Application IA

PREDICTING THE SECONDARY STRUCTURE OF GLOBAL PROTEIN USING NEURAL NETWORK WITH PYTOCH

NGUYEN Manh Tuan

Uapv2100691

Table of Contents

1. Explanation of code:

a. Context :

Predicting the secondary structure of protein is a problem that we have input is a sequence of amino-acid (primary structure) and our output should be the sequence of secondary structure (which contain of *α-helix*, *β pleated sheet* and *coil*).

b. Preprocess Data:

(a) Idea of preprocess data:

For the dataset, I have tested 2 approaches for the problem. First approach is input the whole sequence of each protein and predicting , second approach is like article, using a window slide.

(i) *Full-sequence input*:

We can call the problem is ***<u>many-to-many</u>***. I will input a full lenght sequence of amino acid and the output is full lenght sequence of secondary structure.

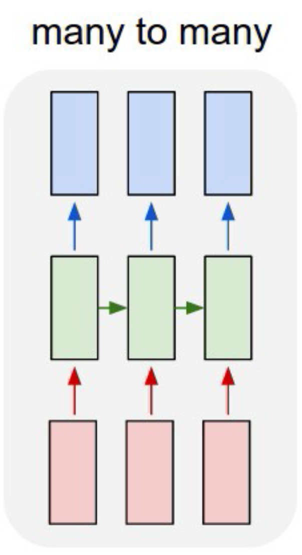


Figure 1: Visualize of many-to-many

After research on the dataset, i found that in the train set, the longest sequence is contain *of 498 timestemps* (498 values).

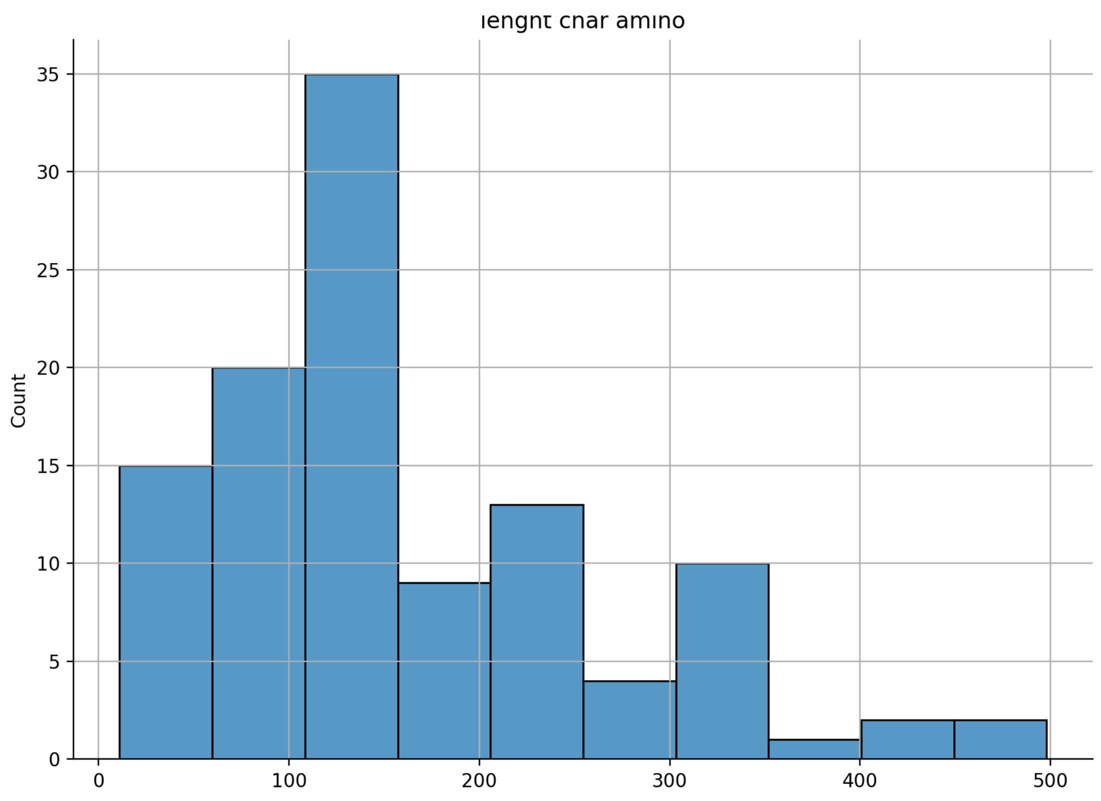


Figure 2: Visualize of raw dataset

As you can see in the figure below, most of dataset is lenght from 100-200, but in order to fit in the Neural Network, i will use zero-padding to make every samples is 498 sequence lenght.

(ii) *Window slide*:
For window slide, as the article showed, i add zero-padding into the beginning and in the last. Then make a window slide through the dataset , each 5 lenght sequence will be extract as 1 sample with label is the secondary structure of middle amino acid.

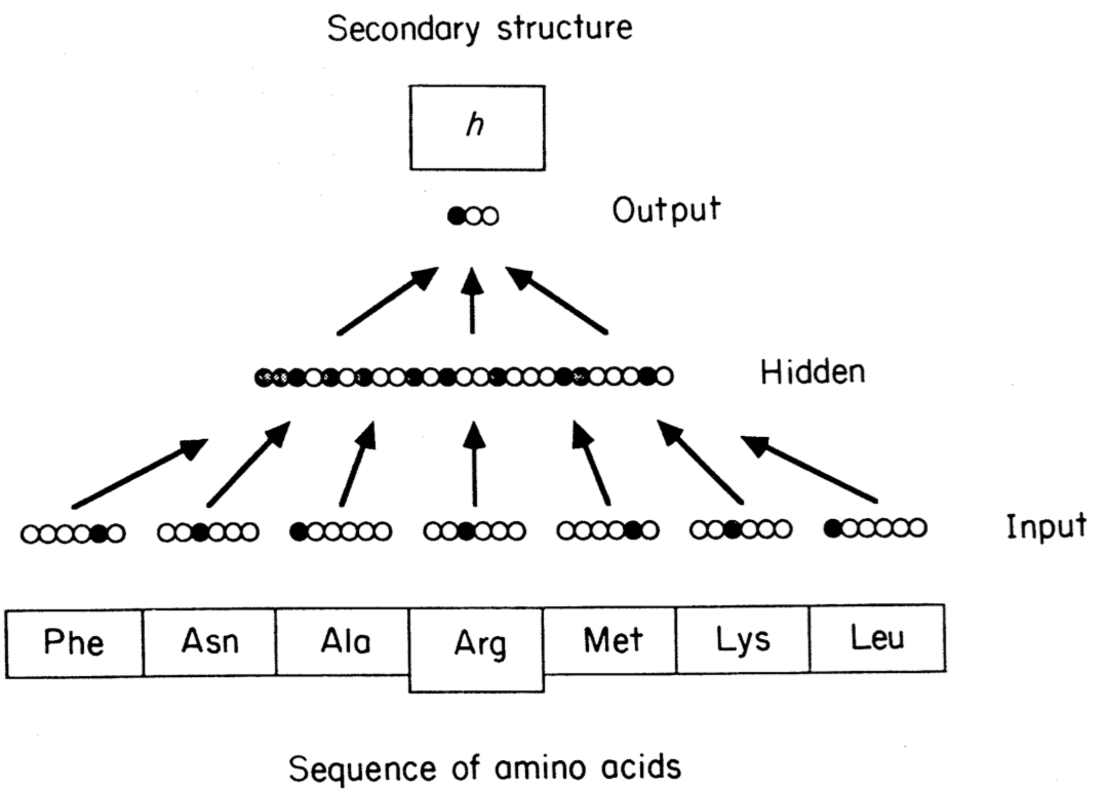


Figure 3: Visualize window slide method

This can be described as ***<u>many-to-one</u>*** problem.

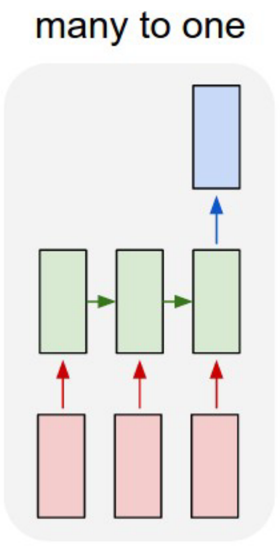
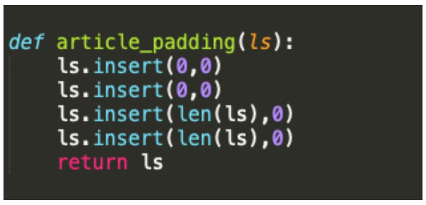


Figure 4: Visualize many-to-one problem

(b) List of function and explanation:

Despite of all normal line for read dataset , i will describe all the main function for the preprocessing dataset. The detail comment can be found in the source code. Input of dataset will be DataFrame type and the output will be Tensor. In this part, i only care to the version of article which is prove to be better.

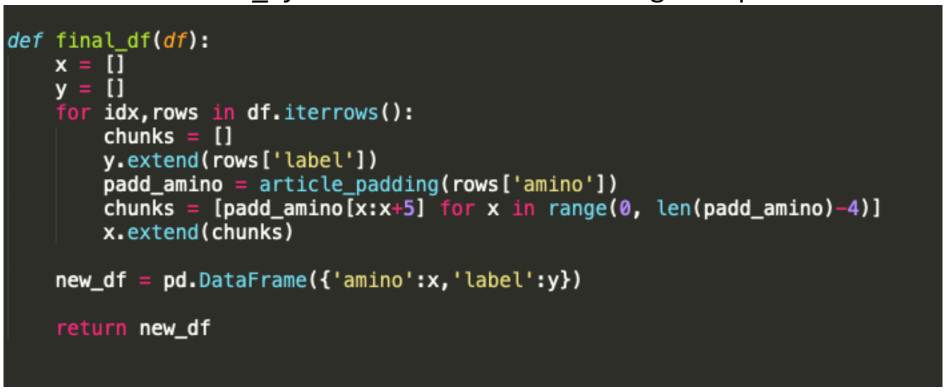
- ***Article_padding(ls)*** : This function is use to zero-padding like article indicate. So i will put 2 zero at the begin of each sequence and 2 zero at the end of each sequence.
    - *Parameter* :
        - *Ls* : list of each sequence protein.
    - *Output* :
        - *Ls* : final list after add zero padding



```python
def article_padding(ls):
    ls.insert(0,0)
    ls.insert(0,0)
    ls.insert(len(ls),0)
    ls.insert(len(ls),0)
    return ls
```

Figure 5: zero-padding function

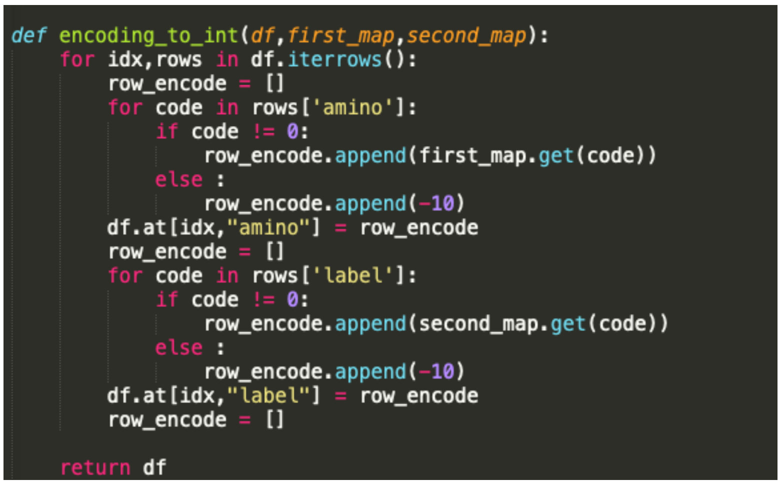
- ***Final_df(df)***: This function is use for converting sequence into sub-sequence of lenght 5 by window slide method.
    - *Parameter*:
        - *Df*: input dataframe that format with each rows is full lenght sequence as raw dataset.
    - *Output*:
        - *New_df* : return dataframe of 5 lenght sequences



```python
def final_df(df):
    x = []
    y = []
    for idx,rows in df.iterrows():
        chunks = []
        y.extend(rows['label'])
        padd_amino = article_padding(rows['amino'])
        chunks = [padd_amino[x:x+5] for x in range(0, len(padd_amino)-4)]
        x.extend(chunks)

    new_df = pd.DataFrame({'amino':x,'label':y})

    return new_df
```

Figure 6: final_df function

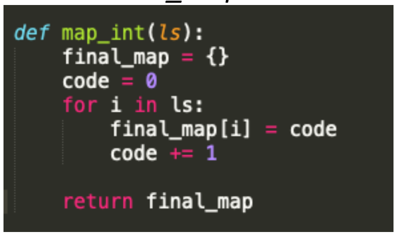
- ***Encoding_to_int():*** This function is first to encode all the amino acid or secondary symbol to number.
    - *Parameter*:
        - *Df* : final dataframe after zero padding.
        - *First_map* : the map from char to int of primary structure (amino acid)
        - *Second_map* : the map from char to int of secondary structure.
    - *Output*:
        - *Df* : final dataframe after convert all element to integer



```
def encoding_to_int(df,first_map,second_map):
    for idx,rows in df.iterrows():
        row_encode = []
        for code in rows['amino']:
            if code != 0:
                row_encode.append(first_map.get(code))
            else :
                row_encode.append(-10)
        df.at[idx,"amino"] = row_encode
        row_encode = []
        for code in rows['label']:
            if code != 0:
                row_encode.append(second_map.get(code))
            else :
                row_encode.append(-10)
        df.at[idx,"label"] = row_encode
        row_encode = []

    return df
```

Figure 7: encoding_to_int function

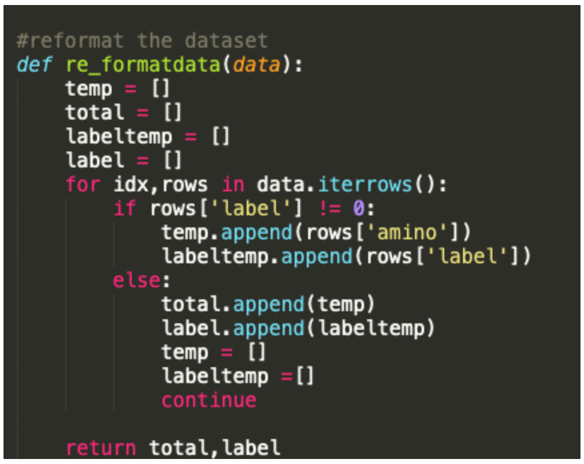
- ***Map_int():*** This function use to create the requirement map from char to int for ***encoding_to_int()*** function. As you can see, with zero padding i put it as -10 because i want the model to see that padding is far from others.
    - *Parameter*:
        - *Ls*: list of all unique char that existed in the dataset.
    - *Output*:
        - *Final_map* : return the map between char and interger.



```
def map_int(ls):
    final_map = {}
    code = 0
    for i in ls:
        final_map[i] = code
        code += 1

    return final_map
```

Figure 8: map_int function

- ***Re_formatdata():*** This function is use to convert the raw dataset to the list of all full lenght sequence of both primary structure and secondary structure.
    - *Parameter*:
        - *Data* : Raw DataFrame from .train or .test file.
    - *Output*:
        - *Total* : list of all full lenght sequences of primary structure.
        - *Label* : list of all full lenght sequences of secondary structure.



```python
#reformat the dataset
def re_formatdata(data):
    temp = []
    total = []
    labeltemp = []
    label = []
    for idx,rows in data.iterrows():
        if rows['label'] != 0:
            temp.append(rows['amino'])
            labeltemp.append(rows['label'])
        else:
            total.append(temp)
            label.append(labeltemp)
            temp = []
            labeltemp =[]
            continue

    return total,label
```

Figure 9: Reformat_data function

c. Model and training:

In order to find the best parameters for each model, i spent most of the time with Keras due to i have a lot of experience with that framework. And then i will begin to test the first 2 best with ***Pytorch***. For ***LSTM and RNN***, i also test with Bidirectional but Bidirectional only better with first approach of mine so i will not talk about this.

For each model, i choose the best optimizer and loss function :

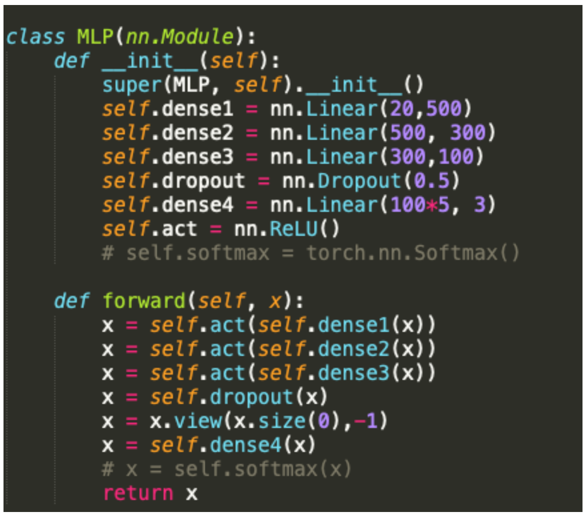
- **Loss function** : *Cross Entropy* – after test with Keras and follow my experience i beleive Cross Entropy will be the best loss function for this problem. Beside that, Pytorch provide also the Binearly Loss but this is multi class classification so Binearly Loss will not bring the best to the model. With CrossEntropyLoss of Pytorch, the result is require to be 1D Tensor so we have to Flatten our label before convert to Tensor with one hot encoding.
- **Optmizer** : I have 2 options is *SGD and Adam*. But most of the time, Adam's slightly better than SGD. But both of them in *RNN/LSTM* is not performing well when the loss change is smaller than 1e-3.

i. *Multilayers Perceptron (MLP)* :

For the **MLP**, i create 3 *Linear (dense)* layers with number of nodes is 500 -> 300 -> 100 . After that, i apply a *Dropout* layers with 0.5 chance and perform the Flattern in order to output the 1D Tensor. So that we can meet with *CrossEntropyLoss*.

For the number of nodes, after test with many case , i found out that 500 nodes should be enough in order to number of param in models is suitable with the complexity of dataset. 3 or 4 Dense layers should be enough because the dataset is not to complicated so we don't need to many hidden layers. All the layers will be apply with ReLU activation except the last Dense (softmax) . But i don't delcare Softmax in MLP class because CrossEntropyLoss requires not to decalare Softmax inside model.

I tried with 128, 64, 16 and 700 with Tanh and ReLU in order to get the result.



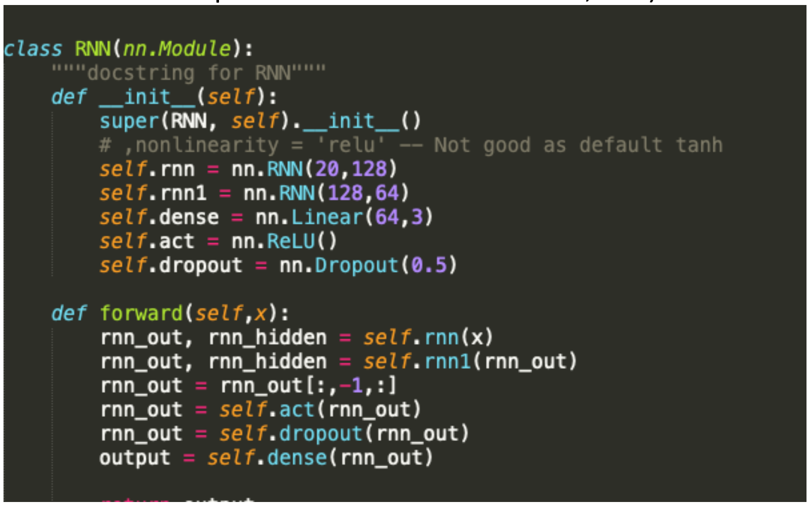
```python
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.dense1 = nn.Linear(20,500)
        self.dense2 = nn.Linear(500, 300)
        self.dense3 = nn.Linear(300,100)
        self.dropout = nn.Dropout(0.5)
        self.dense4 = nn.Linear(100*5, 3)
        self.act = nn.ReLU()
        # self.softmax = torch.nn.Softmax()

    def forward(self, x):
        x = self.act(self.dense1(x))
        x = self.act(self.dense2(x))
        x = self.act(self.dense3(x))
        x = self.dropout(x)
        x = x.view(x.size(0),-1)
        x = self.dense4(x)
        # x = self.softmax(x)
        return x
```

Figure 10: MLP class

ii. *Recurrent Neural Network (RNN)* :

For **RNN**, i create a network of *2 RNN layers with 1 final dense* at the end. I only test for 4-5 cases before to get this model (due to most of the time i tried to improve the first approach of mine). So 2 RNN layers should be enough because each sample is not too long (only 5 timestamps with 20 features – classes). First RNN layers is RNN with 128 nodes in hidden and the second one is 64 nodes in hidden. I have tried with Tanh and ReLU for this. Tanh is surprsingly to be better than ReLU in this case. All the parameter option i will be comment inside source code.

So because this is many-to-one problem, so at the end, i need to reshape the output to only 1 last timestep. (i havent tried with diferent timestep to take such as middle one, etc)

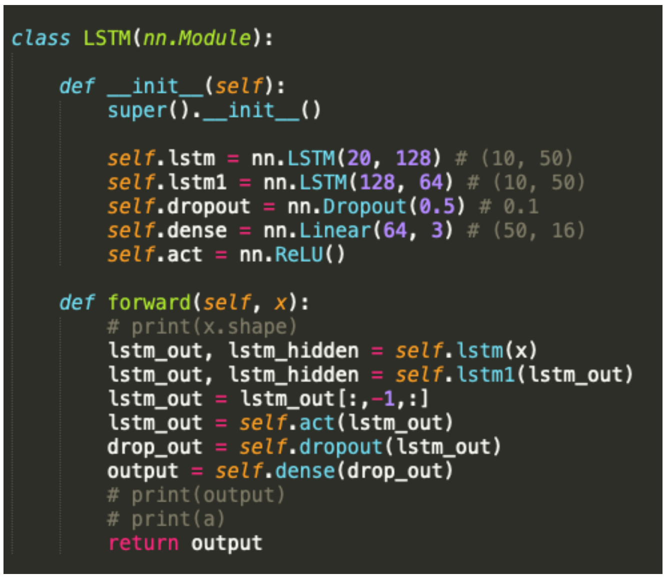


```python
class RNN(nn.Module):
    """docstring for RNN"""
    def __init__(self):
        super(RNN, self).__init__()
        # ,nonlinearity = 'relu' -- Not good as default tanh
        self.rnn = nn.RNN(20,128)
        self.rnn1 = nn.RNN(128,64)
        self.dense = nn.Linear(64,3)
        self.act = nn.ReLU()
        self.dropout = nn.Dropout(0.5)

    def forward(self,x):
        rnn_out, rnn_hidden = self.rnn(x)
        rnn_out, rnn_hidden = self.rnn1(rnn_out)
        rnn_out = rnn_out[:,-1,:]
        rnn_out = self.act(rnn_out)
        rnn_out = self.dropout(rnn_out)
        output = self.dense(rnn_out)
```

Figure 11: RNN class

iii. *Long Short Term Memory (LSTM)* :

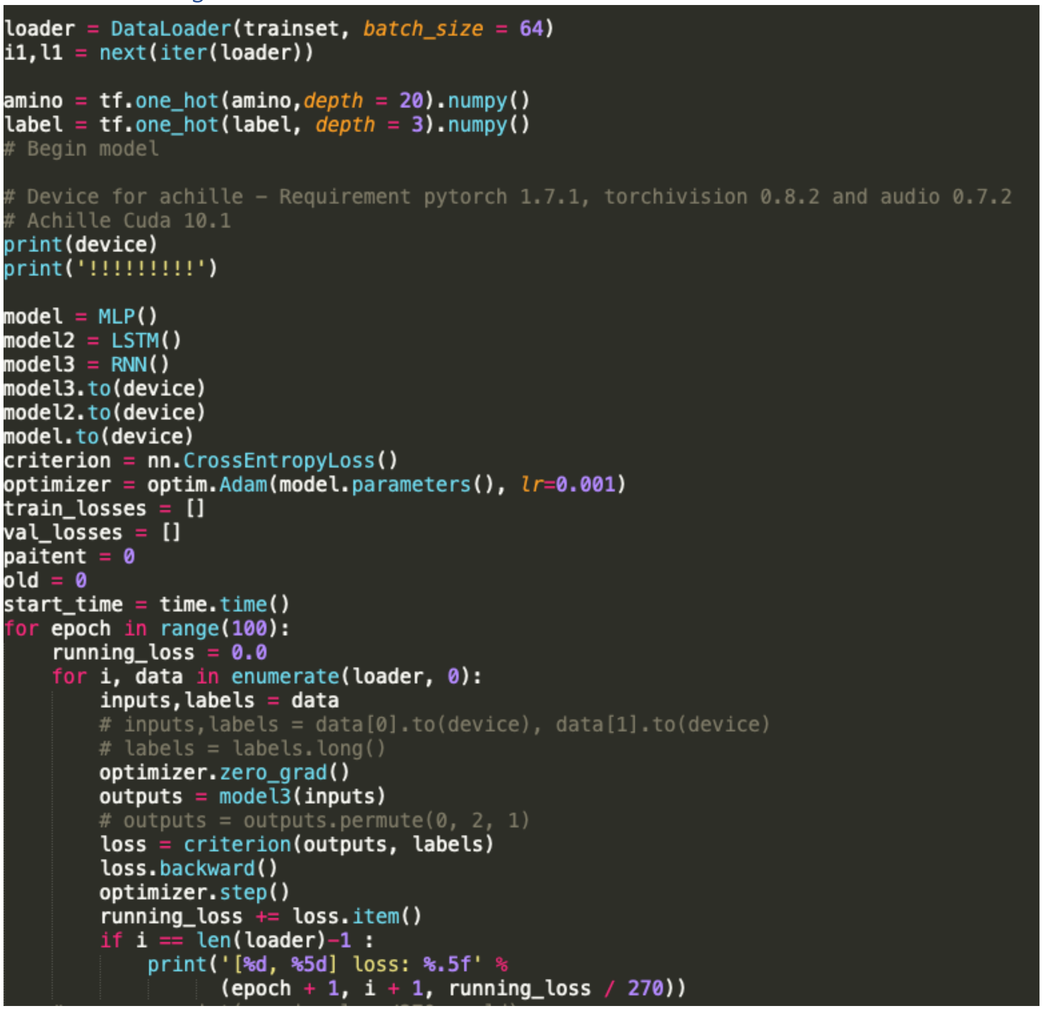
Similar to RNN, i also choose the same number of nodes as RNN. Only that i use the LSTM class provide by Pytorch. We also need to only take the last timestemps in order to create many-to-one problem.



```python
class LSTM(nn.Module):

    def __init__(self):
        super().__init__()

        self.lstm = nn.LSTM(20, 128) # (10, 50)
        self.lstm1 = nn.LSTM(128, 64) # (10, 50)
        self.dropout = nn.Dropout(0.5) # 0.1
        self.dense = nn.Linear(64, 3) # (50, 16)
        self.act = nn.ReLU()

    def forward(self, x):
        # print(x.shape)
        lstm_out, lstm_hidden = self.lstm(x)
        lstm_out, lstm_hidden = self.lstm1(lstm_out)
        lstm_out = lstm_out[:,-1,:]
        lstm_out = self.act(lstm_out)
        drop_out = self.dropout(lstm_out)
        output = self.dense(drop_out)
        # print(output)
        # print(a)
        return output
```

Figure 12: LSTM class

iv. *Training and Test*:

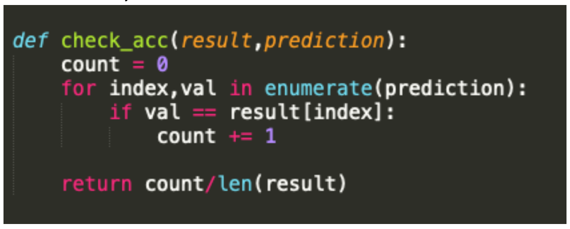


```python
loader = DataLoader(trainset, batch_size = 64)
i1,l1 = next(iter(loader))

amino = tf.one_hot(amino,depth = 20).numpy()
label = tf.one_hot(label, depth = 3).numpy()
# Begin model

# Device for achille - Requirement pytorch 1.7.1, torchivision 0.8.2 and audio 0.7.2
# Achille Cuda 10.1
print(device)
print('!!!!!!!!!')

model = MLP()
model2 = LSTM()
model3 = RNN()
model3.to(device)
model2.to(device)
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
train_losses = []
val_losses = []
paitent = 0
old = 0
start_time = time.time()
for epoch in range(100):
    running_loss = 0.0
    for i, data in enumerate(loader, 0):
        inputs,labels = data
        # inputs,labels = data[0].to(device), data[1].to(device)
        # labels = labels.long()
        optimizer.zero_grad()
        outputs = model3(inputs)
        # outputs = outputs.permute(0, 2, 1)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i == len(loader)-1 :
            print('[%d, %5d] loss: %.5f' %
                  (epoch + 1, i + 1, running_loss / 270))
```

Figure 13: Training code

First of all, i use the ***DataLoader()*** from Pytorch in order to split the trainning into smaller one (create mini-batch version instead of version offline). I choose batch size 64 instead of 32 because with this approach, i have more than 18.000 samples. So with 64 batch size, 1 epoch contain of 270 mini-batches.
As you can see above, i create a device object. In order to check GPU. So if Pytorch found NVIDIA, the data, model will be push to GPU and train on that. I already push data above in DataLoader so i don't need to push it in each mini-batch anymore.

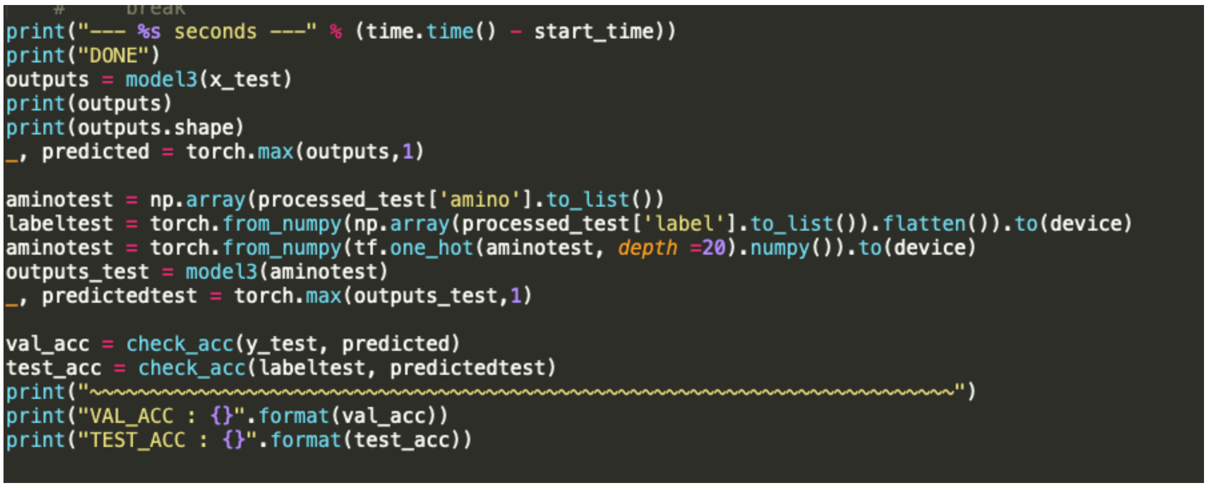
After this, i need to create a function to check the accuracy.



```python
def check_acc(result,prediction):
    count = 0
    for index,val in enumerate(prediction):
        if val == result[index]:
            count += 1

    return count/len(result)
```

Figure 14: accuracy function

The function is really basic, count every matched prediction with label.



```python
#       break
print("--- %s seconds ---" % (time.time() - start_time))
print("DONE")
outputs = model3(x_test)
print(outputs)
print(outputs.shape)
_, predicted = torch.max(outputs,1)

aminotest = np.array(processed_test['amino'].to_list())
labeltest = torch.from_numpy(np.array(processed_test['label'].to_list()).flatten()).to(device)
aminotest = torch.from_numpy(tf.one_hot(aminotest, depth =20).numpy()).to(device)
outputs_test = model3(aminotest)
_, predictedtest = torch.max(outputs_test,1)

val_acc = check_acc(y_test, predicted)
test_acc = check_acc(labeltest, predictedtest)
print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
print("VAL_ACC : {}".format(val_acc))
print("TEST_ACC : {}".format(test_acc))
```

Figure 15: Testing code

So after i get the function check_acc, i will take the model and make a predict on validation and test set with torch.max() function from Pytorch.

d. Comparison:
First , i did a comparision about the speed between Keras MLP and PyTorch MLP. You can see the result in the table below :

	Keras	PyTorch
CPU	Val_acc : 0.5822 Test_acc : 0.61 Time training: 61.53s	Val_acc : 0.5335 Test_acc : 0.5406 Time training: 166.88s
GPU	Val_acc 0.5824 Test_acc : 0.61 Time training: 138.17s	Val_acc : 0.5428 Test_acc : 0.5512 Time training: 95.64s

Table 1: Comaprision between Keras and PyTorch

It's really surprsingly when Keras bring more accuracy compare to PyTorch. I still trying to understand this. But as my idea is that PyTorch made us build from scratch more detaily but Keras is close-framework so maybe they put some twist on Sequential() class. Also i still don't have enough time to test ***Sequential()*** class of PyTorch to compare with Keras but i think they may have the same result. Keras is train faster on GPU but on GPU (Achille GPU NVIDIA) , PyTorch 's surpsingly better than Keras. I think because for PyTorch we have to push both data and model on the GPU so the computation in the same device after pushing is better. For Tensorflow, i beleive that at the training they will begin to push data to GPU when need so this maybe the reason.
GPU time in general is slower than CPU is because when i recorded the test is in daylight so GPU of university must has others use. Most of the time i test the model with achille GPU is faster than CPU.

2. Conclusion:

PyTorch is really good framework but i beleive this framework is still new and it's more suitable for scientist than developer. Due to the ability open of PyTorch, we can easily build from scratch so it's easier for scientist to test and define the way of forward and backward, etc... Also because we break down each component inside Neural Network in PyTorch, this may be longer but easier to de-bug than Keras.
But Keras is also has the advantage. Because Tensorflow is really famous framework, and easy to develop a model so Keras is really good when you need to test hyper parameter , quick develop a model , etc...

For 2 approaches i decalred above, i beleive my approach is better with RNN than artticle approach (i managed to have ***61% for Bidiretctional LSTM***). But the potential of this is really promissing because i only have 88 samples for the trainning set. So if we have more samples, i beleive the first approaches will bring really good result. The approach of article – window slide is good also because they manage to upsize the dataset with each smaller window as input train. But due to each sample only contain 5 timestemps so it's not really good with RNN as i test. But currently, this is the best approach i think due to the cost of increase the dataset is not a good idea.

Github link : https://github.com/tuanct1997/protein_secondary_strcuture