SAMSUNG

Together for Tomorrow!
Enabling People
Education for Future Generations

Samsung Innovation Campus

Artificial Intelligence Course

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 8.

Deep Learning - Part I

AI Course

Samsung Innovation Campus

Course Description (1/2)

Duration: 34 Hours.

Purpose:

- Introduction to artificial intelligence with the TensorFlow library.
- Introduction to artificial intelligence with the Keras library.
- Introduction to deep neural networks, convolutional neural networks, and recurrent neural networks.

Target Audience:

- This course is required to expand the knowledge on artificial intelligence.
- This course is a prerequisite for the capstone project.

Prerequisite:

- A student taking this course should be familiar with the NumPy and Scikit-Learn libraries.
- A student taking this course should be familiar with the linear algebra and machine learning algorithms.

Course Description (2/2)

Objectives:

- Build and train the deep neural networks.
- Build and train the convolutional neural networks for image classification.
- Build and train the recurrent neural networks for time series forecast and natural language processing.
- Train and utilize embedding models to represent text data.
- Optimize the deep learning neural networks preventing the overfitting and vanishing gradient problems.
- Gain proficiency with the deep learning libraries such as TensorFlow and Keras.
- Utilize AutoEncoders for dimensional reduction.
- Explore the latest developments on the generative adversarial networks.

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 8.

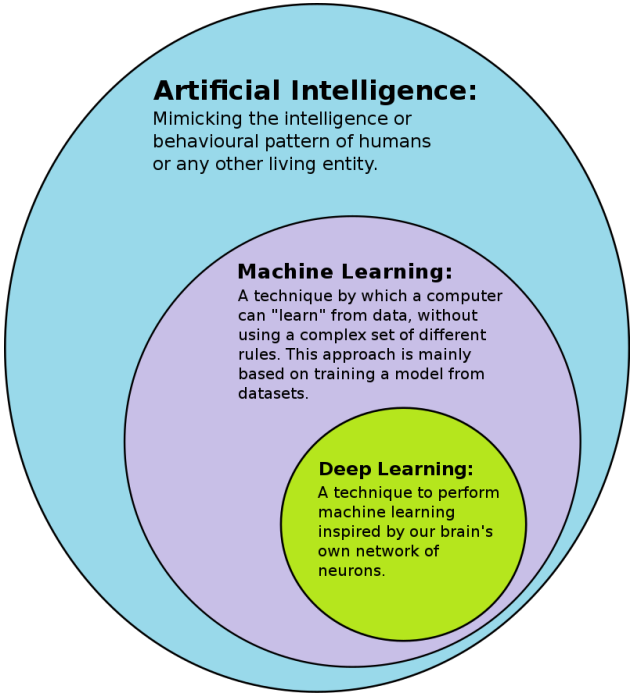
Deep Learning - Part I

UNIT 1.

Introduction to Deep Learning

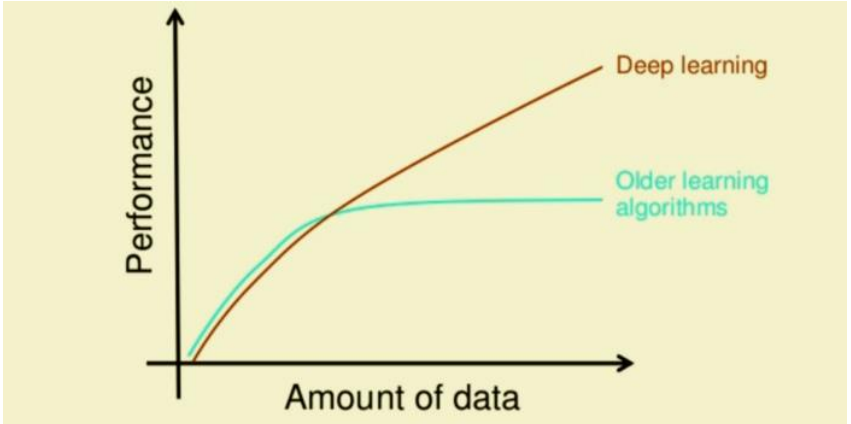
What is Deep Learning?

- Deep learning is a type of machine learning and artificial intelligence
- It imitates the way humans gain certain types of knowledge.



Why deep learning?

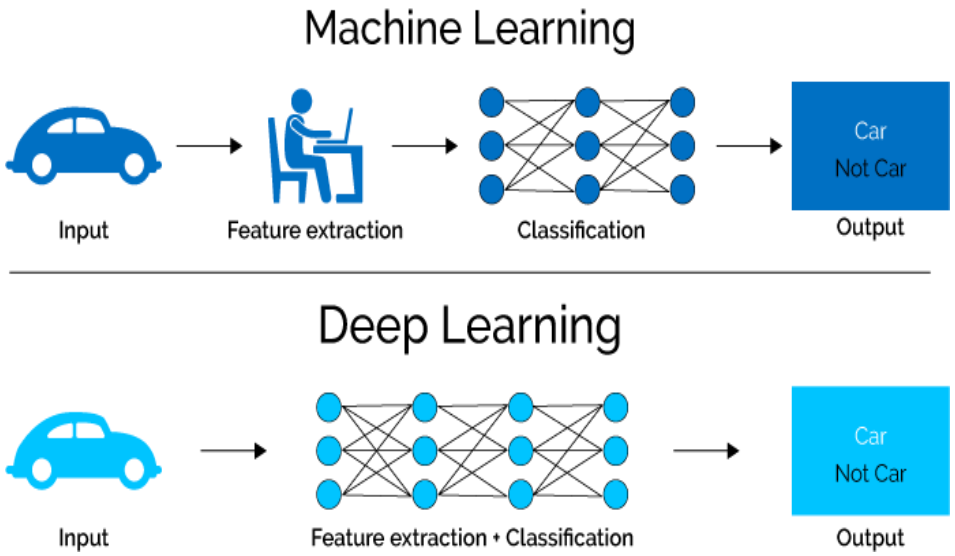
- ❑Deep learning is an important element of data science
- ❑Data science → collecting, analyzing and interpreting large amounts of data
- ❑Deep learning makes this process faster and easier.



How do data science techniques scale with amount of data?

"The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms."

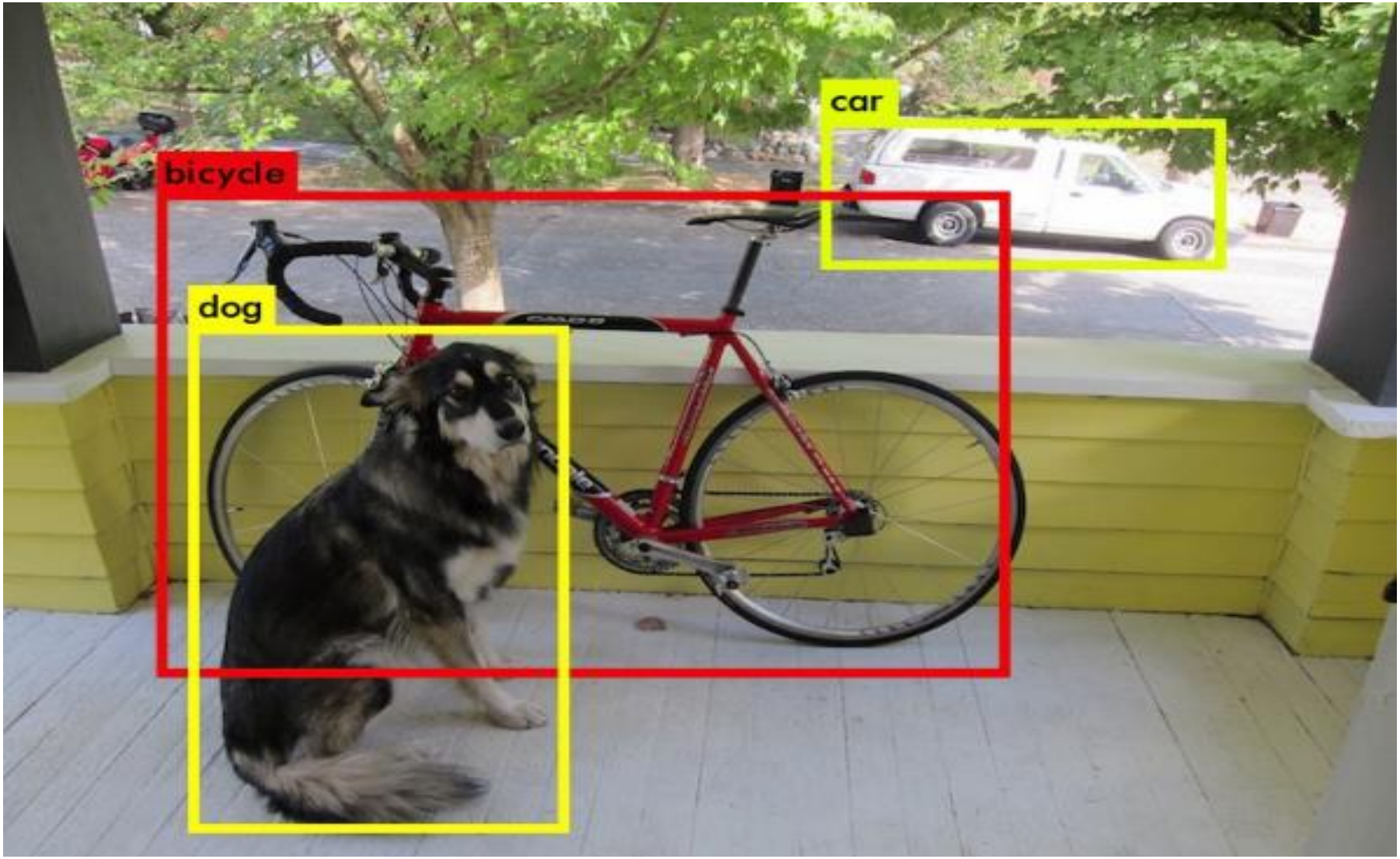
Deep Learning Vs. Machine Learning



- In traditional **Machine learning** techniques
  - Features need to be identified by an domain experts:
    - to reduce the complexity of the data
    - to make patterns more visible to learning algorithms to work.
- The biggest advantage **Deep Learning** algorithms are:
  - they try to learn high-level features from data in an incremental manner
  - This eliminates the need of domain expertise and hard core feature extraction.

Deep Learning Vs. Machine Learning

- Problem Solving approach
  - **Deep Learning** techniques tend to solve the problem end to end
  - **Machine learning** techniques need the problem statements to break down to different parts to be solved first and then their results to be combine at final stage.
- **Example**
  - **Deep Learning** techniques like **Yolo net** take the image as input and provide the location and name of objects at output.
  - In usual **Machine Learning** algorithms like **SVM**, a bounding box object detection algorithm is required first to identify all possible objects to have the **HOG** as input to the learning algorithm in order to recognize relevant objects.


car
bicycle
dog

Deep Learning Vs. Machine Learning

- Computational Time
  - Training
    - Usually, a **Deep Learning** algorithm takes a long time to train due to large number of parameters.
    - Popular **ResNet** algorithm takes about two weeks to train completely from scratch.
    - Where as, traditional **Machine Learning** algorithms take few seconds to few hours to train.
  - Testing
    - The scenario is completely reverse in testing phase. At test time, **Deep Learning** algorithm takes much less time to run.
    - Whereas, if you compare it with k-nearest neighbors (a type of machine learning algorithm), test time increases on increasing the size of data. Although this is not applicable on all machine learning algorithms, as some of them have small testing times too.

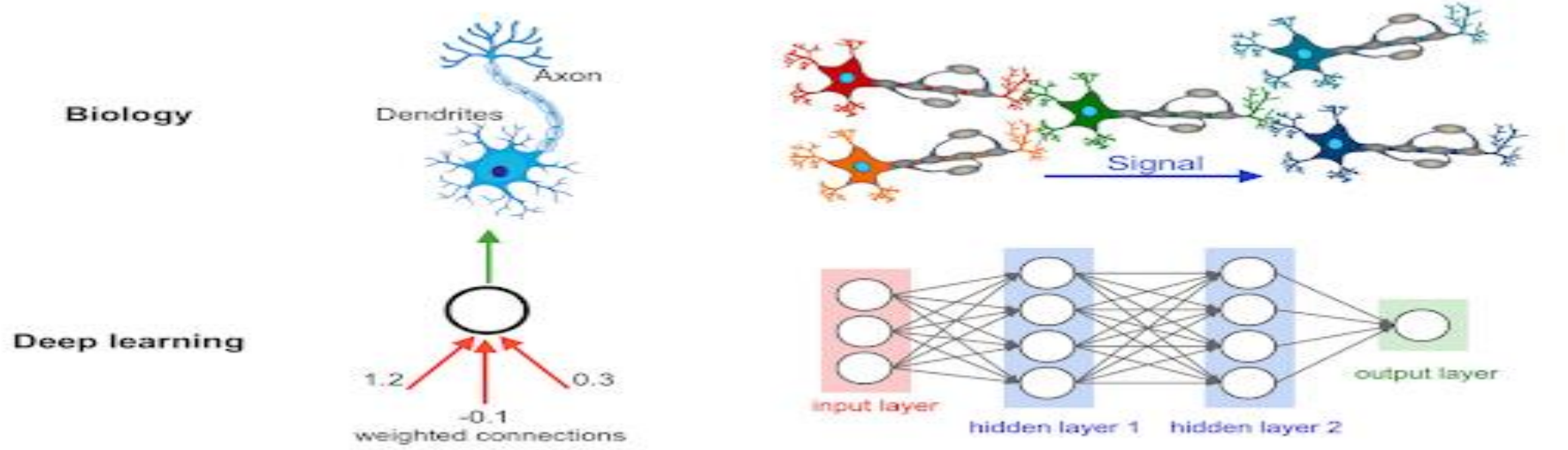
Deep Learning Vs. Machine Learning

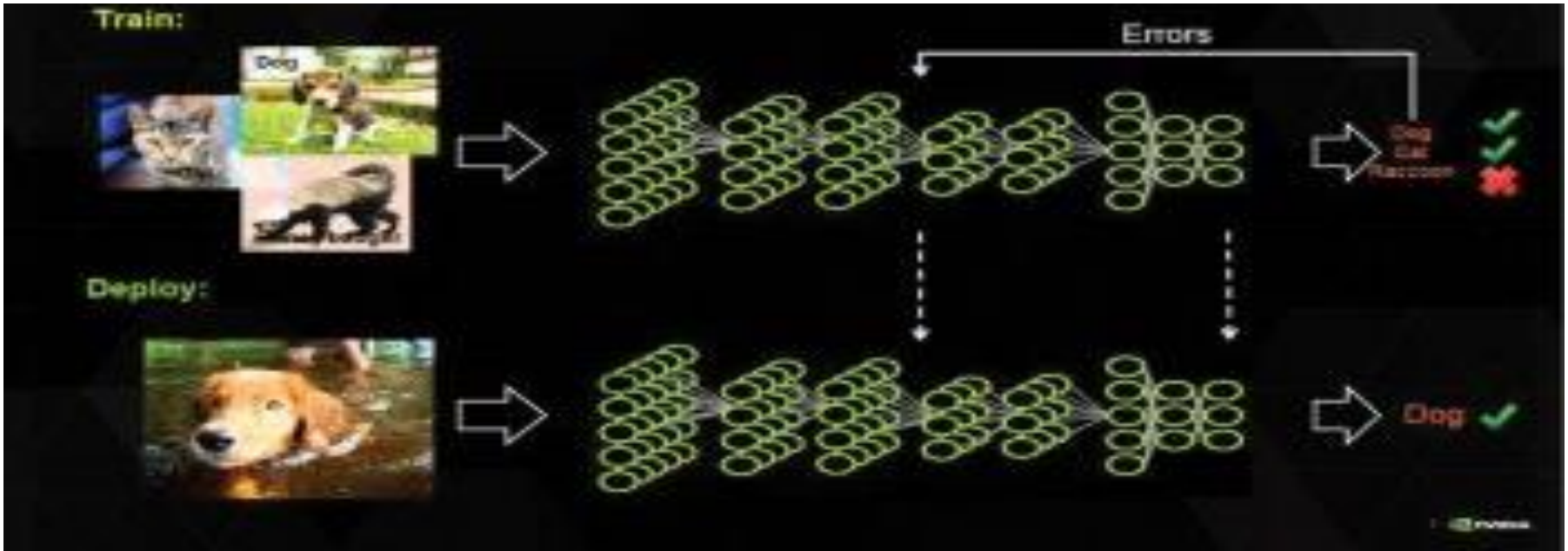
- Interpretability
  - is the main issue why many sectors using other **Machine Learning** techniques over **Deep Learning**.
- **Example:**
  - Suppose we use deep learning to calculate the relevance score of a document.
  - The performance it gives is quite excellent and is near human performance. But there's is an issue.
  - It does not reveal why it has given that score.
  - Indeed mathematically you can find out which nodes of a deep neural network were activated, but we don't know what there neurons were supposed to model and what these layers of neurons were doing collectively. So we fail to interpret the results.
  - Which is not in case of **Machine Learning** algorithms like decision trees, logistic regression etc.

How Deep Learning Works?

- To understand deep learning, imagine how toddler learns.

How Deep Learning Works?





Unit 1.
Introduction to Deep Learning

What this unit is about:

- This is an introduction to machine learning with the TensorFlow library.
- We will implement relatively simple machine learning algorithms using the TensorFlow library.

Expected outcome:

- Proficiency with the TensorFlow paradigm.
- Ability to build and utilize deep neural networks.

How to check your progress:

- Coding Exercises.
- Quiz.

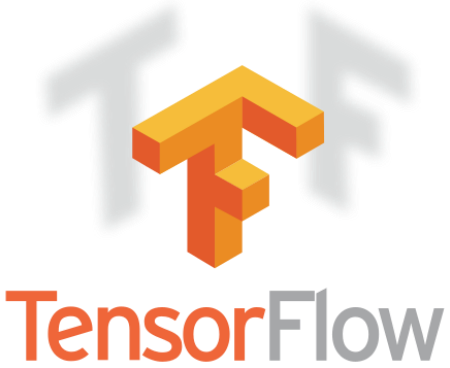
Chapter 8.

Deep Learning - Part I

TensorFlow Basics (1/12)

About the TensorFlow library:

- An open source machine learning/deep learning library developed by the Google Brain team.
- Computations are expressed as graphs.
- Supports Windows, mac OS, Linux, etc.
- Supports Python, Java, R, etc.
- Runs on multiple CPUs and GPUs.
- For more information refer to the official site: https://www.tensorflow.org/



TensorFlow Basics (2/12)

Printing “Hello World”:

Ex) A few steps are required to print out a text message.

```
t>>> import tensorflow as f
>>> hello = tf.constant("Hello World")
>>> sess = tf.Session()
>>> print(sess.run(hello))                    # Print the result.
b'Hello World'
```

TensorFlow Basics (3/12)

Dataflow graph:

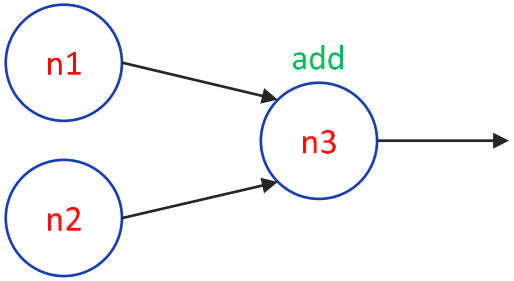
- A tensor is analogous to an array and can represent scalars, matrices and high dimensional arrays.
- A node represents a tensor.
- A dataflow graph is made up by nodes connected by edges.
- A dataflow graph defines the computation in terms of the dependencies between individual operations.
- First define the dataflow graph and then evaluate it with tf.Session.run(). (*)

TensorFlow Basics (4/12)

Dataflow graph:

Ex) Sum of two numbers.

```
>>> n1 = tf.constant(1)
>>> n2 = tf.constant(2)
>>> n3 = n1 + n2                    # Define the addition of two numbers.
>>> with tf.Session() as sess:
...       result = sess.run(n3)
>>> print(result)
3
```

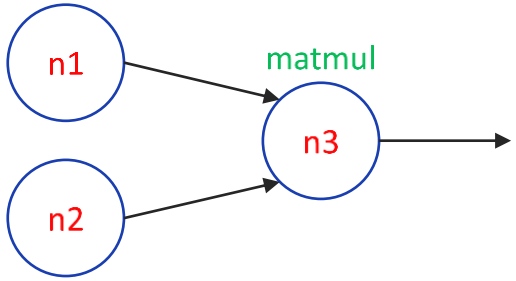


TensorFlow Basics (5/12)

Dataflow graph:

Ex) Matrix multiplication.

```
>>> n1 = tf.constant([[1,2], [3,4]])
>>> n2 = tf.constant([[1], [2]])
>>> n3 = tf.matmul(n1, n2)                    # Define the matrix multiplication.
>>> with tf.Session() as sess:
...       result = sess.run(n3)
>>> print(result)
[[5]
[11]]
```



TensorFlow Basics (6/12)

Creating tensors:

Ex) Define tensors and then run.

```
>>> const_scalar = tf.constant(7)                                  # A scalar as tensor.
>>> const_matrix = tf.constant([[1,2], [3,4]])                     # A matrix as tensor.
>>> mat_fill = tf.fill((4,4),9)                                    # A 4 x 4 matrix filled with 9s.
>>> mat_zeros = tf.zeros((3,3))                                    # A 3 x 3 matrix filled with 0s.
>>> mat_ones = tf.ones((5,5))                                      # A 5 x 5 matrix filled with 1s.
>>> mat_randn = tf.random_normal((3,3), mean=0, stddev=1.0)        # A 3 x 3 random normal matrix.
>>> mat_randu = tf.random_uniform((4,4), minval=0, maxval=1.0)     # A 4 x 4 random uniform matrix.
```

```
>>> my_ops=[const_scalar, const_matrix, mat_fill, mat_zeros, mat_ones, mat_randn, mat_randu]
>>> sess = tf.Session()
>>> for op in my_ops:
...     print(sess.run(op))                                        # Tensor resource is allocated.
...     print('\n')
```

TensorFlow Basics (7/12)

Variables and Placeholders:

- In TensorFlow, variables represent the model parameters.
- Variables are quantities that need to be “learned” ⇔ calculated using the training data.
- Placeholders take the places of “features” or “variables” in the usual sense.
- Placeholders are used to feed an external data into the TensorFlow graph.
- A placeholder’s data type should be specified at the definition.

TensorFlow Basics (8/12)

Variables and Placeholders:

Ex) Variable initialization.

```
>>> my_tensor = tf.random_uniform((4,4),0,1)
>>> my_var = tf.Variable(initial_value=my_tensor)          # The initial value set at the definition.
>>> init = tf.global_variables_initializer()
>>> sess = tf.Session()
>>> sess.run(init)                                         # First, run the initializer.
>>> sess.run(my_var)
```

TensorFlow Basics (9/12)

Variables and Placeholders:

Ex) For a placeholder, the number of rows set to None means an arbitrary number of rows.

```
>>> ph = tf.placeholder(tf.float32, shape=(None,5))
```

Ex) The actual values for the placeholders are allocated through feed_dict.

```
>>> a = tf.placeholder(tf.float32)
>>> b = tf.placeholder(tf.float32)
>>> y = tf.multiply(a, b)
>>> sess       = tf.Session()
>>> print(sess.run(y, feed_dict={a:2, b:3}))          # feed_dict receives the actual values as a dictionary.
6
```

TensorFlow Basics (10/12)

TensorFlow data types:

Data Type	Explanation
tf.float32	32 bit floating point
tf.float64	64 bit floating point
tf.int16	16 bit integer
tf.int32	32 bit integer
tf.int64	64 bit integer
tf.string	String
tf.bool	Boolean

TensorFlow Basics (11/12)

TensorFlow mathematical functions:

Function	Explanation
tf.add(a, b)	$a + b$
tf.subtract(a, b)	$a - b$
tf.multiply(a, b)	$a * b$
tf.mod(a, b)	$a \% b$
tf.abs(a)	$\|a\|$
tf.sign(a)	1 if $a > 0$, else -1
tf.square(a)	a^2
tf.sqrt(a)	$\sqrt{a}$
tf.pow(a, b)	a^b
tf.log(a)	$Log(a)$
tf.exp(a)	e^a
tf.cos(a), tf.sin(a)	$\cos(a), \sin(a)$

TensorFlow Basics (12/12)

TensorFlow matrix functions:

Function	Explanation
tf.transpose(m)	Matrix transpose m^t
tf.matmul(m1, m2)	Matrix multiplication
tf.matrix_determinant(m)	Matrix determinant
tf.matrix_diag([diag_list])	Diagonal matrix
tf.matrix_inverse(m)	Inverse matrix m^{-1}

Coding Exercise #0701

Follow practice steps on ‘ex_0701.ipynb’ file.

Deep Learning - Part I

UNIT 1. Introduction to Deep Learning

UNIT 2. Deep Learning Various Topics

Machine Learning with TensorFlow (1/13)

Linear regression with TensorFlow:

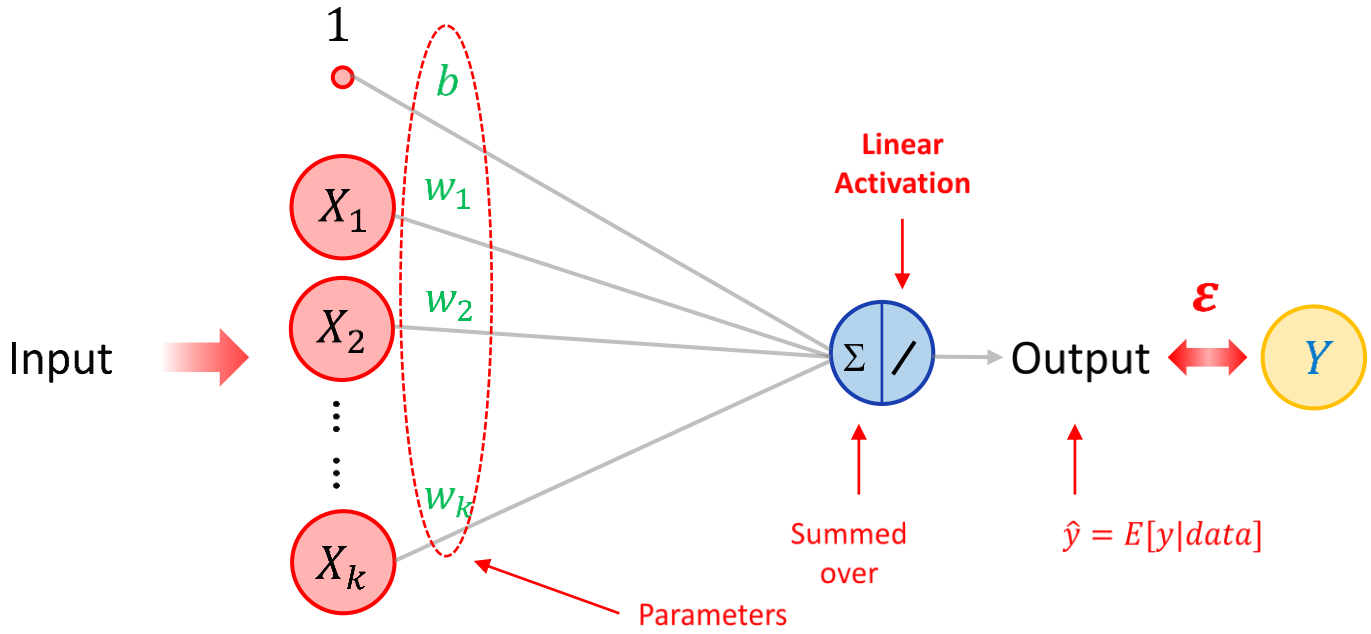
- There is one or more explanatory variables: X_1, X_2,…, X_k
- There is one response variable: Y
- The variables X_i and Y are connected by a linear relation:

$$Y = b + w_1 X_1 + w_2 X_2 + \cdots + w_k X_k + \varepsilon$$

  - The coefficients previously denoted as β_i, from now and on will be called "weights" w_i.
  - The intercept previously denoted as β_0, from now and on will be called "bias" b.
- We will use the "linear" activation function.

Machine Learning with TensorFlow (2/13)

Linear regression with TensorFlow:



- The disagreement between the predicted $\hat{y}$ and the true y has to be minimized by training.

Machine Learning with TensorFlow (3/13)

Linear regression with TensorFlow:

- As we have an over determined system of linear equations, the exact solution does not exist.
- We can minimize the loss (cost) defined as $\left|\boldsymbol{Y}-\widehat{\boldsymbol{Y}}\right|^2$ or $\left|\boldsymbol{Y}-\widehat{\boldsymbol{Y}}\right|$ and get the "best" solution $(b,\boldsymbol{W})$.
- If we define the loss as $L=\left|\boldsymbol{Y}-\widehat{\boldsymbol{Y}}\right|^2$, the solution $(b,\boldsymbol{W})$ can be calculated using matrices.
  - However, with TensorFlow we will take a different approach: Gradient descent algorithm.

Machine Learning with TensorFlow (4/13)

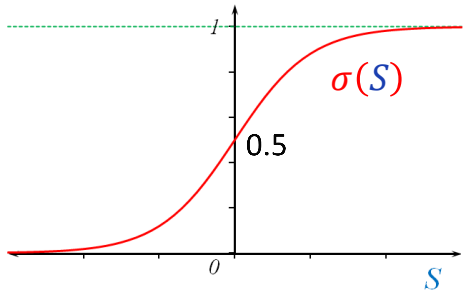
Logistic regression with TensorFlow:

- The linear combinations of variables X_i is the so-called "Logit" denoted here as S :

$$S = b + w_1 X_1 + w_2 X_2 + \cdots + w_k X_k$$

- The conditional probability of Y being equal to 1 is denoted as $P(Y=1 \mid \{x_i\})$.
- A "Sigmoid" function connects the probability with the logit:

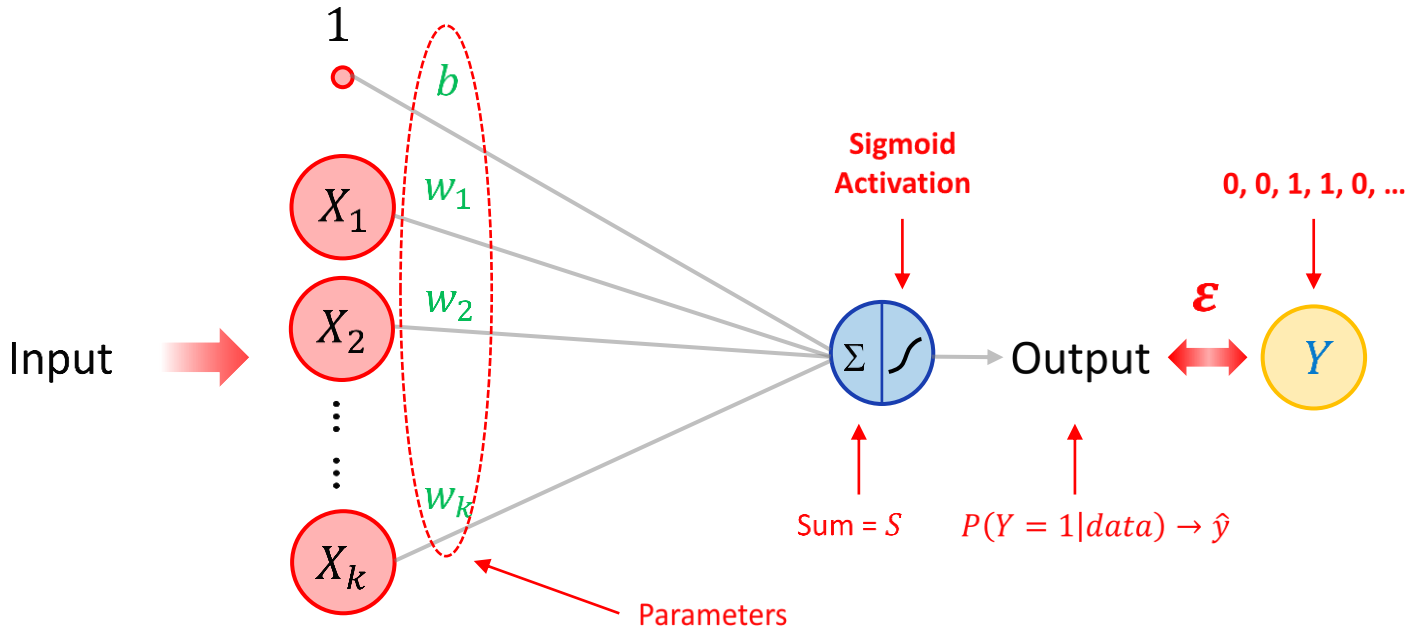
$$\sigma(S) = \frac{e^{S}}{1 + e^{S}}$$



- This Sigmoid is the "activation function" ⇐ the biggest difference with the linear regression.

Machine Learning with TensorFlow (5/13)

Logistic regression with TensorFlow:



- The mismatch between the predicted $\hat{y}$ and the true y has to be minimized by training.

Machine Learning with TensorFlow (6/13)

Logistic regression with TensorFlow:

- We can recall the negative of logarithmic likelihood with the redefined $y_i = -1 \text{ or } +1$. (*)

$$-\sum_{i=1}^{n} Log(1 + e^{-y_i S})$$

- We now rename this as the "entropy" and express it in terms of the predicted probability $\hat{p}_i$.

$$L = -\frac{1}{n}\sum_{i=1}^{n}(y_i log(\hat{p}_i) + (1 - y_i)\, log(1 - \hat{p}_i))$$

  - We went back to the definition $y_i = 0 \text{ or } 1$.
  - This is the loss function we'd like to minimize by the gradient descent algorithm.

- Binary logistic regression can be generalized to the multi-class version using the Softmax activation and multi-class cross entropy as the loss function.

Machine Learning with TensorFlow (7/13)

Loss (cost) functions in TensorFlow:

Name	Formula	TensorFlow Expression
Squared Error (L2 Error)	$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$	tf.reduce_sum(tf.square(y_true – y_model))
Absolute Error (L1 Error)	$L = \sum_{i=1}^{n} \lvert y_i - \hat{y}_i \rvert$	tf.reduce_sum(tf.abs(y_true – y_model))
Binary Cross Entropy	$L = -\frac{1}{n} \sum_{i=1}^{n} (y_i log(\hat{p}_i) + (1 - y_i)\ log(1 - \hat{p}_i))$	tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=y_true, logits=y_model))
Multi-Class Cross Entropy	$L = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} y_{ik}\ log(\hat{p}_{ik})$	tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_true, logits=y_model))

- Cross entropy is the appropriate loss function for the classification.

Machine Learning with TensorFlow (8/13)

Gradient descent algorithm:

- $L(b,\boldsymbol{W})$ is minimized iteratively in small "steps" pushing $(b,\boldsymbol{W})$ along the direction $-\nabla L(b,\boldsymbol{W})$:

  1) $(b,\boldsymbol{W})$ is randomly initialized.

  2) Calculate the gradient $\nabla L(b,\boldsymbol{W})$.

  3) Update $(b,\boldsymbol{W})$ by one step: $(b,\boldsymbol{W}) \leftarrow (b,\boldsymbol{W}) - \eta\, \nabla L(b,\boldsymbol{W})$.

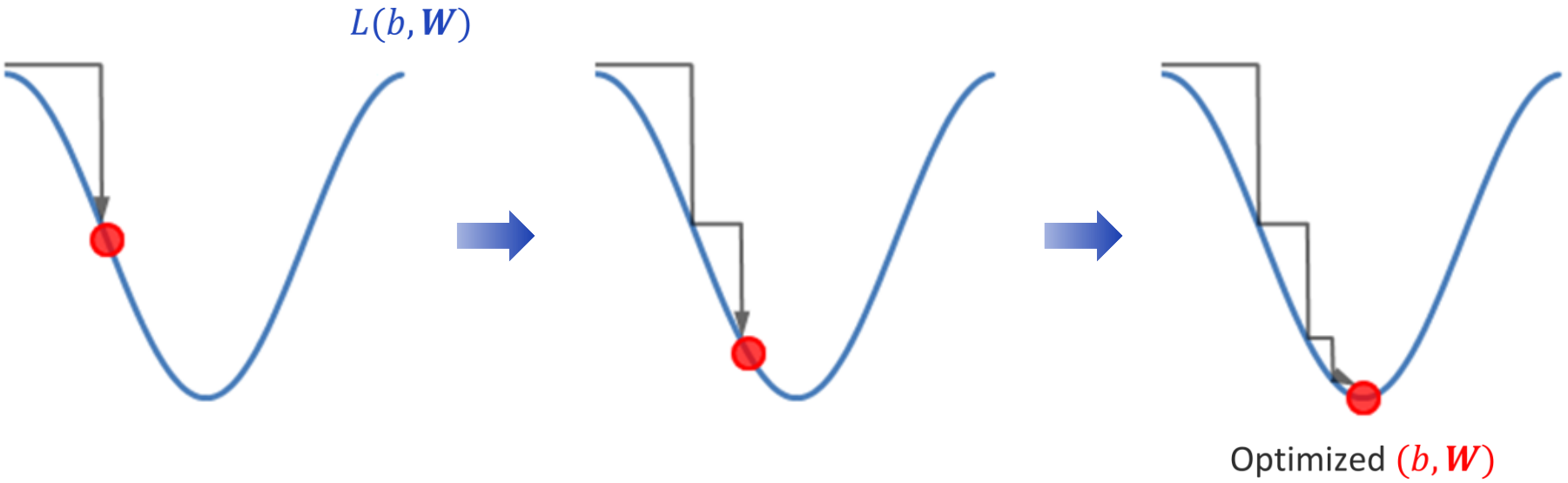
     Convergence speed is controlled by the "Learning rate" η.

  4) Repeat from the step 2) for a fix number of times (epochs).

Machine Learning with TensorFlow (9/13)

Gradient descent algorithm:

- $L(b,\boldsymbol{W})$ is minimized iteratively in small "steps" pushing $(b,\boldsymbol{W})$ along the direction $-\nabla L(b,\boldsymbol{W})$:



Machine Learning with TensorFlow (10/13)

Batch learning (or mini-batch gradient descent):

- Instead of using the whole training dataset for calculating the gradient, only a subset (batch) is used.
- Batch is randomly sampled at every gradient descent step.
- Batch size is small independent of the whole training data size.
- It is advantageous for the generalization (testing).
- It requires less computing power.

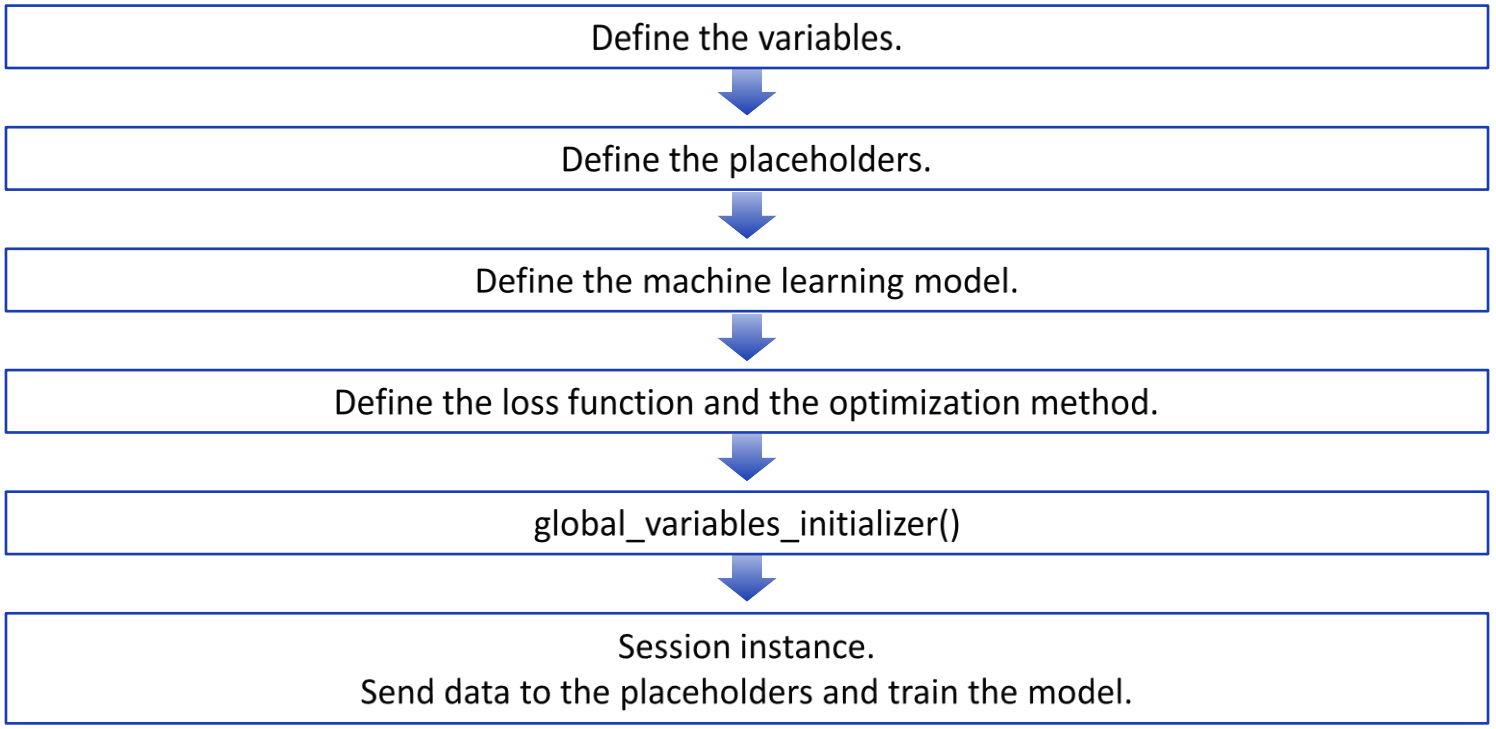
Machine Learning with TensorFlow (11/13)

Gradient descent algorithms:

Algorithm	Explanation	TensorFlow
Gradient Descent	Gradient descent algorithm with batch learning.	tf.train.GradientDescentOptimizer()
Momentum	Takes into account the gradient from the previous step.	tf.train.MomentumOptimizer(nesterov=False)
Nesterov Momentum	Minimizes unnecessary movements.	tf.train.MomentumOptimizer(nesterov=True)
Adagrad	Adaptative step size.	tf.train.AdagradOptimizer()
RMSProp	Improves upon the Adagrad.	tf.train.RMSPropOptimizer()
Adam	Combines the best properties of Adagrad and RMSProp.	tf.train.AdamOptimizer()

Machine Learning with TensorFlow (12/13)

Training work flow:



Machine Learning with TensorFlow (13/13)

MNIST data of handwritten digits:

- Handwritten images of digits (0~9). Image size is 28×28 (748 pixels).
- There are 55,000 training and 10,000 testing cases.
- The digit label is one-hot-encoded.

X0	X1	X2	X3	X4	X5	X6	X7	X8	X9
0	0	0	0	0	1	0	0	0	0

Coding Exercise #0702

Follow practice steps on ‘ex_0702.ipynb’ file.

Coding Exercise #0703

Follow practice steps on ‘ex_0703.ipynb’ file.

Coding Exercise #0704a

Follow practice steps on ‘ex_0704a.ipynb’ file.

Coding Exercise #0704b

Follow practice steps on ‘ex_0704b.ipynb’ file.

Deep Learning - Part I

Tensor Board (1/5)

About the Tensor Board:

- With TensorFlow we first define the graphs and then execute.
- Tensor Board is a tool that helps to visualize the log data of the TensorFlow execution.
- The information needed for the visualization has to be saved as event files.

Tensor Board (2/5)

Starting the Tensor Board:

1) At the command prompt:

```
> tensorboard --logdir= <log folder>
```

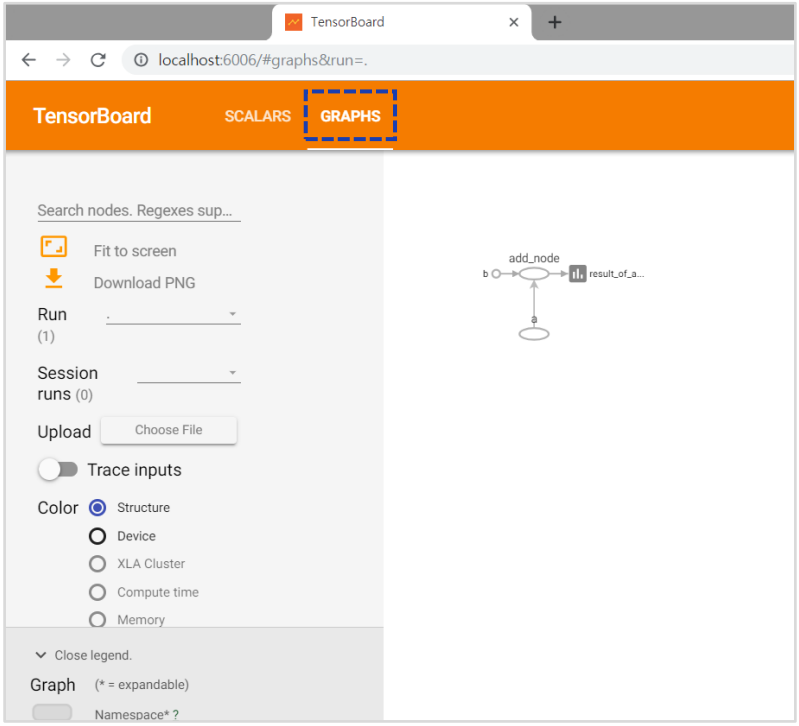
2) Start a web browser and then enter the address at the URL locator.

localhost:6006

- More information can be found at: https://www.tensorflow.org/tensorboard

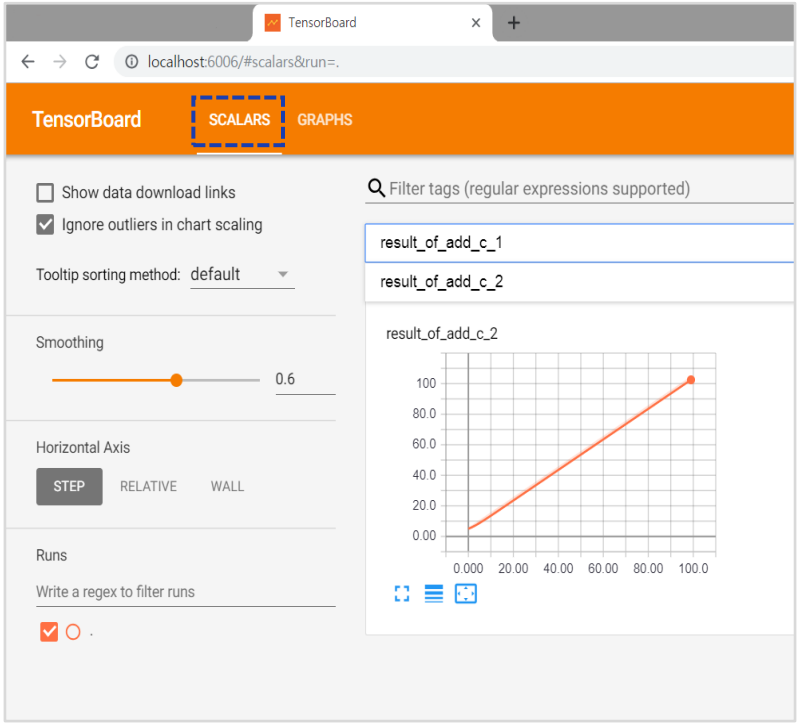
Tensor Board (3/5)

- Visualizing dataflow graphs:



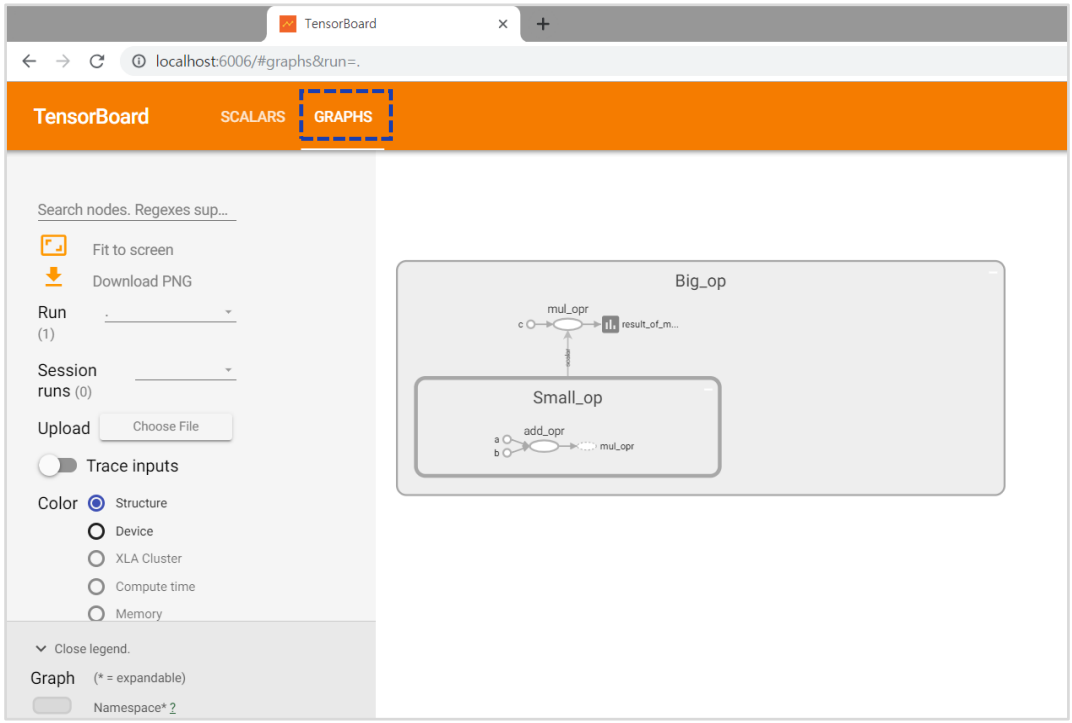
Tensor Board (4/5)

- Visualizing scalars:



Tensor Board (5/5)

- Tensor Board name scope:



  - More information can be found at: https://www.tensorflow.org/tensorboard

Coding Exercise #0705a

Follow practice steps on ‘ex_0705a.ipynb’ file.

Coding Exercise #0705b

Follow practice steps on ‘ex_0705b.ipynb’ file.

Coding Exercise #0705c

Follow practice steps on ‘ex_0705c.ipynb’ file.

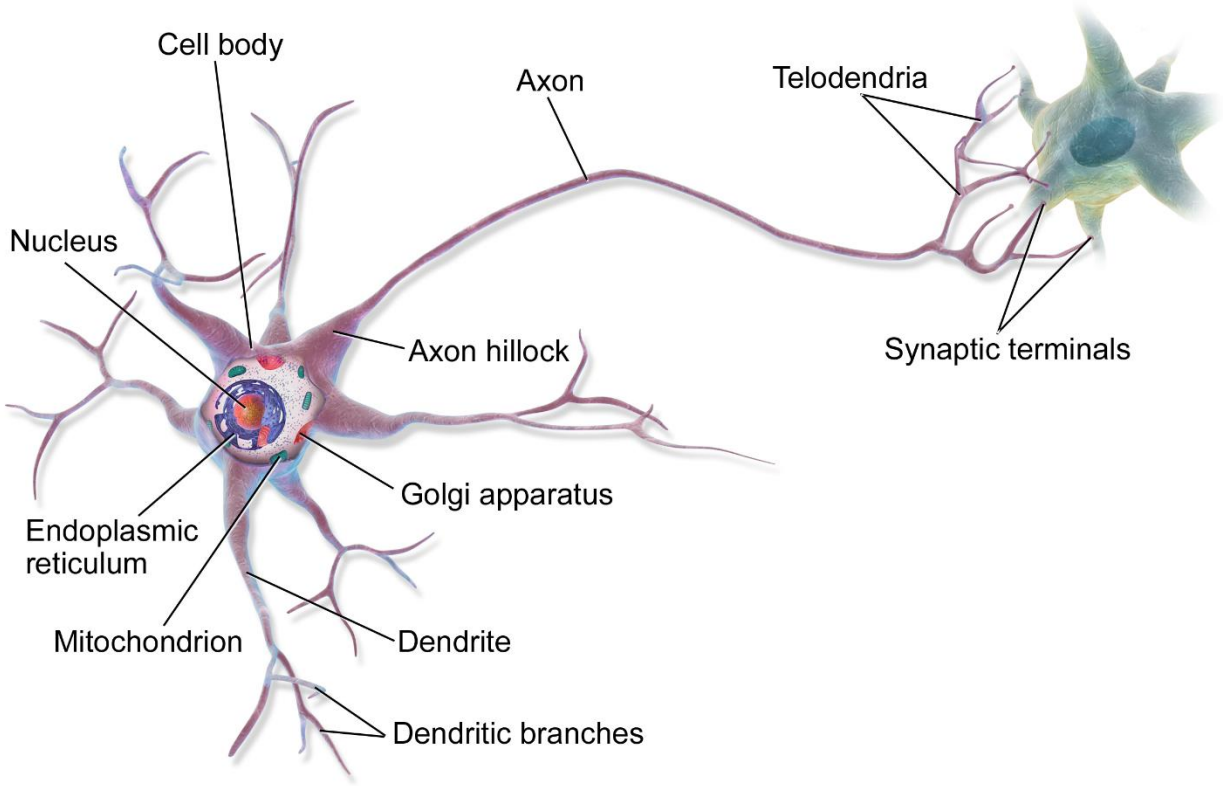
Coding Exercise #0705d

Follow practice steps on ‘ex_0705d.ipynb’ file.

Deep Learning - Part I

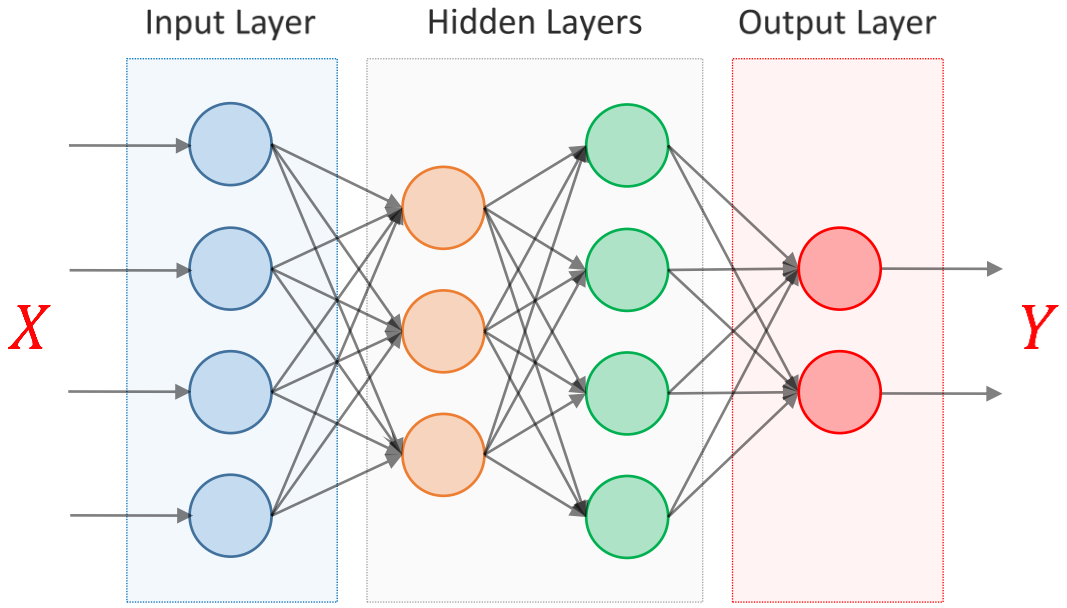
Artificial Neural Network (1/23)

- Biological origin:



Artificial Neural Network (2/23)

About the artificial neural network (ANN):

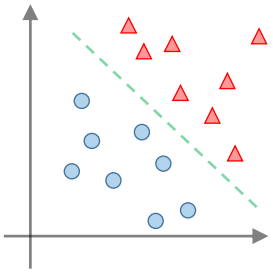


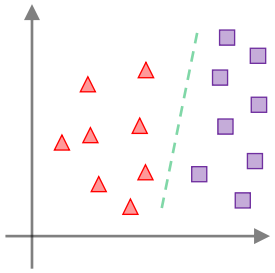
- Mimics the neural connections of a biological brain.
- There can be several hidden layers.

Artificial Neural Network (3/23)

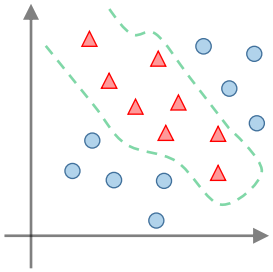
Why Artificial Neural Network (ANN)?

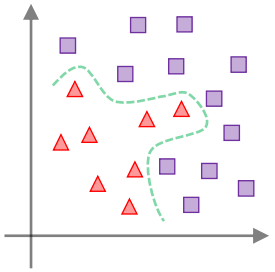
- For a logistic regression (even for the multi-class variant), the decision boundaries are linear.





- How about the cases that require non-linear decision boundaries? ⇒ ANN with hidden layers!



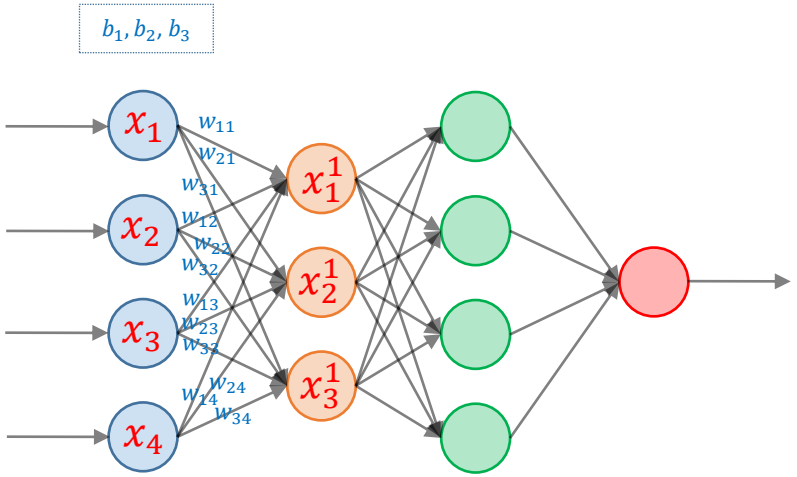


Artificial Neural Network (4/23)

ANN training: forward propagation.

1) When the values of X_i are given, apply the weights and propagate the signal forward to the next layer.

Ex)



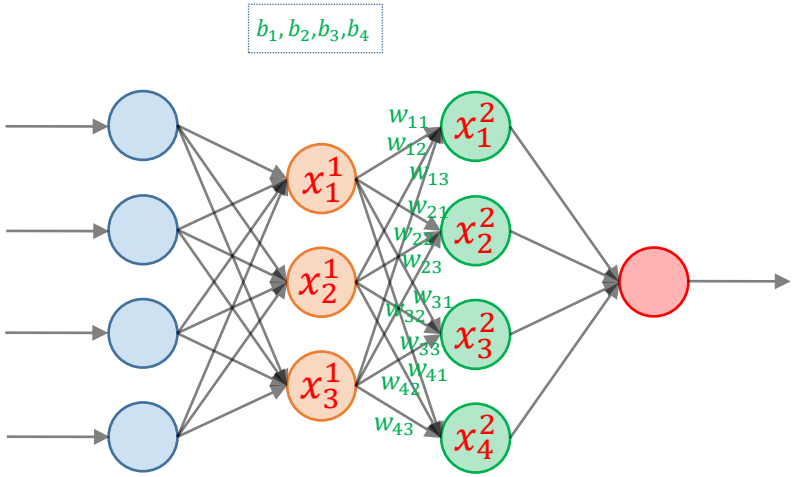
$$\begin{bmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{bmatrix} = Activation\left(\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Artificial Neural Network (5/23)

ANN training: forward propagation.

2) Propagate forward to the next layer.

Ex)



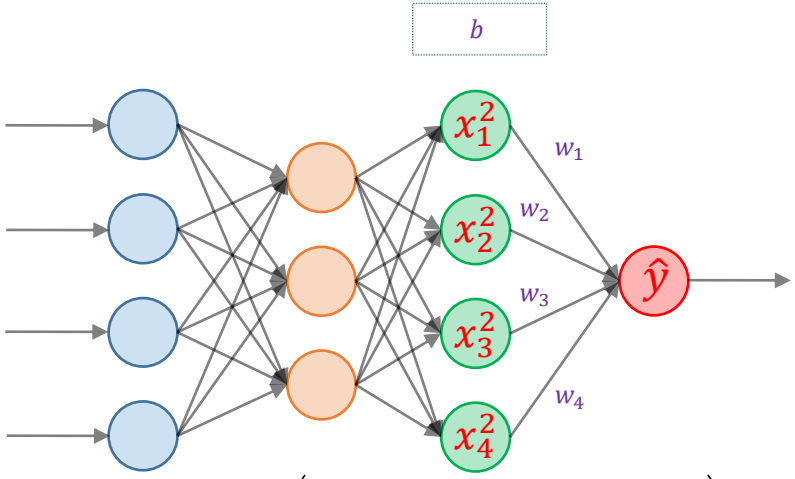
$$\begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{bmatrix} = Activation\left(\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{42} & w_{42} & w_{43} \end{bmatrix} \cdot \begin{bmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}\right)$$

Artificial Neural Network (6/23)

ANN training: forward propagation.

3) Propagate all the way; the difference between the estimated value $\hat{y}$ and the true value y is the error.

Ex)



$\cong y?$

$$Error = y - \hat{y}$$

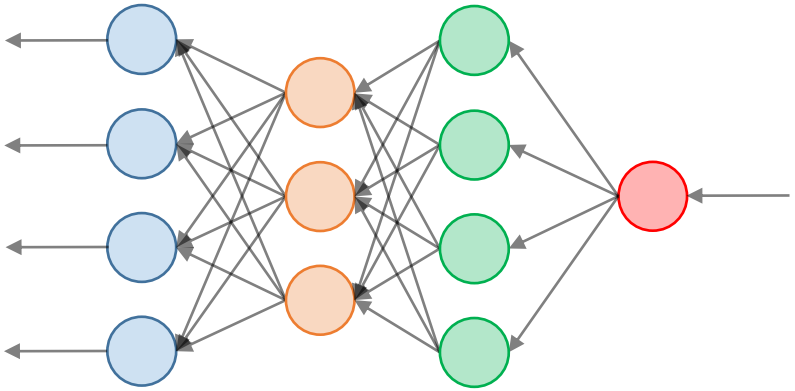
$$\hat{y} = Activation\left(\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix} \cdot \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{bmatrix} + b \right)$$

Artificial Neural Network (7/23)

ANN training: backward propagation.

4) Propagate the error backward and update the parameters by gradient descent algorithm. Repeat from 1).

Ex)



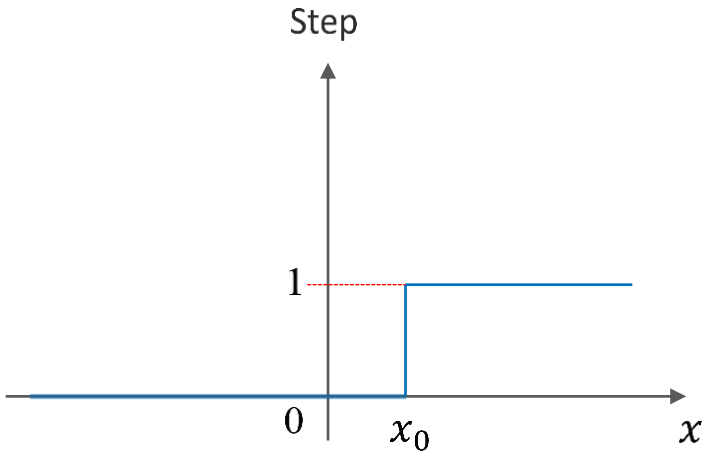
$$W \leftarrow W - \eta \nabla Loss$$
$$b \leftarrow b - \eta \nabla Loss$$

Error or Loss

Artificial Neural Network (8/23)

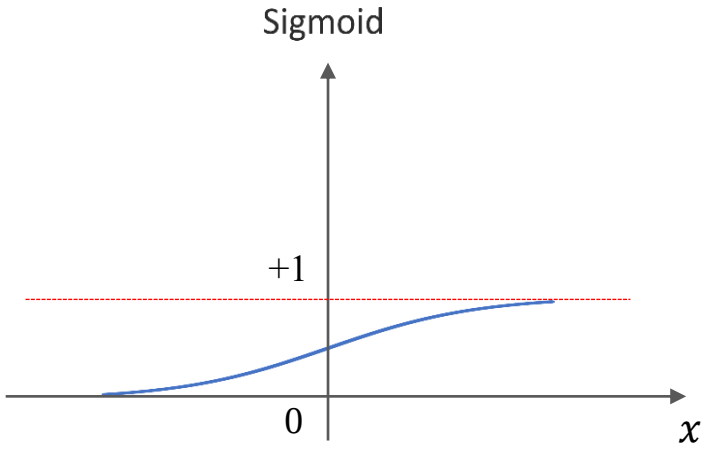
Activation function: Linear threshold or Step.

- $Step(x) = \theta(x - x_0)$



Artificial Neural Network (9/23)

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

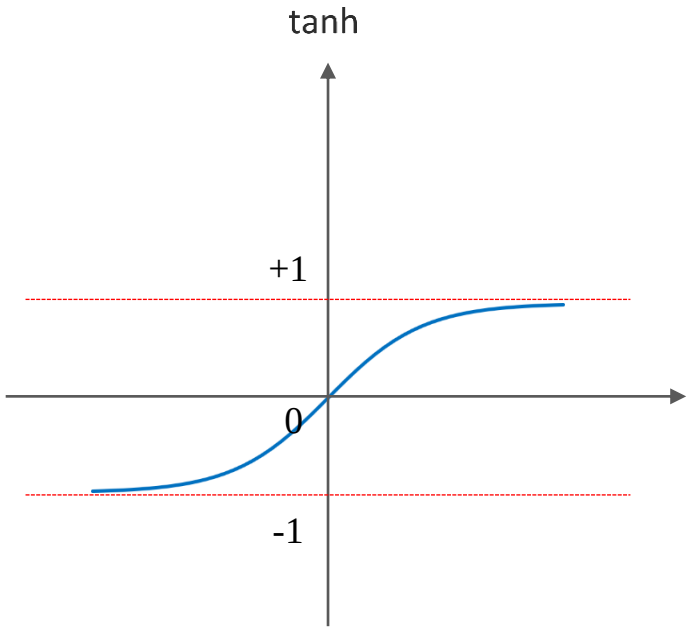


  - When we have more than two classes, it can be generalized to the “Softmax”.

Artificial Neural Network (10/23)

Activation function: Hyperbolic tangent or tanh.

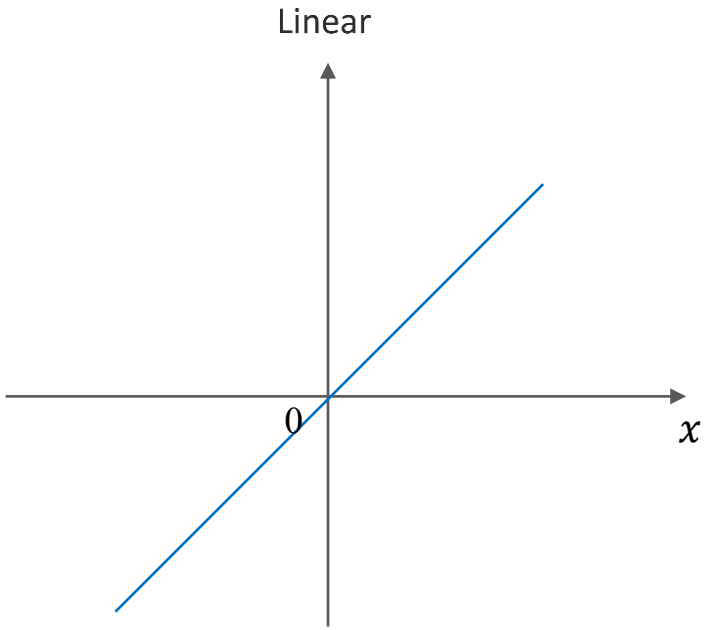
- $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



Artificial Neural Network (11/23)

Activation function: Linear.

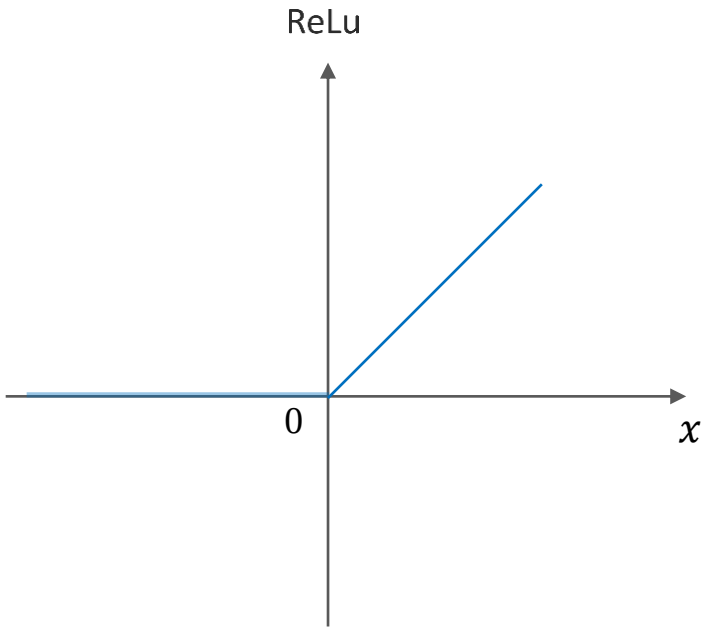
- $Linear(x) = x$



Artificial Neural Network (12/23)

Activation function: ReLu (Rectifier Linear Unit).

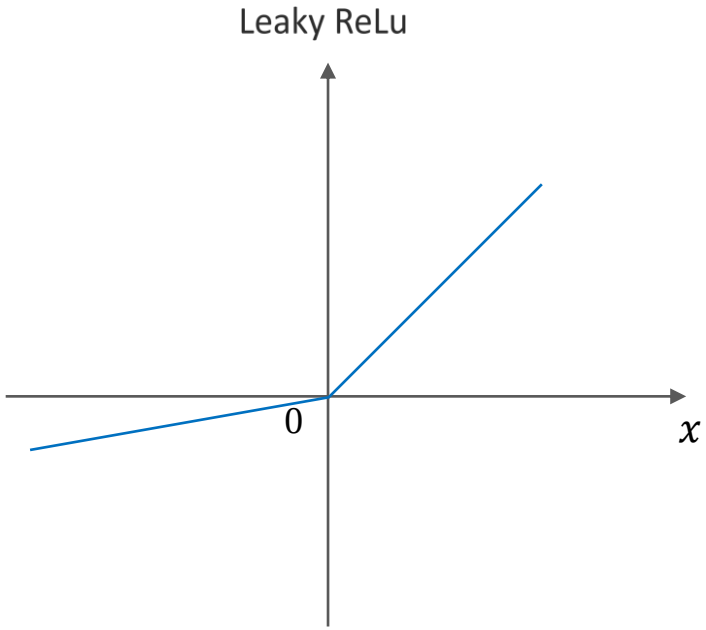
- $ReLu(x) = max(0, x)$



Artificial Neural Network (13/23)

Activation function: Leaky ReLu.

- $Leaky\ ReLu(x) = max(\alpha x, x)\ where\ \alpha \in (0,1)$



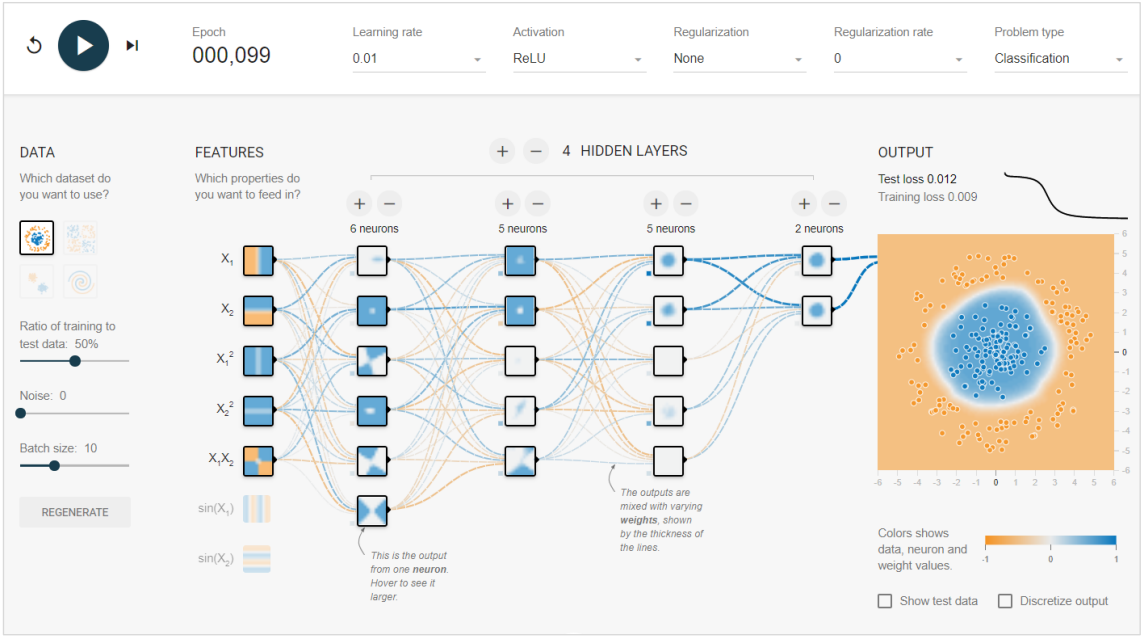
Artificial Neural Network (14/23)

Activation function:

Name	Formula	TensorFlow
Step	$Step(x) = \theta(x - x_0)$	(tf.sign(x-x0)+1)/2
Sigmoid	$\sigma(x) = \frac{1}{1 + e^{-x}}$	tf.sigmoid(x)
Tanh	$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$	tf.tanh(x)
ReLu	$ReLu(x) = \max(0, x)$	tf.nn.relu(x)
Softmax	$\sigma(\boldsymbol{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}$	tf.nn.softmax(x)

Artificial Neural Network (15/23)

Artificial neural network: a recommended site



- Website: https://playground.tensorflow.org

Artificial Neural Network (16/23)

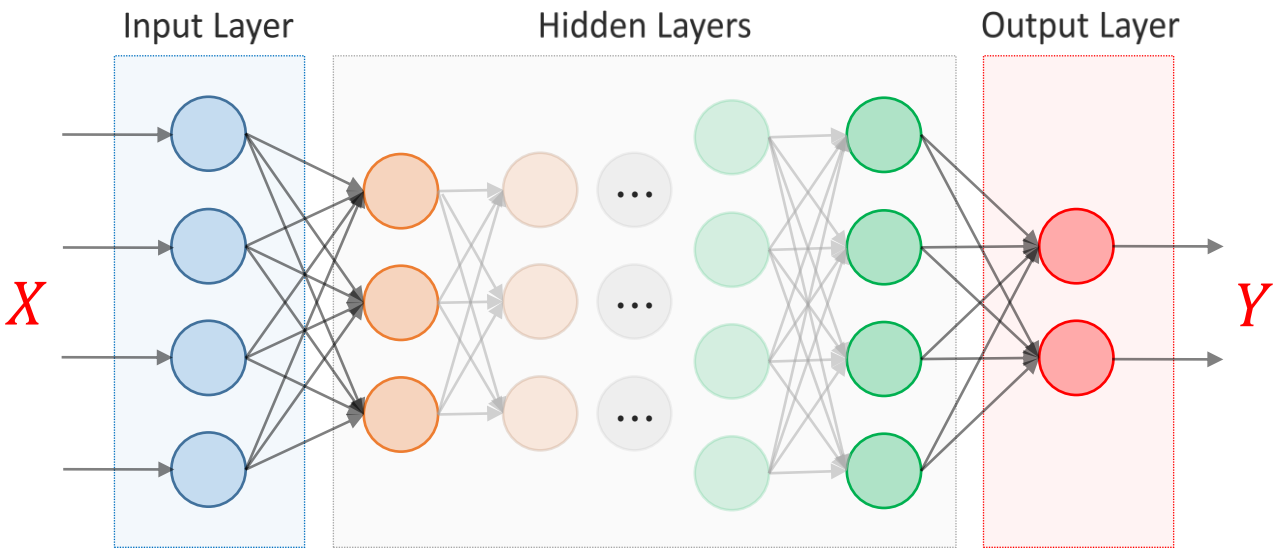
Artificial neural network: a recommended site

- Website: https://playground.tensorflow.org
- Select data from the left side panel.
- Adjust the features, layers and nodes.
- From the top drop down menus, customize the learning rate, activation function, regularization, etc.
- Press the Play button to start training ⇒ training loss decreases as the training progresses.
- The thickness of an edge line represents the weight that can also be changed manually.

Artificial Neural Network (17/23)

About the Deep Neural Network (DNN):

- There may be several hidden layers.
- With more hidden layers, we have a higher degree of abstraction.



Artificial Neural Network (18/23)

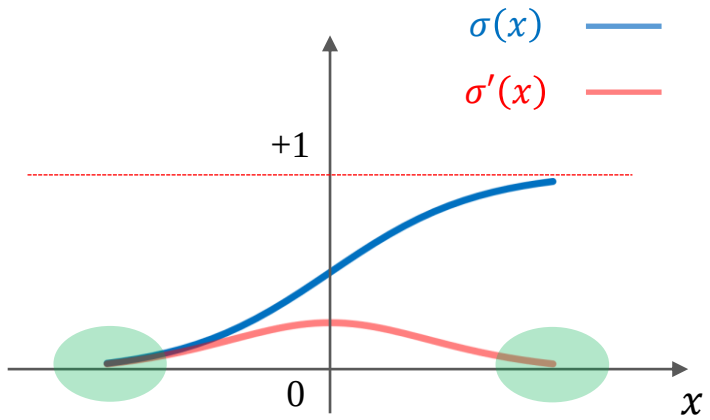
Common problems with the DNN:

- As the 'signal' propagates through the layers, it can die out (vanish) or grow uncontrollably.

  a) We should avoid using the Sigmoid $\sigma(x)$ as the activation function and use the ReLu instead.

  → The derivative $\sigma'(x)$ becomes very small as the absolute value of x or $|x|$ increases.

  → Small $\sigma'(x)$ may cause the so-called vanishing gradient problem.



Artificial Neural Network (19/23)

As the 'signal' propagates through the layers, it can die out (vanish) or grow uncontrollably.

- As the 'signal' propagates through the layers, it can die out (vanish) or grow uncontrollably.

  b) We should initialize the weights such that the variance of the internal nodes is roughly constant.

  → Initializing as a constant such as 1 is not optimal.

  → Better to randomly initialize the weights with center at 0 and standard deviation $\sim \frac{1}{\sqrt{N_{nodes}}}$. (*)

  Here, N_{nodes} = average number of nodes in the neighboring layers (before and after).

- (*) Refer to the research paper at http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

Artificial Neural Network (20/23)

Common problems with the DNN:

- Also, DNNs are prone to the overfitting. To alleviate this problem we can do:

  a) Regularization of the weights: L1 or L2 regularization similar to the Lasso or Ridge.

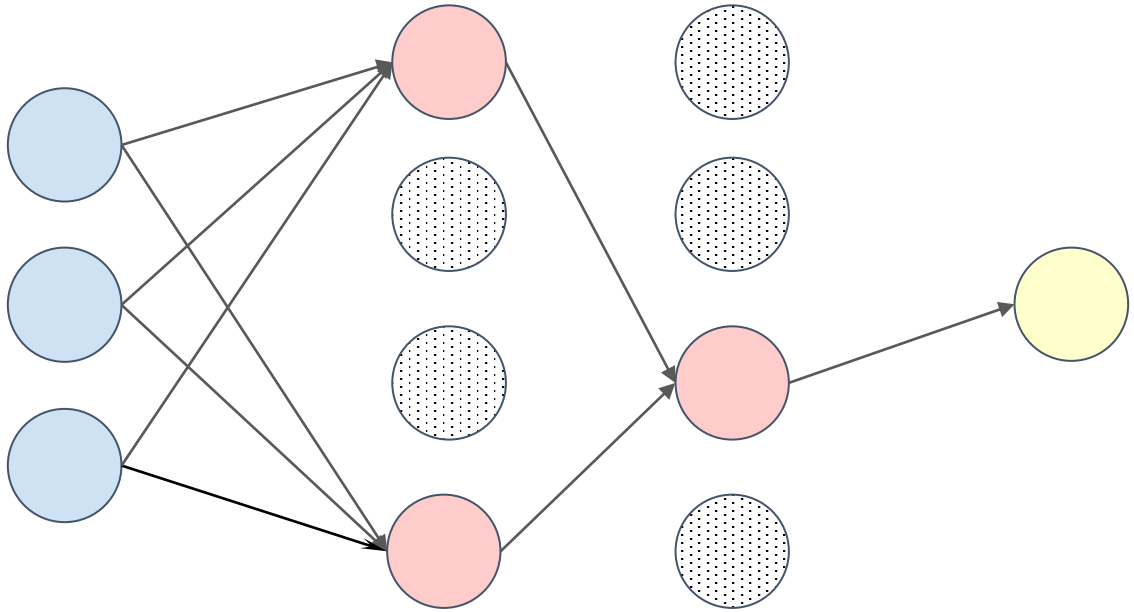
  b) Dropout: randomly exclude certain nodes during the training step to avoid over dependency.

  c) Data augmentation: add noise, create new data by transformation, etc.

Artificial Neural Network (21/23)

Dropout:

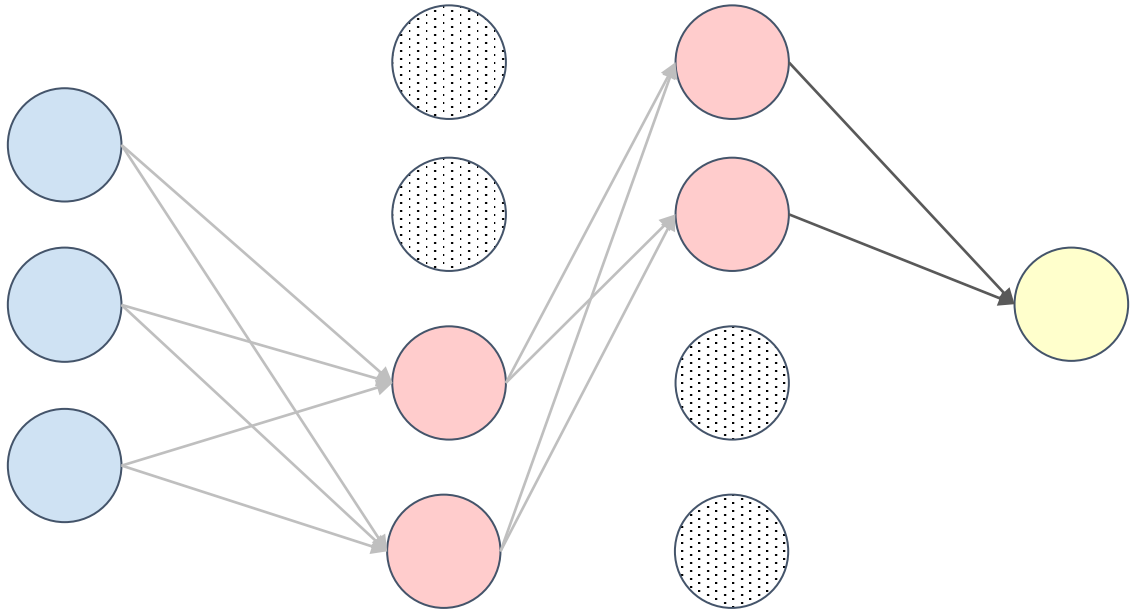
- Randomly exclude certain nodes during the training step to avoid over dependency.



Artificial Neural Network (22/23)

Dropout:

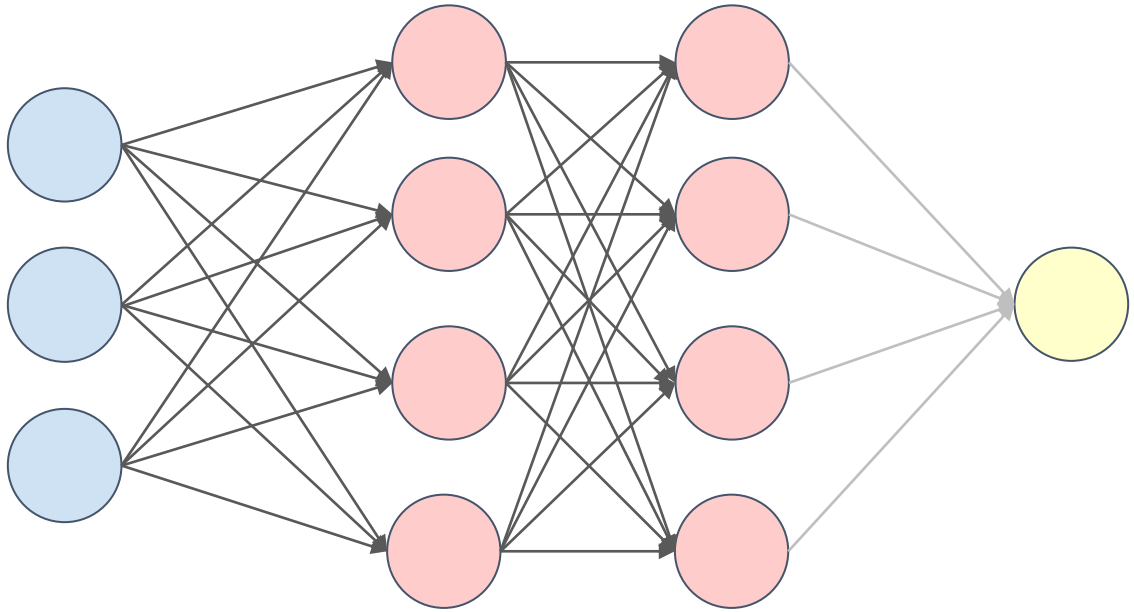
- Randomly exclude certain nodes during the training step to avoid over dependency.



Artificial Neural Network (23/23)

Dropout:

- When predicting, use all the nodes and edges without excluding.



Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 8.

Deep Learning - Part I

UNIT 2.

Deep Learning Various Topics

Samsung Innovation Campus

Unit 2.
Deep Learning Various Topics

What this unit is about:

- Convolutional neural networks for detecting locally correlated patterns in images.
- Recurrent neural network for predicting sequence data.
- Brief overview on the LSTM.
- Brief introduction to the generative adversarial networks (GAN).
- AutoEncoders for dimensional reduction and feature extraction.

Expected outcome:

- Ability to build and utilize various deep learning models for specific purposes.

How to check your progress:

- Coding Exercises.
- Quiz.

Deep Learning - Part I

Convolutional Neural Network (1/17)

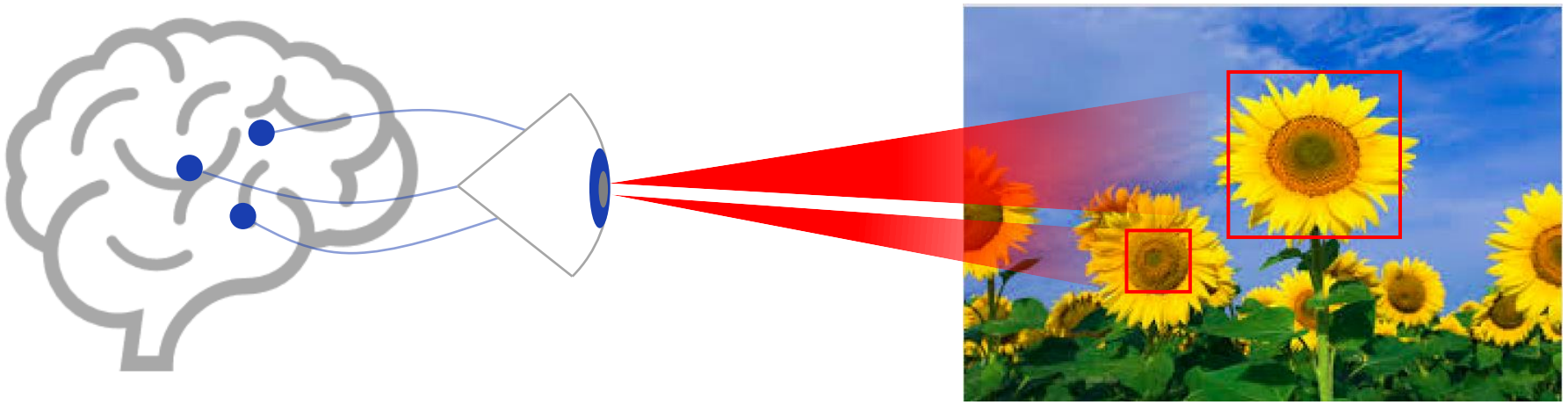
About the Convolutional Neural Network (CNN):

- CNN is a deep neural network commonly used for image classification.
- Background:
  - In 1990s, Yann LeCun's LeNet-5 was one of the pioneering examples of the CNN.
  - In 2012, Alex Krizhevsky's AlexNet winner of the ImageNet challenge.
  - In 2014, C. Szegedy's GoogleNet winner of the ImageNet challenge.

Convolutional Neural Network (2/17)

About the Convolutional Neural Network (CNN):

- Also has a biological origin.
- In the brain, the visual cortex recognizes an image in localized patches.



Convolutional Neural Network (3/17)

Convolution filter:

- Given two functions f and g, the convolution is denoted using an asterisk: $f * g$.
- If f and g are functions of a continuous variable, the convolution is an integral:

$$(f * g) = (x) \int f(x')g(x - x')dx'$$

- If f and g are functions of a discrete variable, the convolution is a sum:

$$(f * g)(i) = \sum_{i'} f(i')g(i - i')$$

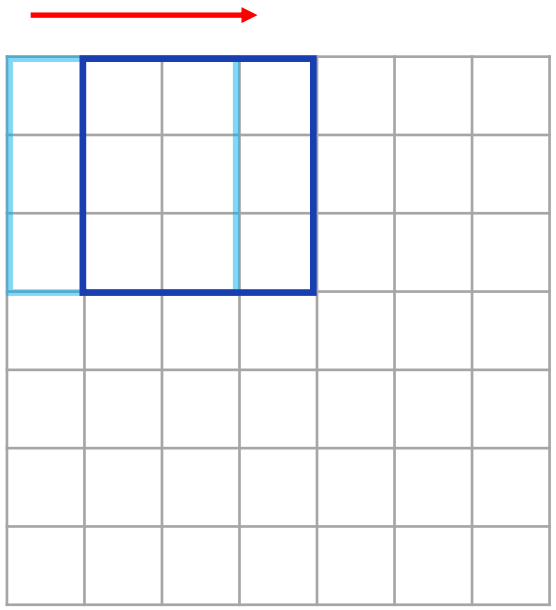
  - f can be the so-called 'kernel' that acts like a filter and g be the image.
  - When a 2D image G is filtered by a finite size 2D kernel K, we have:

$$(K * G)(i, j) = \sum_{(i',j') \in K} K(i', j')G(i + i', j + j')$$

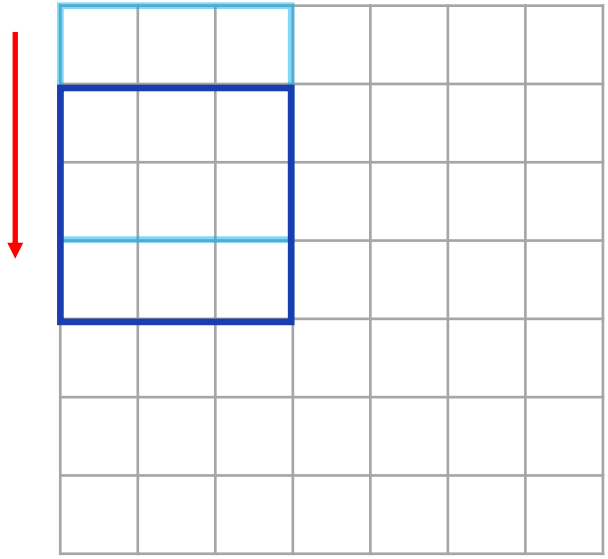
Convolutional Neural Network (4/17)

Convolution filter:

- Moving the filter (kernel) through the image ⇔ Moving (i, j) of $(K * G)(i, j)$.



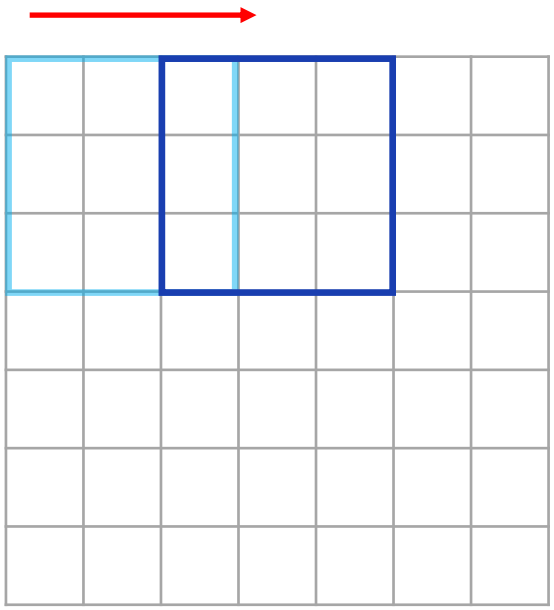
Stride = (1,1)



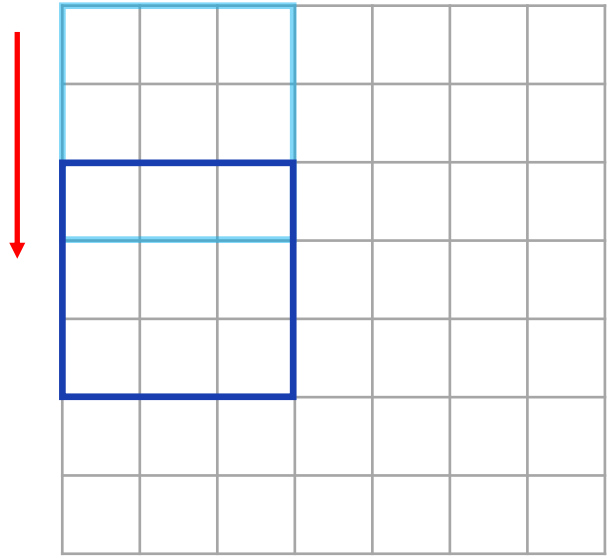
Convolutional Neural Network (5/17)

Convolution filter:

- Moving the filter (kernel) through the image ⇔ Moving (i, j) of $(K * G)(i, j)$.



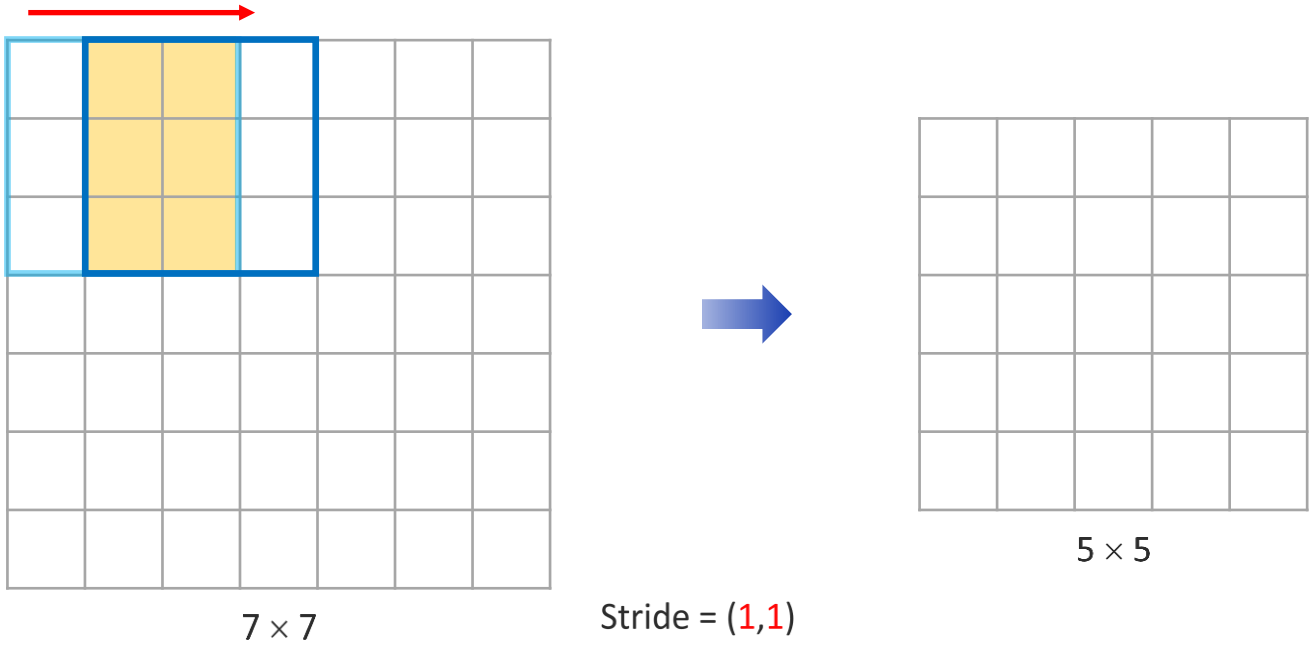
Stride = (2,2)



Convolutional Neural Network (6/17)

Convolution filter:

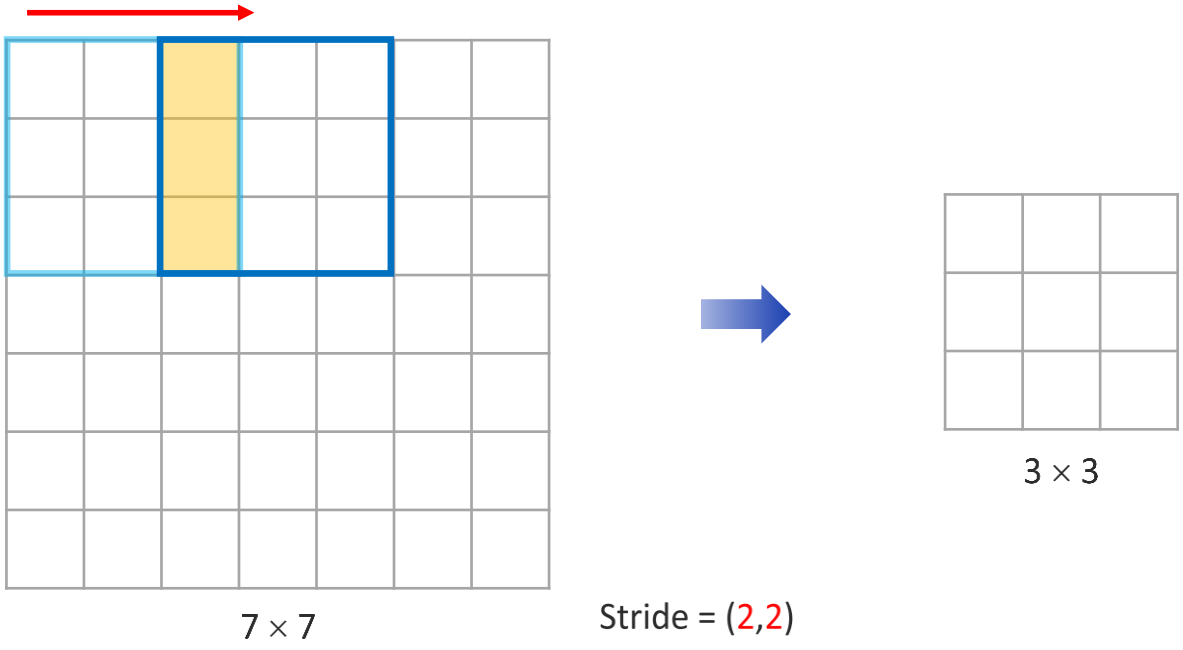
- The filtered image suffers from a undesirable side effect: size reduction.



Convolutional Neural Network (7/17)

Convolution filter:

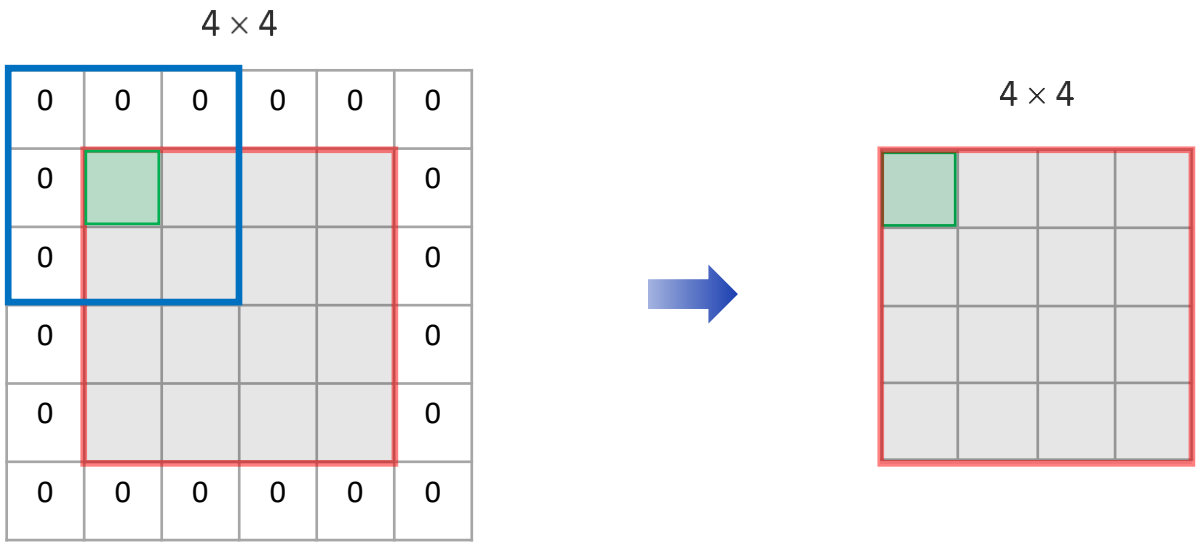
- The filtered image suffers from a undesirable side effect: size reduction.



Convolutional Neural Network (8/17)

Convolution filter:

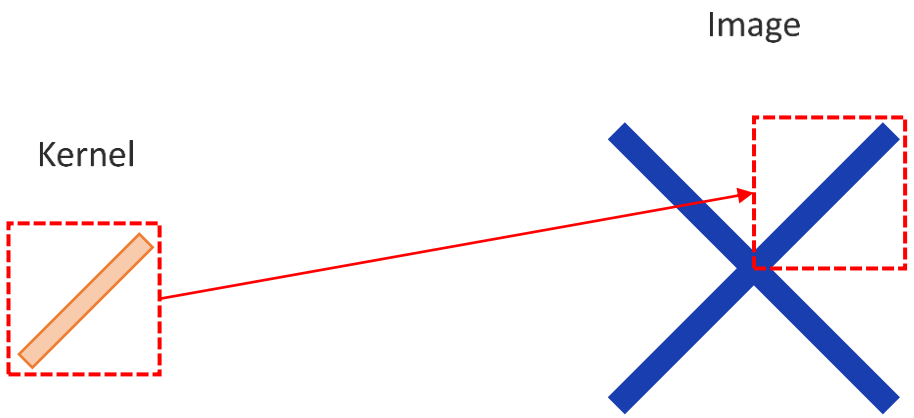
- Padding: additional cells with 0 are attached to the outer boundary, so that the size is maintained.



Stride = (1,1)

Convolutional Neural Network (9/17)

Convolution filters can be used to detect local patterns:



Convolutional Neural Network (10/17)

Convolution filters can be used to detect local patterns:

Kernel matrix

0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
1	0	0	0	0	0	0

$*$

Local image matrix

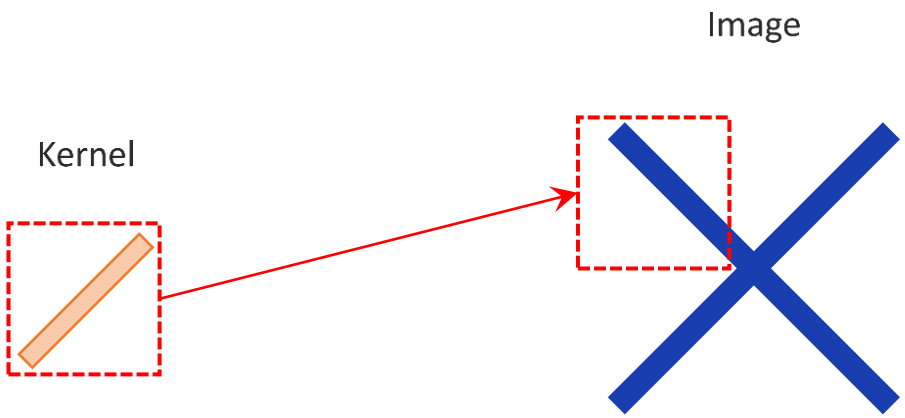
0	0	0	0	0	0	255
0	0	0	0	0	255	0
0	0	0	0	255	0	0
0	0	0	255	0	0	0
0	0	255	0	0	0	0
0	255	0	0	0	0	0
255	0	0	0	0	0	0

- Apply the left kernel to the right image by convolution.

  $\rightarrow 1 \times 255 + \cdots + 1 \times 255 = 7 \times 255 = 1785$ (several pixels match).

Convolutional Neural Network (11/17)

Convolution filters can be used to detect local patterns:



Convolutional Neural Network (12/17)

Convolution filters can be used to detect local patterns:

Kernel matrix

0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
1	0	0	0	0	0	0

$*$

Local image matrix

255	0	0	0	0	0	0
0	255	0	0	0	0	0
0	0	255	0	0	0	0
0	0	0	255	0	0	0
0	0	0	0	255	0	0
0	0	0	0	0	255	0
0	0	0	0	0	0	255

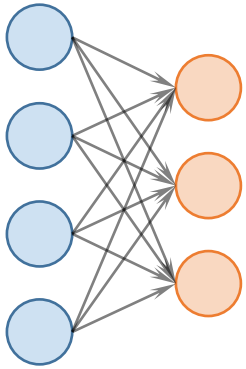
- Apply the left kernel to the right image by convolution.

  $\rightarrow 1 \times 255 = 255$ (there is only one match).

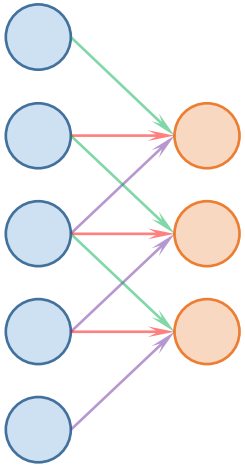
Convolutional Neural Network (13/17)

Convolution filters in a neural network:

- It is possible to implement a convolution filter with the neural network.
- The edges only connect locally ⇒ just like when a kernel is applied.
- The weights are the kernel elements ⇒ can be trained with data!



Fully Connected

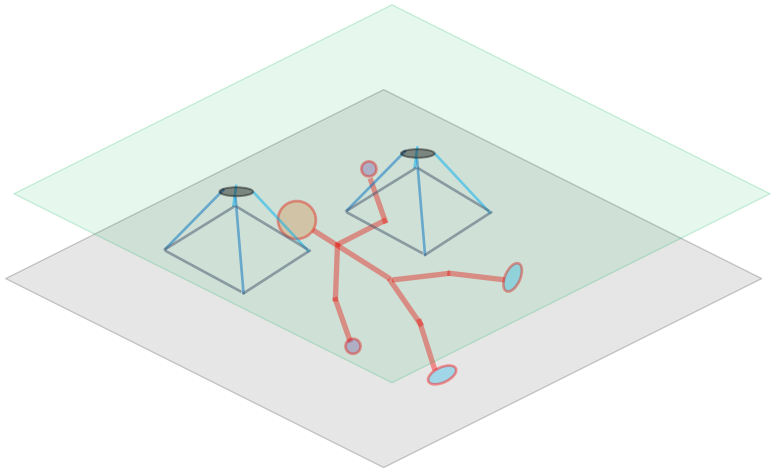


Convolution

Convolutional Neural Network (14/17)

Convolution layer:

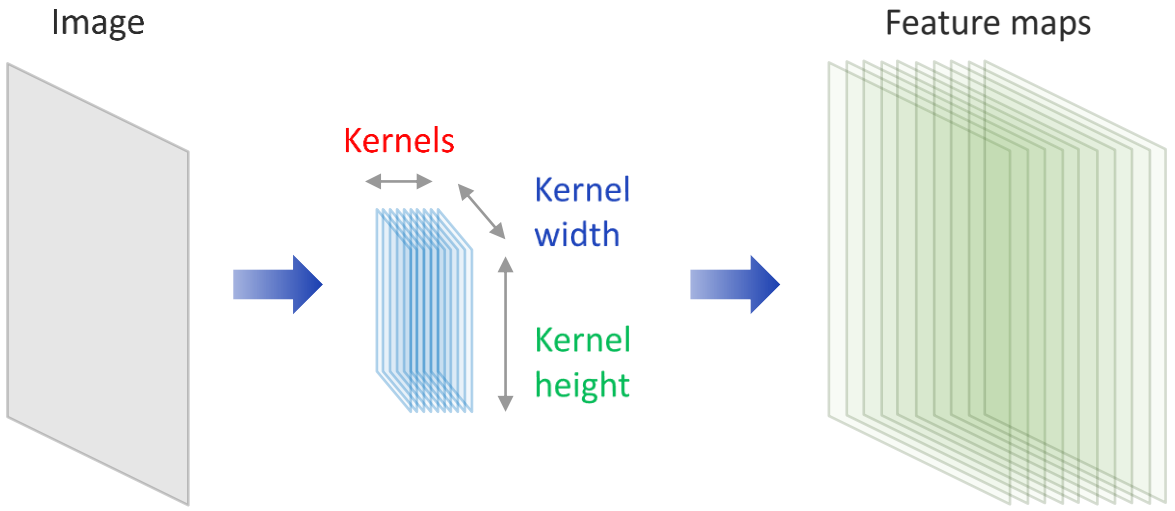
- The image is scanned with a kernel and the next layer (feature map) is produced.



Convolutional Neural Network (15/17)

Convolution layer:

- Usually we have a set of kernels (filters) with a fixed size $width \times height$.
- A kernel produces a feature map; so as many feature maps are produced as the number of kernels.
- The color channels (Red, Green, Blue) can be convoluted with separate kernels.

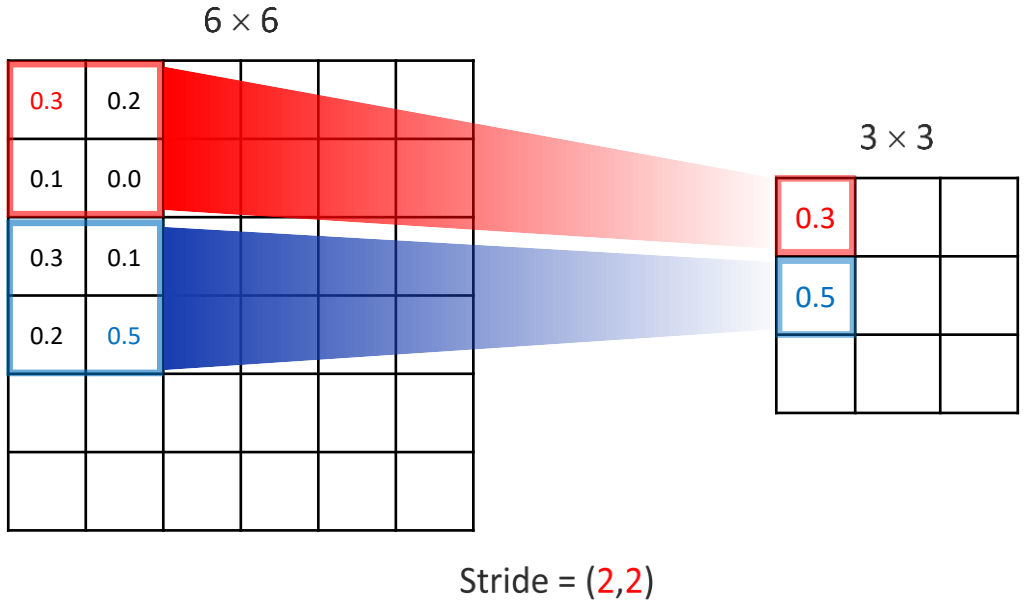


Convolutional Neural Network (16/17)

Pooling (sub-sampling):

- Move the kernel and summarize. Pooling helps in preventing the overfitting problem.

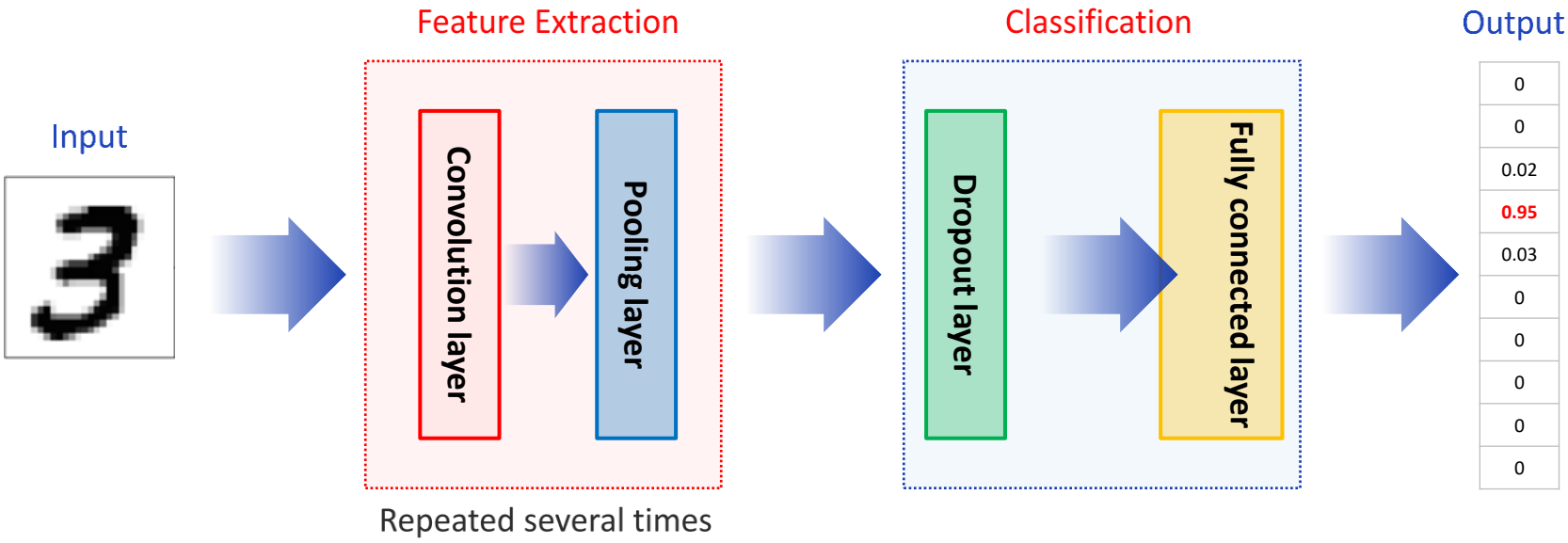
  Ex) Max-pooling.



Convolutional Neural Network (17/17)

Diagram of a Convolutional Neural Network (CNN):

- The convolution and pooling layers are repeated several times.
- There is a fully connected layer just before the output.



Coding Exercise #0707

Follow practice steps on ‘ex_0707.ipynb’ file.

SAMSUNG

Together for Tomorrow!

Enabling People

Education for Future Generations