An overview of artificial neural networks (ANNs)

Nathan Adam

July 31, 2021

1 Biological inspiration

Neural networks are loosely inspired by the human brain, which is a network of approximately 86 billion interconnected neurons (nerve cells).

Presynaptic neurons fire neurotransmitters (signalling molecules) from their axons (nerve fibers) as a result of action potentials (electrical impulses).

Postsynaptic neurons' dendrites receive the neurotransmitters after they cross the synapses (gaps between neurons), and experience action potentials if the neurotransmitters are strong enough [1].

A neuron can have up to 15,000 connections to other neurons via synapses. Thought is facilitated by electrical impulses propagating like a wave through a myriad of neurons.

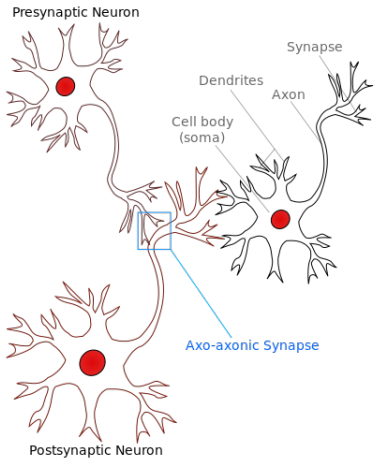

Figure 1: Interconnected neurons [2]

2 What are artificial neural networks (ANNs) and how do they work?

Computers are unable to interpret real-world scenarios in the same way that humans can. The first ANN, Perceptron, was created in 1958 by a psychologist, Frank Rosenblatt, in an attempt to resolve this limitation [3]. Artificial neural networks mimic biological neural networks and can be used for classification, regression and clustering. They enable automation of various processes.

ANNs are composed of an input layer, hidden layer(s) and an output layer. The input layer ingests the relevant features so it has a number of neurons equal to the number of features in the relevant data. The hidden layers are responsible for providing meaningful information for the output layer [4]. There are various ways of deciding upon the number of neurons to use in the hidden layer(s), but for a simple three-layer neural network, one method is to have $\sqrt{nm}$ neurons in the hidden layer where n is the number of input features and m is the number of output neurons [5]. It's also possible to have more than one hidden layer, but for most applications 1-2 layers works well. The output layer produces predictions and should have a number of neurons equal to the number of classes minus one.

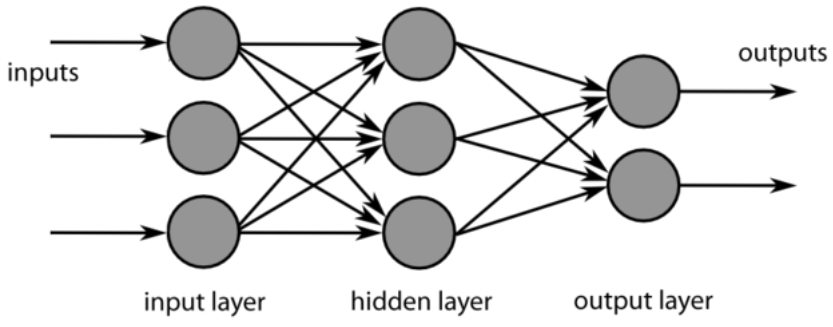

Figure 2: An example of a simple neural network [6]

2.1 Forward propagation

Forward propagation is the process of calculating and storing intermediate variables and creating outputs for a neural network, from the input layer to the output layer. Below, we provide an overview of an iteration of this process from start to finish.

1) A single row of features are ingested by the input layer.

2) A set of weights (which represent the strengths of connections between neurons) is randomly assigned to each neuron (one weight for each neuron in the next layer).

3) The summation function computes the weighted sum of the values for each neuron in the next layer (e.g., input layer to hidden layer, hidden layer to output layer).

4) A bias term (similar to an intercept term) is added to each weighted sum, which shifts the activation function and helps the model fit the data better.

5) Each weighted sum added to the bias term is passed to the activation function, which adds nonlinearity to the network. Nonlinearity is useful because many real-world machine learing problems don't have linearly separable classes[7].

Steps 3-5 are repeated until the network produces an output.

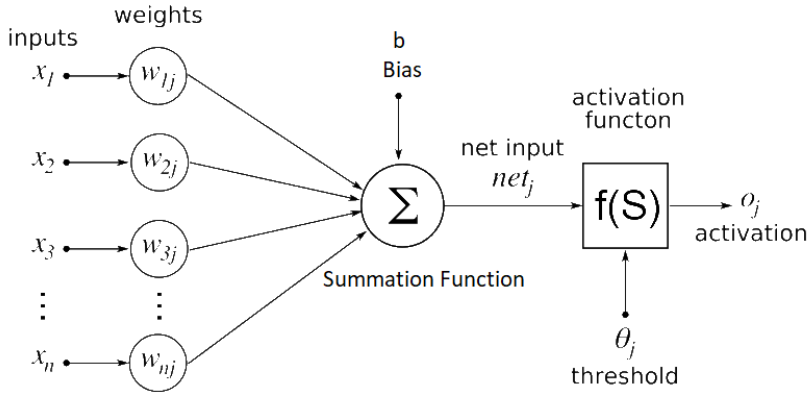

Figure 3: Overview of how data is processed from each neuron in one layer to one neuron in the next layer [8], modified from original

2.2 Backpropagation

Backpropagation refers to the calculation of the gradients of the neural network's parameters, and is somewhat mathematically involved. This entails traversing the network in reverse order by using the chain rule from calculus. Each iteration of backpropagation uses information from each iteration of forward propagation. The end result is that the weights are updated such that the loss function (which represents the difference between predicted and actual values) is minimised [9], thus improving the accuracy of the neural network's predictions.

2.3 Training

Forward and backward propagation work interdependently when training the model, and one instance of each process is performed for each row in the training data. Once the model has been successfully trained, only forward propagation is required to make predictions.

3 Implementing a simple neural network in python

We will implement a simple neural network in python and use the heart dataset to attempt to predict whether or not a person has heart disease.

To begin, let's import the pandas package and use pandas to read the data in. Some categorical features in the dataset are in text format, so we create dummy variables (i.e., we represent whether each value exists with 1s and 0s). We also split the dataset into two arrays: one which contains the features, and one which contains the target variable (heart disease):

```
import pandas as pd

df=pd.read_csv(r'heart.csv')

df=pd.get_dummies(df,columns=['cp','slope','thal']
,drop_first=True)

X=df.loc[:,df.columns!="heart_disease"]
y=df.loc[:,df.columns=="heart_disease"].values.ravel()
```

Next, we split the the two arrays into training and test data, and scale the data so that the neural network can learn from the data effectively:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test=train_test_split
(X_original,y,test_size=0.20,random_state=123)
```

```
ss=StandardScaler()
ss.fit((X_train.astype(float)))
X_train=ss.transform(X_train.astype(float))
X_test=ss.transform(X_test.astype(float))
```

Now we can begin to work on the neural network. We import 'Sequential' and 'Dense' from the Keras package. Sequential lets us specify that we are working with a linear stack of layers. Dense lets us define the number of neurons in each layer and lets us choose activation functions. Dense also enables the computation of weighted sum + bias passed to the activation function (we choose the popular relu and sigmoid functions here). The number of neurons in the single hidden layer is equal to the number of features in the training data (18). The number of outputs is the number of classes minus one ($2 - 1 = 1$). We will use $\sqrt{18 \cdot 1} \approx 4$ neurons in the hidden layer:

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(4, input_dim=X_train.shape[1],
activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

We proceed by defining some additional properties of our model. Since we are performing binary classification, we use binary cross entropy as our loss function, and we will use the popular adam optimizer. The accuracy metric is useful and easy to understand so we will use this in our example:

```
model.compile(loss='binary_crossentropy',
 optimizer='adam', metrics=['accuracy'])
```

Now we can begin to train the model. An epoch is a collective instance of forward propagation and backpropagation. Batch size is the number of samples to work through in each epoch before the weights are updated. These values can be determined by trial and error. In this example we use 100 epochs and a batch size of 32:

```
model.fit(X_train, y_train, epochs=100, batch_size=32)
```

Finally, we can evaluate the model on the training data, make predictions on the test data, and evaluate the performance on the test data. Sklearn.metrics will enable us to evaluate the accuracy scores. The predictions are rounded to the nearest whole number with numpy:

```
import numpy as np
from sklearn.metrics import (accuracy_score)

_, accuracy = model.evaluate(X_train, y_train)

y_pred=model.predict(X_test)
acc_score_test=accuracy_score(y_test,y_pred)
```

Since the weights are initialised randomly, the accuracies will differ each time the code is run. These are the accuracy pairs (training accuracy, test accuracy) for 3 runs:
(0.888,0.738), (0.864, 0.820), (0.860, 0.803). In reality, a lot of feature engineering, parameter tuning and copious amounts of data are required to unlock the full potential of a neural network, but hopefully this example helped you gain some insight into what is involved.

4 Article Summary

In summary, we have discussed:

- The biological motivation behind neural networks
- The structure of neural networks
- Forward propagation
- Backpropagation
- Training neural networks and making predictions
- Example code for a simple neural network, including model evaluation

This was just the tip of the iceberg, but hopefully this article helped you gain some understanding of neural networks!

References

[1] School of Biomedical Sciences Wiki. Presynaptic and postsynaptic neurons. Available at https://teaching.ncl.ac.uk/bms/wiki/index.php/Presynaptic_and_postsynaptic_neurons.

[2] Wikimedia Commons. Axo-axonic synapse.svg. Available at https://commons.wikimedia.org/wiki/File:Axo-axonic_synapse.svg.

[3] Towards Data Science. Neural networks: All you need to know. Available at https://towardsdatascience.com/nns-aynk-c34efe37f15a.

[4] Manning Free Content. Neural network architectures. Available at https://freecontent.manning.com/neural-network-architectures/.

[5] prashanth (https://stats.stackexchange.com/users/86202/prashanth). How to choose the number of hidden layers and nodes in a feedforward neural network? Cross Validated. Available at https://stats.stackexchange.com/q/196854.

[6] Wikimedia Commonsi. File:multilayerneuralnetworkbigger english.png. Available at https://commons.wikimedia.org/wiki/File:MultiLayerNeuralNetworkBigger_english.png.

[7] Manning Free Content. Neural network architectures. Available at https://d2l.ai/chapter_multilayer-perceptrons/backprop.html.

[8] Wikimedia Commonsi. File:artificial neural network.png. Available at https://commons.wikimedia.org/wiki/File:Artificial_neural_network.png.

[9] Simeon Kostadinov. Understanding backpropagation algorithm. Towards data Science. Available at https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.