Convolutional Neural Network using TensorFlow

Youngjae Yu (yj.yu@vision.snu.ac.kr)

Seil Na (seil@vision.snu.ac.kr)

Junhyug Noh (jh.noh@vision.snu.ac.kr)

SEOUL NATIONAL UNIV.
VISION & LEARNING

SK telecom

Contents

Dataset: notMNIST

Dataset: notMNIST

- Similar to MNIST, but much more complex
- 10 classes with letters A to J

Dataset: notMNIST

- Each image has 28 x 28 = 784 pixels.
- Consists of two parts
  - Large (Train): ~500k uncleaned images
  - Small (Test): ~19k hand-cleaned images
- http://commondatastorage.googleapis.com/books1000/
  - notMNIST_large.tar.gz
  - notMNIST_small.tar.gz

Dataset: notMNIST

- Download compressed files (tar.gz)
- Extract and get the dataset
  - notMNIST_{large|small}/{A-J}/
  - Each directory contains images
- Save images as 3-D array
  - Scale the pixel values (-0.5 to 0.5)
  - Remove invalid images
- Merge and make train/validation/test sets
  - Set the manageable size of each dataset
  - Distribute labels uniformly (train/validation from large, test from small)

Dataset: notMNIST

Let’s check the code.

1_dataprocess.ipynb

TensorFlow Basics

TensorFlow Basics

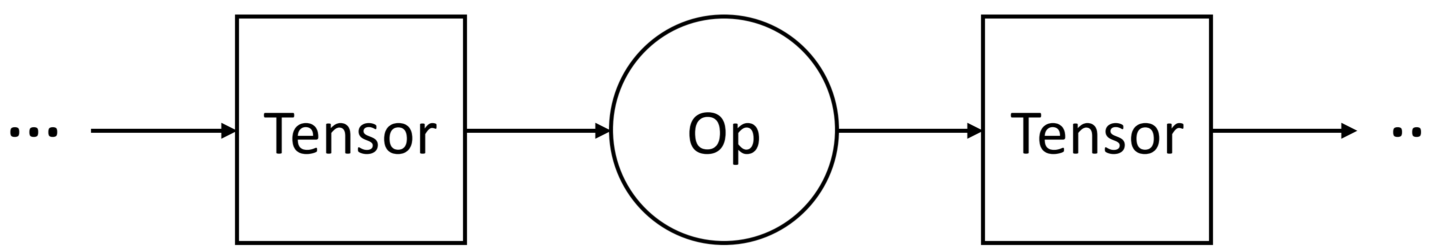
- Graph
  - A TensorFlow computation, represented as a dataflow graph
  - Contains a set of Operation and Tensor objects.
- Session
  - Encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.

```
g = tf.Graph()
with g.as_default():
  # Define operations and tensors in `g`

with tf.Session(graph=g) as sess:
  sess.run(...)
```

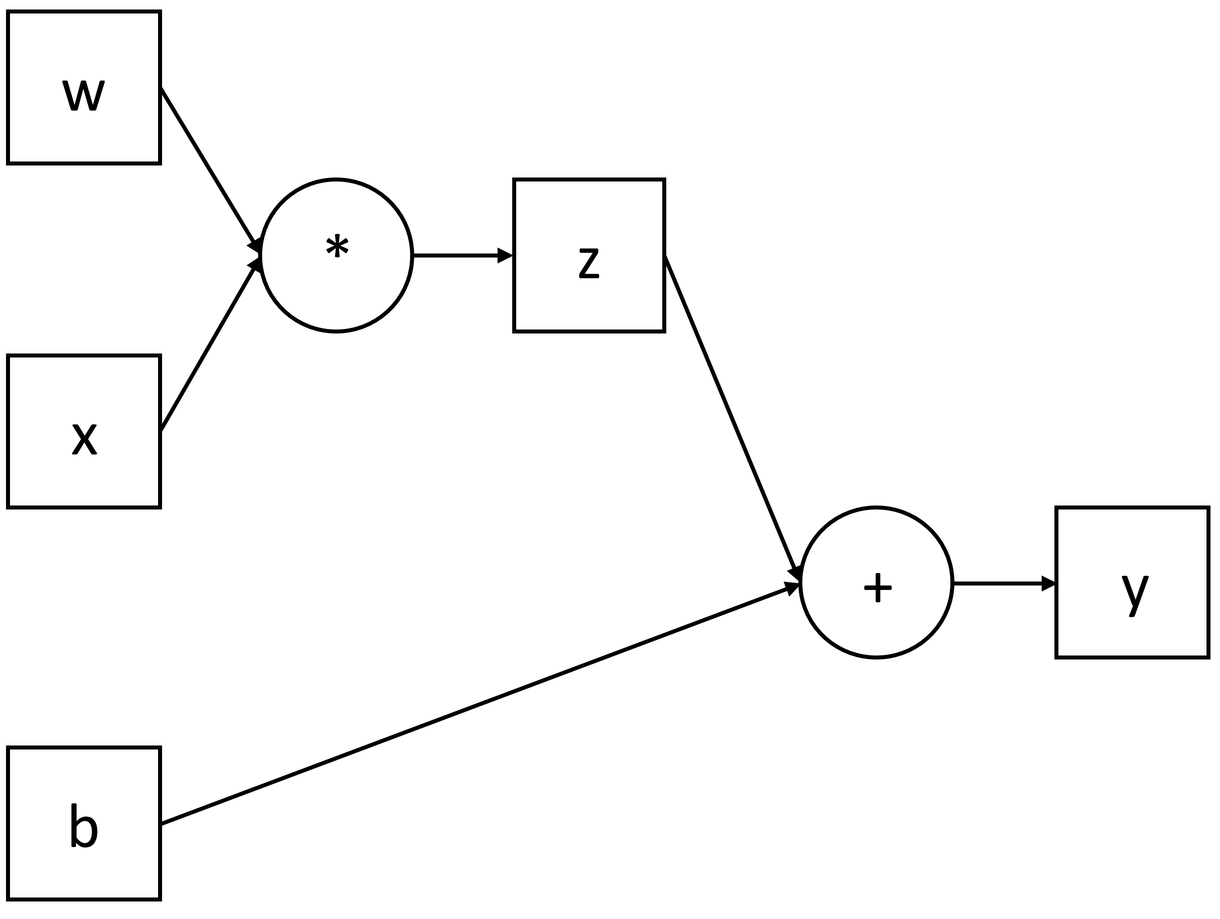
TensorFlow Basics

- Tensor
  - Represents a value produced by an Operation.
- Operation
  - Represents a graph node that performs computation on tensors.



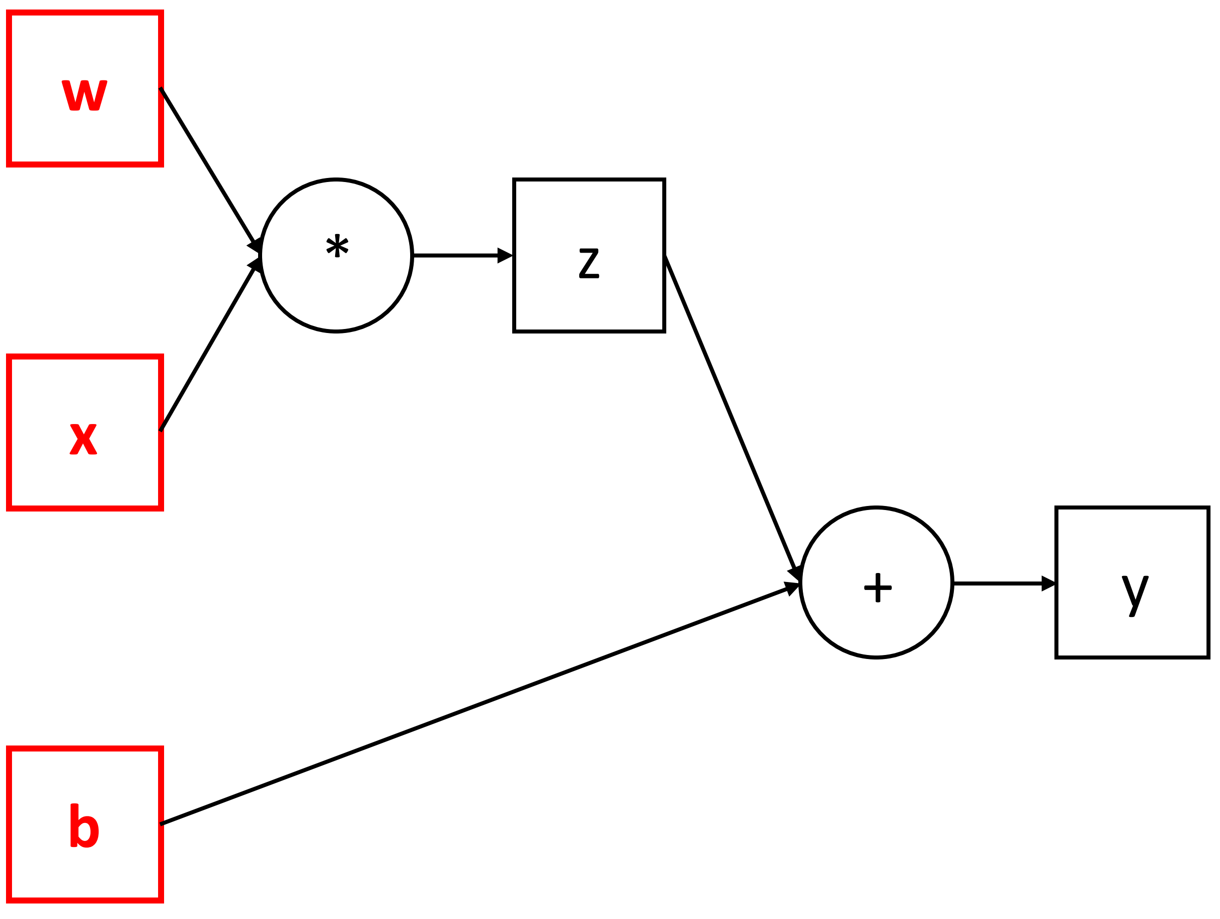
TensorFlow Basics

- Graph example



TensorFlow Basics

- Graph example



TensorFlow Basics

- Constant
  - `tf.constant(value, dtype, …)`
  - **e.g.** `tf.constant([1, 2, 3])`
- Variable
  - `tf.Variable(initial-value, …)`
  - **e.g.** `tf.Variable(tf.zeros(shape=(2,2)))`
- Placeholder
  - `tf.placeholder(dtype, shape, ...)`
  - **e.g.** `tf.placeholder(tf.float32, shape=(10, 10))`

TensorFlow Basics

- Session
  - Encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.
  - When you launch the graph, variables have to be explicitly initialized.
  - Placeholder tensor's value must be fed using the feed_dict optional argument to Session.run(), Tensor.eval(), or Operation.run().

```
with tf.Session(graph=g) as sess:
  tf.initialize_all_variables().run()
  feed_dict = {x: [1]}
  y, z = sess.run([y, z], feed_dict=feed_dict)
```

TensorFlow Basics

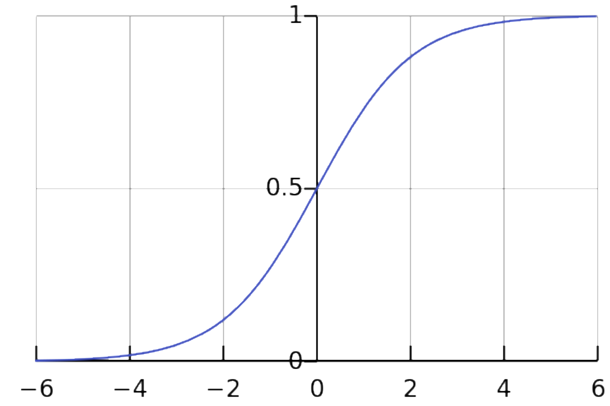
Let’s check the code.

2_tfbasics.ipynb

Softmax Regression

Softmax Regression

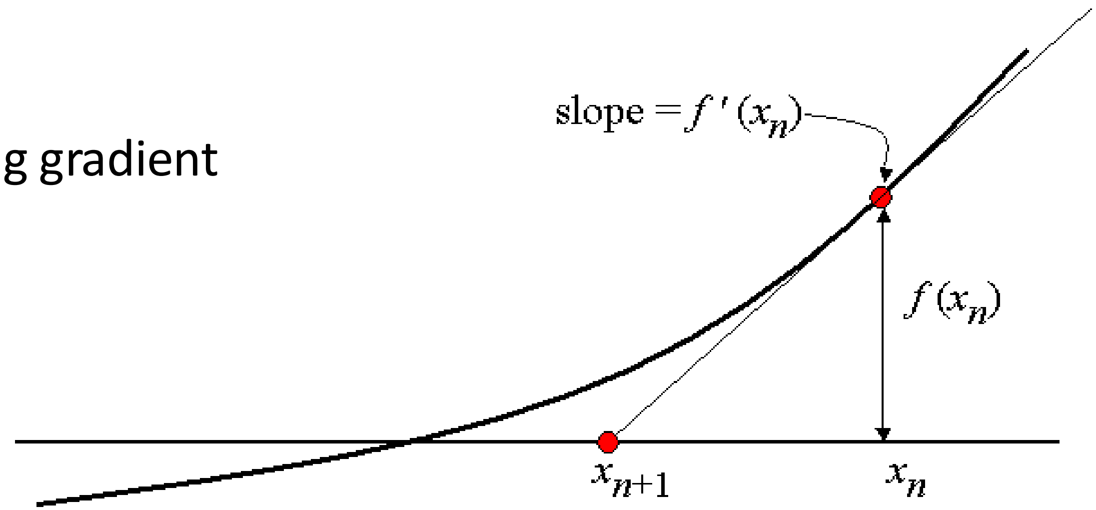
- Binary classification (logistic regression)
  - Logistic function: $\sigma(t) = \frac{e^t}{1+e^t} = \frac{1}{1+e^{-t}}$



  - $t = \alpha + \beta x$
  - $h_\theta(x) = \sigma(\alpha + \beta x) = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}} = \frac{1}{1+e^{-(\alpha+\beta x)}}$
  - May regard $e^{\alpha+\beta x}$ as point for $y = 1$, $\theta = (\alpha, \beta)$ (1 for y = 0)
    - If we normalize (divide) it by sum of all points, we can get the probability.

Softmax Regression

- Two-class classification (logistic regression)
  - Cost function
    - Cross entropy (negative log-likelihood)
    - $J(\theta) = -\sum_{i=1}^{n}(y_i \log h_\theta(x_i) + (1-y_i)\log(1-h_\theta(x_i))$
    - $\hat{\theta}^{MLE} = \underset{\theta}{\operatorname{argmin}} J(\theta)$
  - Need to optimize it using gradient



$$x_{n+1} = x_n - \frac{f'(x_n)}{f(x_n)}$$

Softmax Regression

- Multinomial logistic regression
  - Model

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

  - Cost function

$$\begin{aligned} J(\theta) &= -\left[\sum_{i=1}^{m} (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) + y^{(i)} \log h_\theta(x^{(i)}) \right] \\ &= -\left[\sum_{i=1}^{m} \sum_{k=0}^{1} 1\left\{ y^{(i)} = k \right\} \log P(y^{(i)} = k | x^{(i)}; \theta) \right] \end{aligned}$$

Softmax Regression

- Multinomial logistic regression

$$\left[\theta^{(1)\top}x, \theta^{(2)\top}x, \cdots, \theta^{(K)\top}x\right]$$

Softmax

  - Model

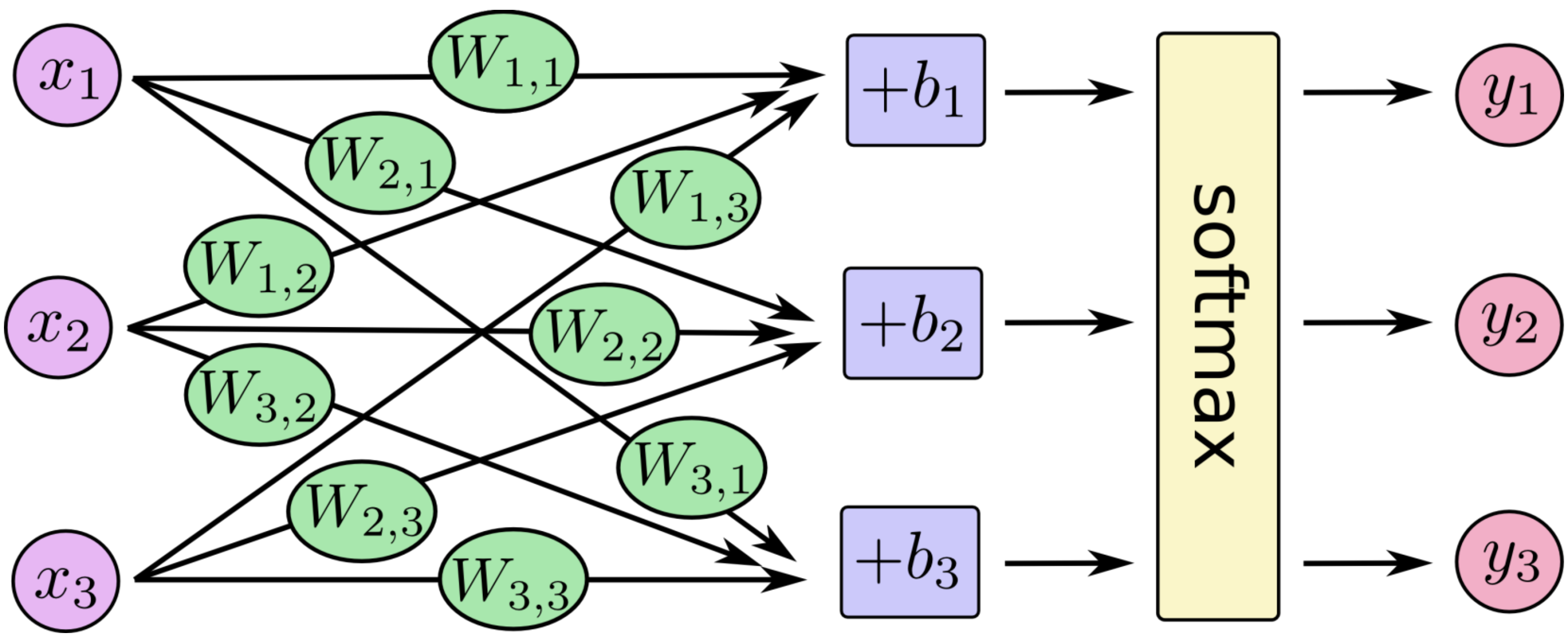
$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top}x)} \begin{bmatrix} \exp(\theta^{(1)\top}x) \\ \exp(\theta^{(2)\top}x) \\ \vdots \\ \exp(\theta^{(K)\top}x) \end{bmatrix}$$

  - Cost function

$$\begin{aligned} J(\theta) &= -\left[\sum_{i=1}^{m} (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) + y^{(i)} \log h_\theta(x^{(i)})\right] \\ &= -\left[\sum_{i=1}^{m} \sum_{k=0}^{1} 1\left\{y^{(i)} = k\right\} \log P(y^{(i)} = k|x^{(i)};\theta)\right] \end{aligned}$$

Softmax Regression

- Multinomial logistic regression



Softmax Regression

- Example: $K = 4$
  - True label: $y = 2$
    - if we represent it as one-hot encoding, $y = [0, 1, 0, 0]$
  - Prediction result
    - $\theta^T x = [-5.0,\ \ 4.0,\ \ 1.0,\ \ -3.0]$
    - $softmax(\theta^T x) = [1.17e-04,\ \ 9.51e-01,\ \ 4.73e-02,\ \ 8.67e-04]$
    - $log\big(softmax(\theta^T x)\big) = [-9.04,\ \ -0.05,\ \ -3.05,\ \ -7.05]$
  - Cost function
    - $-y * log\big(softmax(\theta^T x)\big) = [0, 1, 0, 0] * [9.04,\ \ 0.05,\ \ 3.05,\ \ 7.05]$

Softmax Regression

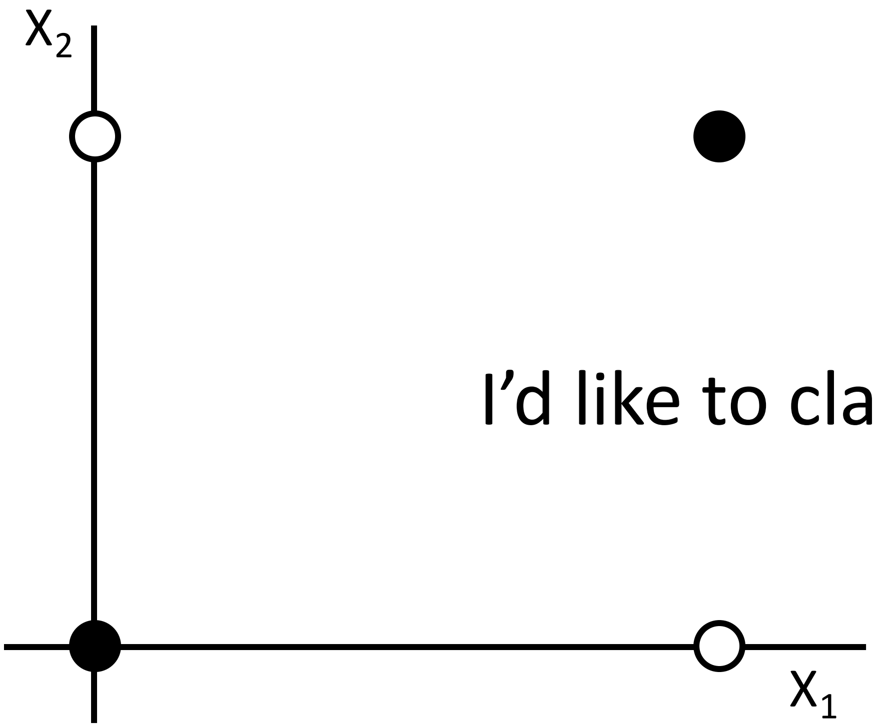
Let’s check the code.

3_softmax_nn.ipynb

Neural Network

Neural Network

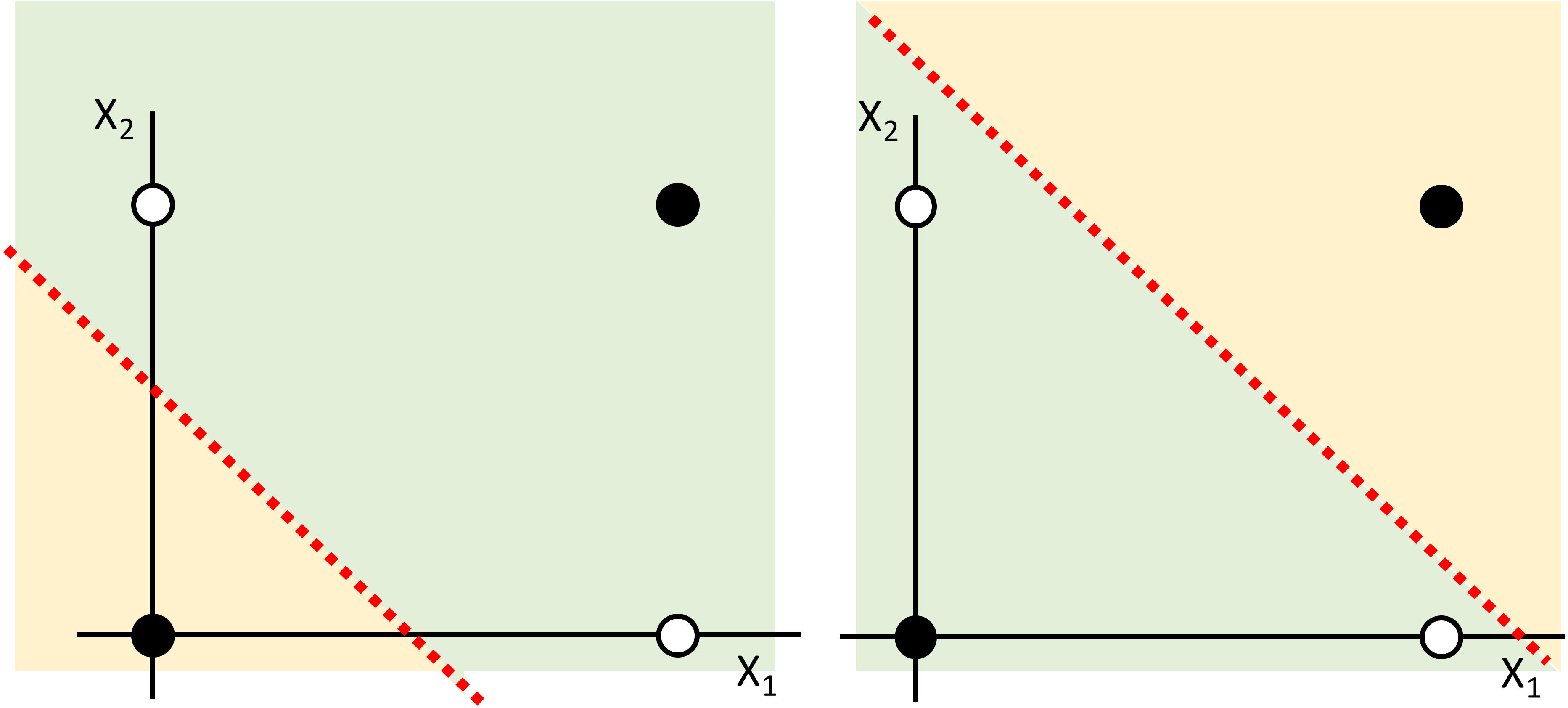
- XOR Problem



I'd like to classify 0(●) and 1(○).

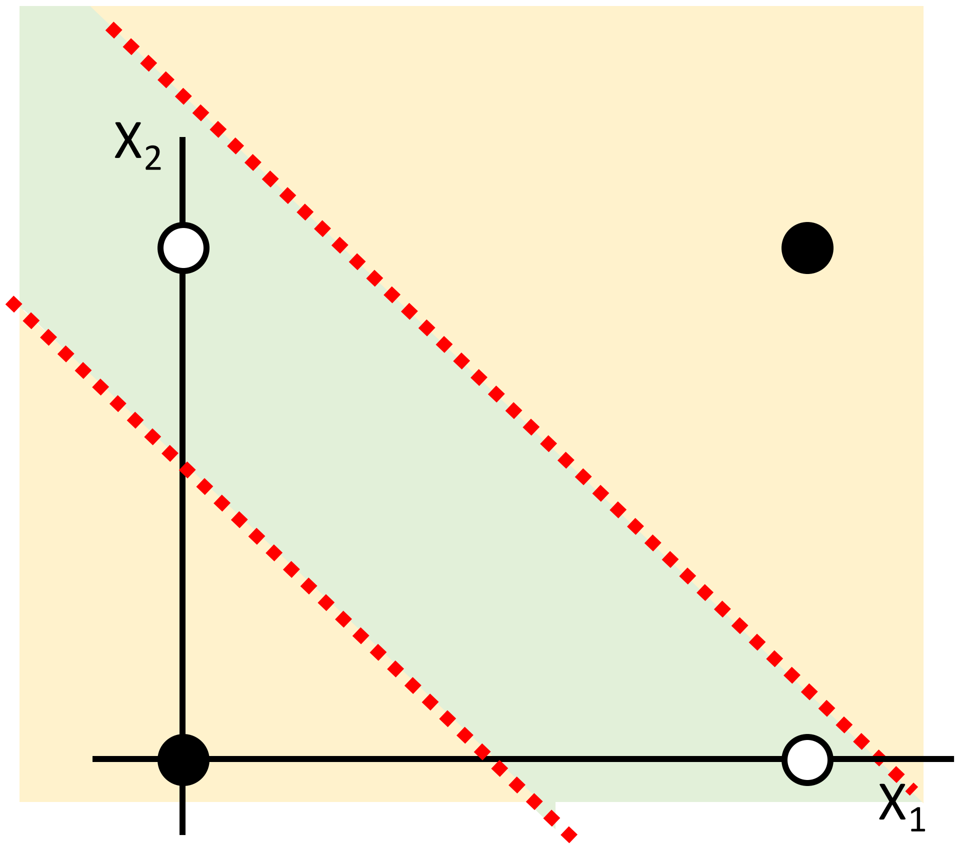
Neural Network

- If only with linear classifier (e.g. logistic regression model), …



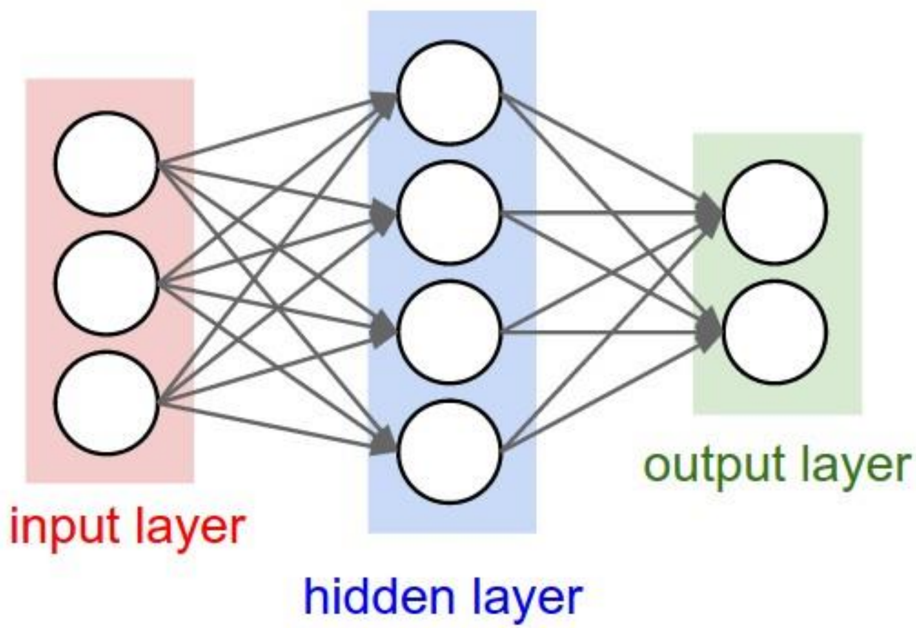
Neural Network

- But if we combine them, ...



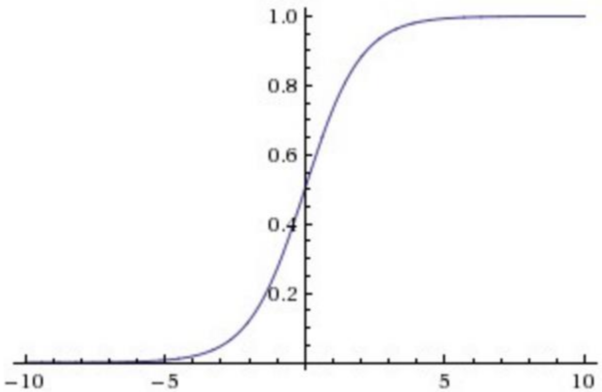
Neural Network

- Put hidden layer between input and output layer!
- Consists of fully-connected layers in which neurons between two adjacent layers are fully pairwise connected.
- Neurons within a single layer share no connections.

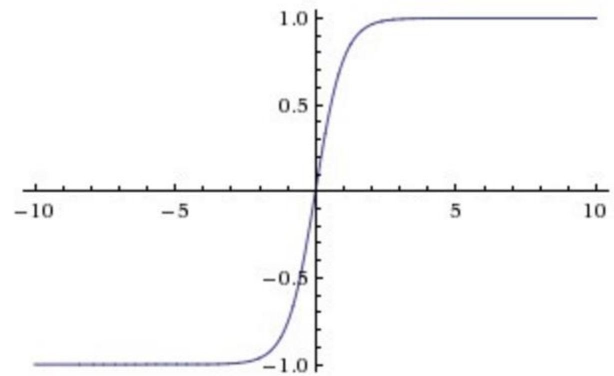


Neural Network

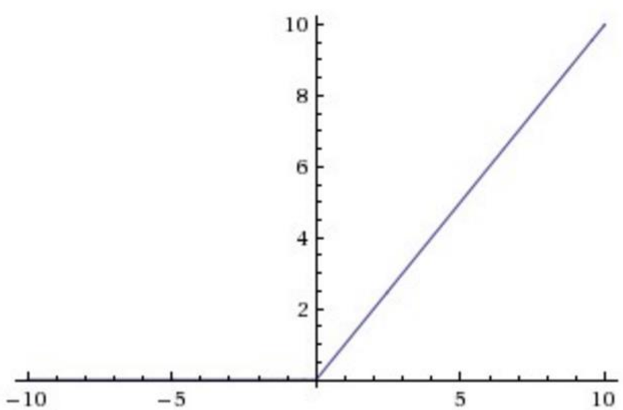
- Activation Functions



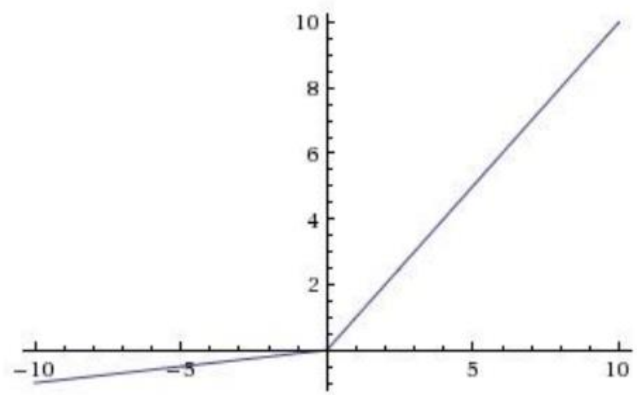
Sigmoid



tanh



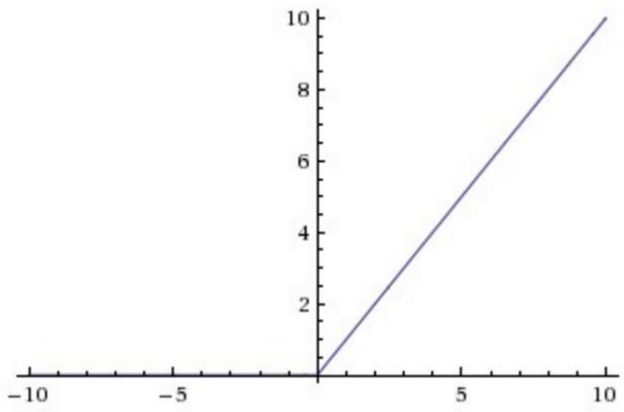
ReLU



Leaky ReLU

Neural Network

- `tf.nn.relu(features, …)`




```
with graph.as_default():
  ...

  weights_1 = tf.Variable(tf.truncated_normal([image_size
  biases_1 = tf.Variable(tf.zeros([size_of_hidden]))
  logits_1 = tf.matmul(tf_train_dataset, weights_1) + bi
  output_1 = tf.nn.relu(logits_1)

  ...
```

Neural Network

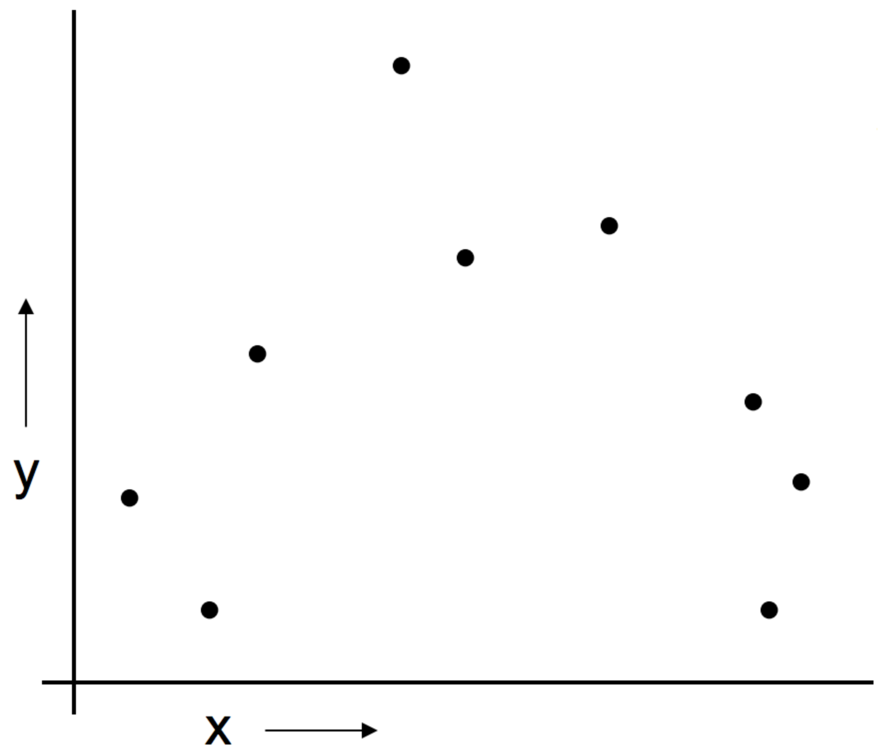
Let’s check the code again.

3_softmax_nn.ipynb

Regularization

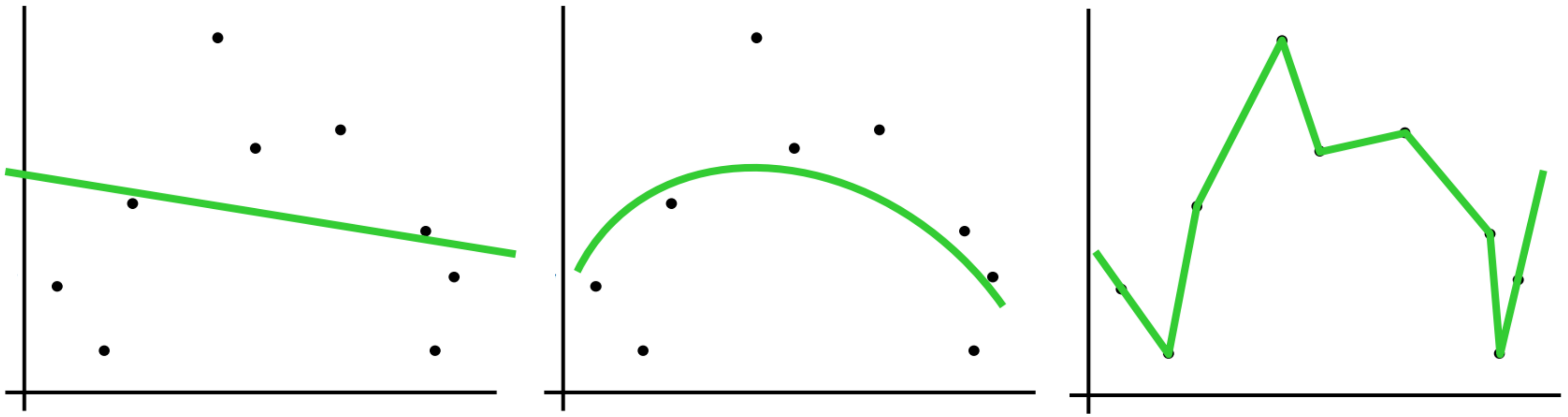
Regularization

- Let's make the model which explains the data below.
- $y = f(x) + \epsilon$



Regularization

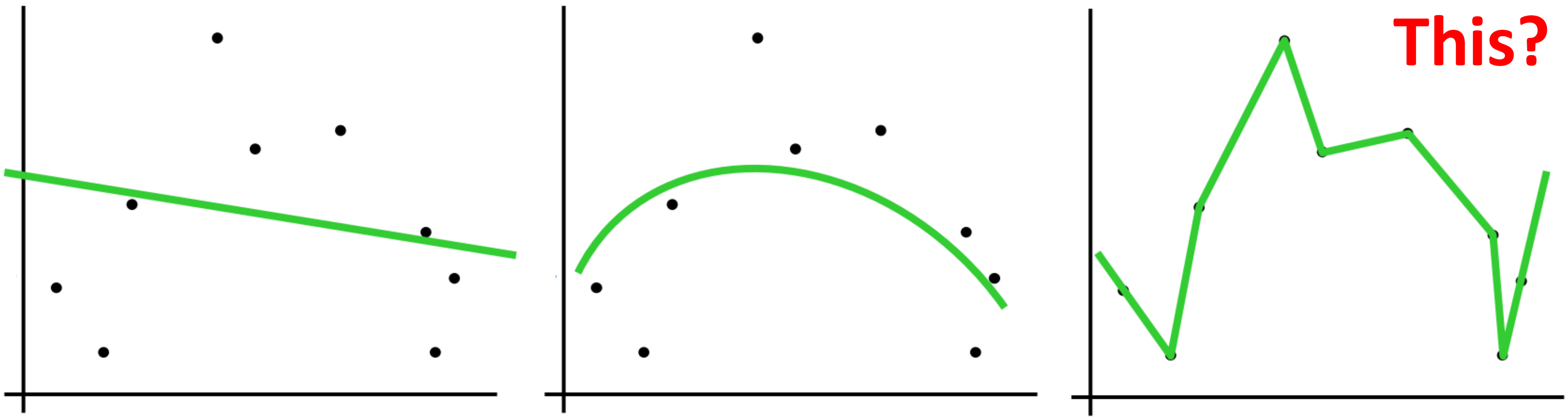
- Three candidates



Need to choose one of them… Which one is the best?

Regularization

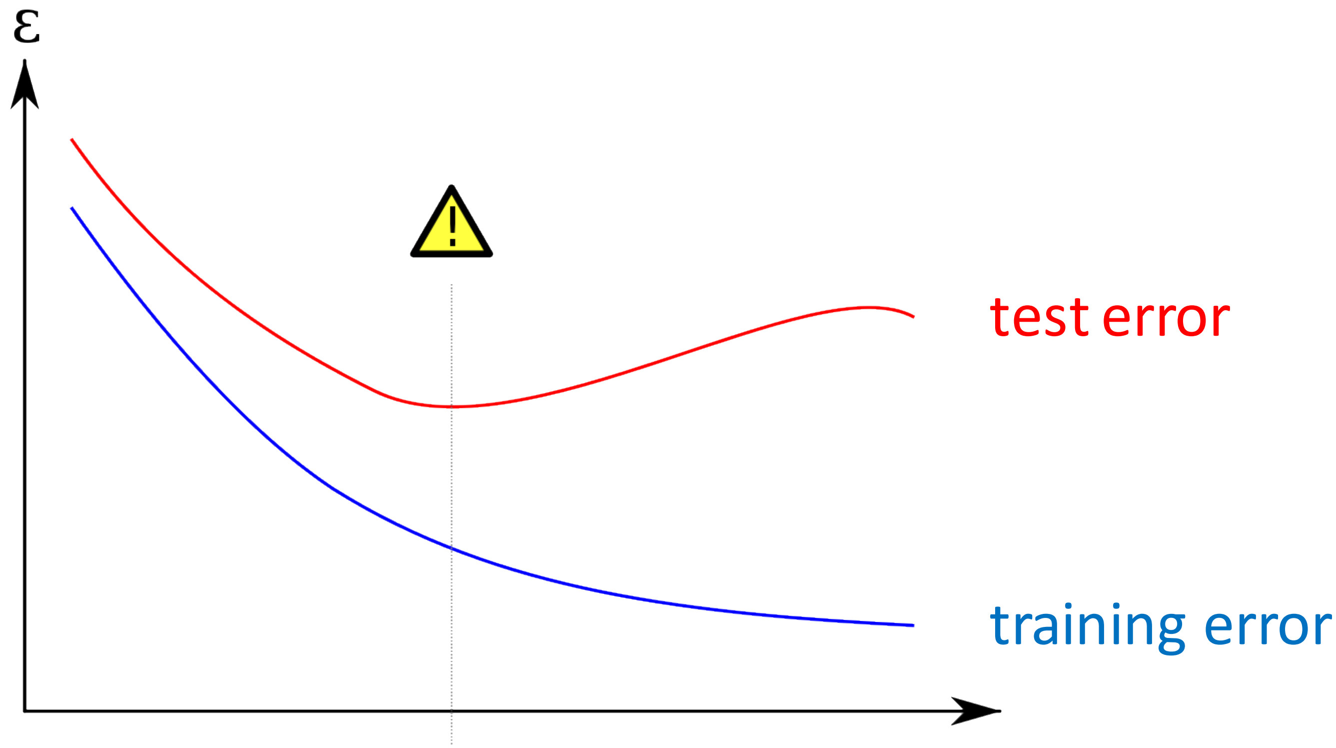
- Three candidates



Need to choose one of them… Which one is the best?

Regularization

- Overfitting problem



Regularization

- Several methods to avoid overfitting problem
    - L2 regularization
    - Dropout
    - …

Regularization

- L2 regularization
  - Add L2 penalty (λw^2) to cost function
  - `tf.nn.l2_loss(t, …)`

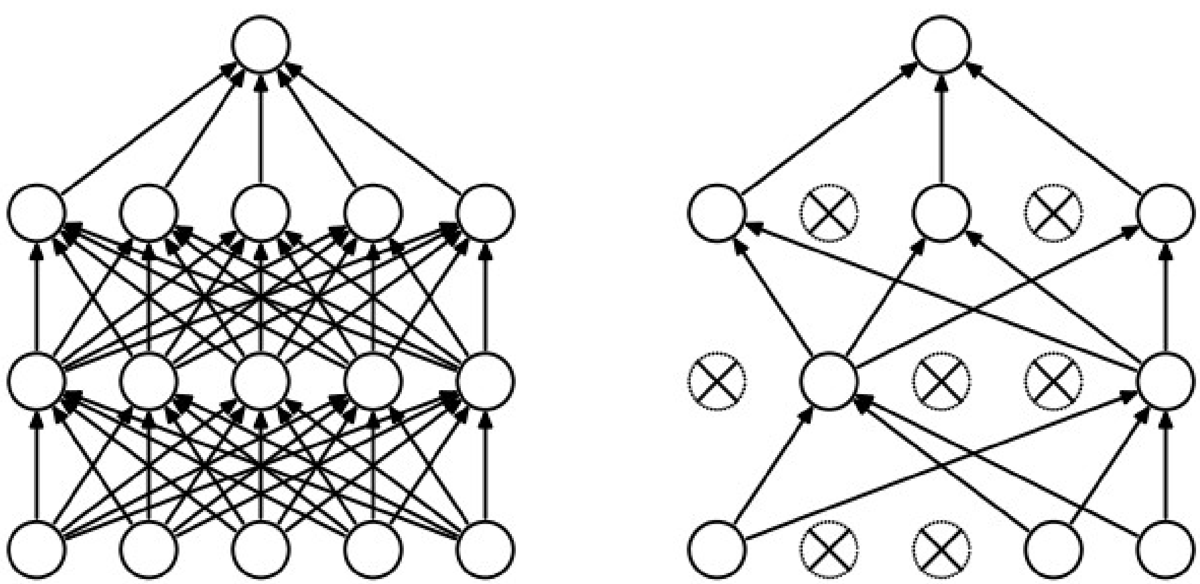
```
with graph.as_default():
  ...

  loss = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(logits,
    + l2_lambda * tf.nn.l2_loss(weights)

  ...
```

Regularization

- Dropout
  - Sampling a Neural Network within the full Neural Network, and only updating the parameters of the sampled network based on the input data



(a) Standard Neural Net (b) After applying dropout.

Regularization

- Dropout
  - `tf.nn.dropout(x, keep_prob, …)`
  - `keep_prob`: The probability that each element is kept

```
with graph.as_default():
  ...

  weights_1 = tf.Variable(tf.truncated_normal([image_size
  biases_1 = tf.Variable(tf.zeros([size_of_hidden]))
  logits_1 = tf.matmul(tf_train_dataset, weights_1) + bia
  output_1 = tf.nn.relu(logits_1)

  dropped_output_1 = tf.nn.dropout(x=output_1,
                                   keep_prob=keep_prob)

  ...
```

Learning Rate

- When training a model, it is often recommended to lower the learning rate as the training progresses.
- Here, we apply exponential decay function to a initial learning rate.

$$learning\_rate = initial\_learning\_rate * decay\_rate^{global\_step/decay\_steps}$$

```
with graph.as_default():
  ...

  global_step = tf.Variable(0)
  learning_rate = tf.train.exponential_decay(learning_rate=initial_le
                                             global_step=global_step,
                                             decay_steps=decay_steps,
                                             decay_rate=decay_rate)
  optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimi

  ...
```

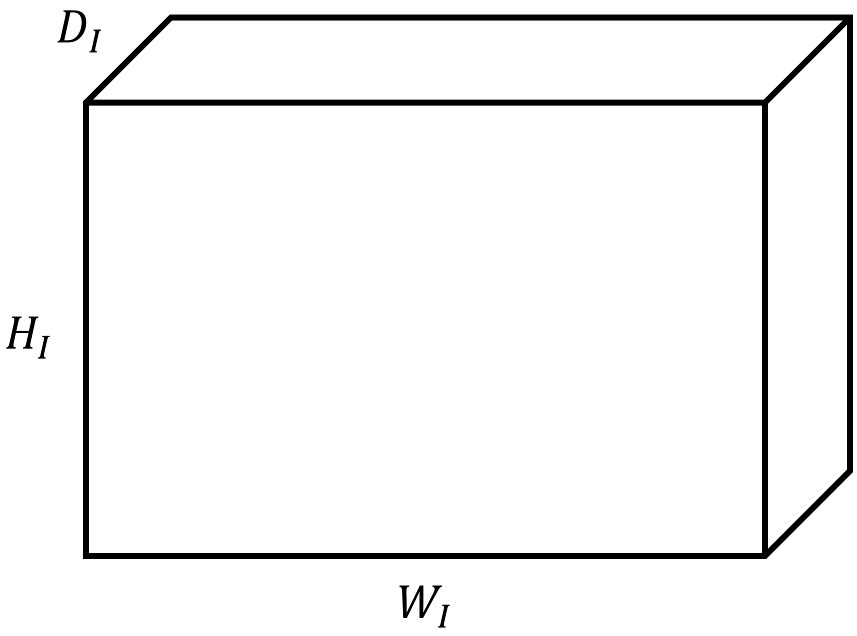
Regularization

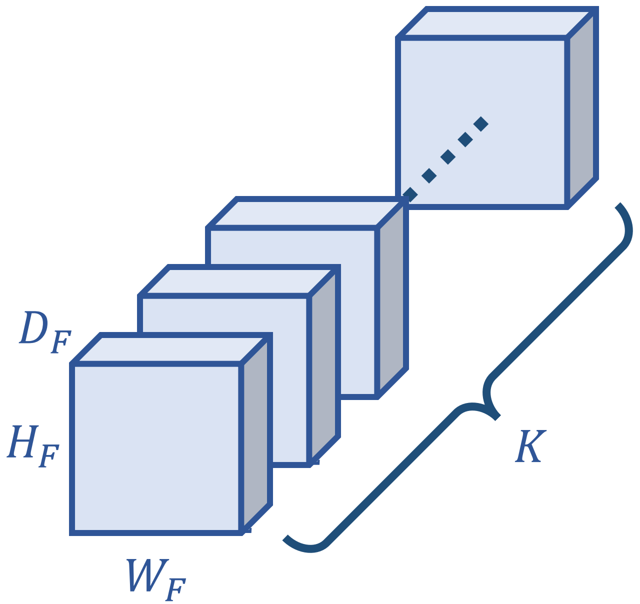
Let’s check the code.

4_regularization.ipynb

Convolutional Neural Network

Convolution

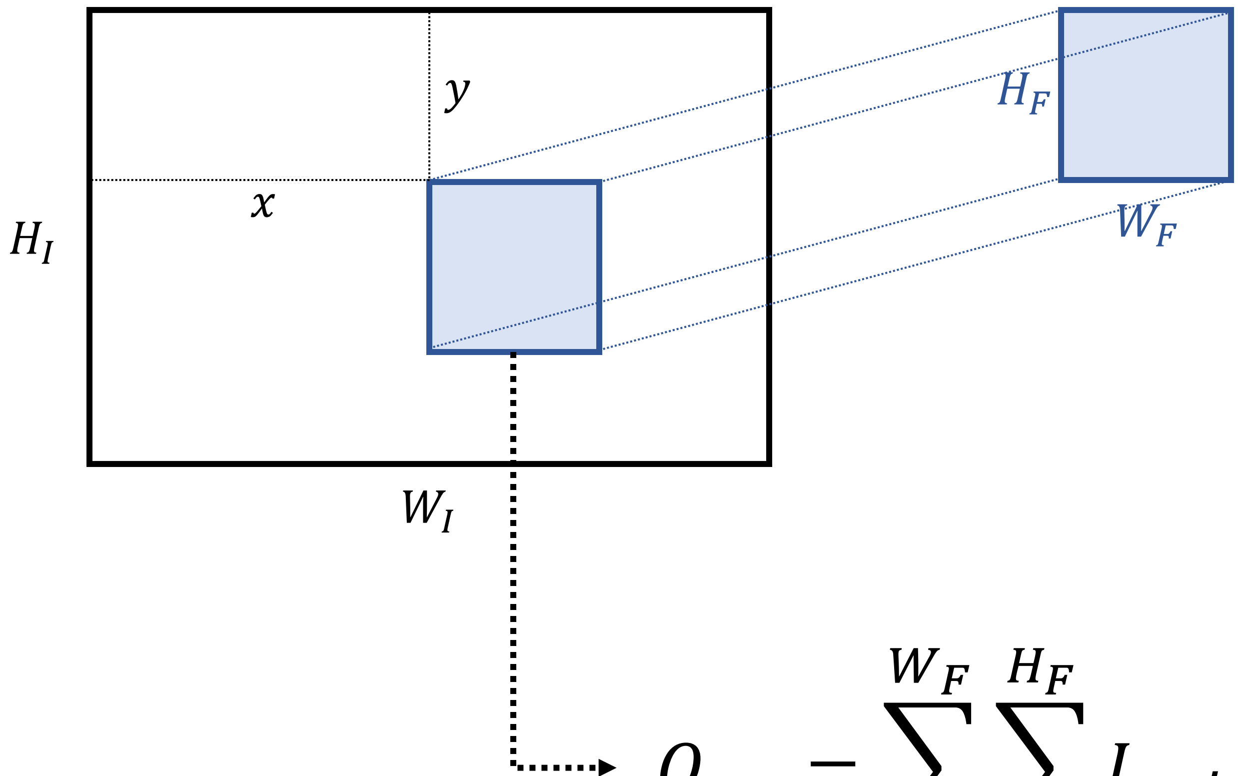




Input I: $[W_I, H_I, D_I] \in \mathbb{R}^3$

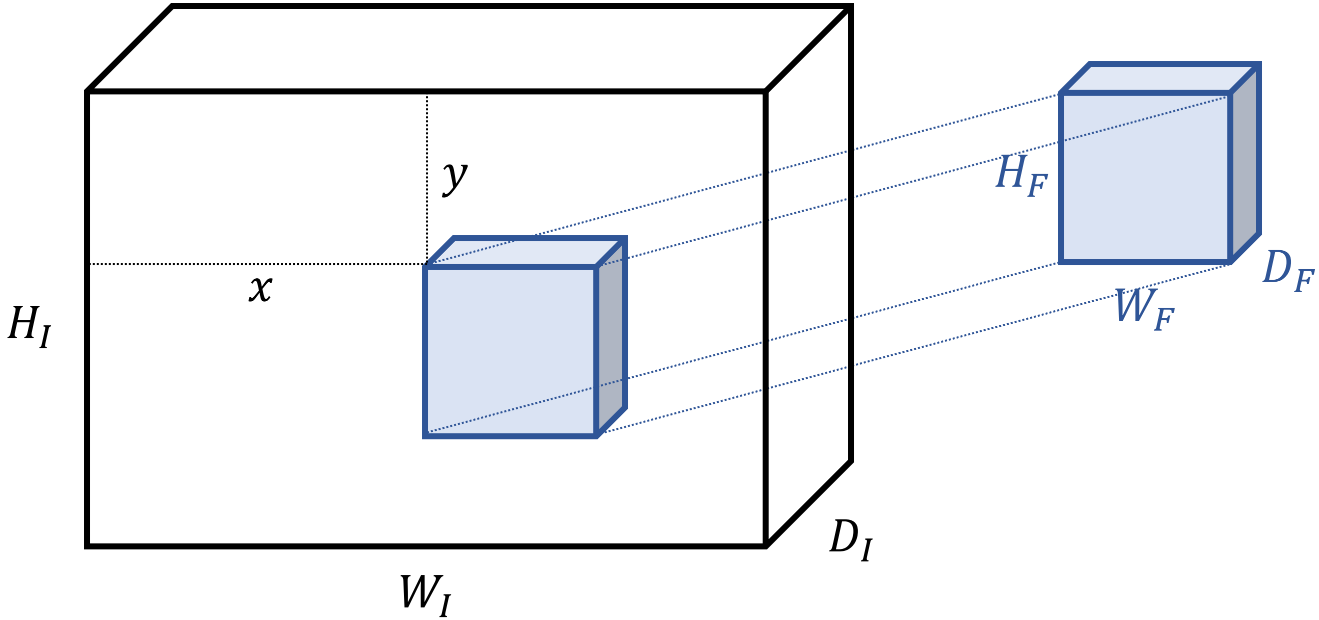
Filter F: $[W_F, H_F, D_F, K] \in \mathbb{R}^4$

Convolution



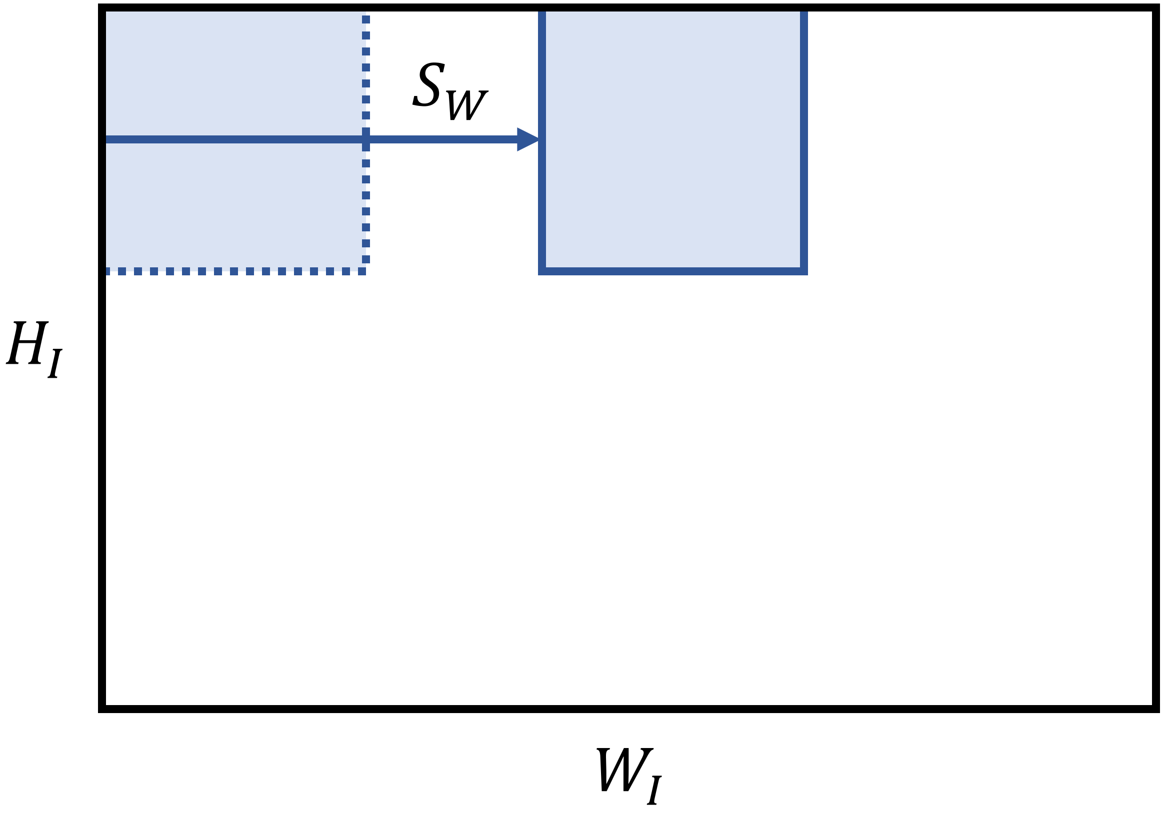
$$O_{x,y} = \sum_{i}^{W_F} \sum_{j}^{H_F} I_{x+i,y+j} \times F_{i,j}$$

Convolution



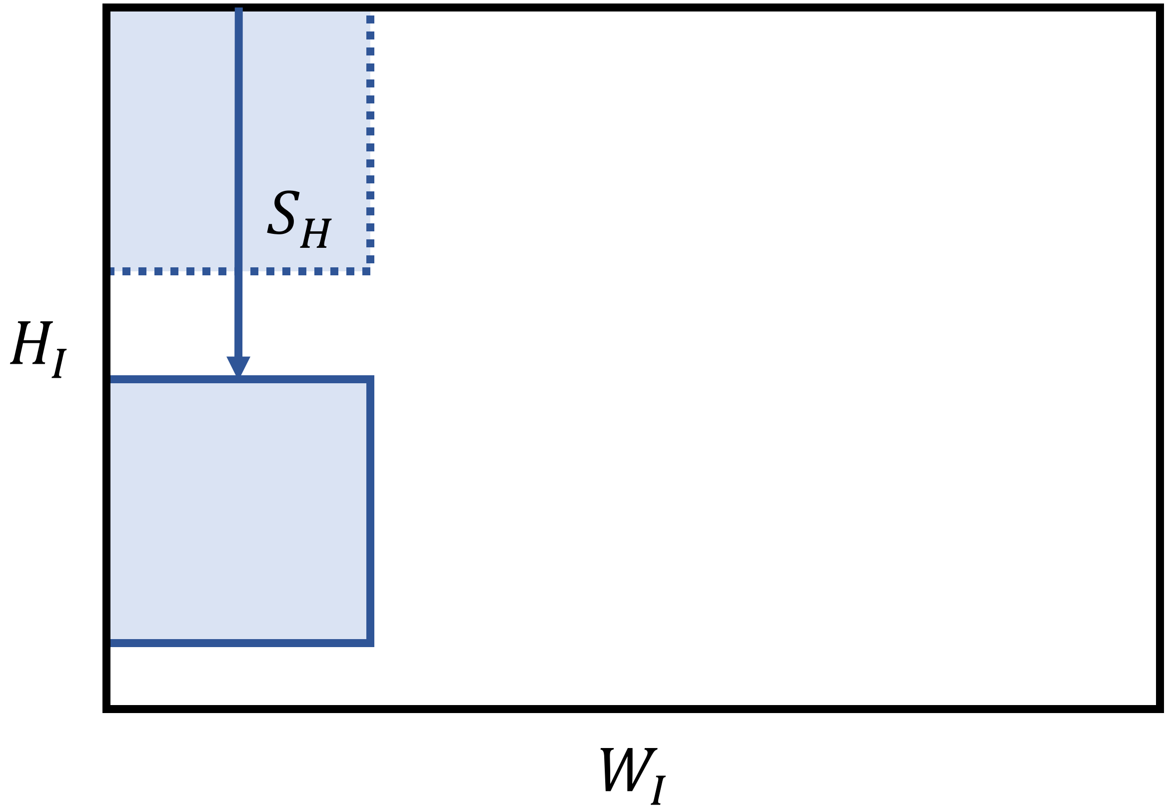
Convolution

- Stride



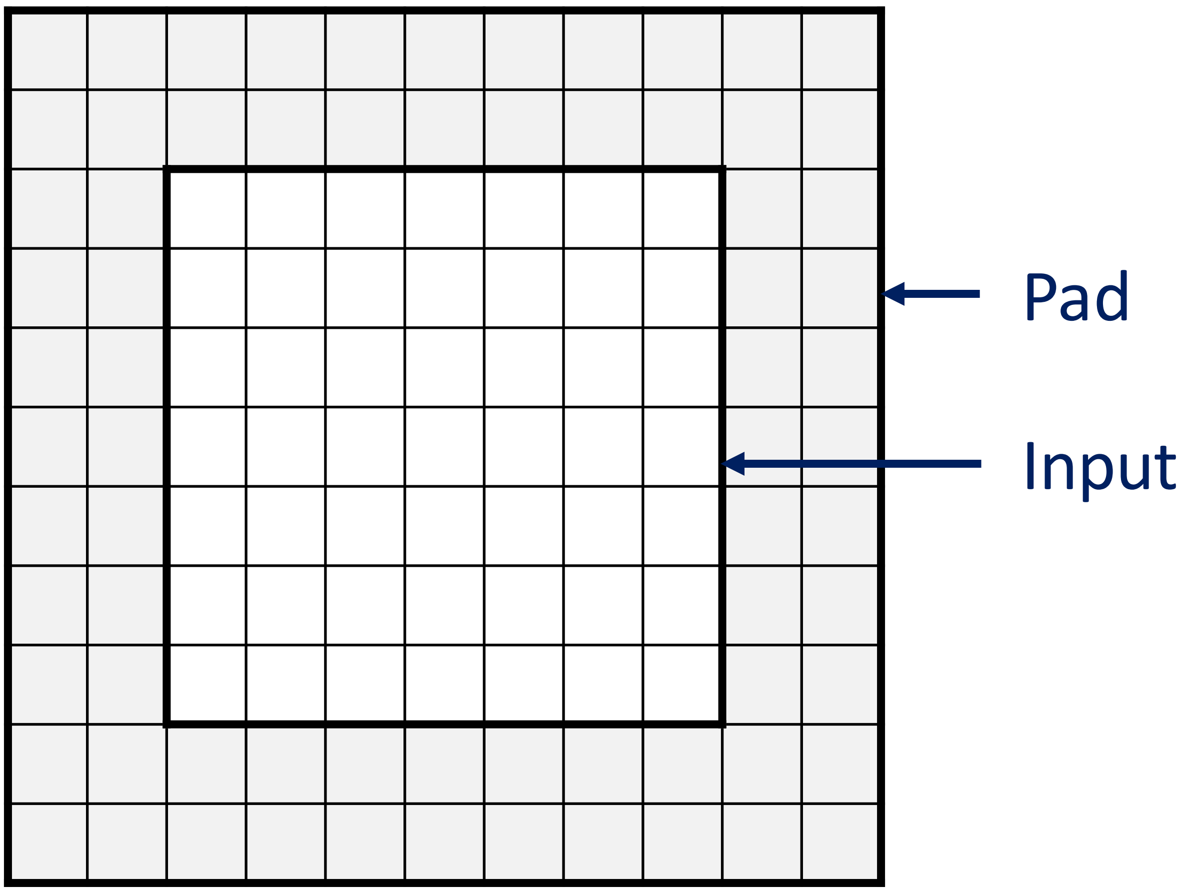
Convolution

- Stride



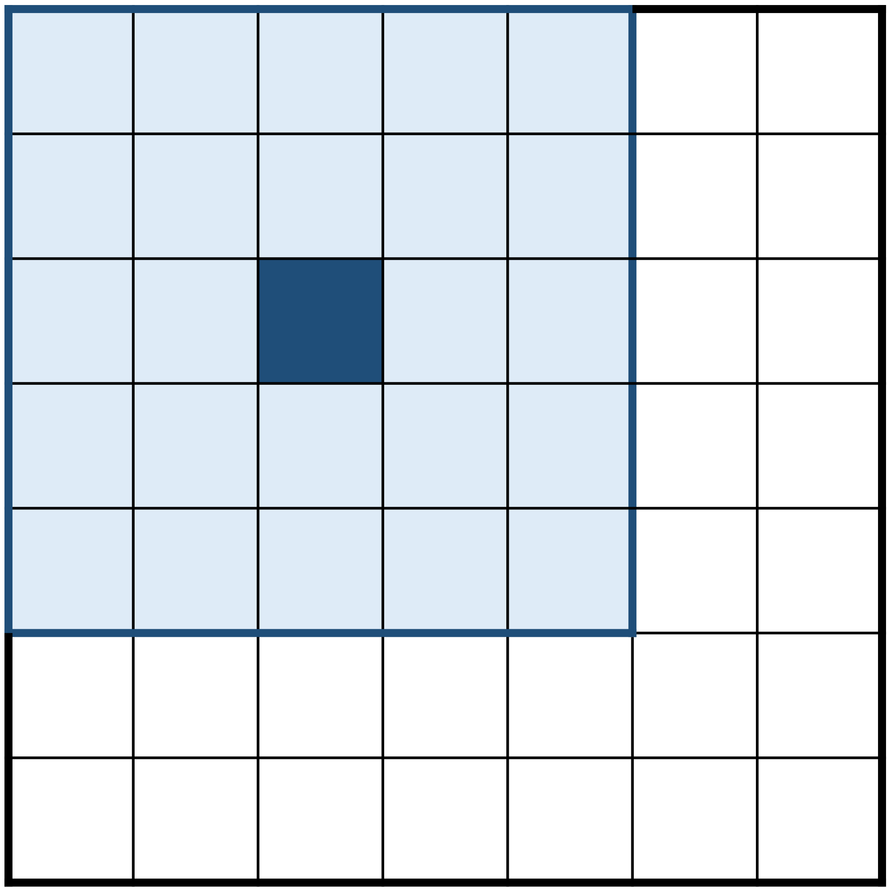
Convolution

- Padding



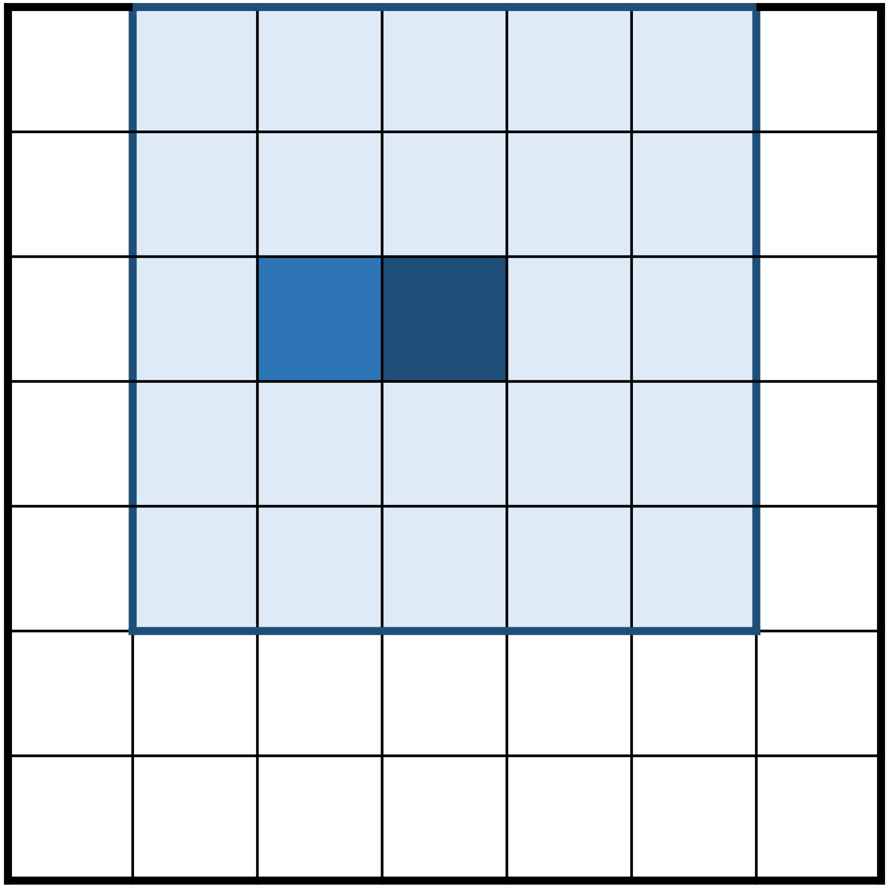
Convolution

- Padding



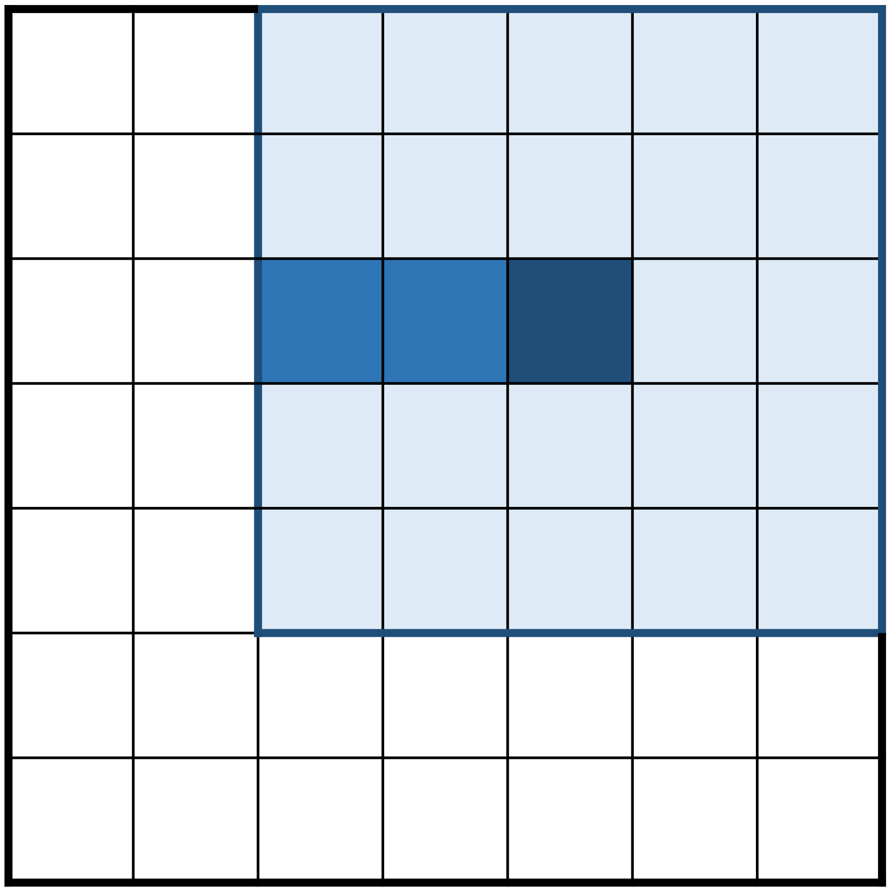
Convolution

- Padding



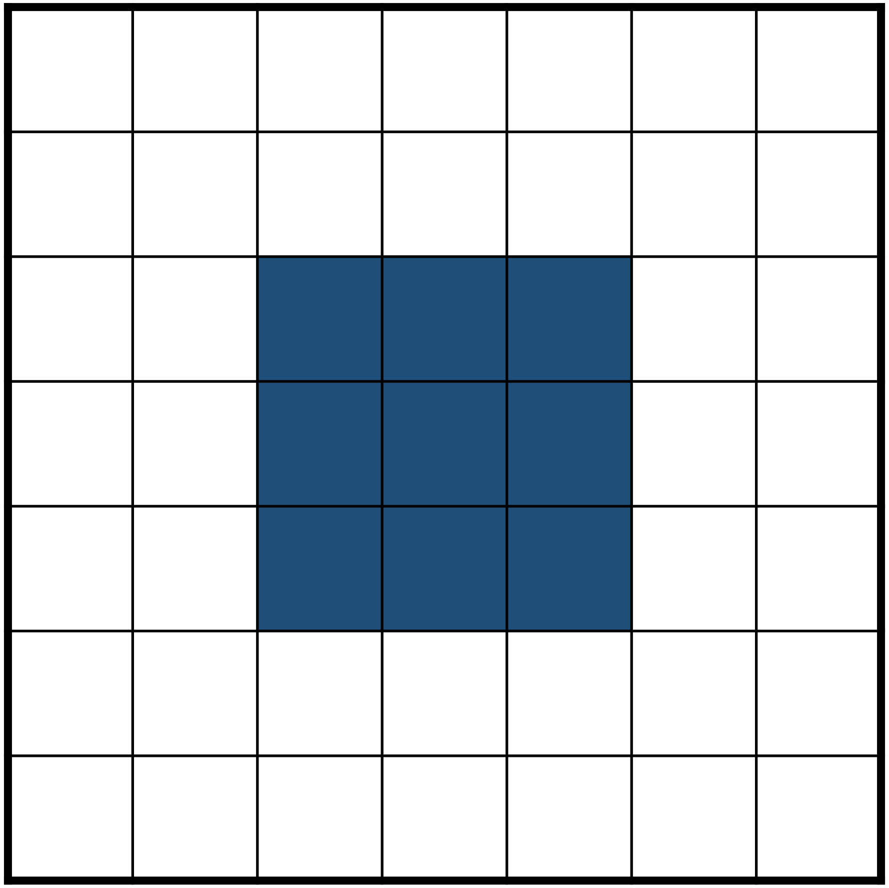
Convolution

- Padding



Convolution

- Padding



Convolution

- Padding

0	0	0	0
0	0	0	0
0	1	4	0
0	2	5	0
0	3	6	0
0	0	0	0
0	0	0	0

Zero

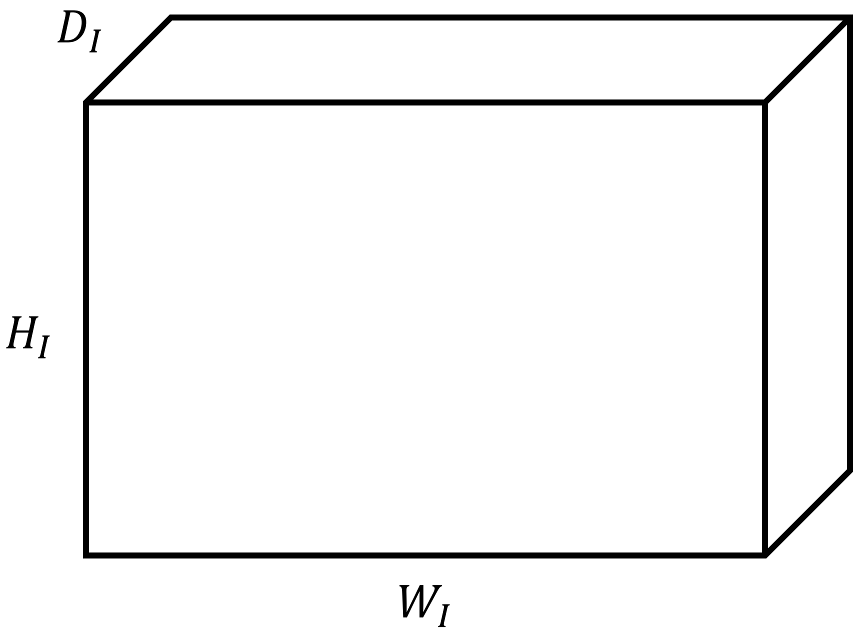
6	3	6	3
5	2	5	2
4	1	4	1
5	2	5	2
6	3	6	3
5	2	5	2
4	1	4	1

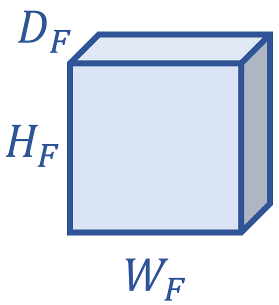
Reflect

2	2	5	5
1	1	4	4
1	1	4	4
2	2	5	5
3	3	6	6
3	3	6	6
2	2	5	5

Symmetric

Pooling

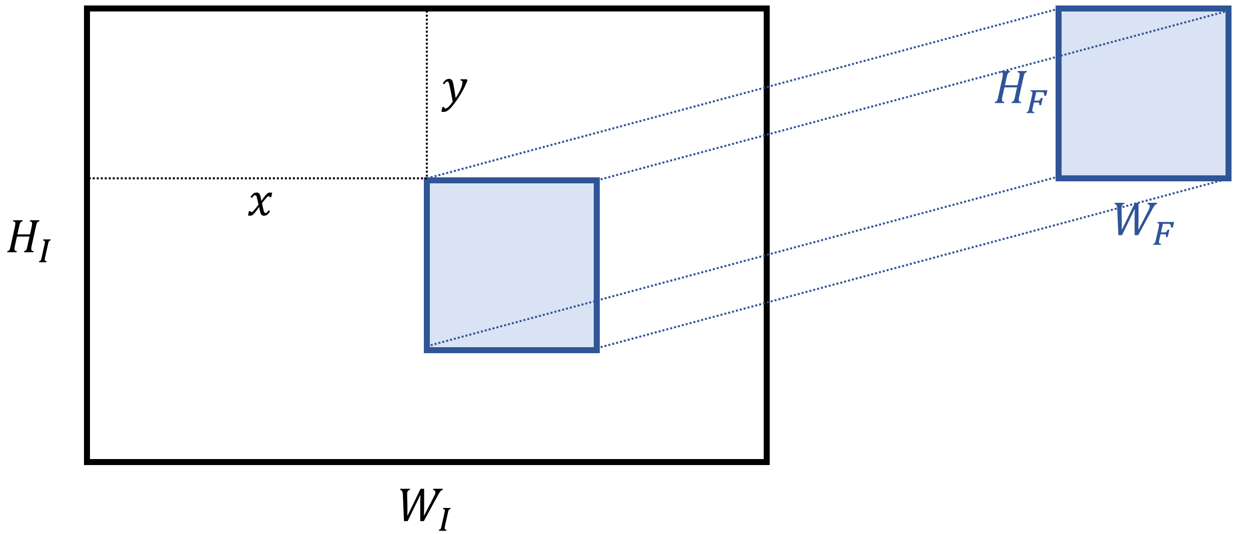




Input I: $[W_I, H_I, D_I] \in \mathbb{R}^3$

Filter F: $[W_F, H_F, D_F] \in \mathbb{R}^3$

Pooling



Max: $$O_{x,y} = \max_{i,j} I_{x+i,y+j}$$

Average: $$O_{x,y} = \frac{1}{W_F \times H_F} \sum_{i}^{W_F} \sum_{j}^{H_F} I_{x+i,y+j}$$

Convolutional Neural Network

- `tf.nn.conv2d(input, filter, strides, padding, …)`
  - `input`: 4-D Tensor with shape `[batch, in_height, in_width, in_channels]`
  - `filter`: 4-D Tensor with shape `[filter_height, filter_width, in_channels, out_channels]`
  - `stride`: The stride of the sliding window for each dimension of input.
  - `padding`: "`SAME`" or "`VALID`"

Convolutional Neural Network

- `tf.nn.conv2d(input, filter, strides, padding, …)`

```
with graph.as_default():
  ...

  layer1_weights = tf.Variable(tf.truncated_normal([patch_si
  layer1_biases = tf.Variable(tf.zeros([out_channel]))


  ...

  def model(data):
    conv = tf.nn.conv2d(data, filter=layer1_weights, strides
    hidden = tf.nn.relu(conv + layer1_biases)

    pool = tf.nn.max_pool(hidden, ksize=[1, 2, 2, 1], stride
    dropped = tf.nn.dropout(x=pool, keep_prob=keep_prob)

  ...
```

Convolutional Neural Network

- `tf.nn.max_pool(value, ksize, strides, padding, …)`
  - `value`: 4-D Tensor with shape `[batch, height, width, channels]`
  - `ksize`: The size of the window for each dimension of the input tensor
  - `strides`: The stride of the sliding window for each dimension of the input tensor
  - `padding`: `"SAME"` or `"VALID"`

Convolutional Neural Network

- `tf.nn.max_pool(value, ksize, strides, padding, …)`

```
with graph.as_default():
  ...

  layer1_weights = tf.Variable(tf.truncated_normal([patch_si
  layer1_biases = tf.Variable(tf.zeros([out_channel]))

  ...

  def model(data):
    conv = tf.nn.conv2d(data, filter=layer1_weights, strides
    hidden = tf.nn.relu(conv + layer1_biases)

    pool = tf.nn.max_pool(hidden, ksize=[1, 2, 2, 1], stride
    dropped = tf.nn.dropout(x=pool, keep_prob=keep_prob)

  ...
```

Convolutional Neural Network

Let’s check the code.

5_cnn.ipynb

Saving and Restoring

Saving and Restoring

- The easiest way to save and restore a model is to use a `tf.train.Saver` object. The constructor adds `save` and `restore` ops to the graph for all, or a specified list, of the variables in the graph.
- Saving

```
with graph.as_default():
   ...
   saver = tf.train.Saver()

with tf.Session(graph=graph) as session:
   ...
   saver.save(session, ckpt_path, global_step=step)
```

Saving and Restoring

- The easiest way to save and restore a model is to use a `tf.train.Saver` object. The constructor adds `save` and `restore` ops to the graph for all, or a specified list, of the variables in the graph.
- Restoring

```
with graph.as_default():
  ...
  saver = tf.train.Saver()

with tf.Session(graph=graph) as session:
  ...
  saver.restore(session, ckpt_path)
```

Saving and Restoring

Let's check the code.

6_save_restore.ipynb

Reference

- TensorFlow official webpage

  https://www.tensorflow.org/

- Stanford CS class - CS231n

  http://cs231n.github.io/

- Udacity - Deep learning course

  https://www.udacity.com/course/deep-learning--ud730

Q & A

Thank You!