EE2211 Pre-Tutorial 12

Dr Feng LIN
feng_lin@nus.edu.sg

Agenda

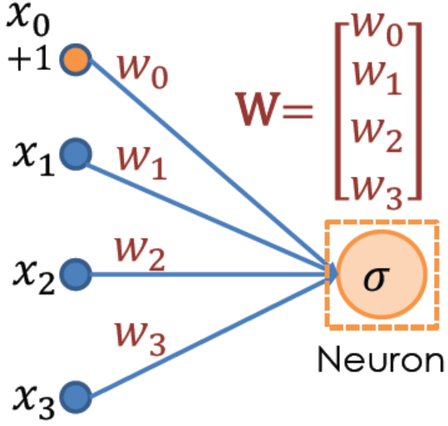
- Recap
- Self-learning
- Tutorial 12

Today’s Attendance

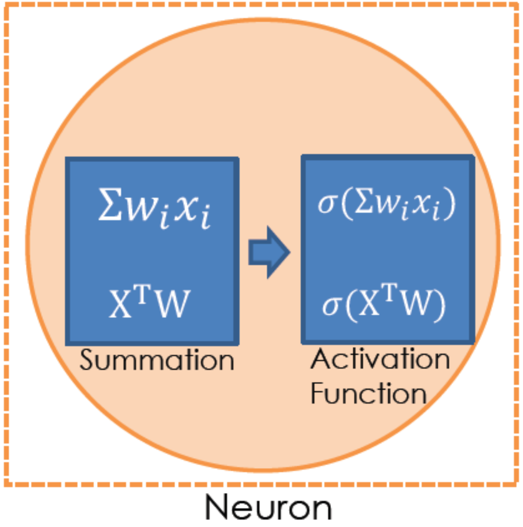
Recap

- Introduction to Neural Networks
  - Perceptron
  - Activation Functions
  - Multi-layer Perceptron
- Training and Testing of Neural Networks
  - Training: Forward and Backward
  - Testing: Forward
- Convolutional Neural Networks

Perceptron



$$\mathbf{X} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$



Output of Neuron: $\sigma(X^T W)$ or $\sigma(\Sigma w_i x_i)$

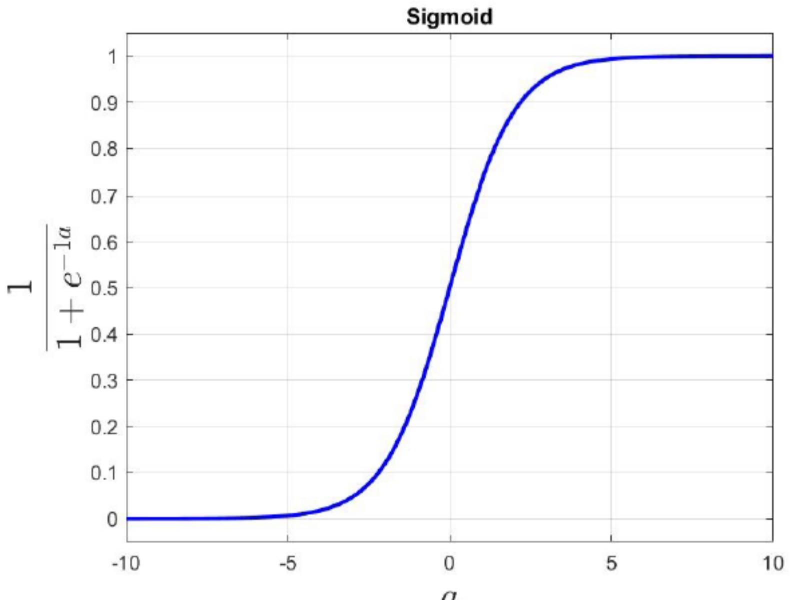
Activation Function: non-linear function to introduce non-linearity into the neural networks!

Goal of training: to learn W!

Activation Functions

Sigmoid Activation Function

$$\sigma(a) = \frac{1}{1 + e^{-\beta a}},$$

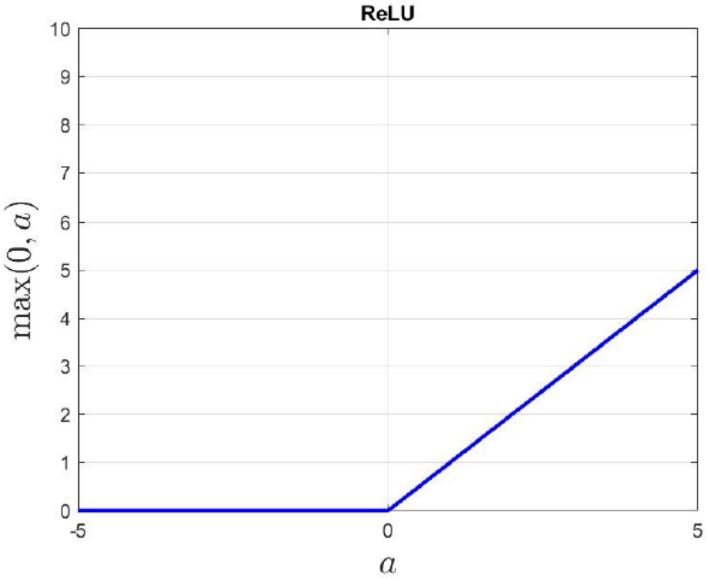


Activation Functions

ReLU Activation Function

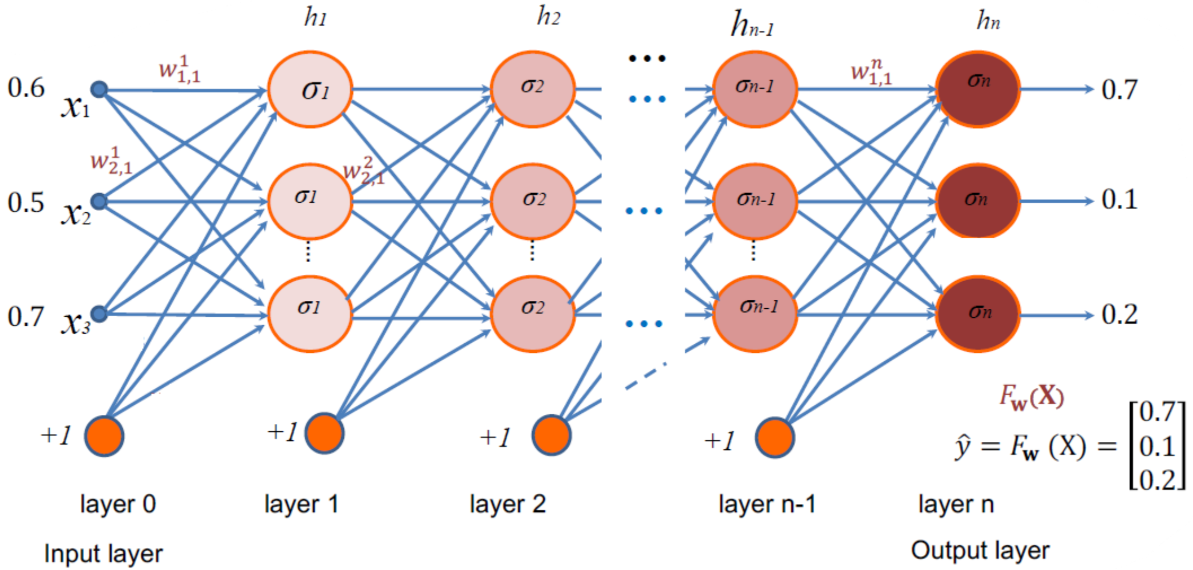
$$\sigma(a) = \max(0, a)$$

Rectified Linear Unit (ReLU)



Goal of Neural Network Training: to Learn W

$$X = \begin{bmatrix} 0.6 \\ 0.5 \\ 0.7 \end{bmatrix}$$

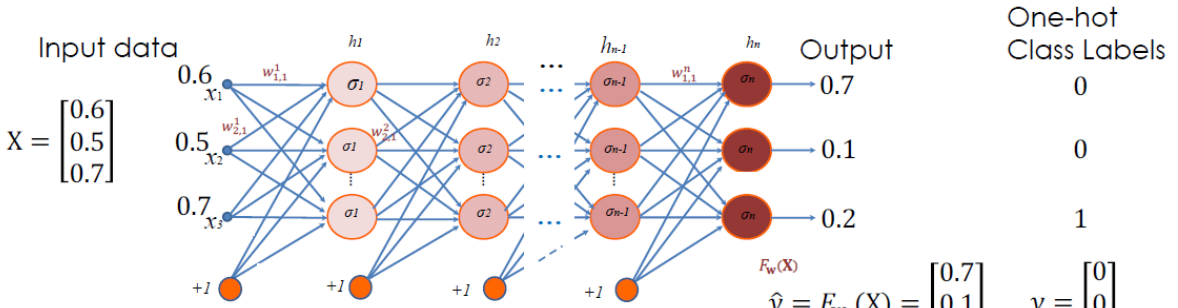


Specifically, W is learned through

1. Random initialization
2. **Backpropagation**

Neural Network Training: Backpropagation

Assume we train a NN for 3-class classification



$$\hat{y} = F_{\mathbf{w}}(\mathrm{X}) = \begin{bmatrix}0.7\\0.1\\0.2\end{bmatrix} \qquad y = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

1. Forward: (weights are fixed)
To compute network responses
To compute the errors at each output

2. Backward: (weights are updated)
To pass back the error from the output to the hidden layers
To update all weights to optimize the network

A loss function for a single sample:

$$\min_{\mathbf{w}} \sum_{i=1}^{C} (\hat{y}_i - y_i)^2$$

or

$$\min_{\mathbf{w}} ||\hat{y} - y||^2$$

Update W!

Neural Network Training: Backpropagation

- Recall that the parameters W are randomly initialized.
- We use **Backpropagation** to update W.
- In essence, **Backpropagation** is gradient descent!
- Assume we have N samples, each sample denoted by X^j and the output of NN by $\hat{y}^j$, loss function is then

$$J = \sum_{j=1}^{N} \left\| \hat{y}^j - y^j \right\|^2, \quad \min_{\mathbf{w}} J$$

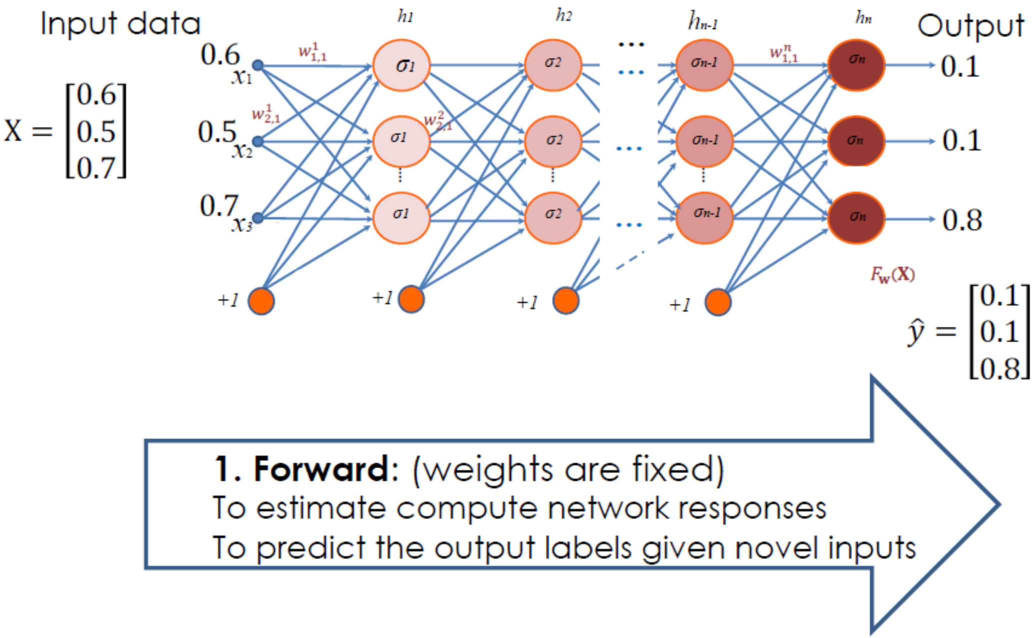
Recall gradient descent in Lec 8: $\mathrm{w} \leftarrow \mathrm{w} - \eta \nabla_{\mathrm{w}} J$

- We would therefore like to compute $\nabla_{\mathrm{w}} J$!
  - J is a function of $\hat{y}$, and $\hat{y}$ is a function of $\mathbf{w}$, i.e., $\hat{y} = F_{\mathbf{w}}(\mathrm{X})$
  - Use gradient descent and chain rule!

Being aware of the basic concept is sufficient for exam. No calculation needed.

Neural Network Testing

Once all network is trained and parameters are updated, we may conduct testing.



Example: One Neuron

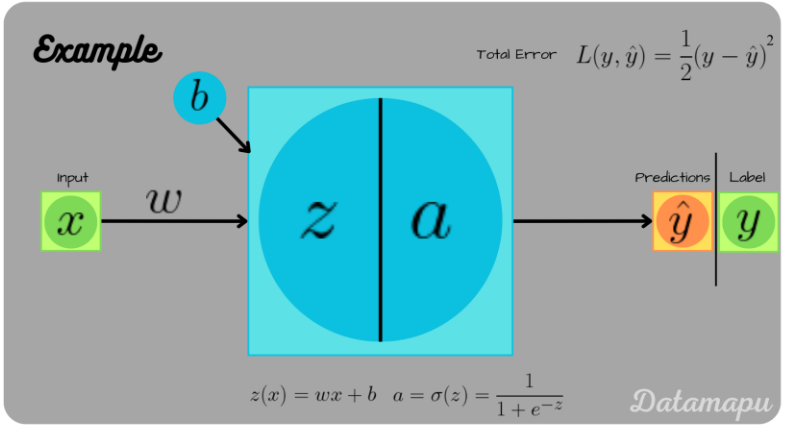


Illustration of a Neural Network consisting of a single Neuron.

https://datamapu.com/posts/deep_learning/backpropagation/

Training Data

We consider the most simple situation with one-dimensional input data and just one sample $x = 0.5$ and labels $y =$ 1.5

Activation Function

As activation function, we use the *Sigmoid function*

$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

Loss Function

As loss function, we use the *Sum of the Squared Error*, defined as

$$L(y,\hat{y}) = \frac{1}{2}\sum_{p=1}^{n}(y_p - \hat{y}_p)^2,$$

with $y_i = (y_1, \dots, y_n)$ the labels and $\hat{y} = (\hat{y}_1, \dots, \hat{y}_n)$ the predicted labels, and n the number of samples. In the examples considered in this post, we are only considering one-dimensional data, which means $n = 1$ and the formula simplifies to

$$L(y,\hat{y}) = \frac{1}{2}(y-\hat{y})^2.$$

Example: One Neuron

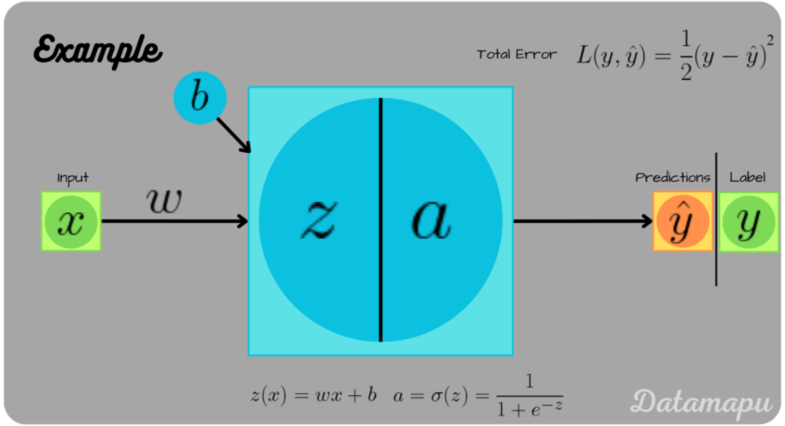


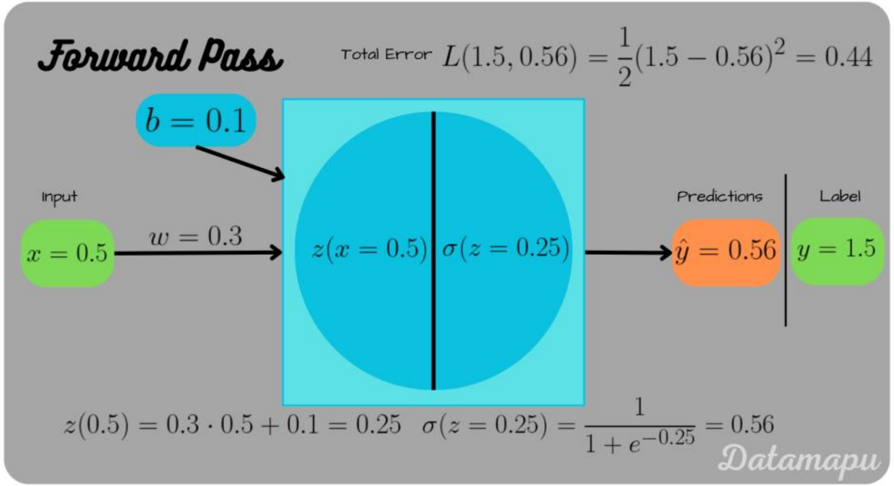
Illustration of a Neural Network consisting of a single Neuron.

To illustrate how backpropagation works, we start with the most simple neural network, which only consists of one single neuron.

In this simple neural net, $z(x) = w \cdot x + b$ represents the linear part of the neuron and a the activation function, which we chose to be the sigmoid function, i.e. $a = \sigma(z) = \frac{1}{1+e^{-z}}$. For the following calculations, we assume the initial weight $w = 0.3$ and the initial bias $b = 0.1$. Further, the learning rate is set to $\alpha = 0.1$. These values are chosen arbitrarily for illustration purposes.

https://datamapu.com/posts/deep_learning/backpropagation/

Example: One Neuron



The Forward Pass

We can calculate the forward pass through this network as

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

$$\hat{y} = \sigma(wx + b),$$

$$\hat{y} = \frac{1}{1 + e^{-(wx+b)}}$$

.

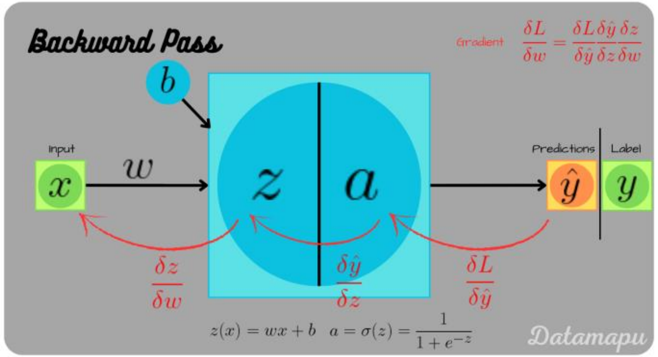
Using the weight, and bias defined above, we get for $x = 0.5$

$$\hat{y} = \frac{1}{1 + e^{-(0.3 \cdot 0.5 + 0.1)}} = \frac{1}{1 + e^{-0.25}} \approx 0.56$$

The error after this forward pass can be calculated as

$$L(1.5, 0.56) = \frac{1}{2}(1.5 - 0.56)^2 = 0.44.$$

Example: One Neuron



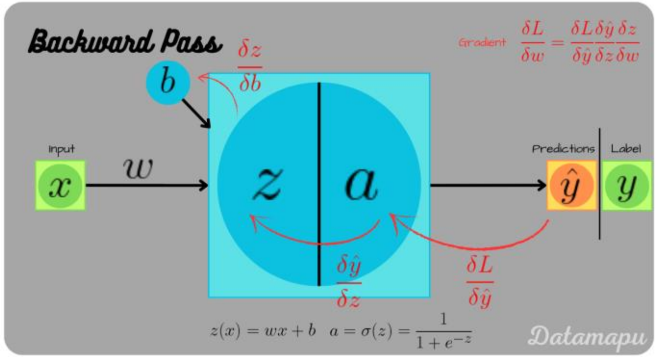


Illustration of backpropagation in a neural network consisting of a single neuron.

The Backward Pass

To update the weight and the bias we use Gradient Descent, that is

$$w_{new} = w - \alpha \frac{\delta L}{\delta w}$$

$$b_{new} = b - \alpha \frac{\delta L}{\delta b},$$

with $\alpha = 0.1$ the learning rate. That is we need to calculate the partial derivatives of L with respect to w and b to get the new weight and bias. This can be done using the chain rule and is illustrated in the plots below.

$$\frac{\delta L}{\delta w} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z} \frac{\delta z}{\delta w}$$

$$\frac{\delta L}{\delta b} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z} \frac{\delta z}{\delta b}$$

Tutorial 8, Question 4

$$\sigma\prime(x) = \sigma(x) \cdot (1 - \sigma(x)).$$

Example: One Neuron

$$\frac{\delta L}{\delta w} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z} \frac{\delta z}{\delta w}$$

$$\frac{\delta L}{\delta b} = \frac{\delta L}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta z} \frac{\delta z}{\delta b}$$

We can calculte the individual derivatives as

$$\frac{\delta L}{\delta \hat{y}} = \frac{\delta}{\delta \hat{y}} \frac{1}{2}(y - \hat{y})^2 = -(y - \hat{y}),$$

$$\frac{\delta \hat{y}}{\delta z} = \frac{\delta}{\delta z}\sigma(z) = \sigma(z) \cdot (1 - \sigma(z)),$$

$$\frac{\delta z}{\delta w} = \frac{\delta}{\delta w}(w \cdot x + b) = x,$$

$$\frac{\delta z}{\delta b} = \frac{\delta}{\delta b}(w \cdot x + b) = 1.$$

For the data we are considering, we get for the first equation

$$\frac{\delta L}{\delta \hat{y}} = -(y - \hat{y}) = -(1.5 - 0.56) = -0.94.$$

The second equation leads to

$$\frac{\delta \hat{y}}{\delta z} = \sigma(z) \cdot (1 - \sigma(z))$$

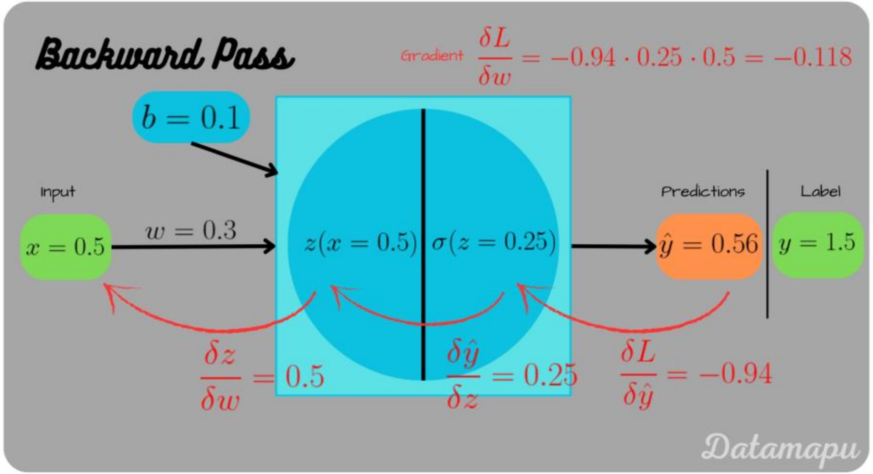
$$\frac{\delta \hat{y}}{\delta z} = \frac{1}{1 + e^{-0.25}} \left(1 - \frac{1}{1 + e^{-0.25}}\right) = 0.56 \cdot 0.44 = 0.25,$$

and finally

$$\frac{\delta z}{\delta w} = x = 0.5,$$

$$\frac{\delta z}{\delta b} = 1.$$

Example: One Neuron



Backpropagation for the weight w.

Putting the equations back together, we get

$$\frac{\delta L}{\delta w} = -0.94 \cdot 0.25 \cdot 0.5 = -0.118$$

$$\frac{\delta L}{\delta b} = -0.94 \cdot 0.25 \cdot 1 = -0.235$$

The calculation for $\frac{\delta L}{\delta w}$ is illustrated in the plot below.

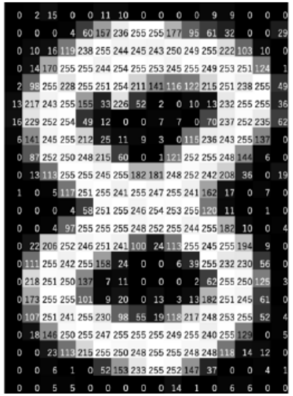
The weight and the bias then update to

$$w_{new} = 0.3 - 0.1 \cdot (-0.118) = 0.312,$$

$$b_{new} = 0.1 - 0.1 \cdot (-0.235) = 0.125.$$

Convolutional Neural Network (CNN)

- A convolutional neural network (CNN) is a special type of neural network that significantly reduces the number of parameters in a deep neural network.
- Very popular in image-related applications
- Each image is stored as a matrix in a computer

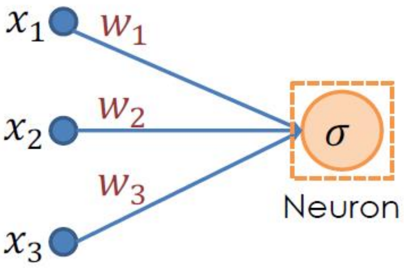


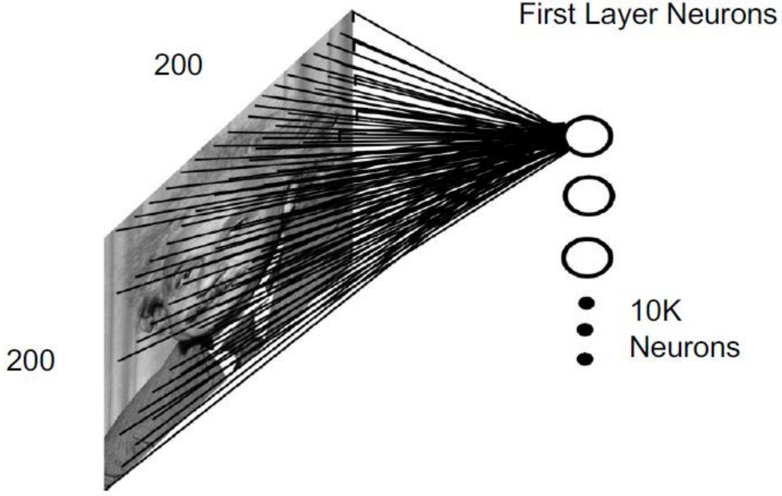
0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

https://medium.com/lifeandtech/convert-csv-file-to-images-309b6fdb8c49

Convolutional Neural Network (CNN)

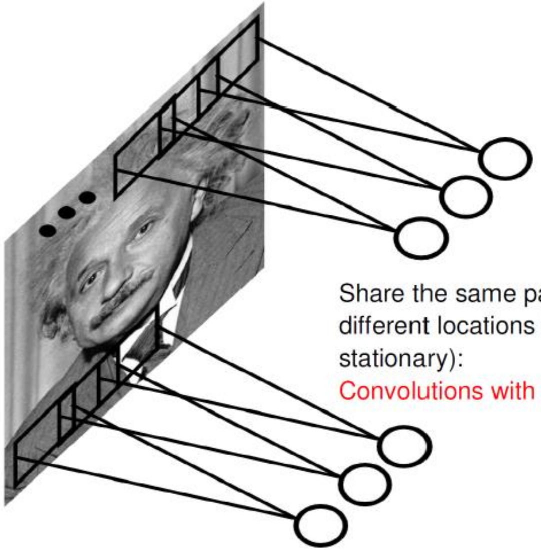
- If we model all matrix entries as inputs all at once
  - Assume we have an image/matrix size of 200x200
  - Assume we have 10K neuros in the first layer
  - We already have 200x200x10K=400 Million parameters to learn!





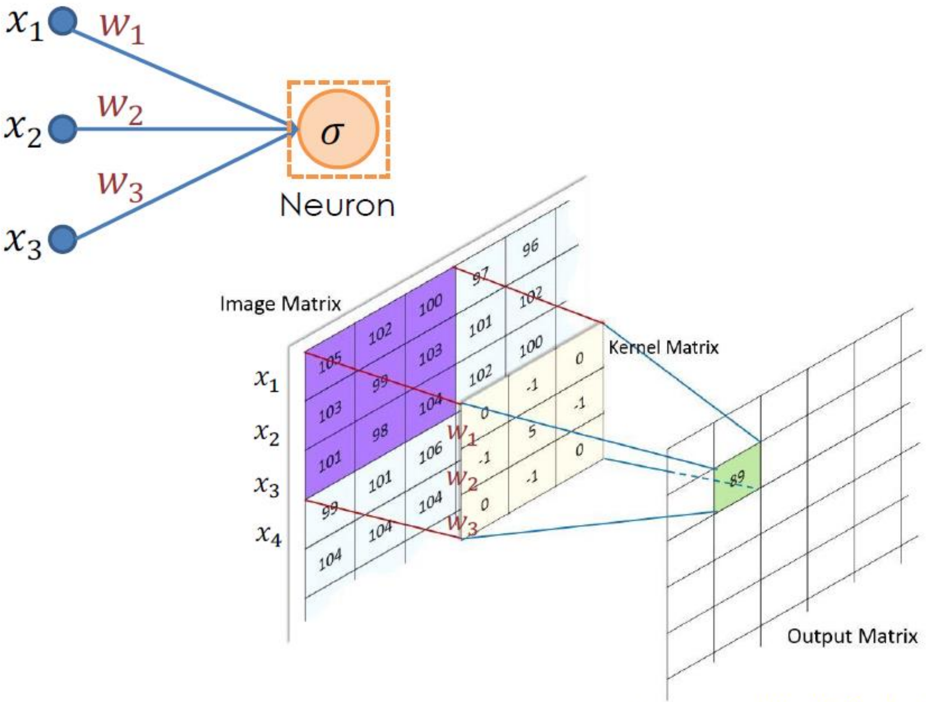
Convolutional Neural Network (CNN)

- Hence, we introduce CNN to reduce the number of parameters.
- Works in **a sliding-window manner**!



Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Convolutional Neural Network (CNN)



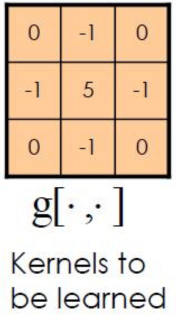
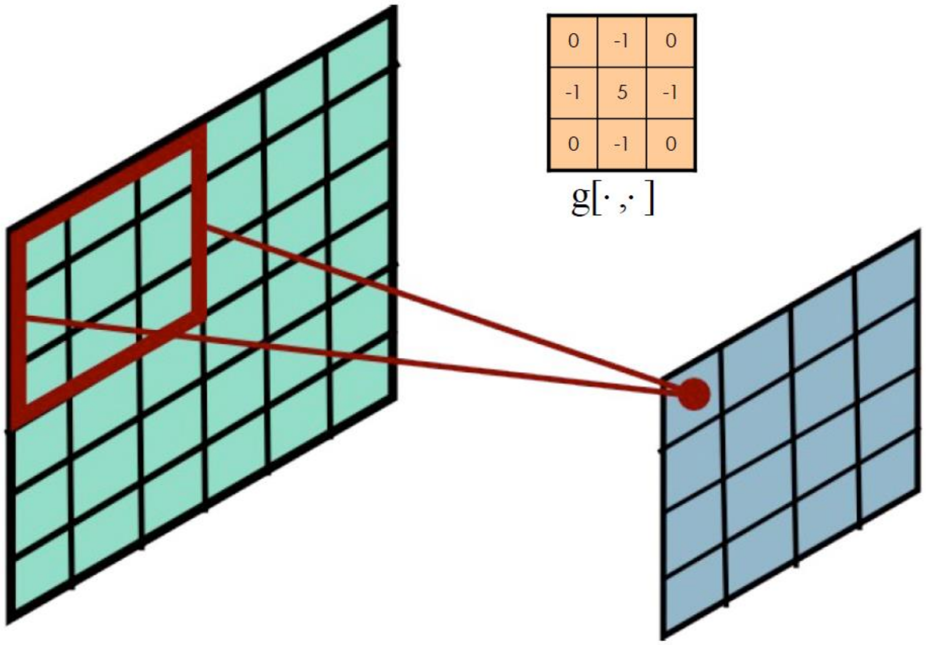
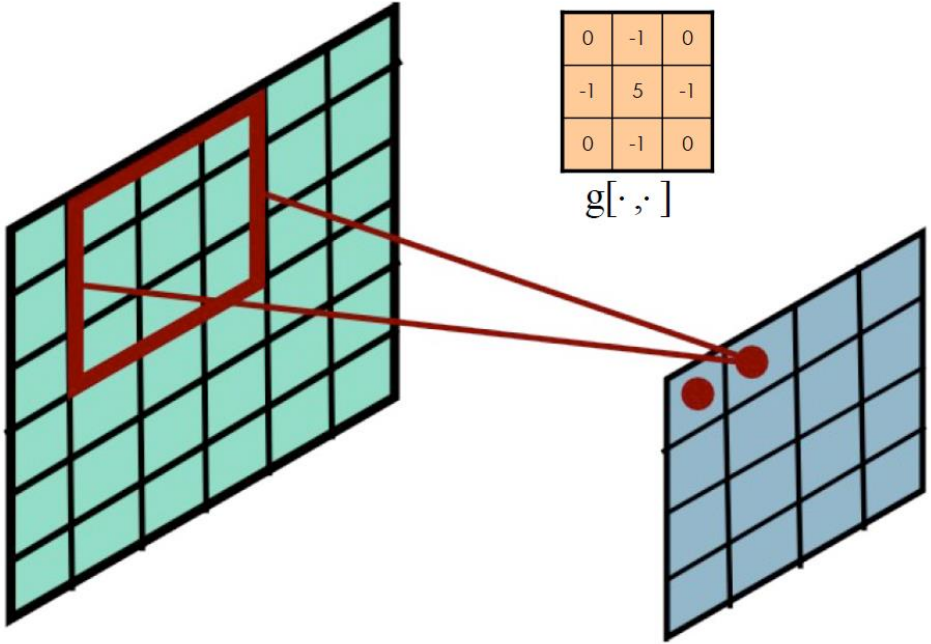


Image source: https://brilliant.org/wiki/convolutional-neural-network/

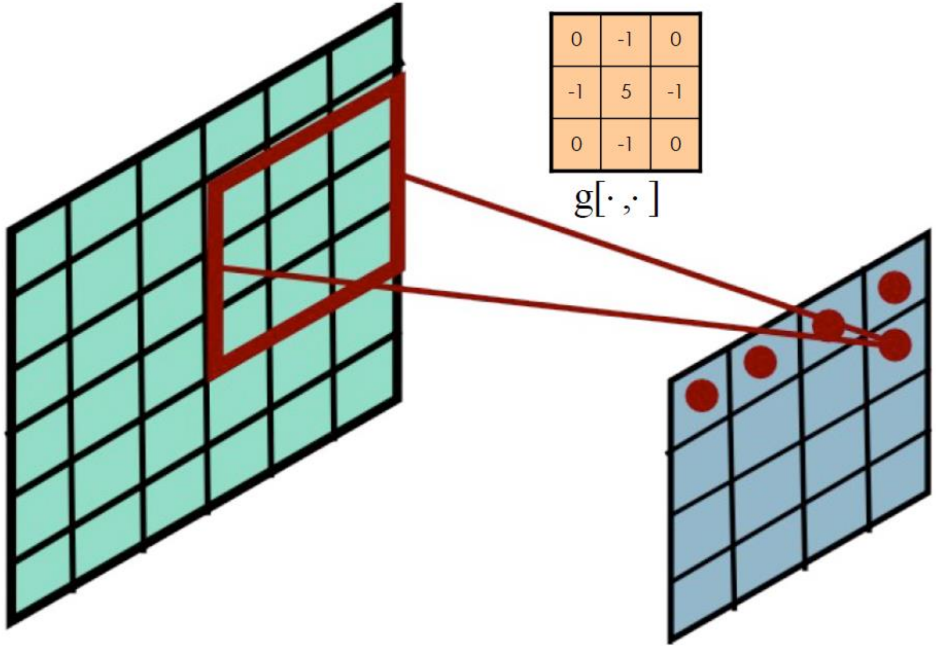
Convolutional Neural Network (CNN)



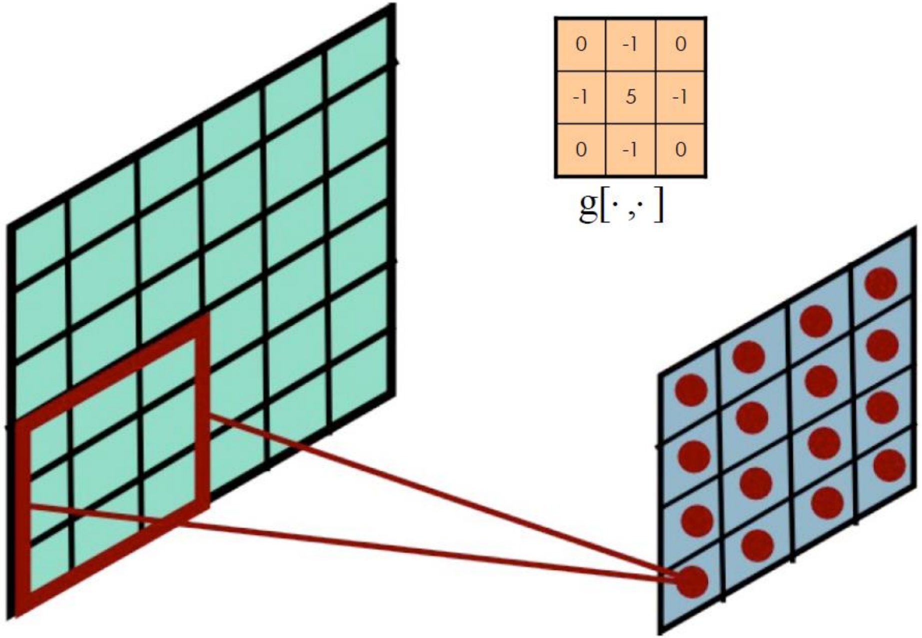
Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)

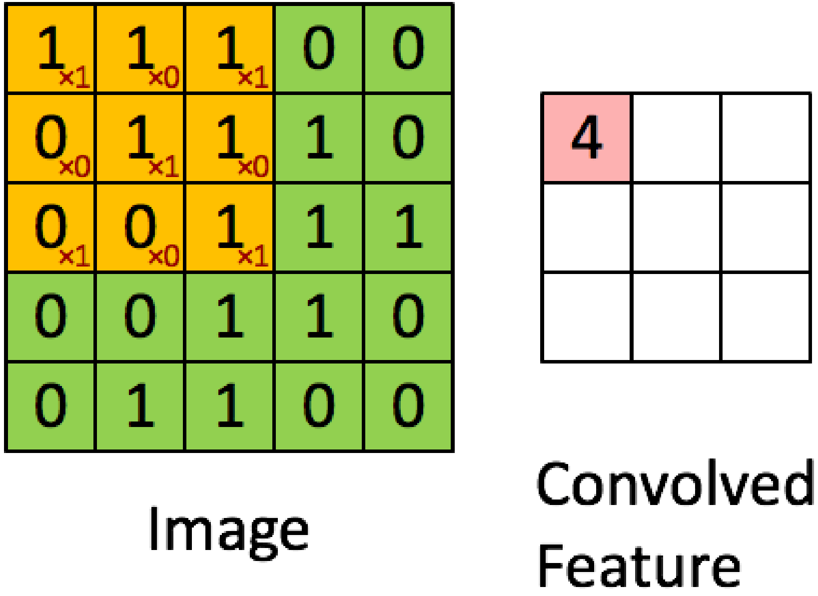


Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)

We take a filter/kernel(3×3 matrix) and apply it to the input image to get the convolved feature. This convolved feature is passed on to the next layer.



1	0	1
0	1	0
1	0	1

Filter/kernel

https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/

Summary

- Introduction to Neural Networks
  - Multi-layer perceptron
  - Activation Functions
- • Training and Testing of Neural Networks
  - Training: Forward and Backward
  - Testing: Forward
- Convolutional Neural Networks

THANK YOU