

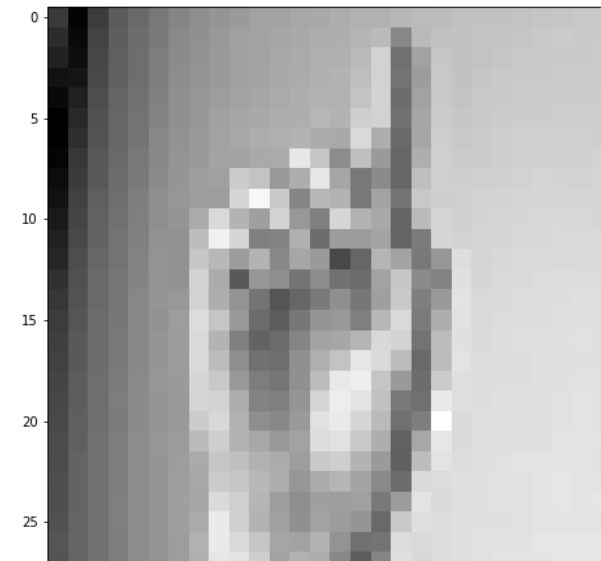
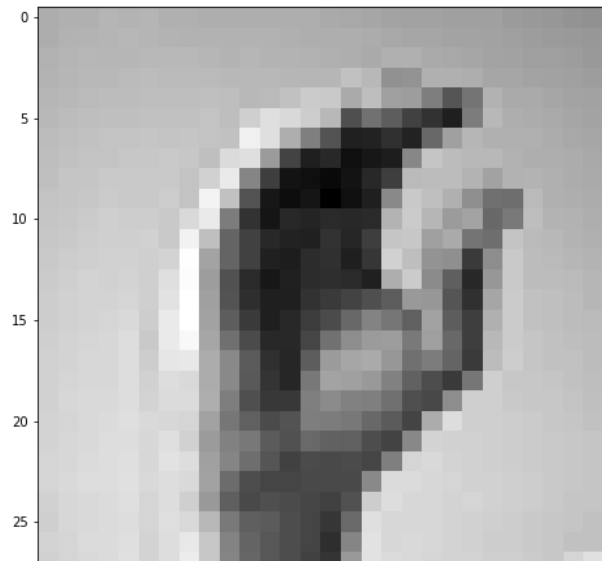
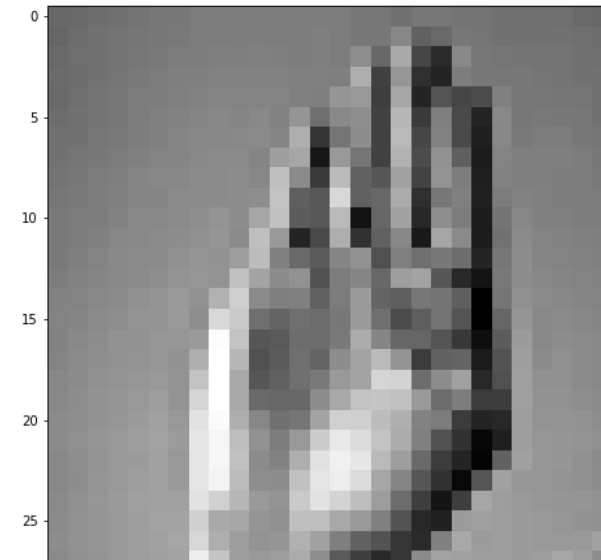
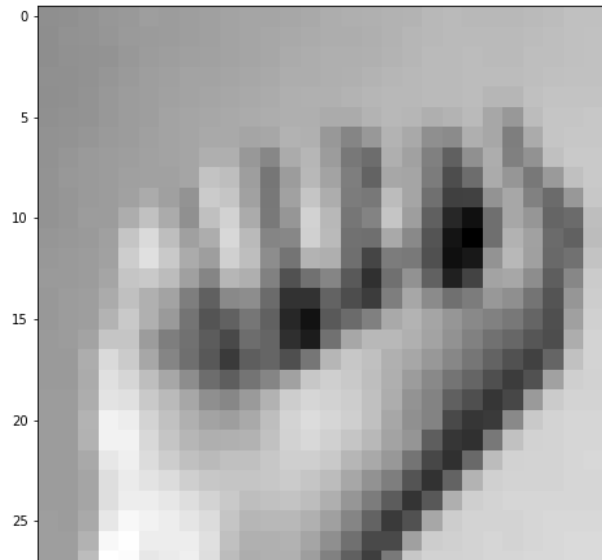
Defining neural networks with Keras

INTRODUCTION TO TENSORFLOW IN PYTHON

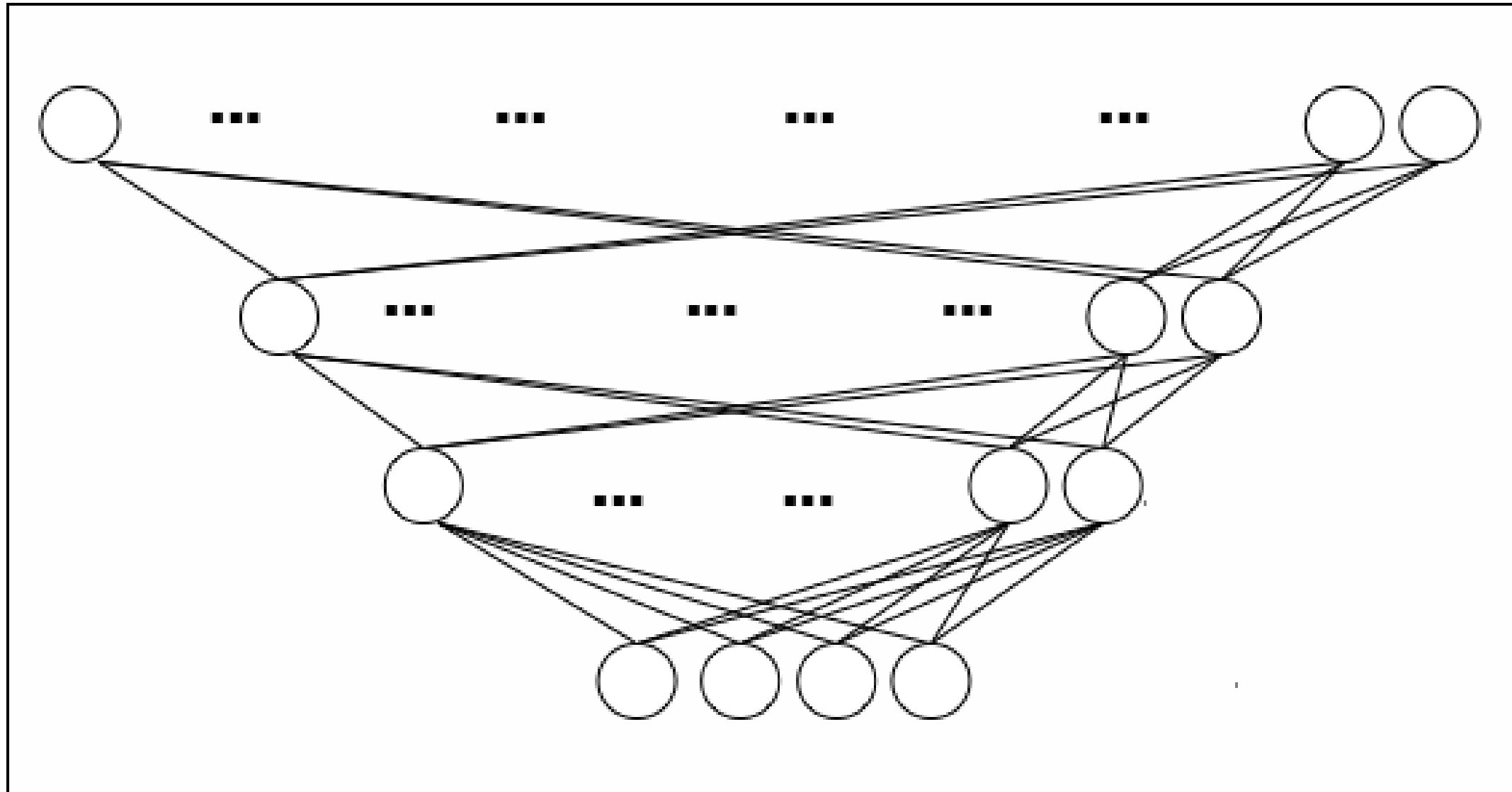


Isaiah Hull
Economist

Classifying sign language letters



The sequential API



The sequential API

- Input layer
- Hidden layers
- Output layer
- Ordered in sequence

Building a sequential model

```
# Import tensorflow  
from tensorflow import keras
```

```
# Define a sequential model  
model = keras.Sequential()
```

```
# Define first hidden layer  
model.add(keras.layers.Dense(16, activation='relu', input_shape=(28*28,)))
```

Building a sequential model

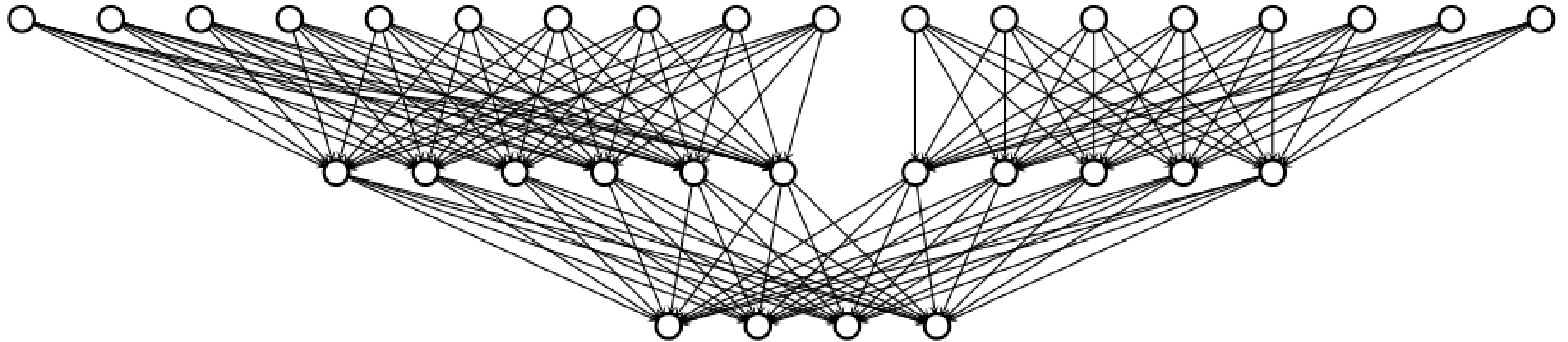
```
# Define second hidden layer  
model.add(keras.layers.Dense(8, activation='relu'))
```

```
# Define output layer  
model.add(keras.layers.Dense(4, activation='softmax'))
```

```
# Compile the model  
model.compile('adam', loss='categorical_crossentropy')
```

```
# Summarize the model  
print(model.summary())
```

The functional API



Using the functional API

```
# Import tensorflow
import tensorflow as tf

# Define model 1 input layer shape
model1_inputs = tf.keras.Input(shape=(28*28,))

# Define model 2 input layer shape
model2_inputs = tf.keras.Input(shape=(10,))
```

```
# Define layer 1 for model 1
model1_layer1 = tf.keras.layers.Dense(12, activation='relu')(model1_inputs)

# Define layer 2 for model 1
model1_layer2 = tf.keras.layers.Dense(4, activation='softmax')(model1_layer1)
```


Using the functional API

```
# Define layer 1 for model 2
model2_layer1 = tf.keras.layers.Dense(8, activation='relu')(model2_inputs)

# Define layer 2 for model 2
model2_layer2 = tf.keras.layers.Dense(4, activation='softmax')(model2_layer1)

# Merge model 1 and model 2
merged = tf.keras.layers.add([model1_layer2, model2_layer2])

# Define a functional model
model = tf.keras.Model(inputs=[model1_inputs, model2_inputs], outputs=merged)

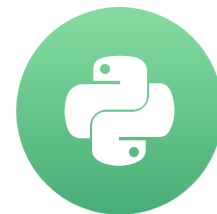
# Compile the model
model.compile('adam', loss='categorical_crossentropy')
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN PYTHON

Training and validation with Keras

INTRODUCTION TO TENSORFLOW IN PYTHON



Isaiah Hull
Economist

Overview of training and evaluation

1. Load and clean data
2. Define model
3. Train and validate model
4. Evaluate model

How to train a model

```
# Import tensorflow
import tensorflow as tf

# Define a sequential model
model = tf.keras.Sequential()
```

```
# Define the hidden layer
model.add(tf.keras.layers.Dense(16, activation='relu', input_shape=(784,)))
```

```
# Define the output layer
model.add(tf.keras.layers.Dense(4, activation='softmax'))
```

How to train a model

```
# Compile model  
model.compile('adam', loss='categorical_crossentropy')
```

```
# Train model  
model.fit(image_features, image_labels)
```

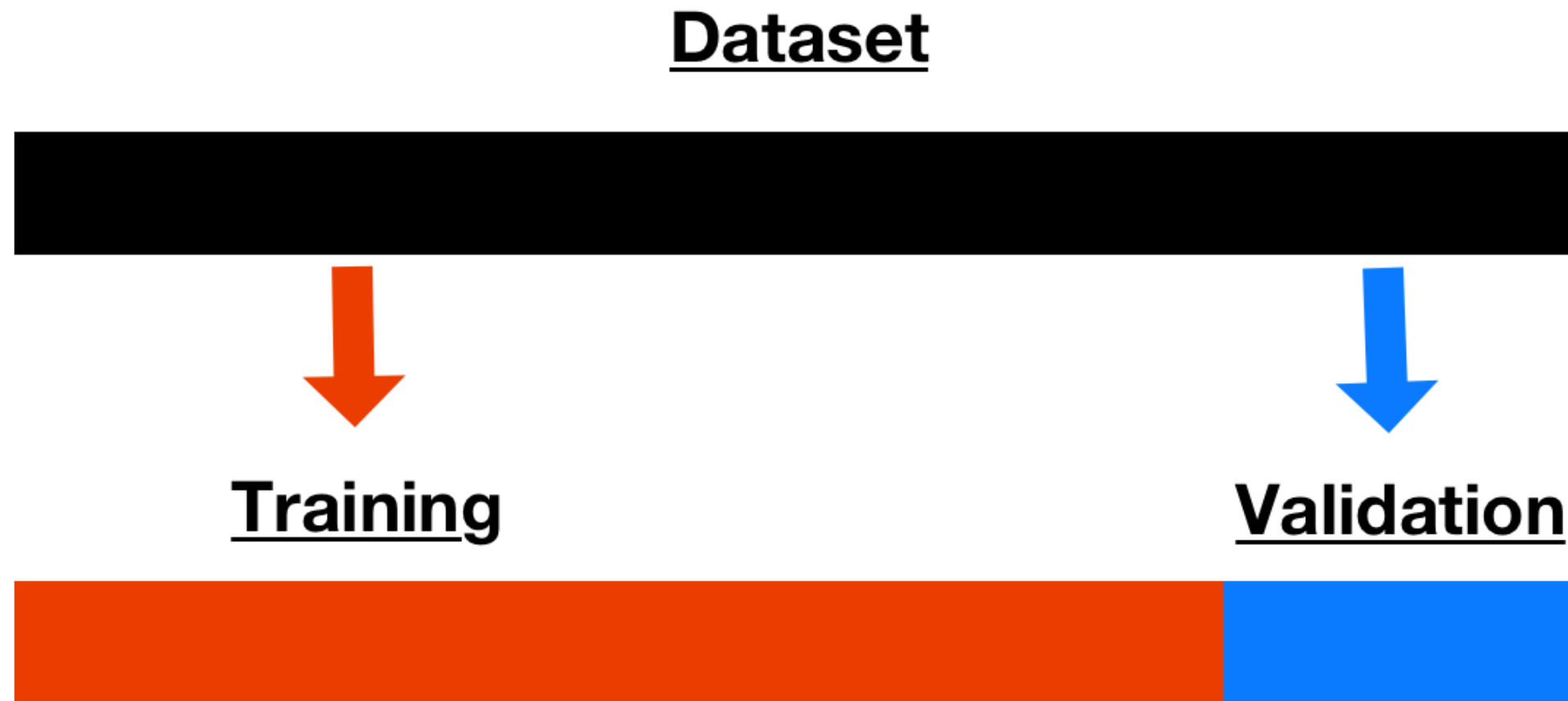
The fit() operation

- Required arguments
 - `features`
 - `labels`
- Many optional arguments
 - `batch_size`
 - `epochs`
 - `validation_split`

Batch size and epochs

		<u>Epochs</u>					
		price	sqft_lot	bedrooms	price	sqft_lot	bedrooms
<u>Batches</u>	221900.0	5650	3	Batch 1	221900.0	5650	3
	538000.0	7212	3		538000.0	7212	3
	180000.0		2		180000.0		2
	604000.0	5000	4		604000.0	5000	4
	510000.0	8080	3		510000.0	8080	3
	1225000.0	101930	4	Batch 2	1225000.0	101930	4
	257500.0	6819	3		257500.0	6819	3
	291850.0		3		291850.0		3
	229500.0	7470	3		229500.0	7470	3
	323000.0	6560	3		323000.0	6560	3
	662500.0	9796	3	Batch 3	662500.0	9796	3
	468000.0	6000	2		468000.0	6000	2
	310000.0		3		310000.0		3
	400000.0	9680	3		400000.0	9680	3
	530000.0	4850	5		530000.0	4850	5

Performing validation



Performing validation

```
# Train model with validation split  
model.fit(features, labels, epochs=10, validation_split=0.20)
```

Performing validation

Train on 1599 samples, validate on 400 samples

Epoch 1/10

1599/1599=====] - 0s 159us/sample - loss: 1.2291 - val_loss: 1.0122

Epoch 2/10

1599/1599=====] - 0s 60us/sample - loss: 0.8873 - val_loss: 0.7181

Epoch 3/10

1599/1599=====] - 0s 61us/sample - loss: 0.6476 - val_loss: 0.5414

Epoch 4/10

1599/1599=====] - 0s 58us/sample - loss: 0.4974 - val_loss: 0.4254

Epoch 5/10

1599/1599=====] - 0s 57us/sample - loss: 0.3958 - val_loss: 0.3544

Epoch 6/10

1599/1599=====] - 0s 62us/sample - loss: 0.3222 - val_loss: 0.2936

Epoch 7/10

1599/1599=====] - 0s 58us/sample - loss: 0.2730 - val_loss: 0.2555

Epoch 8/10

1599/1599=====] - 0s 56us/sample - loss: 0.2320 - val_loss: 0.2131

Epoch 9/10

1599/1599=====] - 0s 59us/sample - loss: 0.1957 - val_loss: 0.1843

Epoch 10/10

1599/1599=====] - 0s 55us/sample - loss: 0.1663 - val_loss: 0.1657

Changing the metric

```
# Recompile the model with the accuracy metric  
model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Train model with validation split  
model.fit(features, labels, epochs=10, validation_split=0.20)
```

Changing the metric

Train on 1599 samples, validate on 400 samples

Epoch 1/10

1599/1599=====] - 0s 174us/sample - loss: 1.2956 - acc: 0.4196 - val_loss: 1.1189 - val_acc: 0.5075

Epoch 2/10

1599/1599=====] - 0s 59us/sample - loss: 0.9356 - acc: 0.7949 - val_loss: 0.7843 - val_acc: 0.8225

Epoch 3/10

1599/1599=====] - 0s 59us/sample - loss: 0.6657 - acc: 0.9037 - val_loss: 0.5588 - val_acc: 0.8925

Epoch 4/10

1599/1599=====] - 0s 58us/sample - loss: 0.4898 - acc: 0.9206 - val_loss: 0.4220 - val_acc: 0.9175

Epoch 5/10

1599/1599=====] - 0s 59us/sample - loss: 0.3734 - acc: 0.9681 - val_loss: 0.3319 - val_acc: 0.9825

Epoch 6/10

1599/1599=====] - 0s 61us/sample - loss: 0.2975 - acc: 0.9762 - val_loss: 0.2907 - val_acc: 0.9075

Epoch 7/10

1599/1599=====] - 0s 60us/sample - loss: 0.2414 - acc: 0.9731 - val_loss: 0.2276 - val_acc: 0.9550

Epoch 8/10

1599/1599=====] - 0s 61us/sample - loss: 0.1912 - acc: 0.9887 - val_loss: 0.2026 - val_acc: 0.9525

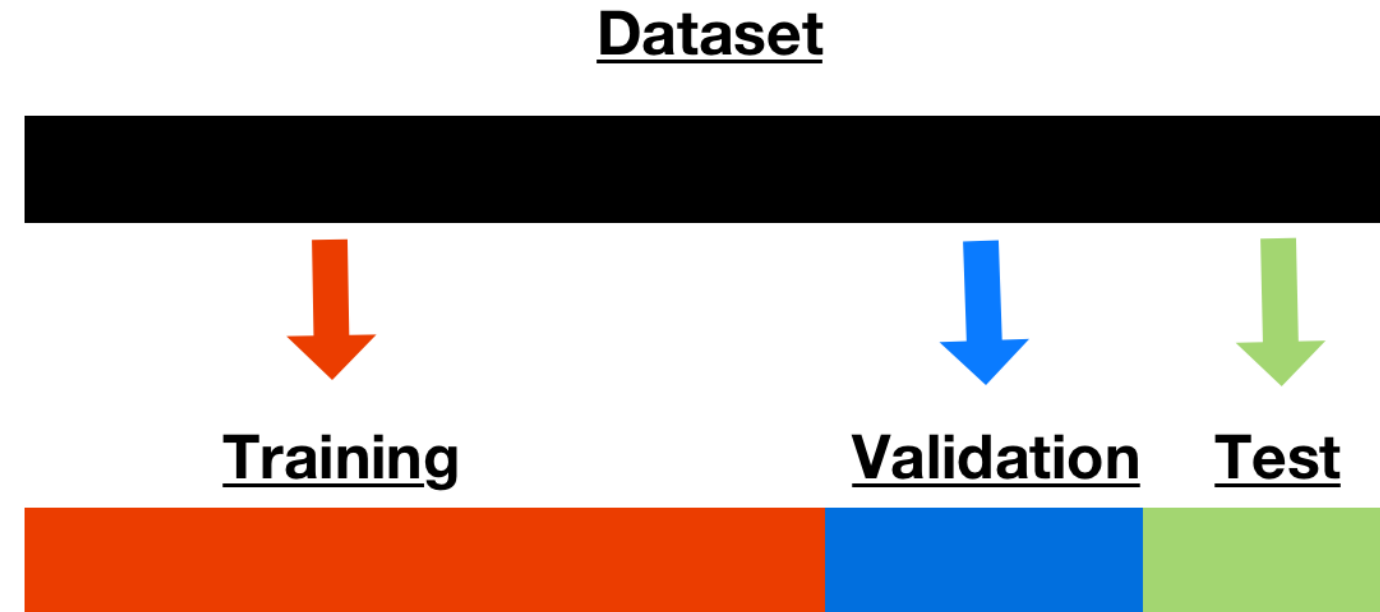
Epoch 9/10

1599/1599=====] - 0s 59us/sample - loss: 0.1649 - acc: 0.9862 - val_loss: 0.1684 - val_acc: 0.9675

Epoch 10/10

1599/1599=====] - 0s 58us/sample - loss: 0.1390 - acc: 0.9912 - val_loss: 0.1374 - val_acc: 0.9825

The evaluation() operation



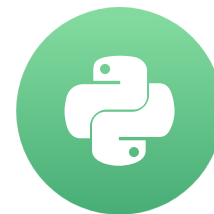
```
# Evaluate the test set  
model.evaluate(test)
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN PYTHON

Training models with the Estimators API

INTRODUCTION TO TENSORFLOW IN PYTHON



Isaiah Hull
Economist

What is the Estimators API?

- High level submodule
- Less flexible
- Enforces best practices
- Faster deployment
- Many premade models

High-Level
TensorFlow APIs

Mid-Level
TensorFlow APIs

Low-level
TensorFlow APIs

Estimators

Layers

Datasets

Metrics

Python

¹ Image taken from https://www.tensorflow.org/guide/premade_estimators

Model specification and training

1. Define feature columns
2. Load and transform data
3. Define an estimator
4. Apply train operation

Defining feature columns

```
# Import tensorflow under its standard alias
import tensorflow as tf

# Define a numeric feature column
size = tf.feature_column.numeric_column("size")
```

```
# Define a categorical feature column
rooms = tf.feature_column.categorical_column_with_vocabulary_list("rooms", \
["1", "2", "3", "4", "5"])
```

Defining feature columns

```
# Create feature column list  
features_list = [size, rooms]
```

```
# Define a matrix feature column  
features_list = [tf.feature_column.numeric_column('image', shape=(784,))]
```

Loading and transforming data

```
# Define input data function
def input_fn():
    # Define feature dictionary
    features = {"size": [1340, 1690, 2720], "rooms": [1, 3, 4]}
    # Define labels
    labels = [221900, 538000, 180000]
    return features, labels
```

Define and train a regression estimator

```
# Define a deep neural network regression
model0 = tf.estimator.DNNRegressor(feature_columns=feature_list,\
    hidden_units=[10, 6, 6, 3])

# Train the regression model
model0.train(input_fn, steps=20)
```

Define and train a deep neural network

```
# Define a deep neural network classifier
model1 = tf.estimator.DNNClassifier(feature_columns=feature_list, \
    hidden_units=[32, 16, 8], n_classes=4)

# Train the classifier
model1.train(input_fn, steps=20)
```

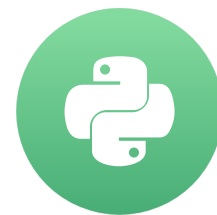
- <https://www.tensorflow.org/guide/estimators>

Let's practice!

INTRODUCTION TO TENSORFLOW IN PYTHON

Congratulations!

INTRODUCTION TO TENSORFLOW IN PYTHON



Isaiah Hull
Economist

What you learned

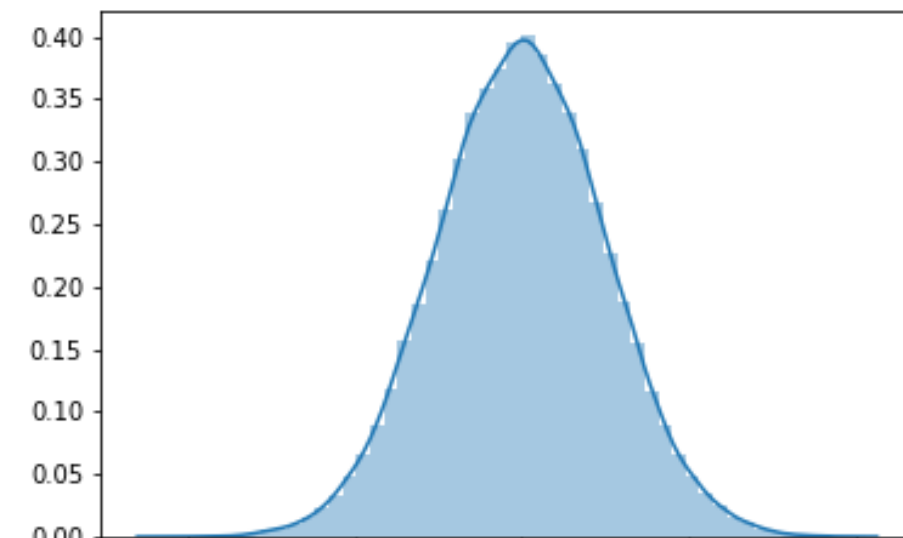
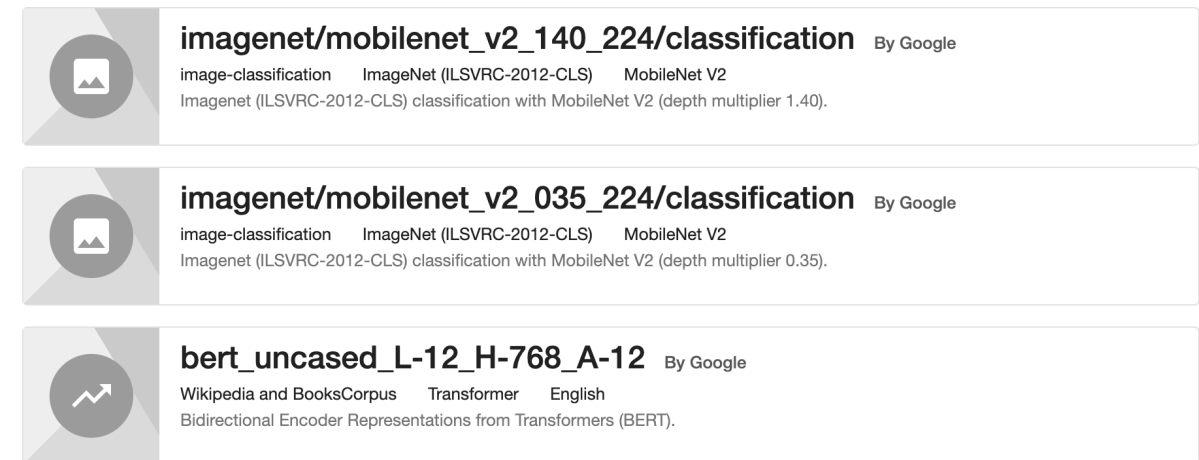
- **Chapter 1**
 - Low-level, basic, and advanced operations
 - Graph-based computation
 - Gradient computation and optimization
- **Chapter 2**
 - Data loading and transformation
 - Predefined and custom loss functions
 - Linear models and batch training

What you learned

- **Chapter 3**
 - Dense neural network layers
 - Activation functions
 - Optimization algorithms
 - Training neural networks
- **Chapter 4**
 - Neural networks in Keras
 - Training and validation
 - The Estimators API

TensorFlow extensions

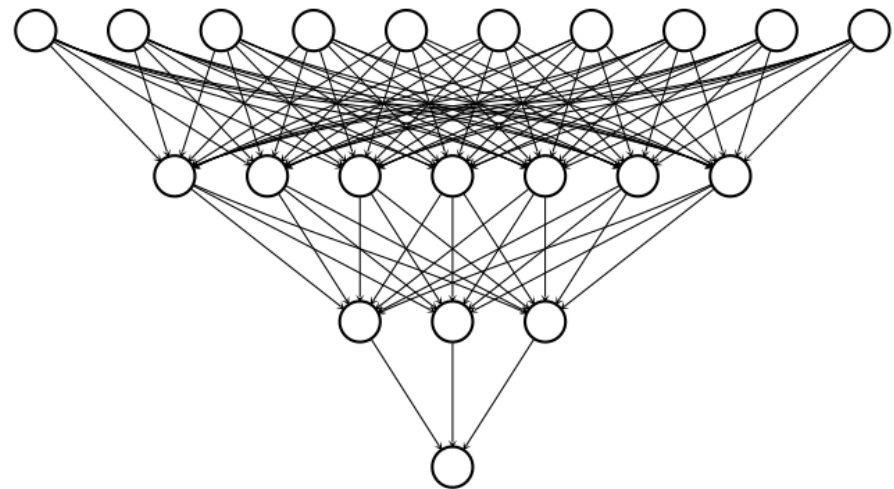
- TensorFlow Hub
 - Pretrained models
 - Transfer learning
- TensorFlow Probability
 - More statistical distributions
 - Trainable distributions
 - Extended set of optimizers



¹ Screenshot from <https://tfhub.dev>.

TensorFlow 2.0

- TensorFlow 2.0
 - `eager_execution()`
 - Tighter `keras` integration
 - `Estimators`
 - `function()`



High-Level
TensorFlow APIs

Estimators

Mid-Level
TensorFlow APIs

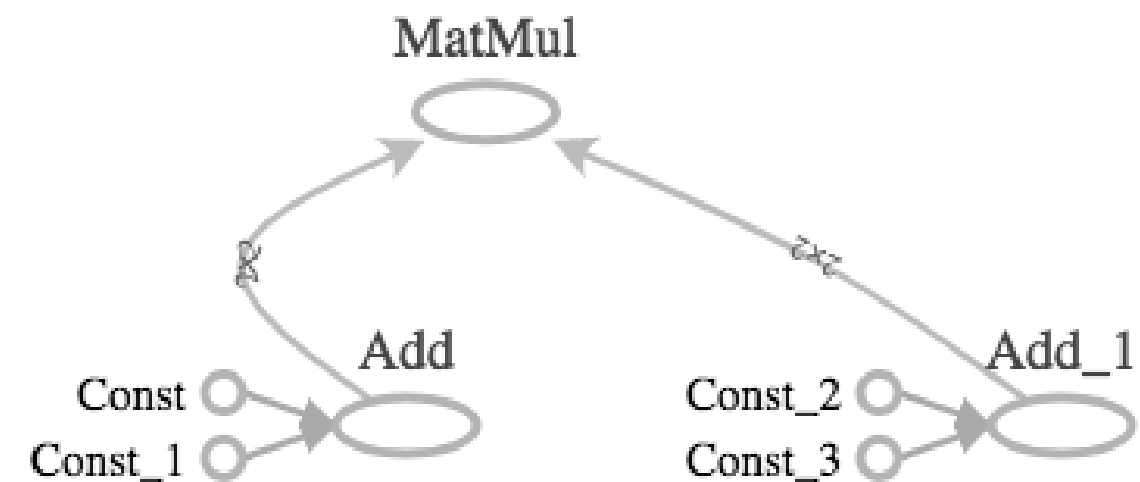
Layers

Datasets

Metrics

Low-level
TensorFlow APIs

Python



¹ Screenshot taken from https://www.tensorflow.org/guide/premade_estimators

Congratulations!

INTRODUCTION TO TENSORFLOW IN PYTHON