```
from google.colab import files, drive

drive.mount('/content/drive/')
```

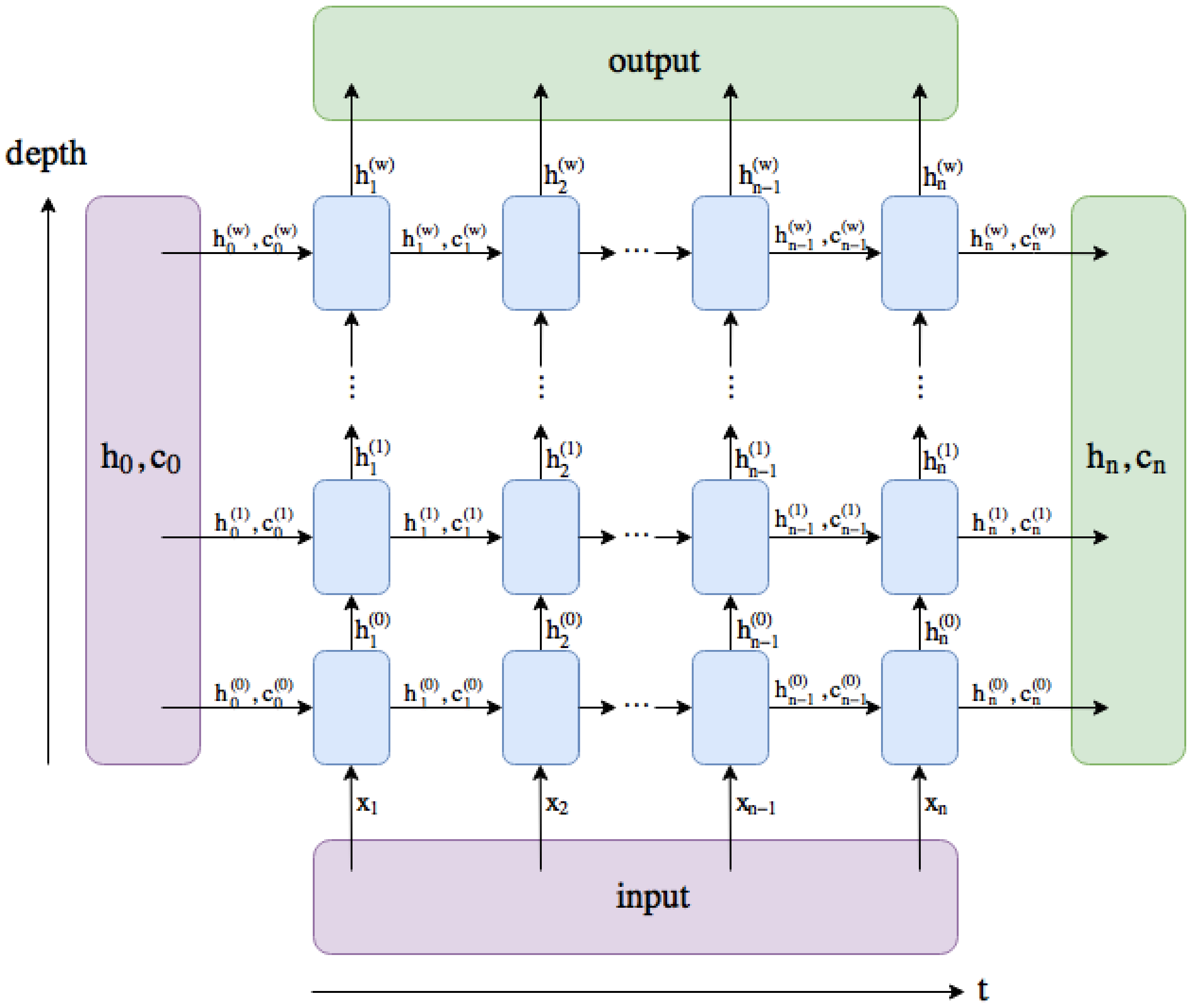
```
## using `TfidfVectorizer` from sklearn to generate tf-idf values for every word in each document.

## read 500 documents from positive reviews and another 500 documents from negative reviews (you are
## allowed to read more documents).

## please set `max_features` to 200 when you build the tfidf vectorizer, meaning that we only use
## top 200 words to form our vocabulary V.
```

```
## In this section, you need to construct the training set (#documents,#max_length_among_documents,
## Vocabulary_size)

## For each word in a document, you use the onehot encoding-like vector representation except that
## we use the tfidf value calculated in the previous step if that word appears in that document,
## rather than 1. The dimension of this vector is the size of the vocabulary. For example, if a word w
## in document d is the 3rd word in the vocabulary, this word is represented as (0,0,tfidf(w,d),0,0,...0).
## If the length of a document (l) is less than the max_length(L), word 1, word 2, ..., word L-l are
## represented as zero vectors (0,0,...,0).


## Meanwhile, you also need to generate labels (#documents, 2) since this is a binary classification
## problem.
```

DataLoaders - train loader and valiation loader

```
## Create train loader and validation loader
```

Model: a general framework for multi-layer RNN



```
from __future__ import unicode_literals, print_function, division

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class Model(nn.Module):

  def __init__(self, input_size, output_size, hidden_size, n_layers):
    super().__init__()

    self.hidden_size = hidden_size
    self.n_layers = n_layers

    self.rnn = nn.RNN(input_size,hidden_size,n_layers,batch_first=True) # rnn layer
    self.fc1 = nn.Linear(hidden_size,output_size) # rnn output (y_t) --> output (y'_t)
    self.fc2 = nn.Linear(output_size,2) #the output from the last time period ->sentiment prediction

  def forward(self,x, hidden):
    batch_size = x.size()[0]

    hidden = self.init_hidden(batch_size)

    rnn_out,hidden = self.rnn(x,hidden)
    rnn_out = self.fc1(rnn_out)
    last_out = rnn_out[:,-1,:].view(batch_size,-1)
    out = F.softmax(self.fc2(last_out))

    return out,hidden

  def init_hidden(self,batch_size):
    hidden = torch.zeros(self.n_layers, batch_size, self.hidden_size).cuda()
    return hidden

model = Model(200,32,256,3)
print(model)
```

Training

```
## training and validating process

## You need to print out the following message for every batch in each epoch.


print('Epoch:{}/{}'.format(epoch,n_epochs), # epoch is the index of epoch
      'Batch:{}'.format(b),  # b is the index of batch
      'Train Loss:{:.5f}'.format(train_loss),
      'Val Loss:{:.5f}'.format(val_loss))
```