Research Report

Image Lab, Min-Gi Yeom, Ha-Noung, Kwak

-Lab Date: 04/09/17 -Due Date: 07/12/17

1 Research Plan

-Python study
04/09/17 Planning, Role allocation
05/09/17 Planning- Python study(www.codeacademy.com)
06/09/17 Python study- chap 1, 2
07/09/17 Python study- chap 3, 4
11/09/17 Python study- chap 5, 6
12/09/17 Python study- chap 7, 8
13/09/17 Python study- chap 9, 10
14/09/17 Python study- chap 11, 12
15/09/17 Python study- chap 13

-Inflearn(https://www.inflearn.com) Machine Learning Lecture
18/09/17 Inflearn -Machine Learning TensorFlow
19/09/17 1-1 Basic Machin Learning concept
20/09/17 2-1 Linear Regression, Hypothesis and cost
21/09/17 2-2 Implementation Linear Regression, Tensorflow
22/09/17 3-1 cost Minimize algorithm of Linear Regression
25/09/17 3-2 Implementation cost Minimize, Tensorflow
26/09/17 4-1 Multi-variable Linear Regression
27/09/17 4-2 Implementation Multi-varible Linear Regression, Tensorflow
28/09/17 5-1 Logistic Classification, Hypothesis
29/09/17 5-2 Logistic Regression, Cost function
02/10/17 6-1 Multinomial(Softmax)
03/10/17 6-2 Implementation Softmax classification, Tensorflow
04/10/17 7-1 Learning rate, overfitting
05/10/17 8-1 Basic concept of Deep Learning, XOR problem
06/10/17 8-2 Backpropagation
09/10/17 9-1 Initial weight
10/10/17 9-2 Dropout

-Break

12/10/17 - 20/10/17 Midterm Exam)

-PyTorchZeroToAll
23/10/17 Basic concept of Machine Learning Deep Learning
24/10/17 PyTorch Overview
25/10/17 Design model
26/10/17 Finding weight that minimize error
27/10/17 Finding weight that minimize error
30/10/17 Updating weight in model which has 1 or more hidden layers
31/10/17 PyTorch's gradient function
01/11/17 Type of Activation Functions
02/11/17 Type of Optimizers(1): step direction
03/11/17 Type of Optimizers(2): step size
06/11/17 Make Binary prediction model (Pass or Fail)
07/11/17 Reading data size at once
08/11/17 Classify data to 3 or more
09/11/17 CNN(1)
10/11/17 CNN(2)

-Application
13/11/17 - 01/12/17 Deep Learning Application(1)
04/12/17 - 27/12/17 Deep Learning Application(2)

2 Daily Report: Python study

2.1 06/09/17

Python study
-1. Python Syntax
-2. String and Console Output

2.2 07/09/17

Python study
-3. Condtionals and Control Flow
-4. Functions, Taking a Vacation

2.3 11/09/17

Python study
-5. Lists Dictionaries, A Day at the Supermarket
-6. Student Becomes the Teacher

2.4 12/09/17

Python study
-7. Lists Functions, Battleship!
-8. Loops, Practice Makes Perfect

2.5 13/09/17

Python study
-9. Exam Statistics
-10. Advanced Topics in Python, Indtroduction to Bitwise Operators

2.6 14/09/17

Python study
-11. Introduction to Classes, Classes
-12. File Input/Output

2.7 15/09/17

Python study
-13. Python Final Project

3 Daily Report: Inflearn, Machine Learning Lecture

3.1 18/09/17

Inflearn -Machine Learning TensorFlow

3.2 19/09/17

1-1 Basic Machin Learning concept
Supervised learning -learning with labeled examples(training sets)
Unsupervised learning -unlabeled data(word clustering)

@Common problem of Supervised Learning:
Image labeling, Email spam filter, predicting exam score
* Predicting final exam score based on time spent(Regression)
* Pass/fail based on time spent (binary classification)

* Letter grade based on time spent (multi-label classification)

3.3 20/09/17

2-1 Linear Regression, Hypothesis and cost
Linear Regression, set the one Linear Hypothesis: H(x)=Wx+b
How fit the line to our (training) data : Cost(Loss) Function

$$cost(w,b) = \frac{1}{m}\sum_{i=1}^{m} ||H(x^i) - y^i||^2$$

$m = number\ of\ data, \quad H(x) = Hypothesis, \quad y = Real\ data\ (true\ value)$

Goal of the Linear Regression is Minimize the cost.

3.4 21/09/17

2-2 Implementation Linear Regression, Tensorflow
1. build graph using TensorFlow operations
2. feed data and run graph(operation) ¿ sess.run(op, feed$_{d}ict = x : x_{d}ata$)
$3.updatevariablesinthegraph(andreturnvalues)$

3.5 22/09/17

3-1 cost Minimize algorithm of Linear Regression
Gradient descent algorithm :
* Minimize cost function
* Gradient descent is used many minimization problems
* For a given cost function, cost (W,b), it will find W,b to minimize cost
* It can be applied to more general function: cost(w1, w2, ...)

How it works?
1. Start with initial guesses, (0,0), Keeping changing W and b a little bit to try and reduce cost(W,b)
2. Each time you change the parameters, you select the gradient which reduceces cost(W,b) the most possible
3. Repeat
4. Do so until you converge to a local minimum
5. Has an interesting property: Where you start can determine which minimum you end up

3.6 25/09/17

3-2 Implementation cost Minimize, Tensorflow
* Gradient descent algorithm

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (Wx^i - y^i)x^i$$

```
inimize: Gradient Descent using derivative: W -= learning_rate * derivative
earning_rate = 0.1
radient = tf.reduce_mean((W * X - Y) * X)
escent = W - learning_rate * gradient
pdate = W.assign(descent)
```

3.7 26/09/17

4-1 Multi-variable Linear Regression
Set the hypothesis as

$$H(x_1, x_2, x_3, \ldots, x_n) = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + b$$

Hypothesis using matrix
H(X)=XW

3.8 27/09/17

4-2 Implementation Multi-varible Linear Regression, Tensorflow

```
x_data = [[73, 80, 75], [93, 88, 93], [89, 91, 90], [96, 98, 100], [73, 66, 70]]
y_data = [[152], [185], [180], [196], [142]]
X = tf.placeholder(tf.float32, shape=[None,3])
Y = tf.placeholder(tf.float32, shape=[None,1])
W = tf.Variable(tf.random_normal([3,1]), name = 'weight')
b = tf.Variable(tf.random_normal([1]), name = 'bias')
hypothesis = tf.matmul(X,W) + b
```

3.9 28/09/17

5-1 Logistic Classification, Hypothesis
For the Binary Classification previous Linear Regression have critical error
Solution ¿ sigmoid function
Logistic Hypothesis:

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

3.10 29/09/17

5-2 Logistic Regression, Cost function
Cost function for sigmoid function

$$c(H(x), y) = \begin{cases} -\log\big(H(x)\big) & : y = 1 \\ -\log\big(1 - H(x)\big) & : y = 0 \end{cases}$$

$$c(H(x), y) = \ ylog\big(H(x)\big) - (1 - y)\log\,(1 - H(x))$$

Minimize cost – Gradient decent algorithm

```
#cost function
cost = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis)))
#Minimize
a = tf.Variable(0.1) #learning rate
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)
```

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

3.11 02/09/17

6-1 Multinomial(Softmax)
Several binomial classifications can make Multinomial classification.

To classify (A, B, C), We can use 3 binomial classifications (A or not), (B or not), (C or not)
Softmax function: substitute of Logistic function for Multinomial classification

$$\begin{bmatrix} W_{A1} & W_{A2} & W_{A3} \\ W_{B1} & W_{B2} & W_{B3} \\ W_{C1} & W_{C2} & W_{C3} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} W_{A1}x_1 + W_{A2}x_2 + W_{A3}x_3 \\ W_{B1}x_1 + W_{B2}x_2 + W_{B3}x_3 \\ W_{C1}x_1 + W_{C2}x_2 + W_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \overline{y_A} \\ \overline{y_B} \\ \overline{y_C} \end{bmatrix}$$

$$\overline{y_A} = \mathit{Hypothesis\ result\ of\ A}$$

$$\mathrm{S}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$$

Cost function of Softmax is Cross-Entropy

$$\mathrm{D(S,L)} = -\sum_i L_i \log(s_i) \qquad (in\ logistic, S = H(x)\ L = y)$$

After we find the cost function, same as logistic regression, find the minimize of cost function by using Gradient descent algorithm.

3.12 03/09/17

6-2 Implementation Softmax classification, Tensorflow

```
#softmax function
Hypothesis = tf.nn.softmax(tf.matmul(X,W) + b)

#Cross entropy cost/loss
Cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis = 1))
Optimizer = tf.train.GradientDescentOptimizer(learning_rate = 0.1).minimize(cost)

#Testing & One-hot encoding
a = sess.run(hypothesis, feed_dict={X: [1, 11, 7, 9]})
print(a, sess.run(tf.arg_max(a,1)))
```

3.13 04/09/17

7-1 Learning rate, overfitting

Learning rate

1. Large learning rate: overshooting
Large steps divergence the value
2. Small learning rate: takes too long, stops at local minimum.
Try several learning rates ¿ observe the cost function, check it goes down in a reasonable rate

-Overfitting

* Our model is very good with training data set(with memorization)
* Not good at test dataset or in real use

-Solutions for overfitting

* More training data
* Reduce the number of features
* Regularization
* Let's not have too big numbers in the weight.

$$\text{L(loss funtion)} = \frac{1}{N}\sum_i D(S(wx+b), L) + \ \varepsilon \sum W^2 \text{, } (\varepsilon\text{: } regularization\ strength)$$

3.14 05/09/17

8-1 Basic concept of Deep Learning, XOR problem
Solving XOR problem : backpropagation(Paul Werbos)

Big problem(1990s)

* Backpropagation just did not work well for normal neural nets with many layers
* Other rising machine learning algorithm: SVM, RandomForest

Breakthrough(2006)

* Neural networks with many layers really could be trained well, if the weights are initialized in a clever way rather than randomly
* Deep machine learning methods are more efficient for difficult problems than shallow methods

3.15 06/09/17

8-2 Backpropagation
Back propagation(chain rule)
chain rule

$$f = wx + b, g = wx, f = g + b$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial x}$$

We can simplify the neural nets with many layers.

3.16 09/09/17

9-1 Initial weight

Need to set the initial weight values wisely
* Not all 0's
* Challenging issue
* Hinton et al. (2006)"A fast learning algorithm for Deep belief Nets"

How can we use RBM to initialize weights?
* Apply the RBM idea on adjacent two layers as a pre-training step
* Continue the first process to all layers
* This will set weights
* Ex: Deep Belief Network

3.17 10/09/17

9-2 Dropout
Dropout: A simple way to prevent Neural Networks from Overfitting[Srivastava et al. 2014]
"randomly set some neurons to zero in the forward pass"

```
#TensorFlow implementation
Dropout_rate = tf.placeholder("float")
_L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1))
L1 = tf.nn.dropout(_L1, dropout_rate)

#Train
Sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys, dropout_rate: 0.7})
```

4 Daily Report: Break

4.1 12/10/17 - 20/10/17

Mid-term Exam

5 Daily Report: PyTorchZeroToAll

5.1 23/10/17

-Basic concept of Machine Learning Deep Learning
1) Human Intelligence
Human use inferring or prediction to decide what to do using information as inputs.
2) Machine Learning Deep Learning
Also, using a lot of information about something and then infer or predict something. For example, these are 'what is number', 'what kind of shoes you were', 'where is here(using images)' etc. We feed images and corresponding true labels(data sets) to train machine. After that, machine learn images if machine get input image then usually print output as true labels. 3) Deep Learning
Deep Learning is group of algorithm that are using deep neural nets which have enormous layer.
$DeepLearning \subset Representationlearning \subset Logisticregression \subset AI$
Deep Learning developers need to have basic algebra, probability and python.

5.2 24/10/17

-PyTorch Overview
1) PyTorch
This is computing package based on Python, and it supply Tensor which is like neuron in machine.
2) Install PyTorch
Before installing PyTorch, install CUDA and cuDNN. CUDA is a parallel computing language and cuDNN is used to accelerate GPU.
In Linux OS, we write
pip3 install http://download.pytorch.org/whl/cu80/torch-0.2.0.post3-cp36-cp36m-manylinux1$_x$86$_6$4$.whlpip3installtorchvision$
$onterminaltoinstallPyTorch.$
$-\ DeepLearningAbbreviation$
1)$DNN : DeepNeuralNet$
2)$CNN : ConvolutionNeuralNet$

$3)RNN : RecurrentNeuralNet$

5.3 25/11/17

-Types of learning
1)Supervised learning
Providing data sets with answers and machine train them. In the future, we use supervised learning to make model.
Sequence of Learning
1. Determine the subject of training
2. Collect training set and test set.
3. Make Model and initialize coefficients
4. Run algorithm using training set(Computing training error)
5. Run algorithm using test set which is separate with training set(Computing test error)
6. Iterate when test error is decreasing

2)Unsupervised learning
Providing only data sets and machine classify data sets, so it has no evaluation of the accuracy of output.
3)Reinforcement learning
The machine takes action in an environment(ex. Go), and get rewards. The goal is to find the action that cause highest rewards.

5.4 26/10/17

-Design model
1.Linear model
$\hat{a} = x * w + b$
x is an input data, w is an weight, b is a bias and $y\wedge$ is the prediction of grond truth. We start w in random value.
For Updating w in our model, we calculate training loss(error) as MSE(Mean Squared Error)

$$loss = \frac{1}{N}\sum_{n=1}^{N}(\hat{y_n} - y_n)^2$$

5.5 27/10/17

-Finding weight that minimize error
1. Gradient descent

We initialize weight(w) randomly and use **Gradient** $= \frac{\partial loss}{\partial w}$
Using gradient and step size(learning rate) update w to move in to smaller loss
Loss is defined by MSE, so loss function is kind of polynomial function that has minimum value when derivative of the function is zero.

$$w_{t+1} = w_t \alpha \frac{\partial loss}{\partial w}$$

2. Setting α(step size)
If step size is too small, then w put in the local minimum not global minimum and if step size is too big, then $\alpha \frac{\partial loss}{\partial w}$ is so big that loss function and w are divergent.

5.6 30/10/17

-Updating weight in model which has 1 or more hidden layers
If network is complicated, we often are using nonlinearities between nodes. So manually computing this gradient need huge computing power and time.
1. Chain Rule
$f = f(g); g = g(x)$
$dfdx = dfdgdgdx$
This rule divide overlapping functions and compute the gradient one by one.
2. Back-propagation
Caculating loss using training data is forward pass. We use loss and local gradient using divided overlapping function to get global gradient of $\frac{\partial loss}{\partial w}$

5.7 31/10/17

-PyTorch's gradient function
1. Autograd

```
    w = Variable(torch.Tensor([1.0]),  requires_grad=True)
    def forward(x):
        return x * w
    def loss(x, y):
        y_pred = forward(x)
        return (y_pred - y) * (y_pred - y)

    l = loss(x_val, y_val)
    l.backward()
    w.data = w.data - 0.01 * w.grad.data
```

In PyTorch, if we set w as 'requires gradient' true and use backward() function, then we get gradient of loss automatically.

5.8 1/11/17

-Type of Activation Functions
This determine activation state(0, 1) of node by the node's input data

1. Sigmoid

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

This is one of the simple activation function. Sigmoid is logistic function. It's maximum approximate to 1 and minimum is approximate to zero.
2. ReLU(Rectified Linear Unit)
ReLU is h(x) = x when x is over 0, otherwise h(x) = 0. It's usually use because ReLU's derivative is so simple that h'(x) = 1 when x is over 0 otherwise h'(x) = 0.
This property has another advantages. In DNN, backpropagation using sigmoid cause loss error data because it has so many hidden layers that gradient of loss need huge amonut of iterative chain rule. But ReLU protect data when running backpropagation

5.9 2/11/17

-Type of Optimizers(1): step direction
1.GD(Gradient Descent)
It is check all data and find gradient at weight w position in loss function and move to most sheer direction.
2.SGD(Stochastic Gradient Descent)
GD is slow because of reflecting all data, so make subset data evenly and use these to move more quickly.
3.Momentum
Using momentum, we move to calculated step direction + a bit of last direction as momentum.
4.NAG(Nestrov Accelerated Gradient)(from Momentum)
First move momentum direction and then calculate move direction for move.
5. Adam(from Momentum)
RMSProp(in Type of Optimizers(2)) + Momentum = suitable step direction and step size.
6. Nadam(from NAG and Adam)
RMSProp + NAG (not Momentum)

5.10 3/11/17

-Type of Optimizers(2): step size
1.Adagrad
If the calculated step direction point to where we wasn't go, increase step size. Otherwise, decrease step size gradually.
2. AdaDelta(from Adagrad)
AdaDelta protect on stop when running Adagrad at where we went.
3. RMSProp

RMSProp block small step size dealing with each instance separately.
4. Adam(from RMSProp)(in Type of Optimizers(1))

5.11 6/11/17

-Make Binary prediction model (Pass or Fail)
We plug into sigmoid from the output of the linear layer to make squash numbers between 0 to 1. In this case, if $\hat{y} > 0.5$ than we assume that output is 1 otherwise 0.
With sigmoid, MSE is not work well because it's optimize for linear regression. So we need new loss function, cross entropy loss that is optimize for logistic regression and classification.

$$loss = -\frac{1}{N}\sum_{n=1}^{N} y_n log\hat{y_n} + (1-y_n)log(1-\hat{y_n})$$

5.12 7/11/17

-Reading data size at once
It's not efficient that we compute the gradients for all data points therefore we divide data set into small batches.
1. One epoch
one forward pass and on backward pass of all the training examples.
2. Batch size
the number of training examples in one forward/backward pass. We adjust the batch size by memory volume.
3. Number of iterations
It's number of passes. The one pass is one forward process + one backward process.
4. Data load sequence as batches
original data $->$ random shuffle $->$ making queue $->$ using queue each element as batch.

5.13 8/11/17

-Classify data to 3 or more
1. Softmax

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

We use softmax with cross entropy cost function that is sum of -Ylog$\hat{Y}$. Y and $\hat{Y}$ is vector like [0.0 1.0 0.0 0.0], thus this is fit to our purpose(classification).

5.14 9/11/17

-CNN(1)
General CNN is consist of feature extraction that is convolution + subsampling and classification that is fully connected layer; Dense Net. 1.Convolution
it is going to look at only small part of image at once. So we use filter that is small than input image data. Filter moves entire image but we look at only small portion at a time. Filter is a kind of vector, so we get output from inner product between filter and the part of image.
Stride is filter's move block step. Padding is attach values at boundary of data to filter can read first pixel of image. If image hasn't padding, then filter can't read some pixel by filter size. Thus, output image size decrease.
2. Activation
After Convolution, we use activation function to output. 3. Max Pooling
After repeating convolution + activation(ReLU) n times, we do max pooling to reduce image size. In this process, we get meta data of image. So the reduce information is small compared to decrease computation time(time complexity).
4. Fully Connected layer
After repeating 1,2 and 3, finally we use it some times and classify output data as softmax.

5.15 10/11/17

-CNN(2)
The addvanced mthod of CNN fine to data case by case.
1. 1x1xn convolution
The convolution filter is 1x1 and depth is more than n. It's control image's depth size.
2. Deep Residual Learning
Plaint net vulnerable to vanishing gradient. One of the solutions is using Residual Net that set up input + weight layer(convolution layer etc.) as output and then output is used as input of activation function.

6 Application(1)

6.1 13/11/17 - 01/12/17

-Deep Learning Application(1)

6.1.1 Abstract

Simple Neural Net for Iris dataset using PyTorch. Multilayer perceptron model, with one hidden layer.

6.1.2 Keyword

hidden layer, PyTorch, Machine Learning,Neural Net, Multilayer perceptron model.

6.1.3 Introduction

This is a collection of simple and easy to read program for Iris dataset classification by using PyTorch library

6.1.4 Method

SECTION 1 : Load and setup data for training

the datasets separated in two files from originai datasets:
iris$_train.csv = datasetsfortrainingpurpose, 80fromtheoriginaldata$
$iris_test.csv = datasetsfortestingpurpose, 20fromtheoriginaldata$

SECTION 2 : Build and Train Model

Multilayer perceptron model, with one hidden layer.
input layer : 4 neuron, represents the feature of Iris
hidden layer : 10 neuron, activation using ReLU
output layer : 3 neuron, represents the class of Iris

optimizer = stochastic gradient descent with no batch-size
loss function = categorical cross entropy
learning rate = 0.01
epoch = 500

SECTION 3 : Testing model

```python
import pandas as pd

#load
datatrain = pd.read_csv('../Datasets/iris/iris_train.csv')

#change string value to numeric
datatrain.set_value(datatrain['species']=='Iris-setosa',['species'],0)
datatrain.set_value(datatrain['species']=='Iris-versicolor',['species'],1)
datatrain.set_value(datatrain['species']=='Iris-virginica',['species'],2)
datatrain = datatrain.apply(pd.to_numeric)

#change dataframe to array
datatrain_array = datatrain.as_matrix()
```

```

#split x and y (feature and target)
xtrain = datatrain_array[:,:4]
ytrain = datatrain_array[:,4]

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
torch.manual_seed(1234)

#hyperparameters
hl = 10
lr = 0.01
num_epoch = 500

#build model
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(4, hl)
        self.fc2 = nn.Linear(hl, 3)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
net = Net()

#choose optimizer and loss function
criterion = nn.CrossEntropyLoss()
  optimizer = torch.optim.SGD(net.parameters(), lr=lr)

#train
for epoch in range(num_epoch):
    X = Variable(torch.Tensor(xtrain).float())
    Y = Variable(torch.Tensor(ytrain).long())

    #feedforward - backprop
    optimizer.zero_grad()
    out = net(X)
    loss = criterion(out, Y)
    loss.backward()
    optimizer.step()

```

```
    if (epoch) % 50 == 0:
        print ('Epoch [%d/%d] Loss: %.4f'
                      %(epoch+1, num_epoch, loss.data[0]))

#load
datatest = pd.read_csv('../Datasets/iris/iris_test.csv')

#change string value to numeric
datatest.set_value(datatest['species']=='Iris-setosa',['species'],0)
datatest.set_value(datatest['species']=='Iris-versicolor',['species'],1)
datatest.set_value(datatest['species']=='Iris-virginica',['species'],2)
datatest = datatest.apply(pd.to_numeric)

#change dataframe to array
datatest_array = datatest.as_matrix()

#split x and y (feature and target)
xtest = datatest_array[:,:4]
ytest = datatest_array[:,4]

#get prediction
X = Variable(torch.Tensor(xtest).float())
Y = torch.Tensor(ytest).long()
out = net(X)
_, predicted = torch.max(out.data, 1)

#get accuration
print('Accuracy of the network %d %%' % (100 * torch.sum(Y==predicted) / 30))
```

7 Application(2)

7.1 04/12/17 - 27/12/17

-Deep Learning Application(2)

7.1.1 Abstract

We classify given real image data correctly using Resnet50 with supervise learning.

7.1.2 Keyword

DNN, Supervised Learning, Machine Learning, Resnet50, Hymenoptera.

7.1.3 Introduction

Ants are similar to bees besides wings. So we classify data to ant or bee image using labeled data. In this study, we use supervised learning and Resnet50 that has 50 layers of residual structure.

7.1.4 Method

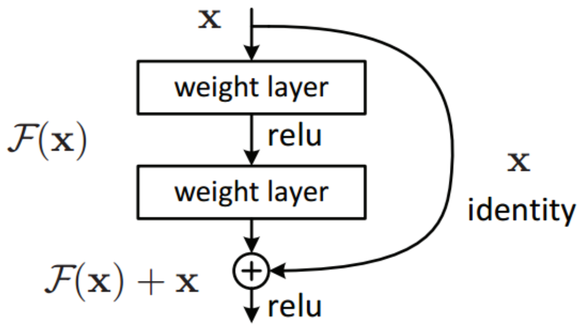

Figure 1: Residual network structure

Resnet50 The Resnet use previous output as input and previous ouput + activated input as output[Figure 1]. So it has much more characteristic than plain net which hasn't residual edge. In [Figure 1]. We recognize that F(x) and x size are same because each of these is vector.

Source Code Basic model setting as follows.
Input: hymenoptera data
Output: classified hymenoptera data
Initial condition: labeled data
Final condition: epoch 25 times

```
f train_model(model, criterion, optimizer, scheduler, num_epochs=25):
  since = time.time()

  best_model_wts = model.state_dict()
  best_acc = 0.0

  for epoch in range(num_epochs):
      print('Epoch {}/{}'.format(epoch, num_epochs - 1))
      print('-' * 10)

```

```
        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                scheduler.step()
                model.train(True)  # Set model to training mode
            else:
                model.train(False)  # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for data in dataloaders[phase]:
                # get the inputs
                inputs, labels = data

                # wrap them in Variable
                if use_gpu:
                    inputs = Variable(inputs.cuda())
                    labels = Variable(labels.cuda())
                else:
                    inputs, labels = Variable(inputs), Variable(labels)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                outputs = model(inputs)
                _, preds = torch.max(outputs.data, 1)
                loss = criterion(outputs, labels)

                # backward + optimize only if in training phase
                if phase == 'train':
                    loss.backward()
                    optimizer.step()

                # statistics
                running_loss += loss.data[0]
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects / dataset_sizes[phase]

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

```

```
            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = model.state_dict()

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

We use resnet50 to training model. Before this, first we divied 2 types of data as folder name(ants, bees).

$$H = -ylog\hat{y} - (1-y)log(1-\hat{y})$$

We calculate loss using cross entropy function H for update weight.

7.1.5 Result and Conclusion

The best accuracy is 93.46 percent. We use various type of ants and bees, but the accuracy is very high. When we train model without hidden layer, it's difficult to find global minimum position. So we know the power of hidden layers.
In future, we run additional model learing in various the number of hidden layers and data to get the perfect number of hidden layer about data size.