**Using Embeddings and CNN for Document Classification**

This project employs three methodologies: first by loading pretrained word embeddings, then by fine-tuning these pretrained embeddings by classifying entire news articles, and finally by using a convolutional neural network to capture the spatial relationships between words.

In this project, I used the AG News dataset. To model the sequences of words in AG News, I introduced a variant of the Vocabulary class SequenceVocabulary, to bundle several tokens vital for modeling sequences.

After describing the dataset and how the vectorized minibatches are constructed, I stepped through the loading of pretrained word vectors into an Embedding layer and demonstrated how they are customized to our setting. Then, the model combines the pretrained Embedding layer with the CNN. In an effort to scale up the complexity of the model to a more realistic construction, I also identify the places where I utilize dropout as a regularization technique. I then discuss the training routine. Finally, I concluded the project by evaluating the model on a test set.

## The AG News Dataset

The [AG News dataset](http://bit.ly/2SbWzpL) is a collection of more than one million news articles collected in 2005 by academics for experimenting with data mining and information extraction methods. The goal of this project is to illustrate the effectiveness of pretrained word embeddings in classifying texts. In this project, I use a slimmed-down version consisting of 120,000 news articles that are split evenly between four categories: Sports, Science/Technology, World, and Business. In addition to slimming down the dataset, I focus on the article headlines as my observations and create the multiclass classification task of predicting the category given the headline.

I preprocessed the text by removing punctuation symbols, adding spaces around punctuation (such as around commas, apostrophes, and periods), and converting the text to lowercase. Additionally, I split the dataset into training, validation, and testing sets by first aggregating the data points by class label and then assigning each data point to one of the three splits. In this way, the class distribution is guaranteed to be identical across the splits.

The NewsDataset.\_\_getitem\_\_() method, shown in the notebook, follows a fairly basic formula: the string representing the input to the model is retrieved from a specific row in the dataset, vectorized by the Vectorizer, and paired with the integer representing the news category (class label).

## Vocabulary, Vectorizer, and DataLoader

In this project I introduce SequenceVocabulary, a subclass of the standard Vocabulary class that bundles four special tokens used for sequence data: the UNK token, the MASK token, the BEGIN-OF-SEQUENCE token, and the END-OF-SEQUENCE token. These tokens serve three different purposes. The UNK token (short for unknown), allows the model to learn a representation for rare words so that it can accommodate words that it has never seen at test time. The MASK token serves as a sentinel for Embedding layers and loss calculations when we have sequences of variable length. Finally, the BEGIN-OF-SEQUENCE and END-OF-SEQUENCE tokens give the neural network hints about the sequence boundaries.

The second part of the text-to-vectorized-minibatch pipeline is the Vectorizer, which both instantiates and encapsulates the use of the SequenceVocabulary. In this project, the Vectorizer performs the function of restricting the total set of words allowed in the Vocabulary by counting and thresholding on a certain frequency. The core purpose of this action is to improve the signal quality for the model and limit the memory model and memory usage by removing noisy and low-frequency words.

After instantiation, the Vectorizer's vectorize() method takes as input a news title and returns a vector that is as long as the longest title in the dataset. It has two key behaviors. The first is that it stores the maximum sequence length locally. Normally, the dataset tracks the maximum sequence length, and at inference time, the length of the test sequence is taken as the length of the vector. However, because we have a CNN model, it's important to maintain a static size even at inference time. The second key behavior, shown in the code snippet in the project, is that it outputs a zero-padded vector of integers, which represent the words in the sequence. Additionally, this vector of integers has the integer for the BEGIN-OF-SEQUENCE token prepended to the beginning and the integer for the END-OF-SEQUENCE token appended to the end. From the classifier's perspective, these special tokens provide evidence of the sequence boundaries, allowing it to react to words near the boundary differently than to words near the center.

## The NewsClassifier Model

In the project, we compared the vectors against one another to discover interesting linguistics insights. However, pretrained word vectors have a much more impactful use: we can use them to initialize the embedding matrix of an Embedding layer.

The process for using word embeddings as the initial embedding matrix involves first loading the embeddings from the disk, then selecting the correct subset of embeddings for the words that are actually present in the data, and finally setting the Embedding layer's weight matrix as the loaded subset. One issue that commonly arises is the existence of words that are in the dataset but are not among the pretrained GloVe embeddings. One common method for handling this is to use an initialization method from the PyTorch library, such as the Xavier Uniform method, as shown in the project.

The NewsClassifier in this example builds on the ConvNet classifier. Specifically, we use the Embedding layer, which maps the input token indices to a vector representation. We use the pretrained embedding subset by replacing the Embedding layer's weight matrix, as demonstrated in in the project.The embedding is then used in the forward() method to map from the indices to the vectors.

## The Training Routine

Training routines consist of the following sequence of operations: instantiate the dataset, instantiate the model, instantiate the loss function, instantiate the optimizer, iterate over the dataset's training partition and update the model parameters, iterate over the dataset's validation partition and measure the performance, and then repeat the dataset iterations a certain number of times.

## Evaluation and Prediction

In this project, the task was to classify news headlines to their respective categories. There are two kinds of methods for understanding how well the model is carrying out the task: a quantitative evaluation using the test dataset, and a qualitative evaluation to personally inspect classification results.

### Evaluating on the test dataset

Here, you simply set the model in eval mode to turn off dropout and backpropagation (using classifier.eval()) and then iterate over the test set in the same manner as the training and validation sets. I experimented with different training options before performing model evaluation. I had a fairly decent accuracy. the test set was used only once in the entire experimentation process.

### Predicting the category of novel news headlines

The goal of training a classifier is to deploy it in production so that it can perform inference or prediction on unseen data. To predict the category of a news headline that isn't already processed and in the dataset, there are a few steps. The first is to preprocess the text in a manner similar to preprocessing data in the training. For inference, we use the same preprocessing function on the input as the one used in training. This preprocessed string is vectorized using the Vectorizer used during training, and converted into a PyTorch tensor. Next, the classifier is applied to it. The maximum of the prediction vector is computed to look up the name of the name of the category.