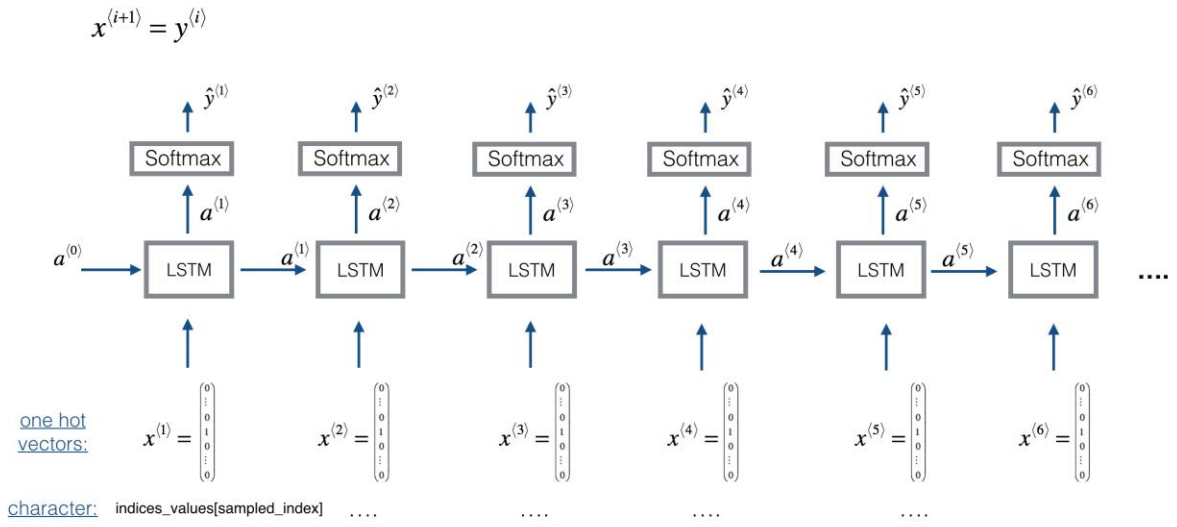
CSC 4343 Homework 2

Your task in this homework is to build a character level LSTM network to generate (English) names.

Download training data from moodle. ("yob2018.txt" contains name, gender, and number. Extract the names from the first entry of each line.) Remove names with special characters and convert the rest to lower cases such that the training data contain only names with characters a-z.

Let the length of the longest name string in your training data be N. You should make all the strings have the same length N by padding the shorter ones with 'space' character.

Implement a LSTM network with the following structure. (The LSTM box contains 2 layers, each has 128 cells).



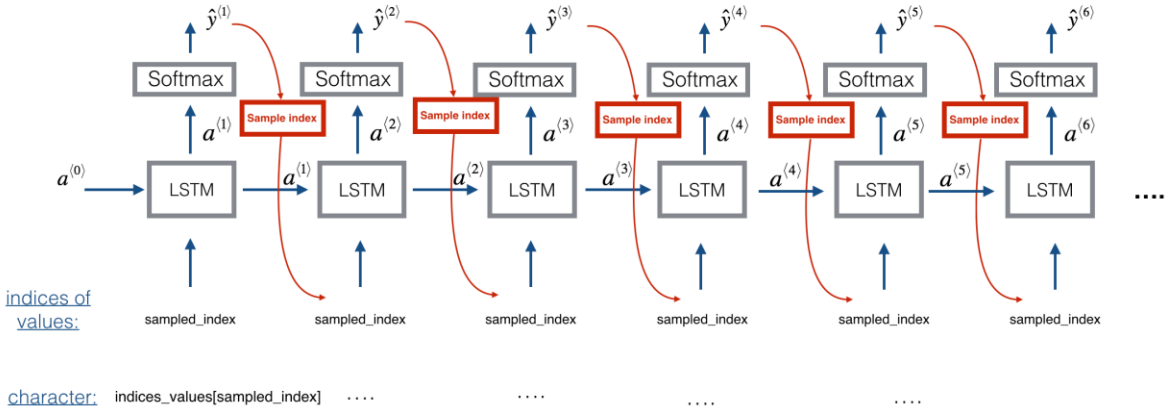
Note that the same name (string of characters) is used both as input and output in training but the input is the string right-shifted by one position. After shifting, add an artificial character (marker for the beginning of the name) to the beginning (the left most) position. For the output string, you should add a different artificial character to the end, which serves as a marker for the end of the string. If you have padded 'space' to the string, this end-of-name character should be added to the actual end position of the name, not the position at the end of the 'space' characters. The input and the output strings are thus of the same length.

You should translate the characters in the strings into one-hot vectors and use the one-hot vectors as the inputs to the LSTM. Because of the two artificial characters, you have 28 characters in total and the length of the one-hot vectors should be 28. (As an example, for the input, $x^{<1>}$ in the above figure should be the one-hot vector corresponding to the first artificial character.) You can use an all zero vector for the padding 'space' character.

Train the model with cross-entropy loss from the softmax outputs at all (**valid**) steps. That is, if you padded a string, the cross-entropy from the padding positions should not be included in the final loss.

After training the model, you can sample (generate names) using the model. Implement the following process to generate a name using the trained model. You should implement a function

"generate_name(M, N)". The function uses your trained model "M" and each call to the function should generate a name (string).



The next character should be sampled following the probability (the output of the softmax layer), not using the one with the largest probability. (You may use numpy.random.multinomial to draw samples from a multinomial distribution.) Due to the sampling, multiple calls to the function should generate different names. If your sample is the beginning-of-name character, ignore it and resample. If your sample is the end-of-name character, stop the process and return the name. Even if you don't see the end-of-name character in the process, once N characters have been generated, you should stop the process and return the name.

Homework Submission:

Upload a Python file (not .ipynb file) named **RNNModel.py** in moodle. You need have the following functions in the file:

1. A function **create_model(max_len)** which returns a PyTorch object (untrained, subclass of nn.Module) that implements the above LSTM model. "max_len" is the max length of the input string.
2. A function **train_model(model, n_epochs)** which trains (as described above) the **model** object passed to it for **n_epochs**. Your training code can assume that the file "yob2018.txt" is in the same directory as the file **RNNModel.py**. At the end of each epoch, print out the average loss during that epoch.
3. A function **load_trained_model()** which should download your trained model saved somewhere online and load it from the downloaded file. The function then returns the model. Do not train the model from scratch in this function. Same as HW1, you should not upload the model file to moodle. Instead, you should share it in your google drive (or other online storage which can be *shared by link*).
4. A function **generate_name(M, N)** which returns a string of length **up to "N"** generated by the (trained) model "M". We will call **load_trained_model()** to obtain your trained model and then call this function to generate a name string using your model.

Make sure we can import these functions from the .py file without running your training code. If we want to train the model, we will explicitly call the train_model function.

We will test your model by code similar to the following:

```
from RNNModel import create_model, train_model, load_trained_model,
generate_name

um = create_model(mlen)
train_model(um, 20)

tm = load_trained_model()
# print 5 generated names (We expect there are different names among the 5.)
for i in range(5):
       print(generate_name(tm, 10))
```