```
import tensorflow as tf
import numpy as np


from matplotlib import pyplot as plt


# fro later graphiing
# %pip install pydot
# %pip install graphviz
import pydot
import graphviz


fashion_mnist = tf.keras.datasets.fashion_mnist

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
32768/29515 [=================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dataset
4423680/4422102 [==============================] - 0s 0us/step
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    #tf.keras.layers.Dense(20),
    tf.keras.layers.Dense(10)
])


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


model.fit(x_train, y_train, epochs=2)
```

```
Epoch 1/2
1875/1875 [==============================] - 3s 1ms/step - loss: 0.7873 - accura
Epoch 2/2
1875/1875 [==============================] - 2s 1ms/step - loss: 0.4758 - accura
<tensorflow.python.keras.callbacks.History at 0x7fe53febe518>
```

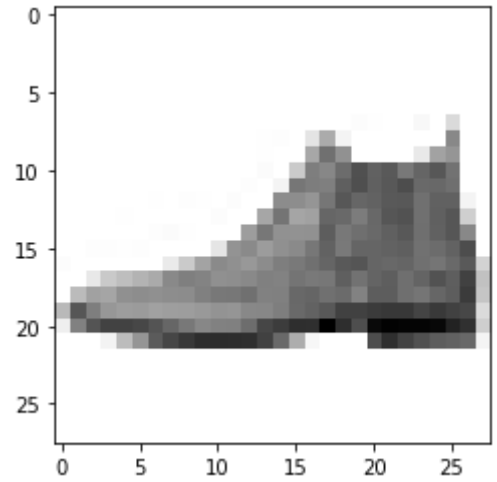
```
model.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.4736 - accuracy: 0.8339
[0.4735662639141083, 0.8338999748229980]
```

```
import copy
def relu(arr):
    cp_arr = copy.copy(arr)
    for i in range(len(cp_arr)):
        if cp_arr[i] < 0:
            cp_arr[i] = 0
    return cp_arr
```

```
plt.imshow(x_test[0], cmap=plt.cm.binary)
```

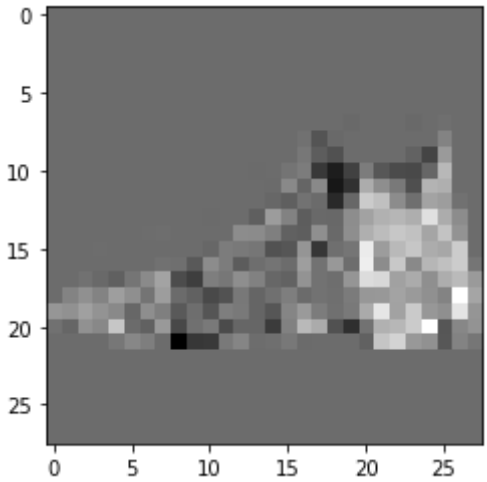
```
<matplotlib.image.AxesImage at 0x7fe538c3d6a0>
```




```
node_weights = model.layers[1].weights[0].numpy()
node_weights = np.rot90(node_weights)

fig, axs = plt.subplots(2, 5) #2*5 = 10 plots
row = -1
for i in range(node_weights.shape[0]):
    if i%5 == 0:
        row += 1
    axs[row][i%5].imshow(node_weights[i].reshape(28,28),\
                         cmap=plt.cm.binary)
```

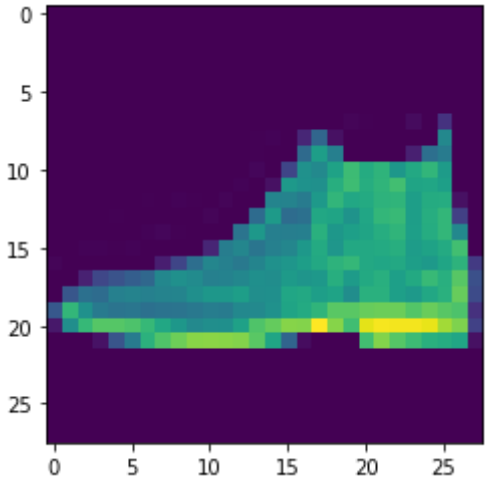
```
pixels = x_test[0].flatten()
pixels = pixels * node_weights[9]
plt.imshow(pixels.reshape(28,28), cmap=plt.cm.binary)
print(sum(pixels))
```

```
-6.706785031299984
```



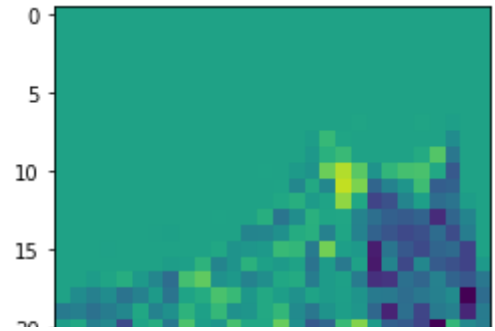
```
plt.imshow(x_test[0])
```

```
<matplotlib.image.AxesImage at 0x7fe537c15c18>
```



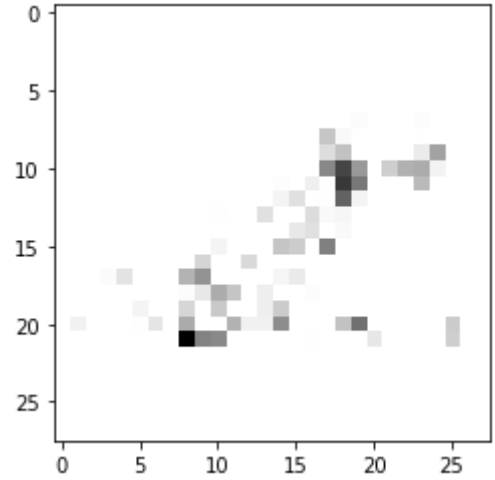
```
pixels = x_test[0].flatten()
pixels = pixels * node_weights[9]
plt.imshow(pixels.reshape(28,28))
print(sum(pixels))
```

```
-6.706785031299984
```




```
pixels = x_test[0].flatten()
pixels = pixels * node_weights[9]

rel_pixels = relu(pixels)
plt.imshow(rel_pixels.reshape(28,28), cmap=plt.cm.binary)
print(sum(rel_pixels))
```

```
1.924366977233805
```




```
pixels = x_test[0].flatten()
pixels = pixels * node_weights[9]

rel_pixels = relu(pixels)
plt.imshow(rel_pixels.reshape(28,28), cmap=plt.cm.binary)
print(sum(rel_pixels))

adjusted = rel_pixels - 3.57
sum(relu(adjusted))
```

```
1.924366977233805
0.0
```


```
predictions = model(x_test[:1])
print(predictions)
predictions = predictions.numpy()[0]
print(predictions)
max_pred = np.amax(predictions)
print(y_test[0],np.where(predictions == max_pred)[0])
```

```
tf.Tensor(
[[-6.582527  -6.5781016 -4.3868876 -4.602887  -4.1049194  3.6411786
  -3.695305   3.9138584  0.7109346  5.459909 ]], shape=(1, 10), dtype=float32)
[-6.582527  -6.5781016 -4.3868876 -4.602887  -4.1049194  3.6411786
 -3.695305   3.9138584  0.7109346  5.459909 ]
9 [9]
```

```
print(model.layers)
print(model.layers[1].bias) #layer 0 was flatten layer,
print(model.layers[1].weights) #only need weight and bias from layer 1
```

```
[<tensorflow.python.keras.layers.core.Flatten object at 0x7fe579a555f8>, <tensor
<tf.Variable 'dense/bias:0' shape=(10,) dtype=float32, numpy=
array([ 0.12425874, -0.29818404, -0.08662248,  0.11312255, -0.42115465,
        1.0733556 ,  0.22332531, -0.11901186, -0.37882358, -0.7103829 ],
      dtype=float32)>
[<tf.Variable 'dense/kernel:0' shape=(784, 10) dtype=float32, numpy=
array([[-0.06520453, -0.03350148, -0.18412533, ...,  0.06820215,
        -0.08782087, -0.05041914],
       [ 0.0156841 , -0.03910213, -0.04130744, ..., -0.11417839,
         0.03114059, -0.11527549],
       [ 0.1634894 , -0.28559834,  0.04229755, ..., -0.07888121,
        -0.1823168 , -0.12011182],
       ...,
       [-0.15815009, -0.12602362,  0.09260384, ..., -0.16931513,
        -0.30648112,  0.03206211],
       [-0.26949814, -0.08491736,  0.09266549, ..., -0.13465682,
        -0.24255377, -0.01093678],
       [-0.04617347, -0.15490828,  0.03880411, ..., -0.12376396,
        -0.19680697,  0.03586299]], dtype=float32)>, <tf.Variable 'dense/bias:0'
array([ 0.12425874, -0.29818404, -0.08662248,  0.11312255, -0.42115465,
        1.0733556 ,  0.22332531, -0.11901186, -0.37882358, -0.7103829 ],
      dtype=float32)>]
```

test_idx = 0

```
test_idx = 0 #choose an index of test data to test
vals= x_test[test_idx].flatten() #ID array of 784 pixel values


#interested in the weightss, one from the input layer (784 10)
weights = model.layers[1].weights[0].numpy()

print(vals.shape)
#each pixel has 10 weights, one for each node it goes through
print(weights.shape)
#I need the rows to be the 10 nodes and columns the 784 pixels
weights = np.rot90(weights)
print(weights.shape)
#I also need the bias for the layer whose weight I am taking)
bias = model.layers[1].bias
```

```
(784,)
(784, 10)
(10, 784)
```

```
#array to holdfinal 10 output weighted sums
node_outputs = []
#for each node in the 10 nodes
for i in range(weights.shape[0]):
    node_sum = 0 #perform weighted sumon all pixels
    #over each pixel in th einput
    for j in range(len(vals)):
        curr_pixel = vals[j]
        curr_w = weights[i][j]
        #perfomr weighted sum
        node_sum += curr_w * curr_pixel
    #add the node's bias
    node_sum += bias[i]
    #add this node's output to the list of outputs
    node_outputs.append(node_sum)

node_outputs = np.array(node_outputs)
#Because I rotated weights 90 degrees,
#   I get my answers backward and must flip them.
node_outputs = np.flip(node_outputs)


print(node_outputs)
max_pred = np.amax(node_outputs)
print(y_test[test_idx], np.where(node_outputs == max_pred)[0])
```

```
[-7.417168   -6.6587415  -4.4192767  -4.492684   -2.6104093   2.1466684
 -3.805508    3.9462483   0.79157424  6.29455   ]
9 [9]
```

test_idx = 30

```
test_idx = 30 #choose an index of test data to test
vals= x_test[test_idx].flatten() #ID array of 784 pixel values


#interested in the weightss, one from the input layer (784 10)
weights = model.layers[1].weights[0].numpy()

print(vals.shape)
#each pixel has 10 weights, one for each node it goes through
print(weights.shape)
#I need the rows to be the 10 nodes and columns the 784 pixels
weights = np.rot90(weights)
print(weights.shape)
#I also need the bias for the layer whose weight I am taking)
bias = model.layers[1].bias
```

```
(784,)
(784, 10)
(10, 784)
```

```
#array to holdfinal 10 output weighted sums
node_outputs = []
#for each node in the 10 nodes
for i in range(weights.shape[0]):
    node_sum = 0 #perform weighted sumon all pixels
    #over each pixel in th einput
    for j in range(len(vals)):
        curr_pixel = vals[j]
        curr_w = weights[i][j]
        #perfomr weighted sum
        node_sum += curr_w * curr_pixel
    #add the node's bias
    node_sum += bias[i]
    #add this node's output to the list of outputs
    node_outputs.append(node_sum)

node_outputs = np.array(node_outputs)
#Because I rotated weights 90 degrees,


print(node_outputs)
max_pred = np.amax(node_outputs)
print(y_test[test_idx], np.where(node_outputs == max_pred)[0])
```

```
[ -8.247157    4.8422604 -10.949278   -4.001472   -2.0579963  -5.8008657
  -6.2375865  -4.6530876 -10.994961  -12.460929 ]
8 [1]
```

x_test = 1000

```
test_idx = 1000 #choose an index of test data to test
vals= x_test[test_idx].flatten() #ID array of 784 pixel values


#interested in the weightss, one from the input layer (784 10)
weights = model.layers[1].weights[0].numpy()

print(vals.shape)
#each pixel has 10 weights, one for each node it goes through
print(weights.shape)
#I need the rows to be the 10 nodes and columns the 784 pixels
weights = np.rot90(weights)
print(weights.shape)
#I also need the bias for the layer whose weight I am taking)
bias = model.layers[1].bias
```

```
(784,)
(784, 10)
(10, 784)
```

```
node_outputs = []
#for each node in the 10 nodes
for i in range(weights.shape[0]):
    node_sum = 0 #perform weighted sumon all pixels
    #over each pixel in th einput
    for j in range(len(vals)):
        curr_pixel = vals[j]
        curr_w = weights[i][j]
        #perfomr weighted sum
        node_sum += curr_w * curr_pixel
    #add the node's bias
    node_sum += bias[i]
    #add this node's output to the list of outputs
    node_outputs.append(node_sum)

node_outputs = np.array(node_outputs)
#Because I rotated weights 90 degrees,


print(node_outputs)
max_pred = np.amax(node_outputs)
print(y_test[test_idx], np.where(node_outputs == max_pred)[0])
```

```
[-15.958203    -4.285274   -21.735277     0.89895624 -19.698717
  -6.798711    -5.7083592   -1.7551712   -7.4891577    1.1686974 ]
0 [9]
```

x_test = 9999

```
test_idx = 1000 #choose an index of test data to test
vals= x_test[test_idx].flatten() #ID array of 784 pixel values


#interested in the weightss, one from the input layer (784 10)
weights = model.layers[1].weights[0].numpy()

print(vals.shape)
#each pixel has 10 weights, one for each node it goes through
print(weights.shape)
#I need the rows to be the 10 nodes and columns the 784 pixels
weights = np.rot90(weights)
print(weights.shape)
#I also need the bias for the layer whose weight I am taking)
bias = model.layers[1].bias
```

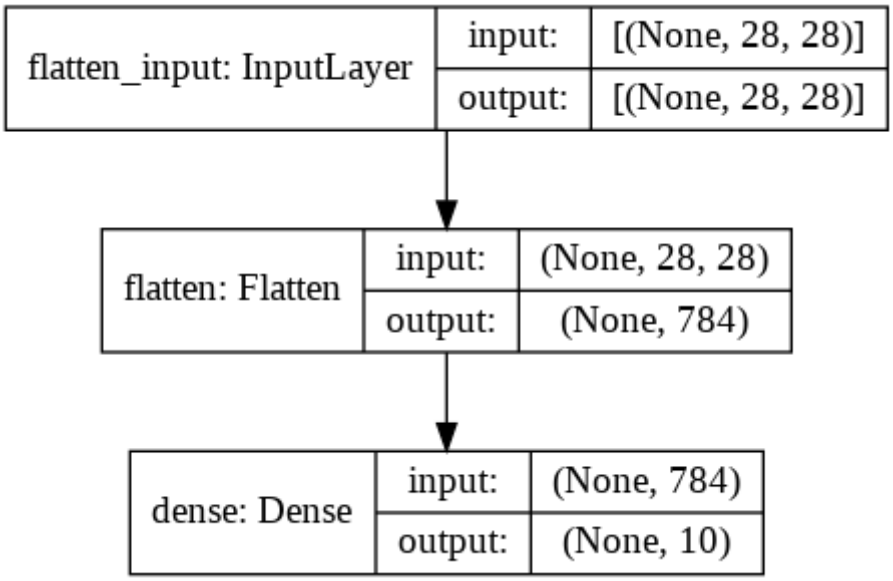
```
(784,)
(784, 10)
(10, 784)
```

```
node_outputs = []
#for each node in the 10 nodes
for i in range(weights.shape[0]):
    node_sum = 0 #perform weighted sumon all pixels
    #over each pixel in th einput
    for j in range(len(vals)):
        curr_pixel = vals[j]
        curr_w = weights[i][j]
        #perfomr weighted sum
        node_sum += curr_w * curr_pixel
    #add the node's bias
    node_sum += bias[i]
    #add this node's output to the list of outputs
    node_outputs.append(node_sum)

node_outputs = np.array(node_outputs)
#Because I rotated weights 90 degrees,


print(node_outputs)
max_pred = np.amax(node_outputs)
print(y_test[test_idx], np.where(node_outputs == max_pred)[0])
```

```
[-15.958203    -4.285274   -21.735277     0.89895624 -19.698717
  -6.798711    -5.7083592   -1.7551712   -7.4891577    1.1686974 ]
0 [9]
```

Plotting the Network Hierarchy

```
tf.keras.utils.plot_model(model, to_file='model_plot.png',\
                          show_shapes=True, show_layer_names=True)
```




```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 784)               0         
_________________________________________________________________
dense (Dense)                (None, 10)                7850      
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```