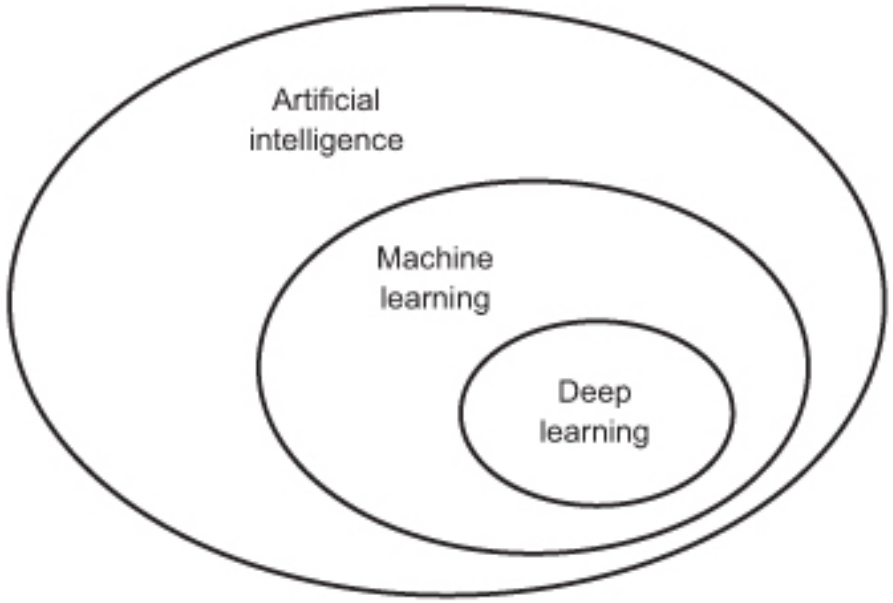
Deep Learning

Anahita Zarei, Ph.D.

Artificial intelligence, machine learning, and deep learning

Artificial intelligence

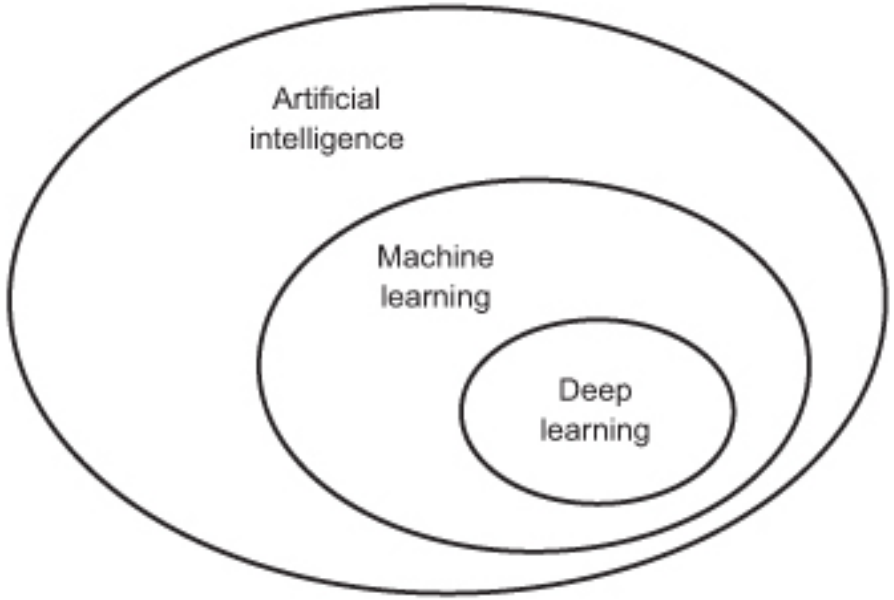
- AI is a general field that encompasses machine learning and deep learning, but that also includes many more approaches that don't involve any learning.
- It traditionally involved programmers handcrafting a sufficiently large set of explicit rules for manipulating knowledge (Symbolic AI)
- Although symbolic AI proved suitable to solve well-defined, logical problems, such as playing chess, it turned out to be intractable to figure out explicit rules for solving more complex, fuzzy problems, such as image classification, speech recognition, and language translation.

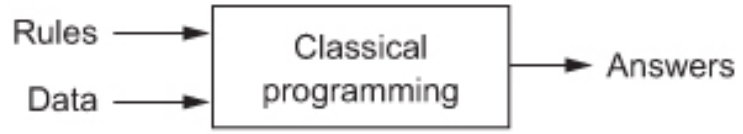


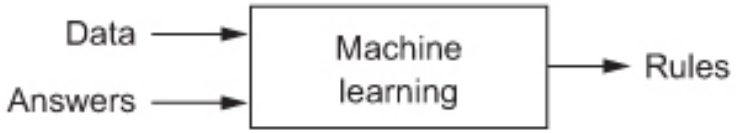
Artificial intelligence, machine learning, and deep learning

Machine Learning

- ML learns on its own how to perform a specified task
- Unlike in classical AI where humans input rules (a program) and data to be processed according to these rules, in machine learning, humans input data as well as the answers expected from the data, and out come the rules. These rules can then be applied to new data to produce original answers.
- A machine-learning system is trained rather than explicitly programmed.
- It's a hands-on discipline in which ideas are proven empirically more often than theoretically.



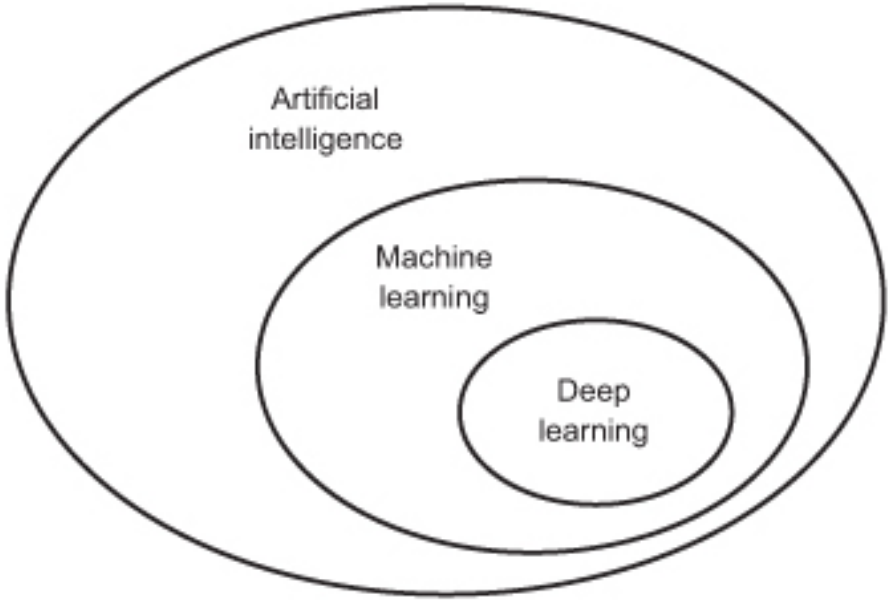




Artificial intelligence, machine learning, and deep learning

Deep Learning

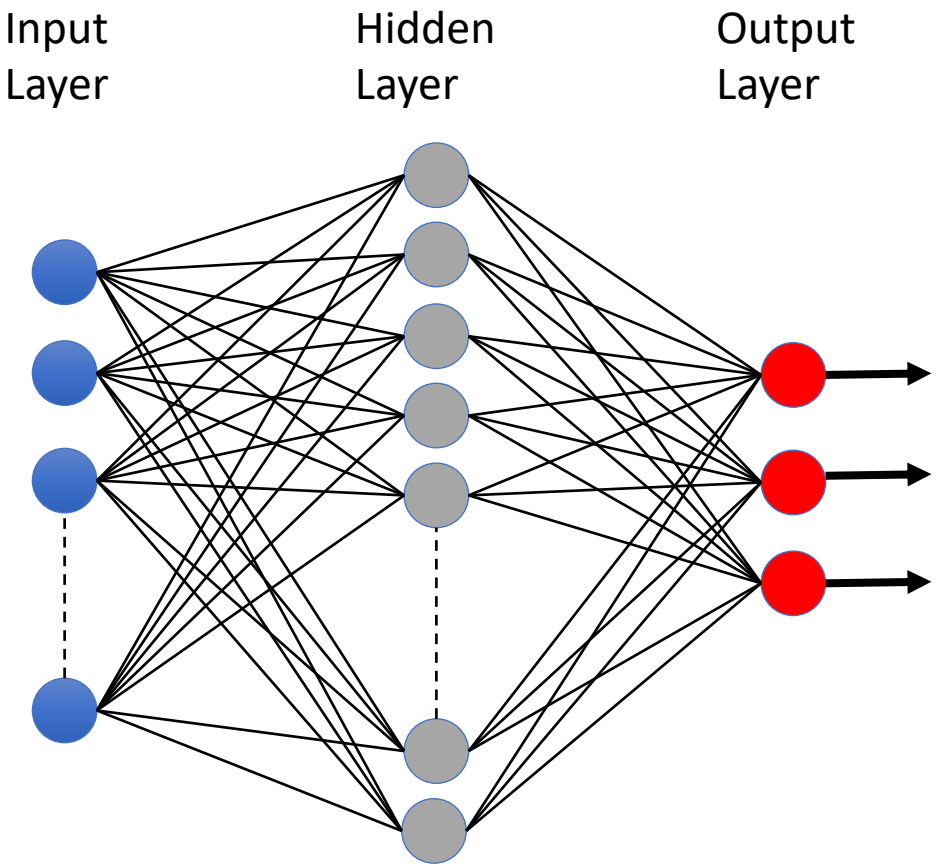
- Deep learning is a subfield of machine learning that puts an emphasis on learning successive layers of increasingly meaningful representations.
- Modern deep learning often involves tens or even hundreds of successive layers of representations and they're all learned automatically from exposure to training data.



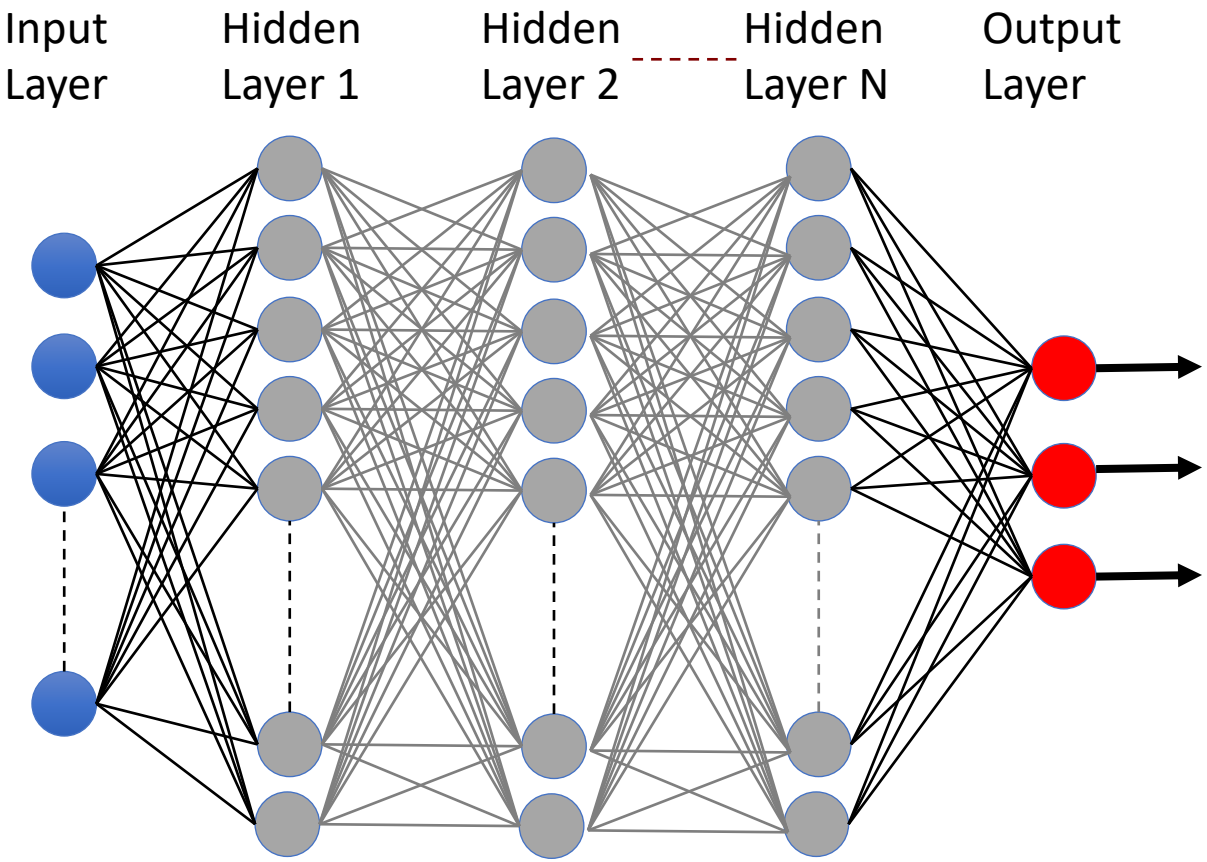
What Deep Learning Has Achieved So Far

- Neural networks and deep learning currently offer one of the most viable solutions to many complex problems including image recognition, speech recognition, natural language processing and bioinformatics.
- Microsoft, Apple, and Google all use Deep Learning-based speech recognition algorithms in their products. Convolutional neural networks outperform most other models in image recognition on benchmark data sets, and there has been substantial advancements in fine-grained sentiment analysis, question answering, and machine translation thanks to deep learning algorithms.
- Although deep learning has led to remarkable achievements in recent years, expectations for what the field will be able to achieve tend to run much higher than what will likely be possible.
- The risk with high expectations for the short term is that, as technology fails to deliver, research investment will dry up, slowing progress for a long time.

Shallow Neural Network



Deep Neural Network

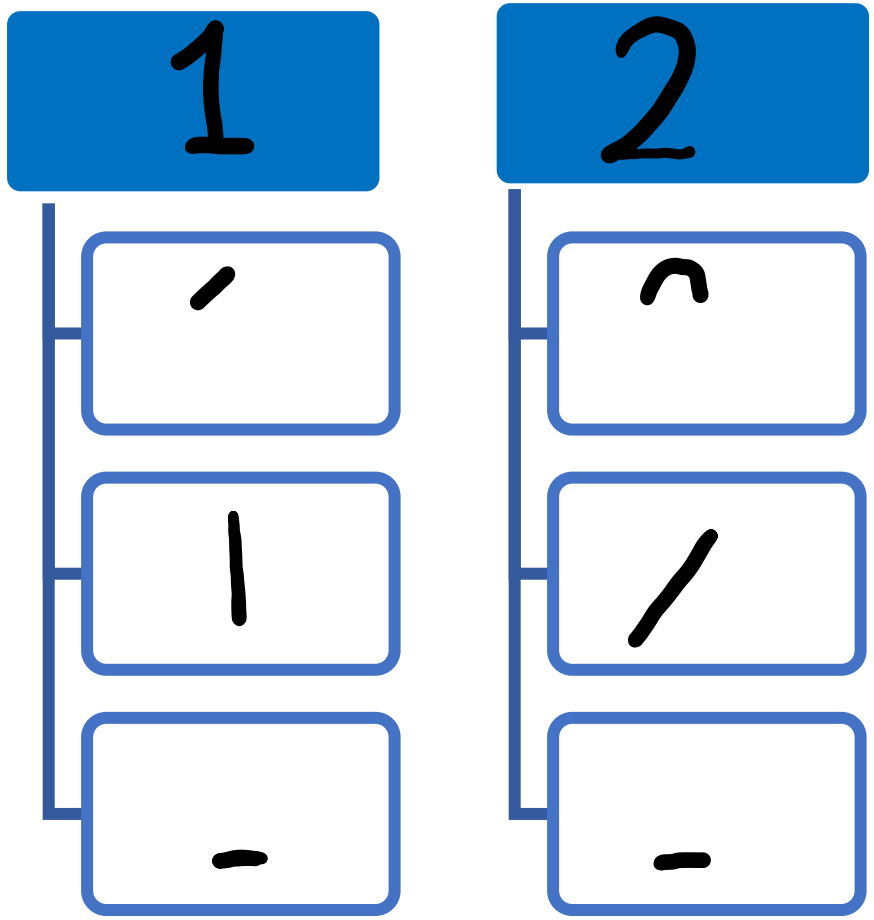


Deep Architecture: Biological Inspiration and Beyond

Human visual system shows a hierarchical learning process.

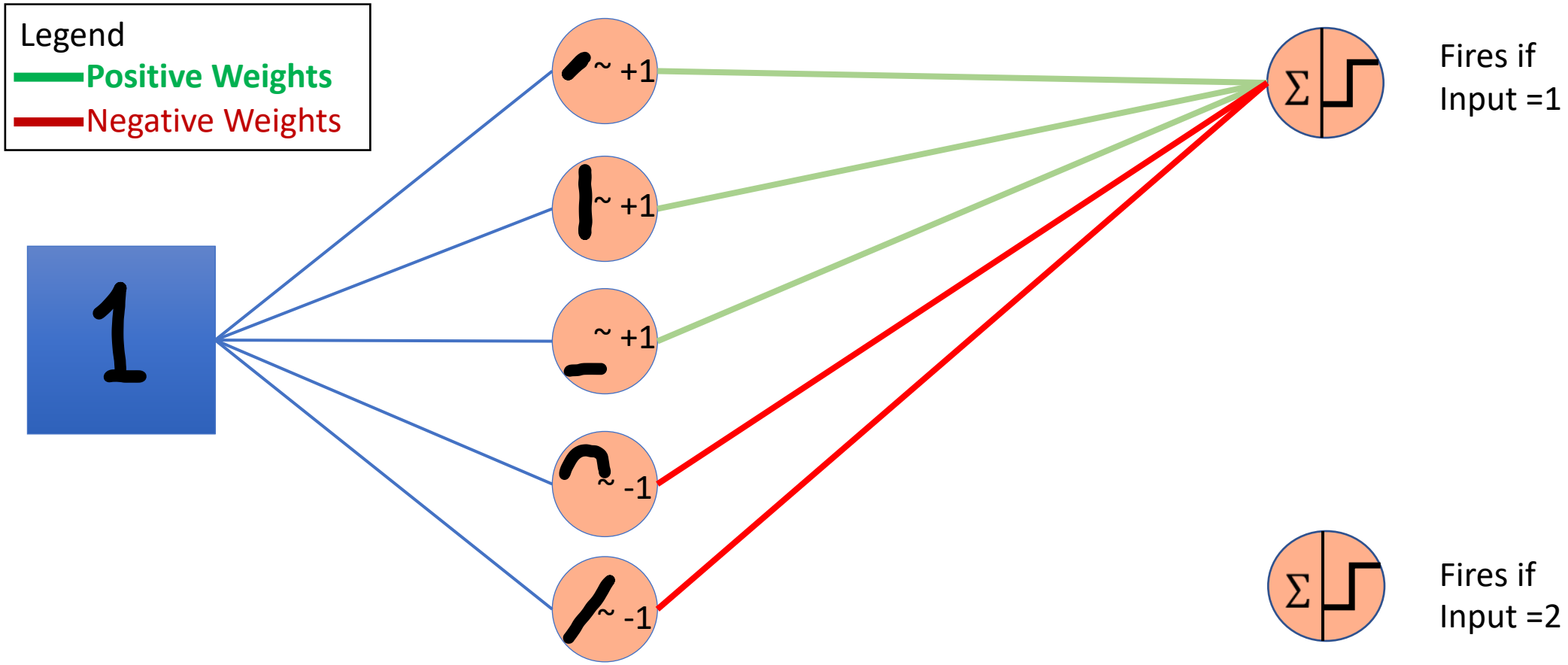
Multiple layers represent human learning more closely.

Example: Decomposing Digits into Basic Components

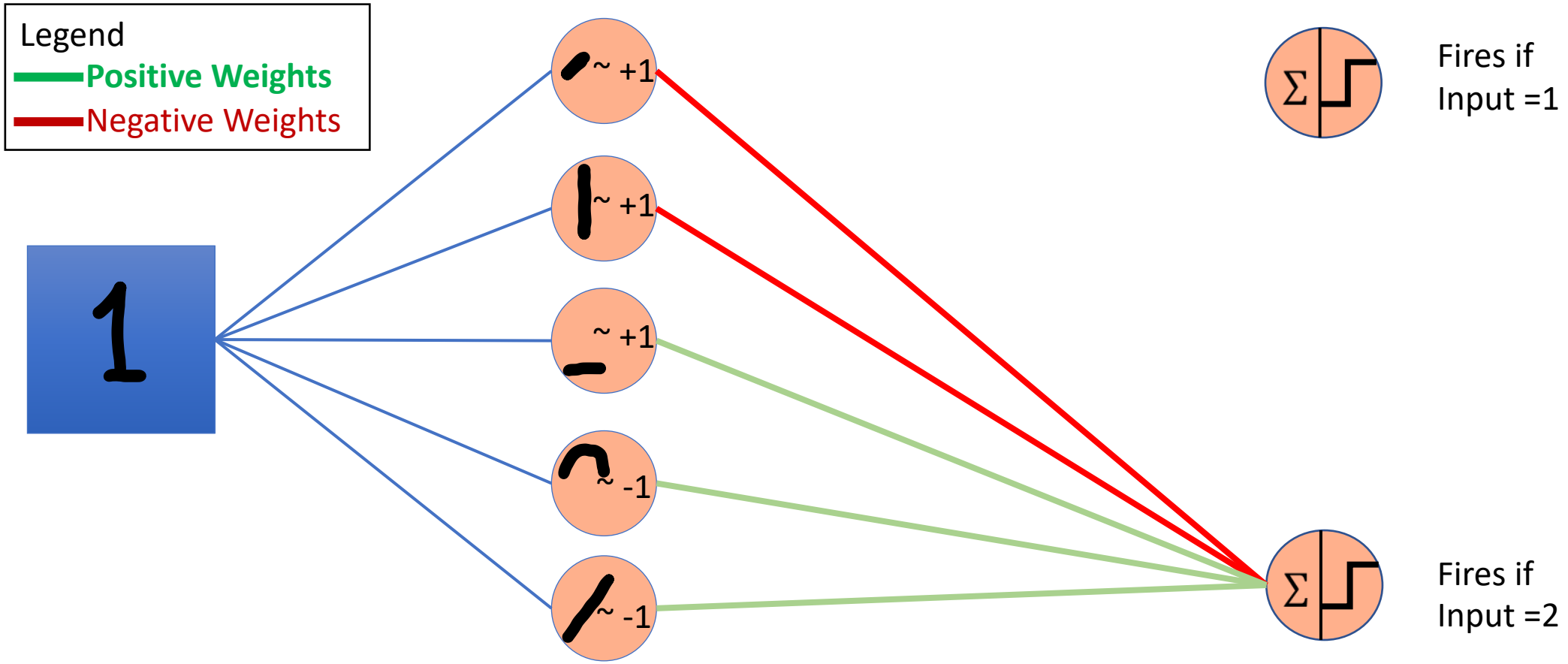


Every digit one and two can be roughly broken down to these basic shapes.

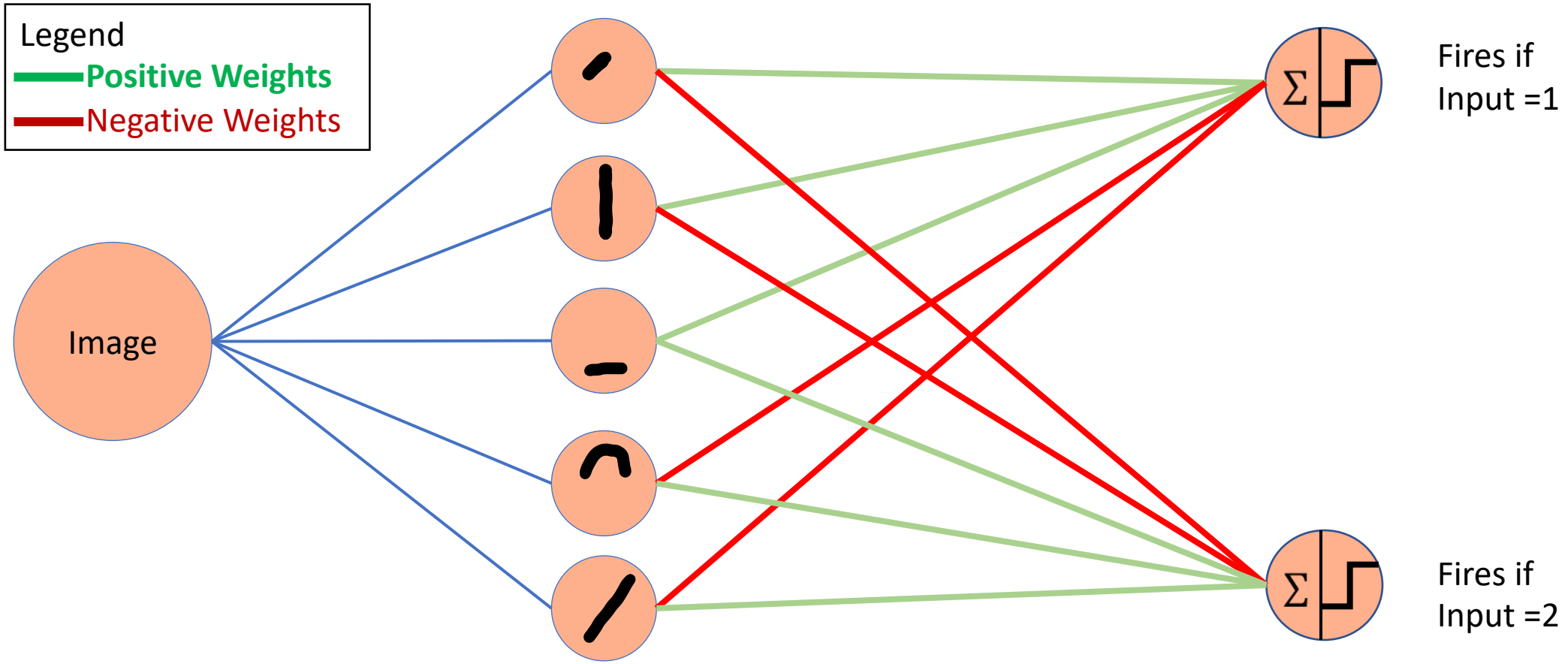
A Neural Network for Classification of 1's and 2's



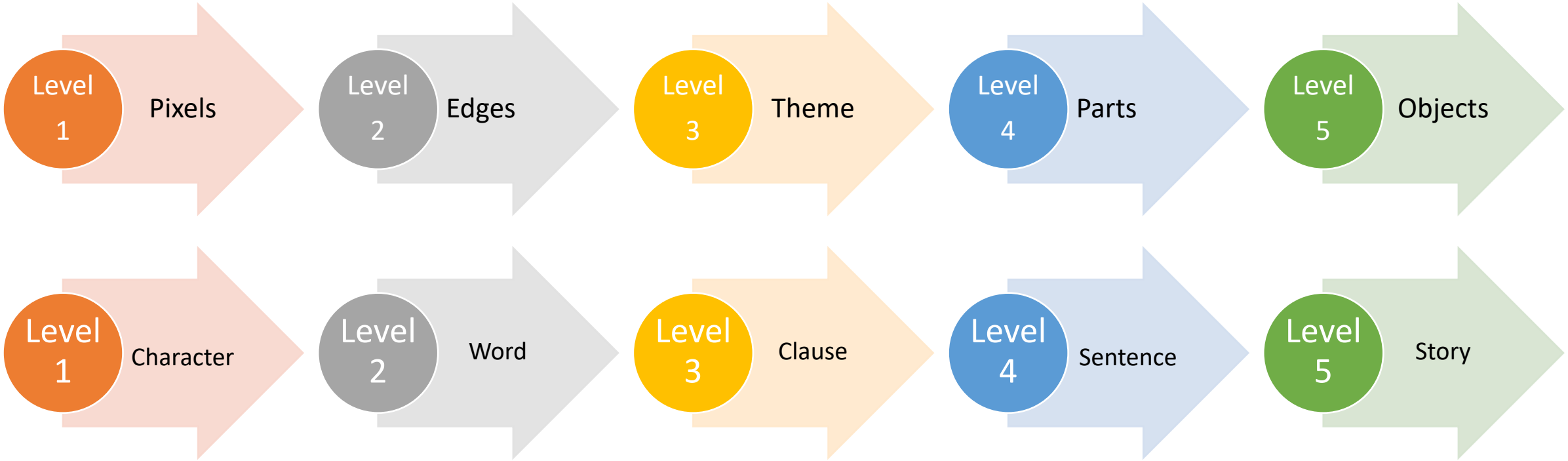
A Neural Network for Classification of 1's and 2's



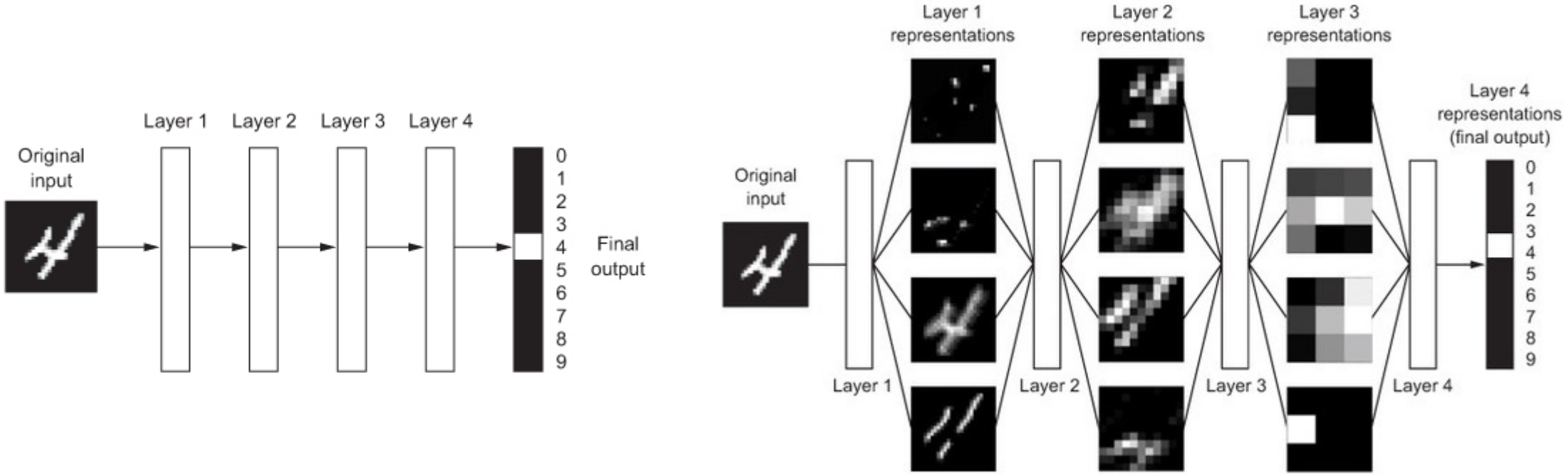
A Neural Network for Classification of 1's and 2's



Hierarchical Representations



Hierarchical Representation - Example



This network transforms the digit image into representations that are increasingly different from the original image and increasingly informative about the final result.

Deep Learning in a Nutshell

The premise of deep learning is to provide *some* human intuition of solving the problem based on a hierarchy of feature representation of the input.

Deep Learning in Practice

Introduction to Keras

- We will use Keras for all our deep learning modeling in this course.
- Keras is a model-level library, providing high-level building blocks for developing deep-learning models.
- Keras is a deep-learning framework that provides a convenient way to define and train almost any kind of deep-learning model.
- It runs both on CPU or GPU.
- It has built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, and so on.
- It relies on a specialized tensor library serving as the *backend engine* of Keras, such as **TensorFlow.**

Model Development in Keras

The typical Keras workflow looks just like that example:

- Define your training data: input tensors and target tensors.
- Define a network of layers (or *model*) that maps your inputs to your targets.

```
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(784,)))
model.add(layers.Dense(10, activation='softmax'))
```

- Configure the learning process by choosing a loss function, an optimizer, and some metrics to monitor.

```
m.compile(optimizer=optimizers.RMSprop(),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

- Iterate on your training data by calling the fit() method of your model.

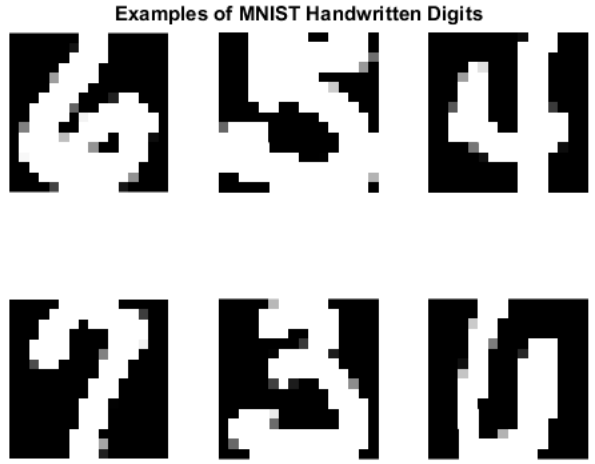
```
model.fit(input_tensor, target_tensor, batch_size=128, epochs=10)
```

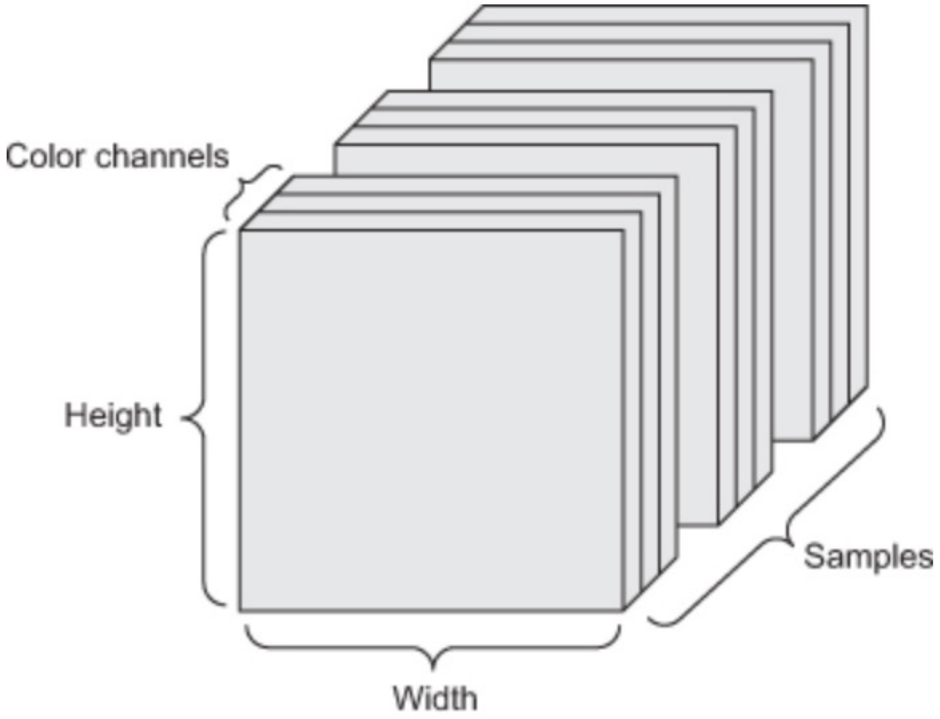
Example 1 - MNIST

- Classify grayscale images of handwritten digits (28 × 28 pixels) into their 10 categories (0 through 9).
- We'll use the MNIST dataset: a set of 60,000 training images, plus 10,000 test images, assembled by the National Institute of Standards and Technology (the NIST in MNIST) in the 1980s.
- MNIST is like the "Hello World" of deep learning!

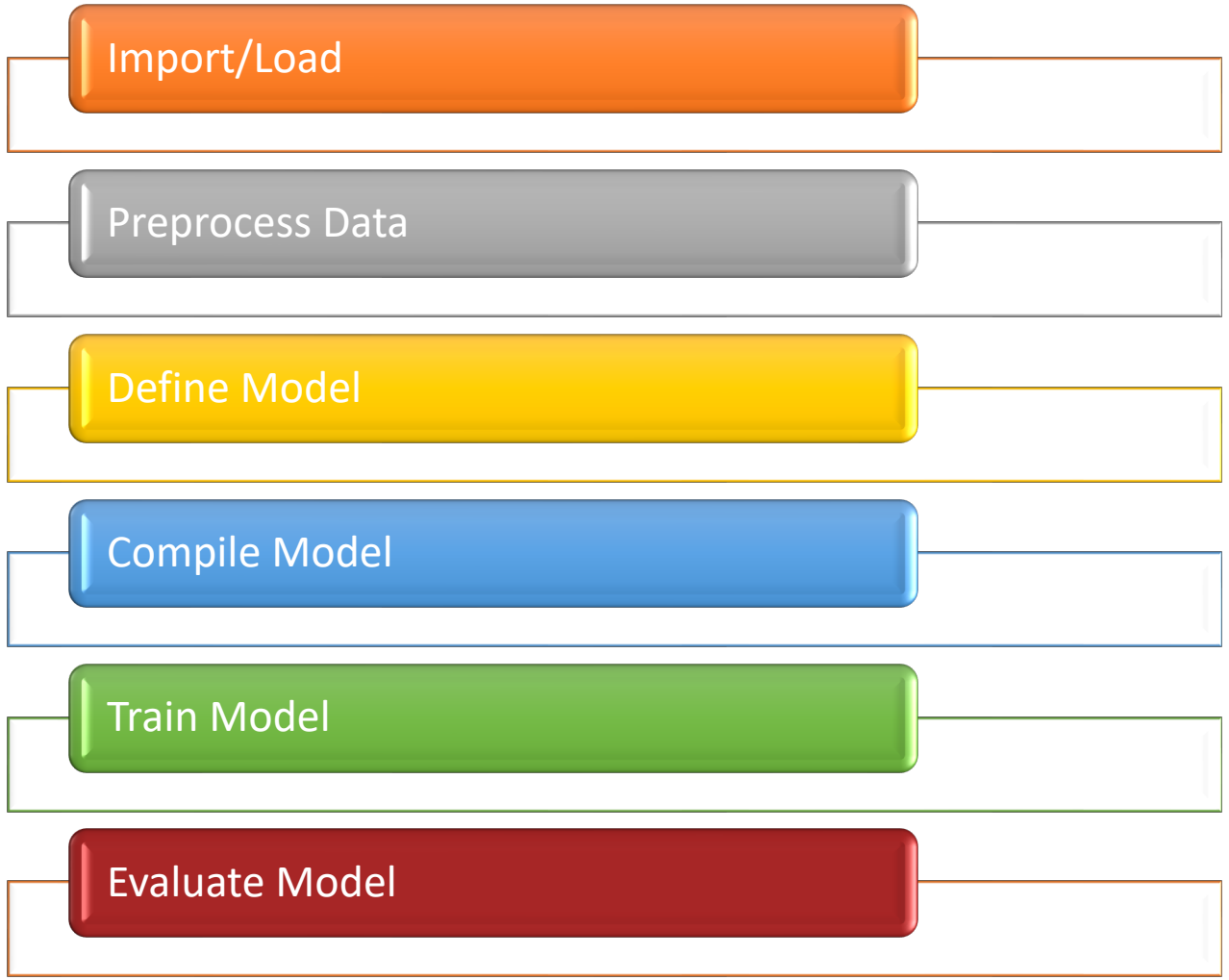
Image Data

- Images typically have three dimensions: height, width, and color depth. MNIST images however are grayscale and have have only a single color channel and could thus be stored in 2D tensors, but by convention image tensors are always 3D.
- The TensorFlow machine-learning framework, from Google, places the color-depth axis at the end: i.e. (samples, height, width, color_depth).
- e.g. a batch of 128 (256x256 pixel) color images with could be stored in a tensor of shape (128, 256, 256, 3).





Overview of the Code



Import

```
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import tensorflow as tf

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense,Dropout
from tensorflow.keras.optimizers import RMSprop, Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping

from keras.utils import np_utils
```

```
tf.random.set_seed(42)
```

Load

The images are encoded as Numpy arrays, and the labels are an array of digits, ranging from 0 to 9. The images and labels have a one-to-one correspondence.

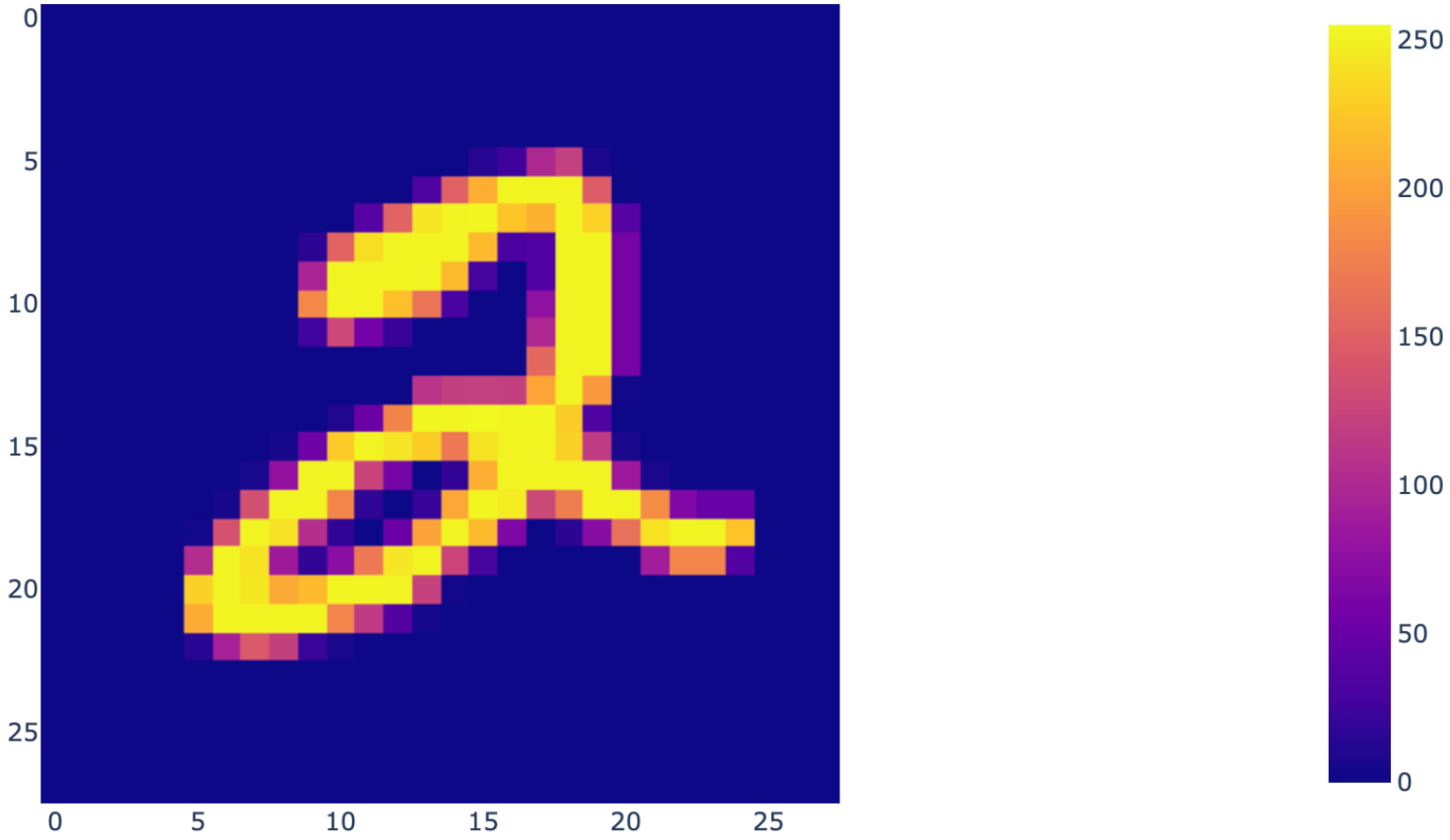
```
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
print('Dimension of the training input is:', X_train.shape)
print('Dimension of the test input is:',X_test.shape)

print('Dimension of the training target is:',Y_train.shape)
print('Dimension of the test target is:',Y_test.shape)
```

```
Dimension of the training input is: (60000, 28, 28)
Dimension of the test input is: (10000, 28, 28)
Dimension of the training target is: (60000,)
Dimension of the test target is: (10000,)
```

Input Example

```
:  1  px.imshow(X_train[5])
```



Preprocess

Before training, we'll preprocess the data by reshaping it into the shape the network expects and scaling it so that all values are in the [0, 1] interval. Previously, our training images, were stored in an array of shape (60000, 28, 28) of type uint8 with values in the [0, 255] interval. We transform it into a float32 array of shape (60000, 28 * 28) with values between 0 and 1.

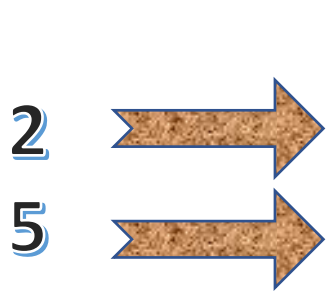
```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
print('Dimension of the training input is:', X_train.shape)
print('Dimension of the test input is:',X_test.shape)
```

```
Dimension of the training input is: (60000, 784)
Dimension of the test input is: (10000, 784)
```

X_train = X_train/255
X_test = X_test/255

Preprocess (one-hot encoding)

- Y_train = np_utils.to_categorical(Y_train, 10)
- Y_test = np_utils.to_categorical(Y_test, 10)



One-Hot Encoded Format

0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0

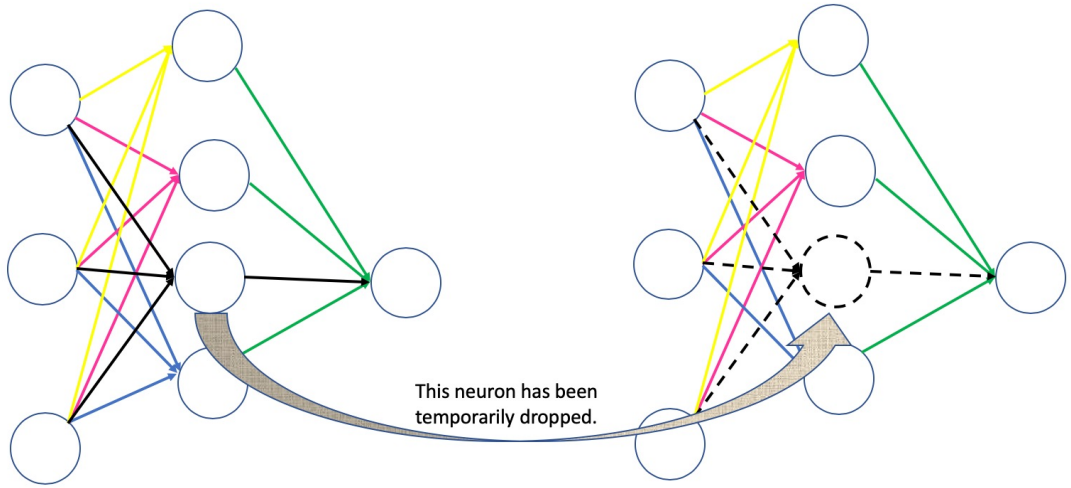
Defining the Model

- The core building block of neural networks is the “layer.”
- Layers extract representations out of the data fed into them—hopefully, representations that are more meaningful for the problem at hand.
- Most of deep learning consists of chaining together simple layers that will implement a form of progressive data distillation.
- Last layer is a 10-way softmax layer. It will return an array of 10 probability scores (summing to 1). Each score will be the probability that the current digit image belongs to one of our 10 digit classes.

```
model = Sequential()
model.add(Dense(128, activation = 'relu', input_shape = (784,)))
model.add(Dropout(0.1))
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.1))
model.add(Dense(10, activation = 'softmax'))
```

Drop-Out

- A dropout temporarily drops connections of neurons from the dense layer randomly within a single training iteration.
- This is a technique for avoiding overfittng. Overfitting happens when the model cannot generalize well beyond the training data. In other words it has learned the training set along with any noise, too well, but it has not learned the underlying structure of the data.
- What dropout technique does, is that it forces the network not to rely on any PARTICULAR neuron and prevents neurons from co-adapting too much by ensuring that the model is robust to the loss of any individual piece of information.
- In other words, at each training step the dropout procedure creates a different network by randomly removing some of the neurons.
- So in the first iteration, some neurons will be randomly deleted and the network will then update the remaining appropriate weights. In the next iteration it will restore those neurons and deletes some new set of randomly chosen nodes. And it will repeat this process.
- After the training, the network will always be used with all the nodes present in the hidden layers.



Model Summary

```
7 model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_23 (Dense)             (None, 128)               100480

 dropout_7 (Dropout)          (None, 128)               0

 dense_24 (Dense)             (None, 128)               16512

 dropout_8 (Dropout)          (None, 128)               0

 dense_25 (Dense)             (None, 10)                1290

=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
```

$(784 + 1) \times 128 = 100{,}480$

$(128 + 1) \times 128 = 16{,}512$

$(128 + 1) \times 10 = 1{,}290$

Compiling the Model

- To make the network ready for training, we need to pick three more things, as part of the compilation step:

  1- Loss function is used to find error or deviation in the learning process. (e.g. mean_squared_error, mean_absolute_error, binary_croosentropy, categorical_crossentropy, etc.)
  2- Optimization is the rule that updates the weights and optimizes them by comparing the prediction and the loss function. (e.g. SGD, RMSProp, Adam, etc.)
  3- Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process. (e.g. accuracy, binary_accuracy, etc.)

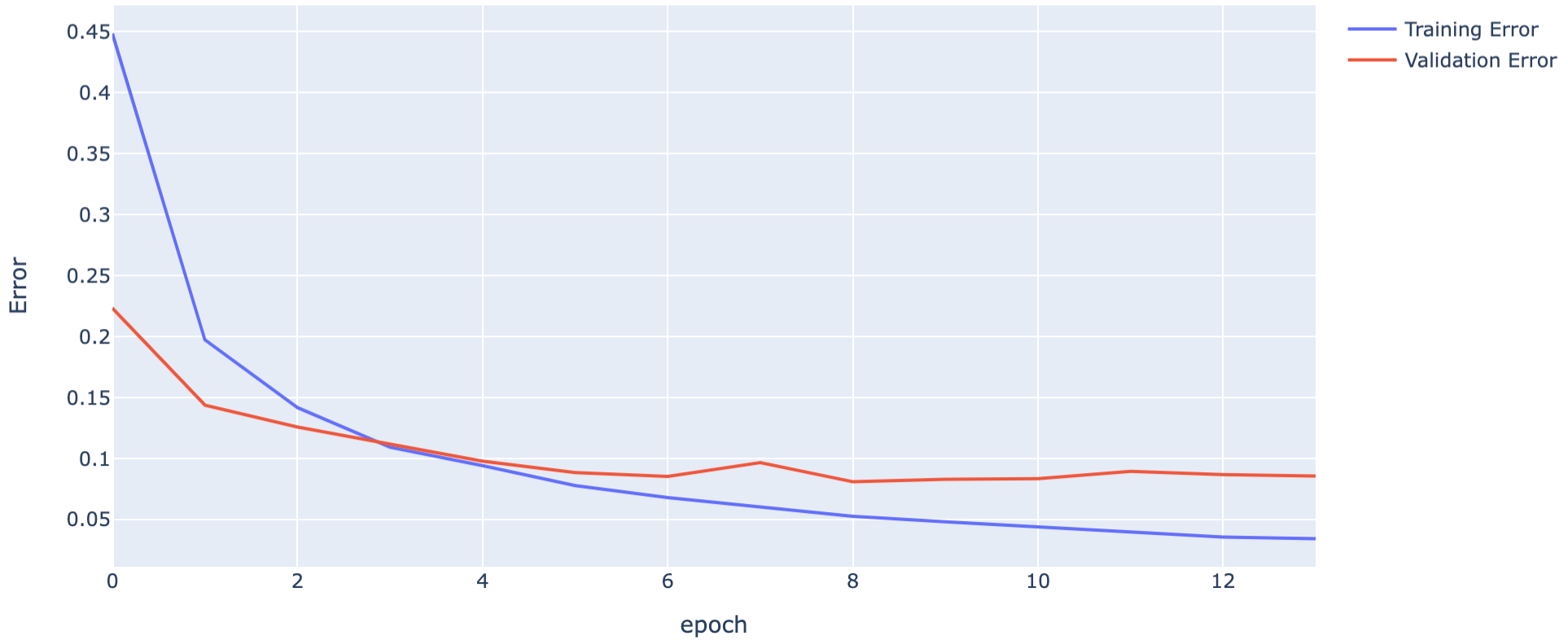
```
model.compile(loss = 'categorical_crossentropy', optimizer = 'RMSprop', metrics = ['categorical_accuracy'])
```

Fitting the Model

- When you call fit, the network will start to iterate on the training data in mini-batches of 256 samples, 5 times over (each iteration over all the training data is called an epoch).
- At each iteration, the network will compute the gradients of the weights with regard to the loss on the batch, and update the weights accordingly.
- Note that deep-learning models don’t process an entire dataset at once; rather, they break the data into small batches. In this example batch size is 256.

```
checkpoint = EarlyStopping(monitor='val_loss',patience=5, verbose=1, mode='auto', restore_best_weights=True)
model.fit(X_train, Y_train, epochs =100, batch_size = 256, validation_split = 1/6, verbose = 1, callbacks = [checkpoint])
```

```
fig = go.Figure()
fig.add_trace(go.Scatter(y=model.history.history['loss'],
                         mode='lines',
                         name='Training Error'))
fig.add_trace(go.Scatter(y=model.history.history['val_loss'],
                         mode='lines',
                         name='Validation Error'))
fig.update_layout(yaxis_title = 'Error', xaxis_title = 'epoch')
fig.show()
```



Evaluating the Model

print(model.metrics_names[0],'is', model.evaluate(X_test, Y_test)[0],'and ', model.metrics_names[1],'is', model.evaluate(X_test, Y_test)[1])

```
loss is 0.0720235109329223 6 and  categorical_accuracy is 0.9783999919891357
```

Example 2 - IMDB

- IMDB dataset is a set of 50,000 reviews from the Internet Movie Database.
- They’re split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting of 50% negative and 50% positive reviews.
- The reviews (sequences of words) have been turned into sequences of integers, where each integer stands for a specific word in a dictionary.

Import

```
from keras import models
from keras import layers
from keras import optimizers

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from keras.datasets import imdb
```

Load

```
from keras.datasets import imdb

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

Do the following to force imdb.load_data to allow pickle if you get this error upon loading the data
ValueError: Object arrays cannot be loaded when allow_pickle=False

```
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)


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

# restore np.load for future normal usage
np.load = np_load_old
```

Example of a Review

The variables train_data and test_data are lists of reviews; each review is a list of word indices (encoding a sequence of words). train_labels and test_labels are lists of 0s and 1s, where 0 stands for negative and 1 stands for positive:

```
: print(train_labels[0])
print(train_data[0])
```

```
1
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 45
, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167,
, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 4
515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8,
38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415,
, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71,
5, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486,
28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334,
]
```

```
"? this film was just brilliant casting location scenery story direction everyone's really suited the part they playe
d and you could just imagine being there robert ? is an amazing actor and now the same being director ? father came f
rom the same scottish island as myself so i loved the fact there was a real connection with this film the witty remar
ks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for
? and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad a
nd you know what they say if you cry at a film it must have been good and this definitely was also ? to the two littl
e boy's that played the ? of norman and paul they were just brilliant children are often left out of the ? list i thi
nk because the stars that play them all grown up are such a big profile for the whole film but these children are ama
zing and should be praised for what they have done don't you think the whole story was so lovely because it was true
and was someone's life after all that was shared with us all"
```

Preprocess

- We can’t feed lists of integers into a neural network. We have to turn the lists into tensors.
- We will use one-hot encoding. e.g. if the first review has only two words with indices of 4 and 2 (i.e. [4,2]), then we'll have a 10,000-dimensional vector that would be all 0s except for indices 4 and 2, which would be 1s. (i.e. [0,0,1,0,1,0,0,...])

```
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1
    return results
# print(list(enumerate(train_data[0])))
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

Validation Set

```
x_val = x_train[:10000]
partial_x_train = x_train[10000:]

# We should also vectorize your labels:

y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

Model

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

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

Model Performance

```
History.history.keys()
```

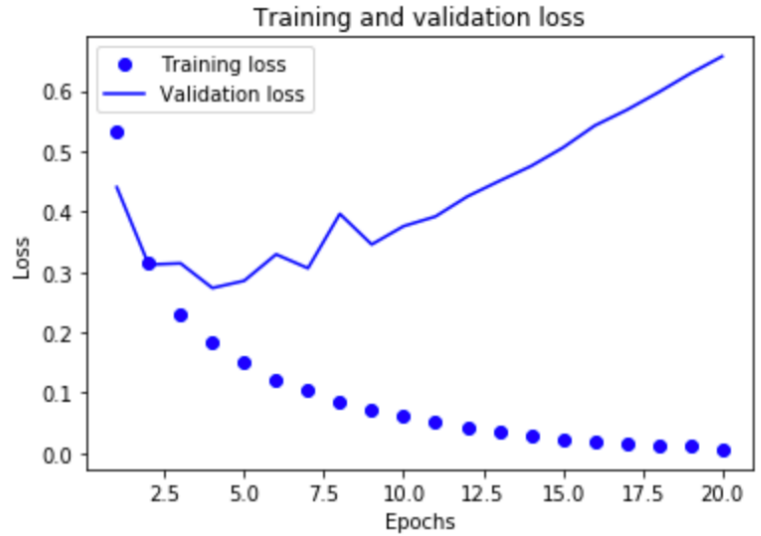
```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

```
loss_values = History.history['loss']
val_loss_values = History.history['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()
```

```
<matplotlib.legend.Legend at 0xd287f2860>
```



Preventing overfitting

To prevent overfitting, we could stop training after 4 epochs. Let's train a new network from scratch for four epochs and then evaluate it on the test data.

```
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'])
model.fit(x_train, y_train, epochs=4, batch_size=512)

results = model.evaluate(x_test, y_test)
results
```

Model Performance on Test Set

```
model.predict(x_test)
```

```
array([[0.9318503 ],
       [0.84786206],
       [0.99937624],
       ...,
       [0.57800645],
       [0.00264496],
       [0.7785329 ]], dtype=float32)
```

As you can see, the network is confident for some samples (0.99 or more, or 0.01 or less) but less confident for others (0.6, 0.4).

Example 3:Reuters

- Reuters dataset is a set of short newswires and their topics, published by Reuters in 1986.
- It’s a simple, widely used toy dataset for text classification.
- There are 46 different topics; some topics are more represented than others, but each topic has at least 10 examples in the training set.
- Your task is to build a network to classify Reuters newswires into 46 mutually exclusive topics.

Import

```
from keras import models
from keras import layers
from keras import optimizers

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from keras.datasets import reuters
```

Load

Do the following to force reuters.load_data to allow pickle if you get this error upon loading the data
ValueError: Object arrays cannot be loaded when allow_pickle=False

```
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)
(x_train, y_train), (x_test, y_test) = reuters.load_data(num_words = 10000)
np.load = np_load_old
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(8982,)
(8982,)
(2246,)
(2246,)
```

Example of a Review

As with the IMDB reviews, each example is a list of integers (word indices). The label associated with an example is an integer between 0 and 45—a topic index:

```
print(x_train[10])
```

```
[1, 245, 273, 207, 156, 53, 74
14, 61, 451, 4329, 17, 12]
```

```
y_train[10]
```

```
3
```

```
'? period ended december 31 shr profit 11 cts vs loss 24 cts net profit 224 271 vs loss 511 349 revs 7 258 688 vs 7 2
00 349 reuter 3'
```

Preprocess

- We can vectorize the input data with the same code as in the imbd example.
- To vectorize the output we use one-hot encoding that consists of embedding each label as an all-zero vector with a 1 in the place of the label index.

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

x_train=vectorize_sequences(x_train)
x_test=vectorize_sequences(x_test)
```

```
x_train.shape
```

```
(8982, 10000)
```

```
y_train = vectorize_sequences(y_train, dimension = 46)
y_test = vectorize_sequences(y_test, dimension = 46)
```

```
# Alternatively

from keras.utils.np_utils import to_categorical
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Validation Set

We set apart 1,000 samples in the training data to use as a validation set.

```
# Validation
x_val = x_train[:1000]
y_val = y_train[:1000]

partial_x_train = x_train[1000:]
partial_y_train = y_train[1000:]
```

Model Architecture

- Note that the number of output classes has gone from 2 in imbd example to 46. i.e. The dimensionality of the output space is much larger.
- In a stack of Dense layers, each layer can only access information present in the output of the previous layer.
- If one layer drops some information relevant to the classification problem, this information can never be recovered by later layers: each layer can potentially become an information bottleneck.
- In the imdb example, we used 16-dimensional intermediate layers, but a 16-dimensional space may be too limited to learn to separate 46 different classes.
- i.e. such small layers may act as information bottlenecks, permanently dropping relevant information.

Model

```
m = models.Sequential()
m.add(layers.Dense(64, activation = 'relu', input_shape = (10000,) ))
m.add(layers.Dense(64, activation = 'relu' ))
m.add(layers.Dense(46, activation = 'softmax'))
```

```
m.compile(optimizer=optimizers.RMSprop(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
m.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 64)                640064
_________________________________________________________________
dense_29 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_30 (Dense)             (None, 46)                2990
=================================================================
Total params: 647,214
Trainable params: 647,214
Non-trainable params: 0
_________________________________________________________________
```

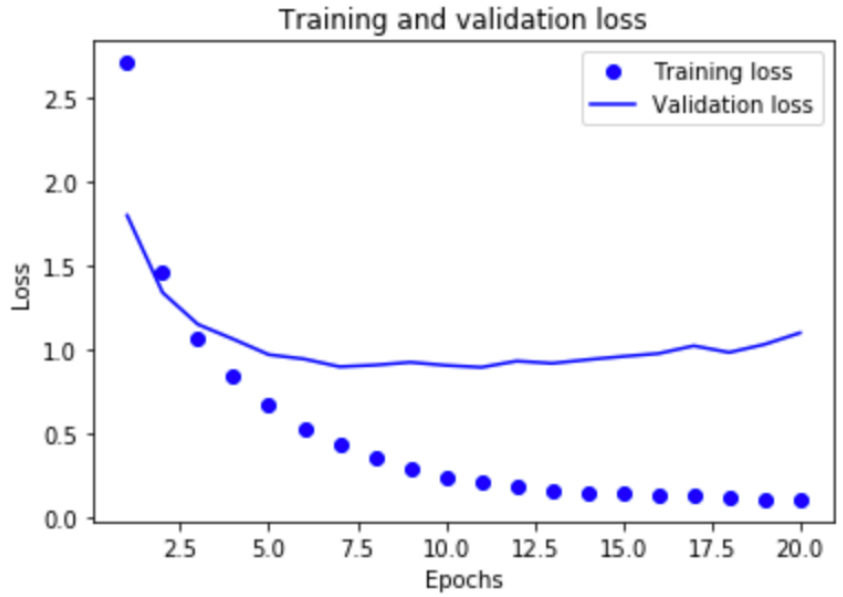
Training the Model

Set the batch_size to 512 and train the network for 20 epochs:

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

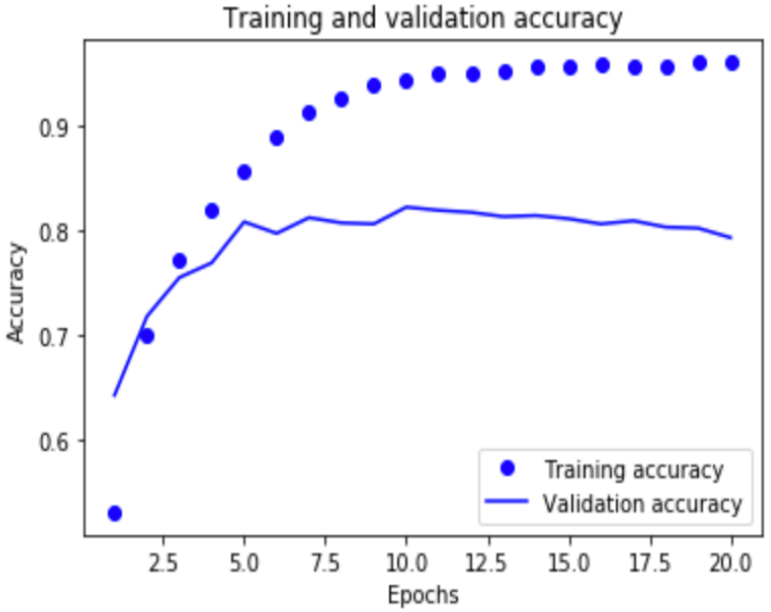
Model Performance

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




```
tr_acc = history.history['acc']
val_acc = history.history['val_acc']
plt.plot(epochs, tr_acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
```

```
<matplotlib.legend.Legend at 0xd296fe2b0>
```



Preventing overfitting

- The network begins to overfit after about 10th epoch.
- Train a new network from scratch for ten epochs and then evaluate it on the test set.

```
history = m.fit(partial_x_train,
                partial_y_train,
                epochs = 10,
                batch_size = 512,
                validation_data = (x_val, y_val))
```

```
results = m.evaluate(x_test, y_test)
results
```

```
2048/2246 [==========================>...] - ETA: 0s
```

```
[1.5695256575451084, 0.7733748887440824]
```

Underfitting

- We mentioned earlier that because the final outputs are 46-dimensional, we should avoid layers with many fewer than 46 hidden units.
- We examine what happens when we introduce an information bottleneck by having intermediate layers that are significantly less than 46-dimensional: for example, 4-dimensional.

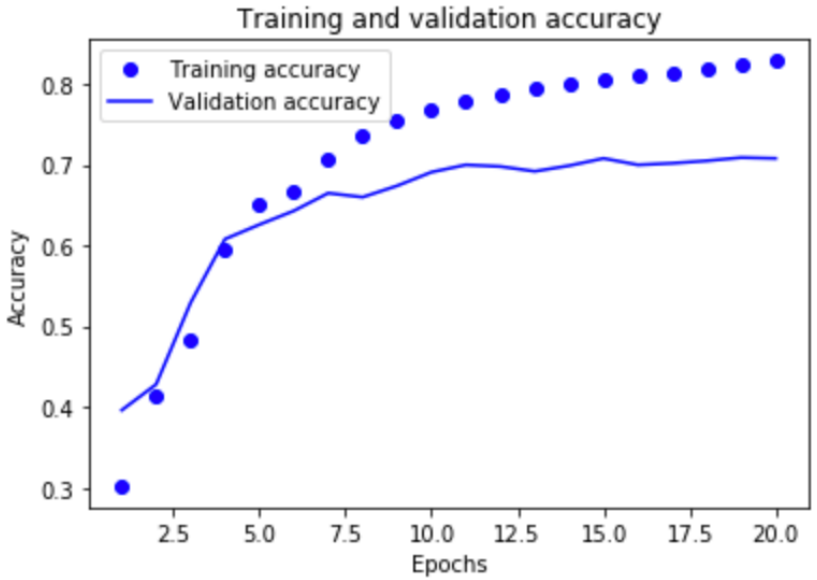
Underfitting

- The network now peaks at ~70% validation accuracy, an 8% absolute drop.
- This drop is mostly due to the fact that you're trying to compress a lot of information (enough information to recover the separation hyperplanes of 46 classes) into an intermediate space that is too low-dimensional.
- The network is able to cram *most* of the necessary information into these four-dimensional representations, but not all of it.

```
m = models.Sequential()
m.add(layers.Dense(64, activation = 'relu', input_shape = (10000,) ))
m.add(layers.Dense(4, activation = 'relu' ))
m.add(layers.Dense(46, activation = 'softmax'))

m.compile(optimizer=optimizers.RMSprop(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

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



Summary

- If you're trying to classify data points among *N* classes, your network should end with a Dense layer of size *N*.
- In a multiclass classification problem, your network should end with a softmax activation so that it will output a probability distribution over the *N* output classes.
- Categorical crossentropy is almost always the loss function you should use for such problems. It minimizes the distance between the probability distributions output by the network and the true distribution of the targets.
- If you need to classify data into a large number of categories, you should avoid creating information bottlenecks in your network due to intermediate layers that are too small.

References and Recommended Readings

- Networks and Deep Learning by Michael Nielsen
- Deep Learning by Ian Goodfellow and Yoshua Bengio and Aaron Courville
- Learning from Data by Yaser Abu-Mustafa, Malik Magdon-Ismail, and Hsuan_Tien Lin
- http://deeplearning.net
- Deep Learning with Python by Chollet