Day 21 Artificial Neural Networks

Dec 1, 2020

Announcements

- **Homework 5** Working with Tensorflow. Due this Friday. This is the last homework assignment!
    - Having trouble installing tensorflow? Use Google Colab (just upload your notebook).
- **Projects** Rubric posted to D2L.
    - Due Dec 14th; Review 3 projects by Dec 16th
    - 8-10 minute video presentation + documented notebook on your analysis
    - 3 In-class work periods for the project

Calendar

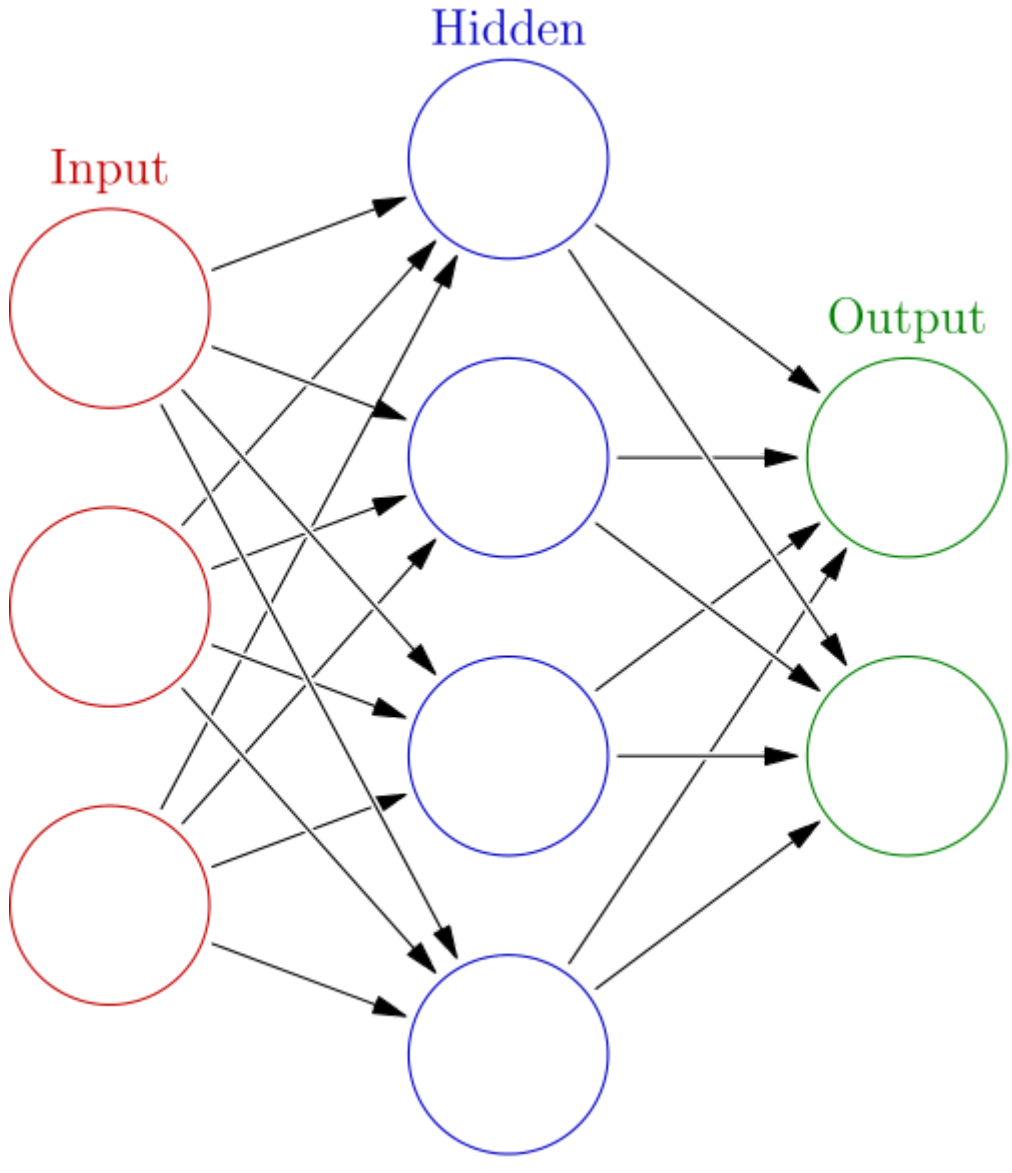
This week

- Tuesday 12/1: Day 20 Neural Networks 1; Project work day 2
- Thursday 12/3: Day 21 Neural Networks 2

Last week of classes

- Tuesday 12/8: Project work day 3
- Thursday 12/10: Project work day 4

Artificial Neural Networks


Hidden
Input
Output

Forward propagation (the basics)

1. Prepare the network
2. Prepare the weight matrices
3. Determine the activity of the hidden layer
4. Apply the activity function to the hidden layer activities
5. Determine the activity of the output layer
6. Apply the activity function to the output layer activities
7. Compare predictions to normalized values

All work this relies on ’dot’ products

We can only multiply matrices together where the columns of the first matrix is the same size of the row of the second project. You get a matrix of size rows of column matrix by columns of the second matrix.

For example, multiplying a M by N martix by a N by P matrix gives a M by P matrix.

The most common 'dot' product you are likely to see is a 1 by N multiplied by an N by 1, which gives a 1 by 1 matrix: a scalar.

$$a \cdot b = [321] \cdot \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix} = (3)(1) + (2)(4) + (1)(2) = 3 + 8 + 2 = 13$$

Prepare the network

- Store the input data and the output data
- Normalize (standardize) those data
- Decide on network properties
    - Input layers: how many features do you have?
    - Output layers: what are you predicting?
    - Hidden layers: You choose

In [2]:

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

# input data (hours of sleep, hours of study)
X = np.array([[3,5], [5,1], [10,2]])

# normalized X
X_norm = X / np.max(X)
print("X_norm =")
print(X_norm)

# output data (test score)
y = np.array([[75], [83], [93]])

# normalized y
y_norm = y/100
print("y_norm =")
print(y_norm)

# Neural Network properties
inputLayerSize = 2
outputLayerSize = 1
hiddenLayerSize = 3

print(X_norm.shape)
print(y_norm.shape)
```

```
X_norm =
[[0.3 0.5]
 [0.5 0.1]
 [1.  0.2]]
y_norm =
[[0.75]
 [0.83]
 [0.93]]
```

```
(3, 2)
(3, 1)
```

Prepare the weight matrices

For our case, we are using one hidden layer, so we have two "simple" weight matrices.

- **W1** is the matrix of weights connecting inputs to the hidden layer, so it must be input layers (2) by hidden layers (3) in size (i.e, 2 by 3)
- **W2** is the matrix of weights connecting hidden layer to the output layer, so it must be hidden layers (3) by output layers (1) in size (i.e, 3 by 1)

For this example, we simply use random numbers drawn from a Gaussian (normal) distribution.

In [3]:

```
# Define synapse weights
W1 = np.random.randn(inputLayerSize, hiddenLayerSize)
W2 = np.random.randn(hiddenLayerSize, outputLayerSize)
print("W1\n---")
print(W1)
print('shape:', W1.shape)
print('\n')
print("W2\n---")
print(W2)
print('shape:', W2.shape)
```

```
W1
---
[[ 0.77318236 -0.8790112  -2.43486401]
 [ 1.6126631  -2.24382628  0.85529834]]
shape: (2, 3)


W2
---
[[ 1.58046393]
 [ 0.55830599]
 [-0.22941993]]
shape: (3, 1)
```

Determine the activity of the hidden layer

To compute the activity of each neuron in hidden layer ($Z^{(2)}$), we take the the product of the data (X) with the weight matrix that connects the input layer to the hidden layer ($W^{(1)}$),

$$X \cdot W^{(1)} = Z^{(2)}$$

In our example X is 3 by 2 and $W^{(1)}$ is 2 by 3, so the resulting product, $Z^{(2)}$ is 3 by 3.

In [5]:

```
# Apply the first weights to inputs to get hidden layer activity
Z2 = np.dot(X_norm, W1)
print("Z2 =")
print(Z2)
print(Z2.shape)
```

```
Z2 =
[[ 1.03828626 -1.3856165  -0.30281003]
 [ 0.54785749 -0.66388823 -1.13190217]
 [ 1.09571498 -1.32777645 -2.26380434]]
(3, 3)
```

Apply the activation function to the activity

Now that we have the activity, we need to apply our activation function to see if the neuron "fires." We choose a sigmoid (like logisitic regression) for the activation function (we are free to choose a variety of possible activations).

- We first create a sigmoid function $f(z) = \frac{1}{1 + e^{-z}}$
- Then, we compute $a^{(2)}$, which is the result of passing the activity values through the activation function

$$a^{(2)} = f(z^{(2)})$$

Where f is the activation function of our choosing. We should get back a 3 by 3 matrix in this example.

In [6]:

```
# Define our activation function
def sigmoid(z):
    #Apply sigmoid activation function to scalar, vector, or matrix
    return 1/(1+np.exp(-z))

# Apply the activation function to our activity levels
a2 = sigmoid(Z2)
print("a2 =")
print(a2)
print(a2.shape)
```

```
a2 =
[[0.7385192  0.20010848 0.42487069]
 [0.63363837 0.33986672 0.24381023]
 [0.74945636 0.2095274  0.09416536]]
(3, 3)
```

Determine the activity of the output layer

Now we use the weights that connect the hidden layer to the output layer ($W^{(2)}$) to determine the activity of the output layer ($z^{(3)}$). We do that by taking the product of $a^{(2)}$ and the weights.

$$z^{(3)} = a^{(2)} \cdot W^{(2)}$$

In this example $a^{(2)}$ is 3 by 3 and $W^{(2)}$ is 3 by 1, the resulting activity, $z^{(3)}$, is 3 by 1.

In [9]:

```
# Apply the second weights to the hidden layer
Z3 = np.dot(a2, W2)
print("Z3 =")
print(Z3)
print('shape: ', Z3.shape)
```

```
Z3 =
[[1.18145092]
 [1.13525728]
 [1.27986574]]
shape:  (3, 1)
```

Apply the activation function to the output layer activity

Now that we have the output layer activity, we need to again apply our activation function to see if the neuron "fires."

- We use the sigmoid function we created
- Then, we compute $\hat{y}$, which is estimate of the normalized y values by passing the activity ($z^{(3)}$) through the activation function

$$\hat{y} = f(z^{(3)})$$

Where f is the activation function of our choosing. We should get back a 3 by 1 matrix in this example.

In [10]:

```
# Apply the activation function to produce prediction for yHat
yHat = sigmoid(Z3)
print('estimates:\n', yHat)
```

```
estimates:
 [[0.76520858]
 [0.75680781]
 [0.78242692]]
```

Compare to the real data (supervised learning)

Now we can simply compare them.

In [11]:

```
# Compare yhat to known y_norm
print("--------------")
print("How did we do?")
print("y_norm:")
print(y_norm)
print("yHat:")
print(yHat)
```

```
--------------
How did we do?
y_norm:
[[0.75]
 [0.83]
 [0.93]]
yHat:
[[0.76520858]
 [0.75680781]
 [0.78242692]]
```

Today’s Class

- We will put you in your project groups
- Today's in-class notebook is an update for us on your progress
- We will drop in to discuss your project with you
- **On-track goal:** start modeling data, discuss issues with models

Questions, Comments, Concerns?