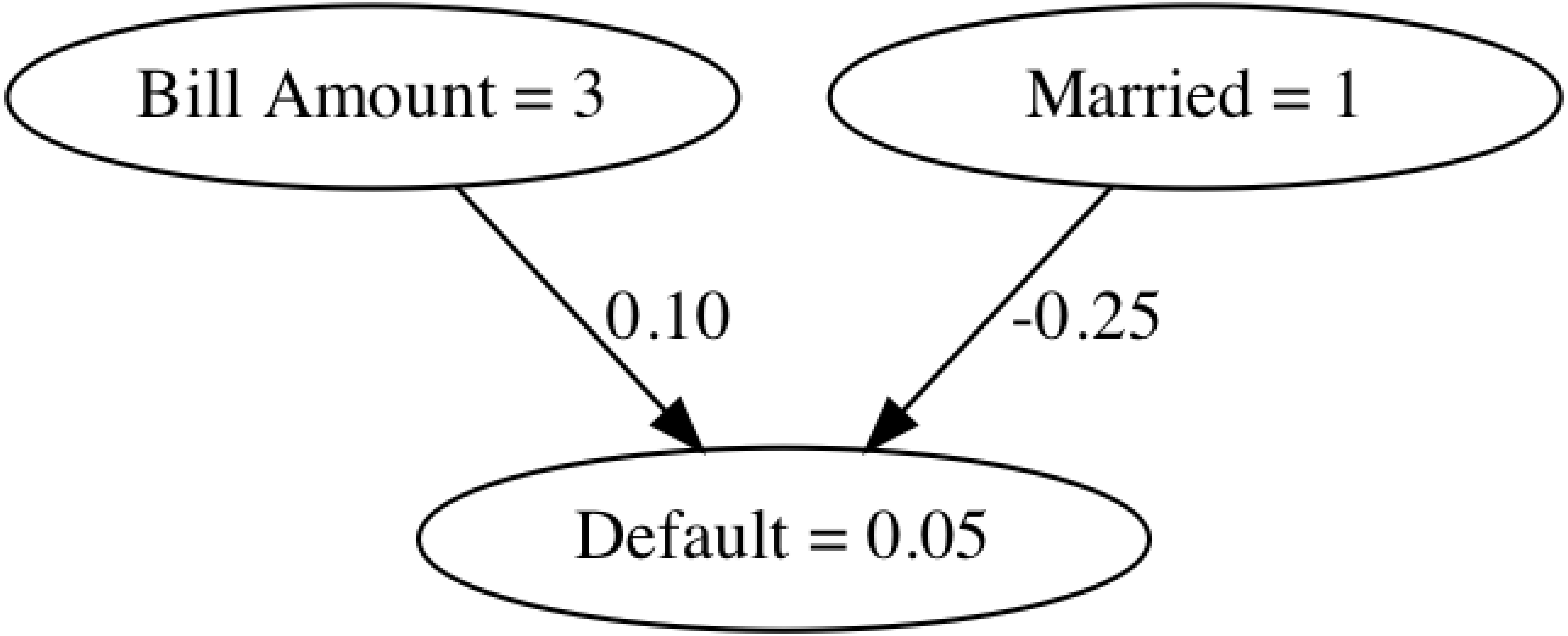
Dense layers

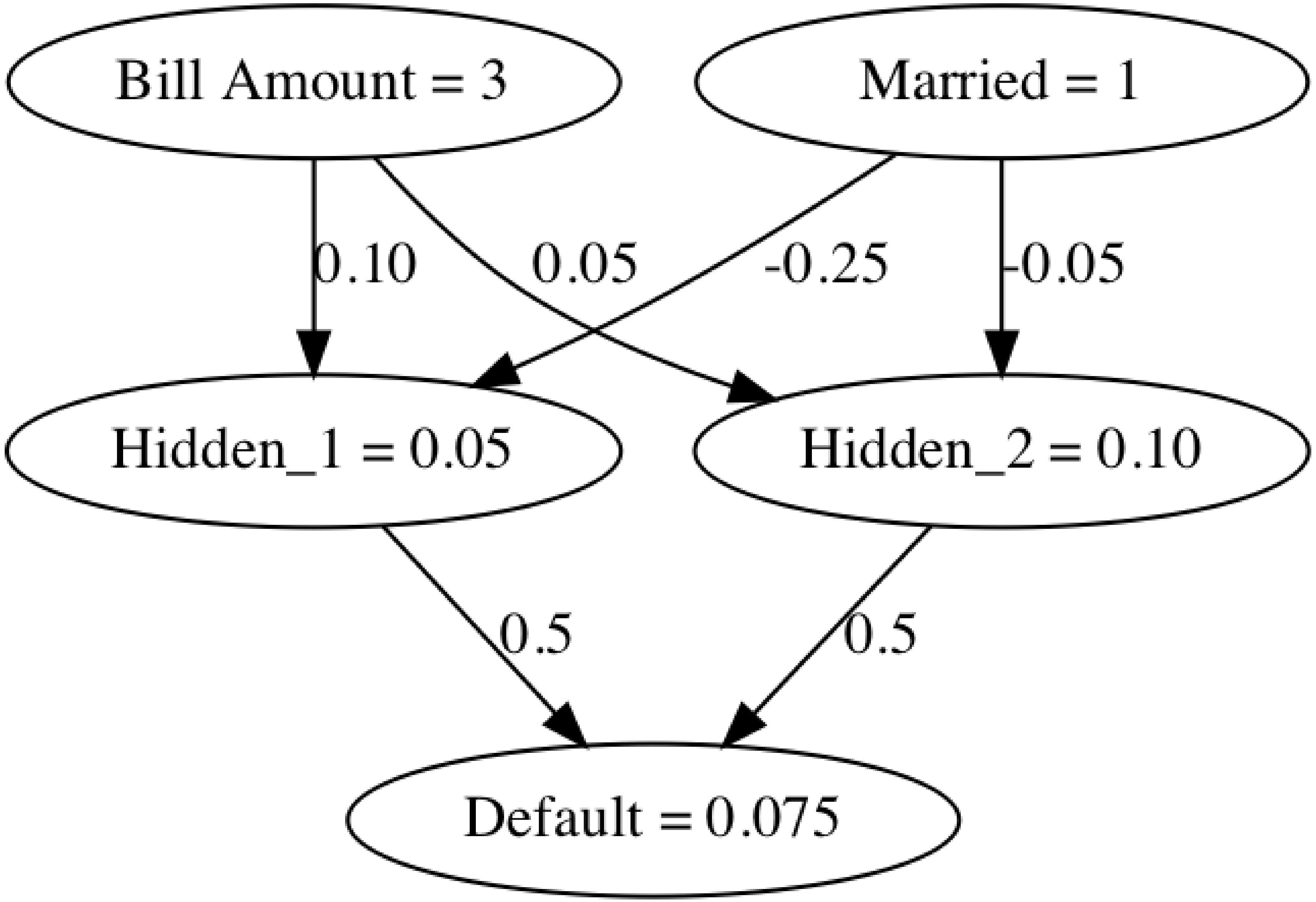
INTRODUCTION TO TENSORFLOW IN PYTHON

Isaiah Hull
Visiting Associate Professor of Finance,
BI Norwegian Business School

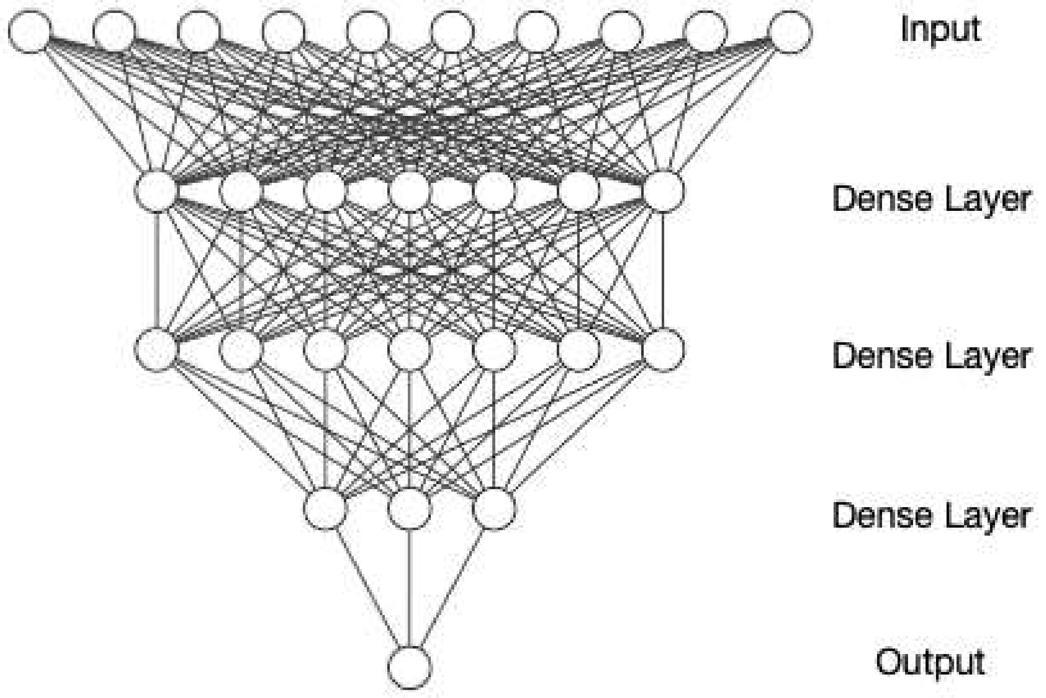
The linear regression model



What is a neural network?



What is a neural network?



- A dense layer applies weights to all nodes from the previous layer.

A simple dense layer

```
import tensorflow as tf
```

```
# Define inputs (features)
inputs = tf.constant([[1, 35]])
```

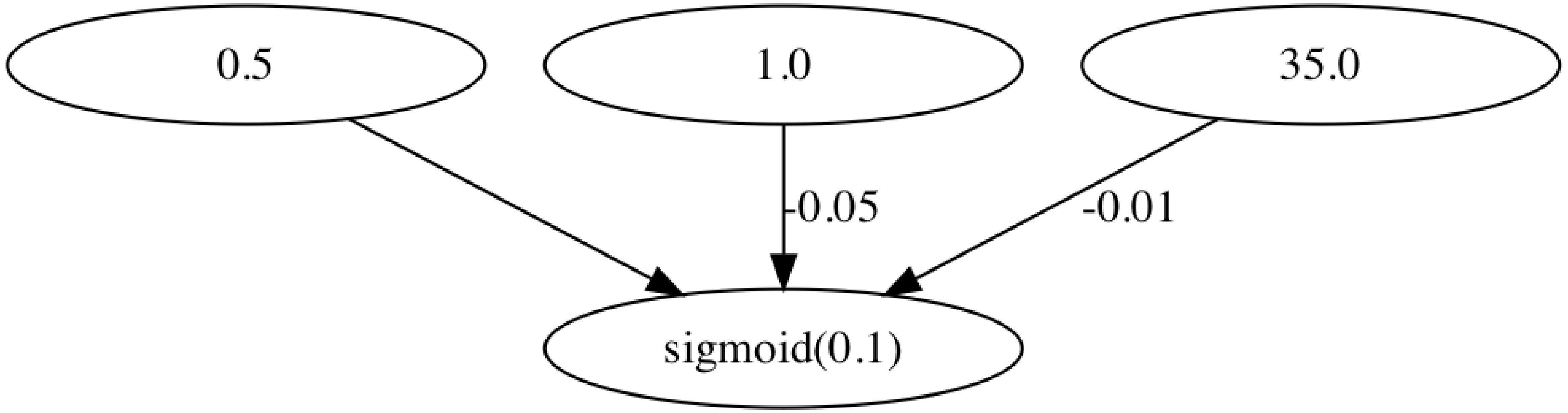
```
# Define weights
weights = tf.Variable([[-0.05], [-0.01]])
```

```
# Define the bias
bias = tf.Variable([0.5])
```

A simple dense layer

```
# Multiply inputs (features) by the weights
product = tf.matmul(inputs, weights)
```

```
# Define dense layer
dense = tf.keras.activations.sigmoid(product+bias)
```



Defining a complete model

```
import tensorflow as tf
```

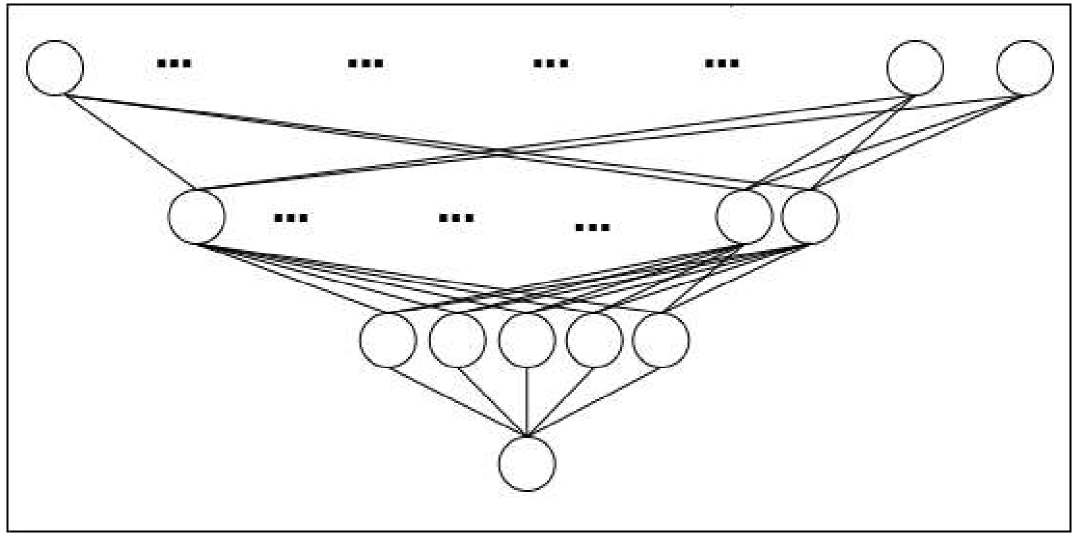
```
# Define input (features) layer
inputs = tf.constant(data, tf.float32)
```

```
# Define first dense layer
dense1 = tf.keras.layers.Dense(10, activation='sigmoid')(inputs)
```

Defining a complete model

```
# Define second dense layer
dense2 = tf.keras.layers.Dense(5, activation='sigmoid')(dense1)
```

```
# Define output (predictions) layer
outputs =  tf.keras.layers.Dense(1, activation='sigmoid')(dense2)
```



High-level versus low-level approach

- **High-level approach**
  - High-level API operations

```
dense = keras.layers.Dense(10,\
 activation='sigmoid')
```

- **Low-level approach**
  - Linear-algebraic operations

```
prod = matmul(inputs, weights)
dense = keras.activations.sigmoid(prod)
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN PYTHON

Activation functions

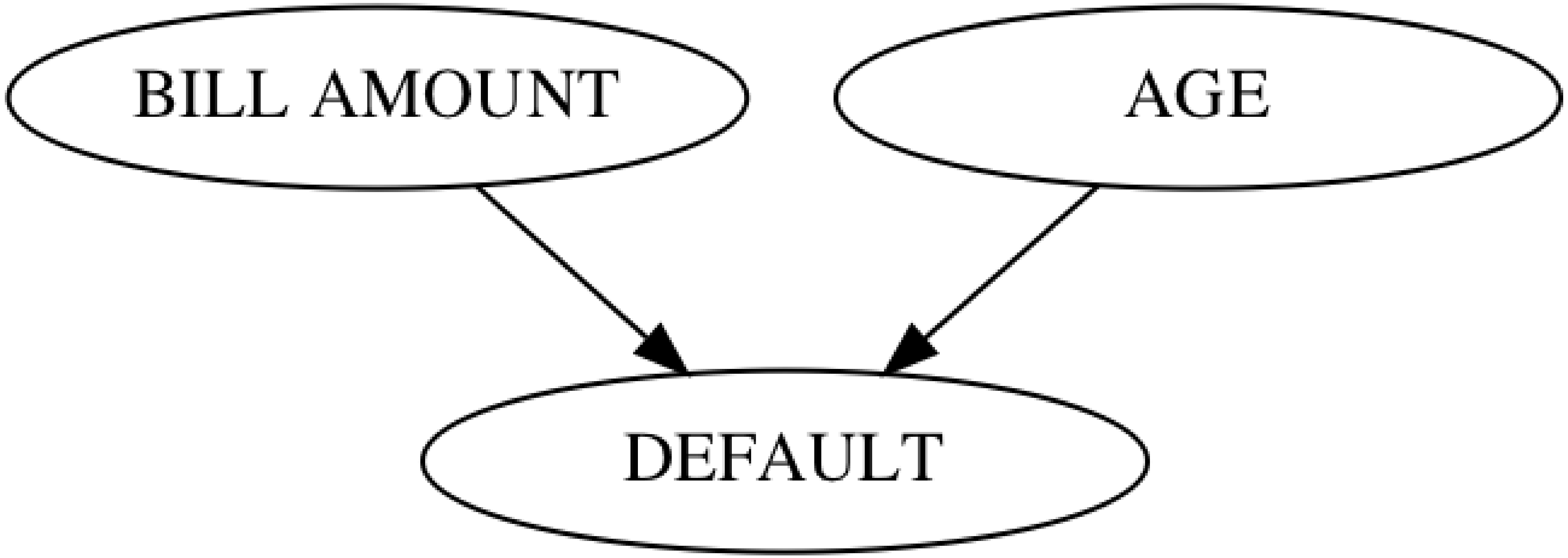
INTRODUCTION TO TENSORFLOW IN PYTHON

Isaiah Hull
Visiting Associate Professor of Finance,
BI Norwegian Business School

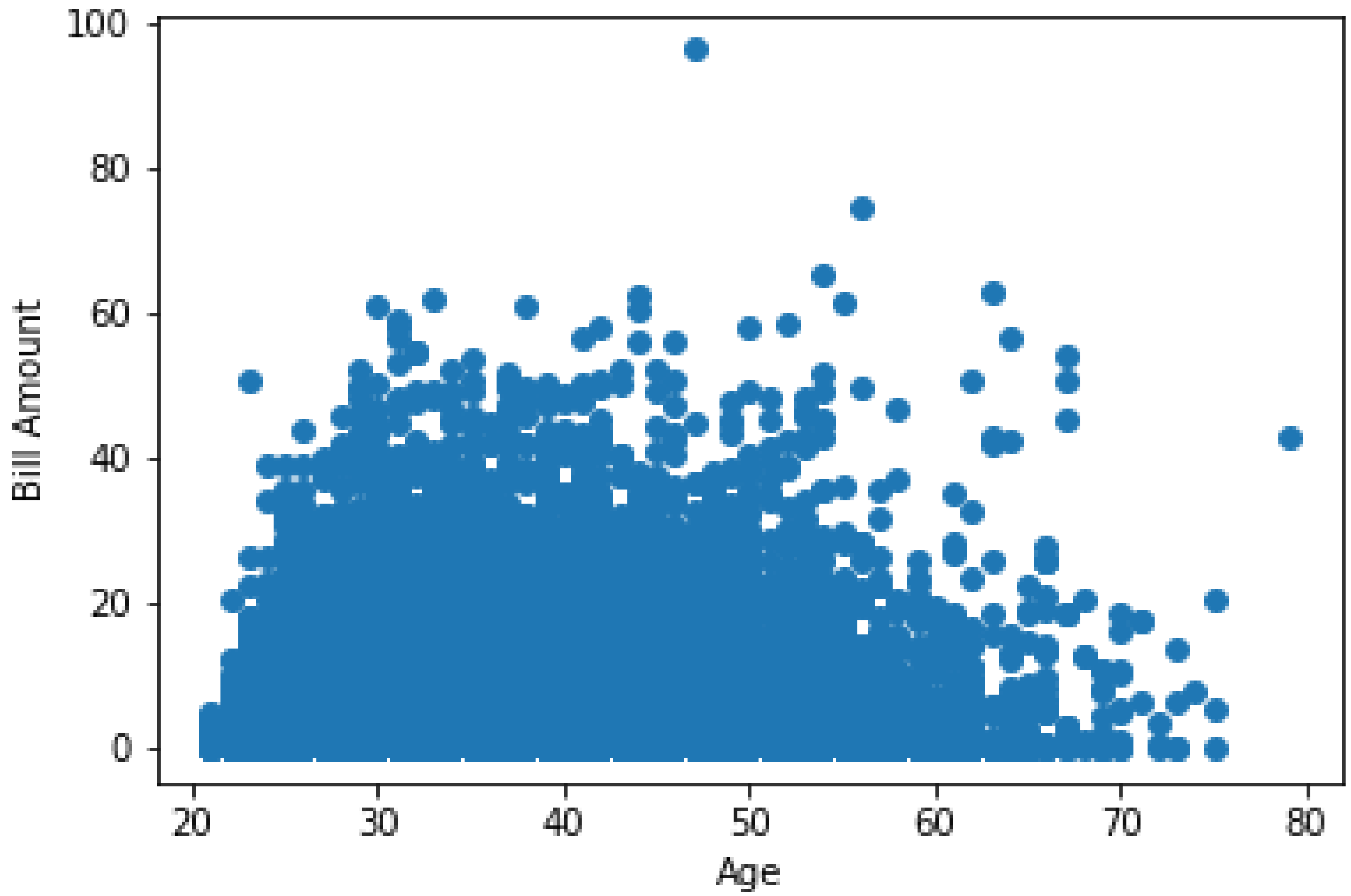
What is an activation function?

- **Components of a typical hidden layer**
  - **Linear:** Matrix multiplication
  - **Nonlinear:** Activation function

Why nonlinearities are important



Why nonlinearities are important



A simple example

```python
import numpy as np
import tensorflow as tf

# Define example borrower features
young, old = 0.3, 0.6
low_bill, high_bill = 0.1, 0.5
```

```python
# Apply matrix multiplication step for all feature combinations
young_high = 1.0*young + 2.0*high_bill
young_low = 1.0*young + 2.0*low_bill
old_high = 1.0*old + 2.0*high_bill
old_low = 1.0*old + 2.0*low_bill
```

A simple example

```
# Difference in default predictions for young
print(young_high - young_low)

# Difference in default predictions for old
print(old_high - old_low)
```

```
0.8
0.8
```

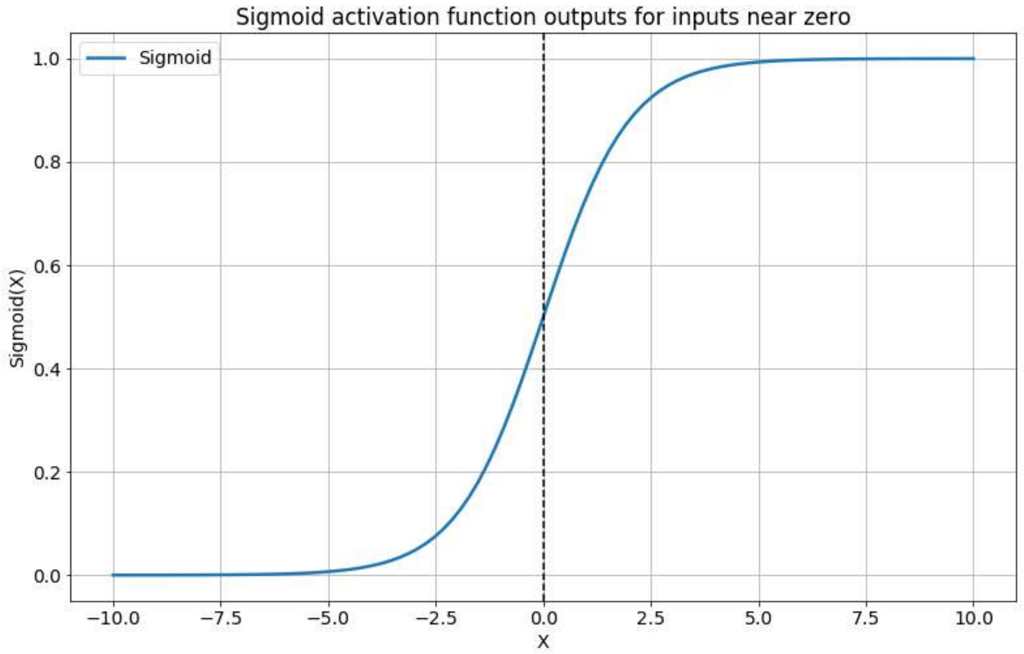
A simple example

```
# Difference in default predictions for young
print(tf.keras.activations.sigmoid(young_high).numpy() -
tf.keras.activations.sigmoid(young_low).numpy())

# Difference in default predictions for old
print(tf.keras.activations.sigmoid(old_high).numpy() -
tf.keras.activations.sigmoid(old_low).numpy())
```

```
0.16337568
0.14204389
```

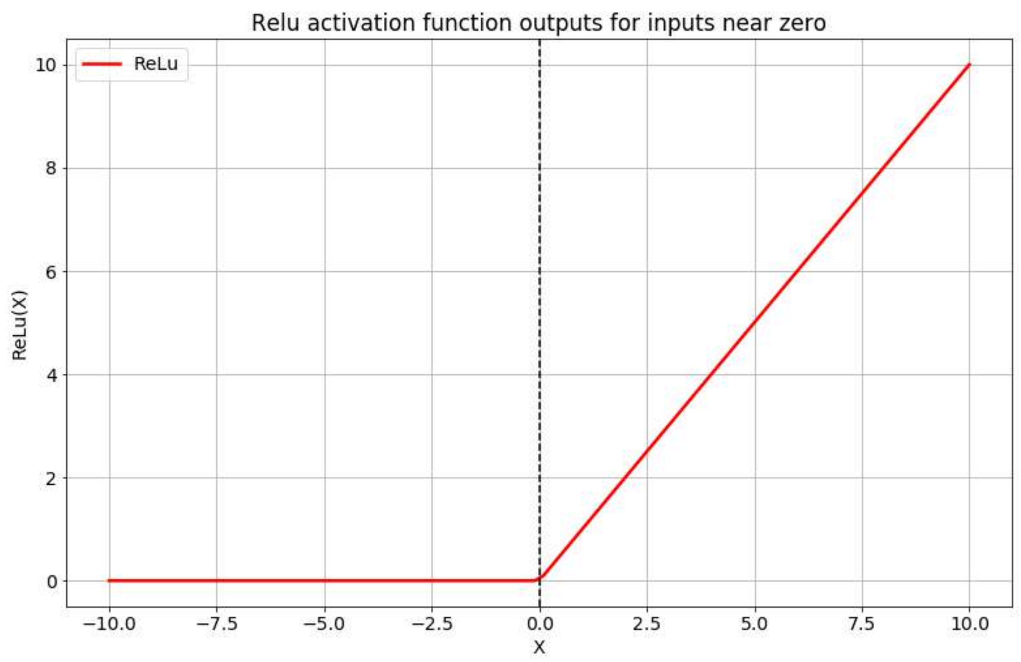
The sigmoid activation function

- **Sigmoid activation function**
  - Binary classification
  - Low-level: `tf.keras.activations.sigmoid()`
  - High-level: `sigmoid`



The relu activation function

- **ReLu activation function**
  - Hidden layers
  - Low-level: `tf.keras.activations.relu()`
  - High-level: `relu`



The softmax activation function

- **Softmax activation function**
  - Output layer (>2 classes)
  - Low-level: `tf.keras.activations.softmax()`
  - High-level: `softmax`

Activation functions in neural networks

```python
import tensorflow as tf
```

```python
# Define input layer
inputs = tf.constant(borrower_features, tf.float32)
```

```python
# Define dense layer 1
dense1 = tf.keras.layers.Dense(16, activation='relu')(inputs)
```

```python
# Define dense layer 2
dense2 = tf.keras.layers.Dense(8, activation='sigmoid')(dense1)
```

```python
# Define output layer
outputs = tf.keras.layers.Dense(4, activation='softmax')(dense2)
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN PYTHON

Optimizers

INTRODUCTION TO TENSORFLOW IN PYTHON

Isaiah Hull
Visiting Associate Professor of Finance,
BI Norwegian Business School

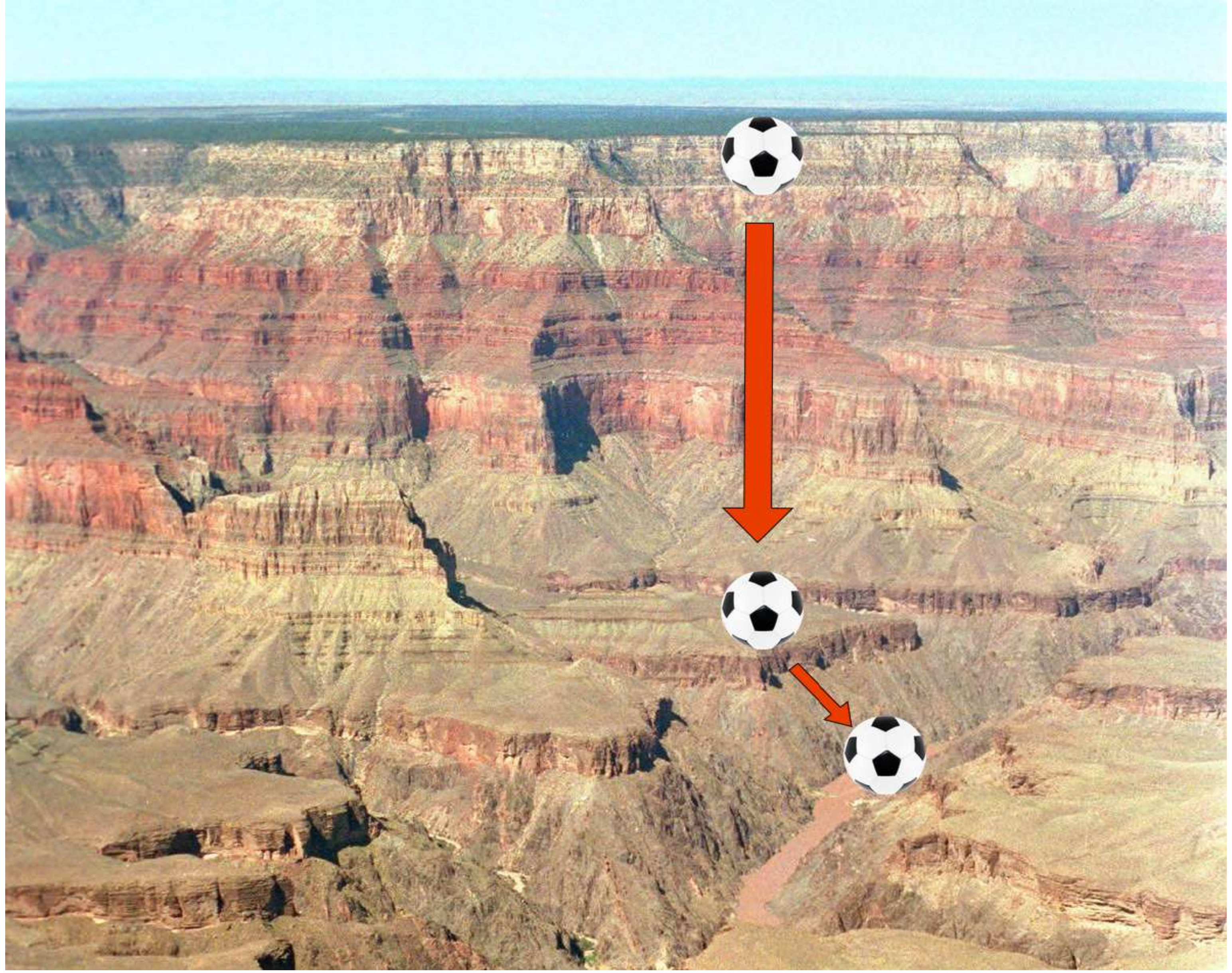
How to find a minimum

[1] Source: U.S. National Park Service

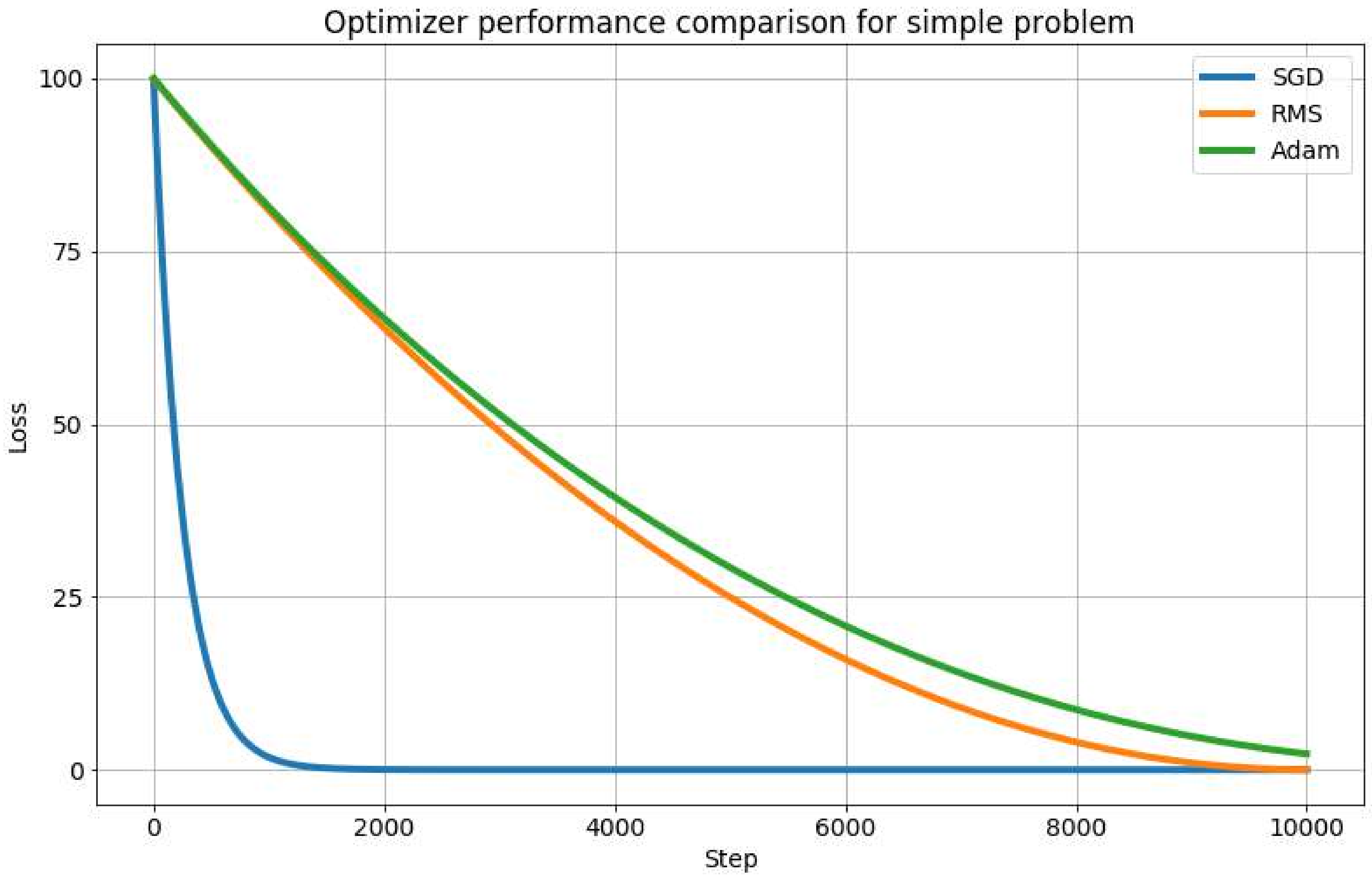
How to find a minimum

[1] Source: U.S. National Park Service

How to find a minimum



[1] Source: U.S. National Park Service


Optimizer performance comparison for simple problem
SGD
RMS
Adam
Loss
100
75
50
25
0
0
2000
4000
6000
8000
10000
Step

The gradient descent optimizer

- Stochastic gradient descent (SGD) optimizer
    - `tf.keras.optimizers.SGD()`
    - `learning_rate`
- Simple and easy to interpret

The RMS prop optimizer

- **Root mean squared (RMS) propagation optimizer**
  - Applies different learning rates to each feature
  - `tf.keras.optimizers.RMSprop()`
  - `learning_rate`
  - `momentum`
  - `decay`
- **Allows for momentum to both build and decay**

The adam optimizer

- **Adaptive moment (adam) optimizer**
  - `tf.keras.optimizers.Adam()`
  - `learning_rate`
  - `beta1`
- **Performs well with default parameter values**

A complete example

```
import tensorflow as tf

# Define the model function
def model(bias, weights, features = borrower_features):
    product = tf.matmul(features, weights)
    return tf.keras.activations.sigmoid(product+bias)
```

```
# Compute the predicted values and loss
def loss_function(bias, weights, targets = default, features = borrower_features):
    predictions = model(bias, weights)
    return tf.keras.losses.binary_crossentropy(targets, predictions)
```

```
# Minimize the loss function with RMS propagation
opt = tf.keras.optimizers.RMSprop(learning_rate=0.01, momentum=0.9)
opt.minimize(lambda: loss_function(bias, weights), var_list=[bias, weights])
```

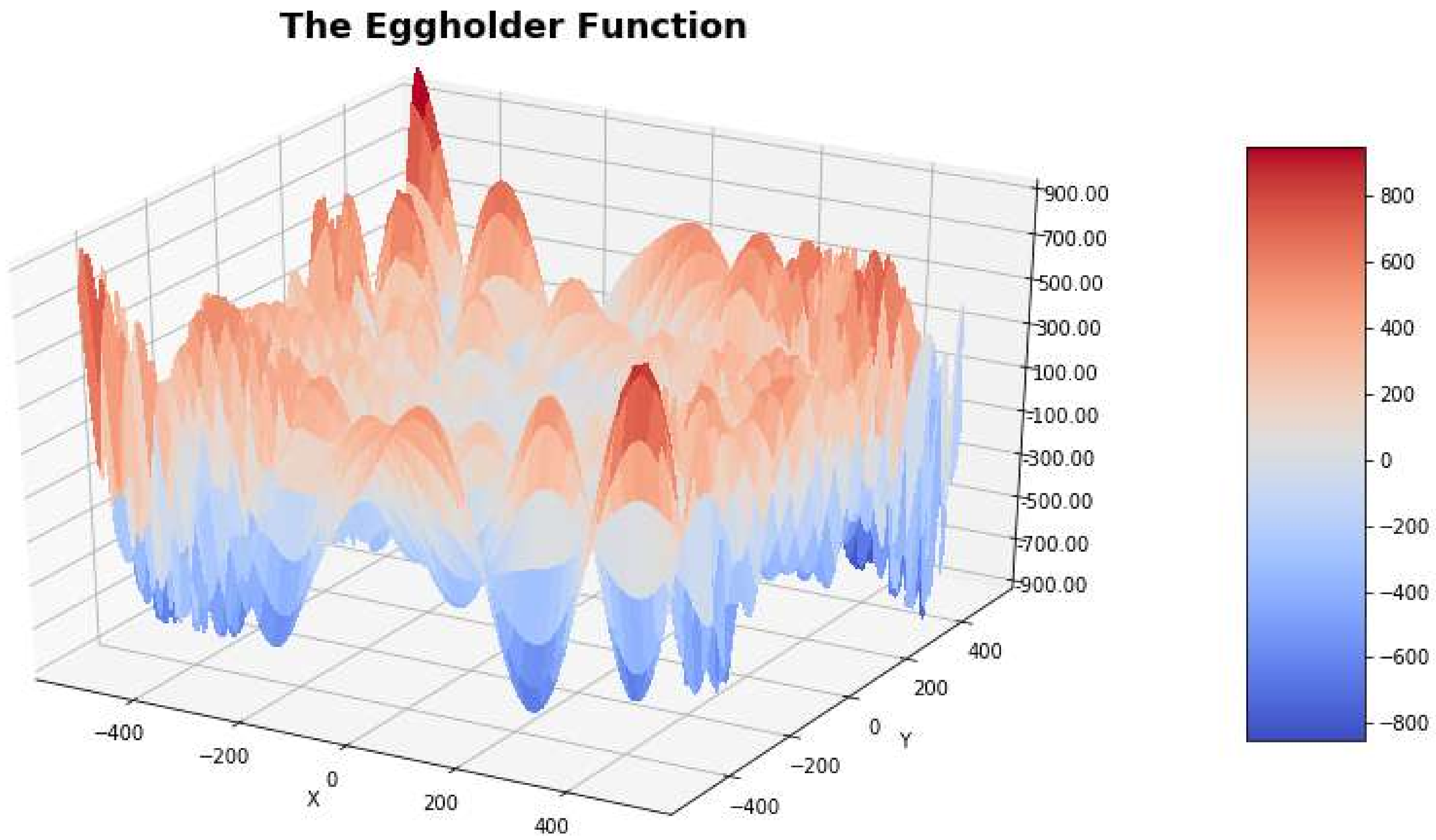
Let's practice!

INTRODUCTION TO TENSORFLOW IN PYTHON

Training a network in TensorFlow

INTRODUCTION TO TENSORFLOW IN PYTHON

Isaiah Hull
Visiting Associate Professor of Finance,
BI Norwegian Business School


The Eggholder Function
900.00
700.00
500.00
300.00
100.00
-100.00
-300.00
-500.00
-700.00
-900.00
−400
−200
0
200
400
X
400
200
0
−200
−400
Y
800
600
400
200
0
−200
−400
−600
−800

Random initializers

- **Often need to initialize thousands of variables**
  - `tf.ones()` may perform poorly
  - Tedious and difficult to initialize variables individually
- **Alternatively, draw initial values from distribution**
  - Normal
  - Uniform
  - Glorot initializer

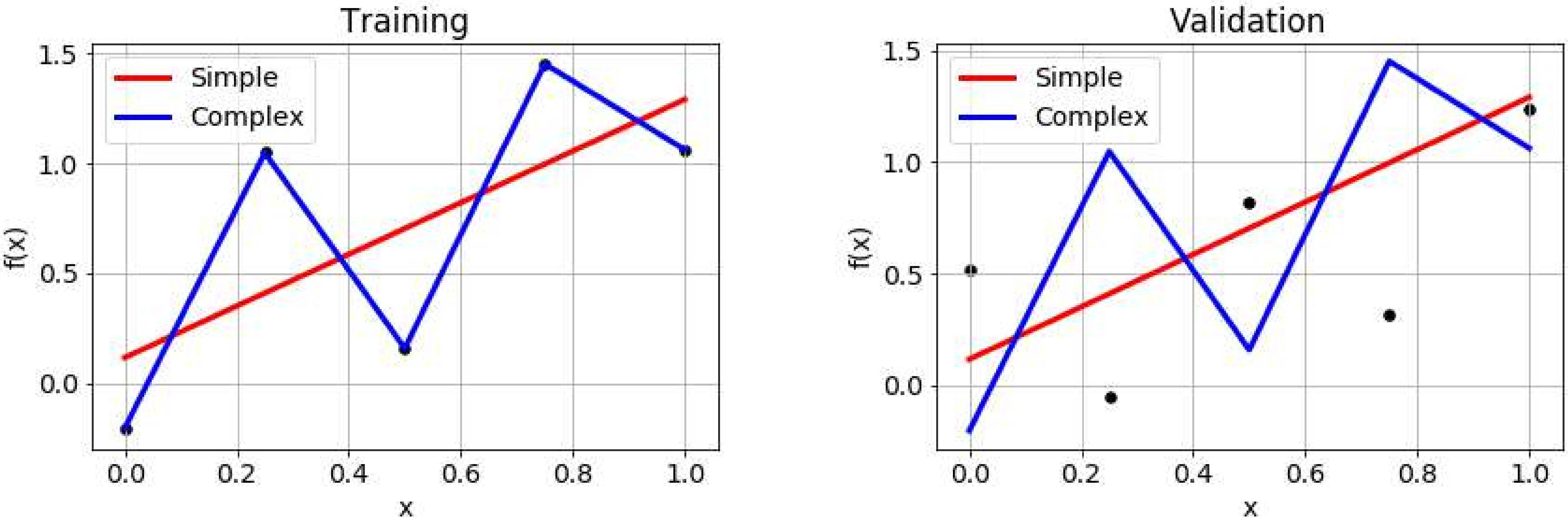
Initializing variables in TensorFlow

```
import tensorflow as tf

# Define 500x500 random normal variable
weights = tf.Variable(tf.random.normal([500, 500]))

# Define 500x500 truncated random normal variable
weights = tf.Variable(tf.random.truncated_normal([500, 500]))
```

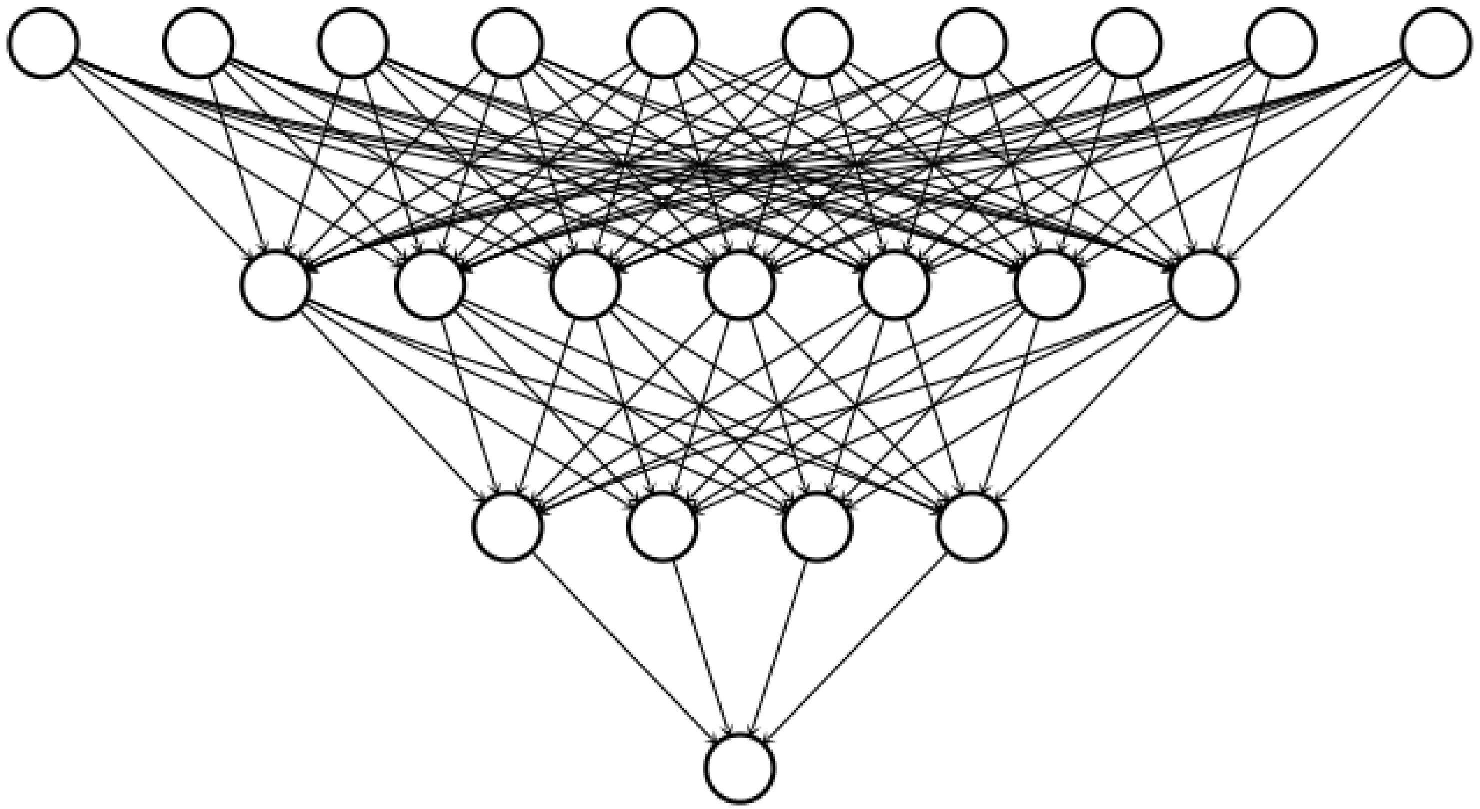
Initializing variables in TensorFlow

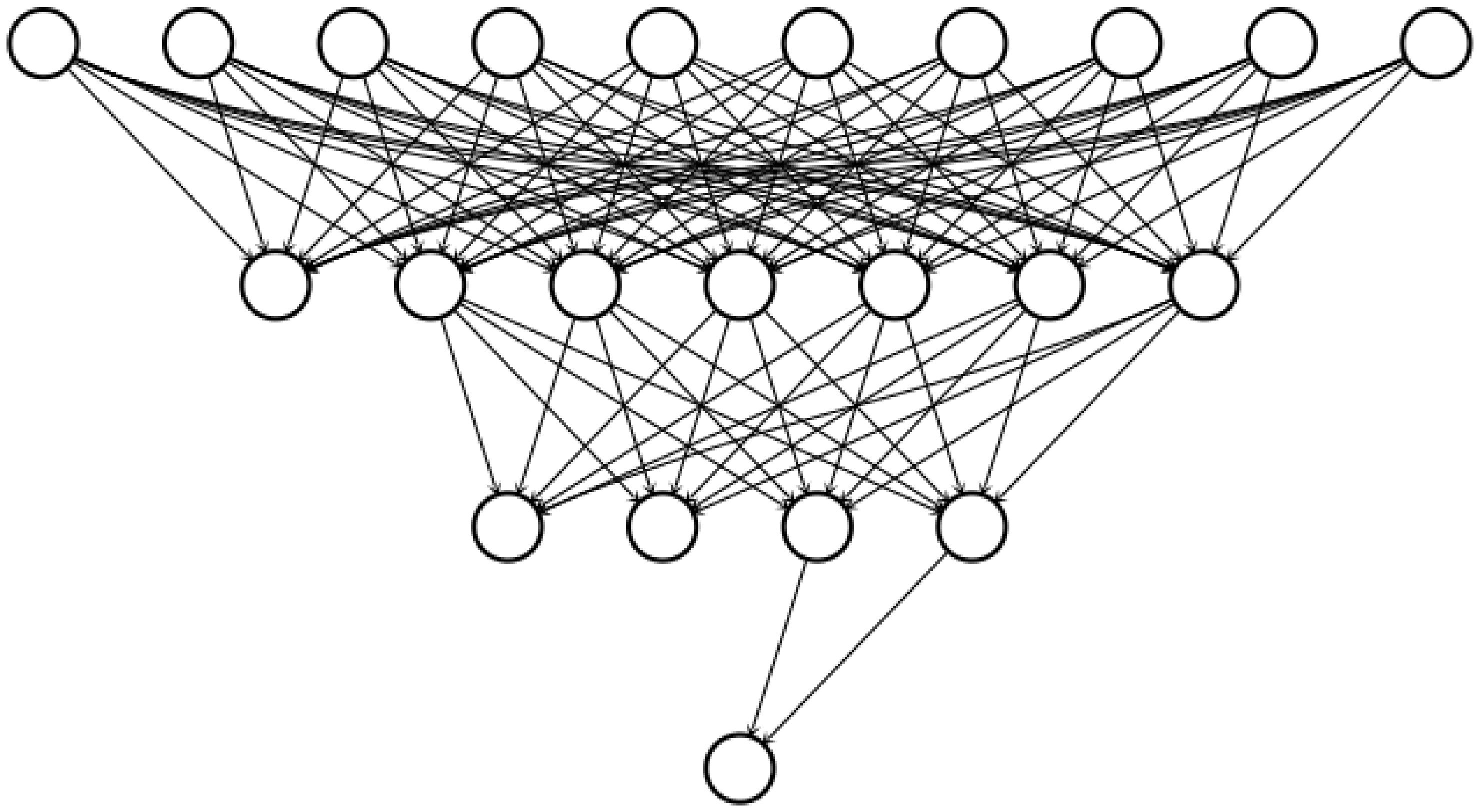
```
# Define a dense layer with the default initializer
dense = tf.keras.layers.Dense(32, activation='relu')

# Define a dense layer with the zeros initializer
dense = tf.keras.layers.Dense(32, activation='relu',\
    kernel_initializer='zeros')
```

Neural networks and overfitting



Applying dropout





Implementing dropout in a network

```
import numpy as np
import tensorflow as tf

# Define input data
inputs = np.array(borrower_features, np.float32)
```

```
# Define dense layer 1
dense1 = tf.keras.layers.Dense(32, activation='relu')(inputs)
```

Implementing dropout in a network

```
# Define dense layer 2
dense2 = tf.keras.layers.Dense(16, activation='relu')(dense1)
```

```
# Apply dropout operation
dropout1 = tf.keras.layers.Dropout(0.25)(dense2)
```

```
# Define output layer
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(dropout1)
```

Let's practice!

INTRODUCTION TO TENSORFLOW IN PYTHON