Lecture 2: Deeper Neural Network

Objective

In the second lecture, you will see

- How to deepen your neural network to learn more complex functions
- Several techniques to fasten your learning process
- The most popular artificial network in computer vision community - Convolutional neural network

Outline of Lecture 2

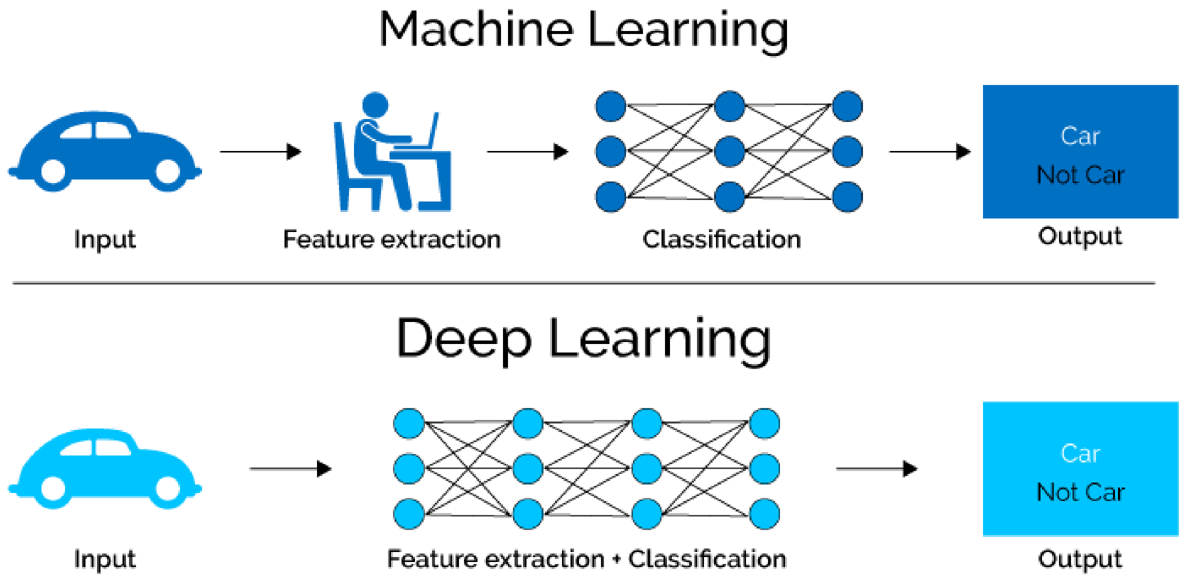
- Layers in deeper networks
- Activation functions
- Initialization
- Normalization
- Convolution
- Pooling
- Convolutional neural network

What is “Deep Learning”?

The term, Deep Learning, refers to training neural networks. Shallow neural networks do not have the enough capacity to deal with high level vision problems. Thus, people usually combine many neurons to build a deep neural network.

The more deeper you go, the more complex features are extracted.

Difference between machine learning and deep learning

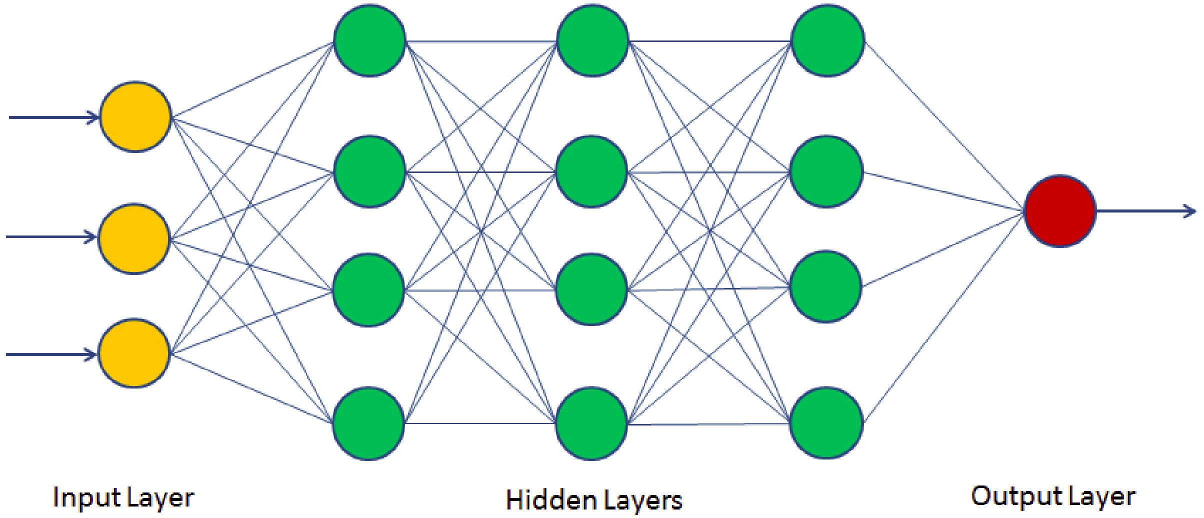


Traditional machine learning methods usually work on hand-crafted features (texture, geometry, intensity features ...).

Deep learning methods combine hand designed feature extraction and classification steps.

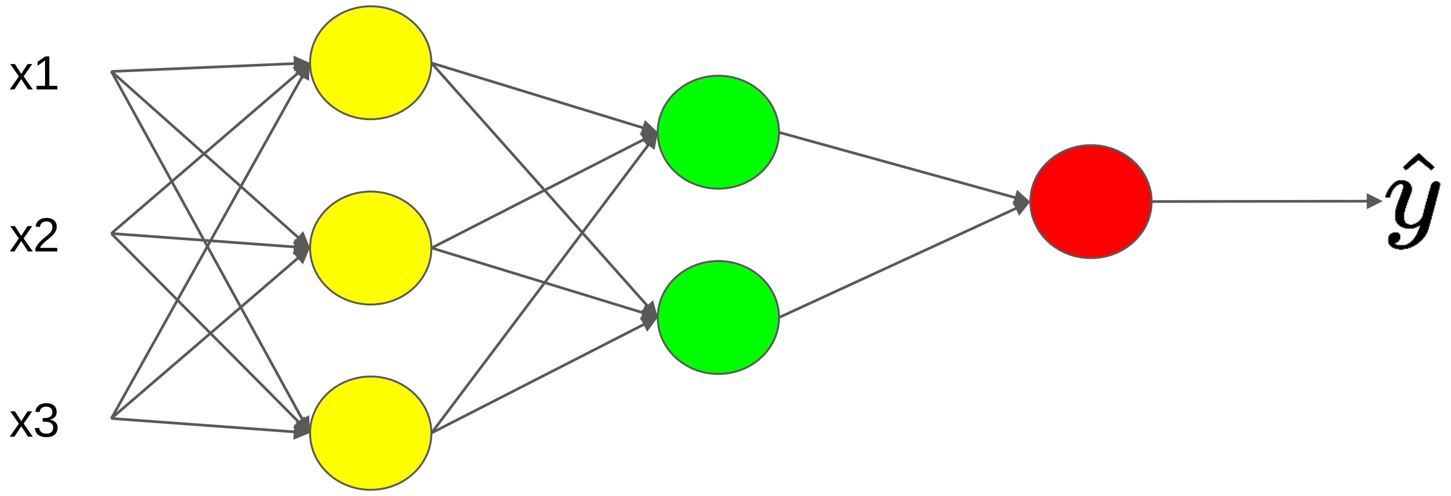
This is also called “end-to-end model”.

Deeper neural network



- Input layer: receives raw inputs and give low level features (lines, conners, ...)
- Hidden layers: transform and combine low level features into high level features (semantic concepts)
- Output layer: use the high level features to predict results

Deeper neural network



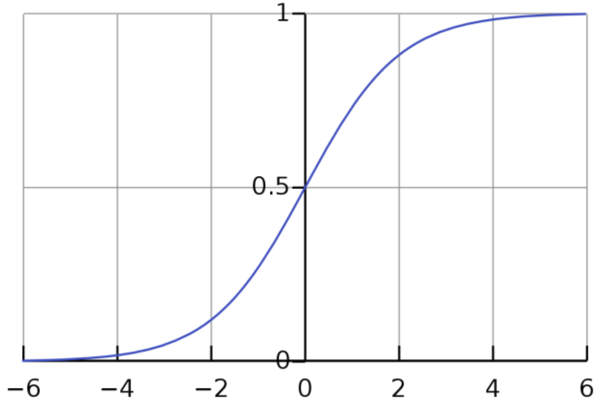
$$\boxed{\mathbf{a}^{[0]} = \sigma(\omega^{[0]}\mathbf{x} + b^{[0]})} \rightarrow \boxed{\mathbf{a}^{[1]} = \sigma(\omega^{[1]}\mathbf{a}^{[0]} + b^{[1]})} \rightarrow \boxed{\hat{\mathbf{y}} = \sigma(\omega^{[2]}\mathbf{a}^{[1]} + b^{[2]})}$$

Parameters get updated layer by layer via back-propagation.

Activation Functions

We have seen that sigmoid function can be used as activation function.

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



In the practice, sigmoid function is always only used in the output layer to transform the output into probability range 0~1.

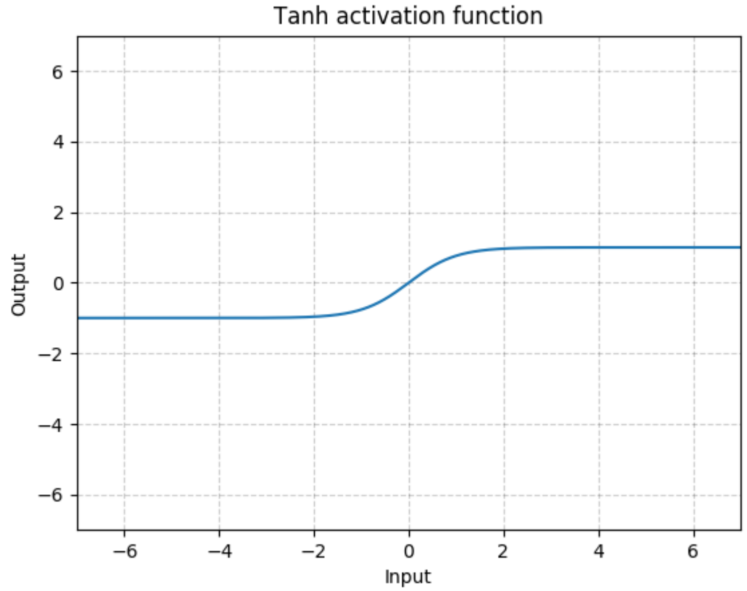
Activation Functions

There are other well used activation function.

1. Tanh

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

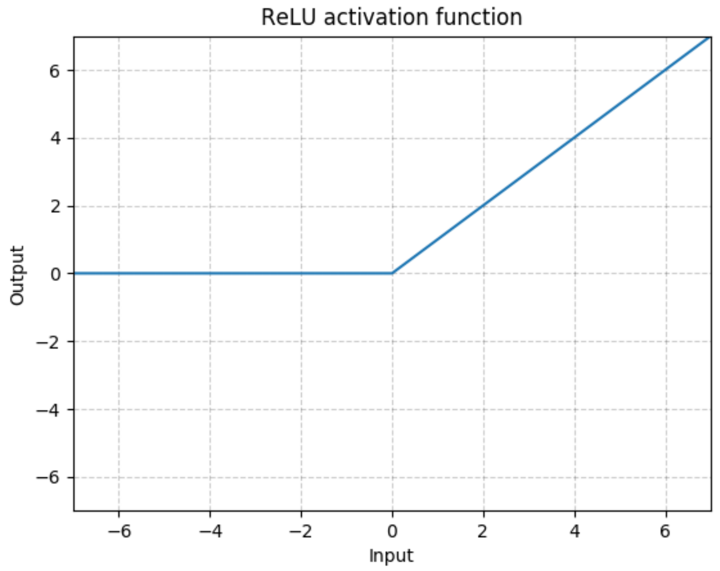
Range from -1 to 1



Activation Functions

2. ReLU (Rectified Linear Unit)

$$ReLU(x) = max(0, x)$$

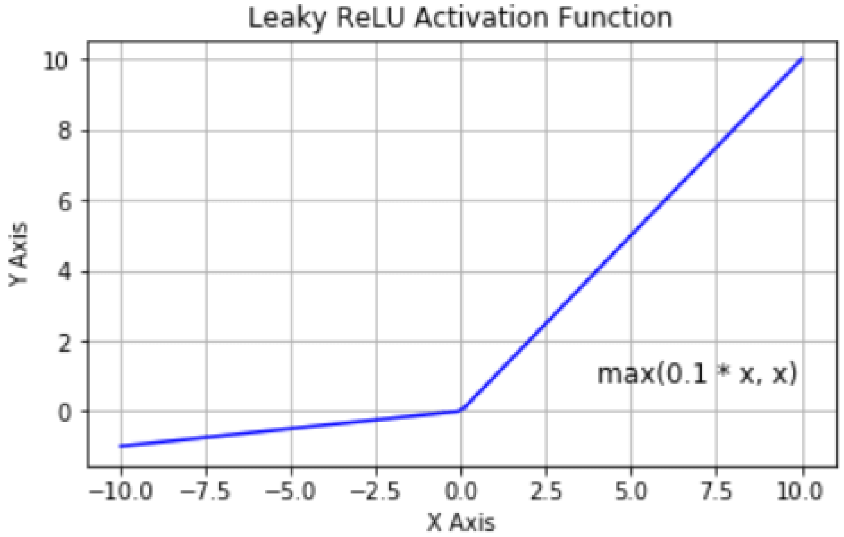


Range from 0 to infinity, which keeps high activation.

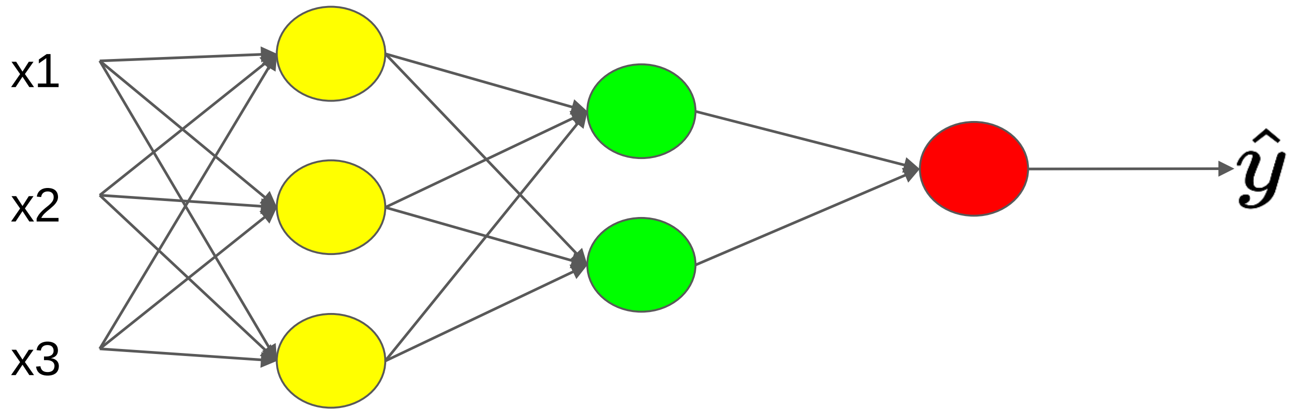
Activation Functions

3. LeakyReLU

$$LeakyReLU(x) = \begin{cases} max(0, x),\ when\ x \geq 0 \\ negative\_slope * x,\ when\ x < 0 \end{cases}$$



Parameter Initialization



If you initialize all the parameters as 0 in a neural network, it will not work, because all the neurons in a same layer will give same outputs and get updated in the same way. But we desire different neuron will get different features. One solution is random initialization.

Parameter Initialization

There are some popular initialization methods.

1. Normal initialization

   Initialize parameters with values drawn from the normal distribution $\mathcal{N}(mean, std^2)$

1. Xavier normal initialization [1]

   Initialize parameters with values sampled from $\mathcal{N}(0, std^2)$ where $std = gain * \sqrt{\frac{2}{fan\_in + fan\_out}}$

   fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor

[1]. Understanding the difficulty of training deep feedforward neural networks - Glorot, X. & Bengio, Y. (2010)

Parameter Initialization

3. Kaiming normal initialization[1]

Initialize parameters with values sampled from $\mathcal{N}(0, std^2)$ where $std = \sqrt{\frac{2}{(1+a)^2 * fan\_in}}$

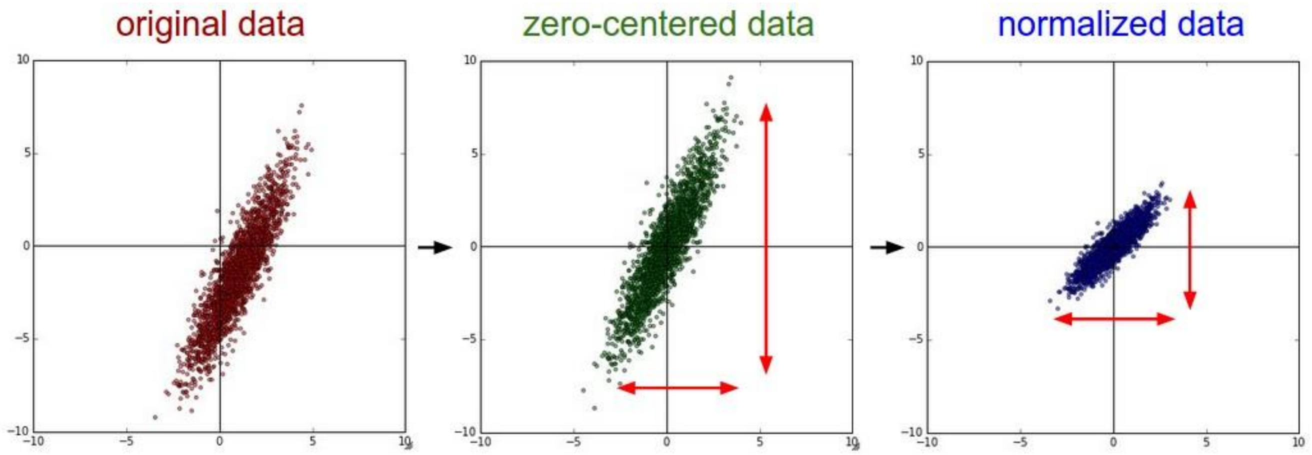
fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor

[1]. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification - He, K. et al. (2015)

Input Normalization

Before training your neural network, conducting a normalization on your inputs will speed up your training. Input normalization contains two steps:

1. Mean subtraction: $X = X - mean(X)$
2. Variance normalization: $X = \frac{X}{std^2(X)}$



Find more details in CS231 (http://cs231n.github.io/neural-networks-2/)

Batch Normalization

During the training, you can also do a normalization on the activations. One of the most used technique is Batch Normalization [1], which conducts a normalization over channels within a mini batch.

Because it’s a differentiable operation, we usually insert the BatchNorm layer immediately after activations, and before non-linearities.

[1]. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift - Sergey Ioffe, Christian Szegedy 2015

Other types of activation normalization

1. Batch Norm
2. Layer Norm
3. Instance Norm
4. Group Norm ...

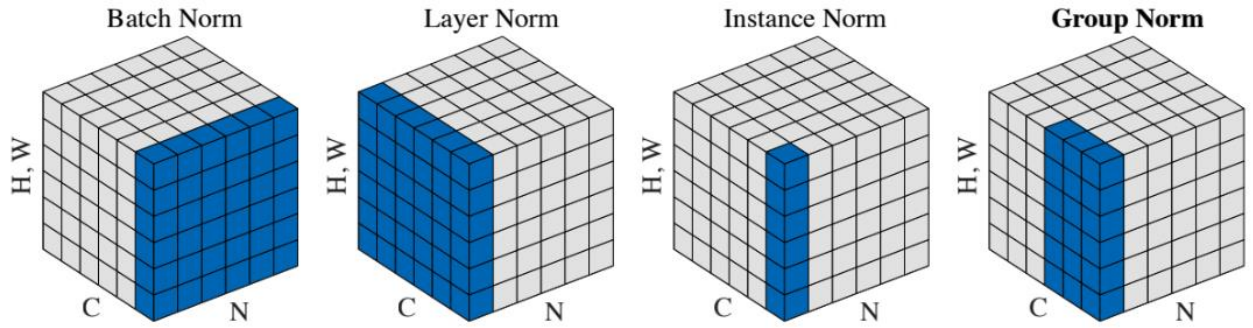
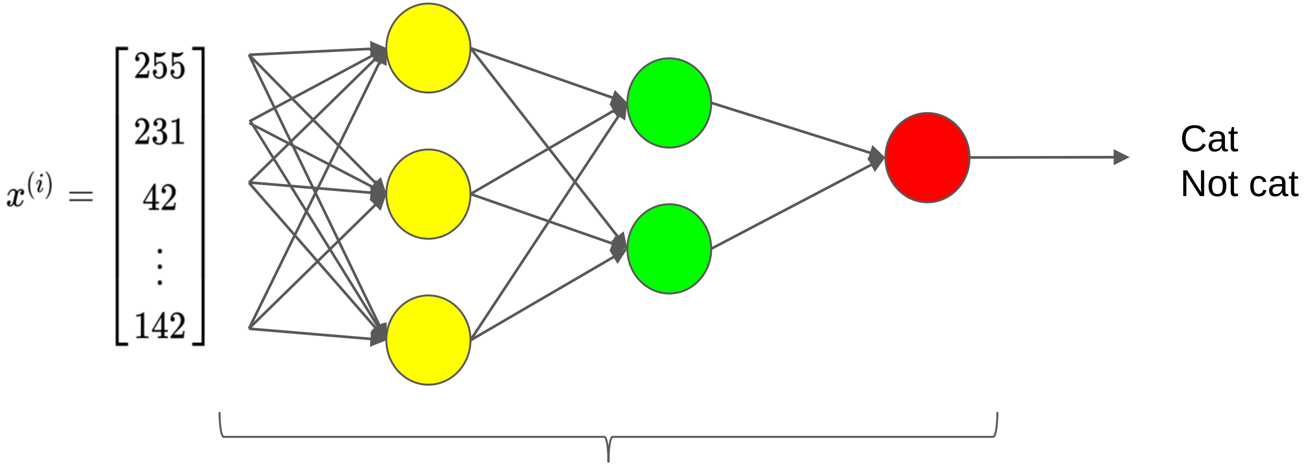


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

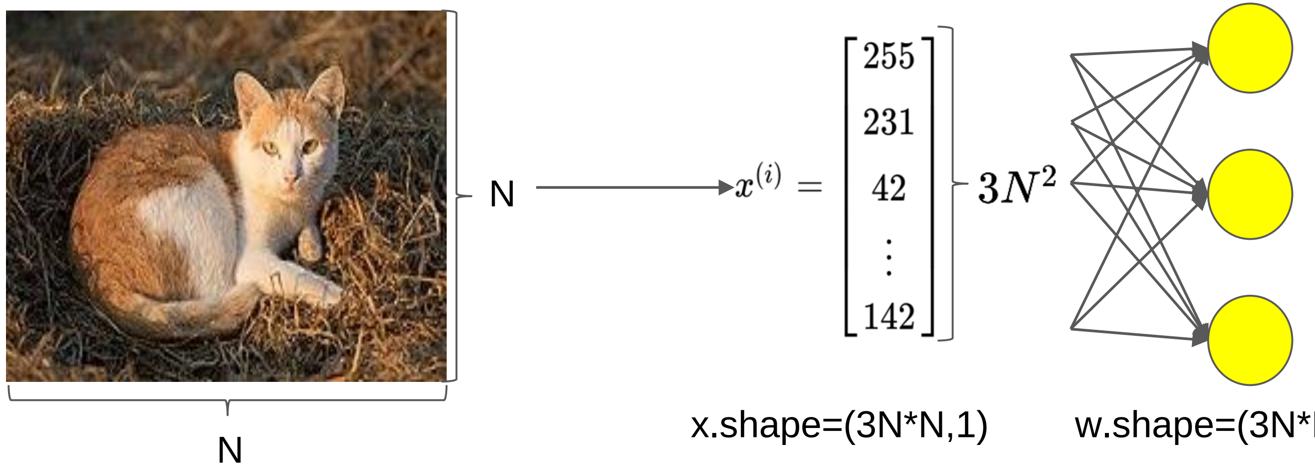
Convolutional Neural Network

Fully-connected layers


$x^{(i)} = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 142 \end{bmatrix}$
Cat
Not cat

From previous slides, we can see fully-connected (FC) layers connect every neuron in one layer to every neuron in the previous layer.

Drawback of fully-connected layer

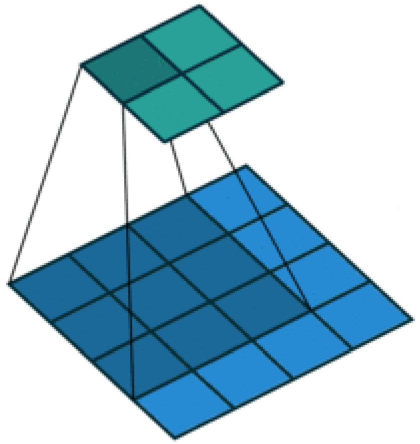

N
N
$x^{(i)} = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 142 \end{bmatrix}$
$3N^2$

x.shape=(3N*N,1)   w.shape=(3N*N,3)

- For low-quality image, e.g. N=100, w.shape=(30K,3), it’s ok.
- But for high-quality image, e.g. N=1K, w.shape=(3M,3), much more computational resources will be needed.

Convolution

Instead of connecting to every neuron in the previous layer, a neuron in the convolutional layer only connects to neurons within a small region.



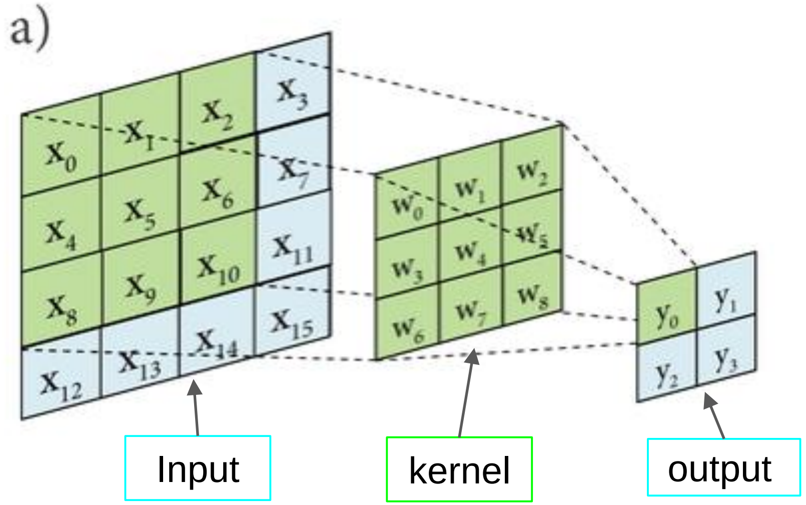
Advantages:

1. Spatial coherence is kept.
2. Lower computational complexity.

Convolution

We don’t have to flatten the input, so the spatial coherence is kept.

A kernel (also called filter) slides across the input feature map. At each location, the product between each element of the kernel and the input element is computed and summed up as the output in the current location.



3D volumes of neurons

A convolutional layer has neurons arranged in 3 dimensions:

- Height
- Width
- Depth (also called channel)

The initial depth of a RGB image is 3. For example, in CIFAR-10, images are of size 32*32*3 (32 wide, 32 high, 3 color channels).

In this case, the kernel has to be 3-dimensional. It will slide across the height, width and depth of the input feature map.

Spatial arrangement

To properly use a convolutional layer, several hyperparameters need to be set.

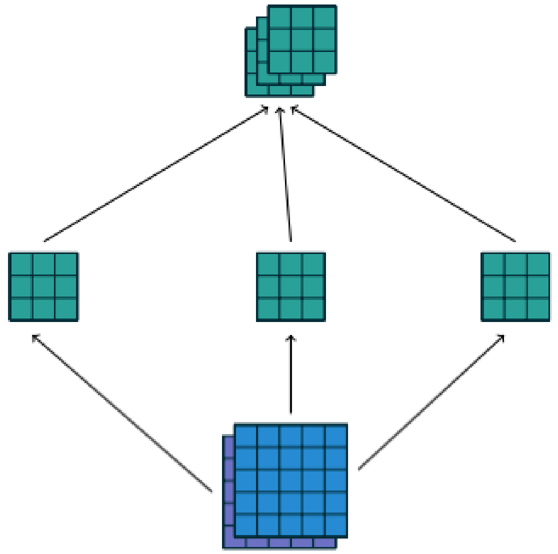
1. Output depth
2. Stride
3. Padding
4. Kernel size
5. Dilation

Output depth = Number of kernels

Previous procedure can be repeated using different kernels to form as many output feature maps as desired.

Different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color.

The final feature map will be the stack of outputs of different kernels.

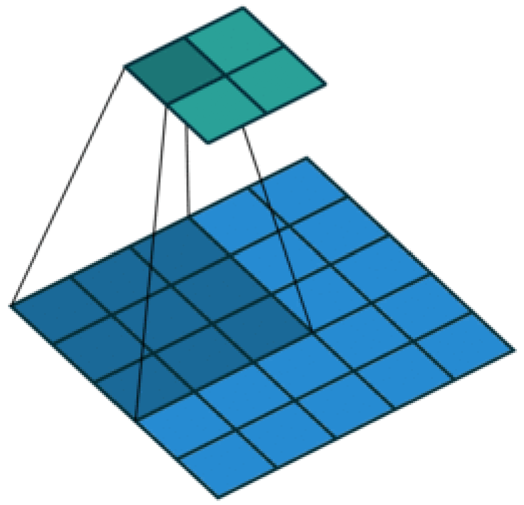


Stride

Stride is the distance between two consecutive positions of the kernel.

In the example, stride is set to 2.

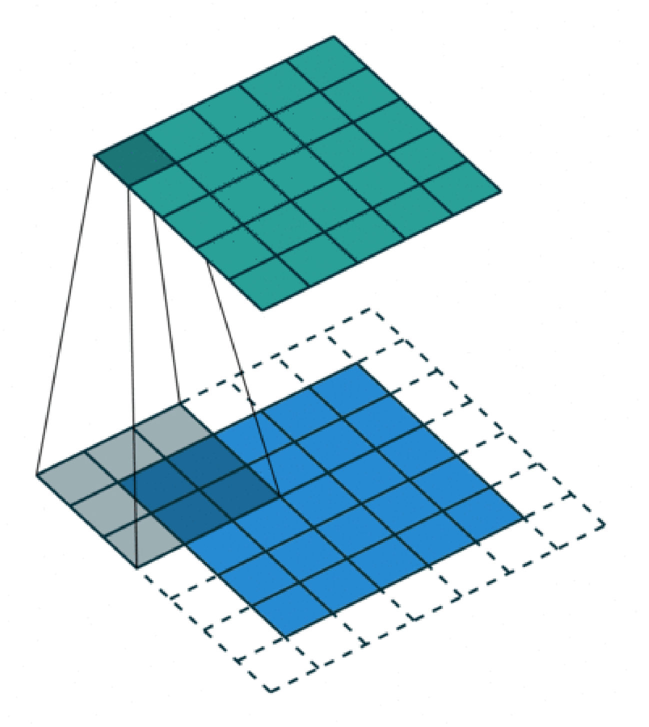
Strides constitute a form of subsampling.



Padding

The most used padding in the neural network is zero padding, which refers to number of zeros concatenated at the beginning and at the end of an axis.

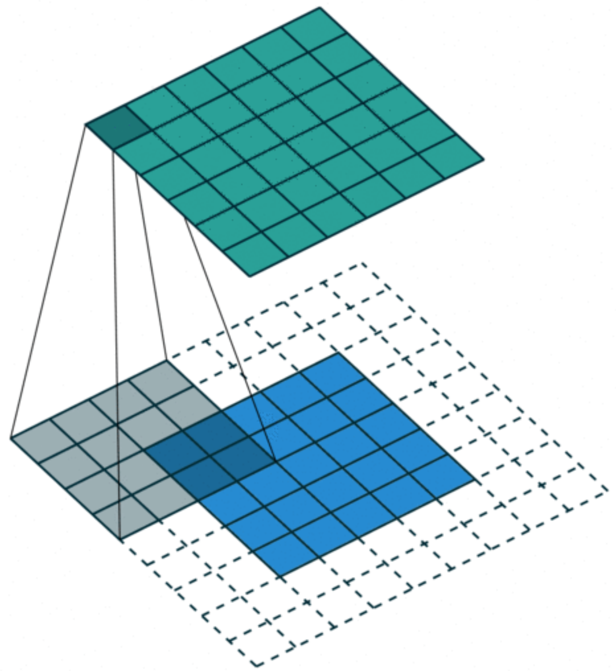
Padding helps to maintain spatial size of a feature map.



Kernel size

Size of the convolving kernel.

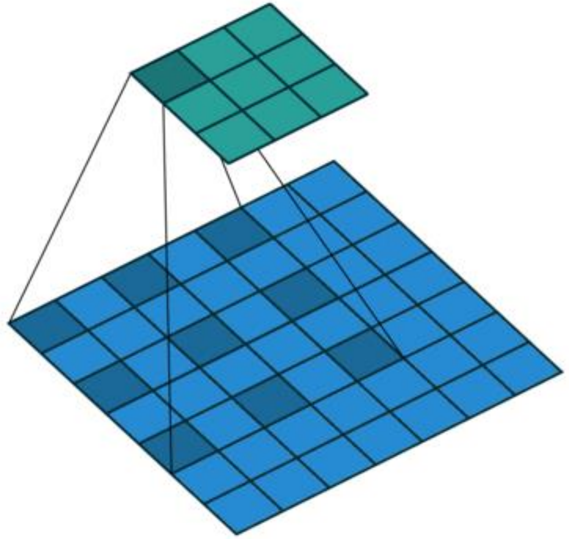
In the example, kernel size is set to 4.



Dilation

Dilated convolutions “inflate” the kernel by inserting spaces between the kernel elements.

The dilation hyperparameter defines the space between kernel elements.



Spatial arrangement

After setting previous hyperparameters, we will have the output dimensions:

$$H_{out} = \frac{H_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1$$

$$W_{out} = \frac{W_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1$$

Receptive field

The receptive field in Convolutional Neural Networks (CNN) is the region of the input space that affects a particular unit of the network.

Previous hyperparameters can directly affect the receptive field in one convolutional layer. When you stack more convolutional layers into a deep convolutional neural network, the receptive field in last layers becomes bigger.

Pooling

Besides convolutional layer, pooling operation also plays an important role in neural networks.

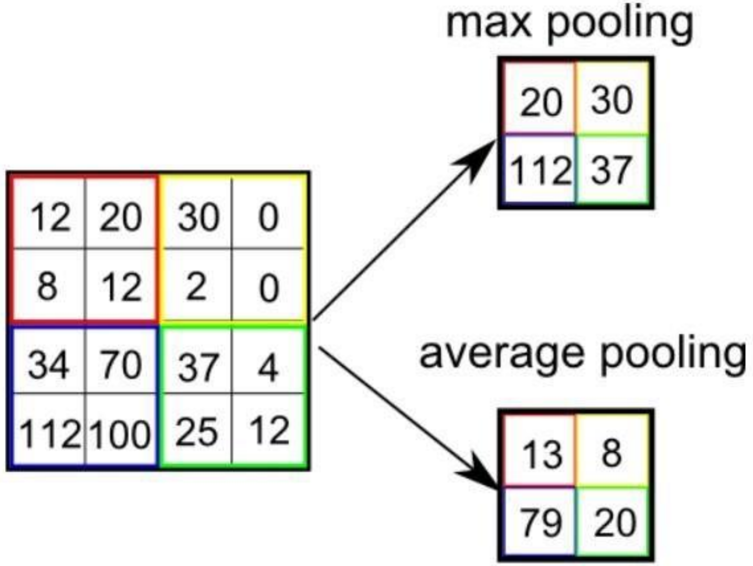
Pooling operations reduce the size of feature maps by using some function to summarize subregions, such as taking the average or the maximum value.

Pooling

Pooling shares some same hyperparameters as convolution.

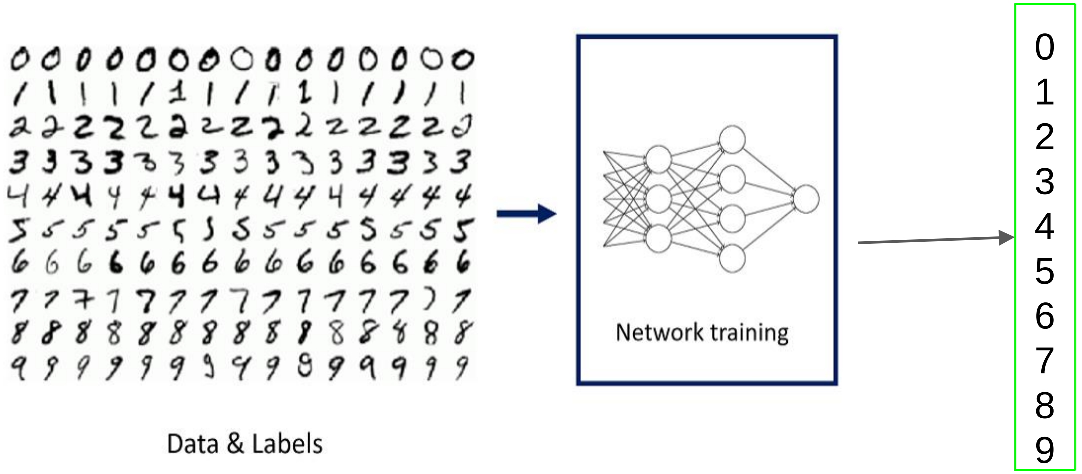
1. Stride
2. Padding
3. Kernel size
4. Dilation

An example of average/max pooling, where stride=2, kernel_size=2.



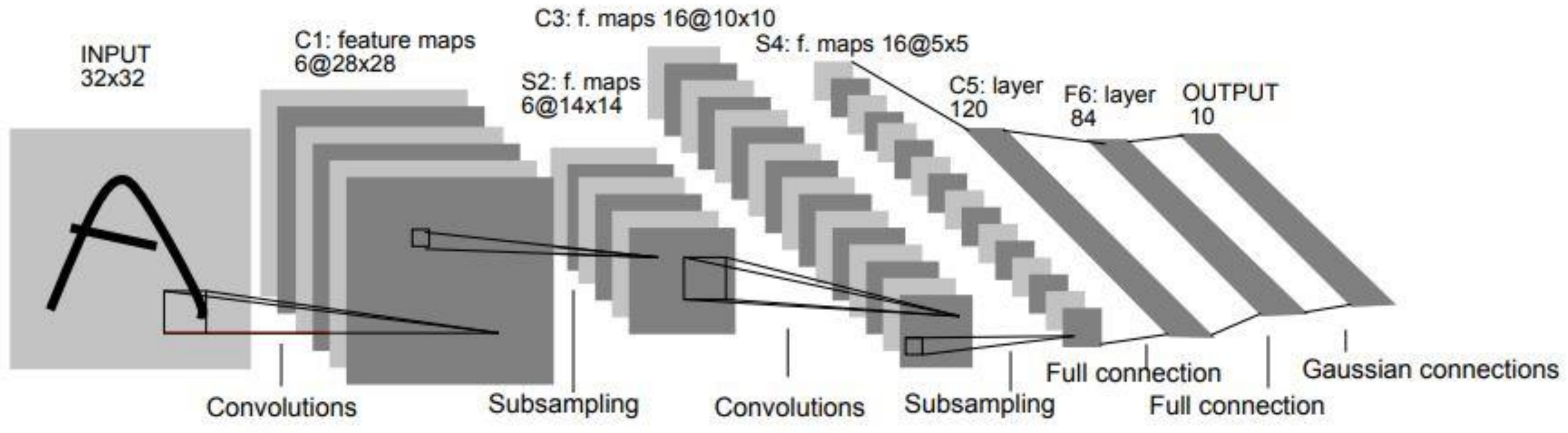
CNN example

LeNet-5 [1] is proposed by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner in 1990’s for handwritten and machine-printed character recognition.



[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.

LeNet-5

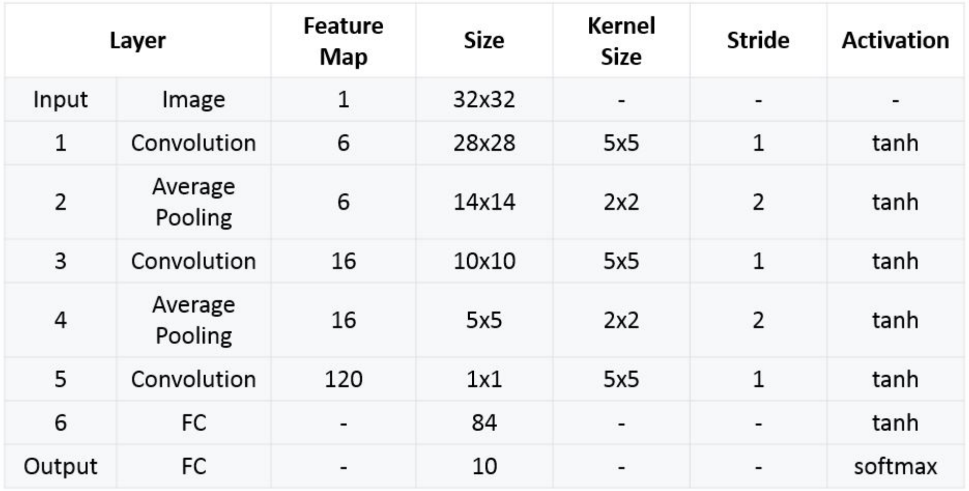


In LeNet-5, subsampling operation corresponds to an average pooling. Basically, LeNet-5 is a combination of convolution, pooling and fully-connected layers.

LeNet-5

Summary of LeNet-5 Architecture

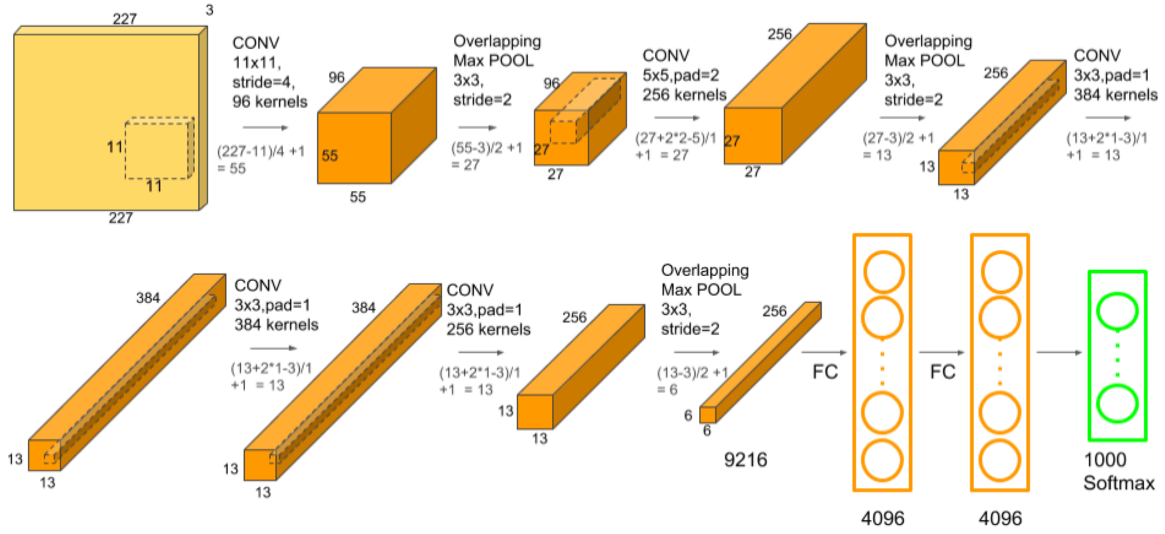
The original gaussian connection is defined as fc layer + MSE loss. Softmax has mostly replaced it nowadays.



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

AlexNet

AlexNet is the name of a deep CNN for large scale image classification, designed by Alex Krizhevsky et al. in 2012.

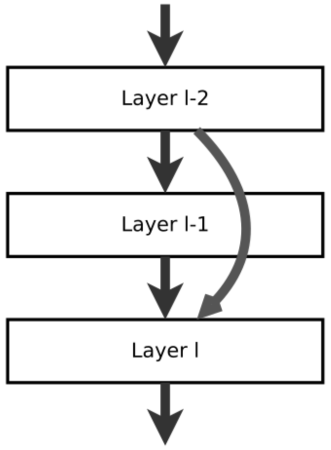


https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

ResNet

ResNet is designed by Kaiming He et al. in 2015, which is the most cited paper in all areas in Google Scholar Metrics 2020.

Authors proposed **skip connections**, or shortcuts to jump over some layers.

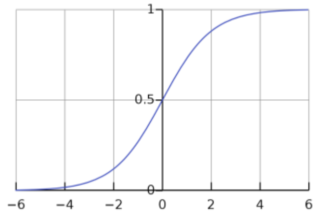


https://arxiv.org/pdf/1512.03385.pdf

Multi-class classification

- Binary classification

sigmoid



0 (not cat)
1 (cat)

- Multi-class classification

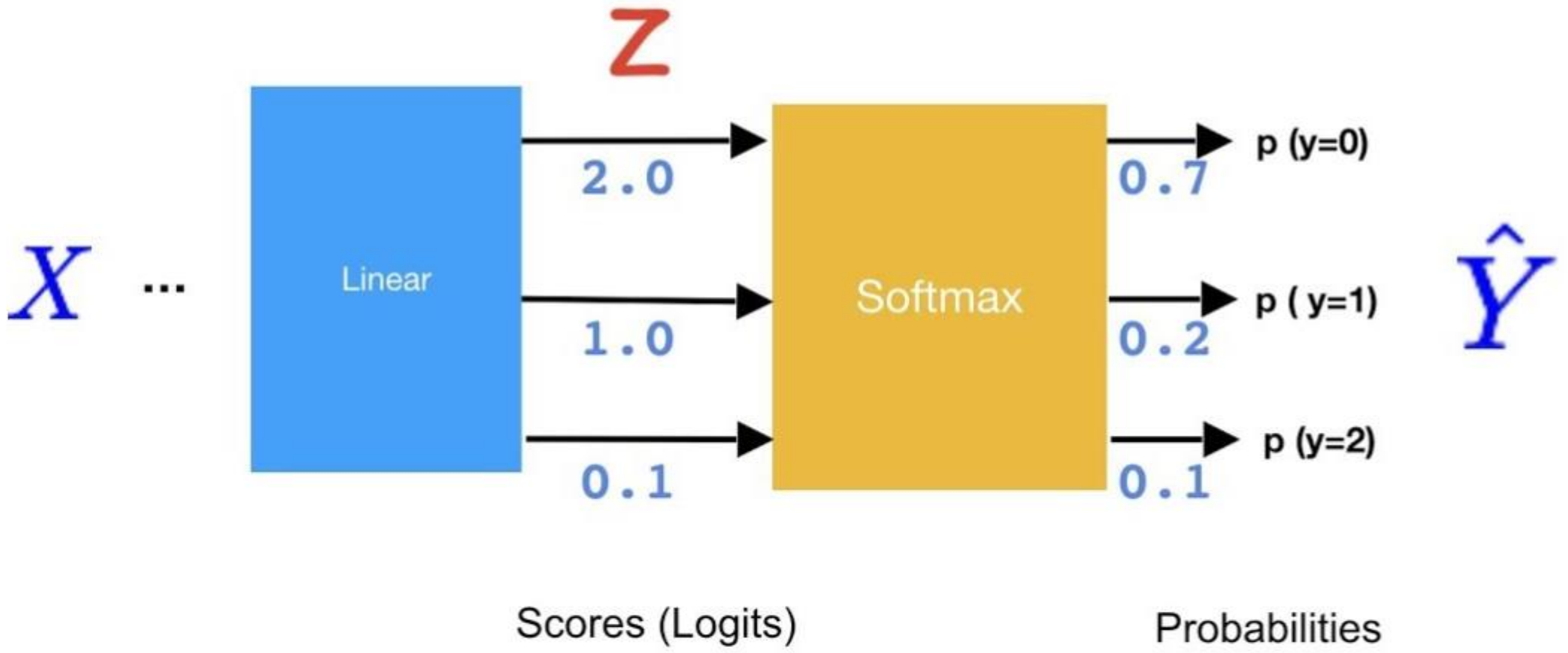
Data & Labels

Sigmoid is not suitable for multi-class.

0
1
2
3
4
5
6
7
8
9

Meet Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



Conclusion

1. We usually need high level features extracted by deep neural networks to solve complex computer vision tasks.
2. Parameter initialization and normalization are helpful to fasten your training.
3. Convolutions are able to extract local features from a patch and keep spatial correspondence, which are more suitable to vision tasks.

Practice-2

Digit recognition with LeNet-5.

In this practice, you don’t have to worry about gradients for back-propagation. You will need to

1. get familiar with Pytorch
2. implement LeNet-5 under pytorch
3. train your network on MNIST dataset