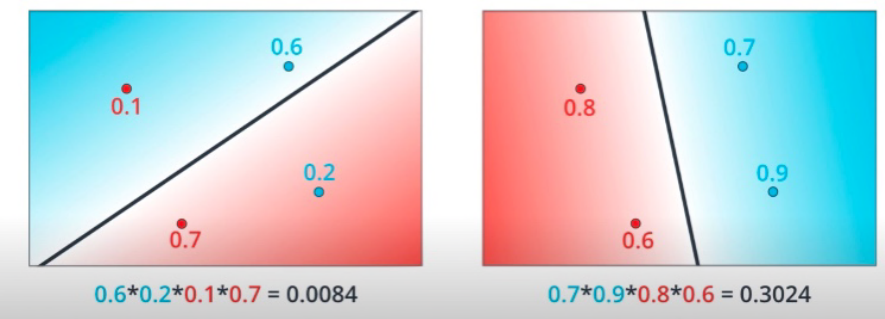
2. For every misclassified point $(x_1,\ldots,x_n)$:

   2.1. If prediction = 0:
   - For i = 1 ...n
     - Change $w_i + \alpha x_i$
   - Change b to b + α

   2.2. If prediction = 1:
   - For i = 1 ...n
     - Change $w_i - \alpha x_i$
   - Change b to b - α

Softmax

- One-Hot Encoding
- Maximum Likelihood



X Don't use product

- Think of $\hat{y}$ as a probability
  $\hat{y} = \sigma(Wx+b) \equiv P(\text{blue})$
  - Probabilities of the data points being correctly labeled.

Instead, Use natural log & Cross Entropy

$0.6 * 0.2 * 0.1 * 0.7 = 0.0084$

$\ln(0.6) + \ln(0.2) + \ln(0.1) + \ln(0.7)$

$-0.51 \quad -1.61 \quad -2.3 \quad -0.36$

$0.7 * 0.9 * 0.8 * 0.6 = 0.3024$

$\ln(0.7) + \ln(0.9) + \ln(0.8) + \ln(0.6)$

$-0.36 \quad -0.1 \quad -.22 \quad -0.51$

Use $-\ln$ to get positive #s

$-\ln(0.6) - \ln(0.2) - \ln(0.1) - \ln(0.7) = 4.8$

$0.51 \quad 1.61 \quad 2.3 \quad 0.36$

$-\ln(0.7) - \ln(0.9) - \ln(0.8) - \ln(0.6) = 1.2$

$0.36 \quad 0.1 \quad .22 \quad 0.51$

Cross Entropy

Smaller sum = better model

- Misclassified points have larger values
- Correct labeled points have prob close to 1
  $\ln(1) = 0$

Events ——— Probabilities

Cross Entropy

Multi-class Cross Entropy

Multi-Class Cross-Entropy

ANIMAL	DOOR 1	DOOR 2	DOOR 3
	p_{11}	p_{12}	p_{13}
	p_{21}	p_{22}	p_{23}
	p_{31}	p_{32}	p_{33}

$y_{1j} = 1$ if  behind 

$y_{2j} = 1$ if  behind 

$y_{3j} = 1$ if  behind 

$$\text{Cross-Entropy} = -\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\ \ln(p_{ij})$$

Binary

$$\text{Error Function} = -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)(\ln(1-\hat{y}_i)) + y_i\ln(\hat{y}_i)$$

$$E(W,b) = -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)(\ln(1-\sigma(Wx^{(i)}+b)) + y_i\ln(\sigma(Wx^{(i)}+b)$$

Multi

ERROR FUNCTION:

$$-\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} y_{ij}\ln(\hat{y}_{ij})$$

Gradient Descent

$$\hat{y} = \sigma(w_1 x_1 + \cdots + w_n x_n + b)$$

$$J = -\frac{1}{m}\sum_{1}^{m}\sum_{1}^{n} y_{ij} \ln(\hat{y}_{ij})$$

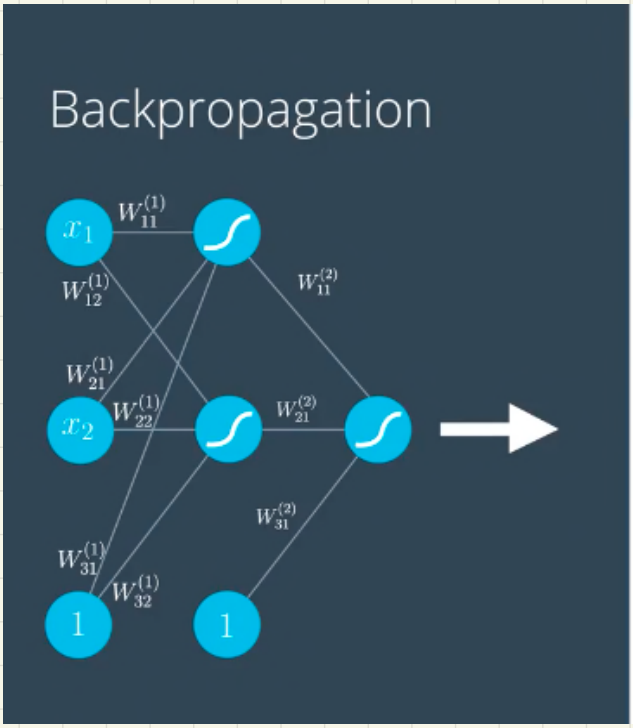
$$\nabla J = \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \frac{\partial J}{\partial b}\right)$$

$$\alpha = 0.1$$

$$w_i' = w_i - \alpha \frac{\partial J}{\partial w_i}$$

$$b' = b - \alpha \frac{\partial J}{\partial b}$$

$$\hat{y} = \sigma(w'x + b) \longrightarrow \text{Minimized}$$

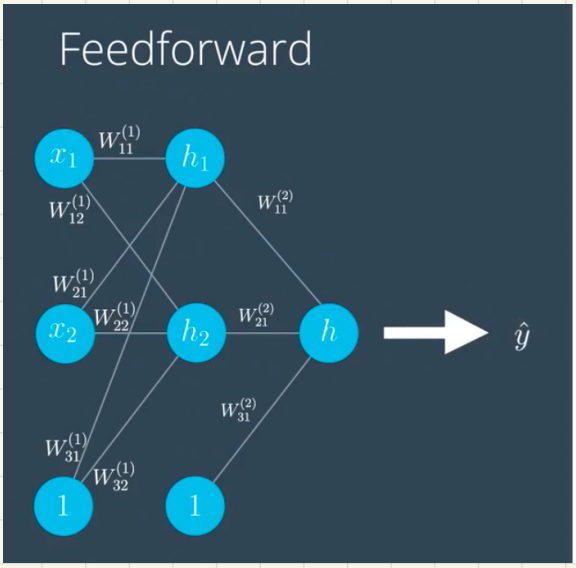


$$\hat{y} = \sigma W^{(2)} \circ \sigma \circ W^{(1)}(x)$$

$$W^{(1)} = \begin{pmatrix} W_{11}^{(1)} & W_{12}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} \\ W_{31}^{(1)} & W_{32}^{(1)} \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} W_{11}^{(2)} \\ W_{21}^{(2)} \\ W_{31}^{(2)} \end{pmatrix}$$

$$\nabla E = \begin{pmatrix} \frac{\partial E}{\partial W_{11}^{(1)}} & \frac{\partial E}{\partial W_{12}^{(1)}} & \frac{\partial E}{\partial W_{11}^{(2)}} \\ \frac{\partial E}{\partial W_{21}^{(1)}} & \frac{\partial E}{\partial W_{22}^{(1)}} & \frac{\partial E}{\partial W_{21}^{(2)}} \\ \frac{\partial E}{\partial W_{31}^{(1)}} & \frac{\partial E}{\partial W_{32}^{(1)}} & \frac{\partial E}{\partial W_{31}^{(2)}} \end{pmatrix}$$

$$W_{ij}^{\prime(k)} \leftarrow W_{ij}^{(k)} - \alpha \frac{\partial E}{\partial W_{ij}^{(k)}}$$

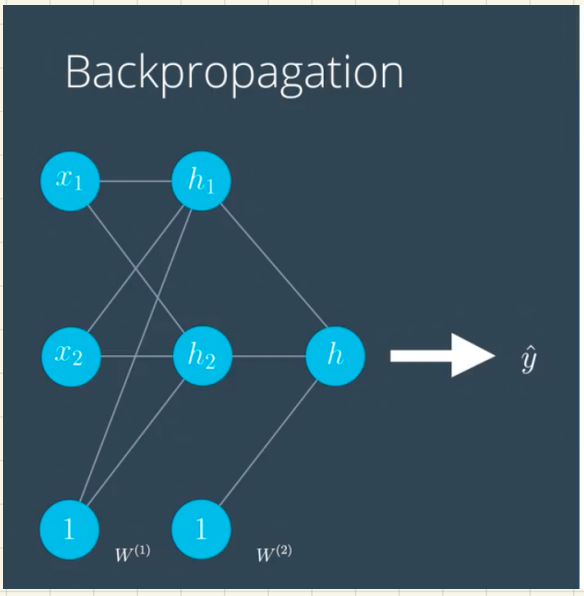


$$h_1 = W_{11}^{(1)}x_1 + W_{21}^{(1)}x_2 + W_{31}^{(1)}$$

$$h_2 = W_{12}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{32}^{(1)}$$

$$h = W_{11}^{(2)}\sigma(h_1) + W_{21}^{(2)}\sigma(h_2) + W_{31}^{(2)}$$

$$\hat{y} = \sigma(h)$$



$$E(W) = -\frac{1}{m}\sum_{i=1}^{m} y_i ln(\hat{y}_i) + (1-y_i) ln(1-\hat{y}_i)$$

$$E(W) = E(W_{11}^{(1)}, W_{12}^{(1)}, ..., W_{31}^{(2)})$$

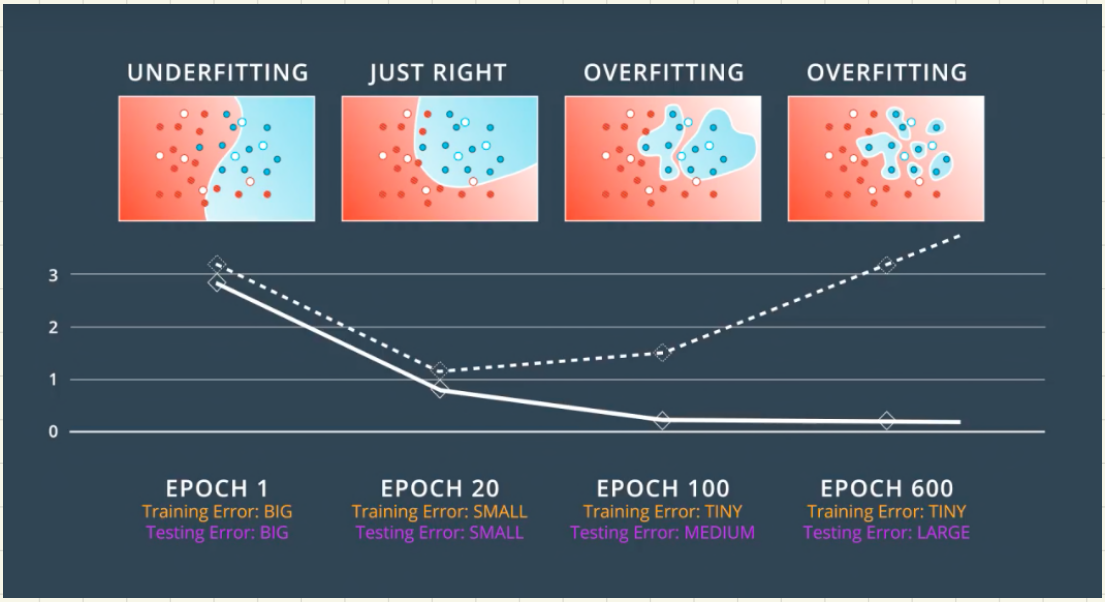
$$\nabla E = (\frac{\partial E}{\partial W_{11}^{(1)}}, ..., \frac{\partial E}{\partial W_{31}^{(2)}})$$

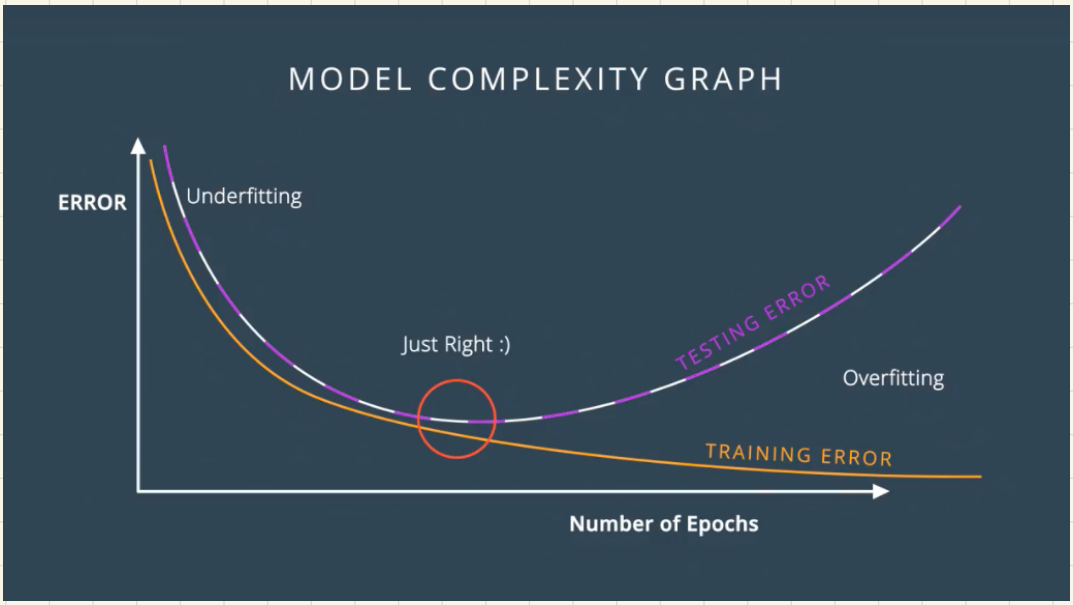
$$\frac{\partial E}{\partial W_{11}^{(1)}} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h} \frac{\partial h}{\partial h_1} \frac{\partial h_1}{\partial W_{11}^{(1)}}$$

Chain Rule

Evaluate the change of weights of the first layer, how much that's going to change J.

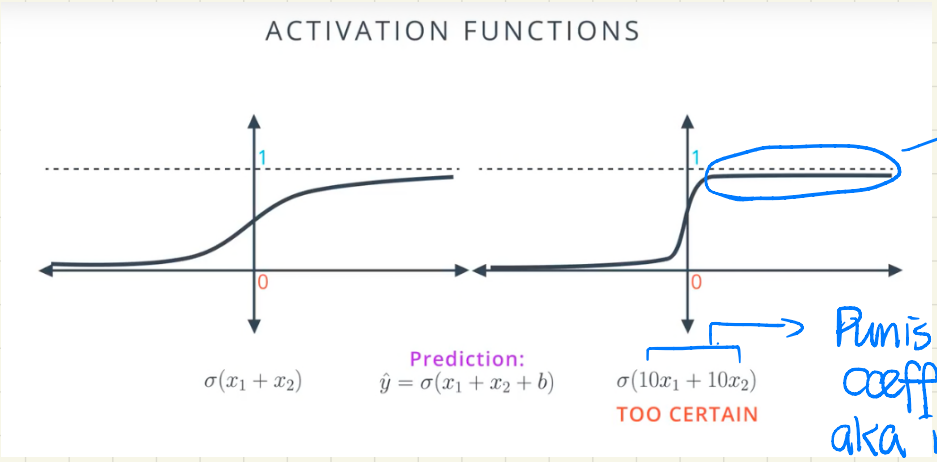
Early Stopping





GD till testing error stops decreasing & just before it starts to increase.

Regularization



$\frac{\partial J}{\partial w_i}$ is basically 0 plateau, hard to tune the parameters.

Punish high coefficients aka weights.

LARGE COEFFICIENTS ⟶ OVERFITTING

PENALIZE LARGE WEIGHTS

$(w_1, \ldots, w_n)$

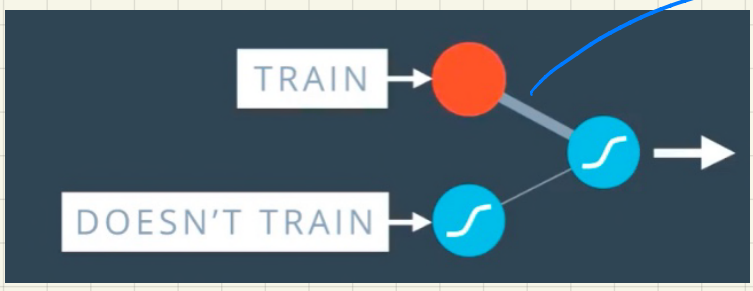
L1 ERROR FUNCTION $= -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i) + y_i ln(\hat{y}_i) + \boxed{\lambda(|w_1| + \ldots + |w_n|)}$

L2 ERROR FUNCTION $= -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i) + y_i ln(\hat{y}_i) + \boxed{\lambda(w_1^{\,2} + \ldots + w_n^{\,2})}$

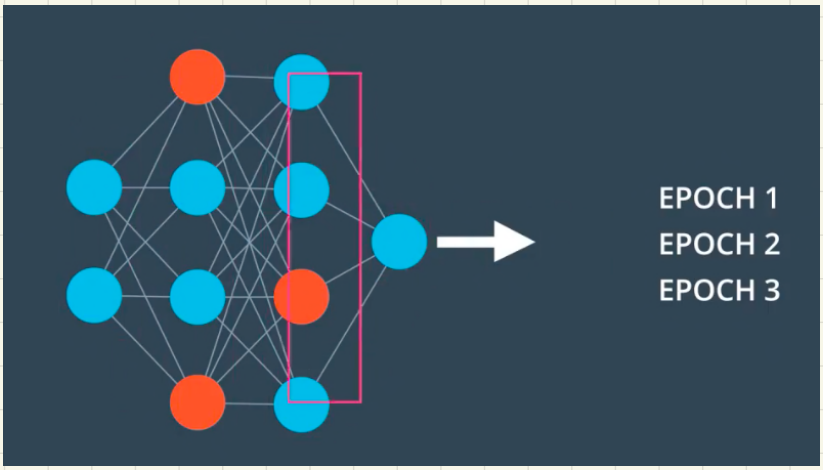
L1: sparsity (1,0,0,1,0)
For feature selection

L2: Sparsity (0.5, 0.3, 0.4 ...)
For training models.

Dropout.



→ Some have large wts. dominate the training process.



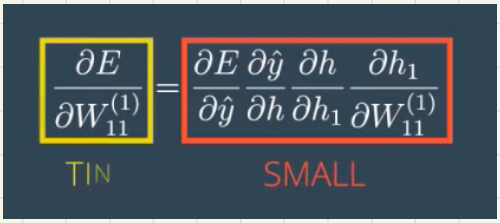
Design a parameter

||

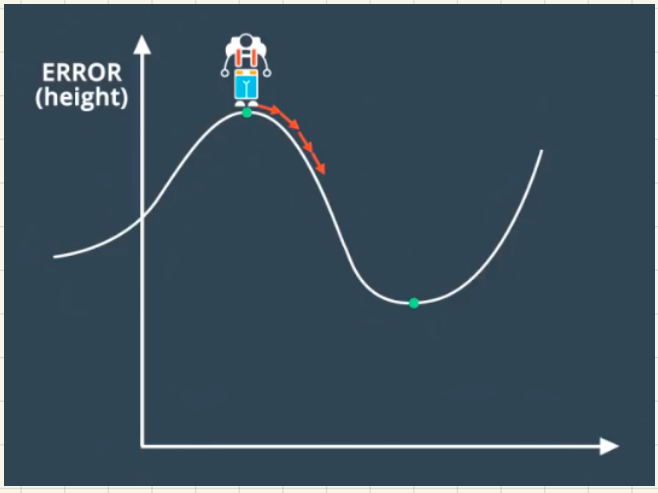
A probability

(of each node being dropped randomly)

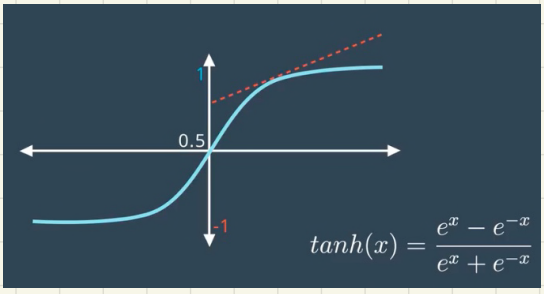
Vanishing Gradients.

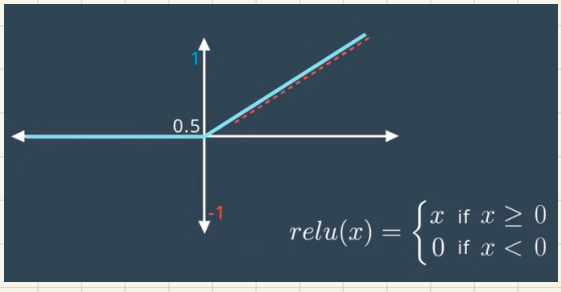


flat activation function curve



Use other activation functions.

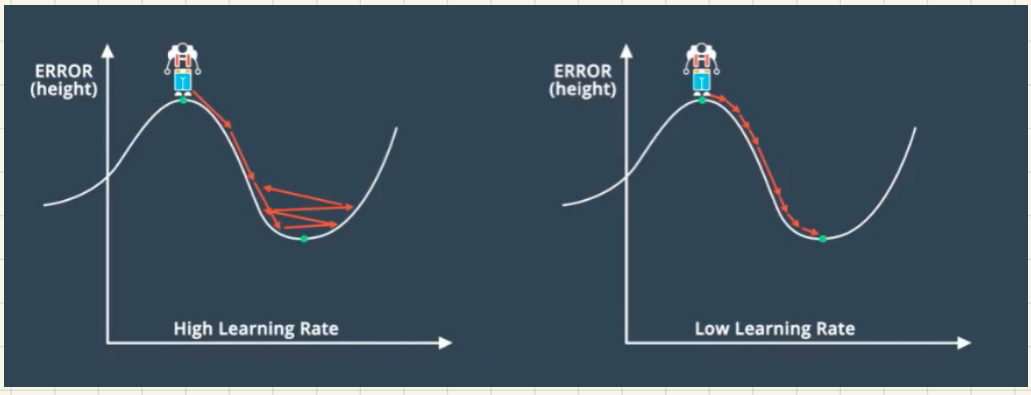




Batch GD:

- Go through a small set of data to get a general idea of what ∇J is like.

Learn Rate Decay

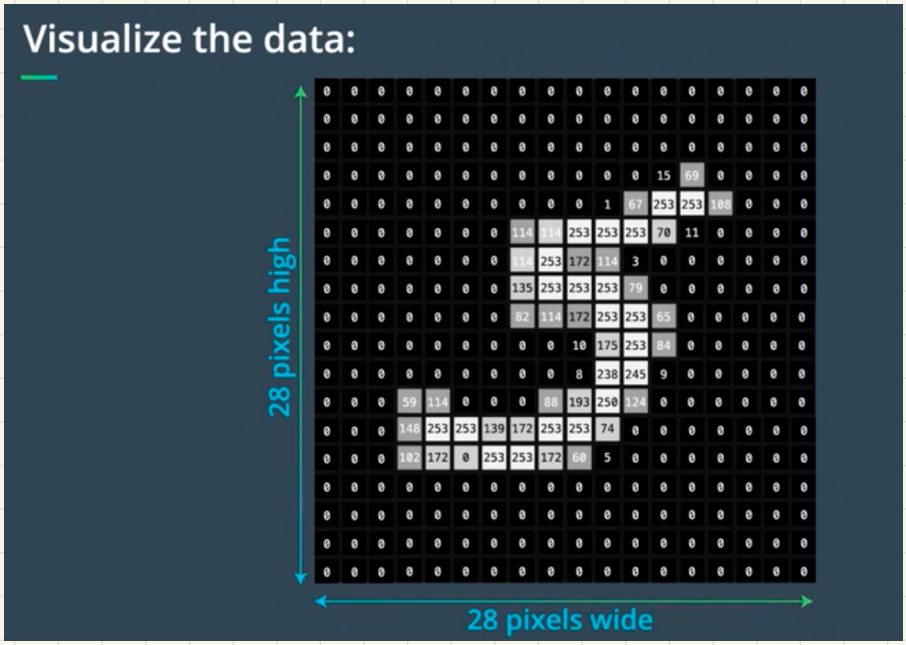


Lesson 5 Concepts

– Convolutional Neural Network –

1. Feature/Pixel → Normalize → Flattening

MLP!

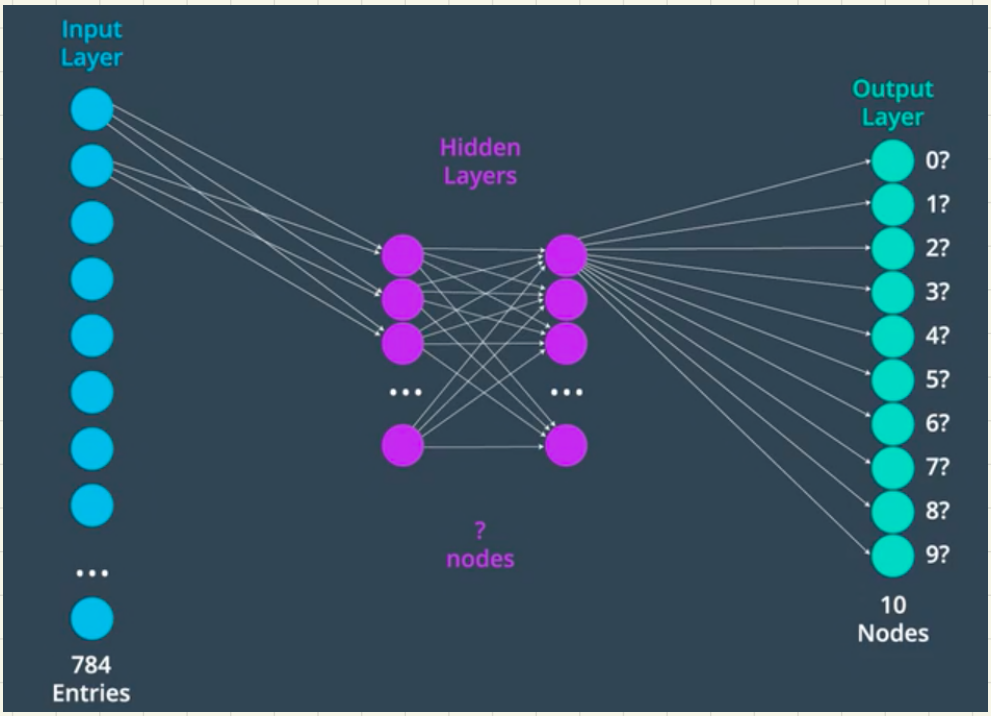


White → 255

Black → 0

Need to normalize

Makes gradient consistent, not so large such that training is too slow



Class Scores

Do research to see how to design the architecture.

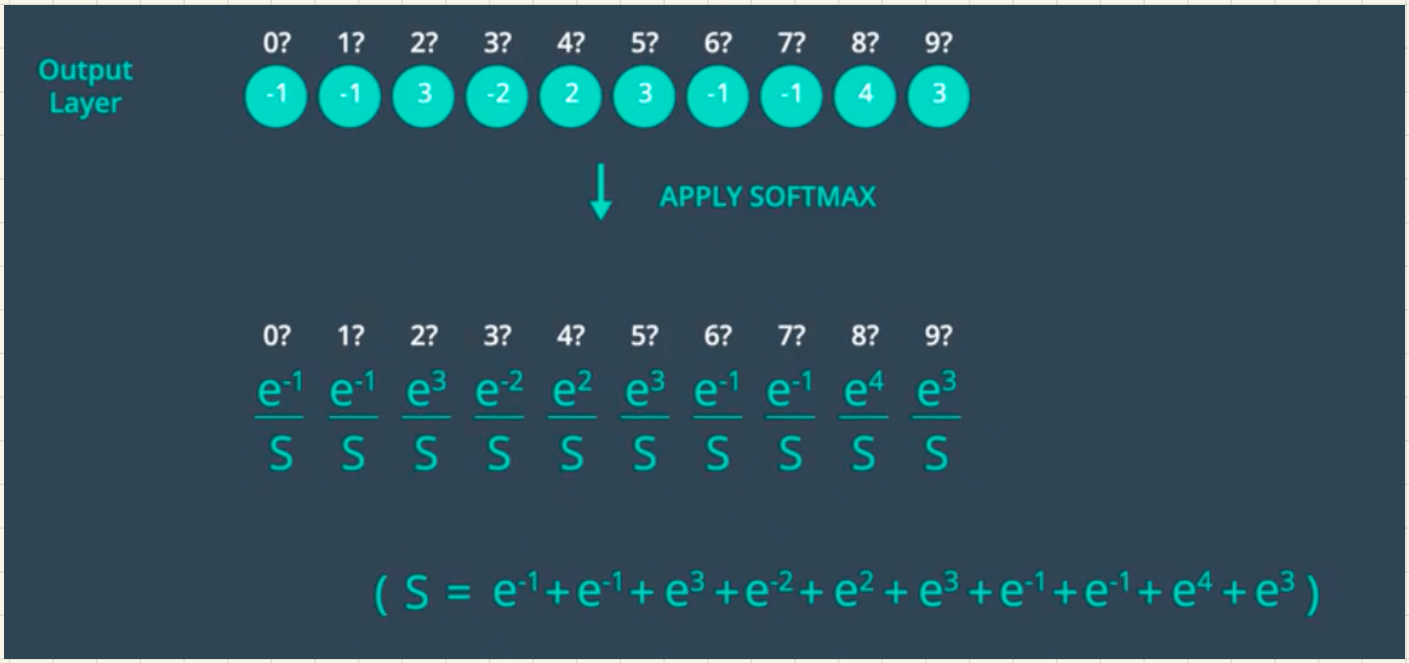
Note

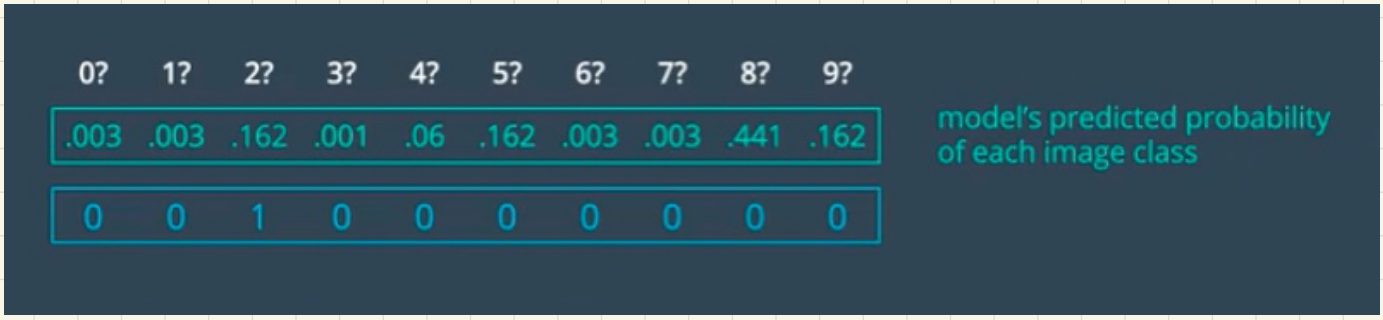
- ↑ hidden layers . ↑ complexity . ↑ over-fitting

Learn From Mistakes

1. Loss function (Error = ?) Categorical C-E L
2. Back Propagation (How bad a particular weight is in making mistakes?)
3. Optimization (GD, Adam··· give better w)

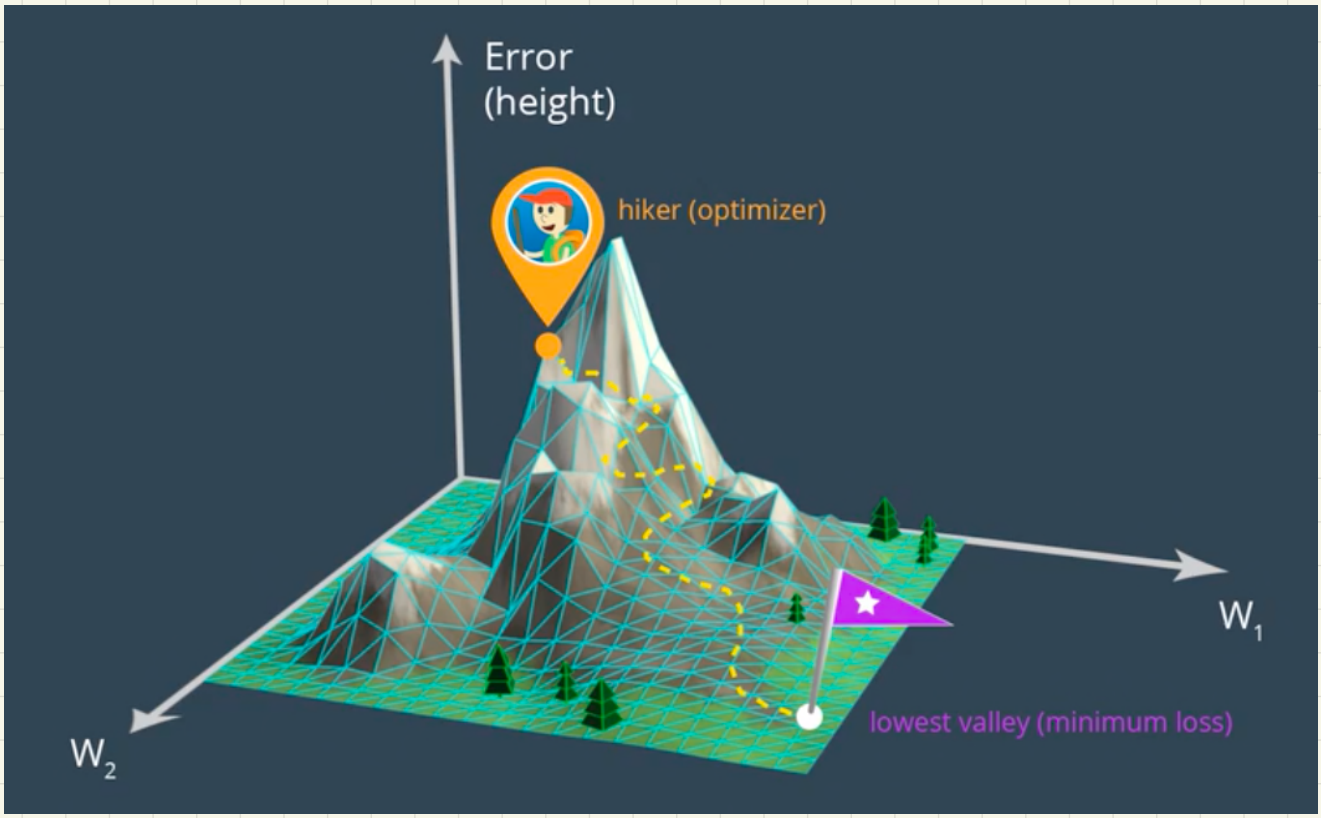
↓ training

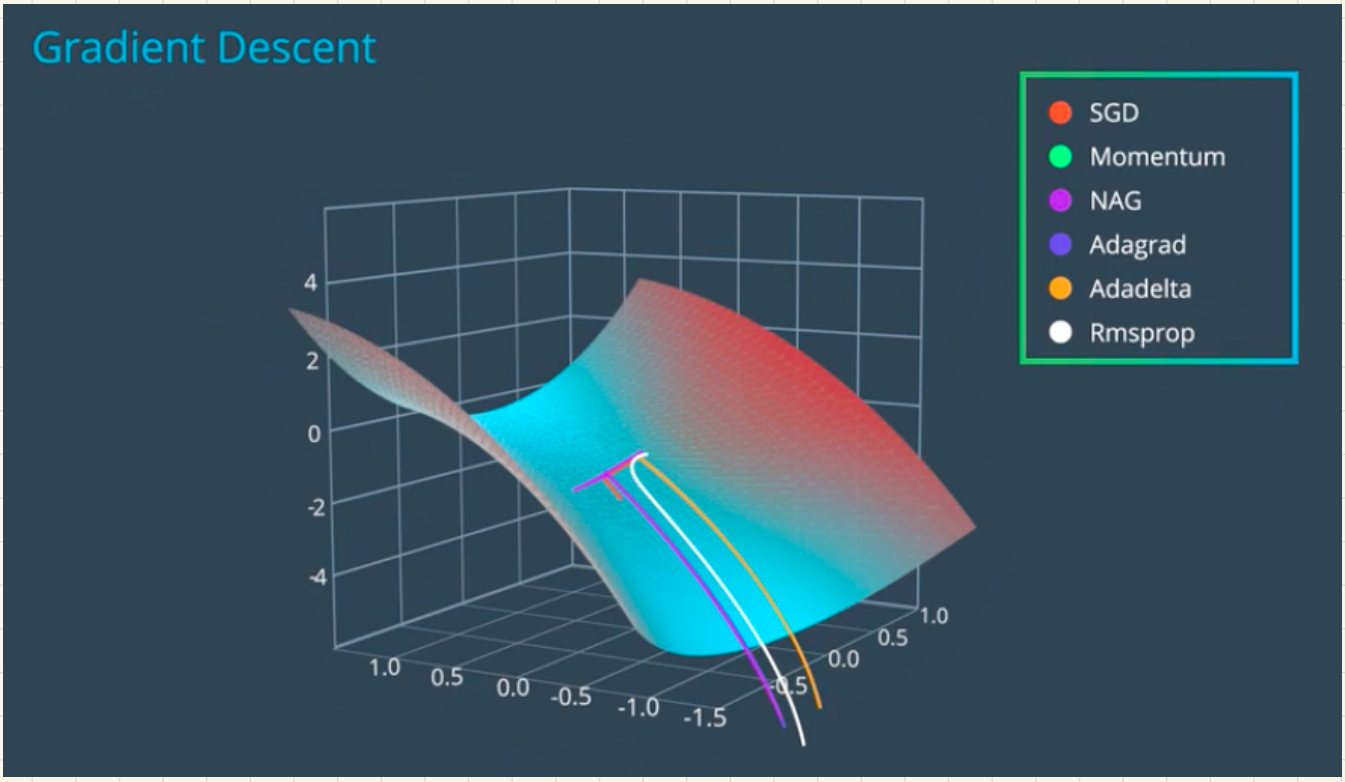




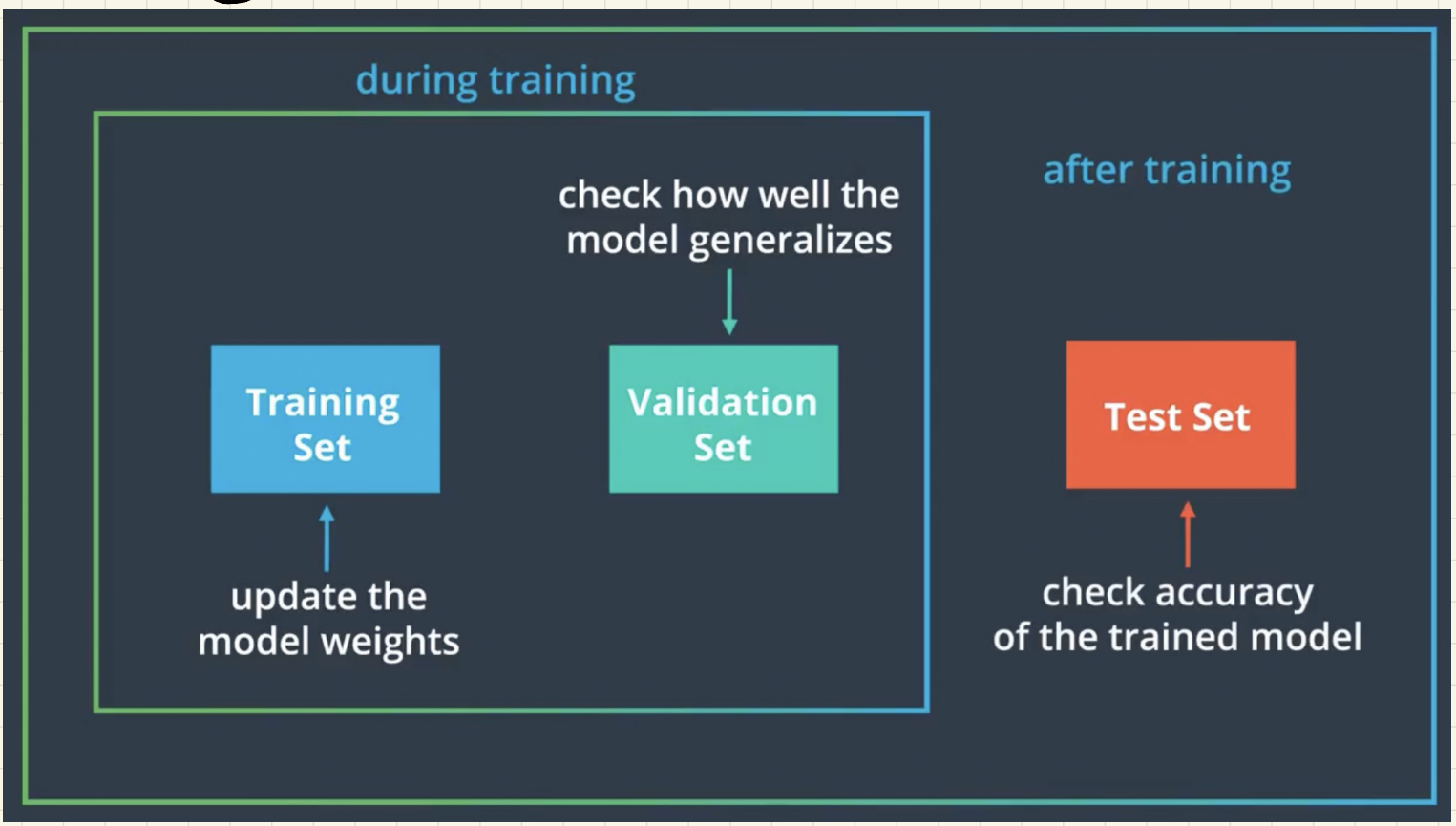
Recall:

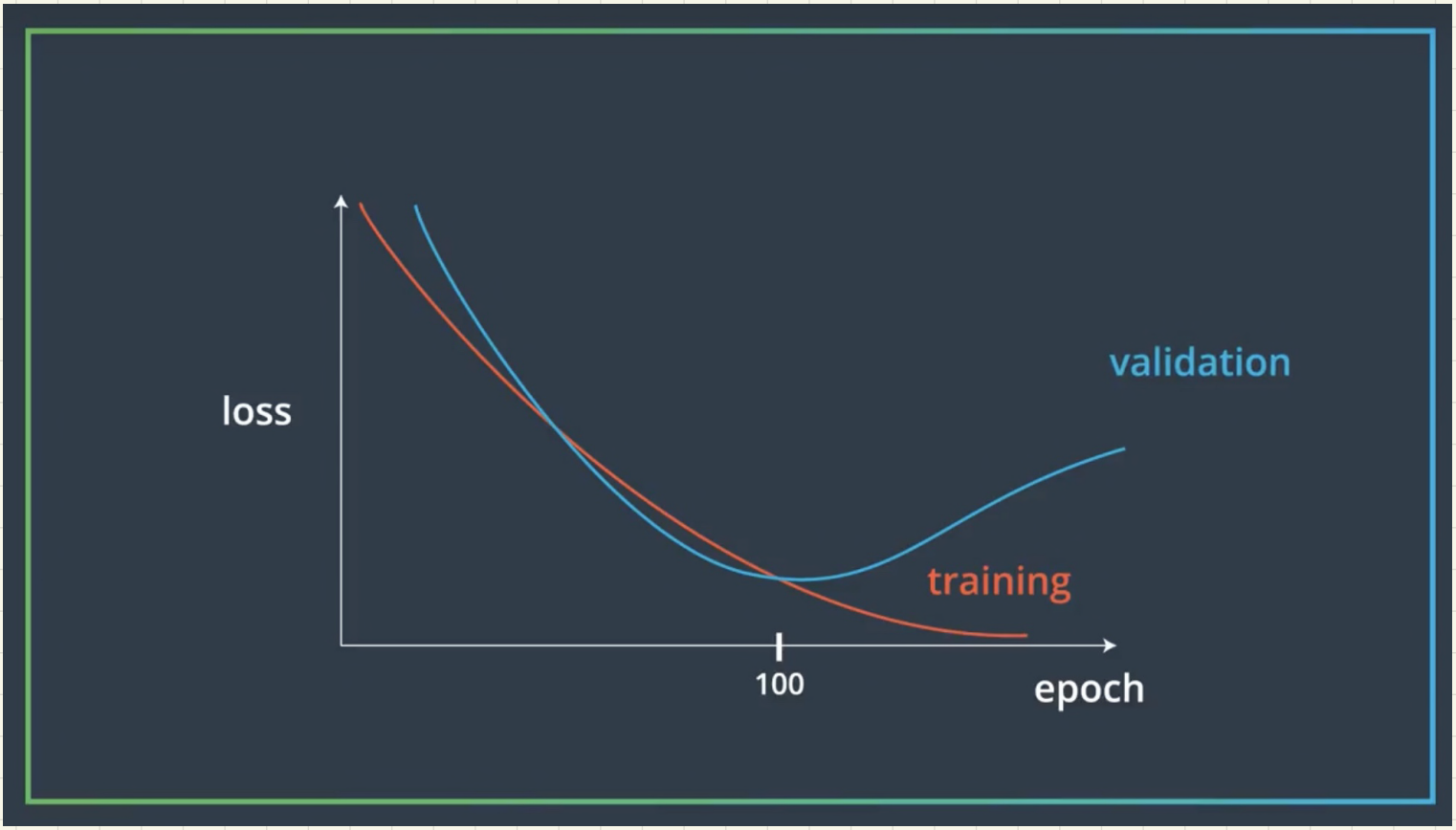
- CCEL = -log(Prob.)
- The lower, the more accurate


Error
(height)
hiker (optimizer)
W1
W2
lowest valley (minimum loss)


Gradient Descent
SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
-2
-4
1.0
0.5
0.0
-0.5
-1.0
-1.5
1.0
0.5
0.0
-0.5

Training Validation






BEST MODEL!
Model 1
val_loss = 0.9
Model 2
val_loss = 0.5
Model 3
val_loss = 1.8

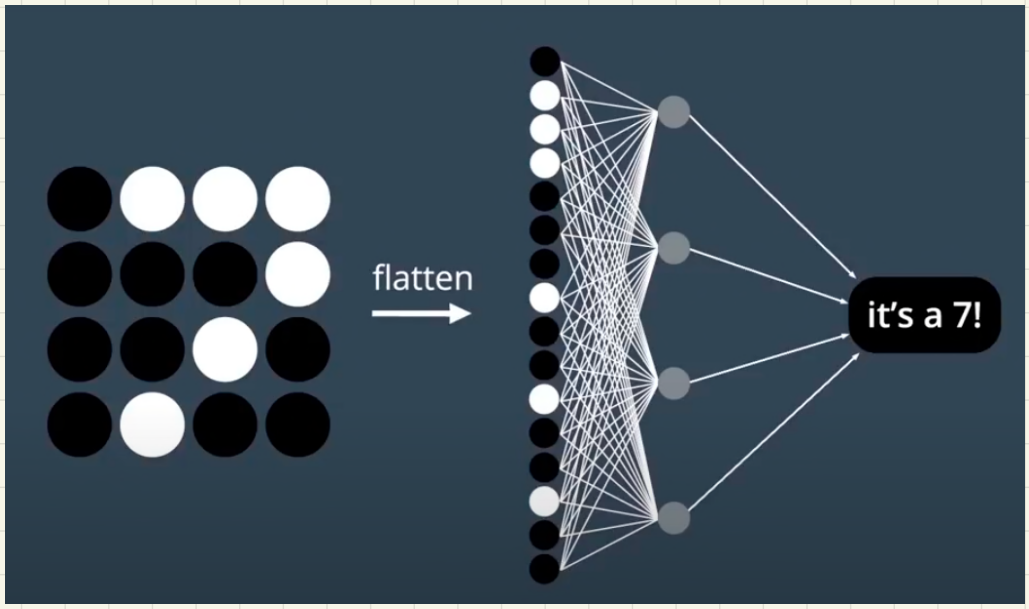
CNN

CNN vs. MLP

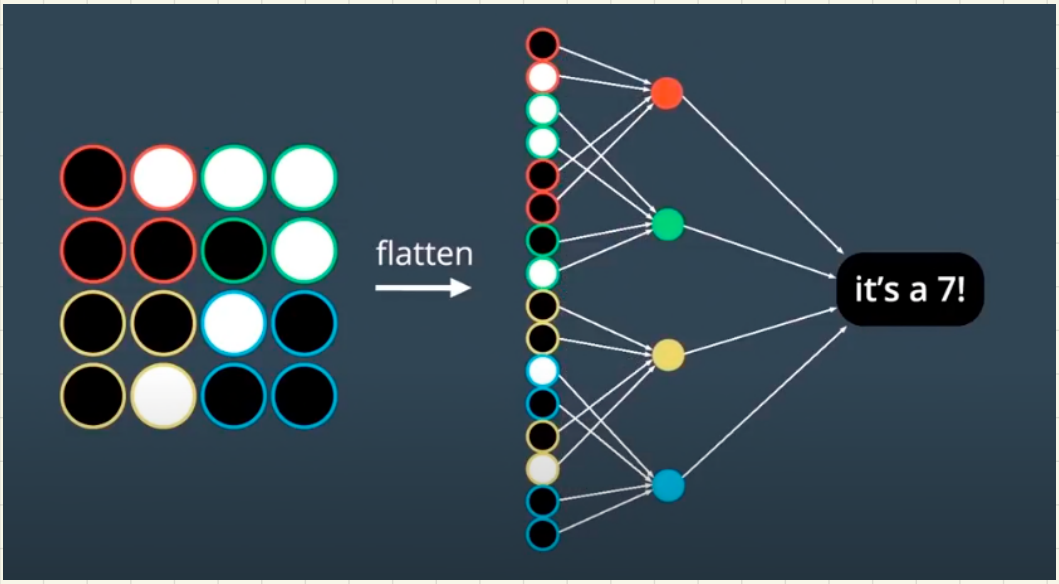
- Only use fully connected layer [Parameters ↑]
- Disgard 2D info when flattening images into vector

Local Connectivity

- CNN uses sparsely connected layer
- Accept matrices as input



In MLP, every hidden node is responsible for gaining the understanding of an entire image all at once.



- Less prone to overfitting
- Truly understand the pattern in the image data

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

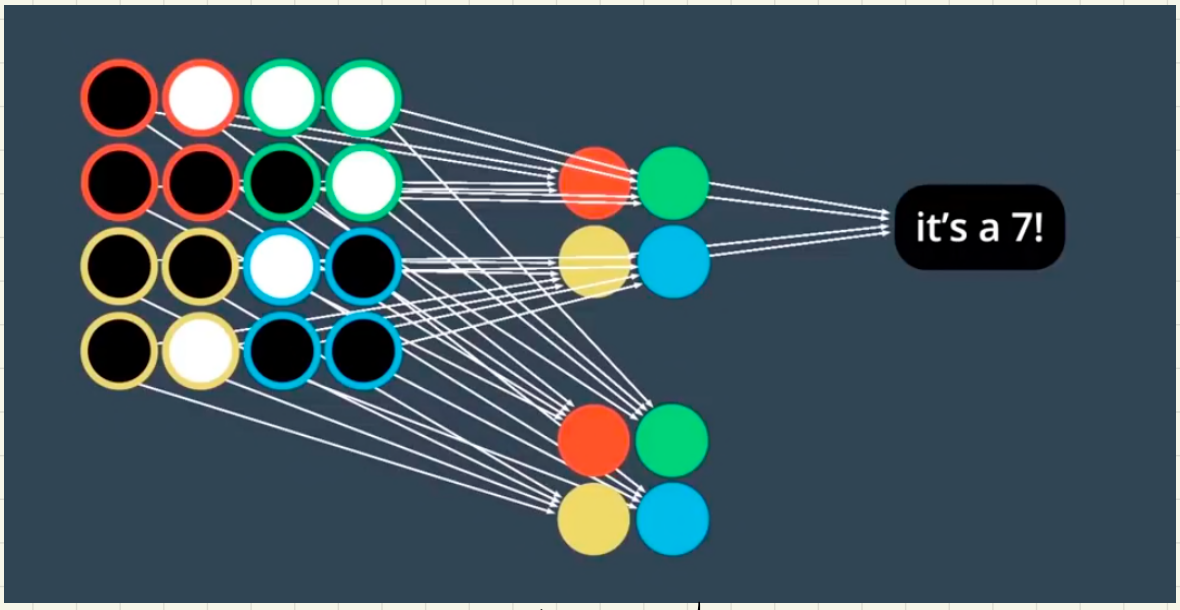
*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.



2 collection of hidden nodes

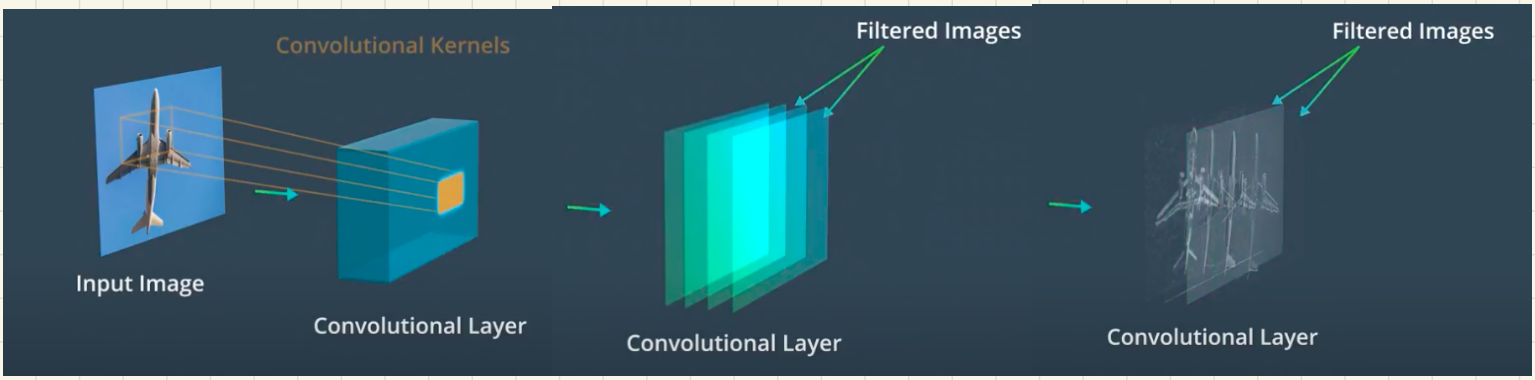
Each collection responsible for a diff. region of img.

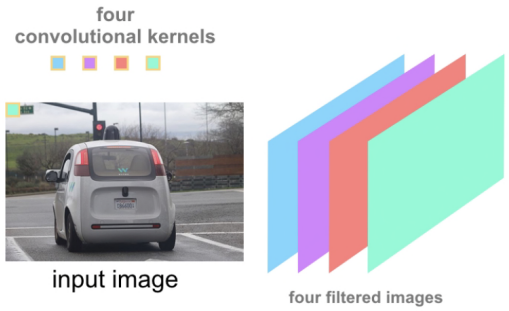
Each collection of diff color share common weights

For instance:

Doesn't matter where the cat is. Parameter Sharing makes it possible to detect patterns at diff. locations.

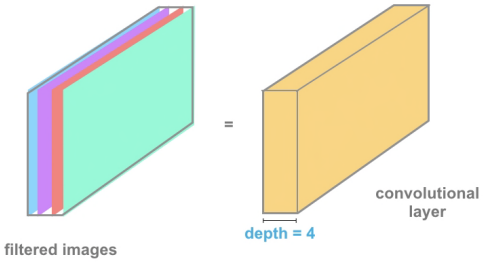
Filters & CNN Layers

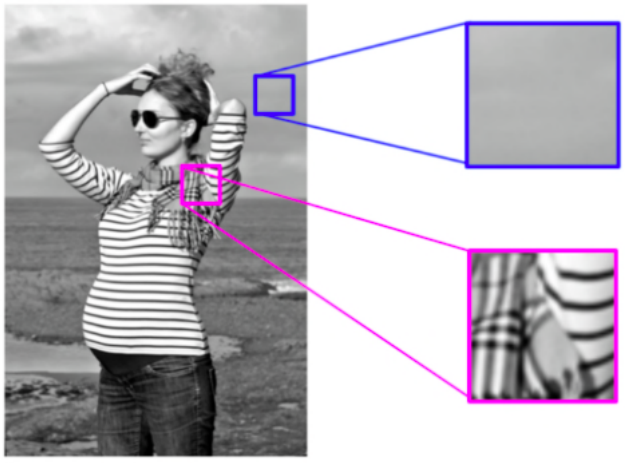




4 kernels = 4 filtered images.

In the example shown, 4 different filters produce 4 differently filtered output images. When we stack these images, we form a complete convolutional layer with a depth of 4!



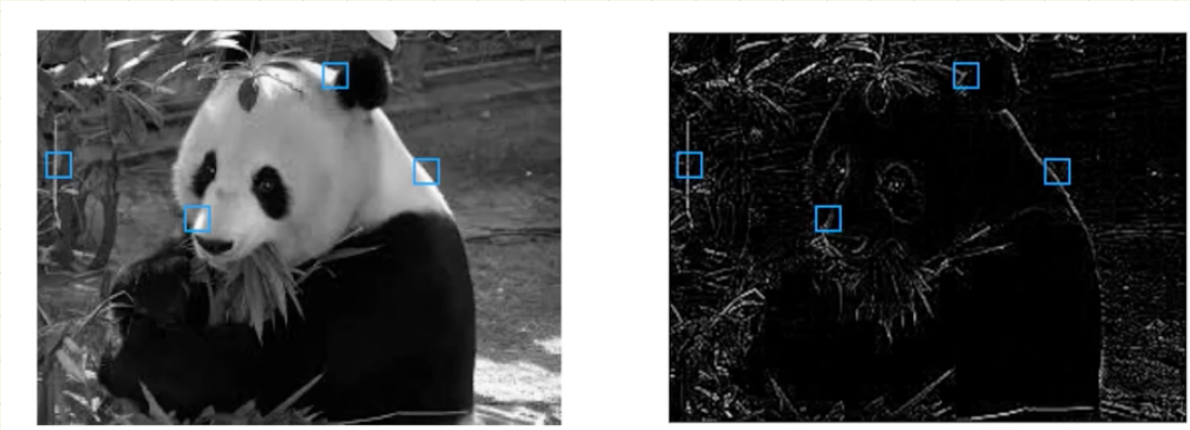


High and low frequency image patterns.

low freq:
not much change to the intensity

High freq:
Changes

High Pass Filters



High Pass Filters:

- Turn low intensity regions into black
- Draw white lines on high intensity region
- Emphasizing edges (area where intensity changes quickly and indicate object boundary)

CONVOLUTION KERNELS

0	-1	0
-1	4	-1
0	-1	0

edge detection filter

0 + -1 + 0 + -1 + 4 + -1 + 0 + -1 + 0 = **0**

Has to add up to 0 or images will be darken or lighten.

Convolve (*): Applying filters

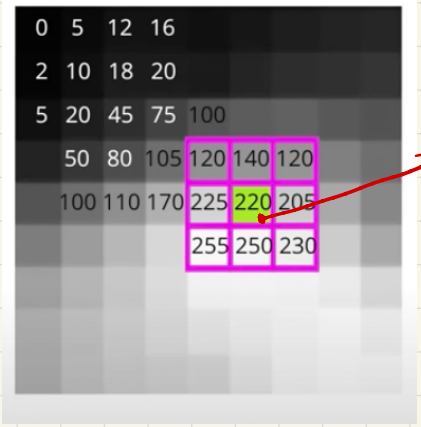
Weights

0	-1	0
-1	4	-1
0	-1	0

*

K F(x,y)

K * F(x,y) = output image



Pixel of interest

Pick a random pixel

0	-140	0
-225	880	-205
0	-250	0

= 60

Not a big value

Doesn't change rapidly

The weighted sum ≡ Pixel value in the output image

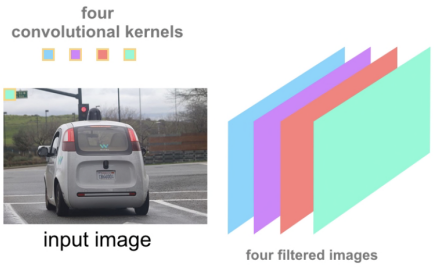
How about corners?

Extend The nearest border pixels are conceptually extended as far as necessary to provide values for the convolution. Corner pixels are extended in 90° wedges. Other edge pixels are extended in lines.

Padding The image is padded with a border of 0's, black pixels.

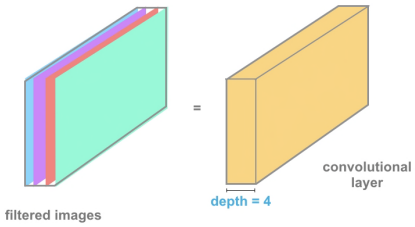
Crop Any pixel in the output image which would require values from beyond the edge is skipped. This method can result in the output image being slightly smaller, with the edges having been cropped.

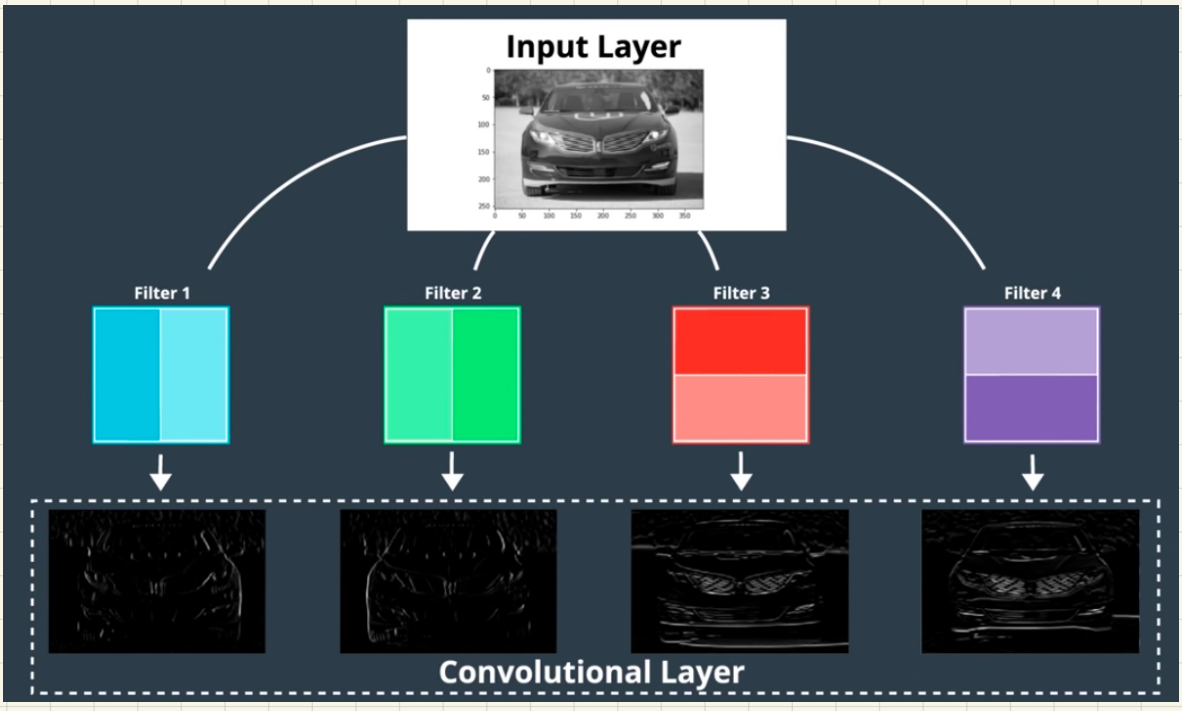
Convolutional Layers



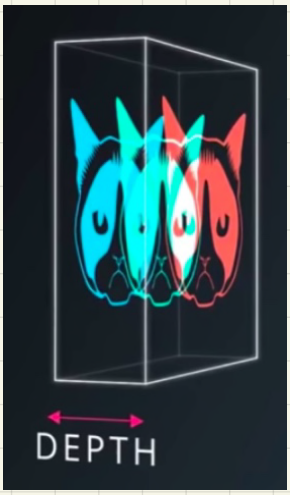
4 kernels = 4 filtered images.

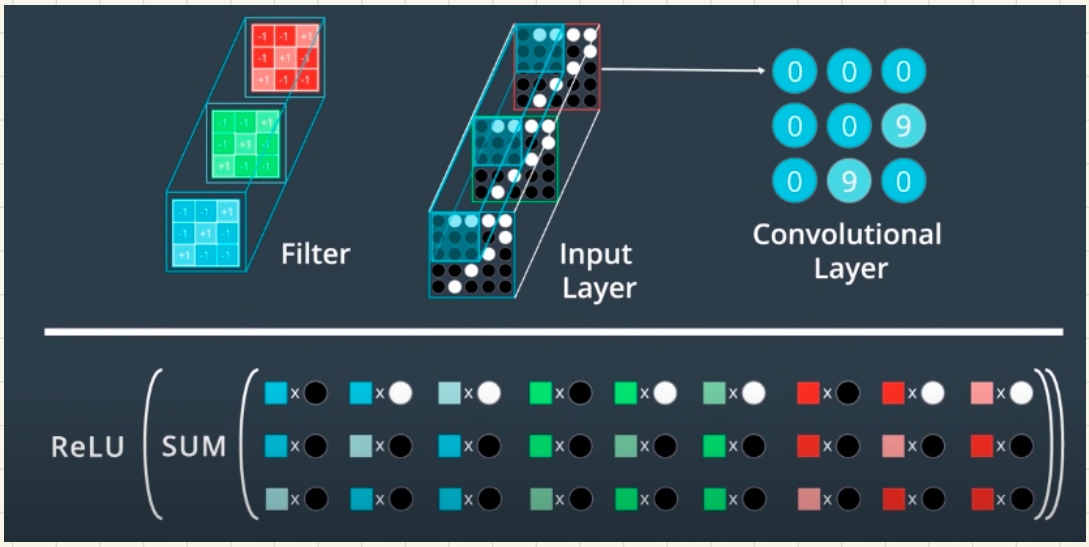
In the example shown, 4 different filters produce 4 differently filtered output images. When we stack these images, we form a complete convolutional layer with a depth of 4!



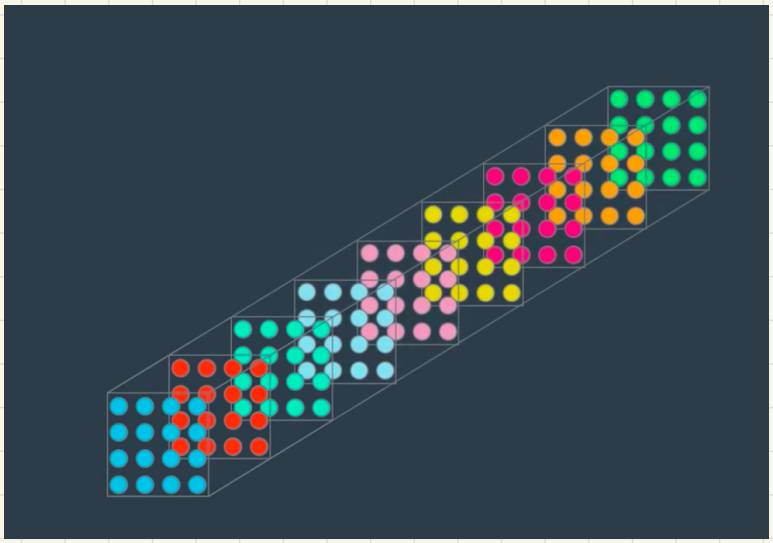


Color Images

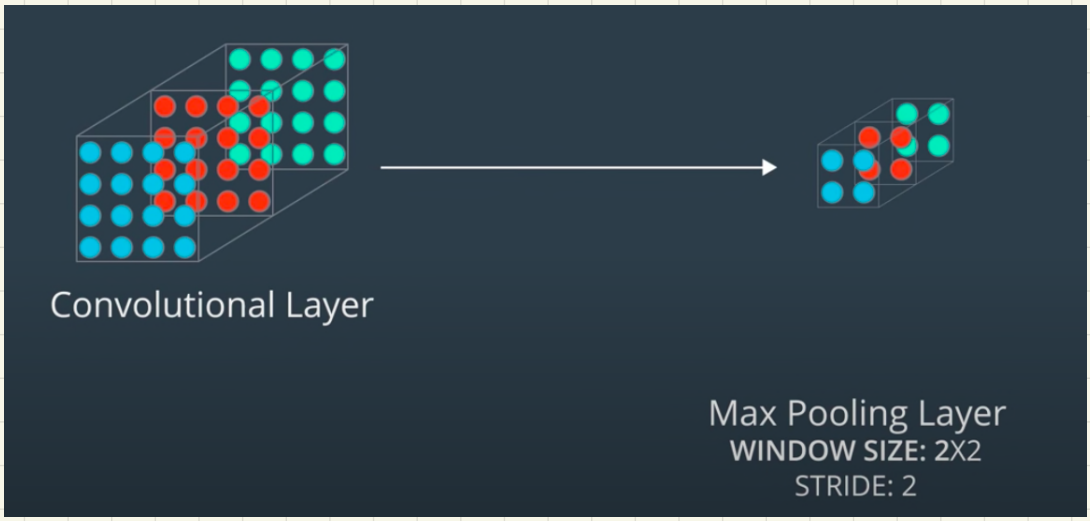




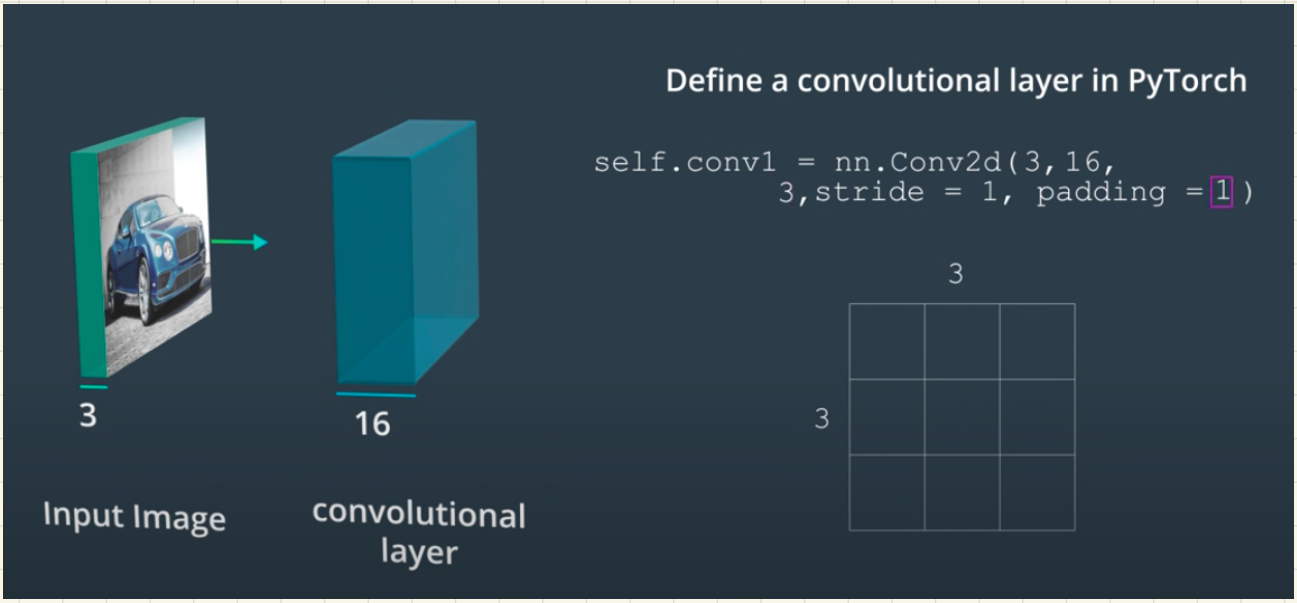
Pooling layer

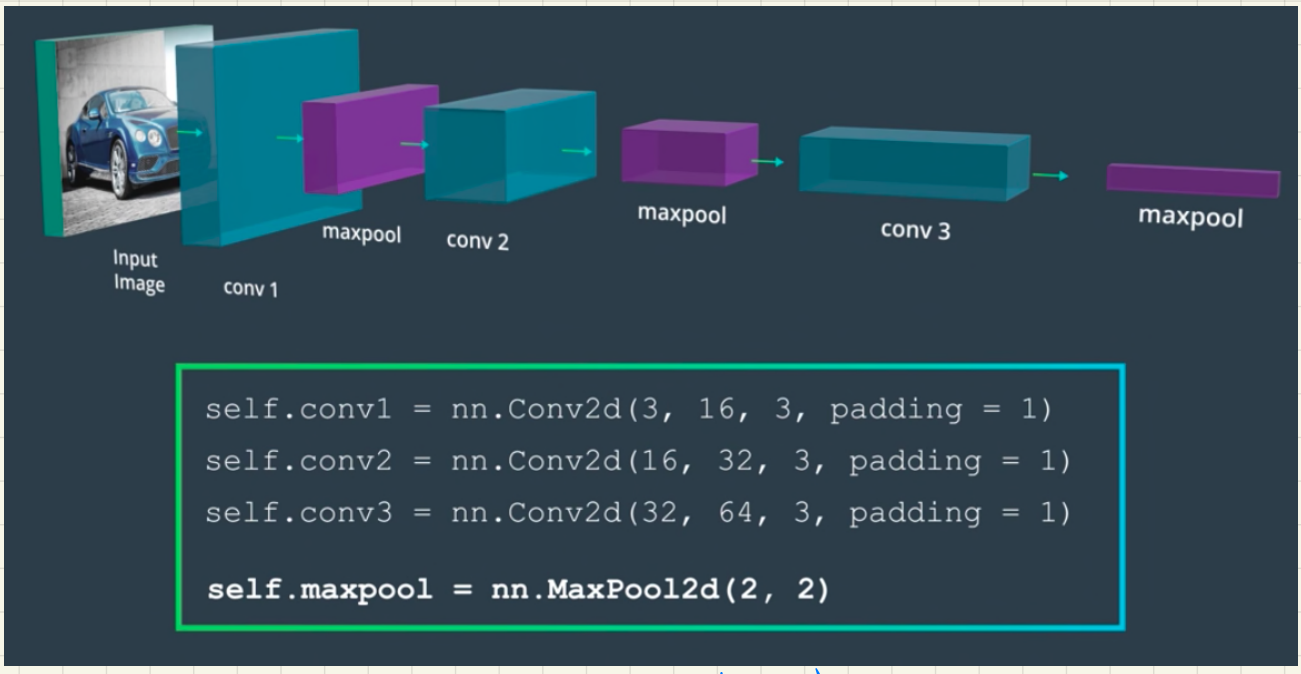


Complicated datasets w/ more objects
=> more filters
=> Dimensions ↑
=> Overfitting



Building a Network





Kernel Size Stride Size → Determines Scaledown factors

Conv layers make the array deep, while maxpooling decrease the height and width.

Deep: more complex, able to detect more features

↓ size: Disgard useless spatial components.