CHAPTER 11: CNN

Theory

In this chapter, we will study CNN (Convolutional Neural Network) which is the base of Deep Learning. CNN is widely used for classification and segmentation problems such as image classification, object detection and so on.

Recap

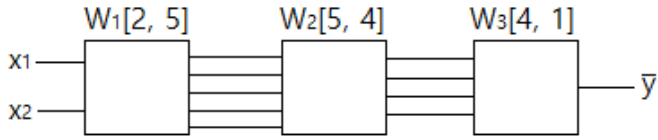
Let's remember the contents of the previous chapter. The most important thing in the last chapter is to have an understanding of the limitation of Sigmoid function and the ReLU function which can be used as the activation function.

The limitation of Sigmoid

Let's consider XOR problem again we discussed the previous chapter.

X1	X2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

And we can implement the 3-layers NN for XOR simply as following structure.

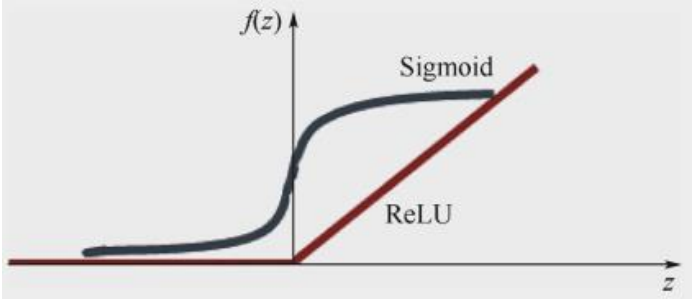


In this NN, we call the first layer as **Input Layer**, and call the last layer as **Output Layer**, and call middle layers as **Hidden Layer**.

But we could confirm that the accuracy was dropdown to 0.5 when the number of hidden layers were increasing.

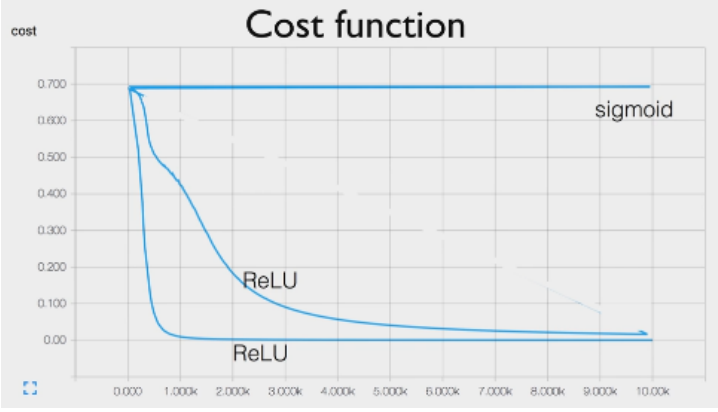
ReLU for Activation Function

To avoid the above issue, we introduce a new function which is called as **ReLU** (Rectified Linear Unit).



ReLU(x) = max(0, x)

We have used ReLU function instead of the sigmoid, and could get the 1.0 accuracy.



Convolutional Neural Network

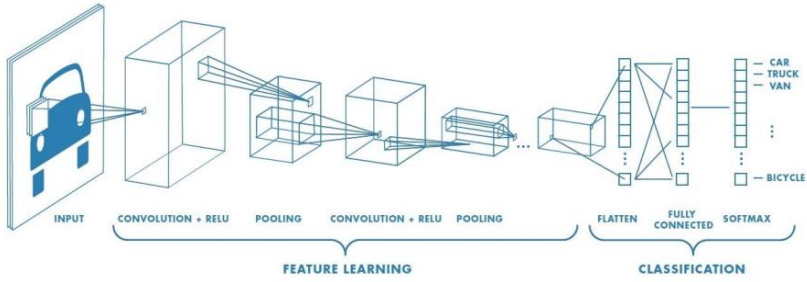
To address this problem, bionic convolutional neural networks are proposed to reduce the number of parameters and adapt the network architecture specifically to vision tasks.

Convolutional neural networks are usually composed of a set of layers that can be grouped by their functionalities.

Let's consider simple image classification problem.

The NN model should be select the appropriate object depends on the inputted image. For example, if the "car" image is inputted into the model then the output related to "car" should be highest.

This model can be constructed as the CNN model which is the combination of convolution layers as shown below architecture.



As we can know from the architecture, in general, CNN is constructed with Convolution Layer, Activation Layer, Pooling Layer, and Fully Connected Layer.

Convolution Layer

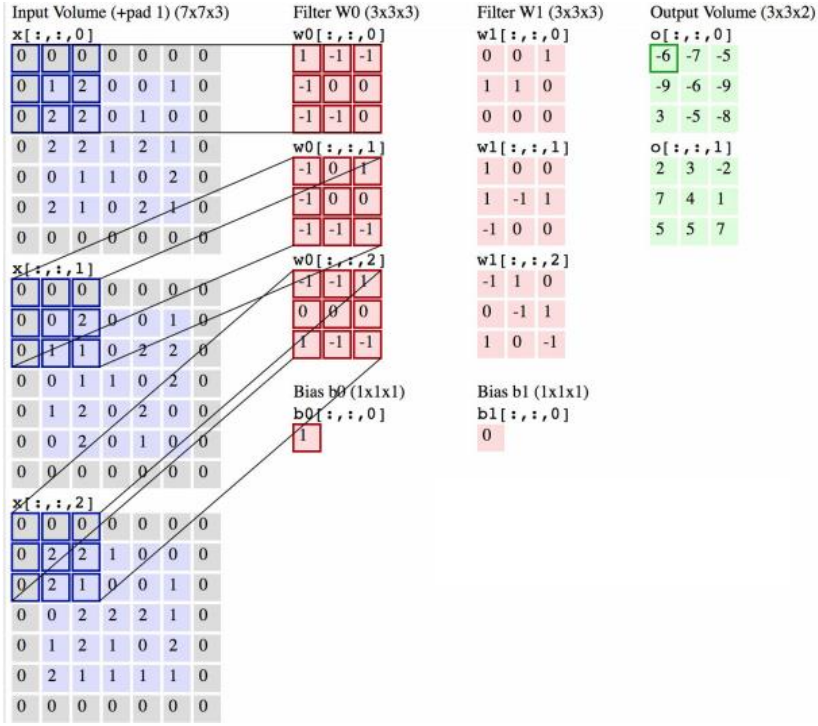
The basic idea of Convolution Layer are:

- Pick a 3*3 matrix F of weights
- Slide this over an image and compute the "Inner Product" (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation.

And the key points are:

- Different convolutions extract different types of low-level “features” from the image

- All that we need to vary to generate these different features is the weights of F



The process is a 2D convolution on the inputs and the “dot products” between weights and inputs are “integrated” across “channels”.

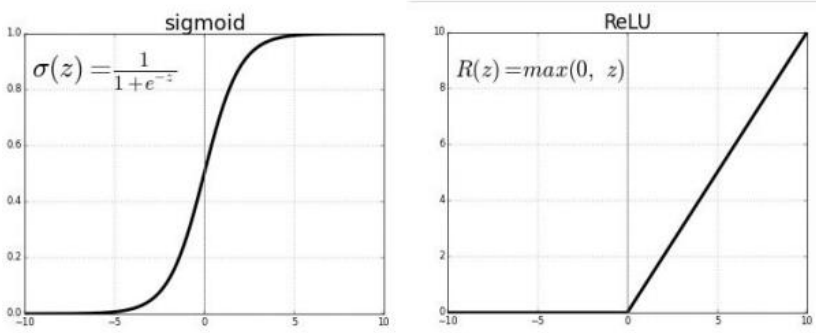
Filter weights are shared across receptive fields and the filter has the same number of layers as input volume channels, and output volume has the same “depth” as the numbers of the filter.

Activation Layer

In order to increase the nonlinearity of the network without affecting receptive fields of convolution layers, the Activation Layer is used.

Several functions such as ReLU, LeakyReLU, Sigmoid can be used as the activation function.

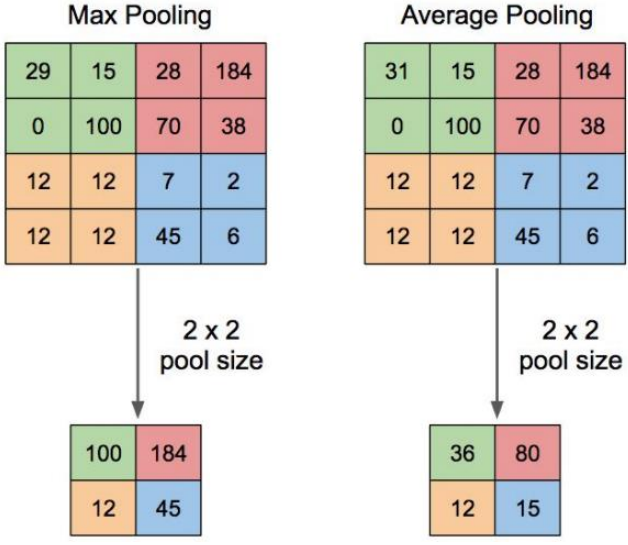
And ReLU is widely used because the result is faster for training, and LeakyReLU addresses the vanishing gradient problem.



And as the special kind of activation layer, Softmax is usually used at the end of FC (Fully-connected) layer outputs.

Pooling Layer

Convolution layers provide activation maps and the Pooling layer applies non-linear down-sampling on activation maps.



There are several kinds of pooling methods such as max pooling, average pooling and so on.

FC Layer

FC layer is a regular neural network and it can view as the final learning phase which maps extracted visual features to desired outputs.

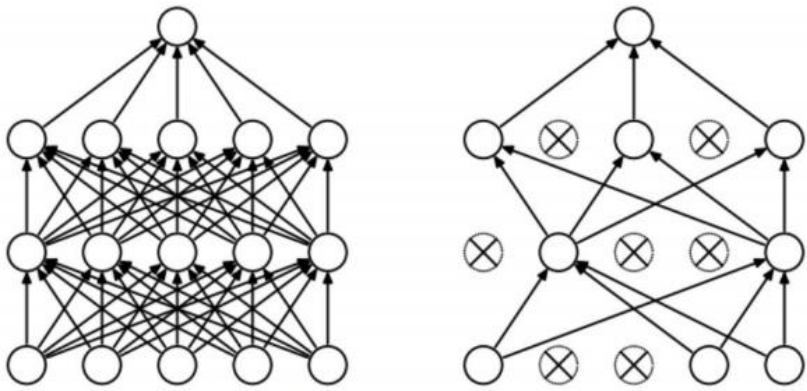
Usually, the FC layer adaptive to classification and encoding problems.

The common output of the layer is a vector which is then passed through softmax to represent the confidence of classification.

Dropout

During training, randomly ignore activations by probability p and during testing, use all activations but scale them by p.

Dropout effectively prevents overfitting by reducing the correlation between neurons.



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

AIM

The aim of the following lab exercise is to implement the CNN which can recognize the handwritten digits using Pycharm and Tensorflow and then train the model and evaluate the trained result.

We will use MNIST dataset for this laboratory.

Following steps are required.

Task 1: Preparation of the development environment

Task 2: Making script using CNN

Task 3: Running script and get results

LAB EXERCISE 11: CNN FOR MNIST

1. Preparation of the development environment
2. Making CNN script using MNIST dataset
3. Running script and get results

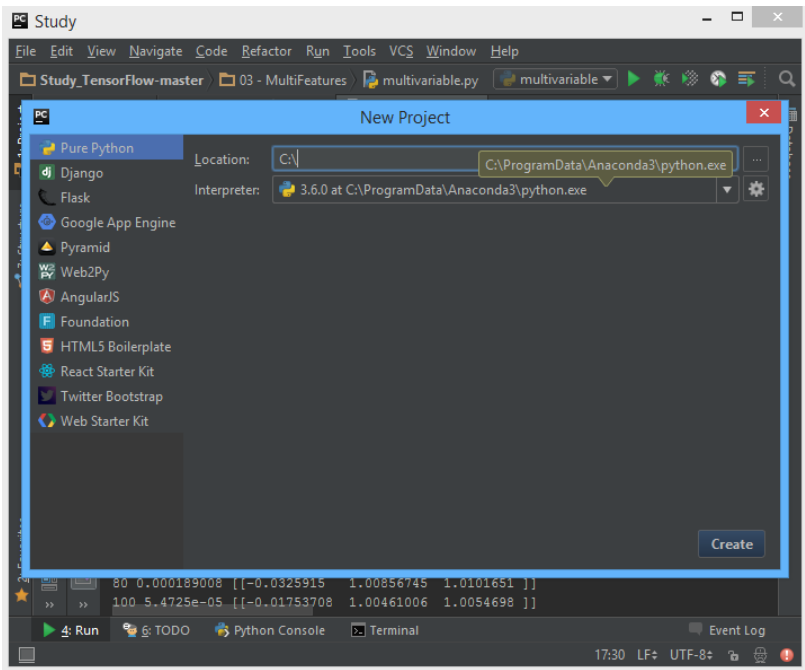
Task 1: Preparation of the development environment

STEP 1: Problem Statement

In this lab, let's create a python script to build and train a CNN which can recognize the handwritten digits from the MNIST dataset.

STEP 2: Run PyCharm and create the project

First, run PyCharm and create a new project using File/New Project menu.

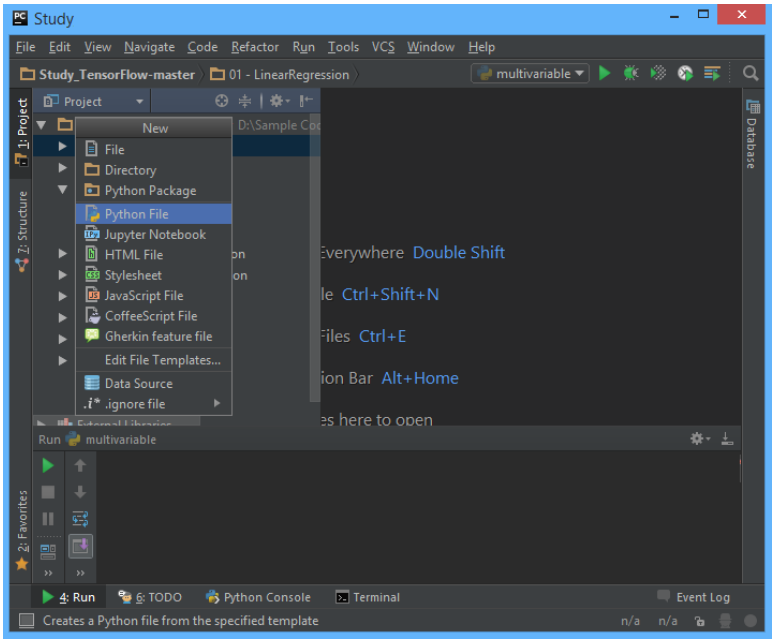


Select the project location and interpreter there and click Create Button. Then the empty project will be created.

STEP 3: Create a python script file

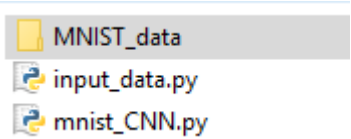
Create a new python file using File/New/Python File.

And give the file name as “mnist_CNN.py”



STEP 4: Download MNIST dataset

We can download the MNIST dataset using input_data.py. This script and MNIST datasets are attached in this laboratory.



Task1 is completed.

Task 2: Making CNN script using MNIST dataset

STEP 5: Import packages

First, import the Tensorflow and Numpy package in order to use in Python script using the code.

```
import tensorflow as tf
import numpy as np
```

STEP 6: Load training data

We can load the training data using input_data.py.

```
import input_data
mnist = input_data.read_data_sets("MNIST_data/",
                                  one_hot=True)
trX, trY, teX, teY = mnist.train.images,
                     mnist.train.labels,
                     mnist.test.images,
                     mnist.test.labels
```

STEP 7: Define model weights

And define X, Y, W as tensorflow variable.

```
trX = trX.reshape(-1, 28, 28, 1)  # 28x28x1 input img
teX = teX.reshape(-1, 28, 28, 1)  # 28x28x1 input img

X = tf.placeholder("float", [None, 28, 28, 1])
Y = tf.placeholder("float", [None, 10])

w2 = init_weights([3, 3, 32, 64])     # 3x3x32 conv, 64
outputs
w = init_weights([3, 3, 1, 32])       # 3x3x1 conv, 32
outputs
w3 = init_weights([3, 3, 64, 128])    # 3x3x32 conv, 128
outputs
w4 = init_weights([128 * 4 * 4, 625])
w_o = init_weights([625, 10])         # FC 625 inputs,
10 outputs (labels)

p_keep_conv = tf.placeholder("float")
p_keep_hidden = tf.placeholder("float")
py_x = model(X, w, w2, w3, w4, w_o, p_keep_conv,
p_keep_hidden)
```

STEP 8: Define the model function

Next, let's define the main CNN model.

```
def model(X, w, w2, w3, w4, w_o, p_keep_conv,
p_keep_hidden):
    l1a = tf.nn.relu(tf.nn.conv2d(X, w, strides=[1, 1,
1, 1], padding='SAME'))
    l1 = tf.nn.max_pool(l1a, ksize=[1, 2, 2, 1],
strides=[1, 2, 2, 1], padding='SAME')
    l1 = tf.nn.dropout(l1, p_keep_conv)

    l2a = tf.nn.relu(tf.nn.conv2d(l1, w2, strides=[1, 1,
1, 1], padding='SAME'))
    l2 = tf.nn.max_pool(l2a, ksize=[1, 2, 2, 1],
strides=[1, 2, 2, 1], padding='SAME')
    l2 = tf.nn.dropout(l2, p_keep_conv)

    l3a = tf.nn.relu(tf.nn.conv2d(l2, w3, strides=[1, 1,
1, 1], padding='SAME'))
    l3 = tf.nn.max_pool(l3a, ksize=[1, 2, 2, 1],
strides=[1, 2, 2, 1], padding='SAME')
    l3 = tf.reshape(l3, [-1,
w4.get_shape().as_list()[0]])
    l3 = tf.nn.dropout(l3, p_keep_conv)

    l4 = tf.nn.relu(tf.matmul(l3, w4))
    l4 = tf.nn.dropout(l4, p_keep_hidden)

    pyx = tf.matmul(l4, w_o)
    return pyx
```

We can create convolution, pooling, dropout, activation layers using tensorflow as above codes.

STEP 9: Define the cost function

And cost function could be defined as follows.

```
cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(l
ogits=py_x, labels=Y))
```

STEP 10: Define the learning rate and optimizer

We can define the learning rate and optimizer as the same as the previous chapter.

```
train_op = tf.train.RMSPropOptimizer(0.001,
0.9).minimize(cost)
predict_op = tf.argmax(py_x, 1)
```

Task2 is completed.

Task 3: Running script and get results

Let's complete the code and run it.

STEP 11: Initialize the tensorflow variables

Initialize all of the tensorflow variables.

```
init = tf.initialize_all_variables()

with tf.Session() as sess:
    sess.run(init)
```

STEP 12: Train the model

If add the training code then the project will be complete here. We print the value of hypothesis and accuracy.

Then the full code is below.

```
import tensorflow as tf
import numpy as np

import input data

batch_size = 128
test_size = 256


def init_weights(shape):
    return tf.Variable(tf.random_normal(shape, stddev=0.01))


def model(X, w, w2, w3, w4, w_o, p_keep_conv, p_keep_hidden):
    l1a = tf.nn.relu(tf.nn.conv2d(X, w, strides=[1, 1, 1, 1],
                                  padding='SAME'))
    l1 = tf.nn.max_pool(l1a, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
    l1 = tf.nn.dropout(l1, p_keep_conv)
```

```
    l2a = tf.nn.relu(tf.nn.conv2d(l1, w2, strides=[1, 1, 1, 1],
                                  padding='SAME'))
    l2 = tf.nn.max_pool(l2a, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
    l2 = tf.nn.dropout(l2, p_keep_conv)

    l3a = tf.nn.relu(tf.nn.conv2d(l2, w3, strides=[1, 1, 1, 1],
                                  padding='SAME'))
    l3 = tf.nn.max_pool(l3a, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
    l3 = tf.reshape(l3, [-1, w4.get_shape().as_list()[0]])
    l3 = tf.nn.dropout(l3, p_keep_conv)

    l4 = tf.nn.relu(tf.matmul(l3, w4))
    l4 = tf.nn.dropout(l4, p_keep_hidden)

    pyx = tf.matmul(l4, w_o)
    return pyx


mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
trX, trY, teX, teY = mnist.train.images, mnist.train.labels,
mnist.test.images, mnist.test.labels
trX = trX.reshape(-1, 28, 28, 1)  # 28x28x1 input img
teX = teX.reshape(-1, 28, 28, 1)  # 28x28x1 input img

X = tf.placeholder("float", [None, 28, 28, 1])
Y = tf.placeholder("float", [None, 10])

w2 = init_weights([3, 3, 32, 64])      # 3x3x32 conv, 64 outputs
w = init_weights([3, 3, 1, 32])        # 3x3x1 conv, 32 outputs
w3 = init_weights([3, 3, 64, 128])     # 3x3x32 conv, 128
outputs
w4 = init_weights([128 * 4 * 4, 625])
w_o = init_weights([625, 10])          # FC 625 inputs, 10
outputs (labels)

p_keep_conv = tf.placeholder("float")
p_keep_hidden = tf.placeholder("float")
py_x = model(X, w, w2, w3, w4, w_o, p_keep_conv, p_keep_hidden)

cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=p
y_x, labels=Y))
train_op = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
predict_op = tf.argmax(py_x, 1)

checkpoint_dir = "cps/"

# Launch the graph in a session
with tf.Session() as sess:
    # tf.global_variables_initializer().run()
    init = tf.initialize_all_variables()
    sess.run(init)
    saver = tf.train.Saver()
```

```
    ckpt = tf.train.get_checkpoint_state(checkpoint_dir)
    if ckpt and ckpt.model_checkpoint_path:
        print ('load learning')
        saver.restore(sess, ckpt.model_checkpoint_path)

    for i in range(100):
        training_batch = zip(range(0, len(trX), batch_size),
                             range(batch_size, len(trX),
                                   batch_size))

        for start, end in training_batch:
            sess.run(train_op, feed_dict={X: trX[start:end],
                                          Y: trY[start:end],
                                          p_keep_conv: 0.8,
                                          p_keep_hidden: 0.5})

        train_indices = np.arange(len(trX))  # Get A Test Batch
        np.random.shuffle(train_indices)
        train_indices = train_indices[0:test_size]
        train_acc = np.mean(np.argmax(trY[train_indices],
                    axis=1) ==
                    sess.run(predict_op, feed_dict={X:
                                          trX[train_indices],
                                          Y: trY[train_indices],
                                          p_keep_conv: 1.0,
                                          p_keep_hidden: 1.0}))

        test_indices = np.arange(len(teX)) # Get A Test Batch
        np.random.shuffle(test_indices)
        test_indices = test_indices[0:test_size]
        test_acc = np.mean(np.argmax(teY[test_indices], axis=1)
                    == sess.run(predict_op, feed_dict={X:
                                          teX[test_indices],
                                          Y: teY[test_indices],
                                          p_keep_conv: 1.0,
                                          p_keep_hidden: 1.0}))
        print(i, test_acc)
        saver.save(sess, checkpoint_dir + 'model.ckpt')
```

Let's run this code. Then we can see the results such as below.

As we can know from the results, this CNN is working well.

```
0 0.98046875
1 0.97265625
2 0.984375
3 0.9765625
4 0.9921875
5 0.9921875
6 0.9921875
```

LAB CHALLENGE

Challenge

In this lab, we have implemented CNN model using MNIST dataset and verify the result of CNN model.

Use another image data set such as face or objects and then try run the CNN model again.

SUMMARY

Convolution neural network (CNN, ConvNet) is a class of deep, feed-forward (not recurrent) artificial neural networks that are applied to analyzing visual imagery and other applications.

CNN model contains the convolution layer, pooling layer, dropout, fully connected layer, activation function.

REFERENCES

- https://en.wikipedia.org/wiki/CNN
- https://en.wikipedia.org/wiki/TensorFlow
- https://en.wikipedia.org/wiki/dropout

INDEX