मशीन लर्निंग का परिचय

चूंकि हम Python सीख रहे हैं, तो मशीन लर्निंग के बारे में भी जरूर बात करेंगे। क्योंकि इसके कई लाइब्रेरीज़ □□□□□□ में लिखे गए हैं। शुरुआत में उन्हें इंस्टॉल करके थोड़ा सा खेलेंगे।

□□□□□□□□□□

इसे इंस्टॉल करें।

```
$ pip install tensorflow
ERROR: Could not find a version that satisfies the requirement tensorflow
ERROR: No matching distribution found for tensorflow
```

(यह त्रुटि संदेश बताता है कि □□□□□□□□□□ का कोई संगत संस्करण नहीं मिला है।)

```
$ type python
python   `/usr/local/Cellar/python@3.9/3.9.1_6/bin/python3'
```

हालांकि, Tensorflow 2 केवल Python 3.5-3.8 को सपोर्ट करता है। हम 3.9 का उपयोग कर रहे हैं।

```
 % type python3
python3 /usr/bin/python3
% python3 -V
Python 3.8.2
```

मैंने देखा कि मेरे सिस्टम में python3 का संस्करण 3.8.2 है। यह □□□□□□ संस्करण के लिए pip कहाँ स्थापित है?

```
% python3 -m pip -V
pip 21.0.1 from /Users/lzw/Library/Python/3.8/lib/python/site-packages/pip (python 3.8)
```

संबंधित pip यहाँ है। तो मैं .zprofile फ़ाइल में थोड़ा बदलाव करूँगा। हाल ही में मैंने अपना shell बदला है। .zprofile पहले के .bash_profile के समान है। इसमें एक लाइन जोड़ दें।

```
alias pip3=/Users/lzw/Library/Python/3.8/bin/pip3
```

(यह कोड ब्लॉक है, इसे अनुवादित नहीं किया जाना चाहिए।)

इस तरह, हम Tensorflow को खेलने के लिए python3 और pip3 का उपयोग करते हैं।

```
% pip3 install tensorflow
...
                absl-py-0.12.0 astunparse-1.6.3 cachetools-4.2.1 certifi-2020.12.5 chardet-4.0.0 flatbuf
```

बहुत सारे लाइब्रेरी इंस्टॉल किए। वेबसाइट के एक उदाहरण का उपयोग किया।

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

यह कोड □□□□□ डेटासेट को लोड करता है और ट्रेनिंग और टेस्टिंग डेटा को 0 से 1 के बीच स्केल करता है। `x_train` और `x_test` में इमेज डेटा होता है, जबकि `y_train` और `y_test` में उनके संबंधित लेबल होते हैं। इमेज डेटा को 255.0 से विभाजित करके, पिक्सेल मान 0 से 1 के बीच स्केल हो जाते हैं, जो न्यूरल नेटवर्क के लिए अधिक उपयुक्त होता है।

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
```

यह कोड □□□□□□□□□□ के `Sequential` मॉडल का उपयोग करके एक न्यूरल नेटवर्क बनाता है। यह मॉडल 28□28 आकार के इनपुट डेटा को समतल (□□□□□□□) करता है, फिर 128 न्यूरॉन्स वाला एक घना (□□□□□) लेयर जोड़ता है जिसमें □□□□ एक्टिवेशन फंक्शन का उपयोग किया जाता है। इसके बाद, 20% ड्रॉपआउट लेयर जोड़ा जाता है, और अंत में 10 न्यूरॉन्स वाला एक और घना लेयर जोड़ा जाता है।

```
predictions = model(x_train[:1]).numpy()
print(predictions)
```

यह कोड मॉडल का उपयोग करके `x_train` डेटासेट के पहले नमूने के लिए भविष्यवाणियाँ (□□□□□□□□□□□□) करता है और उन्हें □□□□□ ऐरे के रूप में प्रिंट करता है।

```shell


```

```
$ /usr/bin/python3 tf.py
              https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 10s 1us/step
[[ 0.15477428 -0.3877643   0.0994779   0.07474922 -0.26219758 -0.03550266
   0.32226565 -0.37141111  0.10925996 -0.0115255 ]]
```

डेटासेट डाउनलोड हो गया है, और फिर परिणाम प्रदर्शित किया गया है।

अगले चरण में, चित्र वर्गीकरण के उदाहरण पर नजर डालते हैं।

```
# TensorFlow   tf.keras
import tensorflow as tf
```

सहायक लाइब्रेरीज़

□□□□□□□ □□□□□□ □□ □□ □□□□□□□ □□□□□□□□□□□□.□□□□□□□ □□ □□□□

```
print(tf.__version__)
```

त्रुटि।

```
ModuleNotFoundError: 'matplotlib'
```

इसे इंस्टॉल करें।

```
% pip3 install matplotlib
```

ठीक है।

```
$ /usr/bin/python3 image.py
2.4.1
```

कॉपी और पेस्ट उदाहरण कोड।

```
# TensorFlow   tf.keras
import tensorflow as tf
```

सहायक लाइब्रेरीज़

□□□□□□□ □□□□□□ □□ □□ □□□□□□□ □□□□□□□□□□□□.□□□□□□□ □□ □□□□

```
fashion_mnist = tf.keras.datasets.fashion_mnist
```

(□□□□□□_□□□□□□□, □□□□□□_□□□□□□□), (□□□□□_□□□□□□□, □□□□□_□□□□□□□) = □□□□□□□□_□□□□□□.□□□□□_□□□□□()

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
print(train_images.shape)
print(len(train_labels))
```

यहाँ परिणाम प्रदर्शित किया गया है। ध्यान दें कि यहाँ `train_images`, `train_labels`, `test_images`, `test_labels` हैं। यह डेटा को प्रशिक्षण डेटासेट और परीक्षण डेटासेट में विभाजित करता है।

```
(60000, 28, 28)
60000
```

यह कोड ब्लॉक एक □□□□□ ऐरे के आकार को दर्शाता है। पहली लाइन `(60000, 28, 28)` एक 3□ ऐरे का आकार दिखाती है, जिसमें 60,000 इमेज हैं और प्रत्येक इमेज का आकार 28□28 पिक्सेल है। दूसरी लाइन `60000` सिर्फ 60,000 का मान दिखाती है, जो संभवतः ऐरे में इमेज की कुल संख्या है।

अब चलिए, छवि को प्रिंट करने का प्रयास करते हैं।

```
print(train_images[0])
```

परिणाम देखें।

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0
    0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62
```

```
   54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134
  144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178
  107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216
  216 163 127 121 122 146 141  88 172  66]]
  ....
```

यहां कुछ परिणामों का अंश दिया गया है।

```
print(len(train_images[0][0]))
```

यह कोड □□□□□□□ में लिखा गया है और यह `train_images` नामक सूची के पहले तत्व के पहले तत्व की लंबाई (□□□□□□□) प्रिंट करता है। यह कोड हिंदी में अनुवाद नहीं किया जा सकता क्योंकि यह एक प्रोग्रामिंग भाषा का कोड है और इसे समझने के लिए □□□□□□□ की समझ आवश्यक है।

आउटपुट 28 है। तो यह स्पष्ट है कि यह एक 28 की चौड़ाई वाला मैट्रिक्स है। प्रिंट करना जारी रखें।

```
    print(len(train_images[0][0][0])
TypeError: 'numpy.uint8'           len()
```

तो यह बहुत स्पष्ट है। प्रत्येक छवि एक 28*28*3 का सरणी है। अंतिम आयाम सरणी में □□□ मान संग्रहीत होते हैं। हालांकि, हमें पता चला कि हमारा विचार गलत हो सकता है।

```
print(train_images[0][1][20])
```

(नोट: कोड ब्लॉक को अनुवादित नहीं किया जाता है, क्योंकि यह प्रोग्रामिंग कोड है और इसे हिंदी में अनुवादित करने की आवश्यकता नहीं है।)

```
0
```

```
print(train_images[0][1])
```

(यह कोड ब्लॉक को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह एक प्रोग्रामिंग कोड है और इसे अपरिवर्तित छोड़ दिया जाना चाहिए।)

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

प्रत्येक छवि 28*28 के सरणी (□□□□□) के रूप में है। कुछ समय तक प्रयास करने के बाद, हमने अंततः रहस्य जान लिया।

पहले आउटपुट ग्राफ़ को देखें।

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

(यह कोड ब्लॉक को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह प्रोग्रामिंग कोड है और इसे अपरिवर्तित छोड़ा जाना चाहिए।)

क्या आप दाईं ओर के रंग बार को देख रहे हैं? 0 से 250 तक। यह वास्तव में दो रंगों के बीच का ग्रेडिएंट है। लेकिन यह कैसे जानता है कि ये दो रंग कौन से हैं? हमने इसे कहाँ बताया है?

फिर हम दूसरी छवि को भी प्रिंट करते हैं।

```
plt.imshow(train_images[1])
```

(यह कोड ब्लॉक में है और इसे अनुवादित नहीं किया जाना चाहिए। यह □□□□□□ कोड है जो `train_images` सूची में दूसरी इमेज को प्रदर्शित करता है।)

बहुत दिलचस्प है। क्या यह `pyplot` डिपेंडेंसी लाइब्रेरी का डिफ़ॉल्ट व्यवहार है? चलिए, अब ऑफिशियल वेबसाइट द्वारा प्रदान किए गए कोड को चलाते हैं।

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

यह कोड एक 5□5 ग्रिड में 25 छवियों को प्रदर्शित करता है। प्रत्येक छवि के नीचे उसकी संबंधित लेबल (श्रेणी का नाम) दिखाया जाता है। `plt.xticks([])` और `plt.yticks([])` का उपयोग करके □ और □ अक्ष के टिक्स को हटा दिया जाता है, और `plt.grid(False)` का उपयोग करके ग्रिड को अक्षम किया जाता है। `plt.imshow()` का उपयोग करके छवि को दिखाया जाता है, और `plt.xlabel()` का उपयोग करके छवि के नीचे लेबल जोड़ा जाता है। अंत में, `plt.show()` का उपयोग करके प्लॉट को प्रदर्शित किया जाता है।

यहाँ ध्यान दें कि चित्र और उनके वर्गीकरण दिखाए गए हैं। अंत में हमें `cmp` पैरामीटर के बारे में पता चला। यदि `cmp` में कुछ नहीं लिखा जाए, तो यह निश्चित रूप से हमारे द्वारा देखे गए रंग का होगा। वास्तव में।

```
    plt.imshow(train_images[i])
```

इस बार हम `pyplot cmap` को खोजते हैं। कुछ संसाधन मिलते हैं।

```
    plt.imshow(train_images[i], cmap=plt.cm.PiYG)
```

कोड को संशोधित करें।

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)   ##
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.Blues)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

हालांकि, एक त्रुटि आई।

```
ValueError: num 1 <= num <= 10           , 11
```

यह क्या मतलब है। पहले का `5,5,i+1` आखिर क्या मतलब है। इसे `2` क्यों नहीं बदला जा सकता। हालांकि हम सहज रूप से जानते हैं कि यह शायद 5 पंक्तियों और 5 स्तंभों का मतलब है। लेकिन यह त्रुटि क्यों आई। `11` की गणना कैसे की गई। `num` का क्या मतलब है। `10` का क्या मतलब है। ध्यान दें कि `2*5=10`। तो शायद जब `i=11` होता है तो त्रुटि होती है। जब इसे `for i in range(10):` में बदला गया, तो निम्नलिखित परिणाम प्राप्त हुए।

यहां हम थोड़ा सा दस्तावेज़ीकरण देखेंगे और पता चलेगा कि `subplot(nrows, ncols, index, **kwargs)` है। हां, यहां तक हमें बहुत स्पष्ट हो गया है।

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    # plt.xticks([])
    plt.yticks([])
```

```
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.Blues)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

(यह कोड ब्लॉक को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह प्रोग्रामिंग कोड है और इसे अपरिवर्तित छोड़ दिया जाना चाहिए।)

ध्यान दें कि `0 25` जैसे नंबरों को `xticks` कहा जाता है। जब हम इस बॉक्स को ज़ूम इन या ज़ूम आउट करते हैं, तो यह अलग-अलग तरीके से दिखाई देता है।

□□□□_□□□□□□

ध्यान दें कि जब आप ज़ूम इन और ज़ूम आउट करते हैं, तो `xticks` और `xlabels` अलग-अलग तरीके से प्रदर्शित हो सकते हैं।

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

यह कोड □□□□□□□□□□ के मॉडल को कंपाइल करने के लिए उपयोग किया जाता है। इसमें:

- `optimizer='adam'` का उपयोग करके □□□□ ऑप्टिमाइज़र को सेट किया गया है।
- `loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)` का उपयोग करके स्पार्स कैटेगोरिकल क्रॉसएन्ट्रॉपी लॉस फ़ंक्शन को सेट किया गया है, जो लॉगिट्स से सीधे काम करता है।
- `metrics=['accuracy']` का उपयोग करके मॉडल की परफॉर्मेंस को मापने के लिए एक्यूरेसी मेट्रिक को सेट किया गया है।

□□□□□.□□□(□□□□□_□□□□□□, □□□□□_□□□□□□, □□□□□□=10)

□□□□_□□□□, □□□□_□□□ = □□□□□.□□□□□□□□□(□□□□_□□□□□□, □□□□_□□□□□□, □□□□□□□□=2)

```
print('\n      :', test_acc)
```

यहाँ मॉडल को परिभाषित करने के लिए `Sequential` क्लास का उपयोग किया गया है। इन पैरामीटर्स पर ध्यान दें: `28,28`, `128`, `relu`, `10`। ध्यान दें कि `compile` और `fit` की आवश्यकता है। `fit` का अर्थ है फिट करना। ध्यान दें कि `28,28` इमेज का आकार है।

```
Epoch 1/10
1875/1875 [==============================] - 2s 928us/step - loss: 0.6331 - accuracy: 0.7769
Epoch 2/10
1875/1875 [==============================] - 2s 961us/step - loss: 0.3860 - accuracy: 0.8615
Epoch 3/10
1875/1875 [==============================] - 2s 930us/step - loss: 0.3395 - accuracy: 0.8755
Epoch 4/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3071 - accuracy: 0.8890
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2964 - accuracy: 0.8927
Epoch 6/10
1875/1875 [==============================] - 2s 985us/step - loss: 0.2764 - accuracy: 0.8955
Epoch 7/10
1875/1875 [==============================] - 2s 961us/step - loss: 0.2653 - accuracy: 0.8996
Epoch 8/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2549 - accuracy: 0.9052
Epoch 9/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2416 - accuracy: 0.9090
Epoch 10/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2372 - accuracy: 0.9086
313/313 - 0s - loss: 0.3422 - accuracy: 0.8798
```

टेस्ट सटीकता: 0.879800021648407

```shell
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),    # 128 -> 28
    tf.keras.layers.Dense(10)
])
```

`Dense` के पहले पैरामीटर को संशोधित करें।

```
Epoch 1/10
1875/1875 [==============================] - 2s 714us/step - loss: 6.9774 - accuracy: 0.3294
```

```
Epoch 2/10
1875/1875 [==============================] - 1s 715us/step - loss: 1.3038 - accuracy: 0.4831
Epoch 3/10
1875/1875 [==============================] - 1s 747us/step - loss: 1.0160 - accuracy: 0.6197
Epoch 4/10
1875/1875 [==============================] - 1s 800us/step - loss: 0.7963 - accuracy: 0.6939
Epoch 5/10
1875/1875 [==============================] - 2s 893us/step - loss: 0.7006 - accuracy: 0.7183
Epoch 6/10
1875/1875 [==============================] - 1s 747us/step - loss: 0.6675 - accuracy: 0.7299
Epoch 7/10
1875/1875 [==============================] - 1s 694us/step - loss: 0.6681 - accuracy: 0.7330
Epoch 8/10
1875/1875 [==============================] - 1s 702us/step - loss: 0.6675 - accuracy: 0.7356
Epoch 9/10
1875/1875 [==============================] - 1s 778us/step - loss: 0.6508 - accuracy: 0.7363
Epoch 10/10
1875/1875 [==============================] - 1s 732us/step - loss: 0.6532 - accuracy: 0.7350
313/313 - 0s - loss: 0.6816 - accuracy: 0.7230
```

टेस्ट सटीकता: 0.7229999899864197

`Test accuracy` `Epoch` `fit` `128` , `a

```python
print(train_labels)
[9 0 0 ... 3 0 5]
print(len(train_labels))
60000
```

यह कोड `train_labels` को प्रिंट करता है, जो एक सरणी है जिसमें संख्याएँ हैं। यह सरणी प्रशिक्षण डेटा के लेबल को दर्शाती है। दूसरी लाइन में, `len(train_labels)` का उपयोग करके `train_labels` सरणी की लंबाई प्रिंट की जाती है, जो 60000 है।

इसका मतलब है कि इन श्रेणियों को `0` `9` तक के नंबरों से दर्शाया जाएगा। संयोग से `class_names` में भी 10 आइटम हैं।

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

फिर से इसे संशोधित करें।

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(5)    # 10 -> 5
])
```

यह कोड □□□□□□□□□□ के `tf.keras` □□□ का उपयोग करके एक सरल न्यूरल नेटवर्क मॉडल बनाता है। मॉडल में तीन लेयर्स हैं:

1. □□□□□□□ □□□□□: यह लेयर 28□28 आकार के इनपुट डेटा को एक सपाट (□□□□□□□) वेक्टर में बदल देती है, जिसका आकार 784 होगा।
2. □□□□□ □□□□□: यह एक पूरी तरह से जुड़ी हुई (□□□□□ □□□□□□□□□) लेयर है जिसमें 28 न्यूरॉन्स हैं और यह □□□□ एक्टिवेशन फंक्शन का उपयोग करती है।
3. □□□□□ □□□□□: यह दूसरी पूरी तरह से जुड़ी हुई लेयर है जिसमें 5 न्यूरॉन्स हैं। यह लेयर किसी भी एक्टिवेशन फंक्शन का उपयोग नहीं करती है, जिसका अर्थ है कि यह रैखिक (□□□□□□) आउटपुट प्रदान करती है।

मॉडल का आउटपुट 5 क्लासेस के लिए होगा, जो पहले के 10 क्लासेस से बदलकर 5 कर दिया गया है।

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

यह कोड □□□□□□□□□□ के मॉडल को कंपाइल करने के लिए है। इसमें `adam` ऑप्टिमाइज़र का उपयोग किया गया है, जो मॉडल को ट्रेन करने के लिए एक लोकप्रिय ऑप्टिमाइज़र है। `loss` फ़ंक्शन के रूप में `SparseCategoricalCrossentropy` का उपयोग किया गया है, जो क्लासिफिकेशन प्रॉब्लम्स के लिए उपयुक्त है, और `from_logits=True` सेट किया गया है क्योंकि मॉडल के आउटपुट लॉगिट्स हैं। `metrics` में `accuracy` का उपयोग किया गया है, जो मॉडल के प्रदर्शन को मापने के लिए एक सामान्य मीट्रिक है।

```
model.fit(train_images, train_labels, epochs=10)
```

यह कोड □□□□□□□□□□ या □□□□□ में एक मॉडल को प्रशिक्षित करने के लिए उपयोग किया जाता है। यह `train_images` और `train_labels` डेटा का उपयोग करके मॉडल को 10 □□□□□□ (युग) तक प्रशिक्षित करता है। प्रत्येक □□□□□ में, मॉडल पूरे प्रशिक्षण डेटा पर एक बार पास करता है और अपने वजन (□□□□□□□) को अपडेट करता है।

गलती हो गई।

```
tensorflow.python.framework.errors_impl.InvalidArgumentError:          9          [0, 5)
         [[node sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossE
```

फ़ंक्शन कॉल स्टैक: □□□□□□_□□□□□□□□□

`Sequential` `Dense` `15` `Epoch`

```python
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(15)
])
```

यह कोड □□□□□□□□□□ के `tf.keras` □□□ का उपयोग करके एक सरल न्यूरल नेटवर्क मॉडल बनाता है। इस मॉडल में तीन लेयर्स हैं:

1. □□□□□□□ □□□□□: यह लेयर 28□28 आकार के इनपुट डेटा को एक 784-आयामी वेक्टर में समतल (□□□□□□□) करती है।
2. □□□□□ □□□□□ **(28 न्यूरॉन्स)**: यह एक घनी (□□□□□) लेयर है जिसमें 28 न्यूरॉन्स हैं और '□□□□' एक्टिवेशन फंक्शन का उपयोग किया गया है।
3. □□□□□ □□□□□ **(15 न्यूरॉन्स)**: यह एक और घनी लेयर है जिसमें 15 न्यूरॉन्स हैं। इस लेयर में कोई एक्टिवेशन फंक्शन नहीं है, जिसका अर्थ है कि यह रैखिक (□□□□□□) आउटपुट देगी।

यह मॉडल आमतौर पर छवि वर्गीकरण (□□□□□ □□□□□□□□□□□□□□□) जैसे कार्यों के लिए उपयोग किया जाता है।

```python
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

यह कोड □□□□□□□□□□ के मॉडल को कंपाइल करने के लिए उपयोग किया जाता है। इसमें:

- □ □□□□□□□□□□**='**□□□□**'**: मॉडल को ट्रेन करने के लिए □□□□ ऑप्टिमाइज़र का उपयोग किया जाता है।
- □ □□□□**=**□□**.**□□□□□**.**□□□□□□**.**□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□**(**□□□□_□□□□□□**=**□□□□**)**: लॉस फ़ंक्शन के रूप में □□□□□□ □□□□□□□□□□ □□□□□□□□□□□□ का उपयोग किया जाता है, जो लॉगिट्स से सीधे काम करता है।
- □ □□□□□□□**=[**'□□□□□□□□**']**: मॉडल के प्रदर्शन को मापने के लिए सटीकता (□□□□□□□□) मेट्रिक का उपयोग किया जाता है।

```python
model.fit(train_images, train_labels, epochs=15)  # 10 -> 15
```

□□□□_□□□□, □□□□_□□□ = □□□□□.□□□□□□□□□(□□□□_□□□□□□, □□□□_□□□□□□, □□□□□□□=2)

```python
print('\n        :', test_acc)
```

```
Epoch 1/15
1875/1875 [==============================] - 2s 892us/step - loss: 6.5778 - accuracy: 0.3771
Epoch 2/15
1875/1875 [==============================] - 2s 872us/step - loss: 1.3121 - accuracy: 0.4910
Epoch 3/15
1875/1875 [==============================] - 2s 909us/step - loss: 1.0900 - accuracy: 0.5389
Epoch 4/15
1875/1875 [==============================] - 1s 730us/step - loss: 1.0422 - accuracy: 0.5577
Epoch 5/15
1875/1875 [==============================] - 1s 709us/step - loss: 0.9529 - accuracy: 0.5952
Epoch 6/15
1875/1875 [==============================] - 1s 714us/step - loss: 0.9888 - accuracy: 0.5950
Epoch 7/15
1875/1875 [==============================] - 1s 767us/step - loss: 0.8678 - accuracy: 0.6355
Epoch 8/15
1875/1875 [==============================] - 1s 715us/step - loss: 0.8247 - accuracy: 0.6611
Epoch 9/15
1875/1875 [==============================] - 1s 721us/step - loss: 0.8011 - accuracy: 0.6626
Epoch 10/15
1875/1875 [==============================] - 1s 711us/step - loss: 0.8024 - accuracy: 0.6622
Epoch 11/15
1875/1875 [==============================] - 1s 781us/step - loss: 0.7777 - accuracy: 0.6696
Epoch 12/15
1875/1875 [==============================] - 1s 724us/step - loss: 0.7764 - accuracy: 0.6728
Epoch 13/15
1875/1875 [==============================] - 1s 731us/step - loss: 0.7688 - accuracy: 0.6767
Epoch 14/15
1875/1875 [==============================] - 1s 715us/step - loss: 0.7592 - accuracy: 0.6793
Epoch 15/15
1875/1875 [==============================] - 1s 786us/step - loss: 0.7526 - accuracy: 0.6792
313/313 - 0s - loss: 0.8555 - accuracy: 0.6418
```

टेस्ट सटीकता: 0.6417999863624573

15 `tf.keras.layers.Dense(88, activation='relu'),` 128 88

```python
```

```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

(यह कोड ब्लॉक है, इसे अनुवादित नहीं किया जाना चाहिए।)

अगला, भविष्यवाणी करते हैं। ध्यान दें कि `Sequential` ऊपर की तरह ही है। पैरामीटर `model` और `tf.keras.layers.Softmax()` पर ध्यान दें।

```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
predictions = probability_model.predict(test_images)
```

इस कोड को हिंदी में समझाएं:

- `probability_model` एक `tf.keras.Sequential` मॉडल है जो दो लेयर्स से बना है: पहले मॉडल (`model`) और उसके बाद एक `Softmax` लेयर।
- `Softmax` लेयर का उपयोग मॉडल के आउटपुट को प्रोबेबिलिटी (संभावना) में बदलने के लिए किया जाता है।
- `predictions` वेरिएबल में `test_images` के लिए मॉडल की प्रेडिक्शन्स स्टोर की जाती हैं। ये प्रेडिक्शन्स प्रोबेबिलिटी के रूप में होती हैं, जो यह बताती हैं कि प्रत्येक इमेज किस क्लास से संबंधित होने की संभावना है।

```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
```

यह कोड एक फ़ंक्शन `plot_image` को परिभाषित करता है जो एक छवि (□□□□□) को प्लॉट करने के लिए उपयोग किया जाता है। यह फ़ंक्शन चार पैरामीटर लेता है:

1. `i`: इंडेक्स जो बताता है कि कौन सी छवि और लेबल को प्लॉट करना है।
2. `predictions_array`: एक ऐरे जिसमें मॉडल द्वारा की गई भविष्यवाणियाँ होती हैं।
3. `true_label`: एक ऐरे जिसमें छवियों के सही लेबल होते हैं।
4. `img`: एक ऐरे जिसमें छवियाँ होती हैं।

फ़ंक्शन के अंदर: - `true_label` और `img` को इंडेक्स `i` के आधार पर चुना जाता है। - `plt.grid(False)` ग्रिड को हटा देता है। - `plt.xticks([])` और `plt.yticks([])` और अक्ष पर टिक्स को हटा देते हैं।

इस फ़ंक्शन का उपयोग छवि को बिना किसी ग्रिड या अक्ष लेबल के प्लॉट करने के लिए किया जाता है।

```
plt.imshow(img, cmap=plt.cm.binary)

predicted_label = np.argmax(predictions_array)
if predicted_label == true_label:
    color = 'blue'
else:
    color = 'red'
```

□□□.□□□□□□□(“{} {:2.0□}% ({})”.□□□□□□□(□□□□□□_□□□□□□[□□□□□□□□□□_□□□□□□], 100*□□.□□□(□□□□□□□□□□□□□_□□□□□□), □□□□□□_□□□□□□[□□□□□_□□□□□□]), □□□□□□=□□□□□□)

इस कोड को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह एक प्रोग्रामिंग कोड है और इसे मूल रूप में ही रखना चाहिए।

```
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)
```

यह फ़ंक्शन `plot_value_array` एक बार ग्राफ़ बनाता है जो किसी विशेष इमेज के लिए मॉडल की भविष्यवाणियों (□□□□□□□□□□□□) को दर्शाता है। यह ग्राफ़ 10 संभावित क्लासेस (0 से 9 तक) के लिए प्रायिकता (□□□□□□□□□□□□) दिखाता है।

- `true_label` वास्तविक लेबल को दर्शाता है।
- `plt.grid(False)` ग्रिड को हटा देता है।
- `plt.xticks(range(10))` □-अक्ष पर 0 से 9 तक के नंबर दिखाता है।
- `plt.yticks([])` □-अक्ष पर कोई टिक नहीं दिखाता है।
- `plt.bar(range(10), predictions_array, color="#777777")` बार ग्राफ़ बनाता है जहां हर बार एक क्लास की प्रायिकता को दर्शाता है।
- `plt.ylim([0, 1])` □-अक्ष की सीमा को 0 से 1 तक सेट करता है।
- `predicted_label = np.argmax(predictions_array)` सबसे अधिक प्रायिकता वाले क्लास को चुनता है।

□□□□□□□□□[□□□□□□□□□□_□□□□□□].□□□_□□□□□□(‘□□□’) □□□□□□□□□[□□□□□_□□□□□□].□□□_□□□□□□(‘□□□□’)

```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

यह दर्शाता है कि इस चित्र की 99% संभावना है कि यह `Ankle boot` है। ध्यान दें कि `plot_image` बाईं ओर का चित्र दिखाता है। `plot_value_array` दाईं ओर का चित्र प्रदर्शित करता है।

```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```

(यह कोड ब्लॉक को हिंदी में अनुवाद करने की आवश्यकता नहीं है क्योंकि यह प्रोग्रामिंग कोड है और इसे मूल रूप में ही रखना चाहिए।)

ध्यान दें कि यहां केवल अधिक परीक्षण परिणाम प्रदर्शित किए गए हैं। इसलिए उपयोग प्रक्रिया हमें काफी स्पष्ट है। इसलिए हम अभी तक नहीं जानते कि पीछे की गणना कैसे की जाती है। लेकिन हम जानते हैं कि उनका उपयोग कैसे करना है। उनके पीछे कैलकुलस है। कैलकुलस को कैसे समझें?

मान लीजिए कि एक संख्या है, 1 से 100 के बीच, और आपको उसे अनुमान लगाना है। हर बार जब आप अनुमान लगाते हैं, मैं आपको बताऊंगा कि आपका अनुमान छोटा है या बड़ा।

- आपने 50 का अनुमान लगाया। मैंने कहा, “छोटा है।”
- आपने 80 का अनुमान लगाया। मैंने कहा, “बड़ा है।”
- आपने 65 का अनुमान लगाया। मैंने कहा, “बड़ा है।”
- आपने 55 का अनुमान लगाया। मैंने कहा, “छोटा है।”
- आपने 58 का अनुमान लगाया। मैंने कहा, “हां, सही अनुमान लगाया।”

मशीन लर्निंग, इस प्रक्रिया को पीछे से अनुकरण करने का एक तरीका है। हालांकि, यह थोड़ा जटिल होता है। इसमें कई 1 100 तक की संख्याएं हो सकती हैं, और कई संख्याओं का अनुमान लगाना पड़ सकता है। साथ ही, हर बार अनुमान लगाने के लिए कई गणनाएं करनी पड़ती हैं। और हर बार यह जांचने के लिए कि अनुमान बड़ा है या छोटा, कई गणनाएं करनी पड़ती हैं।

□□□□□□□□

इसे इंस्टॉल करें। यह 3.9 वर्जन के □□□□□□ को सपोर्ट करता है।

```
$ pip install torch torchvision
Collecting torch
  Downloading torch-1.8.0-cp39-none-macosx_10_9_x86_64.whl (120.6 MB)
     |                 | 120.6 MB 224 kB/s
Collecting torchvision
  Downloading torchvision-0.9.0-cp39-cp39-macosx_10_9_x86_64.whl (13.1 MB)
     |                 | 13.1 MB 549 kB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.9/site-packages (from torch) (1.20.1)
Collecting typing-extensions
  Downloading typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.9/site-packages (from torchvisio
Installing collected packages: typing-extensions, torch, torchvision
Successfully installed torch-1.8.0 torchvision-0.9.0 typing-extensions-3.7.4.3
```

जाँच करें।

```
import torch
x = torch.rand(5, 3)
print(x)
```

(यह कोड ब्लॉक को अनुवादित नहीं किया गया है क्योंकि यह प्रोग्रामिंग कोड है और इसे हिंदी में अनुवादित करने की आवश्यकता नहीं है।)

गलती हो गई।

```
Traceback (most recent call last):
  File "torch.py", line 1, in <module>
    import torch
  File "torch.py", line 2, in <module>
    x = torch.rand(5, 3)
AttributeError:              'torch'   'rand'         (                )
```

इस त्रुटि संदेश को □□□□□□□ पर खोजें। पता चला कि यह समस्या इसलिए हो रही थी क्योंकि हमारी फ़ाइल का नाम भी `torch` था। नाम टकरा रहा था। फ़ाइल का नाम बदलने के बाद सब ठीक हो गया।

```
tensor([[0.5520, 0.9446, 0.5543],
        [0.6192, 0.0908, 0.8726],
        [0.0223, 0.7685, 0.9814],
        [0.4019, 0.5406, 0.3861],
        [0.5485, 0.6040, 0.2387]])
```

एक उदाहरण ढूंढें।

```
# -*- coding: utf-8 -*-

import torch
import math
dtype = torch.float
device = torch.device("cpu")
# device = torch.device("cuda:0") #   GPU
```

यादृच्छिक इनपुट और आउटपुट डेटा बनाएं

□ = □□□□□□.□□□□□□□□□□(-□□□□.□□, □□□□.□□, 2000, □□□□□□□=□□□□□□□, □□□□□□=□□□□□□) □ = □□□□□□.□□□□(□)

वज़न को यादृच्छिक रूप से प्रारंभ करें

□ = □□□□□□.□□□□□□((), □□□□□□□=□□□□□□□, □□□□□□=□□□□□□) □ = □□□□□□.□□□□□□((), □□□□□□□=□□□□□□□, □□□□□□=□□□□□□) □ = □□□□□□.□□□□□□((), □□□□□□□=□□□□□□□, □□□□□□=□□□□□□) □ = □□□□□□.□□□□□□((), □□-□□□□□=□□□□□□□, □□□□□□=□□□□□□)

```
learning_rate = 1e-6
for t in range(2000):
    #       : y
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

#
```

```
loss = (y_pred - y).pow(2).sum().item()
if t % 100 == 99:
    print(t, loss)

#          a, b, c, d
grad_y_pred = 2.0 * (y_pred - y)
grad_a = grad_y_pred.sum()
grad_b = (grad_y_pred * x).sum()
grad_c = (grad_y_pred * x ** 2).sum()
grad_d = (grad_y_pred * x ** 3).sum()

#
a -= learning_rate * grad_a
b -= learning_rate * grad_b
c -= learning_rate * grad_c
d -= learning_rate * grad_d
```

□□□□□(□'परणिाम: □ = {□.□□□□()} + {□.□□□□()} □ + {□.□□□□()} □^2 + {□.□□□□()} □^3')

```shell
99 1273.537353515625
199 849.24853515625
299 567.4786987304688
399 380.30291748046875
499 255.92752075195312
599 173.2559814453125
699 118.2861328125
799 81.72274780273438
899 57.39331817626953
999 41.198158264160156
1099 30.41307830810547
1199 23.227672576904297
1299 18.438262939453125
1399 15.244369506835938
1499 13.113286972045898
```

```
1599 11.690631866455078
1699 10.740333557128906
1799 10.105220794677734
1899 9.6804780960083
1999 9.39621353149414
   : y = -0.011828352697193623 + 0.8360244631767273 x + 0.002040589228272438 x^2 + -0.09038365632295609
```

numpy लाइब्रेरी का उपयोग करके कोड देखें।

```
# -*- coding: utf-8 -*-
import numpy as np
import math
```

यादृच्छिक इनपुट और आउटपुट डेटा बनाएं

□ = □□.□□□□□□□□□(-□□□□.□□, □□□□.□□, 2000) □ = □□.□□□(□)

वज़नों को यादृच्छिक रूप से प्रारंभ करें

□ = □□.□□□□□□□.□□□□□□() □ = □□.□□□□□□□.□□□□□□() □ = □□.□□□□□□□.□□□□□□() □ = □□.□□□□□□□.□□□□□□()

```
learning_rate = 1e-6
for t in range(2000):
    #       : y
    # y = a + b x + c x^2 + d x^3
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

#
loss = np.square(y_pred - y).sum()
if t % 100 == 99:
    print(t, loss)

# Backprop          loss       a, b, c, d   gradients
grad_y_pred = 2.0 * (y_pred - y)
grad_a = grad_y_pred.sum()
```

```
grad_b = (grad_y_pred * x).sum()
grad_c = (grad_y_pred * x ** 2).sum()
grad_d = (grad_y_pred * x ** 3).sum()

#
a -= learning_rate * grad_a
b -= learning_rate * grad_b
c -= learning_rate * grad_c
d -= learning_rate * grad_d
```

□□□□□(□'परिणाम: □ = {□} + {□} □ + {□} □^2 + {□} □^3') "'

ध्यान दें कि यह दो तरीके हैं जिनसे गणना की जा सकती है।

ये दो उदाहरण हैं, जहां पहले □ और □ का एक समूह बनाया गया है। फिर यह माना गया है कि यह एक तीसरी डिग्री का समीकरण (क्यूबिक इक्वेशन) है। इसके बाद कुछ विधियों का उपयोग करके गुणांकों (□□□□□□□□□□□□□) को पुनरावृत्ति (□□□□□□□□□□) के माध्यम से गणना की गई है। ये एल्गोरिदम कैसे काम करते हैं? ध्यान दें कि यहां 2000 बार लूप चलाया गया है, और हर बार फिटिंग थोड़ी और सटीक होती जाती है। यहां हम इसके विस्तार में नहीं जाएंगे।

अंत में

अभी हमें मशीन लर्निंग के पीछे की गणना कैसे होती है, यह समझ में नहीं आता है। हालांकि, फिलहाल यह महत्वपूर्ण नहीं है। हम ऊपर दिए गए ज्ञान के समान ज्ञान का उपयोग करके बहुत सारे काम कर सकते हैं। हम मशीन लर्निंग का उपयोग टेक्स्ट, ऑडियो आदि को प्रोसेस करने के लिए भी कर सकते हैं। जब हम कुछ दर्जन उदाहरणों का परीक्षण कर लेंगे, तब सिद्धांत सीखना भी देर नहीं होगा।

अभ्यास

- □ छात्र ऊपर दिए गए तरीके से एक्सप्लोर करें।