Before you run RNN.py, you should install numpy and pytorch.0.40. All of these packages should be installed under python3.6.0.

```
parser = argparse.ArgumentParser(description='PyTorch Implementation of RNN,GRU, and LSTM')
parser.add_argument('--lr', type=float, default=1e-3, metavar='LR',
                    help='learning rate (default: 1e-4)')
parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')
parser.add_argument('--batch_size', type=int, default=1, metavar='N',
                    help='input batch size for training (default: 128)')
parser.add_argument('--epochs', type=int, default=100, metavar='N',
                    help='number of epochs to train (default: 100)')
parser.add_argument('--embed_size', type=int, default=1,
                    help='intput in RNN')
parser.add_argument('--hidden_size', type=int, default=128,
                    help='hidden dimension in RNN')

args = parser.parse_args()
print(not args.no_cuda)
print(torch.cuda.is_available())
args.cuda = not args.no_cuda and torch.cuda.is_available()
kwargs = {'num_workers': 10, 'pin_memory': True} if args.cuda else {}
```

This part defines the hyper parameters you need to use in training RNN.

The learning rate(lr) defines how fast RNN can learn from the data.

no-cude defines whether we can use GPU to train RNN.

Batch_size defines how many data we can process in one iteration. (**You don’t need to change this setting**)

Epochs defines the number of epochs we run.

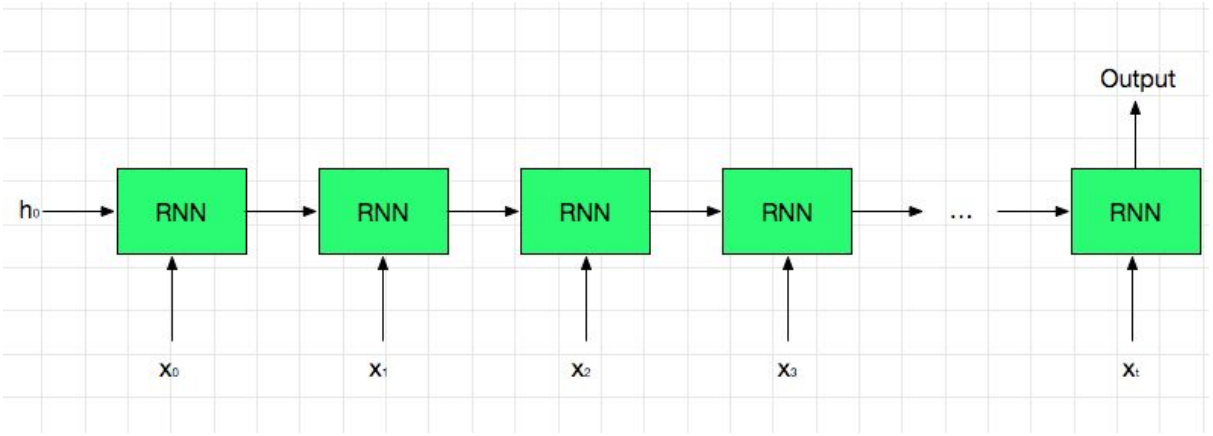
Embed_size defines the dimension of input data(**In your case, the input is a scalar value for each RNN cell, so you don’t need to change this setting**)

Hidden_size defines how many computation units in each RNN cell.

The classes RNN_Cell, GRU_Cell, and LSTM_Cell define how we compute a hidden output when a single data is given.

The class RNN_OneLayer, GRU_OneLayer, LSTM_Onelayer define how we compute the output when multiple data are given (there exists dependence among these data)

The following figure shows how RNN_OneLayer works.



The input to RNN_OneLyaer is one sequential data **x** =(x0,x1,....,xt). h0 is **0** by default(You don't need to initialize h0 by yourself. It has been initialized in RNN_OneLayer.). And the output is a scalar value or 1D vector, which is based on your application(e.g., It aims to predict a score or a distribution.)

There are three parameters in RNN_OneLayer: mode, length, and distributation.

Mode describes which application you need to use. If mode is "Score", then RNN will output a scalar value, which ranges from 0 to 1. If mode is "Distribution", then RNN will output a 1D vector, which demonstrates a data distribution.

length defines the length of the sequential data. For example, if the input data have 20 time steps, then length is set with 20.

distribution defines the length of your distribution if mode is set with "Distribution". For example, if you need to predict the probabilities of "A","B","C","D","E", then distribution is set with 5. If mode is "Score", you should set distribution with None.

GetData() is to preprocess your data. You need to store your input into data array, and output into label array.

train() is to train RNN model. There are four parameters in this function.

Mode describes which application you need to use. It could be "Score" or "Distribution".

epoch defines the number of epochs RNN runs, which has been defines in parser.

model defines which model you choose. It could be RNN,GRU,LSTM.

train_loader is an array where you store your input and output. It is returned by GetData().