Introduction:
As machine learning models become increasingly widespread and their computational cost diminishes, most news aggregators (e.g. Google News, Facebook, etc.) have fully automated their approach to news classification by analyzing the words present in each article. Once news is classified it is fed to web pages that allow users to filter their news by category, directly influencing the news ingested at a massive scale. Given the millions of news articles posted every day, news classifiers must be highly accurate as there are simply too many news entries for humans to manually verify. We took the need for high accuracy across multiple classes as a challenge, and were eager to implement more advanced NLP techniques than the bag-of-words approach used in the spam classifier. Furthermore, we decided on implementing a neural network to capture more advanced features within news text.

Experimental Setup:
The data we ended up using came from a Kaggle dataset containing data from Huffington Post articles, which contained entries with the following fields:

- Short description
- Headline
- Date
- Link
- Authors
- Category

Of these fields, we ended up using only the short description and headline as features in our data, and mapped the category name to discrete integer classes. One thing we did note was that the name of the author seemed quite indicative of the news’ class. Despite this, we decided to not include the author’s name among our features as we wanted to evaluate how successful our model could be using only the text written as input features.

Once we had our modified dataset, we began considering ways to represent words in the headline and description as features. Many online sources suggested using word embeddings, where words are mapped to vectors of real numbers. Word embeddings are one of the most popular ways of representing a vocabulary, allowing one to capture the context of a word in a document, in addition to the semantic and syntactic similarity between different words. Keras, a library that runs on top of TensorFlow, allows embeddings to be added as an input layer easily, motivating our decision to implement our Neural Net using this framework.
After deciding to use word embeddings, we ran into another challenge: creating them. Word2Vec, a popular neural net model, is often used to train word embeddings, but further research indicated our corpus of text was too small to produce effective embeddings. To surpass this limitation, we decided to use GloVe embeddings, a set of pre-trained word vectors. These contained a vocabulary that was 400K words long, where each number was a 1x100 dimensional vector.

Finally, before training our model we needed to properly set up our vocabulary and remove filler words from text entries. We cleaned the text following the steps: (1) removing punctuation, (2) converting all text to lowercase, and (3) removing stopwords (identified using Natural Language Toolkit library). All headline and description entries were then fed to a tokenizer that kept the 10,000 most common words, converted text entries to their integer representation, and then padded the end of the array with zeros (representing blank words).

After implementing the word embeddings and cleaning the text, we began to train the model. We experimented with a range of different architectures, optimizing on our cross-validation set and comparing the results on our test set. We attempted adding additional layers to the network, but found no significant improvement in our test set accuracy beyond two densely connected layers. We further experimented with adding additional nodes, but found that we optimized the CV set with 1000 nodes in the first layer and 200 in the second. Additionally, we experimented with a series of convolutional layers, but this actually resulted in slightly worse accuracy.

Once we had the general architecture complete, we started to play with the hyperparameters of the network. We adjusted alpha, tried various batch sizes, and explored the effects of various activation functions. Furthermore, we tried different regularization techniques, which will be discussed more with the results.

When we considered the goal for this project, we weren’t entirely sure of what to expect. At the low end, guesses just due to random chance would be accurate about 3% of the time. We obviously were shooting to do significantly better than chance, but this gave us a lower bound nonetheless. At the high end, we had seen a few people on Kaggle had reported accuracies of around 60%. Many of these people appeared to have been working on these project for weeks if not months, so we were shooting to be up around 50% perhaps, but maybe not as high as some of the 60+% reports.
**Results:**
The best result that we saw was a test accuracy of 50.59%. This was very similar to our cross-validation accuracy of 50.47%, with slightly higher training accuracy.

![Accuracy of Training and CV Sets](image)

As we can see from the graphs above, we were highly concerned with the overfitting that we experienced. We tried using both L2 regularization and dropout and found dropout to be more effective for this case. However, increasing our dropout rate too high (as seen in the right chart) hurt our overall accuracy. At the same time, keeping it as low as in the first chart led to high variance. For this reason, we decided to keep it low, but run the model for a smaller number of epochs (4 in the final model) to allow the higher accuracy of a low dropout rate while preventing too much overfitting.

**Conclusions:**
Overall we found this to be a successful implementation of a multiclass classification problem with neural nets. We were happy with our result, which achieved a moderate accuracy across 31 classes. We think a large part of this success is due to the GloVe embedding. This not only speeds up the training time but also allows us to capture the similarities between words, which we believe to be a powerful ability and is not captured by something like a more simple bag of words implementation. Although we still have a long way to go before being able to compete with the near-100% accuracy of many tech giants, we were very encouraged by our result over a relatively short period of time.

**Code:**
https://github.com/lukealvoeiro/news-classifier