TensorFlow Tutorial #02

Classification

Seoung Wug Oh, Ph.D. Student
Computational Intelligence and Photography Lab.
Yonsei University

Classification

- Binary classification (logistic regression)
  - **Email:** Spam or Not ?
  - **Tumor:** Malignant or Benign ?

$$y \in \{0, 1\}$$

- Multi-class classification (softmax regression)
  - **Image:** dog ? Cup ? Hat ?
  - **Human Activity:** running ? Walking ? Clapping ?
  - **Digits:** 1? 2? 3? …

$$y \in \{0, 1, 2, 3, \dots, K\}$$

Logistic regression

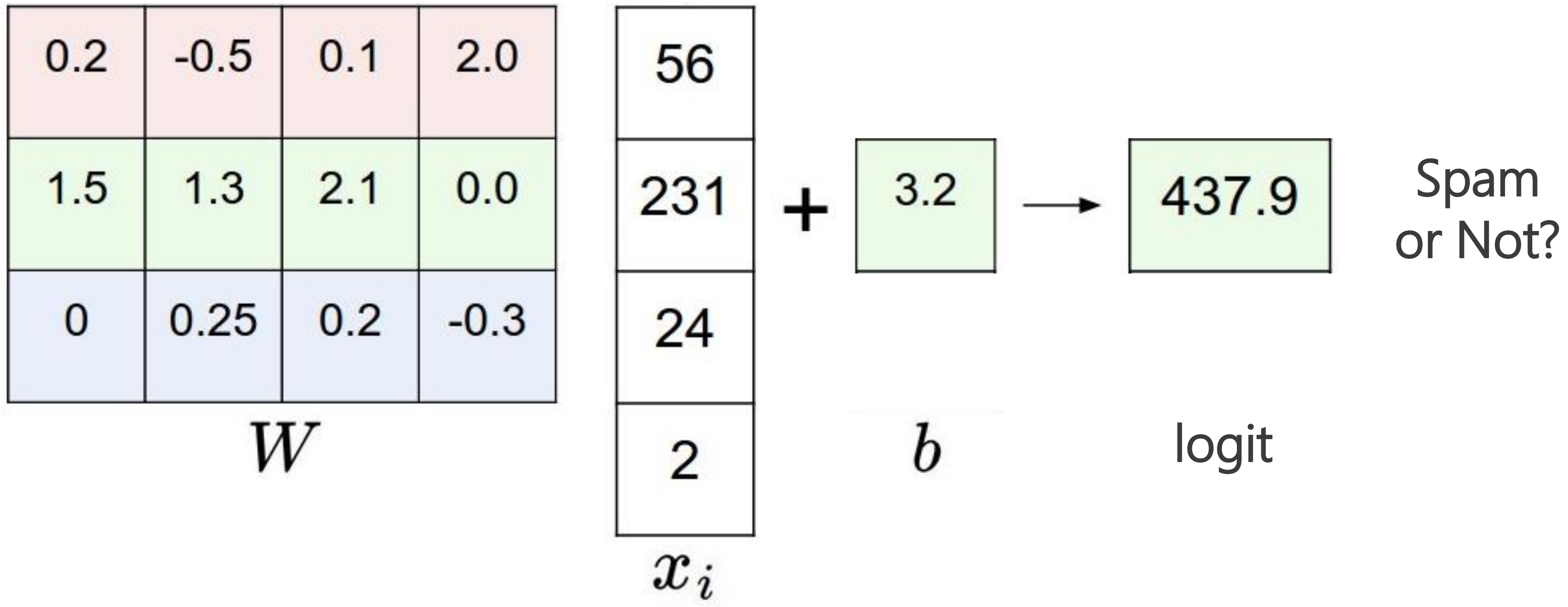
- Logistic Regression
  - Given input x, model that can estimate y.
  - Want $\hat{y} = f(x)$, where $\mathbf{0} \leq \hat{y} \leq 1.$
  - Function f can be in any form.
  - In this session, f is simple linear function.

- Linear Model
  - Relate input x and output $\hat{y}$ linearly

$$logit = \boldsymbol{W} \times x + b$$

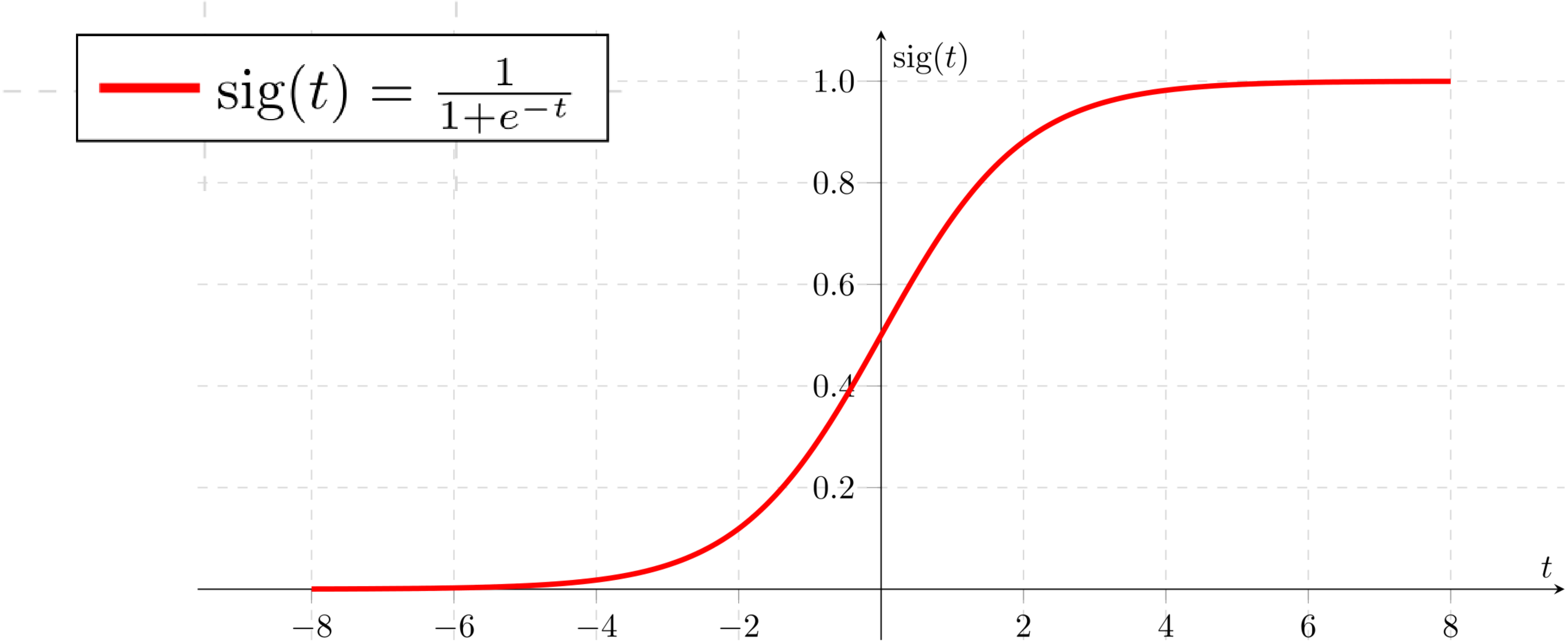
  - $logit$ is unbounded value.
  - Higher values mean higher probability of being positive (1), and vice versa.

Logistic regression



Logistic regression

- Sigmoid function
  - Squeeze $logit$ to output $0 \leq \hat{y} \leq 1$.



Logistic regression

$$\mathrm{sig}(t) = \frac{1}{1+e^{-t}}$$

- Final model
  - $\hat{y} = f(x) = \boldsymbol{sig}(W \times x + b)$
  - Goal:
    Optimize parameter set $\theta = \{W, b\}$ to fit $\hat{y} \approx y$ where y is ground-truth label.
  - If $f(x)$ is spam detector, $f(x) = \mathbf{0.7}$ tells that 70% chance of the mail is being spam mail.

- Binary cross entropy
  - $\mathrm{C} = y \times -\log(\hat{y}) + (1 - y) \times -\log(1 - \hat{y})$,
    where $y \in \{0, 1\}$

Example: Tumor classification

- Problem
  - Given some features, estimate the tumor is whether malignant or benign

- Dataset: Wisconsin Diagnostic Breast Cancer (WDBC)
  - $x = features\ of\ tumor$ (radius, texture, etc)
  - $y \in \{malignant, benign\}$

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
842302	M	17.99	10.38	122.8	1001	0.1184
842517	M	20.57	17.77	132.9	1326	0.08474
84300903	M	19.69	21.25	130	1203	0.1096
84348301	M	11.42	20.38	77.58	386.1	0.1425

Build Computational Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 30]) # Inputs: a batch of features (30 dims)
y = tf.placeholder(tf.float32, [None, 1]) # Labels: a batch of labels
```

Build Computational Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 30]) # Inputs: a batch of features (30 dims)
y = tf.placeholder(tf.float32, [None, 1]) # Labels: a batch of labels

# Set model weights
W = tf.Variable(tf.zeros([30, 1]))
b = tf.Variable(tf.zeros([1]))
```

Build Computational Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 30]) # Inputs: a batch of features (30 dims)
y = tf.placeholder(tf.float32, [None, 1]) # Labels: a batch of labels

# Set model weights
W = tf.Variable(tf.zeros([30, 1]))
b = tf.Variable(tf.zeros([1]))

# Construct model (y = W*X + b)
logit = tf.matmul(x, W) + b
pred = tf.nn.sigmoid(logit)
```

Build Computational Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 30]) # Inputs: a batch of features (30 dims)
y = tf.placeholder(tf.float32, [None, 1]) # Labels: a batch of labels

# Set model weights
W = tf.Variable(tf.zeros([30, 1]))
b = tf.Variable(tf.zeros([1]))

# Construct model (y = W*X + b)
logit = tf.matmul(x, W) + b
pred = tf.nn.sigmoid(logit)

# Directly compute loss from logit (to ensure stability and avoid overflow)
cost = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=logit,
labels=y))
```

Build Computational Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 30]) # Inputs: a batch of features (30 dims)
y = tf.placeholder(tf.float32, [None, 1]) # Labels: a batch of labels

# Set model weights
W = tf.Variable(tf.zeros([30, 1]))
b = tf.Variable(tf.zeros([1]))

# Construct model (y = W*X + b)
logit = tf.matmul(x, W) + b
pred = tf.nn.sigmoid(logit)

# Directly compute loss from logit (to ensure stability and avoid overflow)
cost = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=logit,
labels=y))

# Define optimizer and train_op
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Implementation Note

- Points
  - tf.nn.sigmoid_cross_entropy_with_logits(logits=logit, labels=y)

```
  z * -log(sigmoid(x)) + (1 - z) * -log(1 - sigmoid(x))
= z * -log(1 / (1 + exp(-x))) + (1 - z) * -log(exp(-x) / (1 + exp(-x)))
= z * log(1 + exp(-x)) + (1 - z) * (-log(exp(-x)) + log(1 + exp(-x)))
= z * log(1 + exp(-x)) + (1 - z) * (x + log(1 + exp(-x))
= (1 - z) * x + log(1 + exp(-x))
= x - x * z + log(1 + exp(-x))
```

For x < 0, to avoid overflow in exp(-x), we reformulate the above

```
  x - x * z + log(1 + exp(-x))
= log(exp(x)) - x * z + log(1 + exp(-x))
= - x * z + log(1 + exp(x))
```

Hence, to ensure stability and avoid overflow, the implementation uses this equivalent formulation

```
max(x, 0) - x * z + log(1 + exp(-abs(x)))
```

Training Loop

```
with tf.Session() as sess:
    # Initializing the variables
    sess.run(tf.global_variables_initializer())
```

Training Loop

```
with tf.Session() as sess:
    # Initializing the variables
    sess.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(X_train.shape[0]/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs = X_train[i:i+batch_size,:]
            batch_ys = y_train[i:i+batch_size,:]
            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([train_op, cost], feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += c / total_batch
```

Training Loop

```
with tf.Session() as sess:
    # Initializing the variables
    sess.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(X_train.shape[0]/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs = X_train[i:i+batch_size,:]
            batch_ys = y_train[i:i+batch_size,:]
            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([train_op, cost], feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += c / total_batch

        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))
```

Testing

```
# Test model
correct_prediction = tf.equal(tf.round(pred), y)

# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Accuracy:", accuracy.eval({x: X_test, y: y_test}))
```

Testing

```
# Test model
correct_prediction = tf.equal(tf.round(pred), y)

# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Accuracy:", accuracy.eval({x: X_test, y: y_test}))
```

- Classification
  - Decision boundary = 0.5 (rounding)
  - Accuracy the percentage of correct estimation

```
accuracy.eval({x: X_test, y: y_test}))

sess.run(accuracy, feed_dict={x: X_test, y: y_test})
```

Results

```
>> python logistic_regression.py
...
Epoch: 0095 cost= 0.024283896
Epoch: 0096 cost= 0.024087359
Epoch: 0097 cost= 0.023894221
Epoch: 0098 cost= 0.023704397
Epoch: 0099 cost= 0.023517799
Epoch: 0100 cost= 0.023334336
Optimization Finished!
Accuracy: 0.913043
```

Multi-class Classification (softmax regression)

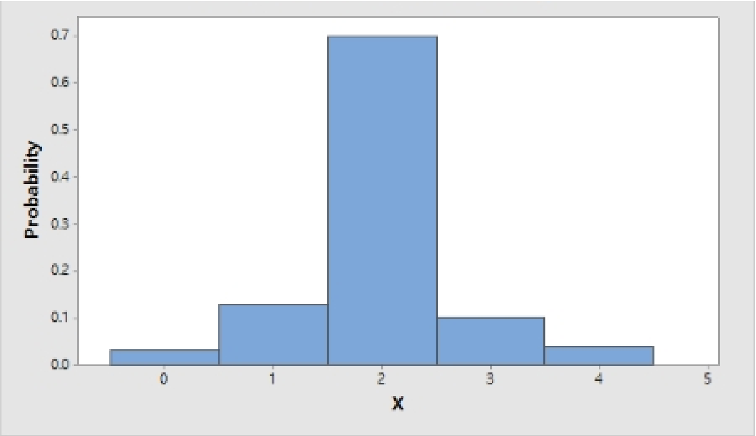
- Binary classification (logistic regression)
  - **Email:** Spam or Not ?
  - **Tumor:** Malignant or Benign ?

$$y \in \{0, 1\}$$

- Multi-class classification (softmax regression)
  - **Image:** dog ? Cup ? Hat ?
  - **Human Activity:** running ? Walking ? Clapping ?
  - **Digits:** 1? 2? 3? …

$$y \in \{0, 1, 2, 3, \dots, K\}$$

Softmax regression

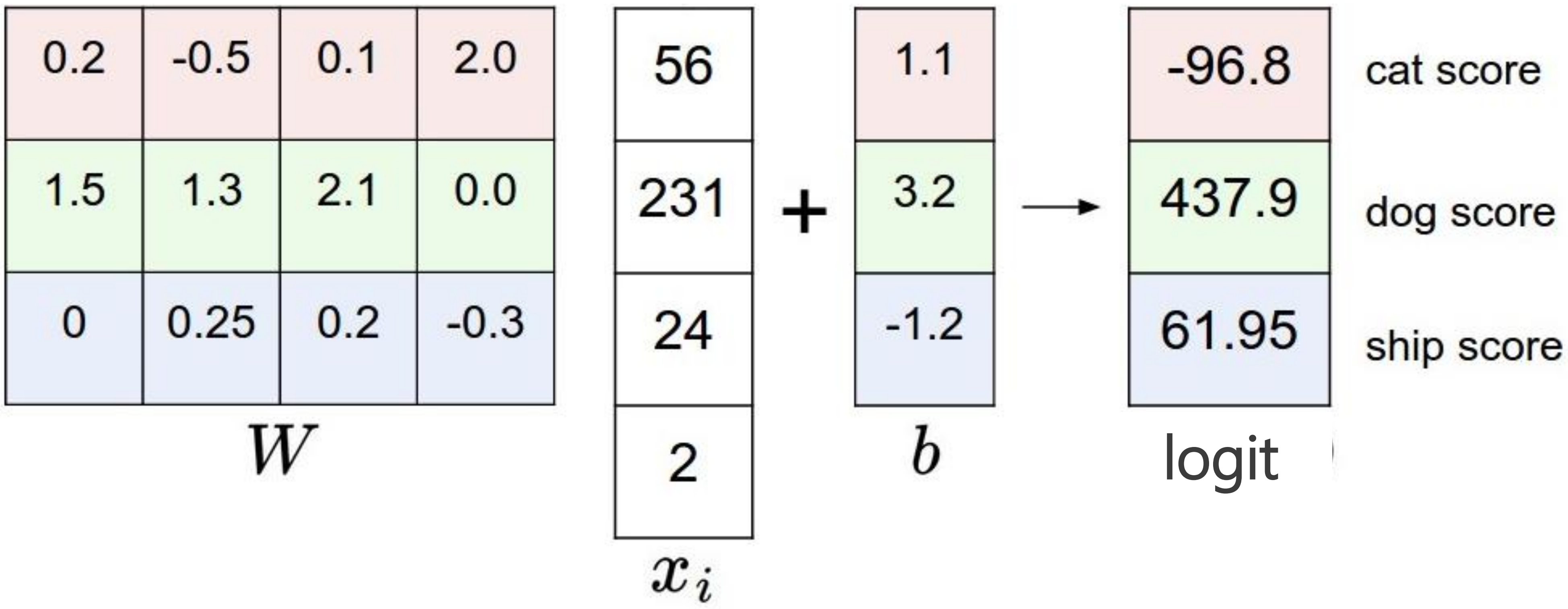


- Logistic Regression
  - $\hat{y} = f(x)$, where $0 \leq \hat{y} \leq 1$.
  - Negative or Possitive

- Softmax regression
  - Now, $\hat{y} = f(x)$ is the **probability distribution** over classes, where $0 \leq \hat{y} \leq 1$ and $\Sigma\hat{y} = 1$.
  - $logit \in \mathbb{R}^K = \boldsymbol{W} \times x + b$
  - Higher values at n-th element mean higher probability of being belong to n-th class

Softmax regression

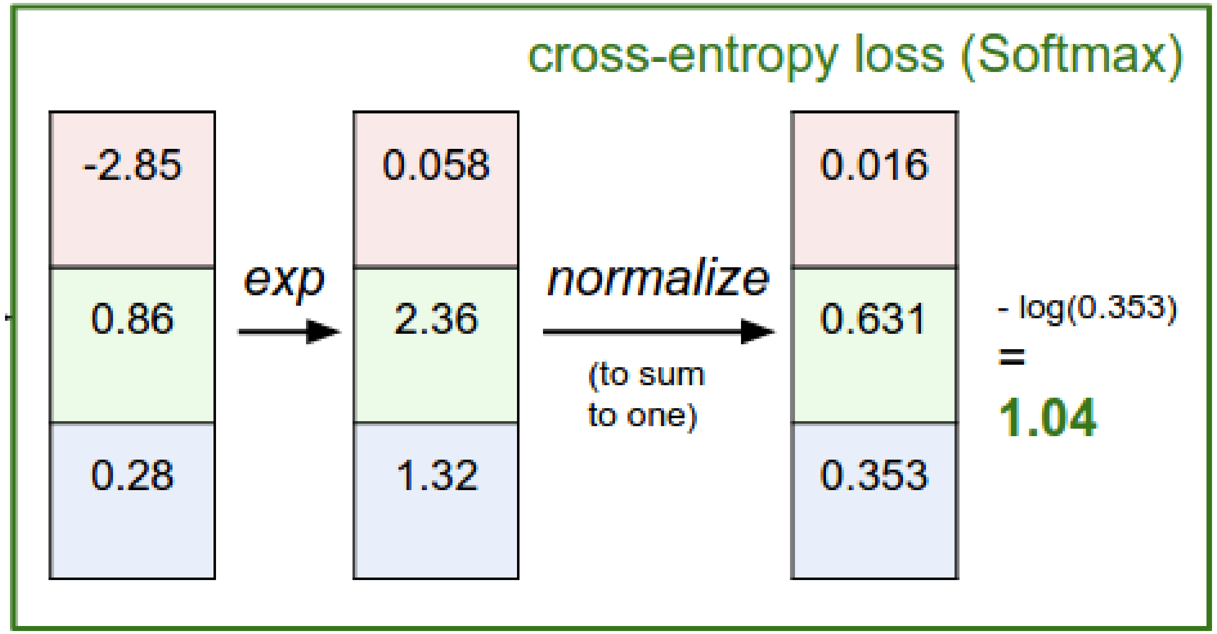
- Logistic Regression

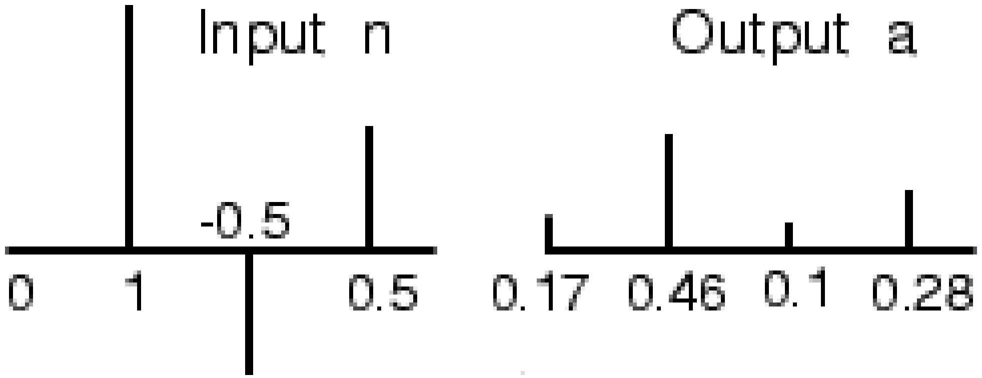


Softmax regression

- Softmax function

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$





Softmax regression

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

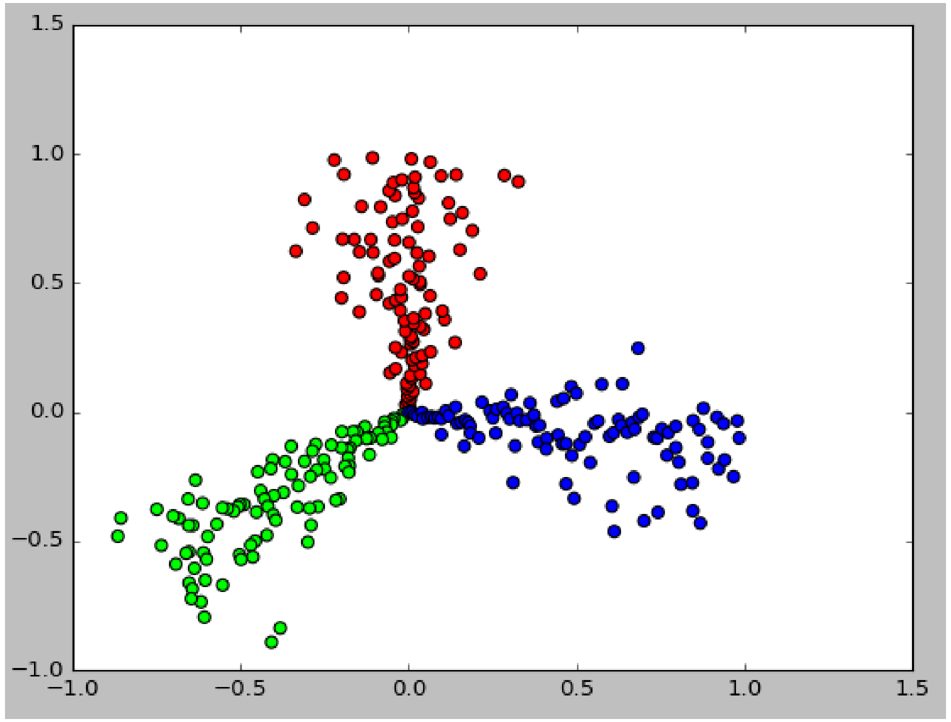
- Final model
  - $\hat{y} = f(x) = \boldsymbol{softmax}(\boldsymbol{W} \times x + b)$
  - Goal:
    Optimize parameter set $\theta = \{W, b\}$ to fit $\hat{y} \approx y$ where y is ground-truth label.

- Multinomial cross entropy
  - $\mathrm{C} = y^k \times -\log(\hat{y}^k)$
    where y is one-hot vector that indicate the ground-truth class
  - E.g.) $\mathrm{y} \in \{0,0,0,0,1,0,0,0,0,0\}$

Example: 2D classification

- Problem
  - Given 2D data, learn a classifier

- Dataset: randomly generated labeled 2D data
  - x = 2D coordinate
  - y = label



Build TF Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 2]) # 2 dimensional input
y = tf.placeholder(tf.float32, [None, 3]) # 3 classes
```

Build TF Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 2]) # 2 dimensional input
y = tf.placeholder(tf.float32, [None, 3]) # 3 classes

# Set model weights
W = tf.Variable(tf.zeros([2, 3]))
b = tf.Variable(tf.zeros([3]))
```

Build TF Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 2]) # 2 dimensional input
y = tf.placeholder(tf.float32, [None, 3]) # 3 classes

# Set model weights
W = tf.Variable(tf.zeros([2, 3]))
b = tf.Variable(tf.zeros([3]))

# Construct model
logit = tf.matmul(x, W) + b
pred = tf.nn.softmax(logit) # Softmax
```

Build TF Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 2]) # 2 dimensional input
y = tf.placeholder(tf.float32, [None, 3]) # 3 classes

# Set model weights
W = tf.Variable(tf.zeros([2, 3]))
b = tf.Variable(tf.zeros([3]))

# Construct model
logit = tf.matmul(x, W) + b
pred = tf.nn.softmax(logit) # Softmax

# Directly compute loss from logit (to ensure stability and avoid overflow)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logit,
labels=y))
```

Build TF Graph

```
# Place holders
x = tf.placeholder(tf.float32, [None, 2]) # 2 dimensional input
y = tf.placeholder(tf.float32, [None, 3]) # 3 classes

# Set model weights
W = tf.Variable(tf.zeros([2, 3]))
b = tf.Variable(tf.zeros([3]))

# Construct model
logit = tf.matmul(x, W) + b
pred = tf.nn.softmax(logit) # Softmax

# Directly compute loss from logit (to ensure stability and avoid overflow)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logit,
labels=y))

# Define optimizer and train_op
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Training Loop (same as logistic regression)

```
with tf.Session() as sess:
    # Initializing the variables
    sess.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(X_train.shape[0]/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs = X_train[i:i+batch_size,:]
            batch_ys = y_train[i:i+batch_size,:]
            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([train_op, cost], feed_dict={x: batch_xs, y: batch_ys})
            # Compute average loss
            avg_cost += c / total_batch

        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))
```

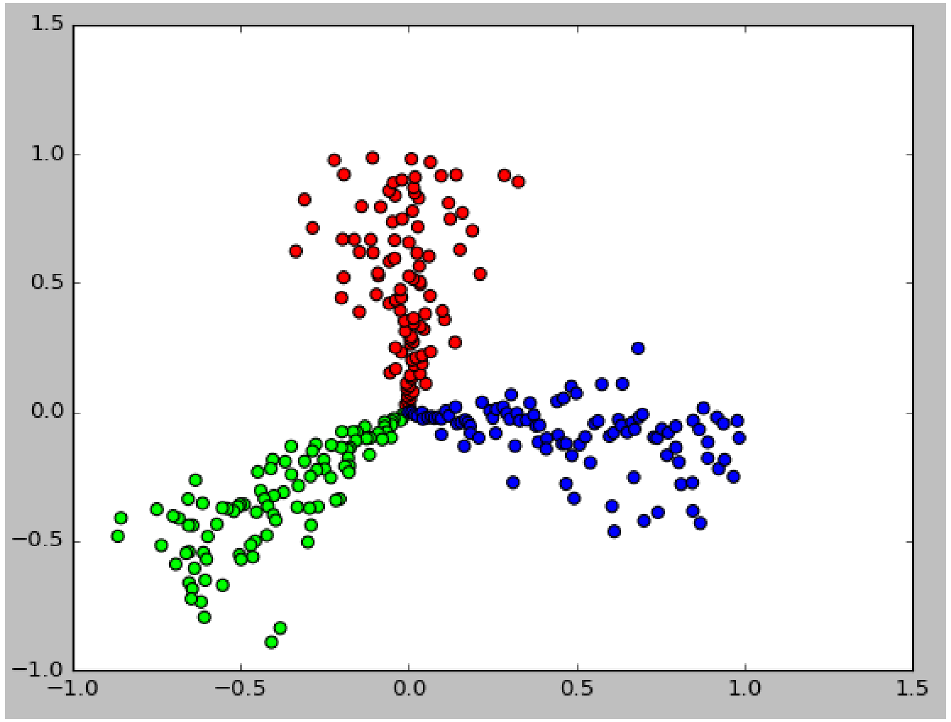
Result

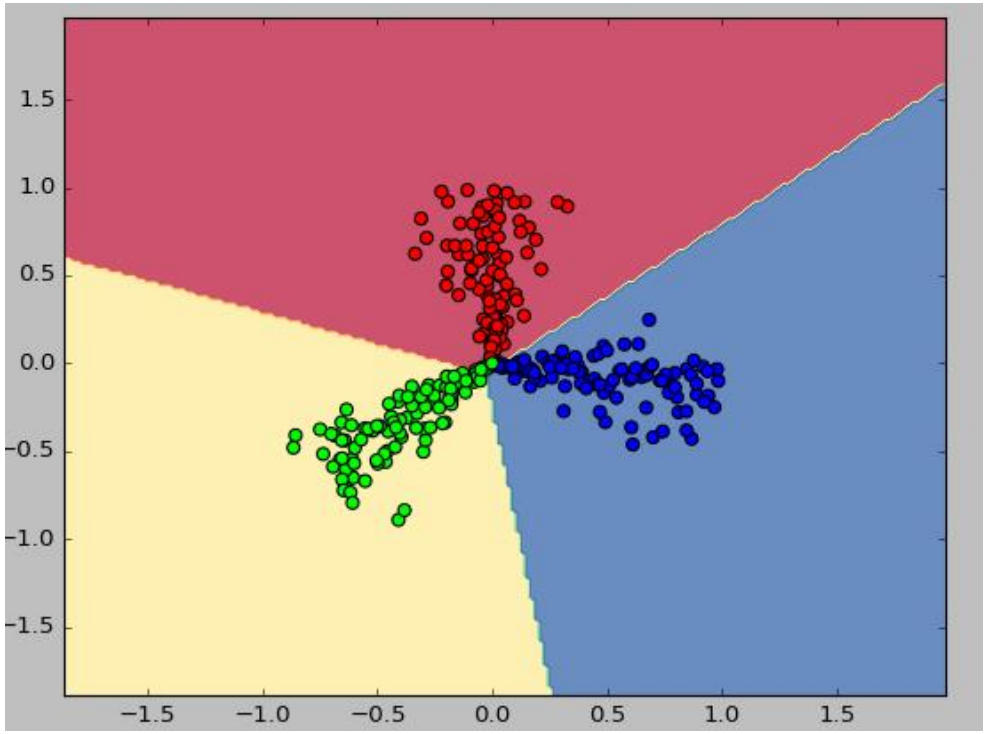
- Training log

Epoch: 0244 cost= 0.254794386
Epoch: 0245 cost= 0.254382825
Epoch: 0246 cost= 0.253973836
Epoch: 0247 cost= 0.253567417
Epoch: 0248 cost= 0.253163518
Epoch: 0249 cost= 0.252762134
Epoch: 0250 cost= 0.252363236
Optimization Finished!

Result

- Decision boundary

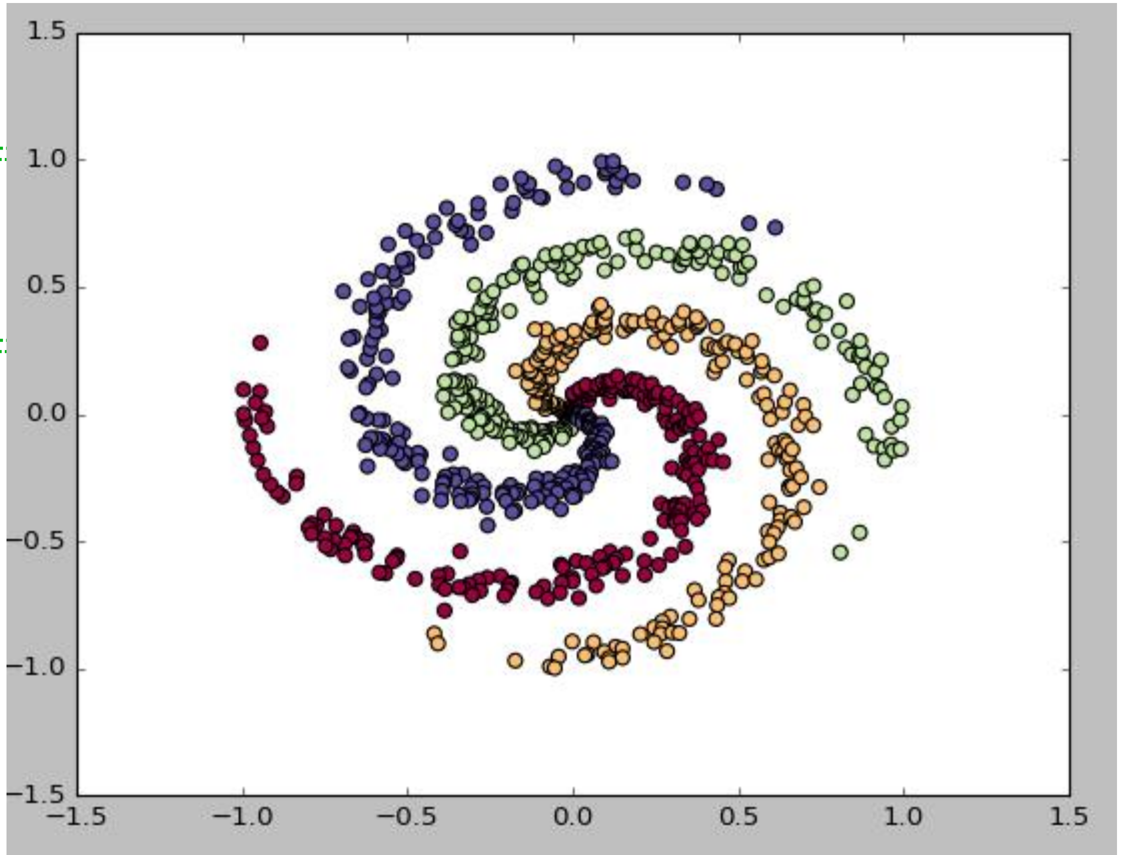




Practice: 2D spiral data

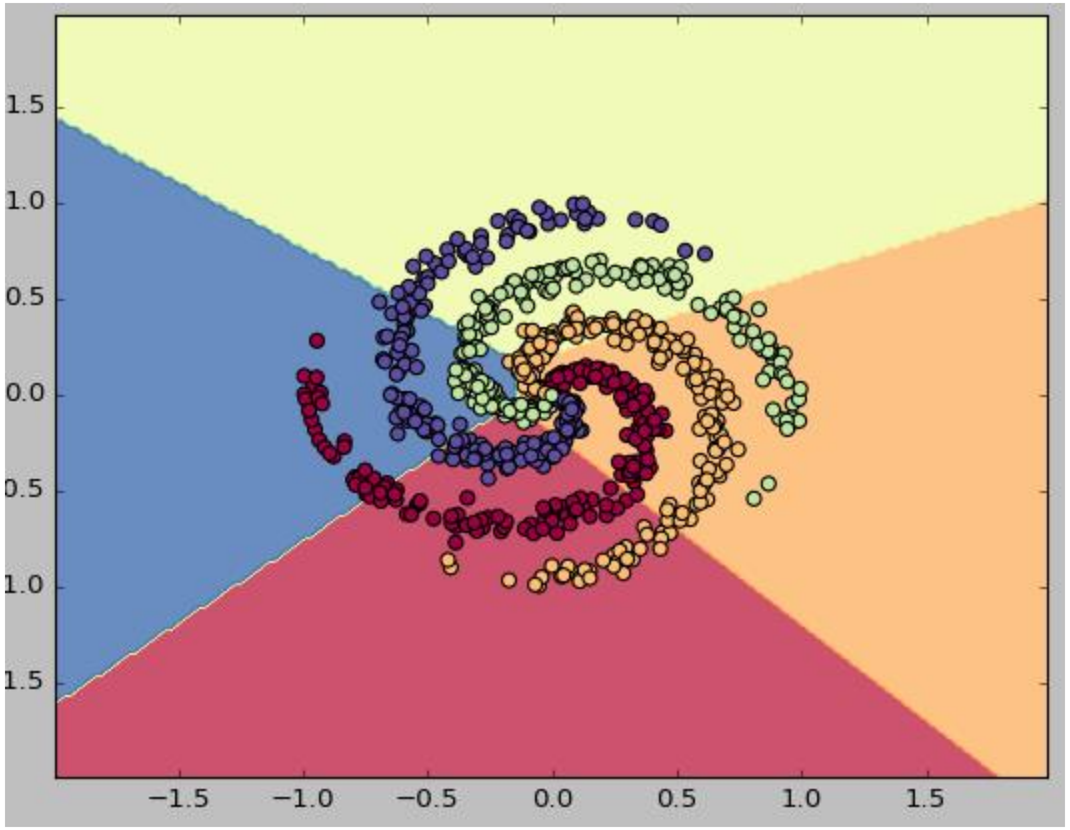
- Starter Code is provided:
  - Session1_softmax_regression_2D_spiral_starter.py
- Build TF graph your own

```
####### Build graph ###########################

# Write your code here

##############################################
```



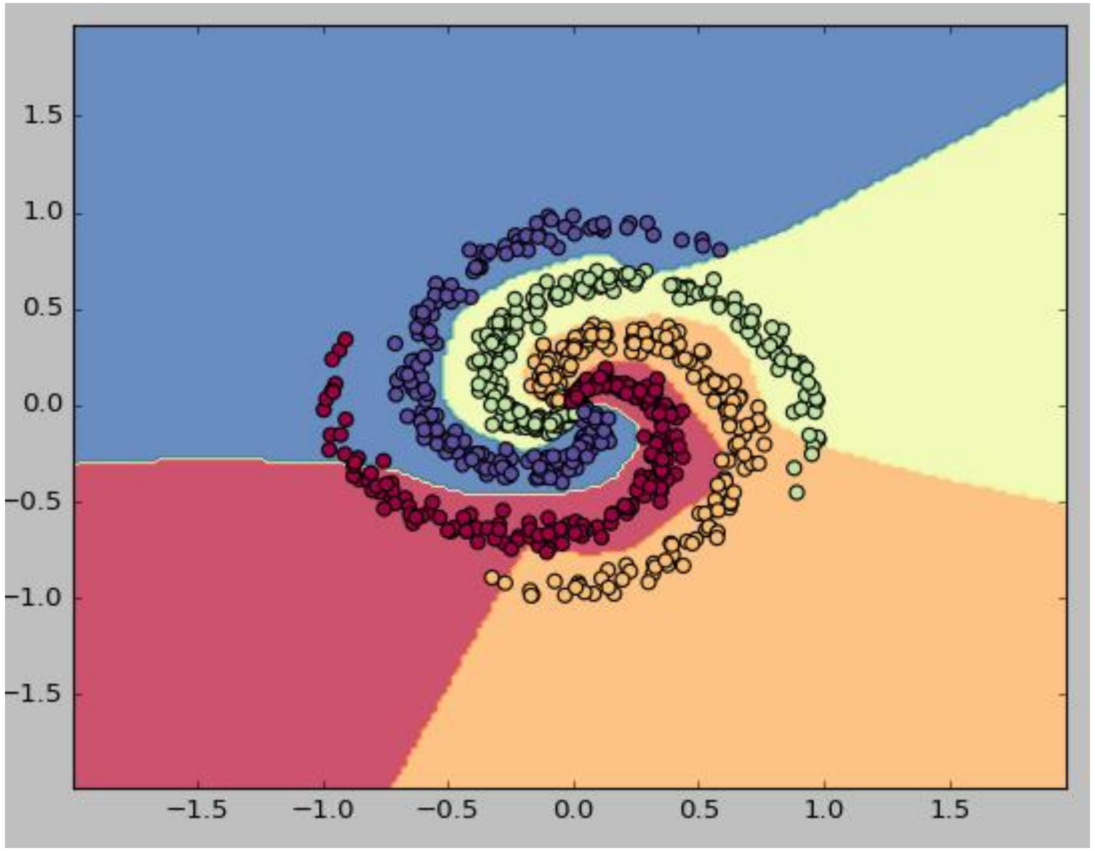
Practice: 2D spiral data

- Limitation of linear model



Practice: 2D spiral data

- Let's move on to Neural Network !



Acknowledgement

- Stanford CS231n
  - http://cs231n.stanford.edu/
  - http://cs231n.github.io/

- Andrew Ng's ML course
  - https://www.coursera.org/learn/machine-learning

- 모두를 위한 머신러닝/딥러닝 강의
  - https://hunkim.github.io/ml/