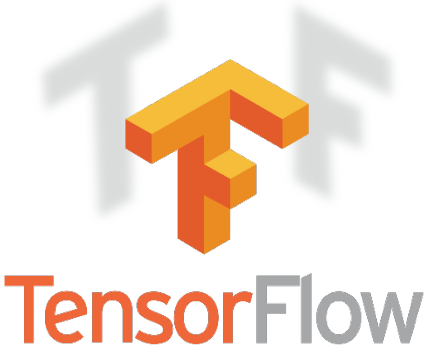
Intro to NN and Keras 2 Dense Neural Net Example – "to classify handwritten digits"

CJ Chung



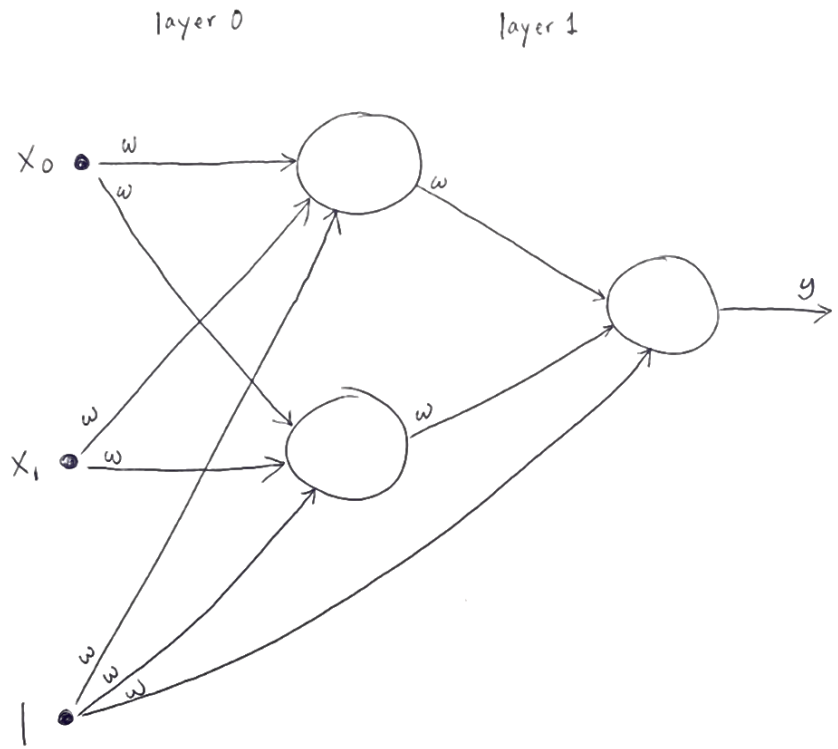
Purpose of reviewing this topic

- Reviewing basic NN with Keras topics
- For Hyperparameter Optimization with Keras using EA

Review: How are weight values updated using BP?

```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

model = Sequential([
    Dense(2, input_dim=2, activation='tanh'),
    Dense(1, activation='tanh')
])

model.compile(
    loss='mean_squared_error',
    optimizer=optimizers.SGD(learning_rate=0.1), # sto
    #optimizer=optimizers.RMSprop(learning_rate=0.01),
    metrics=['binary_accuracy']
)
model.fit(X, y, batch_size=1, epochs=200)
```

XOR.ipynb



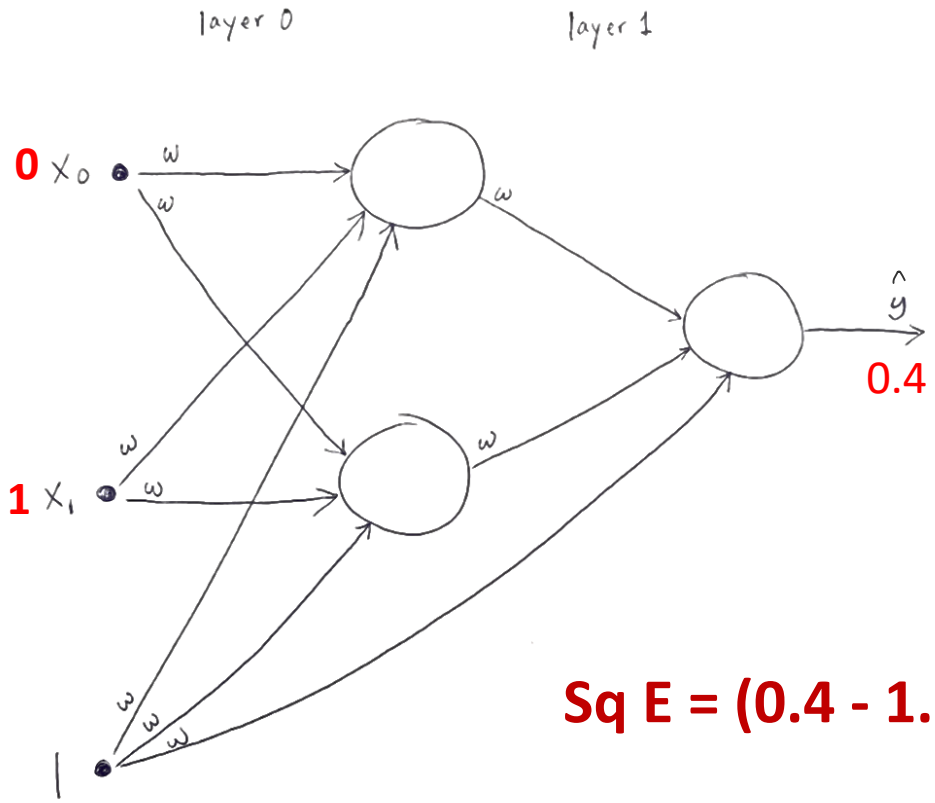
- batch_size: **Number of samples per gradient update**.
- An epoch is an iteration over the entire X and y data provided

- Total number of gradient updates: 4/1 * 200 = **800**

Review: How are weight values updated?

```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

model = Sequential([
    Dense(2, input_dim=2, activation='tanh'),
    Dense(1, activation='tanh')
])

model.compile(
    loss='mean_squared_error',
    optimizer=optimizers.SGD(learning_rate=0.1), # sto
    #optimizer=optimizers.RMSprop(learning_rate=0.01),
    metrics=['binary_accuracy']
)
model.fit(X, y, batch_size=1, epochs=200)
```

XOR.ipynb



What is the **squared error** when (0, 1) is presented and $\hat{y}$ is 0.4?

Sq E = $(0.4 - 1.0)^2$

Note that the batch_size is 1

Data Representation for NN/Tensorflow

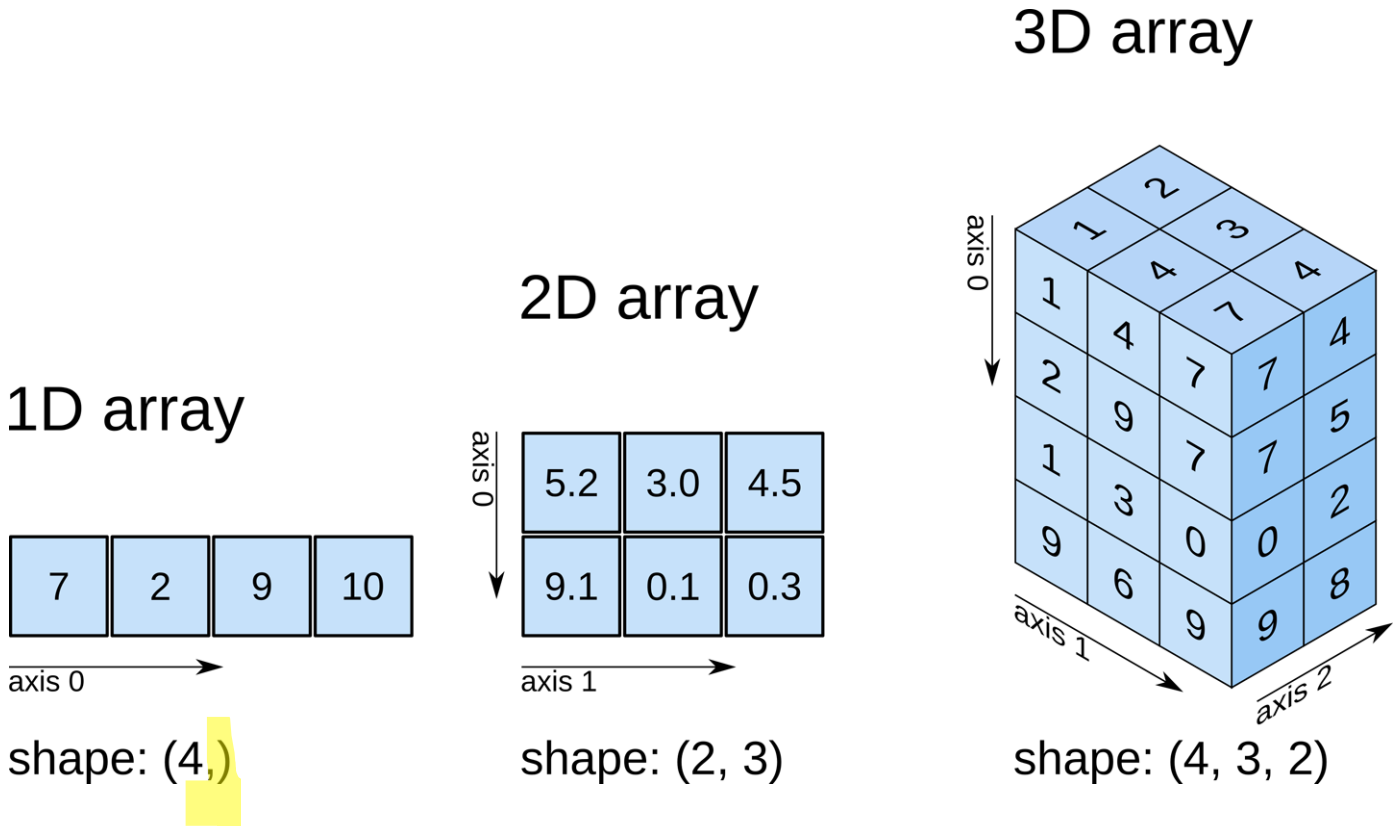
- What is a **tensor**? A container for a numerical data. Tensors are multi-dimensional arrays with a uniform type (called a dtype).

- Scalars (0D, rank-0 tensors)
- Vectors (1D, rank-1 tensors)
- Matrices (2D, rank-2 tensors)

Data_representation_NN.ipynb

NumPy Shape

- A tuple of integers giving the size of the array along each dimension.
- A dimension is also called an axis



Real-world examples of data tensors

- *Vector data* — 2D tensors of shape (samples, features)
- *Timeseries data or sequence data* — 3D tensors of shape (samples, timesteps, features)
- *Images* — 4D tensors of shape (samples, height, width, channels)
- *Video* — 5D tensors of shape (samples, frames, height, width, channels)

4D tensor For Color Images

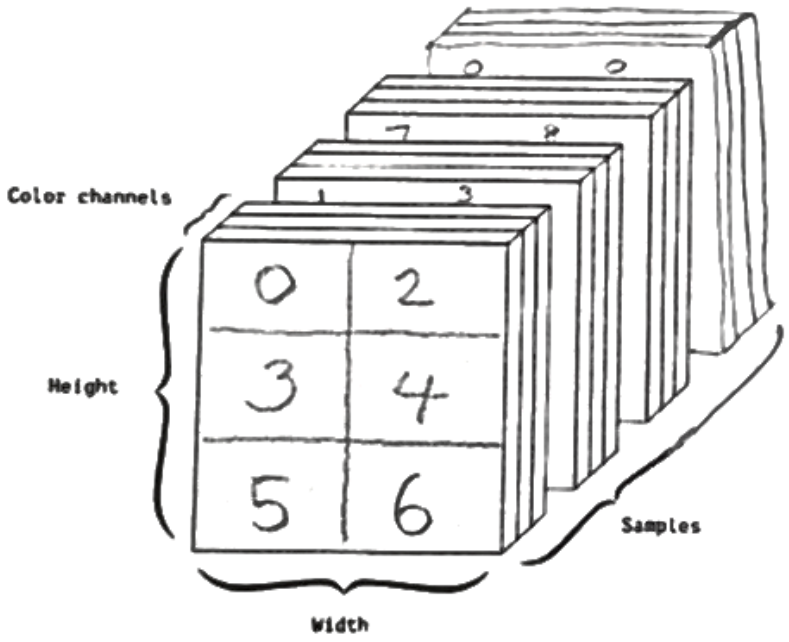
```
# sample, height, width, channel
#   4       3       2       3
cimage = np.array([
                    [ [[0,1,2],[2,1,0]],[[3,0,0],[4,0,1]],[[5,0,0],[6,0,1]] ], sample 0
                    [ [[1,2,0],[3,4,1]],[[0,0,0],[0,0,1]],[[0,0,0],[0,0,1]] ], sample 1
                    [ [[7,7,0],[8,8,1]],[[0,0,0],[0,0,1]],[[0,0,0],[0,0,1]] ],       2
                    [ [[0,0,0],[0,0,1]],[[0,0,0],[0,0,1]],[[0,0,0],[0,0,1]] ],       3
                ])
print(cimage.shape)
```

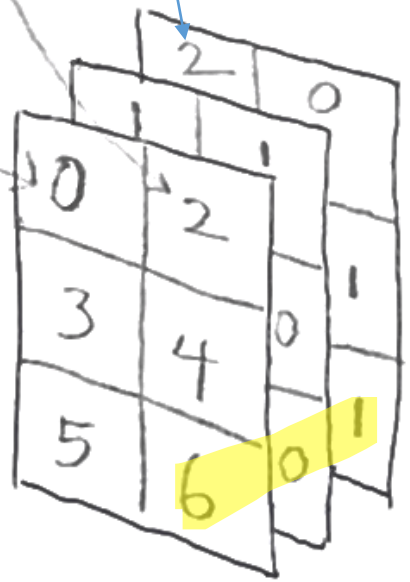
col 0 col 1

Col 0 of channel 2

Row (Height) 0 Row 1 Row 2

```
(4, 3, 2, 3)
```



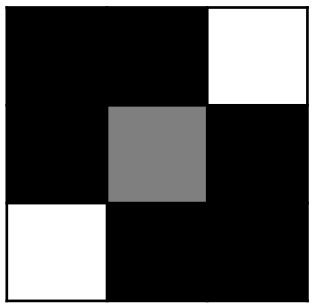


[b,g,r]

4D tensor For Gray Images

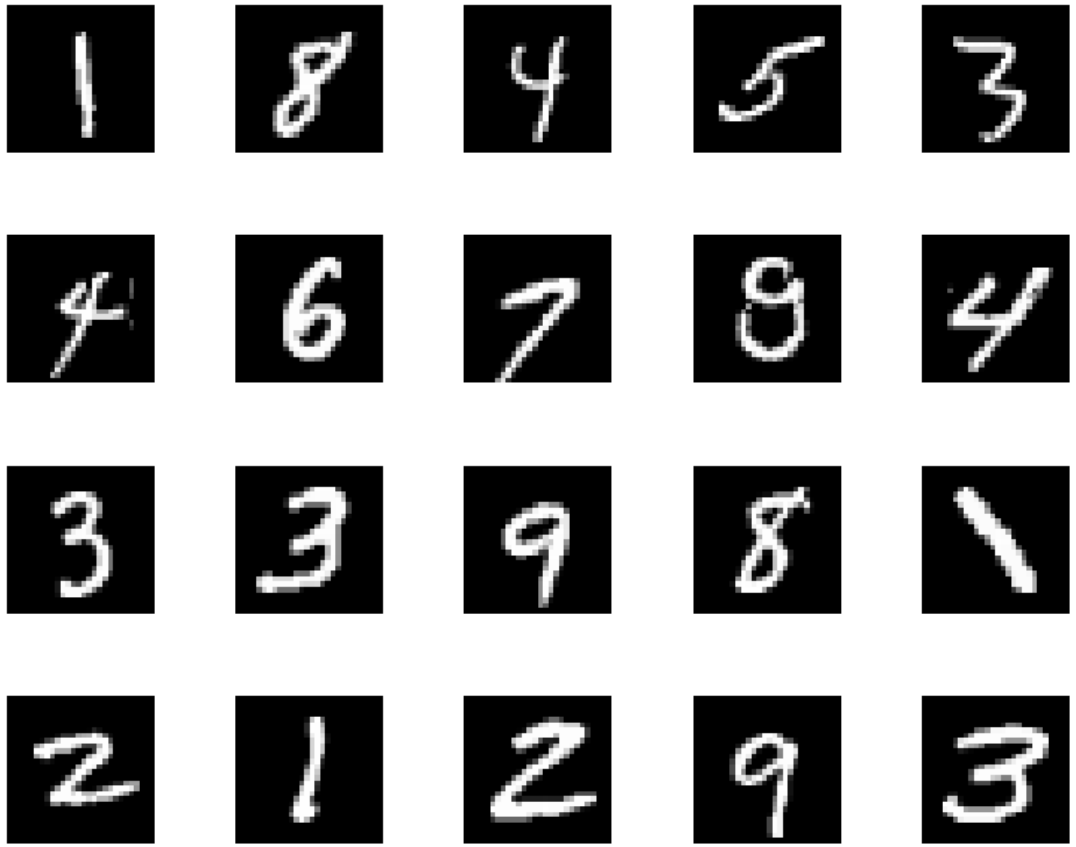
Assume each cell has 0 – 9 values.
0 is black, 9 is white, and 5 is gray.

```
g_image = np.array([# Slash /
                        [[[0],[0],[9]],
                         [[0],[5],[0]],
                         [[9],[0],[0]]],

                        [[[0],[0],[8]],
                         [[0],[8],[0]],
                         [[8],[0],[0]]],

                    # Backslash \
                        [[[9],[0],[0]],
                         [[0],[9],[0]],
                         [[0],[0],[9]]],

                        [[[8],[0],[0]],
                         [[0],[8],[0]],
                         [[0],[0],[8]]]
                    ])
```



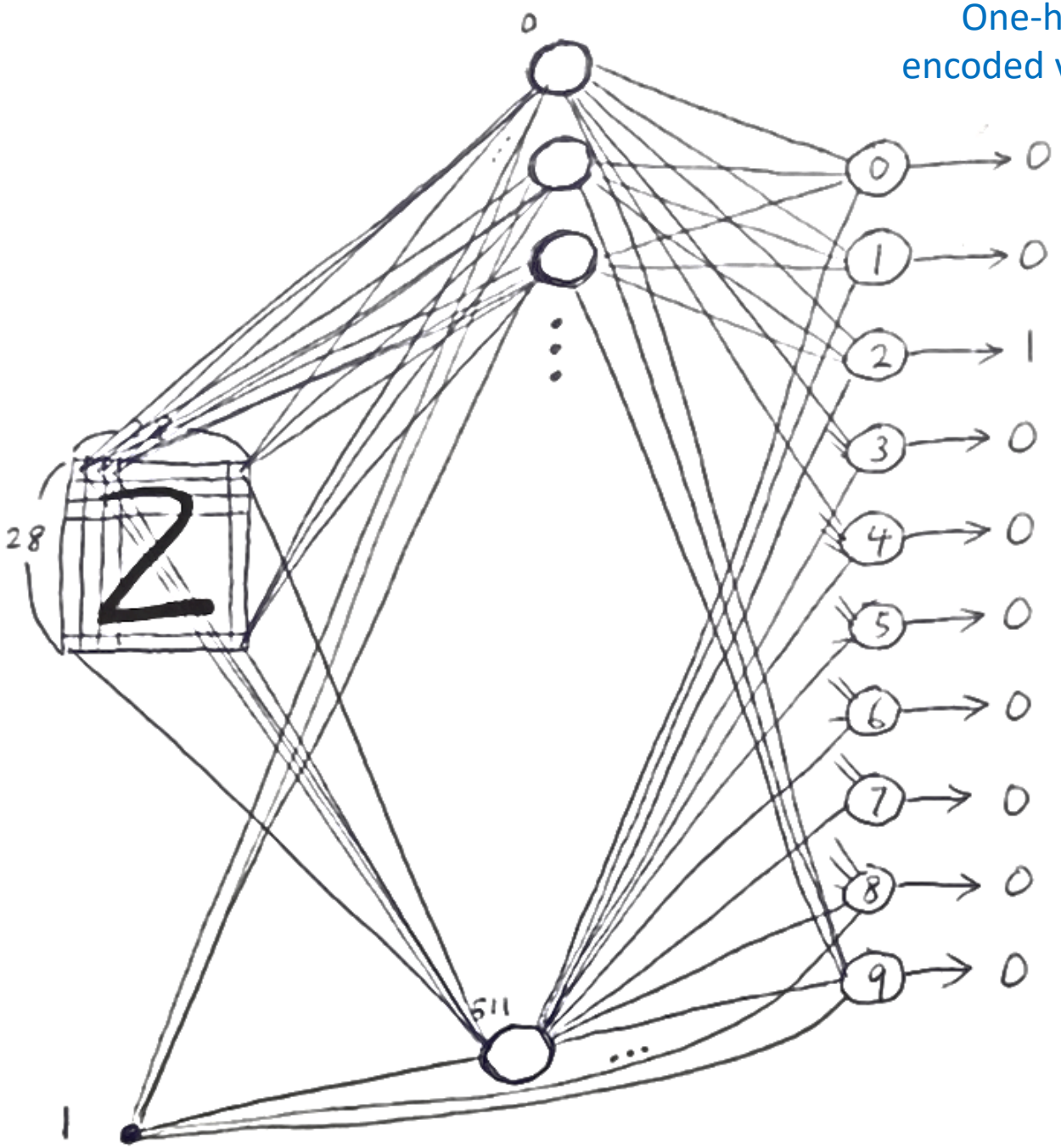
Shape = (4, 3, 3, 1)

Test MNIST.ipynb ("Hello World" of DL)



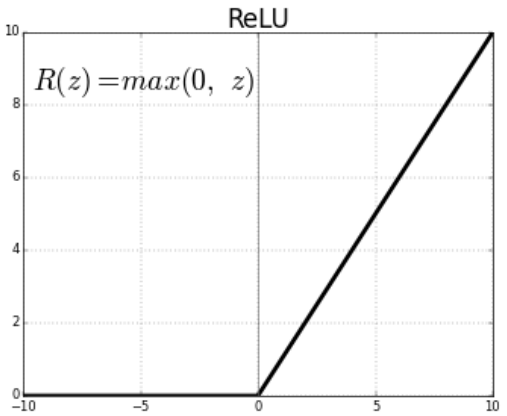
Read the book
section 2.1
pages 29-32

ML dataset by **NIST**: National Institute of Standards and Technology, 1980s)

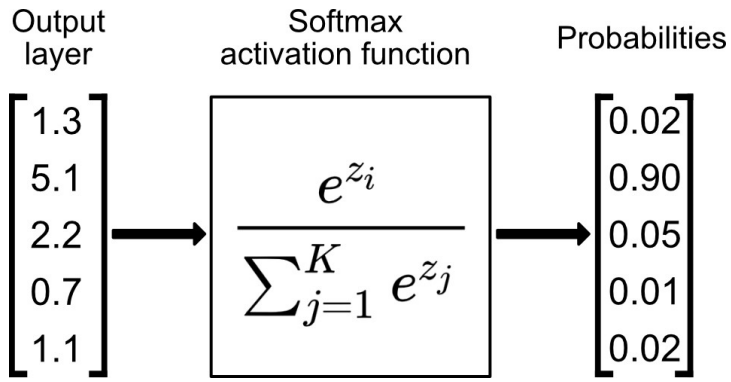


One-hot encoded vector

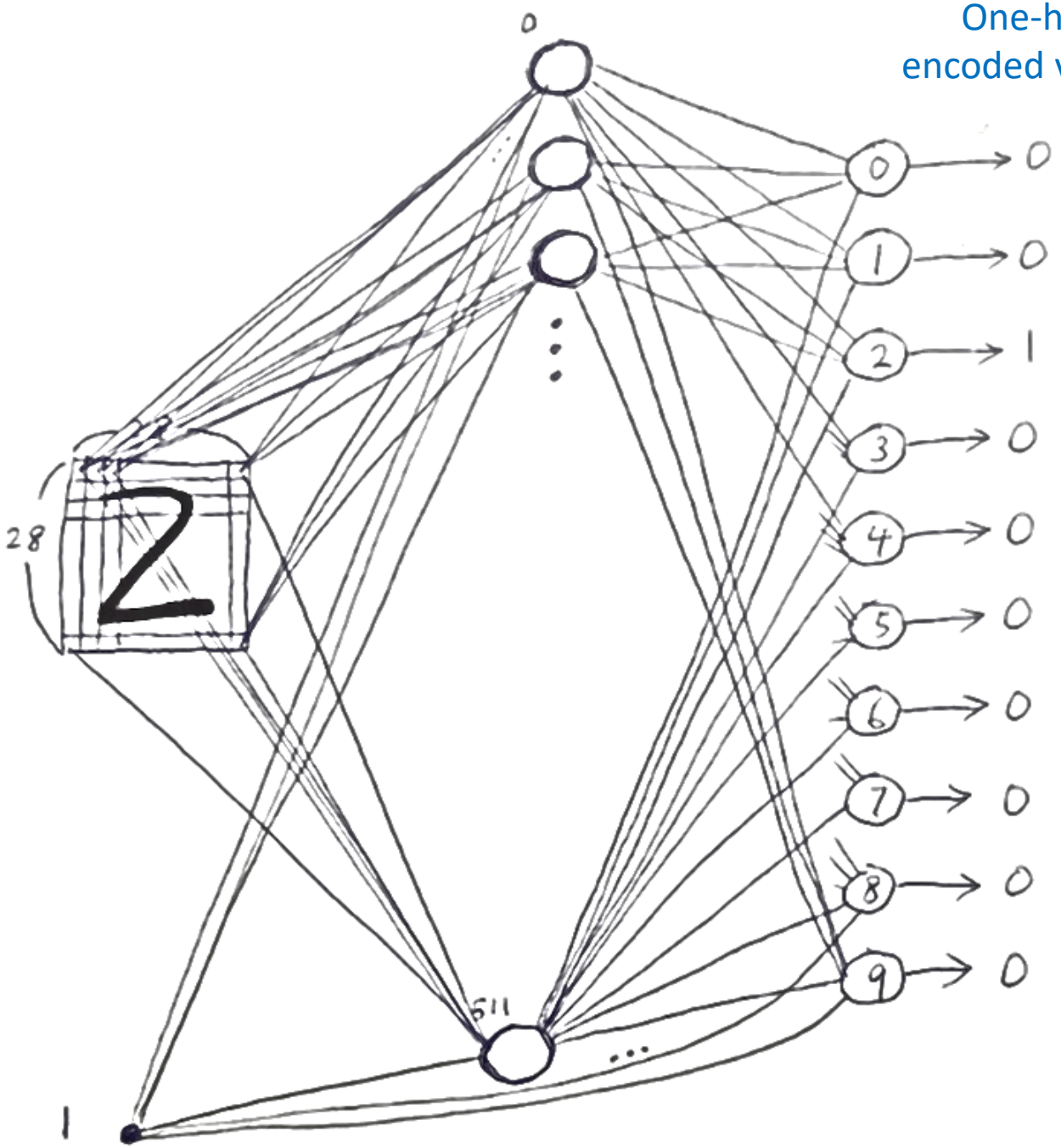
Activation functions



Relu



Softmax: each output value is between 0 and 1 and the sum of all outputs equals 1

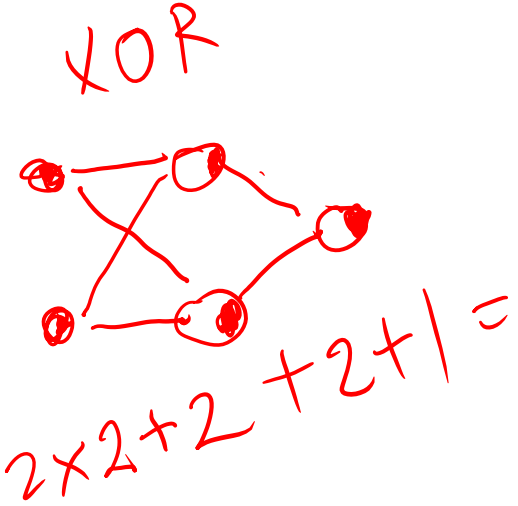


One-hot encoded vector

How many connections (weight values) to optimize?

(I+1)*H +(H+1)*O

(28x28+1)*512+(512+1)*10 = 407,050



```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy',
                metrics=['accuracy'])
train_images2 = train_images.reshape((60000, 28 * 28)) # See next slide
train_images2 = train_images2.astype('float32') / 255
test_images2 = test_images.reshape((10000, 28 * 28))
test_images2 = test_images2.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
history = network.fit(train_images2, train_labels, epochs=20, batch_size=128,
                      verbose=0)
test_loss, test_acc = network.evaluate(test_images2, test_labels)
print('test_accuracy:', test_acc)
```

```
10000/10000 [==============================] - 0s 44us/step
test_accuracy: 0.9834
```

MNIST_old.ipynb on Canvas

Reshape

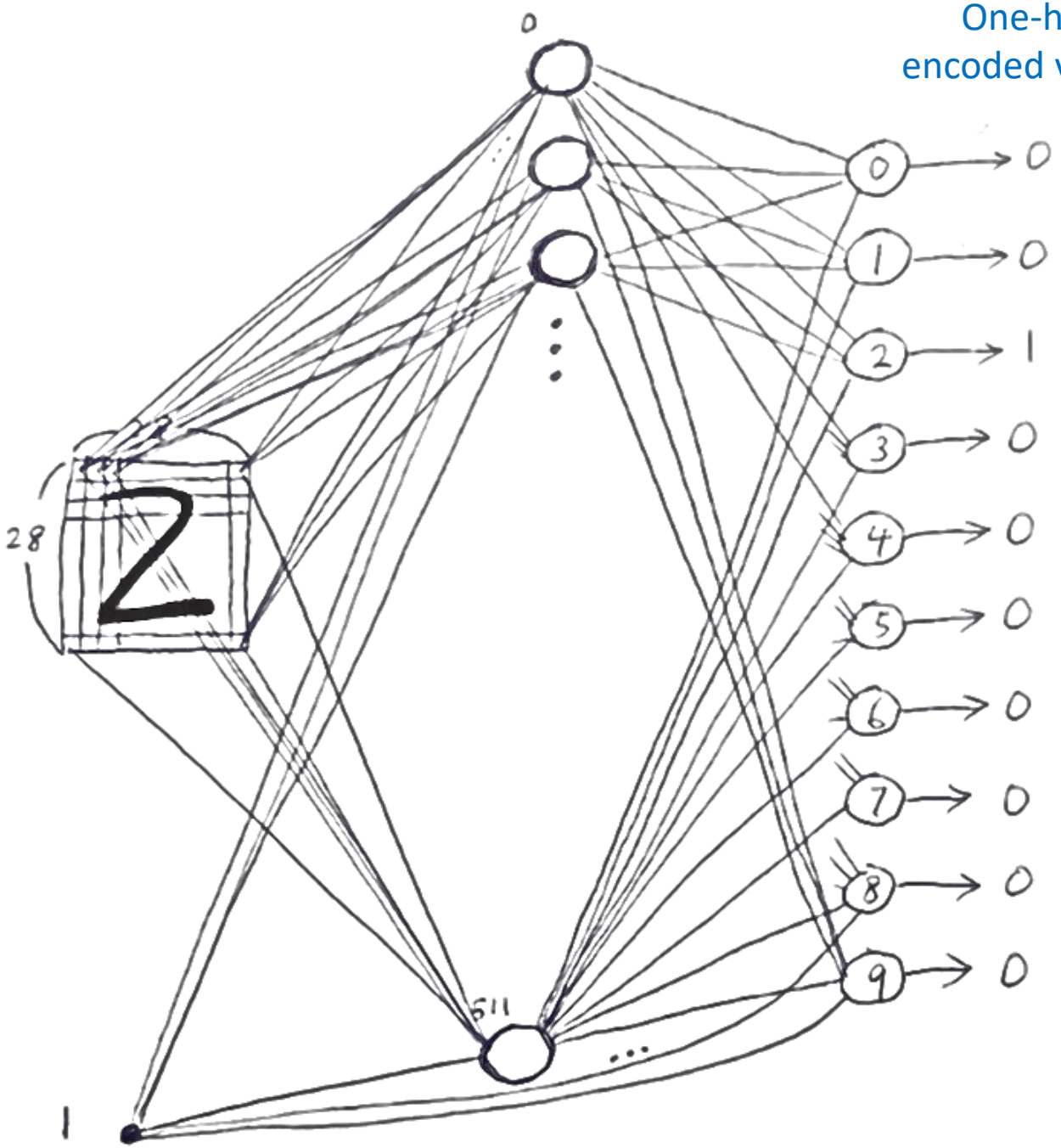
To convert an image into one vector

```
import numpy as np
# Suppose dataset ds has 4 samples of 3x2 (HeightxWidth) gray images
ds = np.array([ [[0,0],[0,0],[0,0]], # sample 0
                [[1,2],[3,4],[5,6]], # sample 1
                [[7,7],[8,8],[9,9]], # sample 2
                [[4,0],[0,1],[0,0]], # sample 3
              ])
print(ds.shape)
```

```
(4, 3, 2)
```

```
ds1 = ds.reshape(4, 3*2)
print(ds1)
print(ds1.shape)
print(ds1[0].shape)
```

```
[[0 0 0 0 0 0]
 [1 2 3 4 5 6]
 [7 7 8 8 9 9]
 [4 0 0 1 0 0]]
(4, 6)
(6,)
```



Reviews

- What's the activation function of hidden neurons?
- What's the activation function of output neurons?
- What optimizer was used?
- What loss function was used to calculate output layer errors?
- What does `to_categorical()` method do?
- What are the outputs of `model.evaluate()` method?

Review

- *An optimizer* — The mechanism through which the model will update itself based on the training data it sees, so as to improve its performance.
- *A loss function* — How the model will be able to measure its performance (errors) on the training data, and thus how it will be able to steer itself in the right direction. What did we use for MNIST?
  - sparse_categorical_crossentropy when your target labels are integers. This loss function is mathematically same as the categorical_crossentropy. It just has a different interface
  - categorical_crossentropy when your target labels are one-hot encoded vectors
- *A metric* — a function that is used to judge the performance of your model. Metric functions are similar to loss functions, except that the results from evaluating a metric are not used when training the model. Note that you may use any loss function as a metric.

```
model.compile(
  optimizer='rmsprop',
  loss='sparse_categorical_crossentropy', # target labels are integers
  metrics=['accuracy']
)
model.fit(X, y, batch_size=1, epochs=1300, verbose=1)
```

```
Epoch 1/1300
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.0667 - loss: 1.1290
Epoch 2/1300
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.2472 - loss: 1.0494
Epoch 3/1300
```

verbose=1

Without metrics

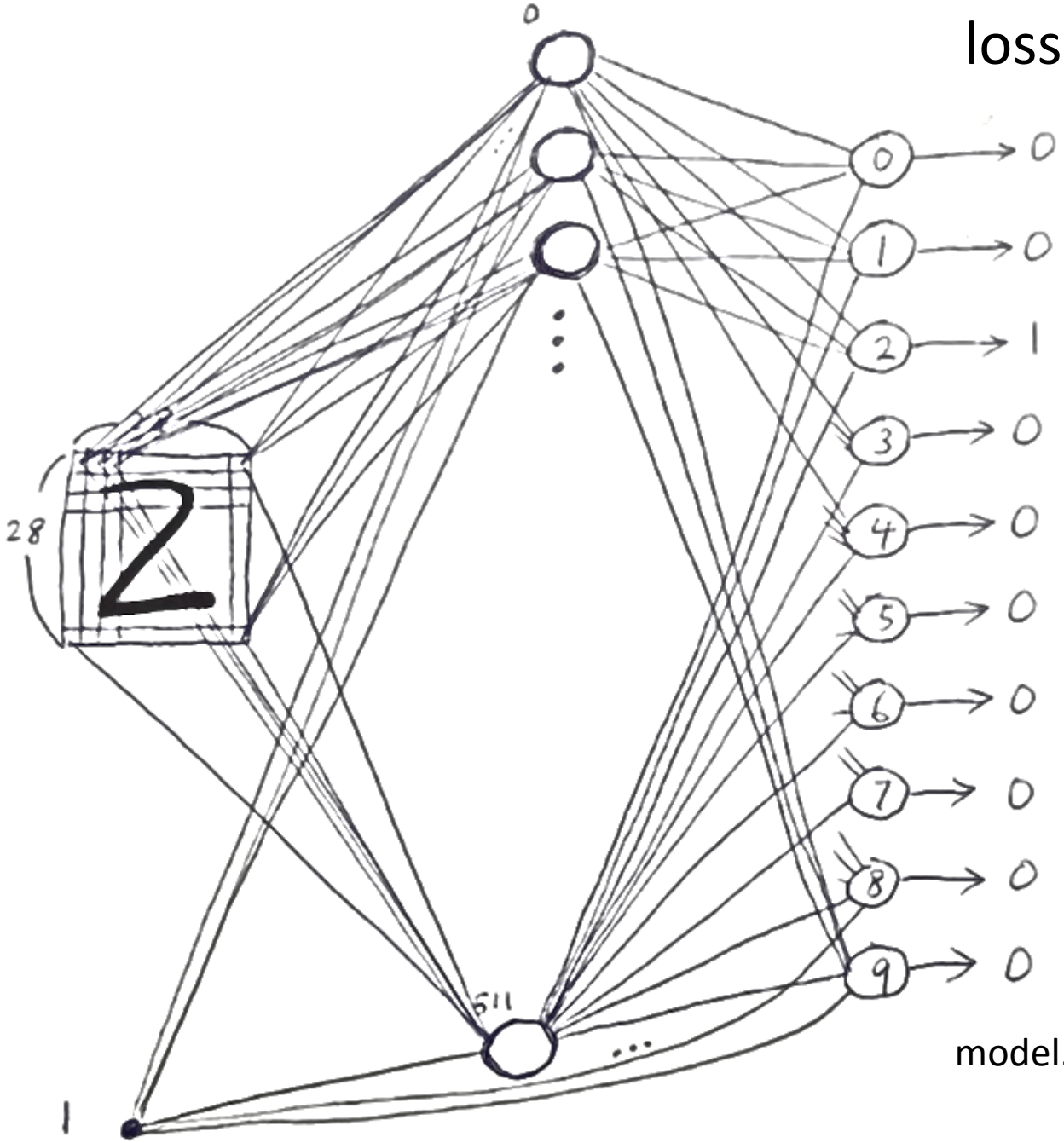
```
model.compile(
  optimizer='rmsprop',
  loss='sparse_categorical_crossentropy', # target labels are integers
  #metrics=['accuracy']
)
model.fit(X, y, batch_size=1, epochs=1300, verbose=1)
```

```
Epoch 1/1300
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 1.1418
Epoch 2/1300
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 1.1166
Epoch 3/1300
```

verbose=0: suppresses all output related to the training, evaluation, or prediction process.

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

model = Sequential([
    Dense(512, activation='relu', input_shape=(28*28,)),
    Dense(10, activation='softmax')
])
model.compile(
    optimizer='rmsprop',
    loss='sparse_categorical_crossentropy', # target labels are integers
    metrics=['accuracy’]
)

train_images = train_images.reshape((60000, 28 * 28)) # (60000, 28, 28)
train_images = train_images.astype('float32') / 255
test_images2 = test_images.reshape((10000, 28 * 28))
test_images2 = test_images2.astype('float32') / 255

model.fit(train_images, train_labels, epochs=20, batch_size=128, verbose=1)
test_loss, test_acc = model.evaluate(test_images2, test_labels) # unseen
```

MNIST.ipynb
on Canvas



loss ='categorical_crossentropy'

```
train_labels = to_categorical(train_labels)
```

model.fit(train_images, train_labels, epochs=20, batch_size=128)

```
[30] print(train_labels)
```

0 1 2

```
[5 0 4 ... 5 6 8]
```

```
train_labels_1hot = keras.utils.to_categorical(train_labels)
print(train_labels_1hot)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [1. 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.]]
```

5

0

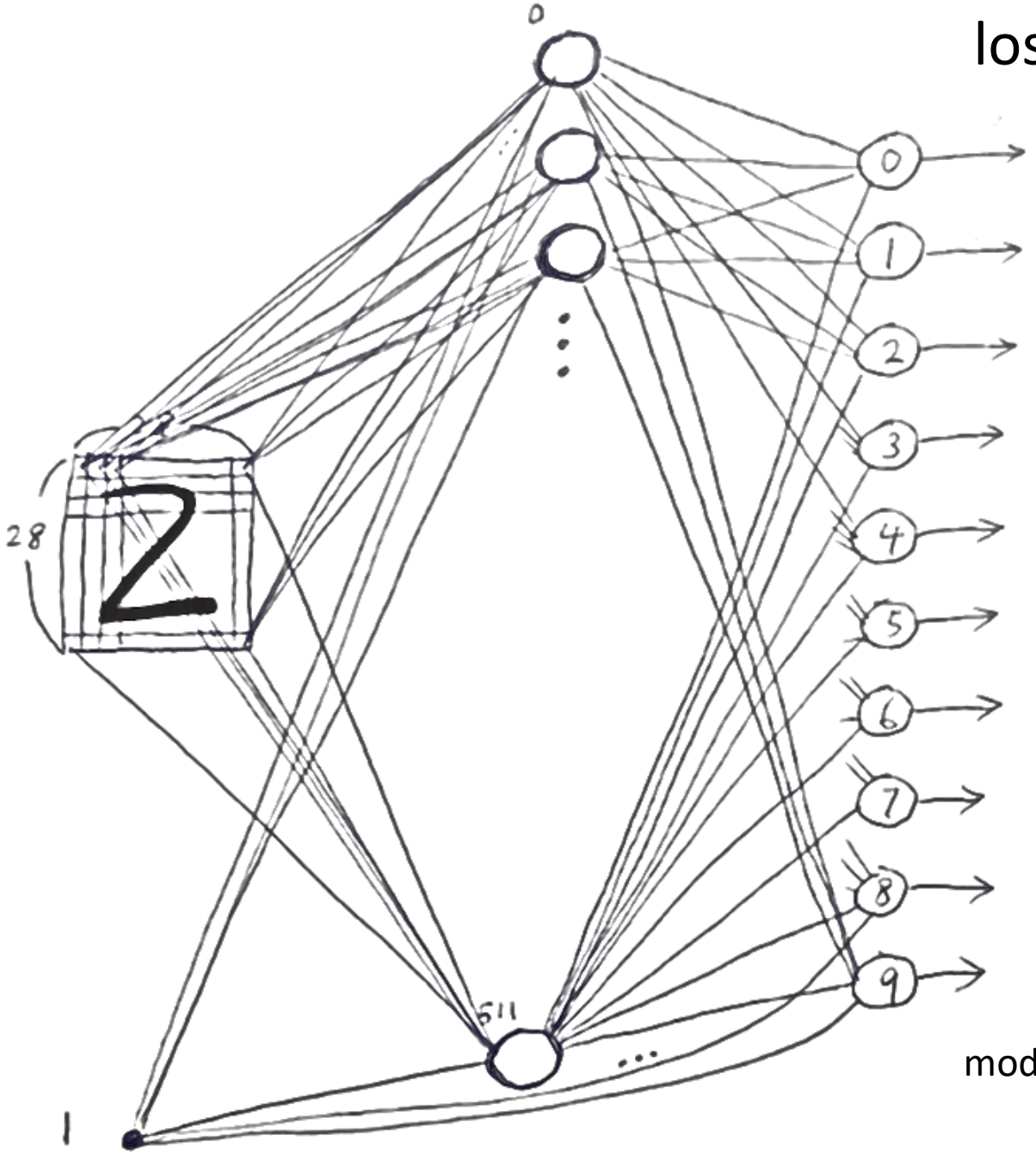
8

```
[33] print(train_labels_1hot[0])
     print(train_labels_1hot[2])
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

5

4



loss ='sparse_categorical_crossentropy'

No need to call to_categorical()

2

model.fit(train_images, train_labels, epochs=20, batch_size=128)

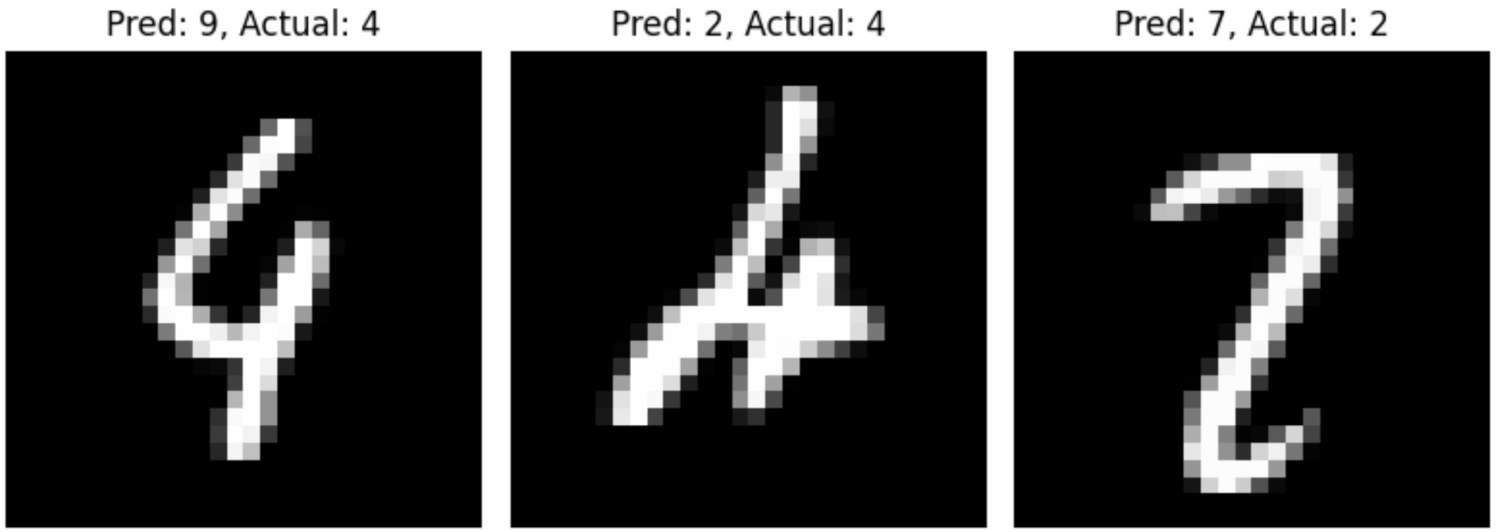
```
test_loss, test_acc = model.evaluate(test_images2, test_labels) # unseen images
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9812 - loss: 0.0780
```

```
print('test_loss:    ', test_loss)
print('test_accuracy:', test_acc)
```

```
test_loss:     0.06610522419214249
test_accuracy: 0.9837999939918518
```

3 misclassified sample images



Chollet Reading

- https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf (Free PDF book)

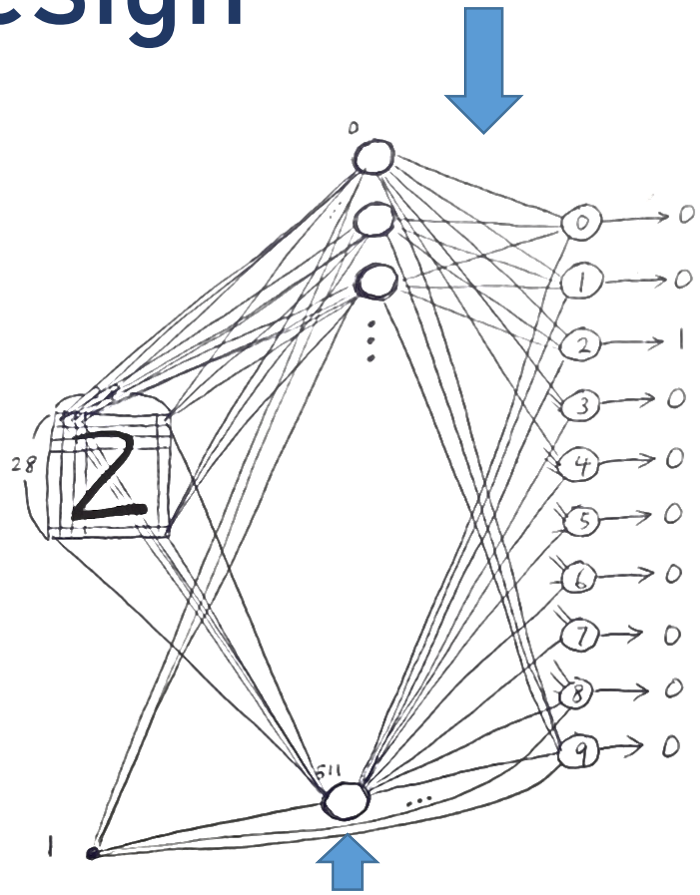
- Chapter 2

- The **MNIST.ipynb** code is very important. You must understand every line of the code.

Dense NN (DNN) Design

- Feed forward
- Fully connected (Dense)

Can we have another hidden layer?



Why 512?

Yes.
Designing NN architecture is an art, not science yet. We use heuristics.

Designing (optimizing) NN architecture by using evolutionary computation is the main topic in this class.

Feed forward, fully connected dense neural net

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

model = Sequential([
    Dense(512, activation='relu', input_shape=(28*28,)),
    Dense(10, activation='softmax')
])
model.compile(
    optimizer='rmsprop',
    loss='sparse_categorical_crossentropy', # target labels are integers
    metrics=['accuracy’]
)

train_images = train_images.reshape((60000, 28 * 28)) # (60000, 28, 28)
train_images = train_images.astype('float32') / 255
test_images2 = test_images.reshape((10000, 28 * 28))
test_images2 = test_images2.astype('float32') / 255

model.fit(train_images, train_labels, epochs=20, batch_size=128, verbose=1)
test_loss, test_acc = model.evaluate(test_images2, test_labels) # unseen
```

batch_size?
Number of samples per gradient update.