Building first neural network

Tushar B. Kute,
http://tusharkute.com

Basic Steps

- 1. Load Data.
- 2. Define Model.
- 3. Compile Model.
- 4. Fit Model.
- 5. Evaluate Model.
- 6. Tie It All Together.

Load dataset: Pima Indian Diabetes

- We are going to use the Pima Indians onset of diabetes dataset.
- This is a standard machine learning dataset available for free download from the UCI Machine Learning repository.
- It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years.
- It is a binary classification problem (onset of diabetes as 1 or not as 0). The input variables that describe each patient are numerical and have varying scales.

Load dataset: Pima Indian Diabetes

- Below lists the eight attributes for the dataset:

  1. Number of times pregnant.
  2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
  3. Diastolic blood pressure (mm Hg).
  4. Triceps skin fold thickness (mm).
  5. 2-Hour serum insulin (mu U/ml).
  6. Body mass index.
  7. Diabetes pedigree function.
  8. Age (years).
  9. Class, onset of diabetes within five years.

Load the data

- Whenever we work with machine learning algorithms that use a stochastic process (e.g. random numbers), it is a good idea to initialize the random number generator with a fixed seed value.
- This is so that you can run the same code again and again and get the same result.
- This is useful if you need to demonstrate a result, compare algorithms using the same source of randomness or to debug a part of your code.

Load the data

- Now we can load our Pima Indians dataset. You can now load the file directly using the NumPy function loadtxt().
- There are eight input variables and one output variable (the last column).
- Once loaded we can split the dataset into input variables (X) and the output class variable (Y).

Define Model

- Models in Keras are defined as a sequence of layers.
- We create a Sequential model and add layers one at a time until we are happy with our network topology.
- The first thing to get right is to ensure the input layer has the right number of inputs.
- This can be specified when creating the first layer with the input dim argument and setting it to 8 for the 8 input variables.

Define Model

- How do we know the number of layers to use and their types? This is a very hard question.
- There are heuristics that we can use and often the best network structure is found through a process of trial and error experimentation.
- Generally, you need a network large enough to capture the structure of the problem if that helps at all.
- Here, we will use a fully-connected network structure with three layers.

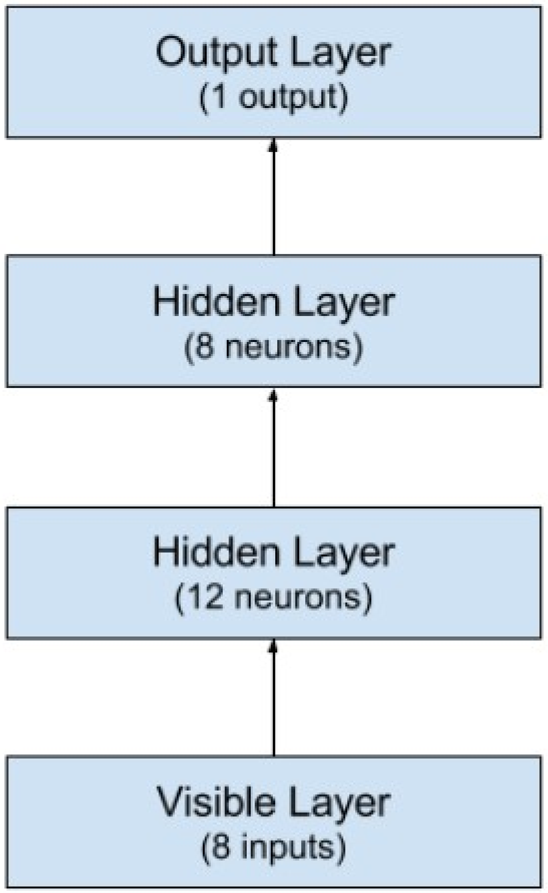
Define Model

- Fully connected layers are defined using the Dense class. We can specify the number of neurons in the layer as the first argument, the initialization method as the second argument as init and specify the activation function using the activation argument.
- In this case we initialize the network weights to a small random number generated from a uniform distribution (uniform), in this case between 0 and 0.05 because that is the default uniform weight initialization in Keras.
- Another traditional alternative would be normal for small random numbers generated from a Gaussian distribution.

Define Model

- We will use the rectifier (relu) activation function on the first two layers and the sigmoid activation function in the output layer. It used to be the case that sigmoid and tanh activation functions were preferred for all layers.
- These days, better performance is seen using the rectifier activation function.
- We use a sigmoid activation function on the output layer to ensure our network output is between 0 and 1 and easy to map to either a probability of class 1 or snap to a hard classification of either class with a default threshold of 0.5. We can piece it all together by adding each layer.
- The first hidden layer has 12 neurons and expects 8 input variables. The second hidden layer has 8 neurons and finally the output layer has 1 neuron to predict the class (onset of diabetes or not).

Define Model



Compile the model

- Compiling the model uses the efficient numerical libraries under the covers (the so-called backend) such as Theano or TensorFlow.
- The backend automatically chooses the best way to represent the network for training and making predictions to run on your hardware.
- When compiling, we must specify some additional properties required when training the network.
- Remember training a network means finding the best set of weights to make predictions for this problem.

Compile the model

- We must specify the loss function to use to evaluate a set of weights, the optimizer used to search through different weights for the network and any optional metrics we would like to collect and report during training.
- In this case we will use logarithmic loss, which for a binary classification problem is defined in Keras as binary crossentropy.
- We will also use the efficient gradient descent algorithm adam for no other reason that it is an efficient default.

Fit the model

- We have defined our model and compiled it ready for efficient computation. Now it is time to execute the model on some data. We can train or fit our model on our loaded data by calling the fit() function on the model.
- The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the nb epoch argument.
- We can also set the number of instances that are evaluated before a weight update in the network is performed called the batch size and set using the batch size argument.
- For this problem we will run for a small number of epochs (150) and use a relatively small batch size of 10. Again, these can be chosen experimentally by trial and error.

Evaluate the model

- We have trained our neural network on the entire dataset and we can evaluate the performance of the network on the same dataset. This will only give us an idea of how well we have modeled the dataset (e.g. train accuracy), but no idea of how well the algorithm might perform on new data.
- We have done this for simplicity, but ideally, you could separate your data into train and test datasets for the training and evaluation of your model.
- You can evaluate your model on your training dataset using the evaluate() function on your model and pass it the same input and output used to train the model.
- This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics you have configured, such as accuracy.

Thank you

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

MITUSkillologies

Web Resources
http://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com