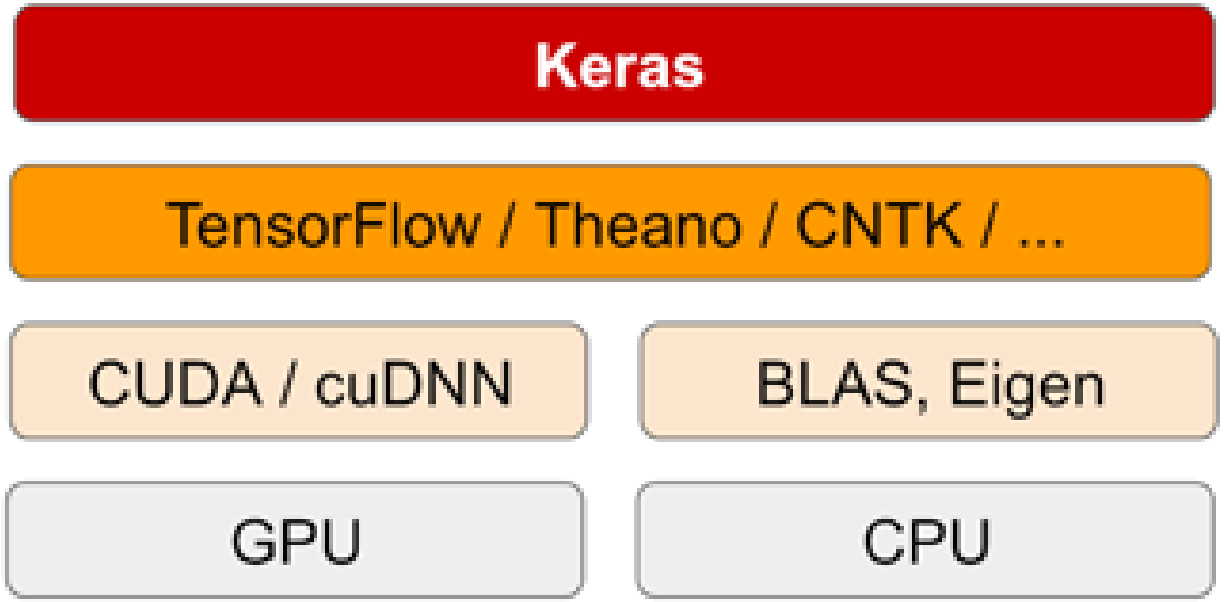
Introduction to Keras

Prof. Kuan-Ting Lai

2019/2/19

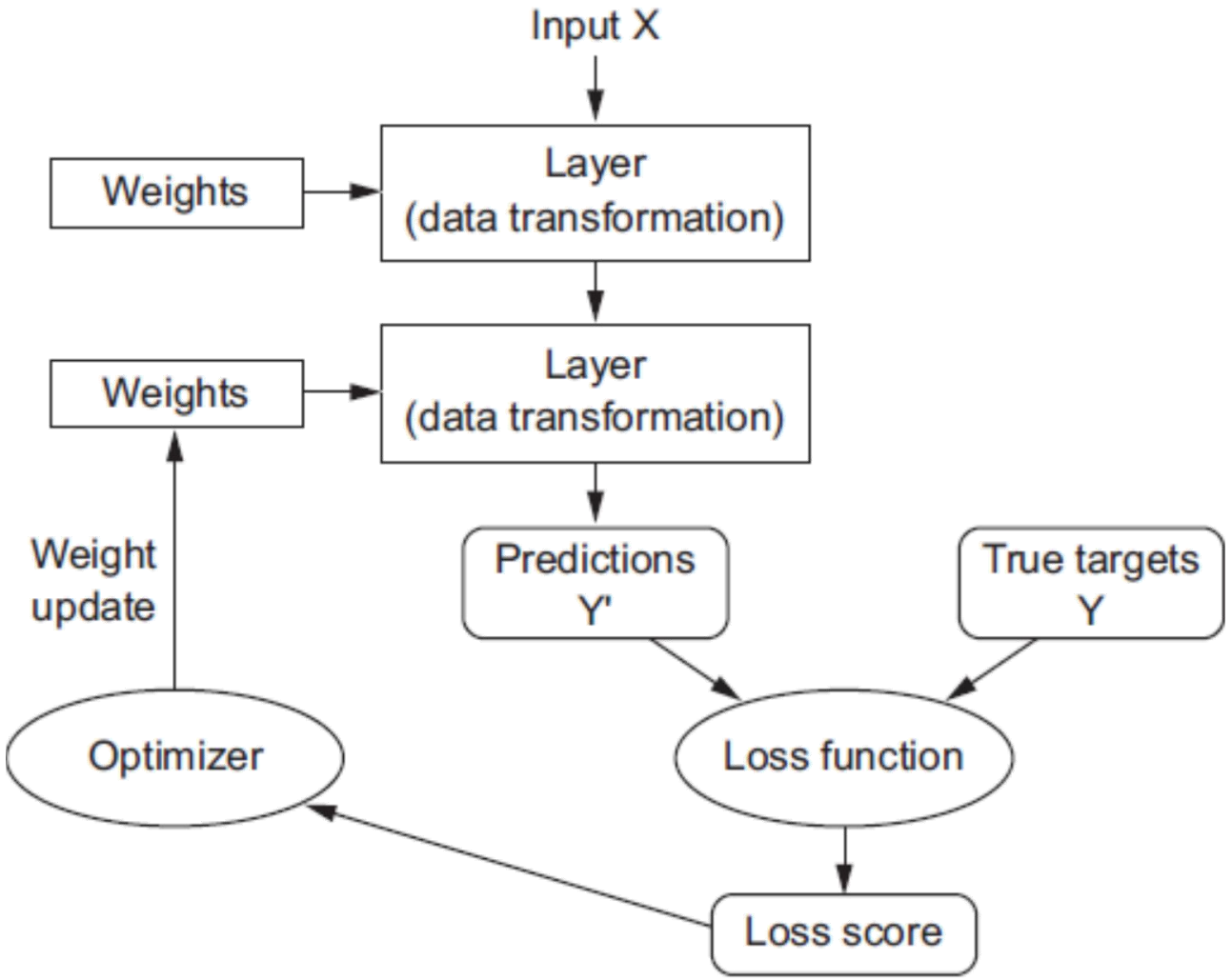
Keras (keras.io)

- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano
- Developed by Francois Chollet
- Officially supported by TensorFlow now



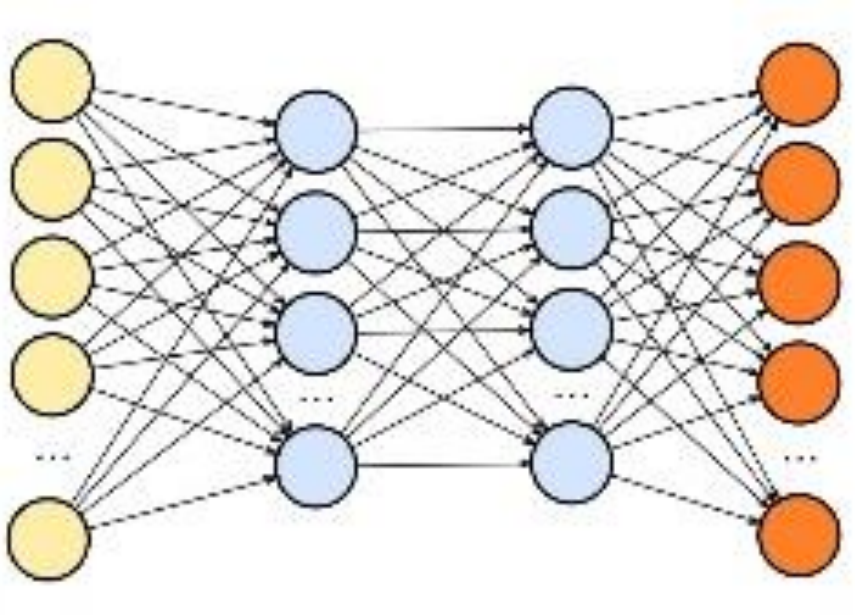
Terminologies of a Neural Network

- Layers
- Loss function
- Optimizer



Build Your Own Network with Keras

- Learning deep learning with Keras is like playing LEGO



=

First Look a Neural Network

- Classify grayscale images of handwritten digits (28 × 28 pixels) into their 10 categories (0 ~ 9)
- Use the MNIST dataset created by Yann LeCun
- MNIST has 60,000 training and 10,000 test images

Loading MNIST in Keras on Colab

mnist.ipynb ☆

File Edit View Insert Runtime Tools Help

CODE TEXT CELL CELL

```
[2] from keras.datasets import mnist
```

```
Using TensorFlow backend.
```

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

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```

```
[4] train_images.shape
```

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

```
[5] len(train_labels)
```

```
60000
```

```
[6] train_labels
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
[7] test_images.shape
```

```
(10000, 28, 28)
```

```
[8] len(test_labels)
```

```
10000
```

```
test_labels
```

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

The Network Architecture

- layer: a layer in the deep network for processing data, like a filter
- Dense layer: fully connected neural layer
- Softmax layer: Output probabilities of 10 digits (0 ~ 9)

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Compile Your Model

- Loss function: measure performance on training data
- Optimizer: the mechanism for updating parameters
- Metrics to evaluate the performance on test data (accuracy)

```
network.compile(optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy'])
```

Preparing the data & Labels

- Preparing the data

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

- Preparing the labels

```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Training

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:306
Instructions for updating:
Use tf.cast instead.
Epoch 1/5
60000/60000 [==============================] - 6s 102us/step - loss: 0.2578 - acc: 0.9249
Epoch 2/5
60000/60000 [==============================] - 6s 94us/step - loss: 0.1026 - acc: 0.9694
Epoch 3/5
60000/60000 [==============================] - 6s 94us/step - loss: 0.0671 - acc: 0.9802
Epoch 4/5
60000/60000 [==============================] - 6s 93us/step - loss: 0.0499 - acc: 0.9850
Epoch 5/5
60000/60000 [==============================] - 6s 93us/step - loss: 0.0376 - acc: 0.9889
<keras.callbacks.History at 0x7f1e5b9c5d68>
```

Complete Code

```
[10] from keras import models
     from keras import layers
     network = models.Sequential()
     network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
     network.add(layers.Dense(10, activation='softmax'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in
Instructions for updating:
Colocations handled automatically by placer.
```

```
[14] network.compile(optimizer='rmsprop',
     loss='categorical_crossentropy',
     metrics=['accuracy'])
```

```
[11] train_images = train_images.reshape((60000, 28 * 28))
     train_images = train_images.astype('float32') / 255
     test_images = test_images.reshape((10000, 28 * 28))
     test_images = test_images.astype('float32') / 255
```

```
[12] from keras.utils import to_categorical
     train_labels = to_categorical(train_labels)
     test_labels = to_categorical(test_labels)
```

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/5
60000/60000 [==============================] - 6s 102us/step - loss: 0.2578 - acc: 0.9249
Epoch 2/5
60000/60000 [==============================] - 6s 94us/step - loss: 0.1026 - acc: 0.9694
Epoch 3/5
60000/60000 [==============================] - 6s 94us/step - loss: 0.0671 - acc: 0.9802
Epoch 4/5
60000/60000 [==============================] - 6s 93us/step - loss: 0.0499 - acc: 0.9850
Epoch 5/5
60000/60000 [==============================] - 6s 93us/step - loss: 0.0376 - acc: 0.9889
<keras.callbacks.History at 0x7f1e5b9c5d68>
```

Evaluation

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
10000/10000 [==============================] - 1s 56us/step
test_acc: 0.9776
```

Deep Learning for Classification & Regression

- Choosing the right last-layer activation and loss function

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

IMDb Movie Review Datasets

- Internet Movie Database
- 50,000 polarized reviews (50% positive and 50% negative reviews)
- https://www.kaggle.com/iarunava/imdb-movie-reviews-dataset
- Goal
  - Predict if a review is positive or negative (binary classification)

Loading the IMDB dataset

- Packaged in Keras

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)
```

- Decode data back to English

```
word_index = imdb.get_word_index()
reverse_word_index = dict(
    [(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join(
    [reverse_word_index.get(i - 3, '?') for i in train_data[0]])
```

word_index is a dictionary mapping words to an integer index.

Reverses it, mapping integer indices to words

Decodes the review. Note that the indices are offset by 3 because 0, 1, and 2 are reserved indices for "padding," "start of sequence," and "unknown."

Preprocessing the Data

- Turn data into tensors
  - Pad the list to make all reviews have the same length
  - Transform integer data into one-hot encoding format

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Creates an all-zero matrix of shape (len(sequences), dimension)

Sets specific indices of results[i] to 1s

Vectorized training data

Vectorized test data

Building Your Network

- Select activation function

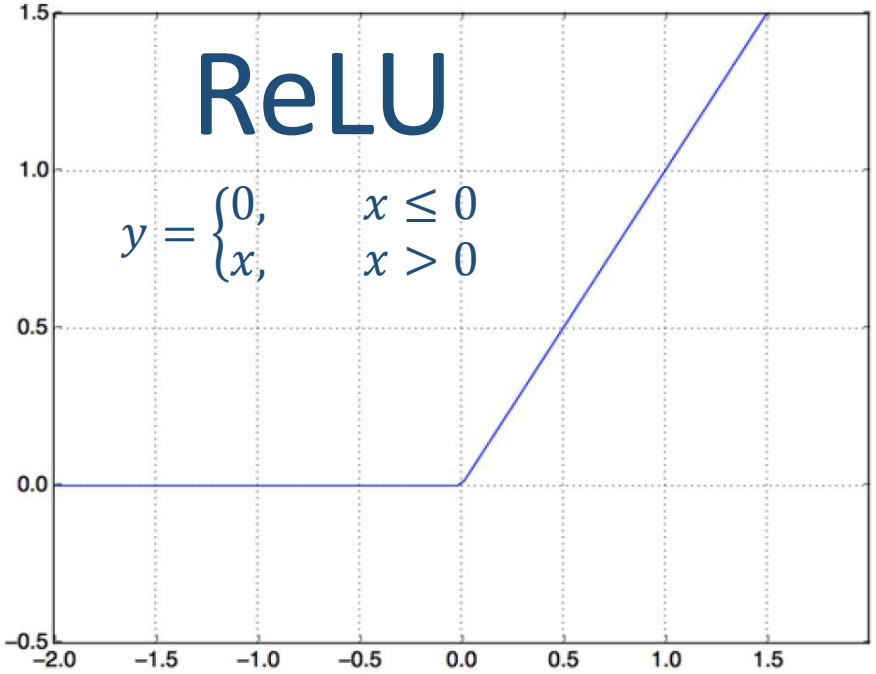


Figure 3.4 The rectified linear unit function

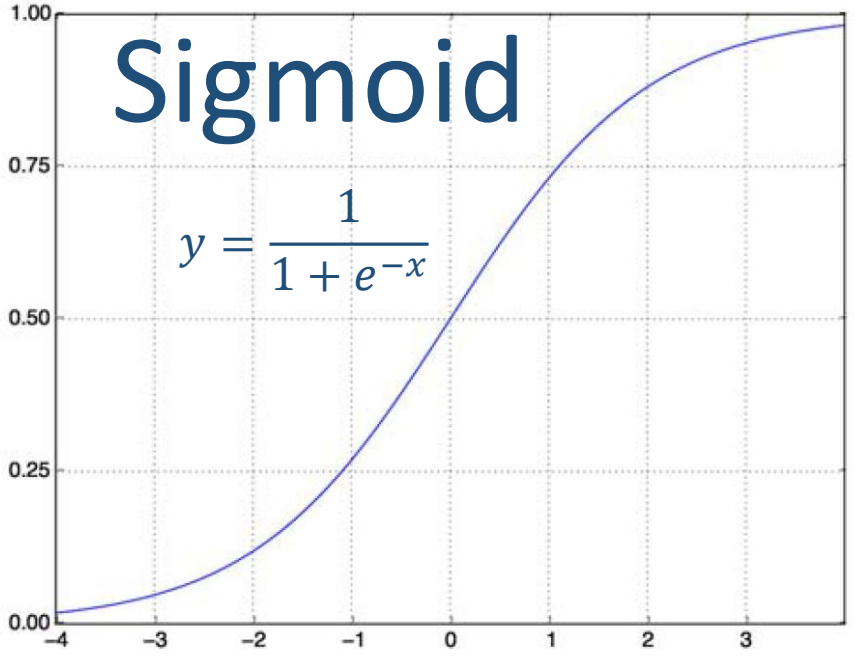


Figure 3.5 The sigmoid function

Why We Need Activation Functions?

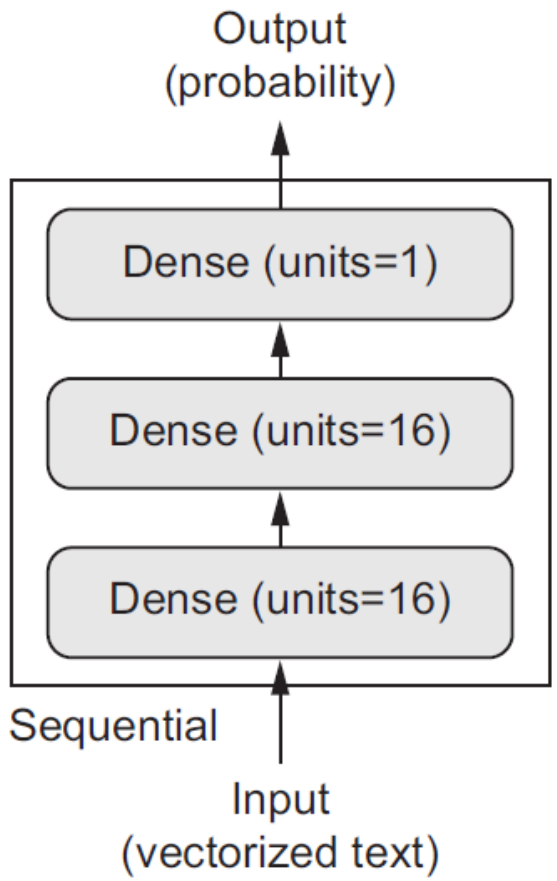
- Without an activation function, the Dense layer would consist of two linear operations—a dot product and an addition.
- So the layer could only learn *linear transformations* (affine transformations) of the input data.
- Such a hypothesis space is too restricted and wouldn't benefit from multiple layers of representations.

Create a three-layer network

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu',
input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid’))

model.compile(optimizer='rmsprop’,
              loss='binary_crossentropy’,
              metrics=['accuracy'])
```



Customizing the Optimizer & Loss & Metric

Listing 3.5 Configuring the optimizer

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Listing 3.6 Using custom losses and metrics

```
from keras import losses
from keras import metrics

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss=losses.binary_crossentropy,
              metrics=[metrics.binary_accuracy])
```

Split a Validation Set

- Use a separate data to monitor the model's accuracy during training
- Select first 10,000 data as validation data

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Training the model

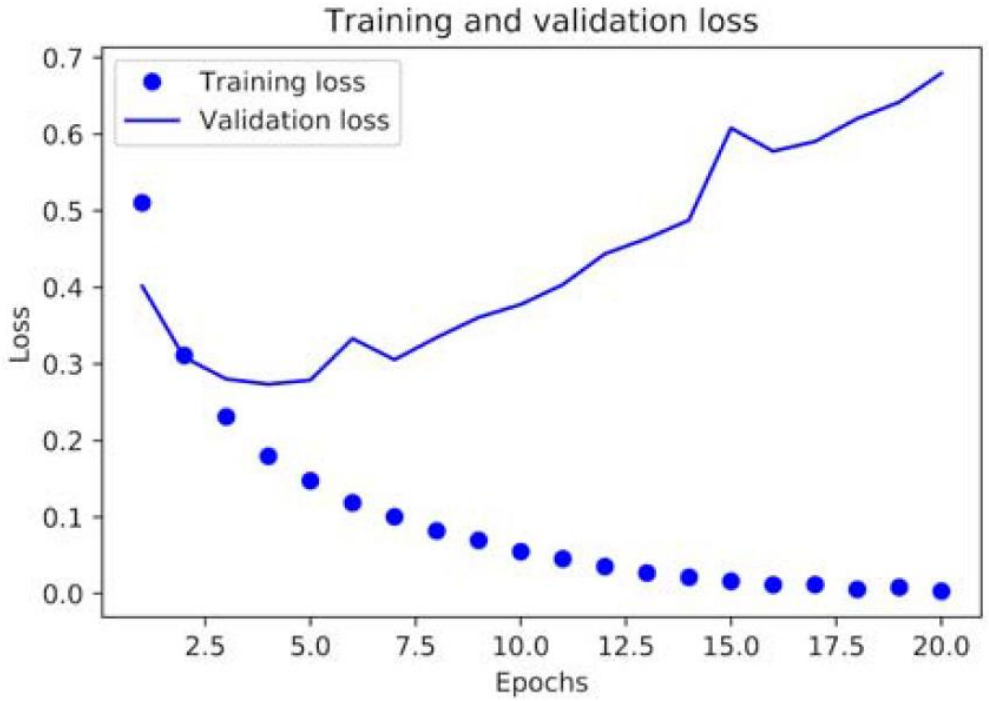
- Batch size = 512
- Epochs = 20

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

Plot the Training and Validation Loss

```
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, 'bo',
label='Training loss')
plt.plot(epochs, val_loss_values, 'b',
label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Plot the Training and Validation Accuracy

```
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
plt.plot(epochs, acc_values, 'bo',
       label='Training acc')
plt.plot(epochs, val_acc_values, 'b’,
       label='Validation acc')
plt.title('Training and validation
accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```