

Học máy với Python

Session 3

Tổng quan

- ❑ Computation graph và Back-Prop
- ❑ Làm quen với tensorflow
 - > Một số vấn đề cơ bản
 - > Triển khai Multi-layer Perceptron (MLP)

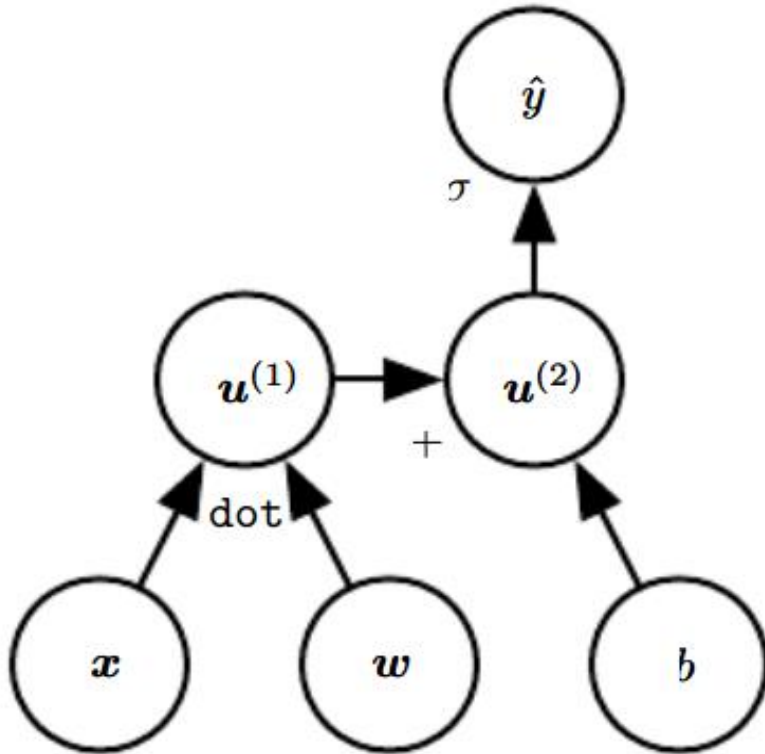
Computation graph và Back-prop

□ Khái niệm computation graph:

- > Bao gồm các node và cạnh
- > Mỗi node là 1 biến
- > Mỗi cạnh có hướng từ node x tới node y là một phép toán mà từ x sinh ra y

Computation graph và Back-prop

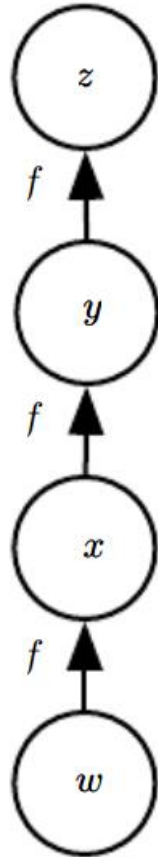
□ Ví dụ về computation graph:



$$\hat{y} = \sigma \left(x^{\top} w + b \right)$$

Computation graph và Back-prop

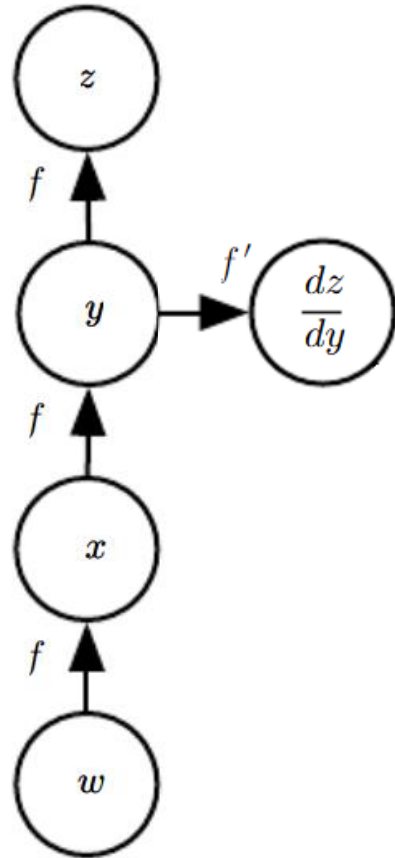
- Back-propagation: dựa trên **chain rule**



$$x = f(w), y = f(x), z = f(y)$$

Computation graph và Back-prop

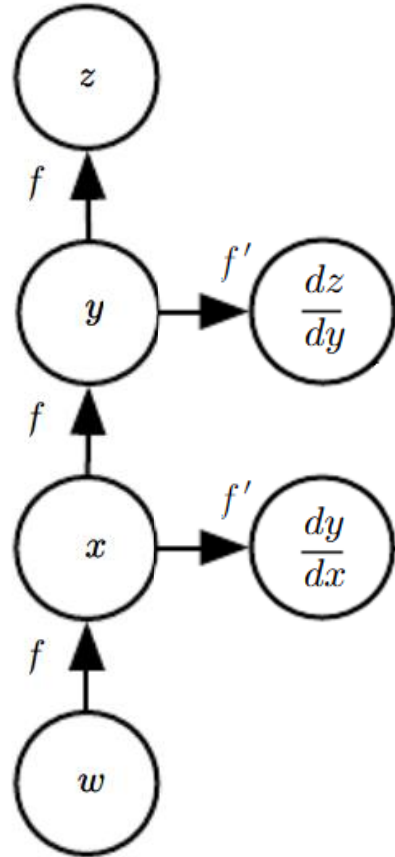
- Back-propagation: dựa trên **chain rule**



$$x = f(w), y = f(x), z = f(y)$$

Computation graph và Back-prop

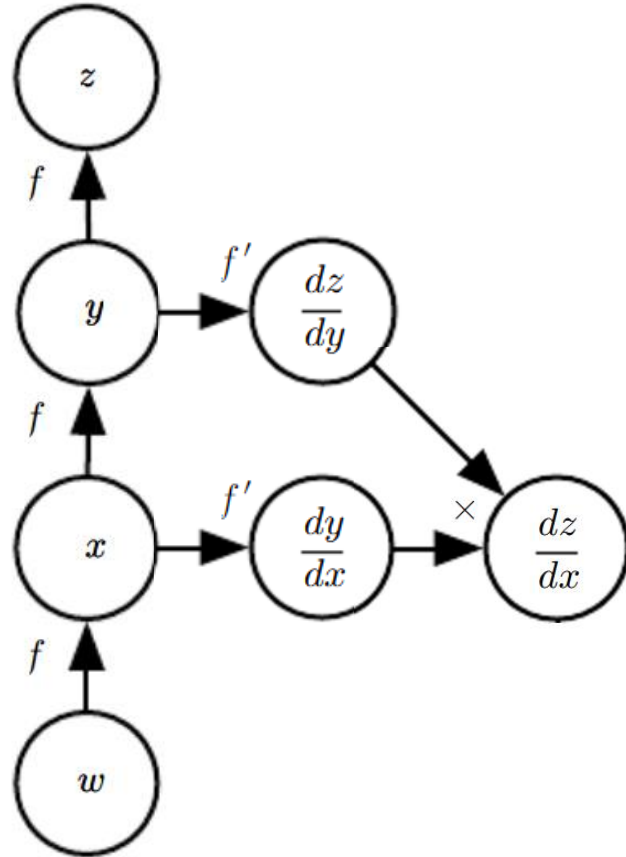
□ Back-propagation: dựa trên **chain rule**



$$x = f(w), y = f(x), z = f(y)$$

Computation graph và Back-prop

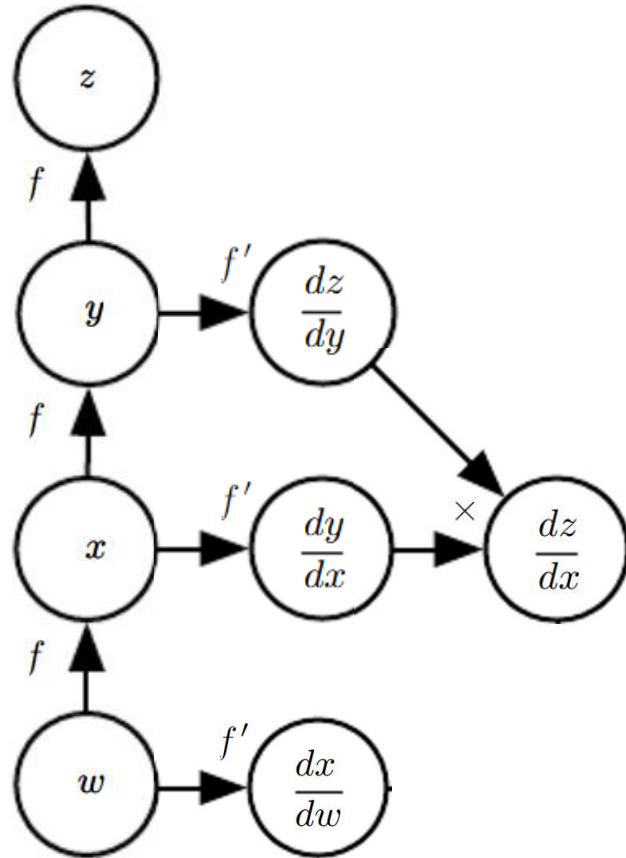
- Back-propagation: dựa trên **chain rule**



$$x = f(w), y = f(x), z = f(y)$$

Computation graph và Back-prop

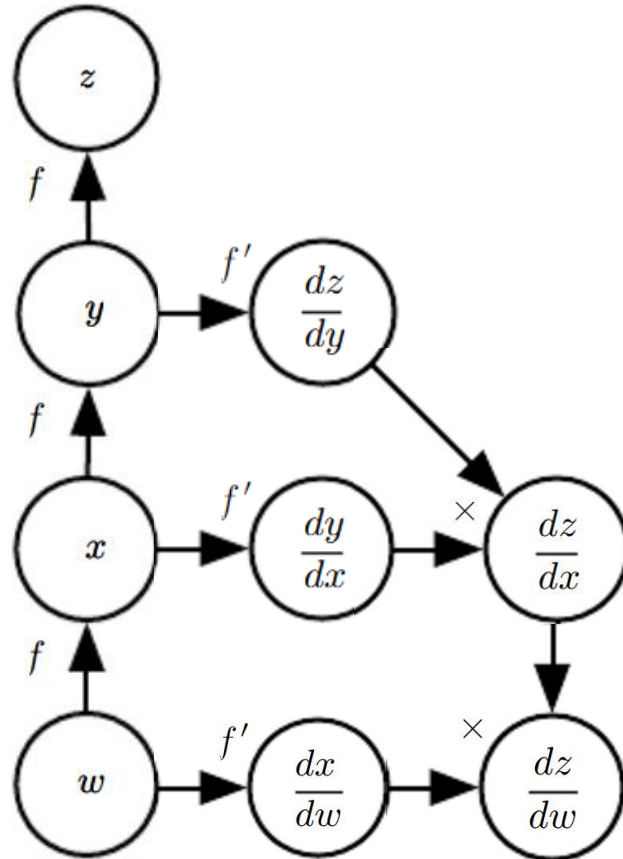
- Back-propagation: dựa trên **chain rule**



$$x = f(w), y = f(x), z = f(y)$$

Computation graph và Back-prop

- Back-propagation: dựa trên **chain rule**



$$x = f(w), y = f(x), z = f(y)$$

Computation graph và Back-prop

□ Back-propagation tổng quát:

=> Xem **chương 6, Deep learning book:**

<https://www.deeplearningbook.org/>

Computation graph và Back-prop

- ❑ Back-propagation hoạt động khi nào:
 - > Tồn tại biểu diễn dưới dạng computation graph cho model
 - > Tất cả các phép toán trong computation graph phải tồn tại đạo hàm

Làm quen với tensorflow

□ Nội dung chính:

1. Một số vấn đề cơ bản
2. Xây dựng mô hình Multi-layer Perceptron

1. Một số vấn đề cơ bản

- ❑ Tensorflow về bản chất là môi trường hỗ trợ việc xây dựng một computation graph, cung cấp các module có sẵn cho việc thực hiện Back-prop trên đó
- ❑ Tensorflow là một framework hỗ trợ việc thao tác với các tensor
- ❑ Tensor shape:
 - > scalar: shape = None
 - > vector: shape = [M,]
 - > matrix: shape = [M, N]
 - > 3-tensor: shape = [M, N, P]
(~ a list of matrix)
 - > n-tensor: [M₁, M₂, ..., M_n]
(~ a list of (n-1)-tensors)

1. Một số vấn đề cơ bản

□ Các loại tensor: về cơ bản có

> constant:

```
x = tf.constant(value)
```

> placeholder:

```
x = tf.placeholder(dtype, shape)
```

> variable:

```
x = tf.get_variable(name, shape, initializer)
```

1. Một số vấn đề cơ bản

❑ Ví dụ: xây dựng mô hình tính $\text{result} = x1 * x2$ với $x1=5$, $x2=6$

> import thư viện và xây dựng computation graph

```
3 import tensorflow as tf
4
5 x1 = tf.constant(5)
6 x2 = tf.constant(6)
7
8 result = tf.multiply(x1, x2)
```


1. Một số vấn đề cơ bản

❑ Ví dụ: xây dựng mô hình tính `result = x1 * x2` với `x1=5`, `x2=6`

> Nếu in thử giá trị của `result`, ta sẽ chỉ thu được cái “vỏ”
thay vì “giá trị”

```
12 print result
```

```
>> output: Tensor("Mul:0", shape=(), dtype=int32)
```

1. Một số vấn đề cơ bản

□ Ví dụ: xây dựng mô hình tính $\text{result} = x1 * x2$ với $x1=5$, $x2=6$

> Mở một phiên làm việc và chạy

```
19 sess = tf.Session()
20 output = sess.run(result)
21 print output
22 sess.close()
```

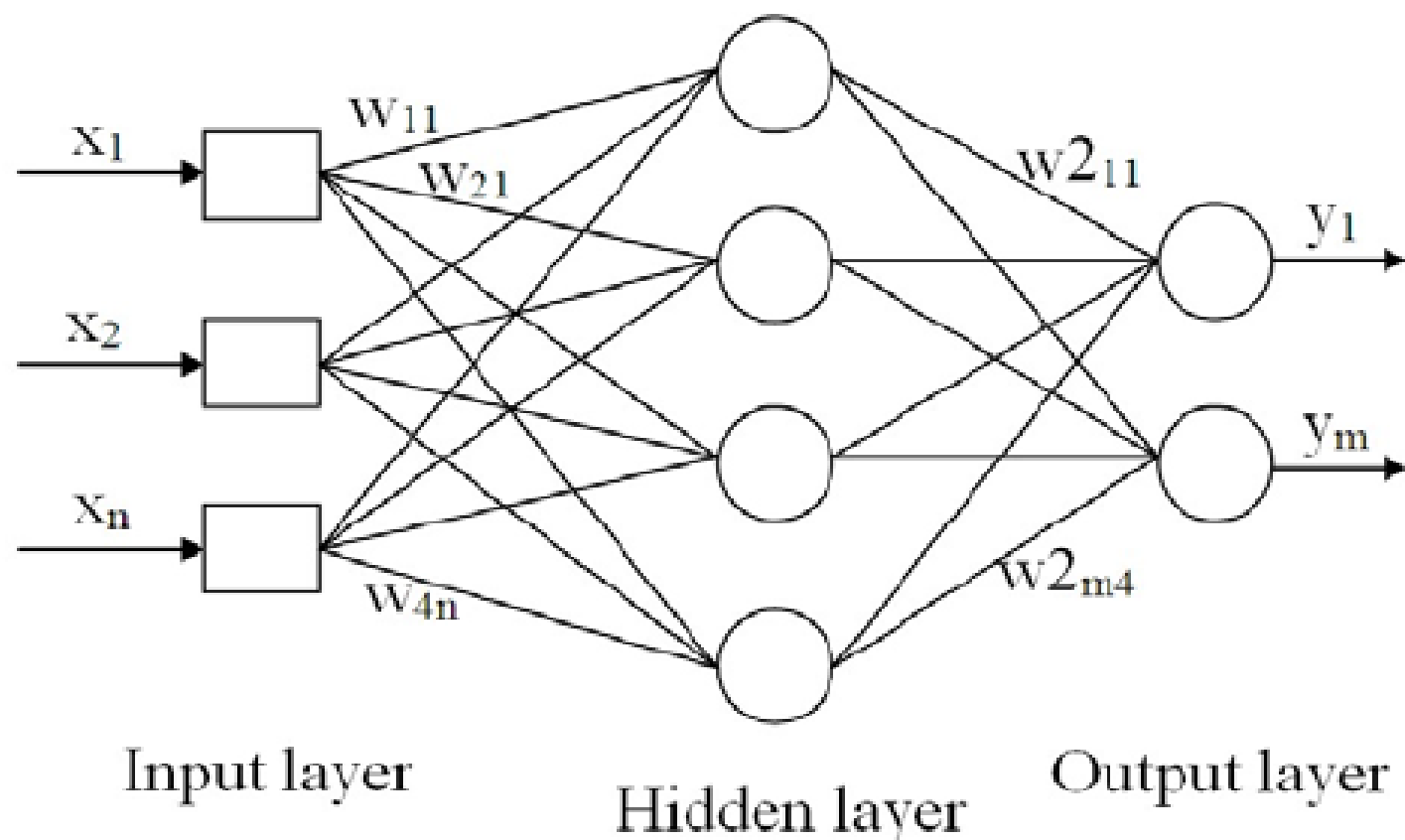
```
27 with tf.Session() as sess:
28     output = sess.run(result)
29     print result
```

>> output: 30

=> Nhận xét: nếu không dùng lệnh **with**, ta phải đóng session một cách thủ công (sess.close())

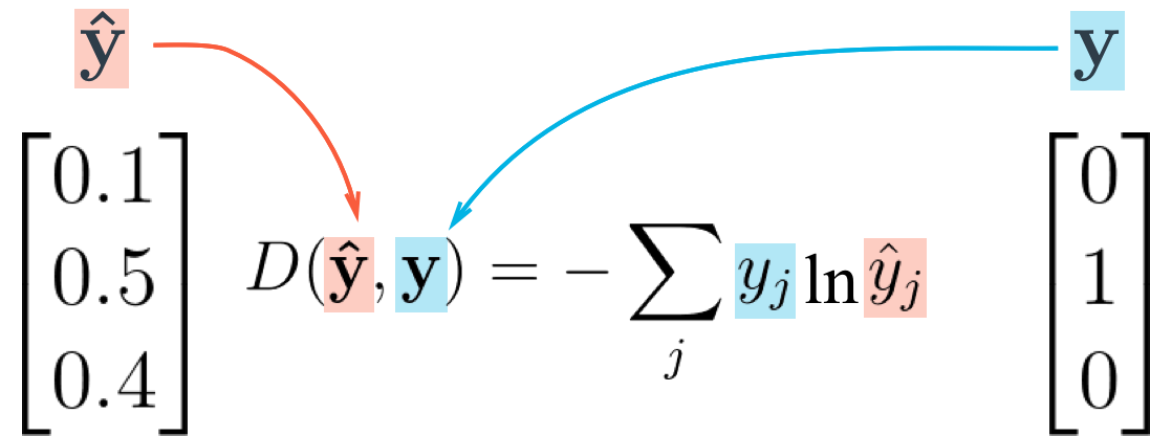
2. Triển khai MLP

□ Nhắc lại kiến thức:



2. Triển khai MLP

□ Hàm lỗi phân lớp:



The diagram illustrates the cross-entropy loss calculation. On the left, a predicted probability vector $\hat{\mathbf{y}}$ is shown as a column vector with values 0.1, 0.5, and 0.4. On the right, a target label vector \mathbf{y} is shown as a column vector with values 0, 1, and 0. A red arrow points from the $\hat{\mathbf{y}}$ label to the first element of the vector, and a blue arrow points from the \mathbf{y} label to the second element of the vector. The equation $D(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_j y_j \ln \hat{y}_j$ is centered between the two vectors, with the y_j term highlighted in blue and the \hat{y}_j term highlighted in red.

$$\begin{bmatrix} 0.1 \\ 0.5 \\ 0.4 \end{bmatrix} \quad D(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_j y_j \ln \hat{y}_j \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\Rightarrow \text{cross-entropy}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{k=1}^K y_k \cdot \log_k(\hat{y}_k)$$

2. Triển khai MLP

□ Mô hình hóa:

$\text{hidden} = \mathbf{x} \cdot \mathbf{W}_1 + \mathbf{b}_1$ # $\mathbf{X}: [1, K]$; $\mathbf{W}_1: [K, M]$; $\mathbf{b}_1: [M,]$

$\text{hidden} = \text{activation}(\text{hidden})$ # $[1, M]$

$\text{logits} = \text{hidden} \cdot \mathbf{W}_2 + \mathbf{b}_2$ # $[1, \text{num_classes}]$

$\text{outputs} = \text{softmax}(\text{logits})$ # $[1, \text{num_classes}]$

$\text{loss} = \text{cross-entropy}(\text{outputs}, \text{one-hot}(\text{real-label}))$

$\text{predicted-label} = \underset{i}{\text{argmax}}(\text{outputs}_i)$ # $[1,]$

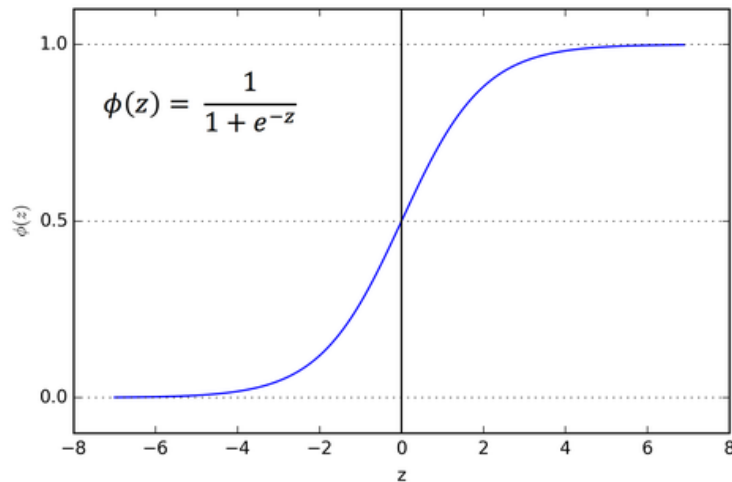
2. Triển khai MLP

□ Hàm softmax:

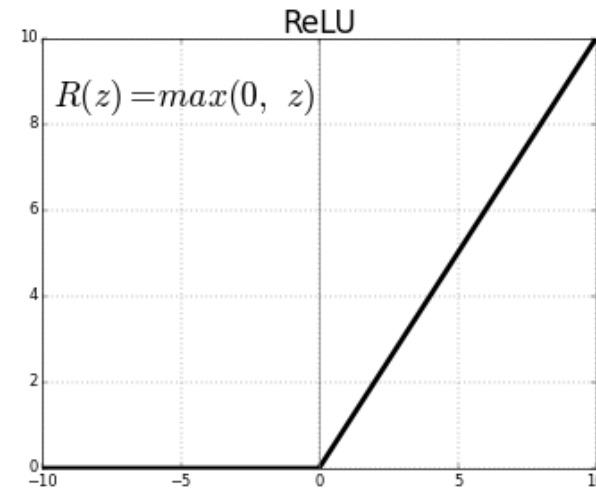
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

□ Hàm activation:

> sigmoid



> ReLU



2. Triển khai MLP

- ❑ Các bước để xây dựng mô hình trên tensorflow
 - > **B1**: Xây dựng computation graph
 - > **B2**: Mở một phiên làm việc (session), truyền (feed) dữ liệu vào graph và chạy

2. Triển khai MLP

- ❑ Xây dựng **class MLP**

```
69  class MLP:
70      def __init__(self, vocab_size, hidden_size):...
75
76      def build_graph(self):...
133
134      def trainer(self, loss, learning_rate):...
```


2. Triển khai MLP

- ❑ Xây dựng **class MLP**: hàm **init**

```
70 def __init__(self, vocab_size, hidden_size):  
71     self._vocab_size = vocab_size  
72     self._hidden_size = hidden_size
```

2. Triển khai MLP

- ❑ Xây dựng **class MLP**: hàm **build_graph**

```
74 def build_graph(self):  
75     self._X = tf.placeholder(tf.float32, shape=[None, self._vocab_size])  
76     self._real_Y = tf.placeholder(tf.int32, shape=[None, ])  
77  
78     weights_1 = tf.get_variable(  
79         name='weights_input_hidden',  
80         shape=(self._vocab_size, self._hidden_size),  
81         initializer=tf.random_normal_initializer(seed=2018),  
82     )  
83     biases_1 = tf.get_variable(  
84         name='biases_input_hidden',  
85         shape=(self._hidden_size),  
86         initializer=tf.random_normal_initializer(seed=2018)  
87     )
```

2. Triển khai MLP

- ❑ Xây dựng **class MLP**: hàm **build_graph**

```
88         weights_2 = tf.get_variable(  
89             name='weights_hidden_output',  
90             shape=(self._hidden_size, NUM_CLASSES),  
91             initializer=tf.random_normal_initializer(seed=2018),  
92         )  
93         biases_2 = tf.get_variable(  
94             name='biases_hidden_output',  
95             shape=(NUM_CLASSES),  
96             initializer=tf.random_normal_initializer(seed=2018)  
97         )
```

2. Triển khai MLP

❑ Xây dựng **class MLP**: hàm **build_graph**

```
99         hidden = tf.matmul(self._X, weights_1) + biases_1
100         hidden = tf.sigmoid(hidden)
101         logits = tf.matmul(hidden, weights_2) + biases_2
102
103         labels_one_hot = tf.one_hot(indices=self._real_Y, depth=NUM_CLASSES,
104                                     dtype=tf.float32)
105         loss = tf.nn.softmax_cross_entropy_with_logits(labels=labels_one_hot,
106                                                         logits=logits)
107         loss = tf.reduce_mean(loss)
```

2. Triển khai MLP

- ❑ Xây dựng **class MLP**: hàm **build_graph**: lấy **predicted-labels** để tính **accuracy**

```
109         probs = tf.nn.softmax(logits)
110         predicted_labels = tf.argmax(probs, axis=1)
111         predicted_labels = tf.squeeze(predicted_labels)
112
113         return predicted_labels, loss
```

2. Triển khai MLP

- ❑ Xây dựng **class MLP**: hàm **trainer**: chọn thuật toán để tối ưu hàm loss

```
115     def trainer(self, loss, learning_rate):  
116         train_op = tf.train.AdamOptimizer(learning_rate).minimize(loss)  
117         return train_op
```

2. Triển khai MLP

❑ B1: Xây dựng computation graph

```
170         # create a computation graph
171         with open('../datasets/words_idfs.txt') as f:
172             vocab_size = len(f.read().splitlines())
173
174         mlp = MLP(
175             vocab_size=vocab_size,
176             hidden_size=50
177         )
178         predicted_labels, loss = mlp.build_graph()
179         train_op = mlp.trainer(loss=loss, learning_rate=0.1)
```

2. Triển khai MLP

- ❑ B2: Mở một phiên làm việc, truyền dữ liệu và chạy

```
193         # open a session to run
194         with tf.Session() as sess:
195             train_data_reader, test_data_reader = load_dataset()
196             step, MAX_STEP = 0, 1000 ** 2
197
198             sess.run(tf.global_variables_initializer())
199             while step < MAX_STEP:
200                 train_data, train_labels = train_data_reader.next_batch()
201                 plabels_eval, loss_eval, _ = sess.run(
202                     [predicted_labels, loss, train_op],
203                     feed_dict={
204                         mlp._X: train_data,
205                         mlp._real_Y: train_labels
206                     }
207                 )
208                 step += 1
209                 print 'step: {}, loss: {}'.format(step, loss_eval)
```


2. Triển khai MLP

❑ B2: Mở một phiên làm việc, truyền dữ liệu và chạy

> hàm **load_dataset**

```
170     def load_dataset():
171         train_data_reader = DataReader(
172             data_path='../datasets/20news-train-tfidf.txt',
173             batch_size=50,
174             vocab_size=vocab_size
175         )
176         test_data_reader = DataReader(
177             data_path='../datasets/20news-test-tfidf.txt',
178             batch_size=50,
179             vocab_size=vocab_size
180         )
181         return train_data_reader, test_data_reader
```

2. Triển khai MLP

❑ B2: Mở một phiên làm việc, truyền dữ liệu và chạy

> class **DataReader**

```
121 class DataReader:
122     def __init__(self, data_path, batch_size, vocab_size):...
145
146     def next_batch(self):...
```

2. Triển khai MLP

❏ **class DataReader:** hàm **init**

```
121 class DataReader:
122     def __init__(self, data_path, batch_size, vocab_size):
123         self._batch_size = batch_size
124         with open(data_path) as f:
125             d_lines = f.read().splitlines()
126
127         self._data = []
128         self._labels = []
129         for data_id, line in enumerate(d_lines):...
130
131         self._data = np.array(self._data)
132         self._labels = np.array(self._labels)
133
134         self._num_epoch = 0
135         self._batch_id = 0
```

2. Triển khai MLP

❏ **class DataReader**: hàm **init**: line 129

```
129     for data_id, line in enumerate(d_lines):
130         vector = [0.0 for _ in range(vocab_size)]
131         features = line.split('<fff>')
132         label, doc_id = int(features[0]), int(features[1])
133         tokens = features[2].split()
134         for token in tokens:
135             index, value = int(token.split(':')[0]), \
136                             float(token.split(':')[1])
137             vector[index] = value
138         self._data.append(vector)
139         self._labels.append(label)
```

2. Triển khai MLP

❏ **class DataReader**: hàm **init**: line 129

```
129     for data_id, line in enumerate(d_lines):
130         vector = [0.0 for _ in range(vocab_size)]
131         features = line.split('<fff>')
132         label, doc_id = int(features[0]), int(features[1])
133         tokens = features[2].split()
134         for token in tokens:
135             index, value = int(token.split(':')[0]), \
136                             float(token.split(':')[1])
137             vector[index] = value
138         self._data.append(vector)
139         self._labels.append(label)
```

2. Triển khai MLP

❏ **class DataReader:** hàm **next_batch**

```
147     def next_batch(self):
148         start = self._batch_id * self._batch_size
149         end = start + self._batch_size
150         self._batch_id += 1
151
152         if end + self._batch_size > len(self._data):
153             end = len(self._data)
154             self._num_epoch += 1
155             self._batch_id = 0
156             indices = range(len(self._data))
157             random.seed(2018)
158             random.shuffle(indices)
159             self._data, self._labels = self._data[indices], self._labels[indices]
160
161         return self._data[start:end], self._labels[start:end]
```

2. Triển khai MLP

- ❑ **Lưu các tham số mô hình:** có thể lưu tại bất cứ bước lặp nào của quá trình training

```
trainable_variables = tf.trainable_variables()
for variable in trainable_variables:
    save_parameters(
        name=variable.name,
        value=variable.eval(),
        epoch=train_data_reader._num_epoch
    )
```

2. Triển khai MLP

❑ Lưu các tham số mô hình: hàm `save_parameters`

```
310 def save_parameters(name, value, epoch):
311     filename = name.replace(':', '-colon-') + '-epoch-{}.txt'.format(epoch)
312     if len(value.shape) == 1: # is a list
313         string_form = ','.join([str(number) for number in value])
314     else:
315         string_form = '\n'.join(['\n'.join([str(number)
316                                             for number in value[row]])
317                                 for row in range(value.shape[0])])
318
319     with open('../saved-paras/' + filename, 'w') as f:
320         f.write(string_form)
```


2. Triển khai MLP

❑ Lưu các tham số mô hình: hàm `save_parameters`

Nội dung file

1	-0.450477,0.69281,1.19272,1.40951,-1.39532,-1.90461,0.46044,
2	1.96216,-1.09849,0.427538,0.810766,1.39989,0.338526,-2.33891
3	1.64311,-0.911746,0.330272,0.972807,-1.04616,1.89584,-1.4516
4	-1.76935,-3.79334,-4.09076,-1.2307,-2.96816,-1.92106,3.73486
5	-0.370901,0.237953,-0.345284,-3.19793,2.69816,-1.19778,-2.04
6	-0.405415,0.575845,0.929856,-3.26487,-3.37635,0.116976,-1.21
7	0.32717,-0.669439,-1.87453,2.23346,1.30141,2.47804,-2.98699,
8	-0.627788,-0.914528,2.59333,1.59767,3.11817,-3.31037,-0.7706
9	-0.36349,0.0887211,-0.52825,-3.55277,-2.00161,-1.63245,-2.76

2. Triển khai MLP

❑ Khôi phục các tham số đã lưu:

sử dụng ngay sau khi khởi tạo variables

```
trainable_variables = tf.trainable_variables()
for variable in trainable_variables:
    saved_value = restore_parameters(variable.name, epoch)
    assign_op = variable.assign(saved_value)
    sess.run(assign_op)
```

2. Triển khai MLP

❏ Khôi phục các tham số đã lưu: Hàm `restore_parameters`

```
376 def restore_parameters(name, epoch):
377     filename = name.replace(':', '-colon-') + '-epoch-{}.txt'.format(epoch)
378     with open('../saved-paras/' + filename) as f:
379         lines = f.read().splitlines()
380         if len(lines) == 1: # is a vector
381             value = [float(number) for number in lines[0].split(',')]
382         else: # is a matrix
383             value = [[float(number) for number in lines[row].split(',')]
384                     for row in range(len(lines))]
385     return value
```

2. Triển khai MLP

❑ Đánh giá model trên test data:

```
349     num_true_preds = 0
350     while True:
351         test_data, test_labels = test_data_reader.next_batch()
352         test_plabels_eval = sess.run(
353             predicted_labels,
354             feed_dict={
355                 mlp._X: test_data,
356                 mlp._real_Y: test_labels
357             }
358         )
359         matches = np.equal(test_plabels_eval, test_labels)
360         num_true_preds += np.sum(matches.astype(float))
361
362         if test_data_reader._batch_id == 0:
363             break
364     print 'Epoch:', epoch
365     print 'Accuracy on test data:', num_true_preds / len(test_data_reader._data)
```

2. Triển khai MLP

❑ Đánh giá model trên test data:

```
335 test_data_reader = DataReader(  
336     data_path='../datasets/20news-test-tfidf.txt',  
337     batch_size=50,  
338     vocab_size=vocab_size  
339 )  
340 with tf.Session() as sess:  
341     epoch = 10  
  
342     trainable_variables = tf.trainable_variables()  
343     for variable in trainable_variables:  
344         saved_value = restore_parameters(variable.name, epoch)  
345         assign_op = variable.assign(saved_value)  
346         sess.run(assign_op)
```