

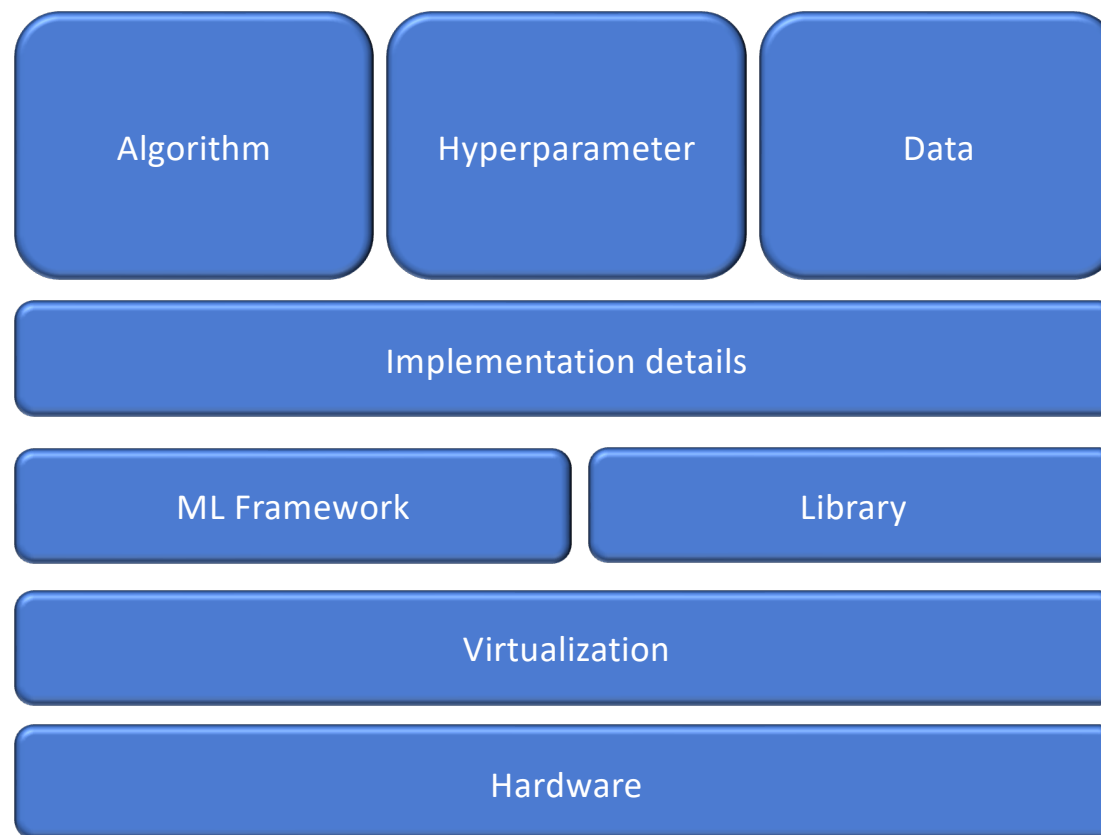
Lecture 5

I-Hsin Chung

Seetharami Seelam

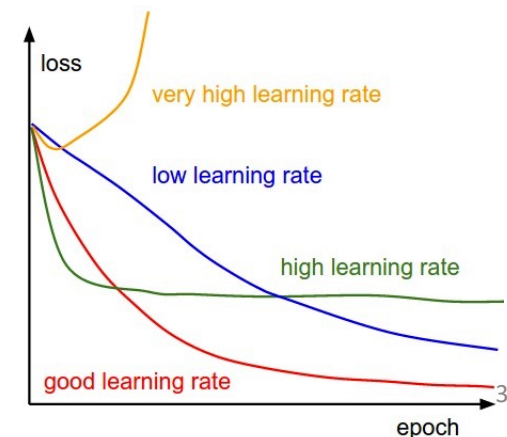
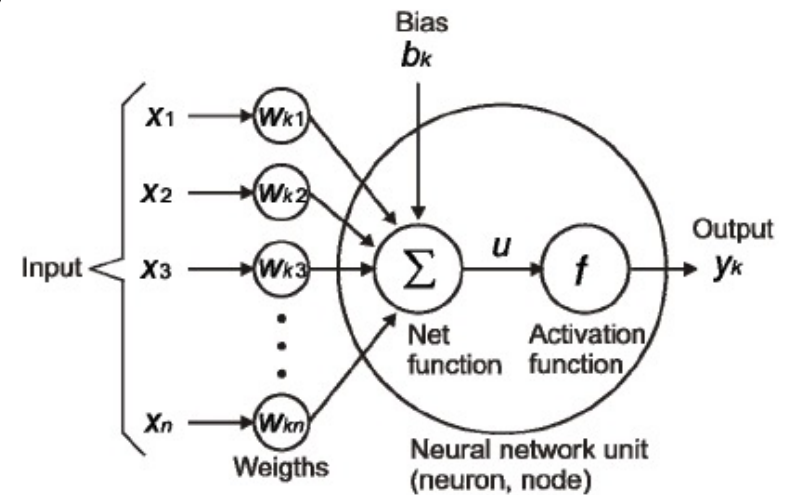
Hao Yu

Performance factors



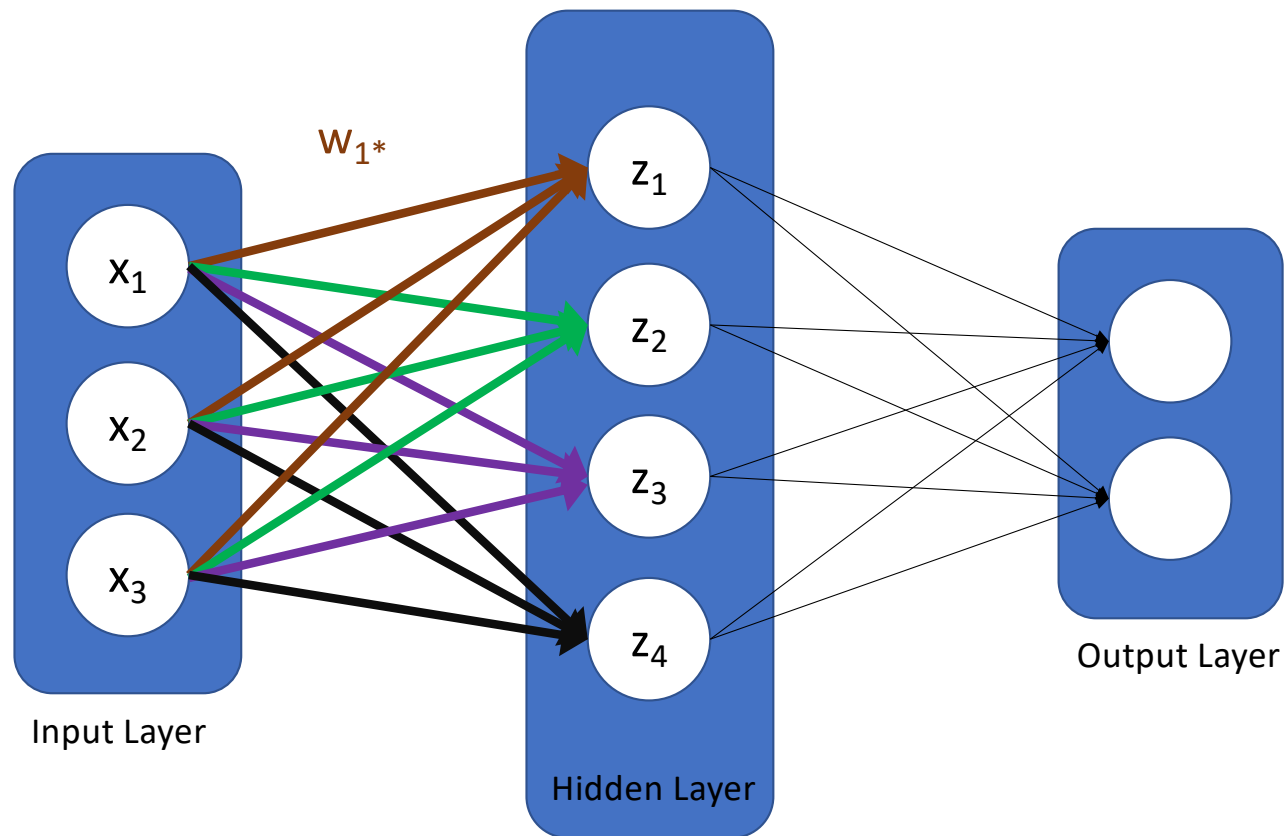
Neural network terminology

- Activation function
 - The activation function translates the input signals to output signals.
- Cost/Loss function
 - Used to measure the accuracy of the NN
- Learning rate
 - the amount of minimization in the cost function in each iteration
 - the rate at which we descend towards the minima of the cost function is the learning rate
 - $w = w - \eta \frac{dL(w)}{dw}$



Neural Network

$$z = b + \sum w_i x_i$$



Forward propagation

- Weighted Sum

$$\begin{array}{ccccccc}
 & j \times i & & i \times \text{batch_size} & & j \times \text{batch_size} & & j \times \text{batch_size} \\
 \begin{bmatrix} - & w_{1*} & - \\ - & \vdots & - \\ - & w_{j*} & - \end{bmatrix} & \begin{bmatrix} | & & | \\ x_{*1} & \dots & x_{*bs} \\ | & & | \end{bmatrix} & + & \begin{bmatrix} b_1 & & b_1 \\ \vdots & \dots & \vdots \\ b_j & & b_j \end{bmatrix} & = & \begin{bmatrix} | & & | \\ z_{*1} & \dots & z_{*bs} \\ | & & | \end{bmatrix} \\
 w_{1*} : 1^{\text{st}} \text{ neuron weights} & x_{*1} : 1^{\text{st}} \text{ data sample} & & b_{1*} : 1^{\text{st}} \text{ neuron bias} & & & &
 \end{array}$$

- Bias

- To offset the result - It helps the models to shift the activation function towards the positive or negative side.
- A neural network can approximate any linear function of the form $y=wx + b$.
 - When $c=0$, then $y=wx$ and the neural network can approximate only the functions passing through origin.
 - If a function includes the constant term b , it can approximate any of the linear functions in the plane.

Activation Function

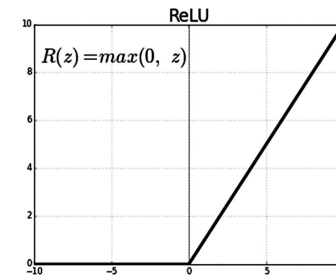
- Defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network
- All hidden layers typically use the same activation function.
- The output layer will typically use a different activation function from the hidden layers and is dependent upon the type of prediction required by the model.
- Activation functions are also typically differentiable – for backward propagation.

<https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/>

Activation for hidden layers – commonly used

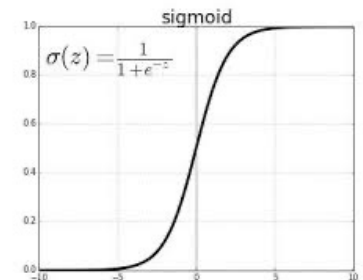
- Rectified Linear Activation (**ReLU**)

- simple to implement
- overcomes the vanishing gradient problem, allowing models to learn faster and perform better.
- It is recommended as the default for both Multilayer Perceptron (MLP) and Convolutional Neural Networks (CNNs).



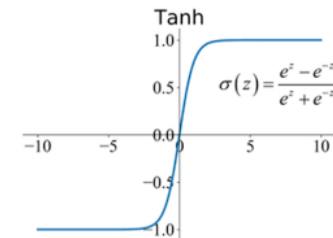
- Logistic (**Sigmoid**)

- The input to the function is transformed into a value between 0.0 and 1.0. Inputs that are much larger than 1.0 are transformed to the value 1.0, similarly, values much smaller than 0.0 are snapped to 0.0.
- A general problem with both the sigmoid and tanh functions is that they saturate. Once saturated, it becomes challenging for the learning algorithm to continue to adapt the weights to improve the performance of the model.

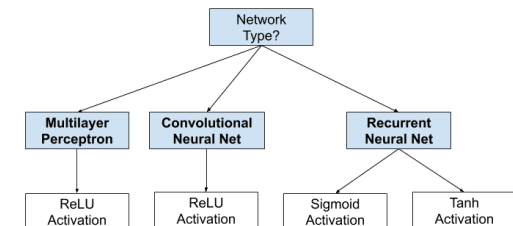


- Hyperbolic Tangent (**Tanh**)

- The function takes any real value as input and outputs values in the range -1 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.



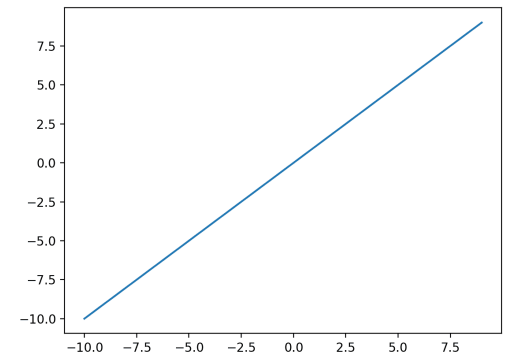
How to Choose an Hidden Layer Activation Function



Activation for output layers – commonly used

- Linear
 - The linear activation function is also called “*identity*” (multiplied by 1.0) or “*no activation*.”
 - This is because the linear activation function does not change the weighted sum of the input in any way and instead returns the value directly.
- Logistic (Sigmoid)
- Softmax
 - The softmax function outputs a vector of values that sum to 1.0 that can be interpreted as probabilities of class membership.

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$



Loss (error) function

Estimates the error for the current state of the model (repeatedly)

$$l(y, \hat{y}) = \frac{1}{2} (y - \hat{y})^2 \quad (\text{mean square error})$$

- gradient decent not work well – non-convex; many local optimal

$$l(y, \hat{y}) = -(\hat{y} \log(y) + (1 - \hat{y}) \log(1 - y))$$

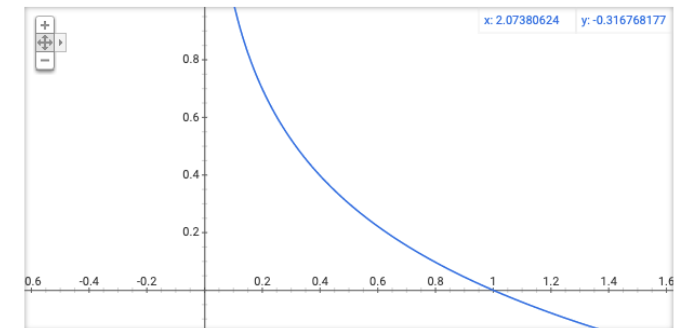
(binary cross-entropy loss)

- If $\hat{y} = 1$ then $\mathcal{L}(y, \hat{y}) = -\log(y)$, want $\log(y)$ large, want y large
- If $\hat{y} = 0$, then $\mathcal{L}(y, \hat{y}) = -\log(1-y)$, want $\log(1-y)$ large, want y small

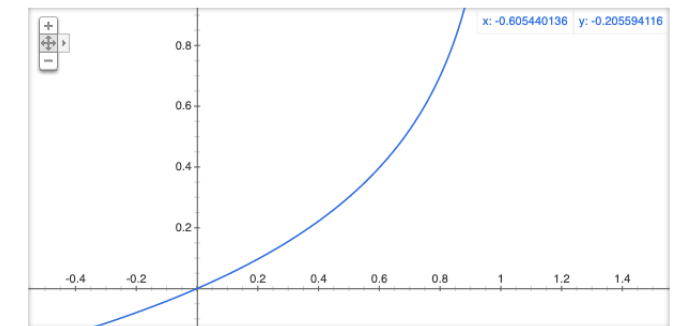
Cost Function

$$\mathcal{L}(w, b) = \frac{1}{m} \sum_{i=1}^m l(y^{(i)}, \hat{y}^{(i)})$$

Graph for $-\log(x)$



Graph for $-\log(1-x)$



Regression loss function

A regression predictive modeling problem involves predicting a real-valued quantity.

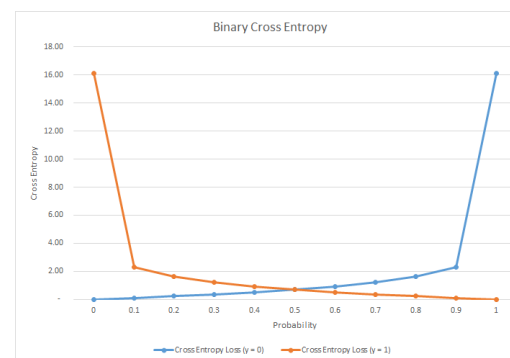
- Mean Squared Error Loss
 - Calculates the average of the squared differences between the predicted and actual values.
 - The result is always positive regardless of the sign of the predicted and actual values and a perfect value is 0.0.
 - The squaring means that larger mistakes result in more error than smaller mistakes, meaning that the model is punished for making larger mistakes.
- Mean Squared Logarithmic Error Loss
 - First calculates the natural logarithm of each of the predicted values, then calculate the mean squared error
 - When you may not want to punish a model as heavily as mean squared error.
- Mean Absolute Error Loss
 - Calculates the average of the absolute difference between the actual and predicted values.
 - On some regression problems, the distribution of the target variable may be mostly Gaussian, but may have outliers, e.g. large or small values far from the mean value. MAE is more robust to outliers.

Binary Classification Loss Functions

Binary classification are those predictive modeling problems where examples are assigned one of two labels

- Binary Cross-Entropy Loss
 - intended for use with binary classification where the target values are in the set $\{0, 1\}$.
- Hinge Loss
 - primarily developed for use with Support Vector Machine (SVM) models.
 - intended for use with binary classification where the target values are in the set $\{-1, 1\}$
- Squared Hinge Loss
 - calculates the square of the score hinge loss.
 - has the effect of smoothing the surface of the error function and making it numerically easier to work with.

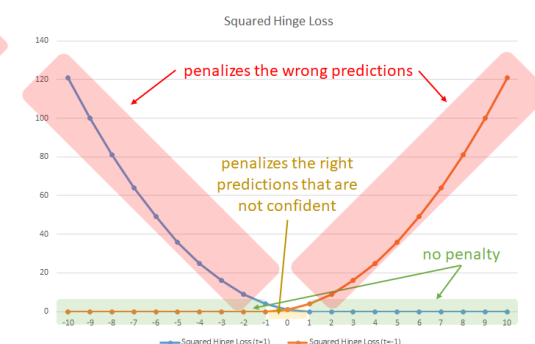
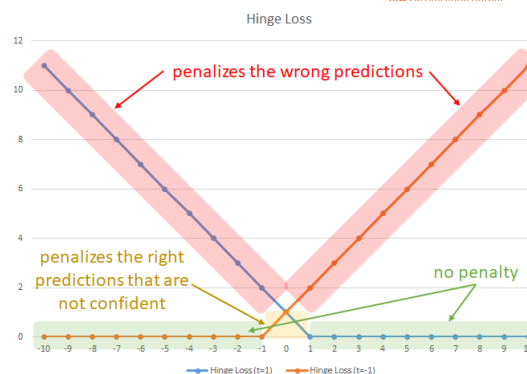
$$L = -y * \log(p) - (1 - y) * \log(1 - p) = \begin{cases} -\log(1 - p), & \text{if } y = 0 \\ -\log(p), & \text{if } y = 1 \end{cases}$$



$$\ell(y) = \max(0, 1 - t \cdot y)$$

intended output $t = \pm 1$
class labels: +1 or -1
raw classification output

$$L(y, \hat{y}) = \sum_{i=0}^N (\max(0, 1 - y_i \cdot \hat{y}_i)^2)$$



Multi-Class Classification Loss Functions

Multi-Class classification are those predictive modeling problems where examples are assigned one of more than two classes.

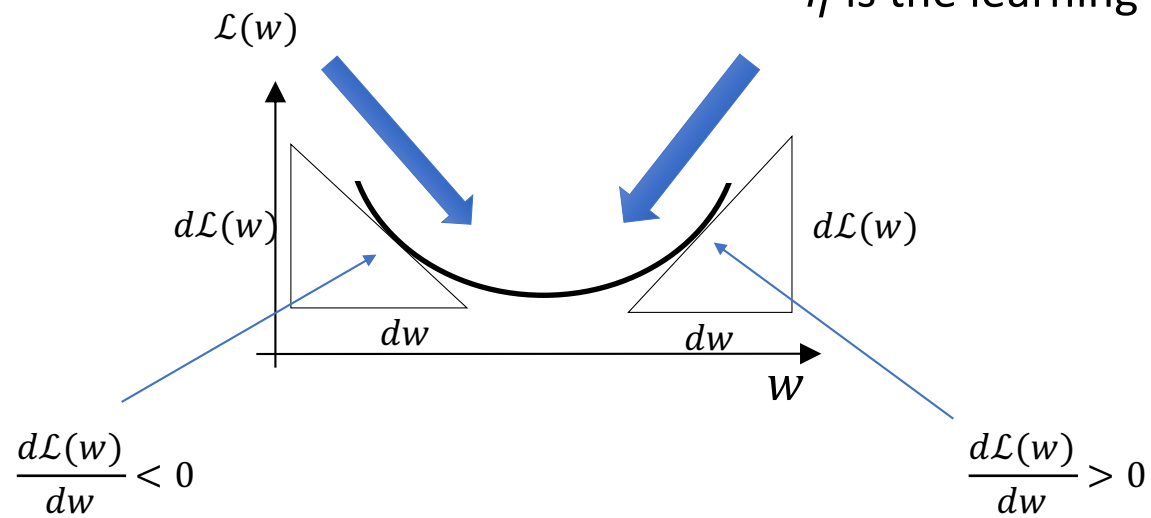
- Multi-Class Cross-Entropy Loss
 - for use with multi-class classification where the target values are in the set, where each class is assigned a unique integer value.
- Sparse Multiclass Cross-Entropy Loss
 - A possible cause of frustration when using cross-entropy with classification problems with a large number of labels is the one hot encoding process.
 - **Sparse cross entropy** addresses this by performing the *same* cross-entropy calculation of error, without requiring that the target variable be one hot encoded prior to training.
- Kullback Leibler Divergence Loss
 - The **Kullback-Liebler Divergence** is a measure of how a probability distribution differs from another distribution. A KL-divergence of zero indicates that the distributions are identical.

Gradient Descent

Repeat

$$w = w - \eta \frac{d\mathcal{L}(w)}{dw}$$

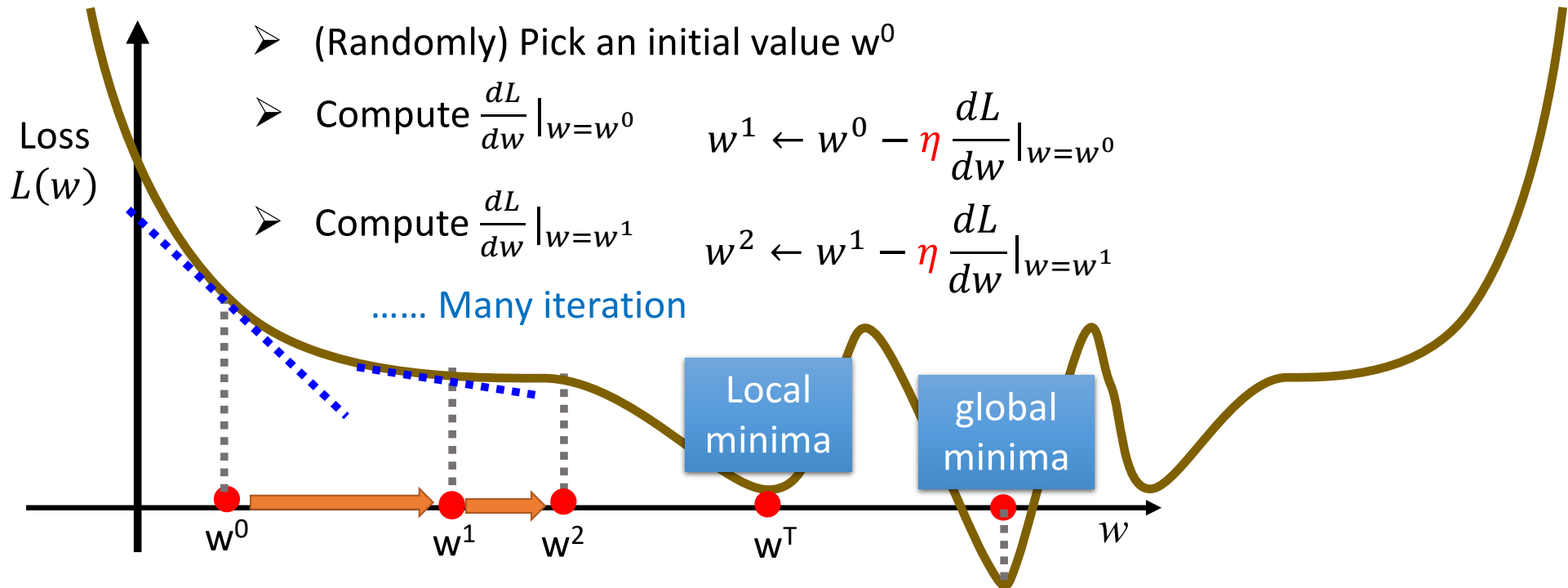
η is the learning rate



Gradient Descent

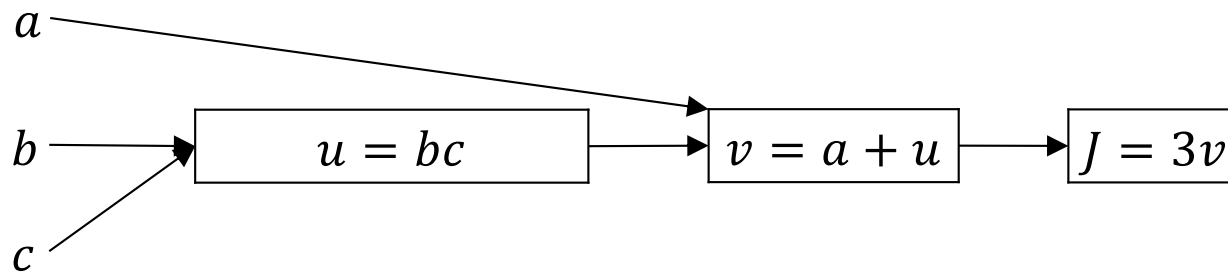
$$w^* = \arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w :



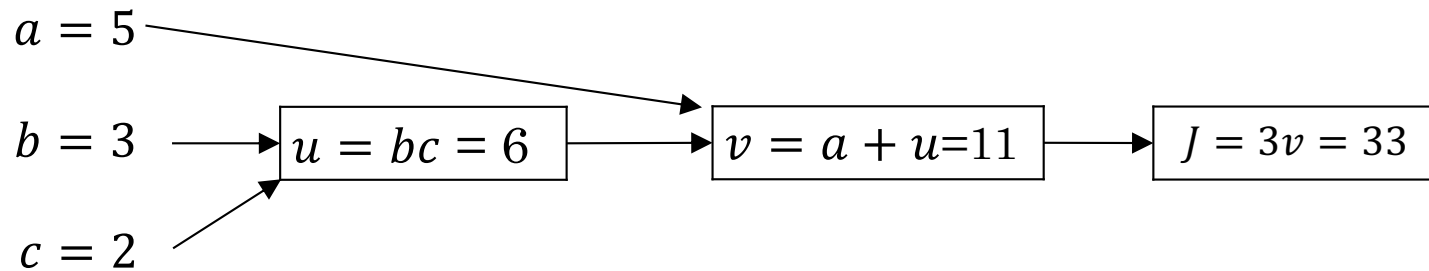
Computation graph

- $J(a,b,c)=3(a+bc)$
 - $u=bc$
 - $v=a+u$
 - $J=3v$



- $J(5,3,2) \rightarrow u=6 \rightarrow v=11 \rightarrow J=33$

Computing derivatives



- $\frac{dJ}{dv} = 3, (v = 11 \rightarrow 11.001 \text{ then } J = 33 \rightarrow 33.003)$
- $\frac{dJ}{da} = \frac{dJ}{dv} \frac{dv}{da} = 3 \times 1 = 3$ (chain rule), ($a = 5 \rightarrow 5.001$ then $J = 33 \rightarrow 33.003$)
- $\frac{dJ}{du} = \frac{dJ}{dv} \frac{dv}{du} = 3 \times 1 = 3$, ($u = 6 \rightarrow 6.001$ then $J = 33 \rightarrow 33.003$)
- $\frac{dJ}{db} = \frac{dJ}{dv} \frac{dv}{du} \frac{du}{db} = 3 \times 1 \times 2 = 6$, ($b = 3 \rightarrow 3.001$, $u = 6 \rightarrow 6.002$ then $J = 33 \rightarrow 33.006$)
- $\frac{dJ}{dc} = \dots = 9$

A simple numerical example

- Learn output = 2 x input
- Step 1: model initialization
 - Model generic form: $y=Wx$
 - Model 1: $y=3x$, Model 2: $y=5x$, Model 3: $y=0.5x$
 - All these models with different randomly generated coefficient W will converge

A simple numerical example

- Step 2: Forward propagation

| Input | Output of model 1: $y=3x$ |
|-------|---------------------------|
| 0 | 0 |
| 1 | 3 |
| 2 | 6 |
| 3 | 9 |
| 4 | 12 |

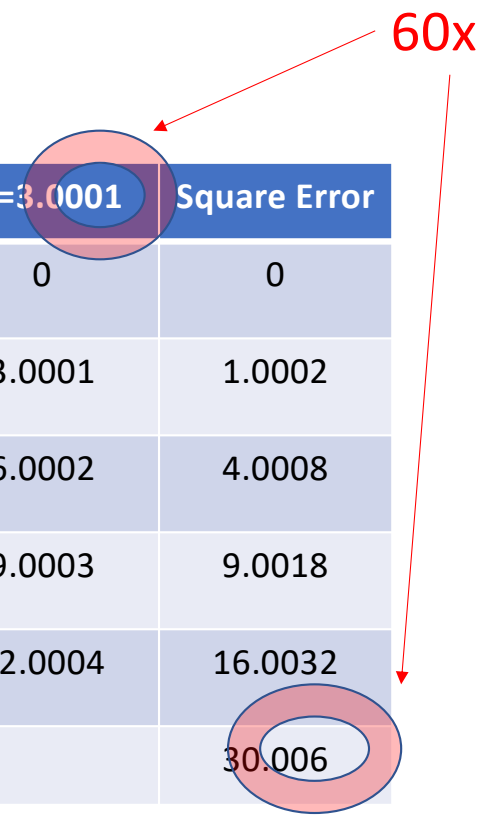
A simple numerical example

- Step 3: Loss function

| Input | Actual | Desired | Absolute Error | Square Error |
|-------|--------|---------|----------------|--------------|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 2 | 1 | 1 |
| 2 | 6 | 4 | 2 | 4 |
| 3 | 9 | 6 | 3 | 9 |
| 4 | 12 | 8 | 4 | 16 |
| Total | | | 10 | 30 |

A simple numerical example

- Step 4: Differentiation

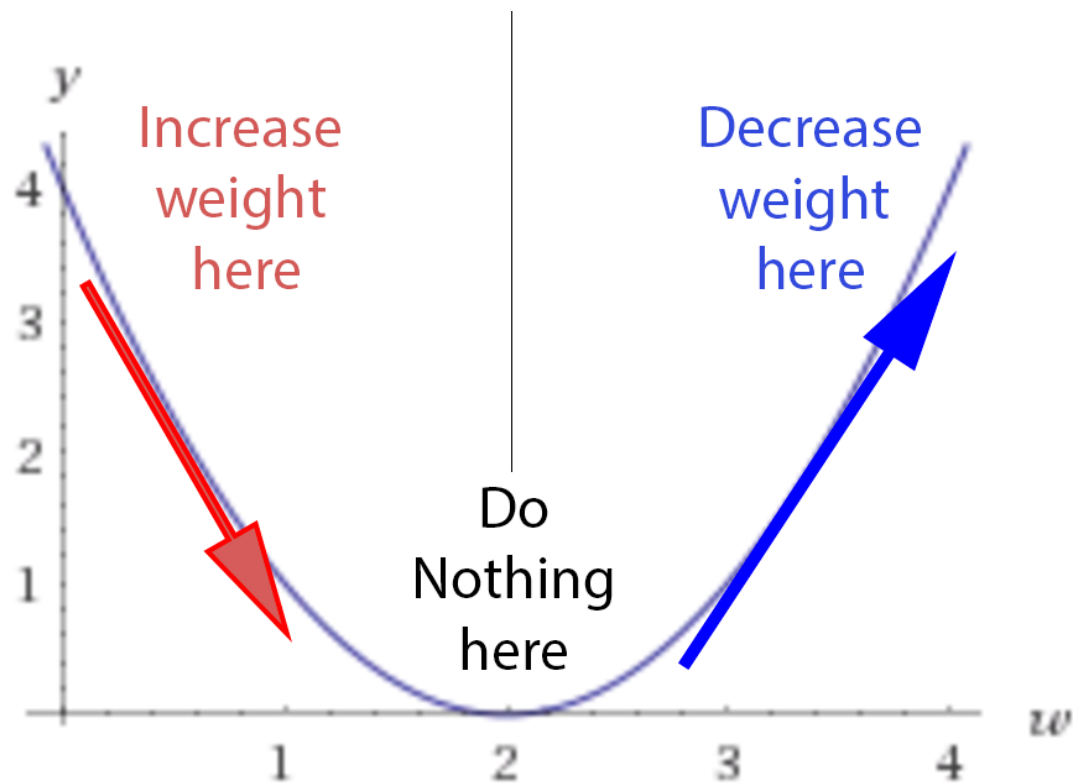


A red arrow points from the text "60x" to a magnified view of the table's data. The magnified view shows two concentric circles around the values 3.0001 and 30.006, indicating a 60-fold increase in their relative change.

| Input | Desired | W=3 | Square Error | W=3.0001 | Square Error |
|-------|---------|-----|--------------|----------|--------------|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 1 | 3.0001 | 1.0002 |
| 2 | 4 | 6 | 4 | 6.0002 | 4.0008 |
| 3 | 6 | 9 | 9 | 9.0003 | 9.0018 |
| 4 | 8 | 12 | 16 | 12.0004 | 16.0032 |
| Total | | | 30 | | 30.006 |

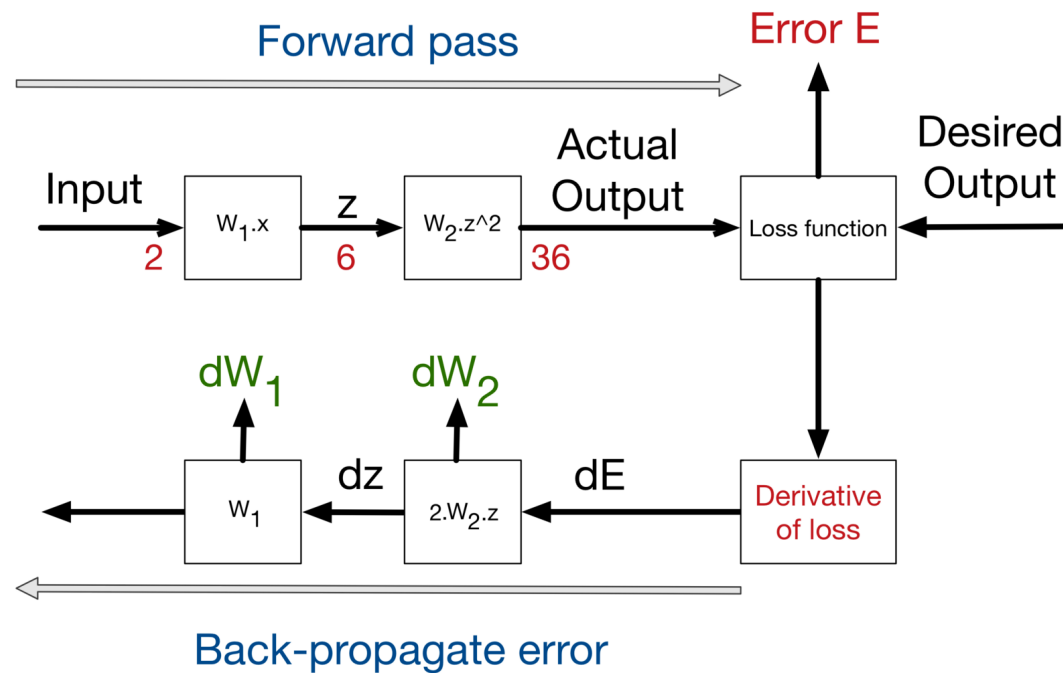
- what we really care about is the rate of which the error changes **relatively** to the changes on the weight.

A simple numerical example



A simple numerical example

- Step 5 - Backward propagation



A simple numerical example

- Step 6: Weight update

$$w^1 \leftarrow w^0 - \eta \frac{dL}{dw} \Big|_{w=w^0}$$

- 1 unit of change in weights leads to 60 total units of change in the loss function.

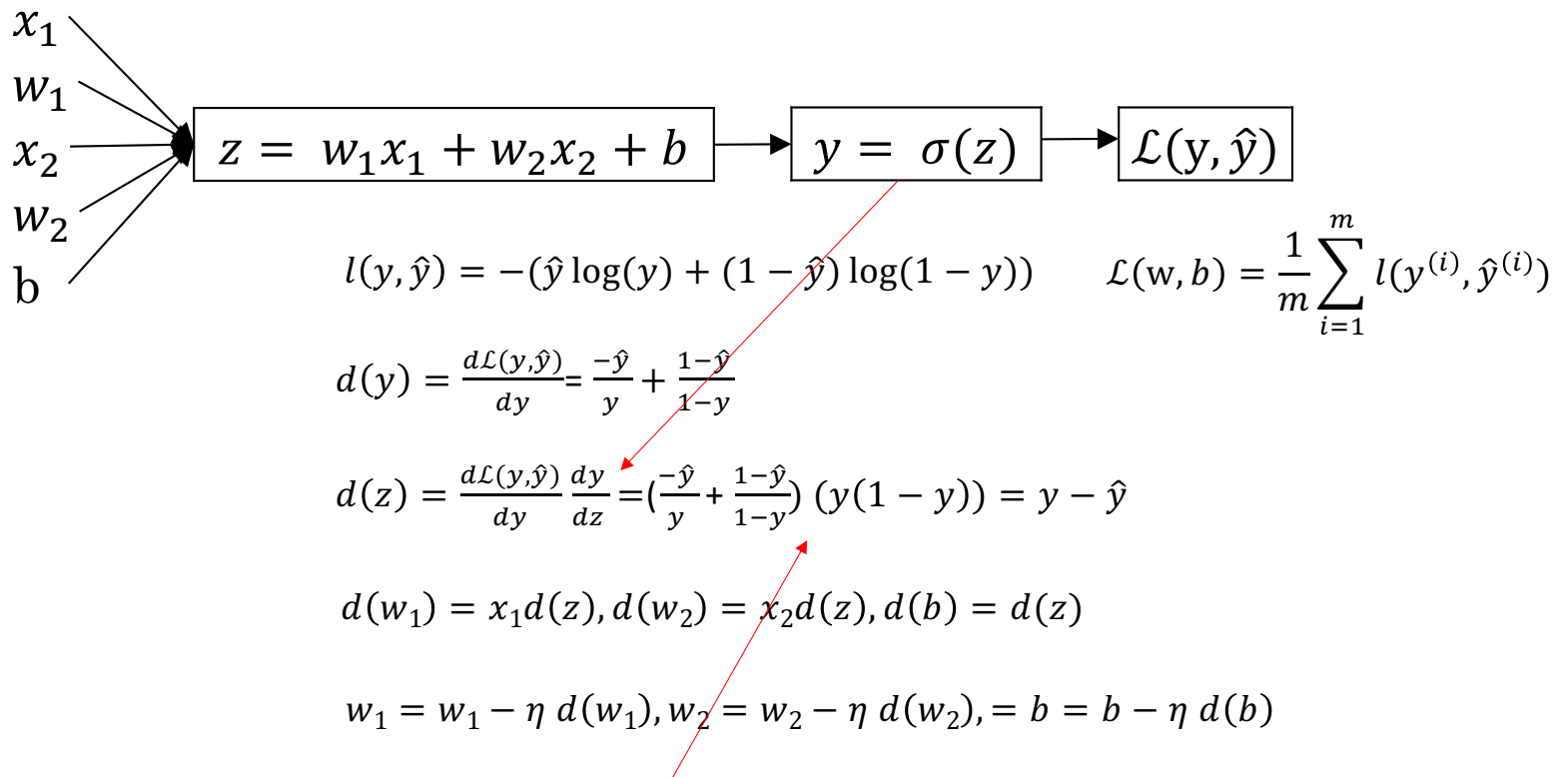
Loss error is currently at 30 units, by extrapolating the rate, in order to reduce the error to 0, we need to reduce the weights by 0.5 units.

- Step 7: Iterate until convergence

Iterations

| iteration | | 1 | | 2 | | 3 | | 4 | | 5 | |
|-----------|---------|----|---------|-----|---------|-------|----------|-----------|------------|------------|------------|
| | | w | | w | | w | | w | | w | |
| Input | Desired | 3 | Sqr Err | 2.5 | Sqr Err | 2.375 | Sqr Err | 2.3046875 | Sqr Err | 2.25827026 | Sqr Err |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | 1 | 2.5 | 0.25 | 2.375 | 0.140625 | 2.3046875 | 0.09283447 | 2.25827026 | 0.06670353 |
| 2 | 4 | 6 | 4 | 5 | 1 | 4.75 | 0.5625 | 4.609375 | 0.37133789 | 4.51654053 | 0.26681412 |
| 3 | 6 | 9 | 9 | 7.5 | 2.25 | 7.125 | 1.265625 | 6.9140625 | 0.83551025 | 6.77481079 | 0.60033176 |
| 4 | 8 | 12 | 16 | 10 | 4 | 9.5 | 2.25 | 9.21875 | 1.48535156 | 9.03308105 | 1.06725647 |
| | | | 30 | | 7.5 | | 4.21875 | | 2.78503418 | | 2.00110587 |

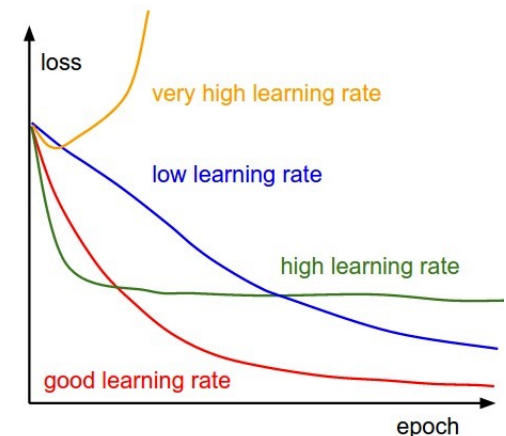
Backward Propagation



<https://medium.com/binaryandmore/beginners-guide-to-deriving-and-implementing-backpropagation-e3c1a5a1e536>

Learning rate

- During training, the backpropagation of error estimates the amount of error for which the weights of a node in the network are responsible. Instead of updating the weight with the full amount, it is scaled by the learning rate.
- The learning rate hyperparameter controls the rate or speed at which the model learns.
- the optimal learning rate cannot be calculated for a given model on a given dataset. Configuring the learning rate is challenging and time-consuming.
 - Trial and error.
 - A sensitivity analysis of the learning rate for the chosen model, also called a grid search may help
 - Try to find where a good learning rates may reside
 - Explore the relationship between learning rate and performance.
 - It is common to grid search learning rates on a log scale from 0.1 to 10^{-5} or 10^{-6}



Learning rate

- Add Momentum to the Learning Process
 - An exponentially weighted average of the prior updates to the weight can be included when the weights are updated
 - The momentum is known to speed up learning and to help not getting stuck in local minima.
- Use a Learning Rate Schedule
 - To vary the learning rate over the training process.
 - For example, one learning rate schedule is to decrease the learning rate linearly from a large initial value to a small value. This allows large weight changes in the beginning of the learning process and small changes or fine-tuning towards the end of the learning process.
- Adaptive Learning Rates
 - improves deep learning model performance by automatically adjusting learning rates during training.
 - Examples: AdaGrad, Adam, etc.

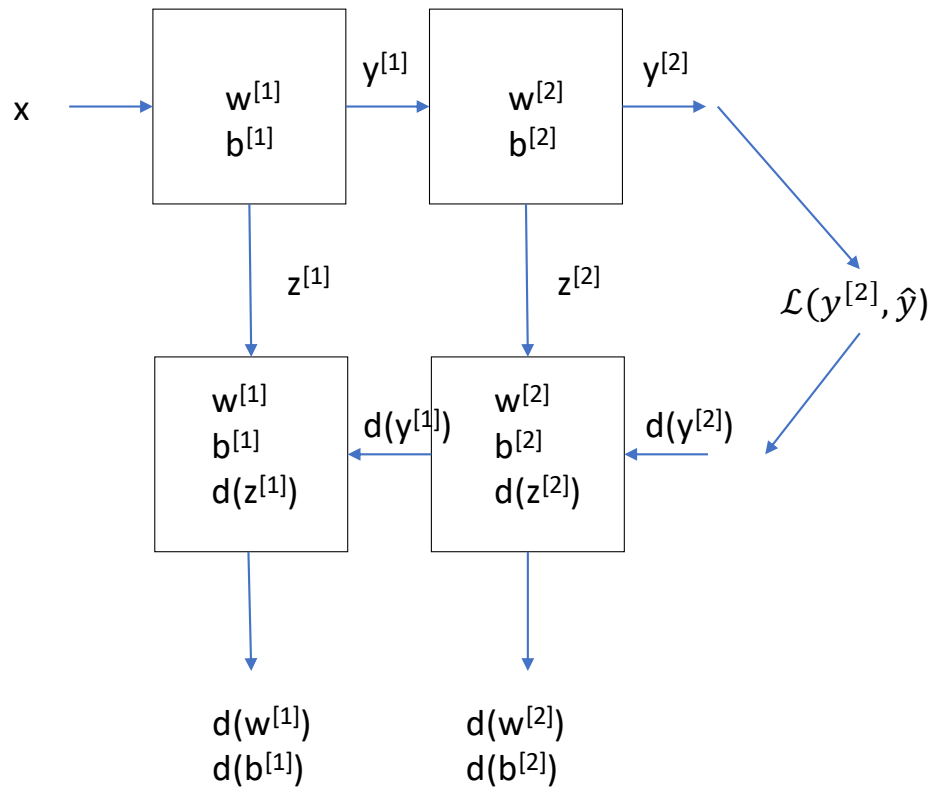
$$\Delta w_{ij} = \left(\eta * \frac{\partial E}{\partial w_{ij}} \right)$$

weight increment learning rate weight gradient

$$\Delta w_{ij} = \left(\eta * \frac{\partial E}{\partial w_{ij}} \right) + (\gamma * \Delta w_{ij}^{t-1})$$

momentum factor weight increment, previous iteration

Forward and backward functions



$$dZ^{[2]} = Y^{[2]} - \hat{Y}$$

$$dW^{[2]} = \left(\frac{1}{m}\right) dZ^{[2]} Y^{[1]T}$$

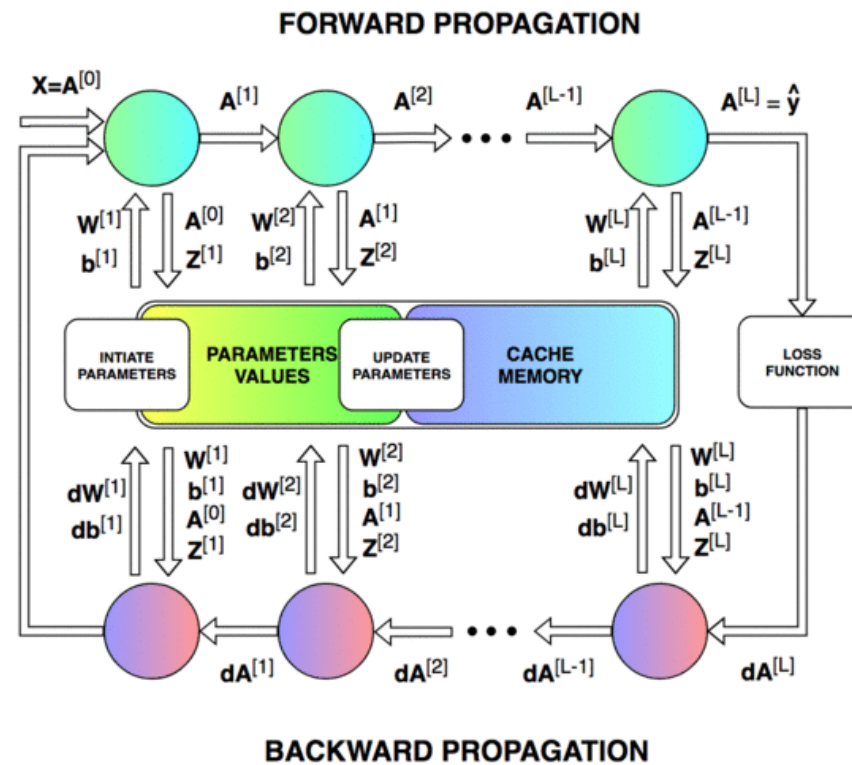
$$db^{[2]} = dZ^{[2]}$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * \sigma^{[1]'}(Z^{[1]})$$

$$dW^{[1]} = \left(\frac{1}{m}\right) dZ^{[1]} X^T$$

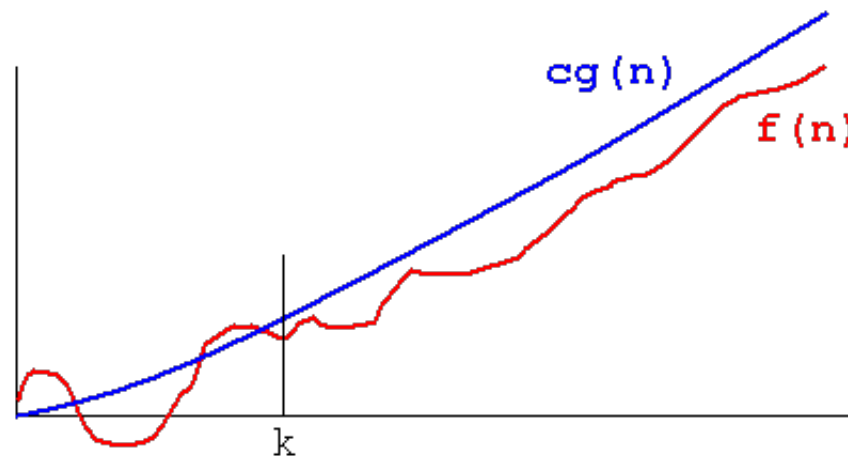
$$db^{[1]} = dZ^{[1]}$$

Forward and backward functions



Algorithm complexity – Big O notation

- **Definition:** A theoretical measure of the execution of an [algorithm](#), usually the time or memory needed, given the problem size n , which is usually the number of items. Informally, saying some equation $f(n) = O(g(n))$ means it is less than some constant multiple of $g(n)$. The notation is read, "f of n is big oh of g of n".
- **Formal Definition:** $f(n) = O(g(n))$ means there are positive constants c and k , such that $0 \leq f(n) \leq cg(n)$ for all $n \geq k$. The values of c and k must be fixed for the function f and must not depend on n .



Source: NIST

- Complexity ranking

| Function | Common name |
|--------------|--------------|
| $n!$ | factorial |
| 2^n | exponential |
| $n^d, d > 3$ | polynomial |
| n^3 | cubic |
| n^2 | quadratic |
| $n\sqrt{n}$ | |
| $n \log n$ | quasi-linear |
| n | linear |
| \sqrt{n} | root - n |
| $\log n$ | logarithmic |
| 1 | constant |

- Big O examples

| $T(n)$ | Complexity |
|---------------------------|---------------|
| $5n^3 + 200n^2 + 15$ | $O(n^3)$ |
| $3n^2 + 2^{300}$ | $O(n^2)$ |
| $5 \log_2 n + 15 \ln n$ | $O(\log n)$ |
| $2 \log n^3$ | $O(\log n)$ |
| $4n + \log n$ | $O(n)$ |
| 2^{64} | $O(1)$ |
| $\log n^{10} + 2\sqrt{n}$ | $O(\sqrt{n})$ |
| $2^n + n^{1000}$ | $O(2^n)$ |

Slide credit: UPC

Neural Network time complexity

- Forward propagation – weighted sum & activation function

Total t training examples

$$Z_{jt} = W_{ji} X_{it}, Y_{it} = \sigma(Z_{jt}) \Rightarrow O(j \cdot i \cdot t + j \cdot t) = O(j \cdot i \cdot t)$$

Multiple layers

$$O(t \cdot (ij + jk + kl + \dots)) \Rightarrow O(t \cdot \sum_{all\ layers} (input_dim \times output_dim))$$

- Backward propagation

$$\underset{jxt}{dZ^{[1]}} = \underset{jxk}{W^{[2]T}} \underset{kxt}{dZ^{[2]}} * \underset{jxt}{\sigma^{[1]'}(Z^1)} \quad \underset{jxi}{dW^{[1]}} = \underset{jxt}{dZ} \underset{txi}{X^T} \Rightarrow O(j \cdot t \cdot i)$$

Multiple layers

$$O(t \cdot \sum_{all\ layers} (input_dim \times output_dim))$$

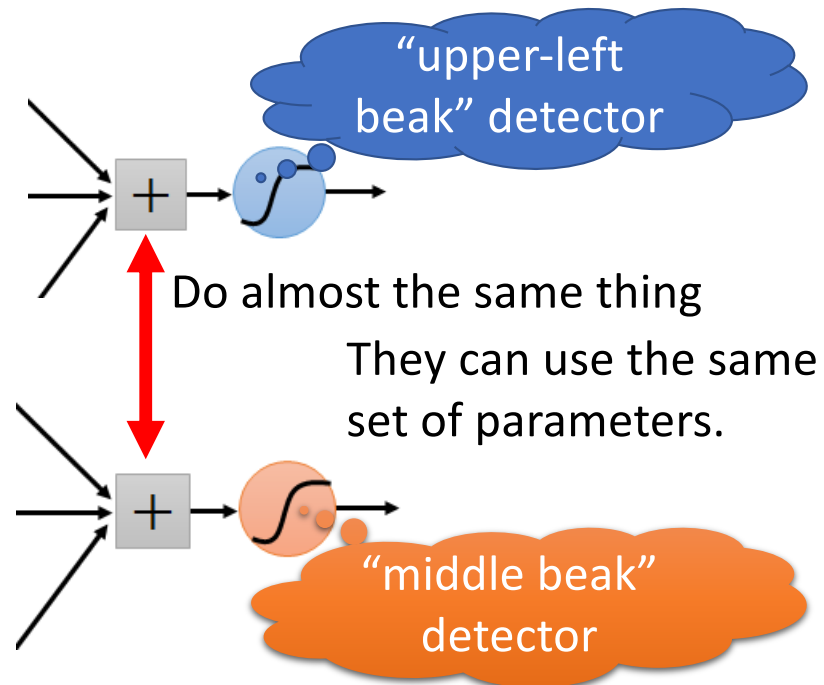
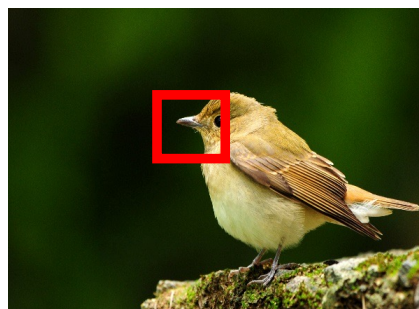
- n epochs $O(n \cdot t \cdot \sum_{all\ layers} (input_dim \times output_dim))$

Popular DNN

- Fully-Connected NN – feed forward, a.k.a. multilayer perceptron (MLP)
- Convolutional NN (CNN) – feed forward, sparsely-connected w/ weight sharing
- Recurrent NN (RNN) – feedback
 - Long Short-Term Memory (LSTM) – feedback + storage

Why CNN for Image

- The same patterns appear in different regions.



Why CNN for Image

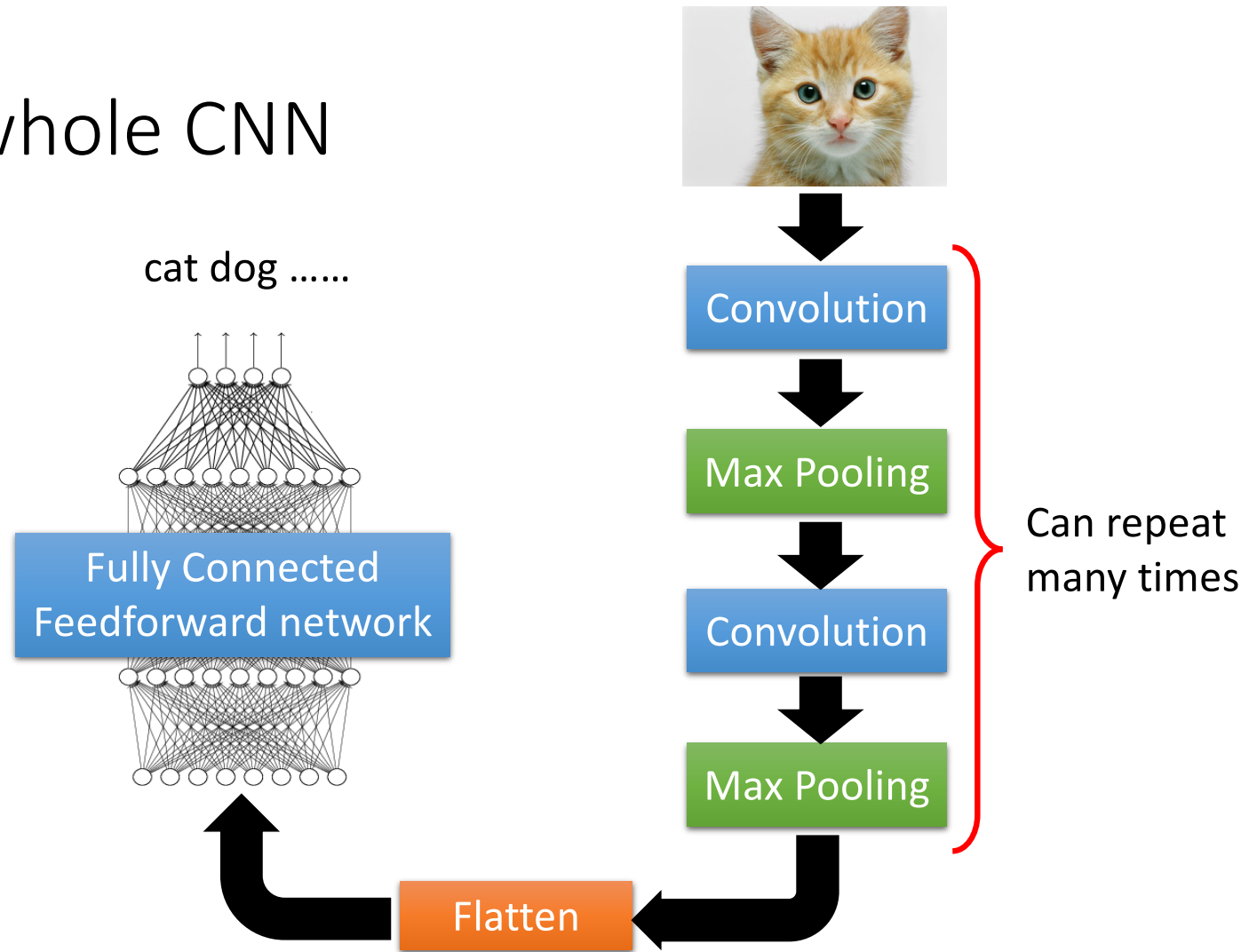
- Subsampling the pixels will not change the object



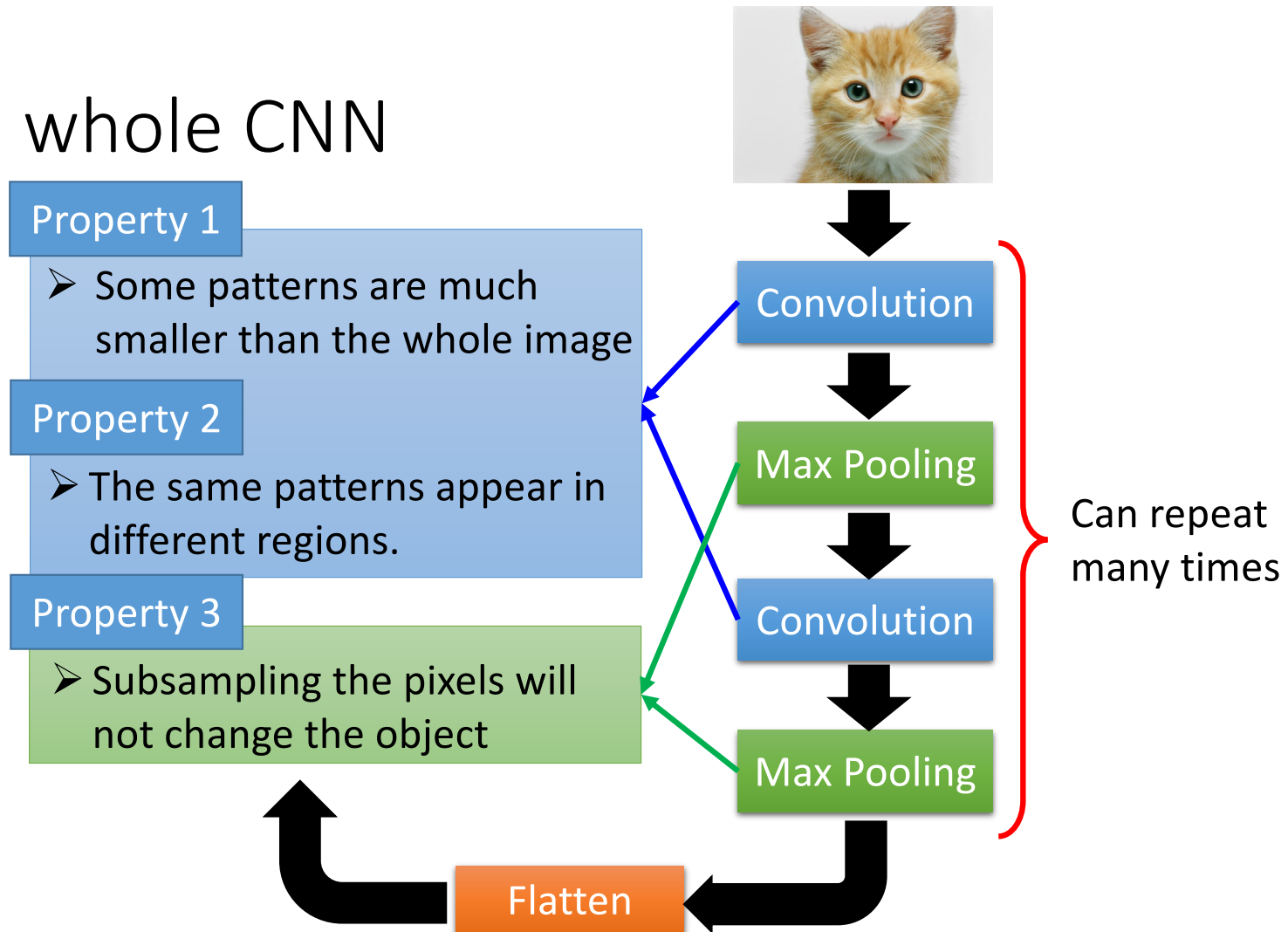
We can subsample the pixels to make image smaller

➡ Less parameters for the network to process the image

The whole CNN



The whole CNN



CNN – Convolution

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

Those are the network parameters to be learned.

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

Matrix

| | | |
|----|---|----|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2

Matrix

⋮

Property 1

Each filter detects a small pattern (3 x 3).

CNN – Convolution

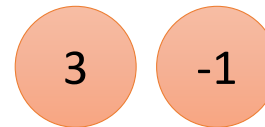
stride=1

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1



CNN – Convolution

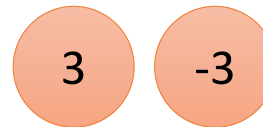
If stride=2

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

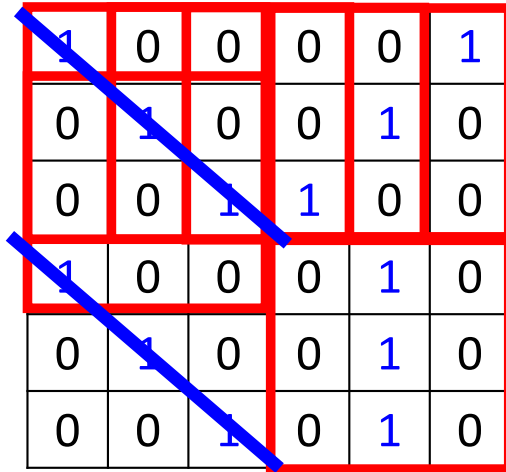
Filter 1



We set stride=1 below

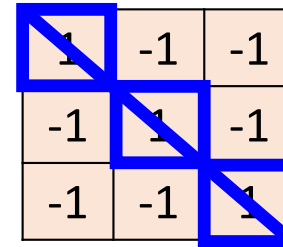
CNN – Convolution

stride=1



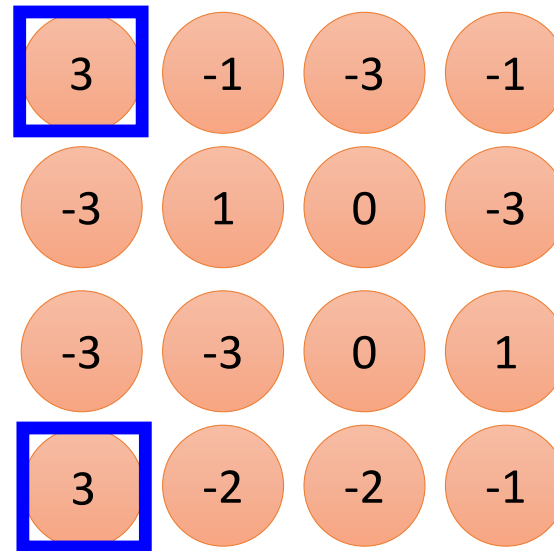
| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image



| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1



| | | | |
|----|----|----|----|
| 3 | -1 | -3 | -1 |
| -3 | 1 | 0 | -3 |
| -3 | -3 | 0 | 1 |
| 3 | -2 | -2 | -1 |

Property 2

CNN – Convolution

stride=1

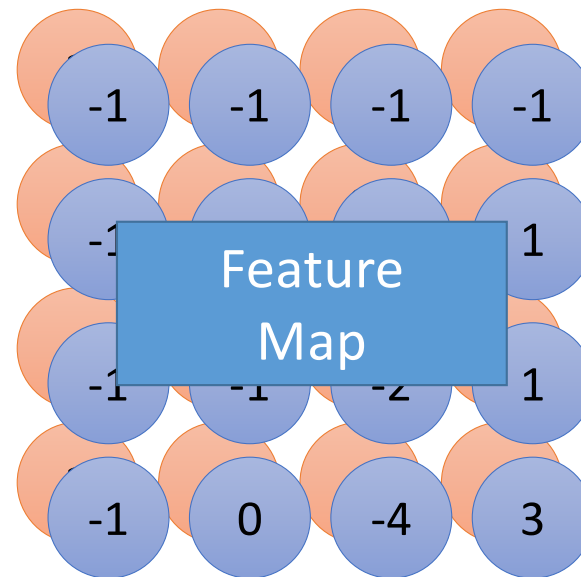
| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

| | | |
|----|---|----|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

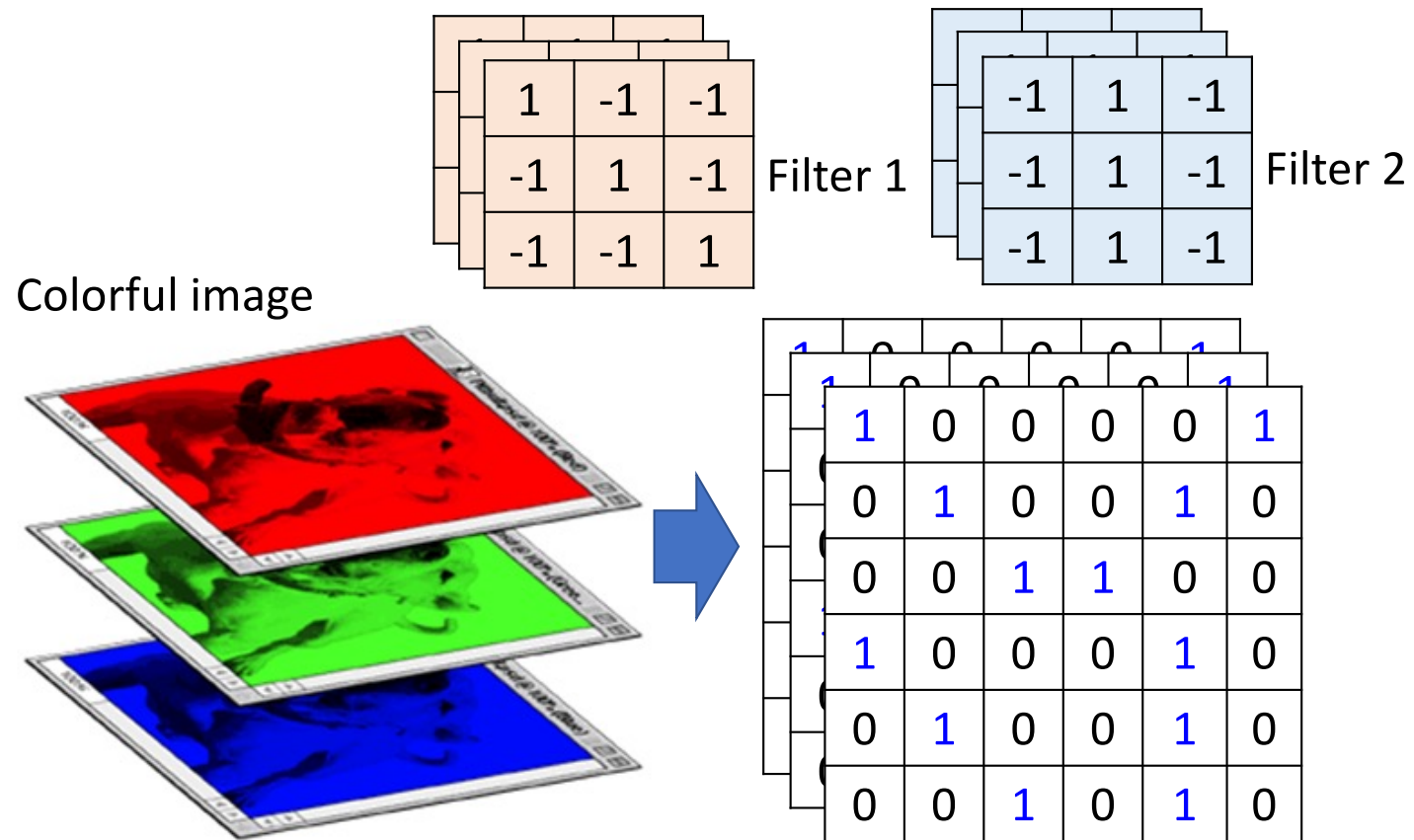
Filter 2

Do the same process for every filter

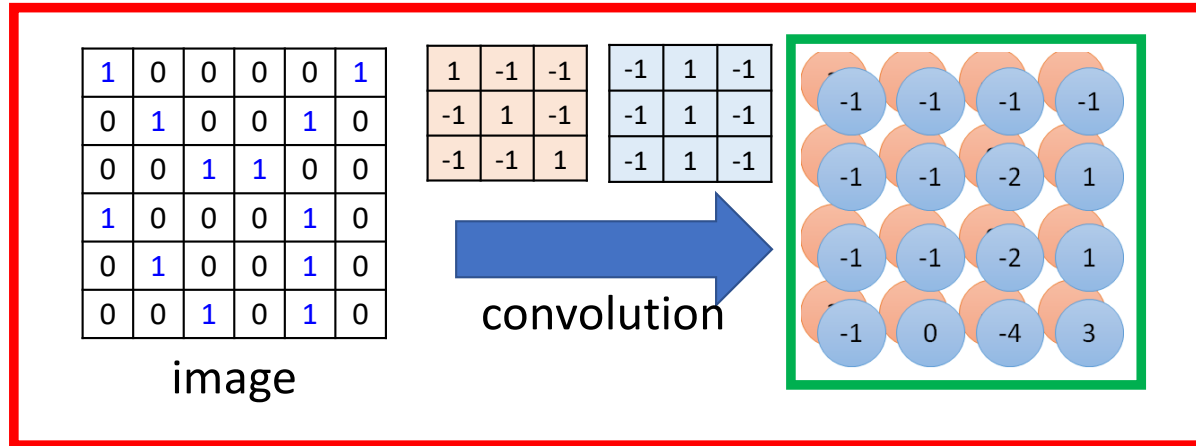


4 x 4 image

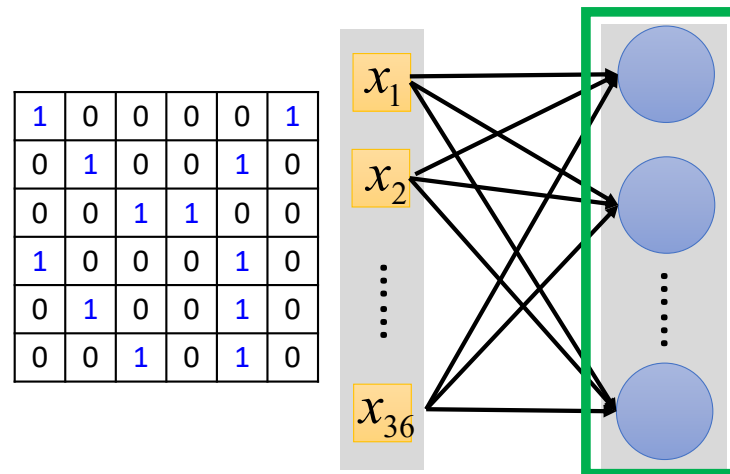
CNN – Colorful image

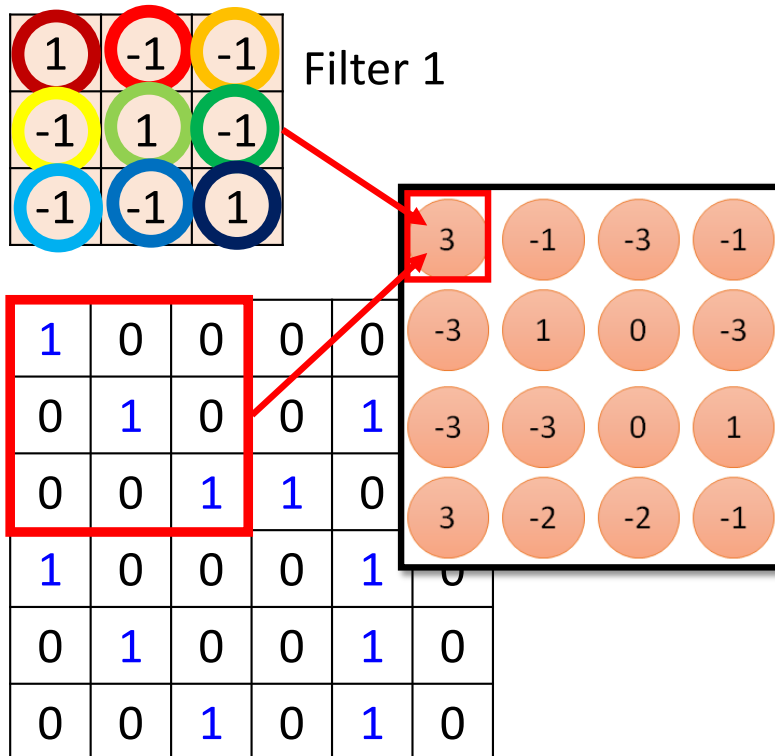


Convolution v.s. Fully Connected



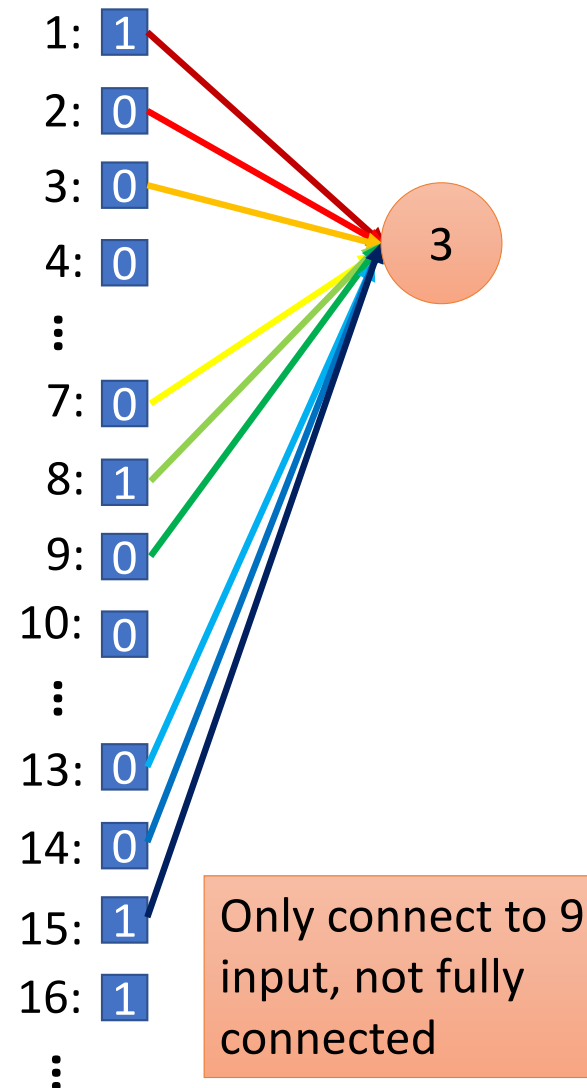
Fully-
connected

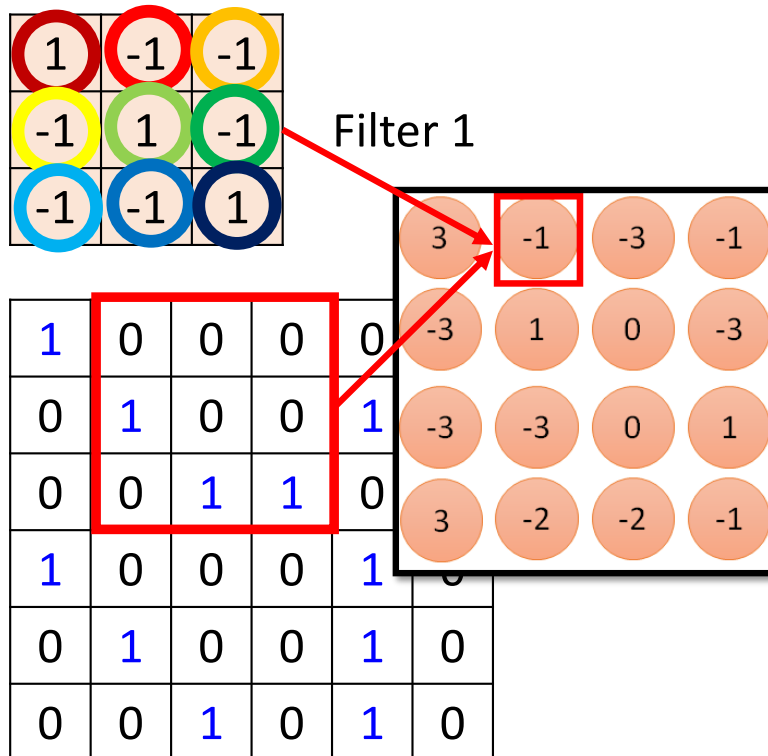




6 x 6 image

Less parameters!

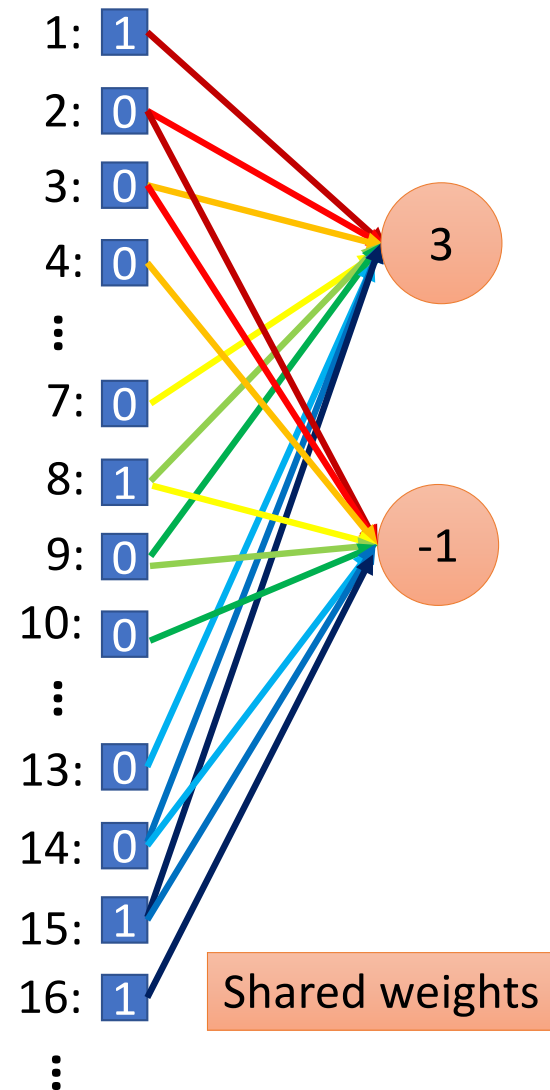




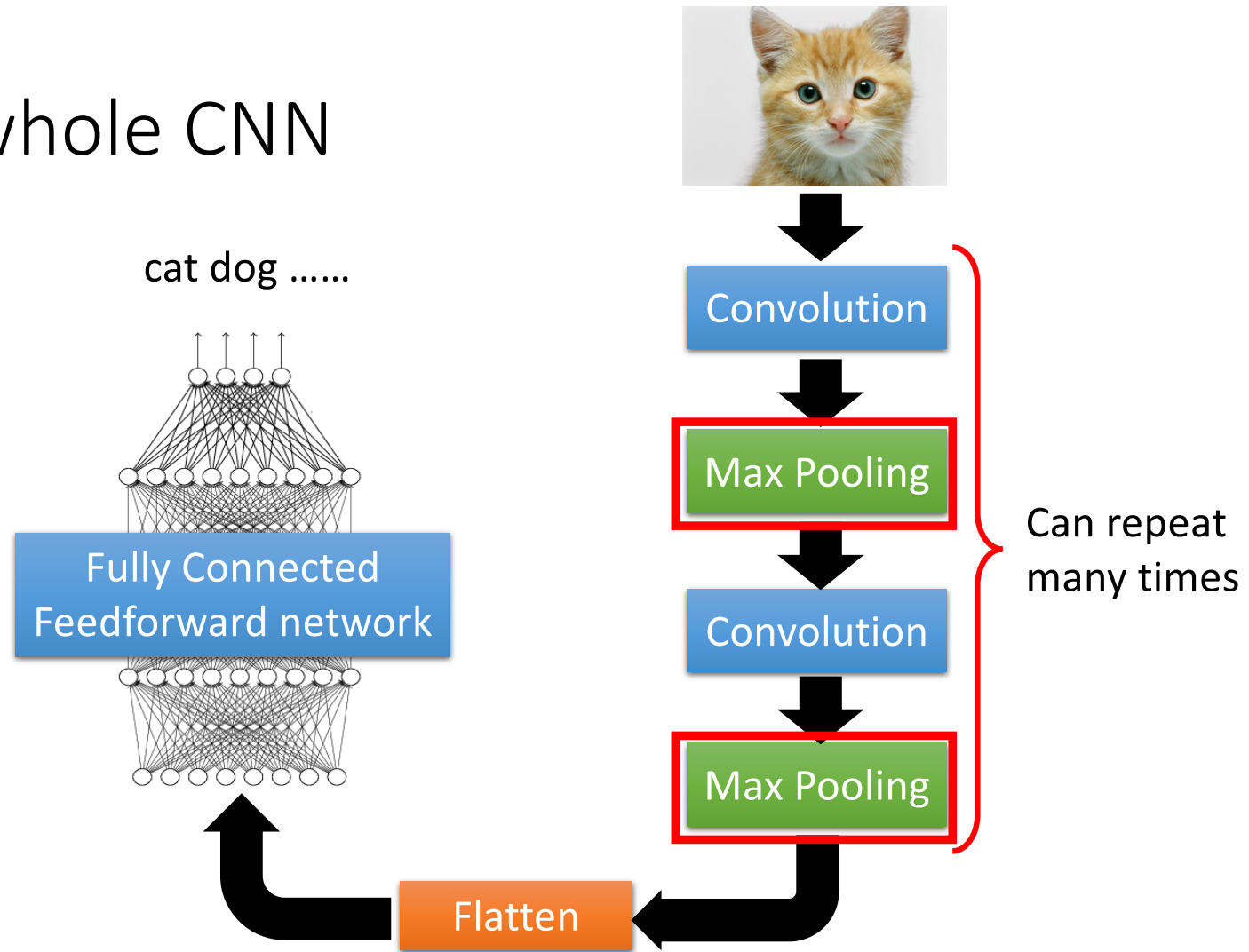
6 x 6 image

Less parameters!

Even less parameters!



The whole CNN



CNN – Max Pooling

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

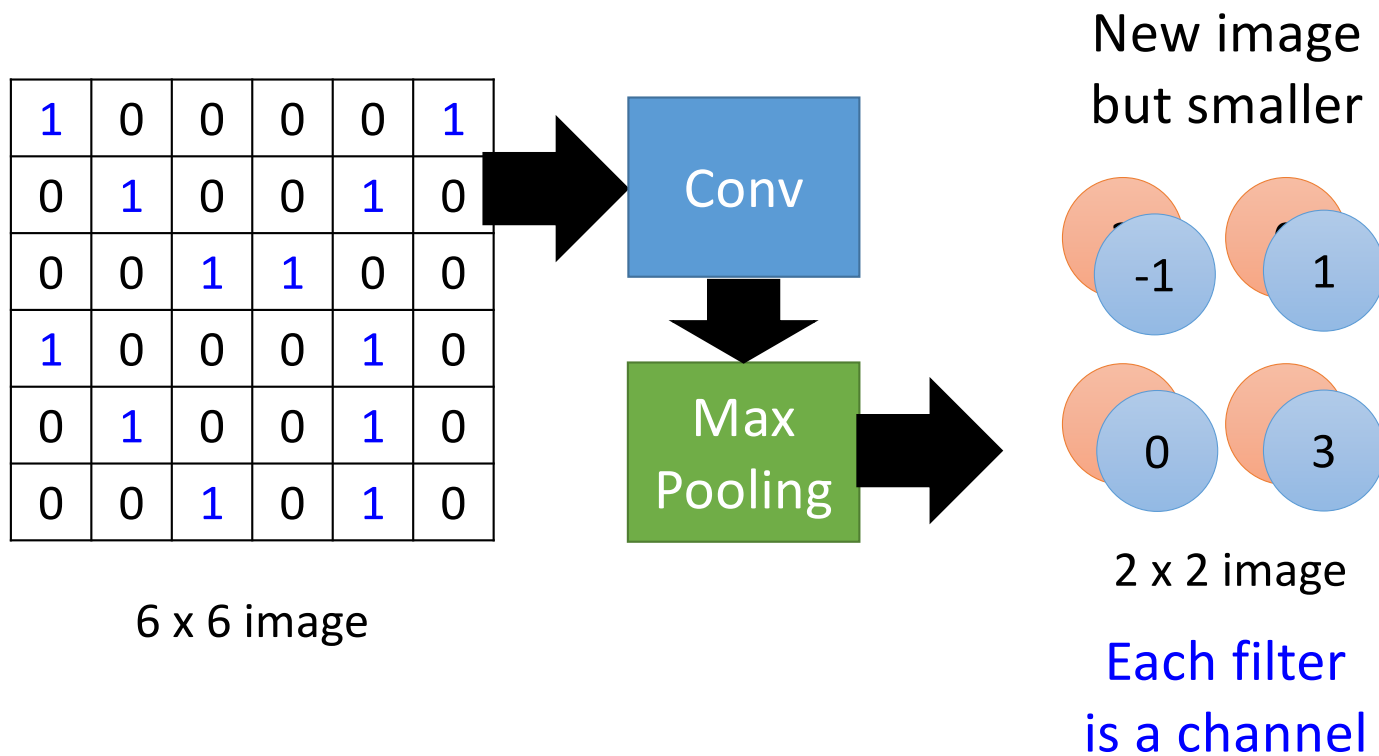
| | | |
|----|---|----|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2

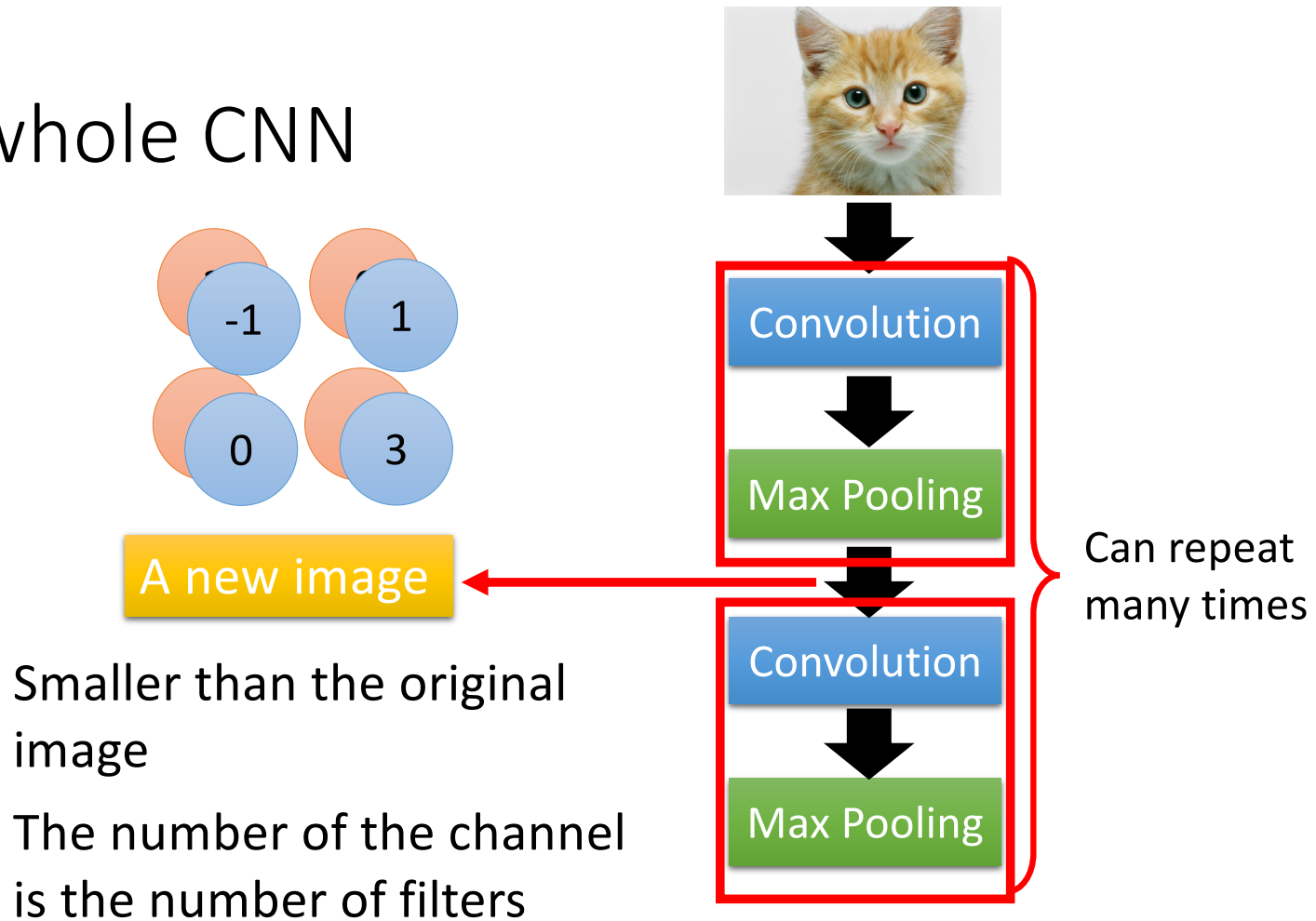
| | | | |
|----|----|----|----|
| 3 | -1 | -3 | -1 |
| -3 | 1 | 0 | -3 |
| -3 | -3 | 0 | 1 |
| 3 | -2 | -2 | -1 |

| | | | |
|----|----|----|----|
| -1 | -1 | -1 | -1 |
| -1 | -1 | -2 | 1 |
| -1 | -1 | -2 | 1 |
| -1 | 0 | -4 | 3 |

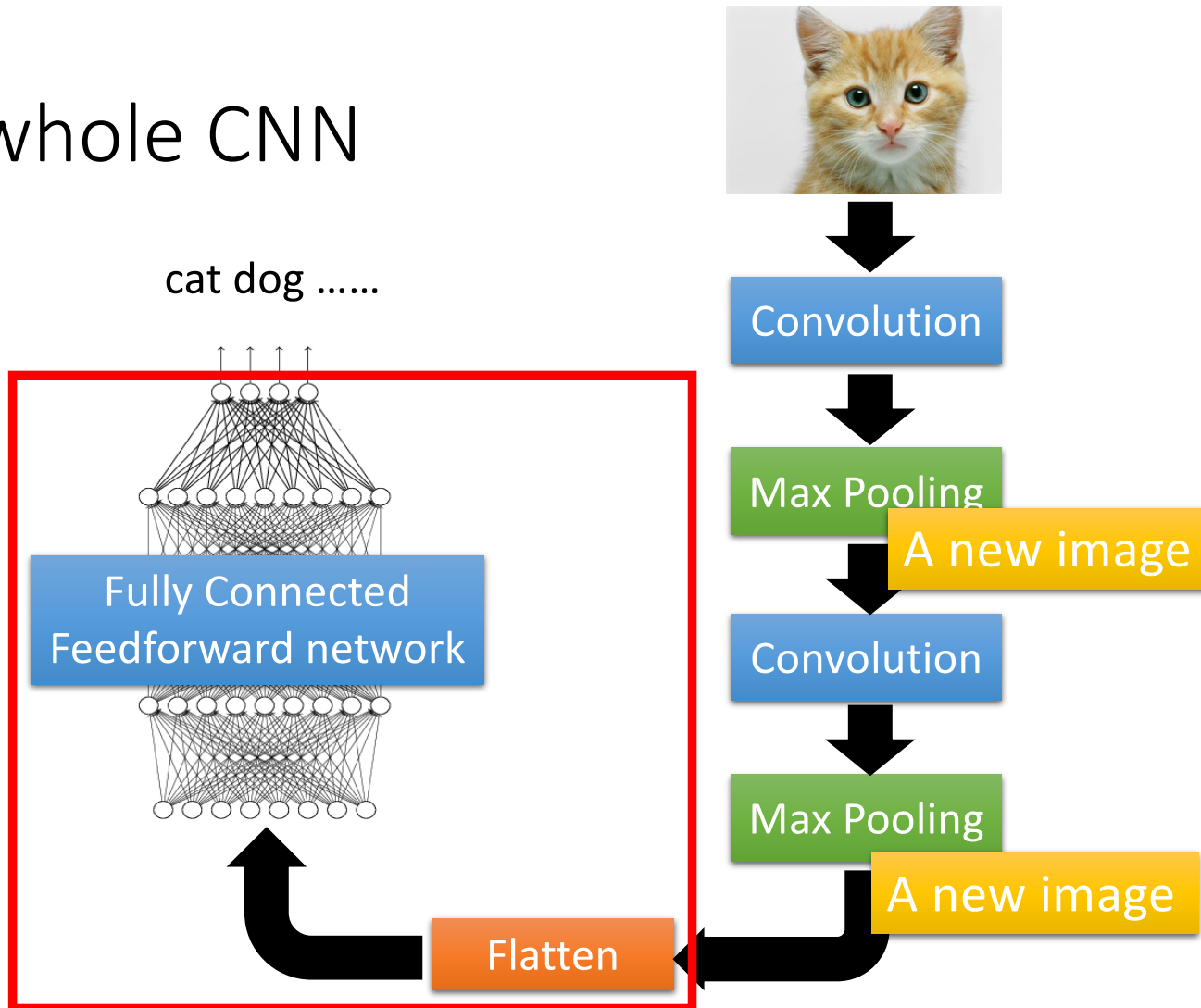
CNN – Max Pooling



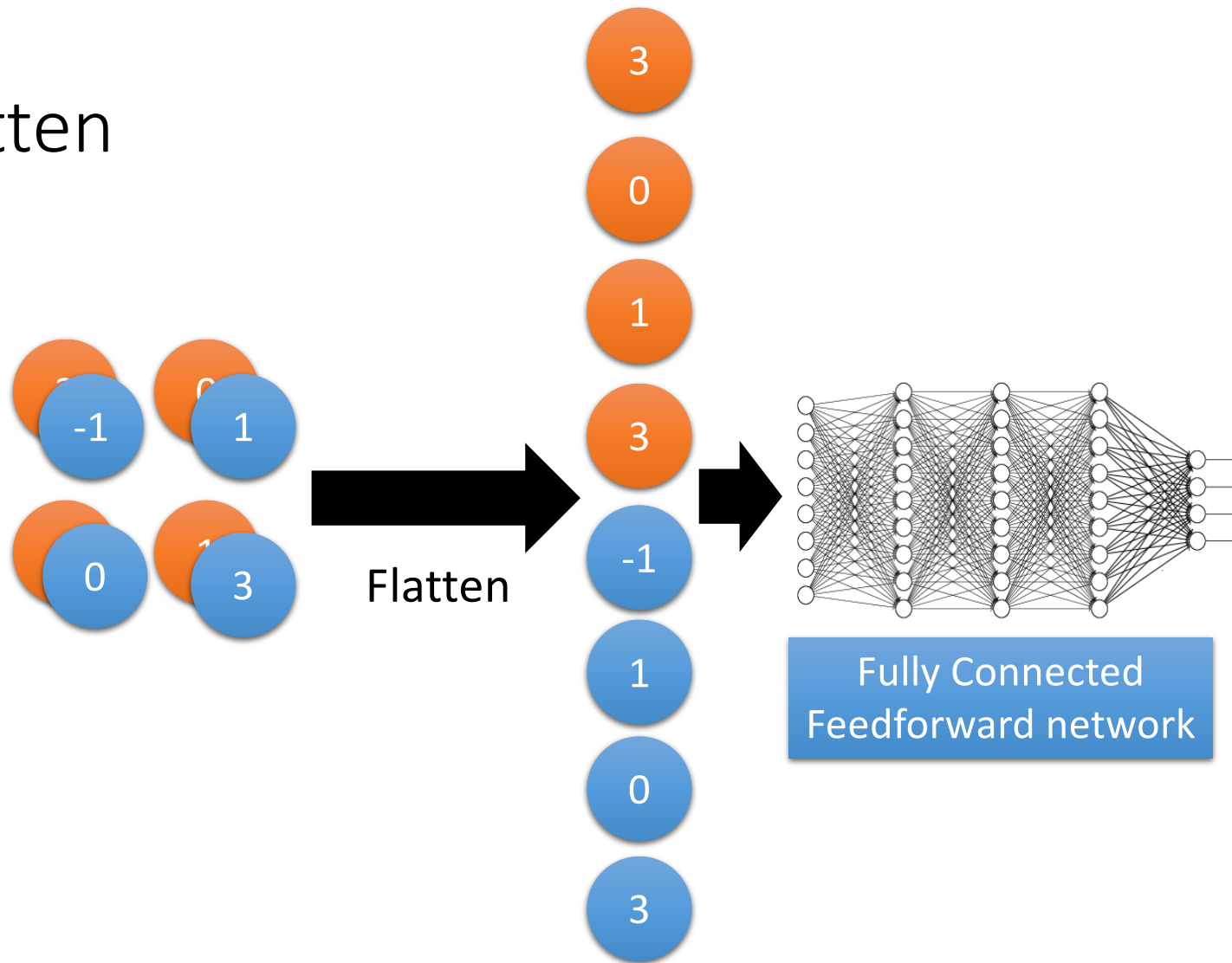
The whole CNN



The whole CNN



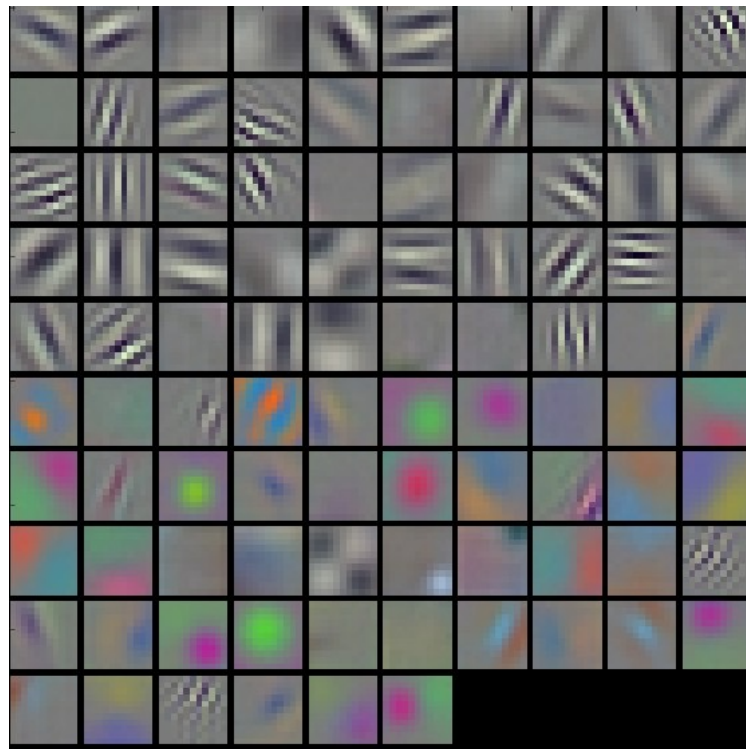
Flatten



First Convolution Layer

- Typical-looking filters on the trained first layer

11 x 11
(AlexNet)

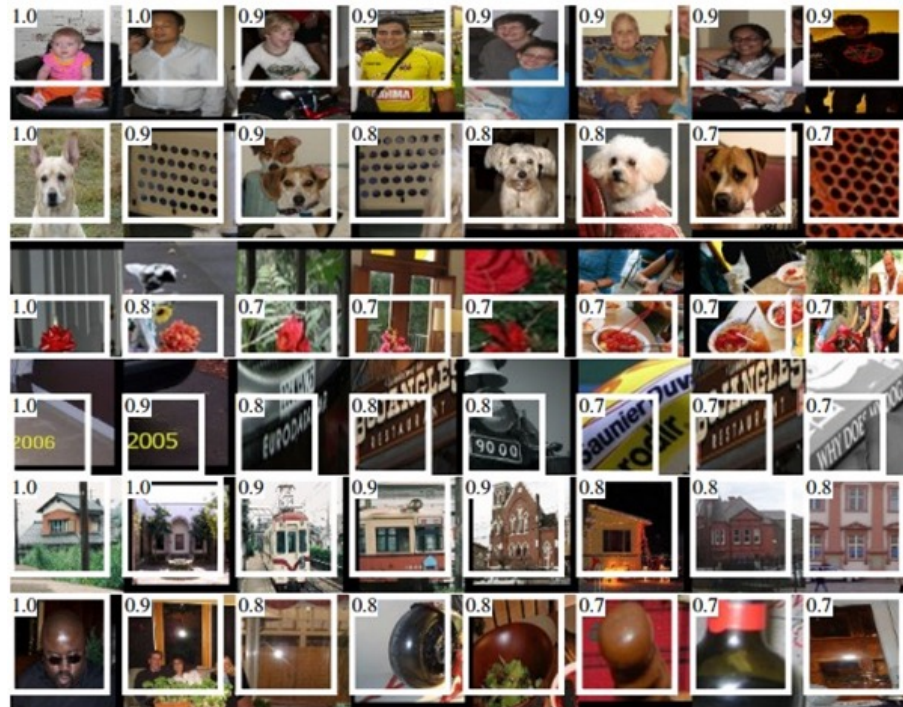


<http://cs231n.github.io/understanding-cnn/>

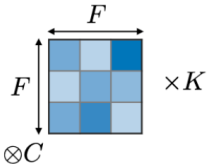
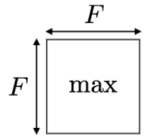
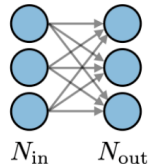
How about higher layers?

- Which images make a specific neuron activate

Ross Girshick, Jeff
Donahue, Trevor
Darrell, Jitendra Malik, “Rich
feature hierarchies for accurate
object detection and semantic
segmentation”, CVPR, 2014



Complexity of CNN

| | CONV | POOL | FC |
|----------------------|--|--|---|
| Illustration |  |  |  |
| Input size | $I \times I \times C$ | $I \times I \times C$ | N_{in} |
| Output size | $O \times O \times K$ | $O \times O \times C$ | N_{out} |
| Number of parameters | $(F \times F \times C + 1) \cdot K$ | 0 | $(N_{in} + 1) \times N_{out}$ |
| Remarks | <ul style="list-style-type: none"> • One bias parameter per filter • In most cases, $S < F$ • A common choice for K is $2C$ | <ul style="list-style-type: none"> • Pooling operation done channel-wise • In most cases, $S = F$ | <ul style="list-style-type: none"> • Input is flattened • One bias parameter per neuron • The number of FC neurons is free of structural constraints |

- $F \times F$: filter dimension, K : number of filters, C : number of channels, $I \times I$: input image dimension, $O \times O$: output image dimension, S : stride

<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks>