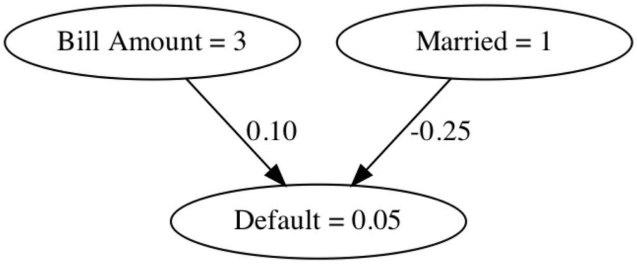
Neural Networks

May 17, 2021

Table of Contents

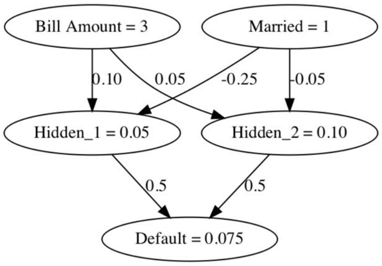
1 Dense layers

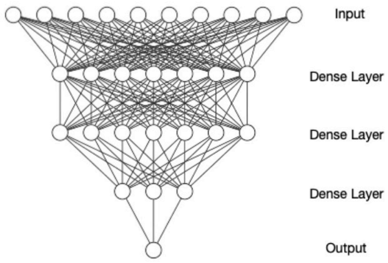
1.1 `[note-1]` The linear regression model



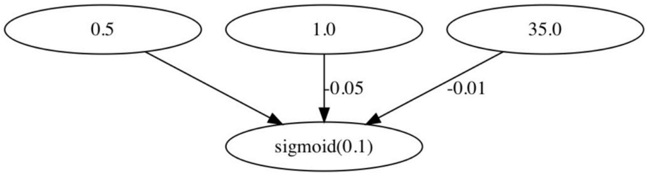
1.2 `[note-2]` What is a neural network?

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





1.3 [code-1] A simple dense layer



[28]:
```python
import tensorflow as tf
```

[29]:
```python
# Define inputs (features)
inputs = tf.constant([[1, 35]], tf.float32)
```

[30]:
```python
# Define weights
weights = tf.Variable([[-0.05], [-0.01]])
```

[31]:
```python
# Define the bias
bias = tf.Variable([0.5])
```

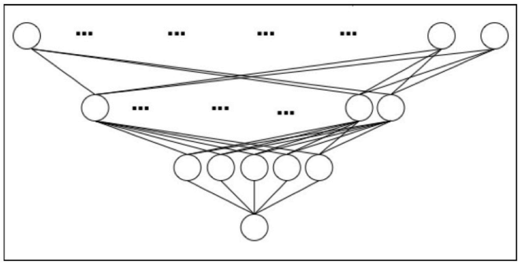
[32]:
```python
# Multiply inputs (features) by the weights
product = tf.matmul(inputs, weights)

product
```

[32]:
```
<tf.Tensor: shape=(1, 1), dtype=float32, numpy=array([[-0.4]], dtype=float32)>
```

[21]:
```python
# Define dense layer
dense = tf.keras.activations.sigmoid(product + bias)

dense
```

[21]:
```
<tf.Tensor: shape=(1, 1), dtype=float32, numpy=array([[0.5249792]],
dtype=float32)>
```

1.4 [code-2] Defining a complete model



[22]:
```
import pandas as pd
import tensorflow as tf

data = pd.read_csv("../Datasets/1. Borrower features.csv", header=None)
```

[24]:
```
# Define input (features) layer
inputs = tf.constant(data, tf.float32)

inputs.shape
```

[24]:
```
TensorShape([100, 10])
```

[25]:
```
# Define first dense layer
dense1 = tf.keras.layers.Dense(10, activation='sigmoid')(inputs)

dense1.shape
```

[25]:
```
TensorShape([100, 10])
```

[26]:
```
# Define second dense layer
dense2 = tf.keras.layers.Dense(5, activation='sigmoid')(dense1)

dense2.shape
```

[26]:
```
TensorShape([100, 5])
```

[27]:
```
# Define output (predictions) layer
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(dense2)

outputs.shape
```

[27]:
```
TensorShape([100, 1])
```

1.5 [note-3] High-level versus low-level approach

- **High-level approach**:

    - high-level API operations
    - example:

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

- **Low-level approach**:
    - linear-algebraic operations
    - example:

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

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```

[ ]:

```
```

[ ]:

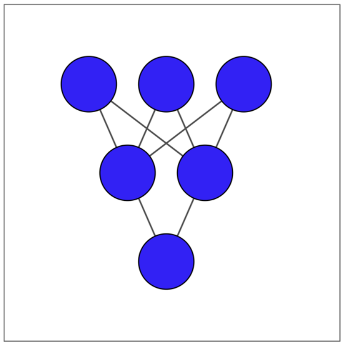
```
```

[ ]:

```
```

1.6 [task-1] The linear algebra of dense layers

▶ **Task diagram**



▶ **Package pre-loading**

[1]:

```
import numpy as np
from tensorflow import Variable, ones, matmul, keras
```

▶ **Data pre-loading**

[2]:

```
borrower_features = np.array([[2., 2., 43.]], dtype=np.float32)
```

▶ Task practice 1/2

[3]:

```
# From previous step
bias1 = Variable(1.0)
weights1 = Variable(ones((3, 2)))
product1 = matmul(borrower_features, weights1)
dense1 = keras.activations.sigmoid(product1 + bias1)

# Initialize bias2 and weights2
bias2 = Variable(1.0)
weights2 = Variable(ones((2, 1)))

# Perform matrix multiplication of dense1 and weights2
product2 = matmul(dense1, weights2)

# Apply activation to product2 + bias2 and print the prediction
prediction = keras.activations.sigmoid(product2 + bias2)
print('\n prediction: {}'.format(prediction.numpy()[0, 0]))
print('\n actual: 1')
```

```

 prediction: 0.9525741338729858

 actual: 1
```

▶ Task practice 2/2

[4]:

```
# From previous step
bias1 = Variable(1.0)
weights1 = Variable(ones((3, 2)))
product1 = matmul(borrower_features, weights1)
dense1 = keras.activations.sigmoid(product1 + bias1)

# Initialize bias2 and weights2
bias2 = Variable(1.0)
weights2 = Variable(ones((2, 1)))

# Perform matrix multiplication of dense1 and weights2
product2 = matmul(dense1, weights2)

# Apply activation to product2 + bias2 and print the prediction
prediction = keras.activations.sigmoid(product2 + bias2)
print('\n prediction: {}'.format(prediction.numpy()[0, 0]))
print('\n actual: 1')
```

```

 prediction: 0.9525741338729858

 actual: 1
```

1.7 [task-2] The low-level approach with multiple examples

▶ **Task instruction**

$$products1 = \begin{bmatrix} 3 & 3 & 23 \\ 2 & 1 & 24 \\ 1 & 1 & 49 \\ 1 & 1 & 49 \\ 2 & 1 & 29 \end{bmatrix} \begin{bmatrix} -0.6 & 0.6 \\ 0.8 & -0.3 \\ -0.09 & -0.08 \end{bmatrix}$$

▶ **Package pre-loading**

[5]:

```
from tensorflow import constant, float32
```

▶ **Data pre-loading**

[6]:

```
borrower_features = constant([[3., 3., 23.], [2., 1., 24.], [1., 1., 49.],
                              [1., 1., 49.], [2., 1., 29.]],
                             dtype=float32)
bias1 = constant([0.1], dtype=float32)
```

▶ **Task practice**

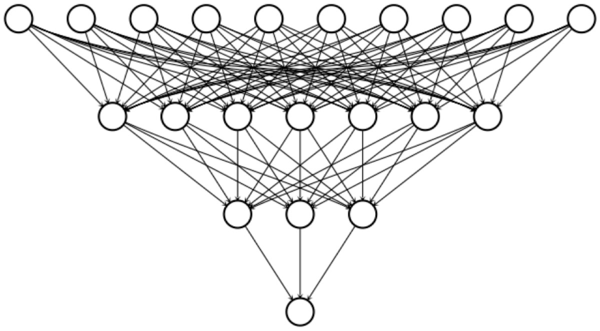
[7]:

```
# Compute the product of borrower_features and weights1
products1 = matmul(borrower_features, weights1)

# Apply a sigmoid activation function to products1 + bias1
dense1 = keras.activations.sigmoid(products1 + bias1)

# Print the shapes of borrower_features, weights1, bias1, and dense1
print('\n shape of borrower_features: ', borrower_features.shape)
print('\n shape of weights1: ', weights1.shape)
print('\n shape of bias1: ', bias1.shape)
print('\n shape of dense1: ', dense1.shape)
```

```

 shape of borrower_features:  (5, 3)

 shape of weights1:  (3, 2)

 shape of bias1:  (1,)

 shape of dense1:  (5, 2)
```

1.8 [task-3] Using the dense layer operation

▶ **Task diagram**



▶ Package pre-loading

[8]:

```
import pandas as pd
```

▶ Data pre-loading

[9]:

```
df = pd.read_csv("../Datasets/1. Borrower features.csv", header=None)
borrower_features = constant(df, dtype=float32)
```

▶ Task practice

[10]:

```
# Define the first dense layer
dense1 = keras.layers.Dense(7, activation='sigmoid')(borrower_features)

# Define a dense layer with 3 output nodes
dense2 = keras.layers.Dense(3, activation='sigmoid')(dense1)

# Define a dense layer with 1 output node
predictions = keras.layers.Dense(1, activation='sigmoid')(dense2)

# Print the shapes of dense1, dense2, and predictions
print('\n shape of dense1: ', dense1.shape)
print('\n shape of dense2: ', dense2.shape)
print('\n shape of predictions: ', predictions.shape)
```

```

 shape of dense1:  (100, 7)

 shape of dense2:  (100, 3)

 shape of predictions:  (100, 1)
```

[34]:

```
import tensorflow
```

[38]:

```
dense = tensorflow.keras.layers.Dense(10, activation='sigmoid')(dense2)
```

[39]:

```
dense.shape
```

[39]:

```
TensorShape([100, 10])
```

[40]:

```
dense2.shape
```

[40]:

```
TensorShape([100, 5])
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

2 Requirements

[ ]:

```
from platform import python_version
import sklearn

python_version = ('python=={}'.format(python_version()))
numpy_version = ('numpy=={}'.format(np.__version__))
pandas_version = ('pandas=={}'.format(pd.__version__))
tensorflow_version = ('tensorflow=={}'.format(tf.__version__))
scikit_learn_version = ('scikit-learn=={}'.format(sklearn.__version__))

writepath = '../../requirements.txt'
requirements = []
packages = [
    numpy_version, pandas_version, tensorflow_version, scikit_learn_version
]

try:
    with open(writepath, 'r+') as file:
        for line in file:
            requirements.append(line.strip('\n'))
except:
    with open(writepath, 'w+') as file:
        for line in file:
            requirements.append(line.strip('\n'))
```

```python
with open(writepath, 'a') as file:
    for package in packages:
        if package not in requirements:
            file.write(package + '\n')

max_characters = len(python_version)
for package in packages:
    if max(max_characters, len(package)) > max_characters:
        max_characters = max(max_characters, len(package))

print('#' * (max_characters + 8))
print('#' * 2 + ' ' * (max_characters + 4) + '#' * 2)
print('#' * 2 + ' ' * 2 + python_version + ' ' *
      (max_characters - len(python_version) + 2) + '#' * 2)
for package in packages:
    print('#' * 2 + ' ' * 2 + package + ' ' *
          (max_characters - len(package) + 2) + '#' * 2)
print('#' * 2 + ' ' * (max_characters + 4) + '#' * 2)
print('#' * (max_characters + 8))
```