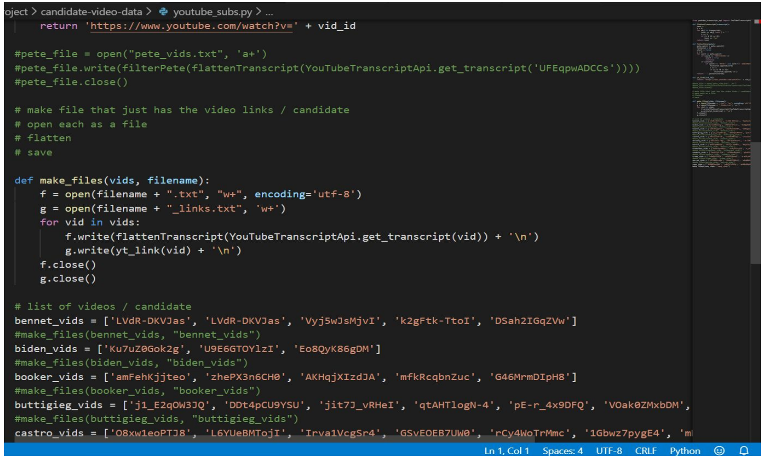
Classifying 2020 Presidential Candidate Speeches

Daniel Oshiro

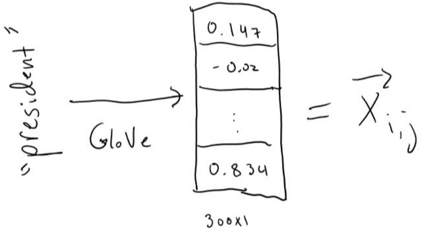
LIGN 167 Final Project, Fall 2019

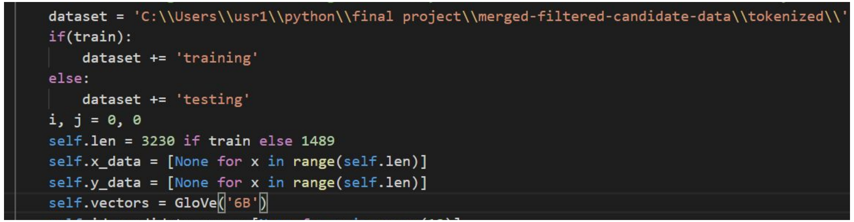
Gathering and Cleaning Data



- I used data from the Council on Foreign Relations Site: [“The 2020 Candidates in Their Own Words”]
    - This data was in a variety of forms, including op-ed articles, speech transcripts, policy positions, videos and podcasts
    - I used [this api] to download transcripts from YouTube videos, and cleaned the data by filtering and tokenizing the data, and splitting into paragraphs.

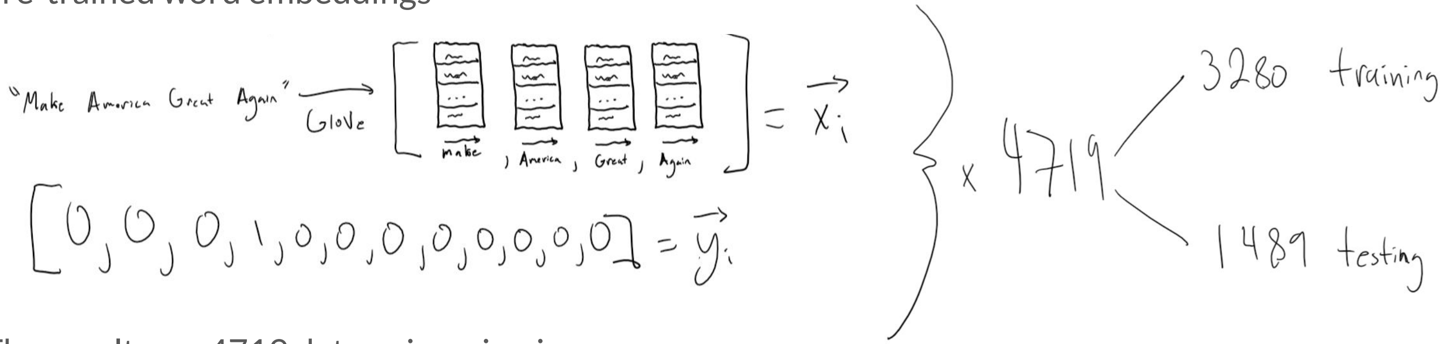
Turning Data into Suitable Input for Learning





```
dataset = 'C:\\Users\\usr1\\python\\final project\\merged-filtered-candidate-data\\tokenized\\'
if(train):
    dataset += 'training'
else:
    dataset += 'testing'
i, j = 0, 0
self.len = 3230 if train else 1489
self.x_data = [None for x in range(self.len)]
self.y_data = [None for x in range(self.len)]
self.vectors = GloVe('6B')
```

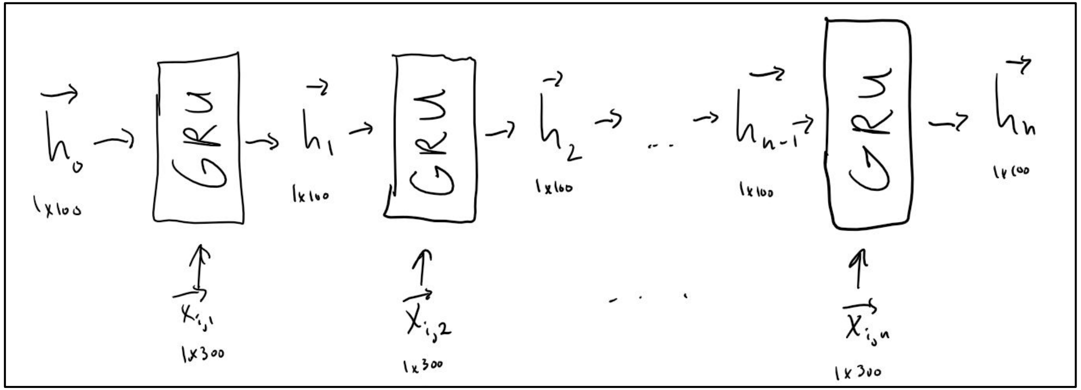
- After cleaning and tokenizing the data, I used the 300 dimensional, [GloVe](#) “6B” model for pre-trained word embeddings



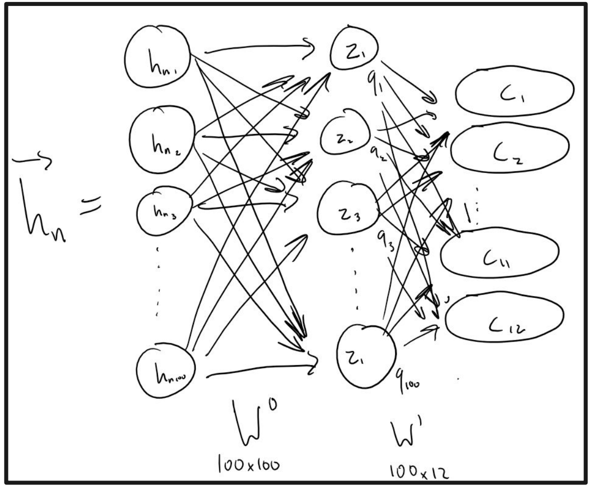
- The result was 4719 data pairs x_i, y_i
  - X_i was a list of 1x300 word embeddings corresponding to each paragraph
    - The longest x_i was 394 words while the shortest was just 5!
  - Y_i is a 1x12 one-hot vector corresponding to which candidate said the paragraph
  - These data pairs were split into a training set of size 3230 and a test set of size 1489

Model 1: RNN Implementation

GRU-based RNN that processes each paragraph

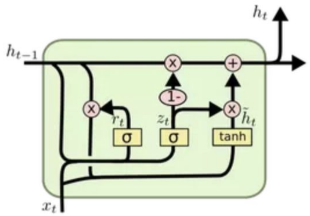


MLP that classifies final hidden output of RNN



- The RNN has a hidden dimension of 100, and takes in 300 dimensional word embeddings at each time step
- For the Recurrent Unit, I used a Gated Recurrent Unit (GRU) in order to avoid the vanishing gradient problem
- The final hidden state is used as the input to a fully-connected multi-layer perceptron with a single, 100 dimensional hidden layer
- The output of the MLP is run through Softmax and the prediction is made by taking the max

Model 1: GRU Cell and Training

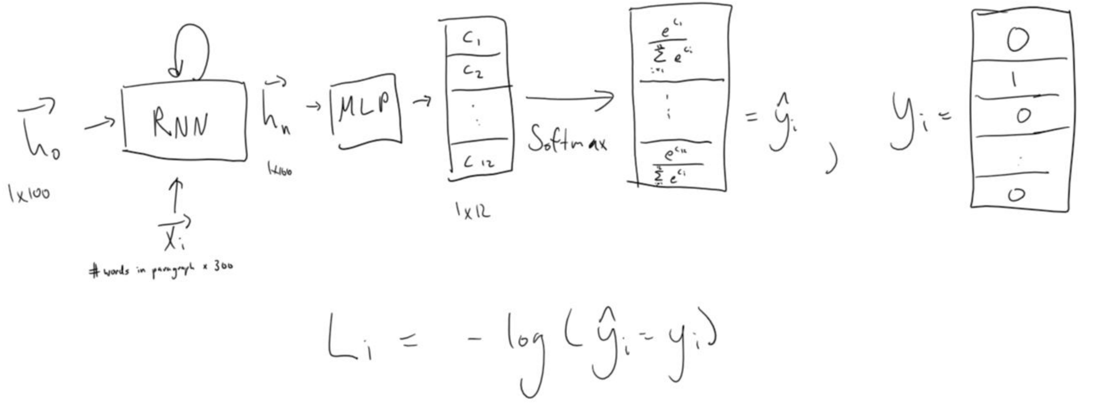


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

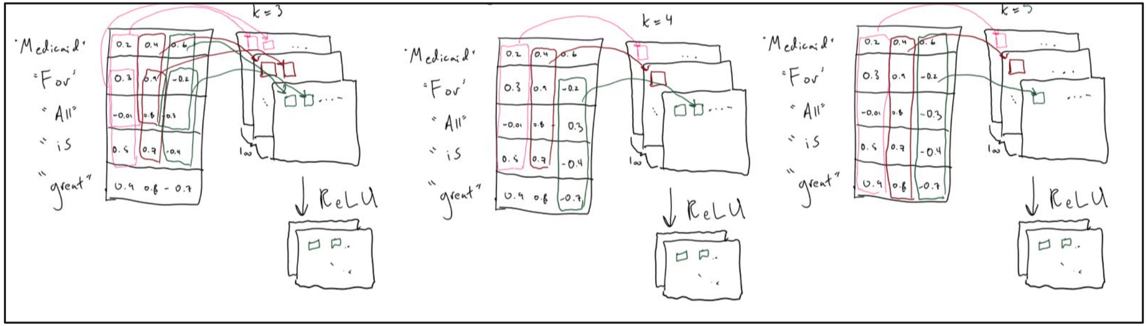
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$



- The GRU acts similarly to an RNN in that it takes in a hidden state from the previous time step and the next word in the sequence
- The GRU calculates the next hidden state using an update gate, z_t, and a relevance gate, r_t in order to save prior states, thus avoiding the vanishing gradient problem

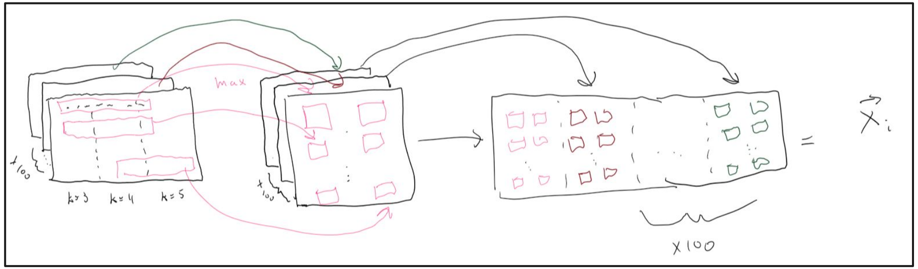
- For the loss function, I ran the output of the MLP through the Softmax to interpret each entry as a probability and summed the negative log probabilities
- Training was carried out over 40 Epochs, with shuffled batches of size 256 training examples

Model 2: CNN Implementation

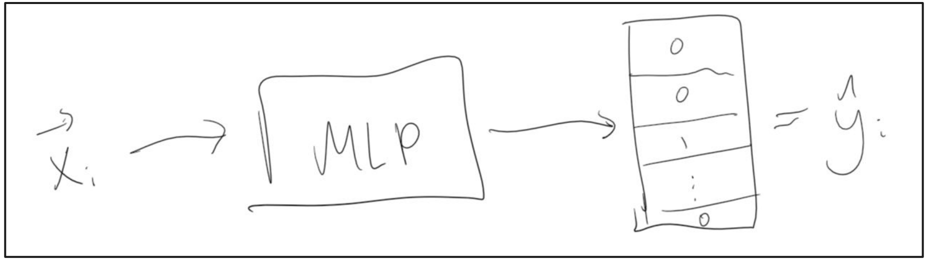


- To the left is the convolution layer, which, for each of the word embedding dimensions, reads 3, 4, and 5 consecutive words
- These tensors are then passed through a ReLU activation function

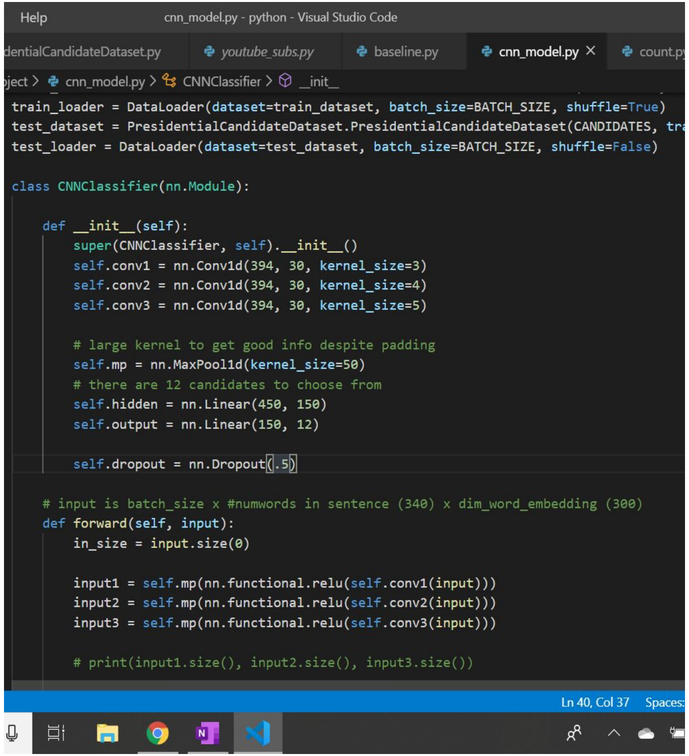
- To the right is the concatenation and max-pooling layer
- The convolution tensors are concatenated, a max-pool is done to keep the two most important features, and the layers are unraveled to make a single 2d tensor, xi



Model 2: Classification and Training



- The output from the CNN is passed into an MLP that returns a 12-dimensional vector used as a guess
- Just as in the RNN, y-hat is passed through a softmax, then a negative log probability to calculate the loss of the network




```python
train_loader = DataLoader(dataset=train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_dataset = PresidentialCandidateDataset.PresidentialCandidateDataset(CANDIDATES, tra
test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE, shuffle=False)

class CNNClassifier(nn.Module):

    def __init__(self):
        super(CNNClassifier, self).__init__()
        self.conv1 = nn.Conv1d(394, 30, kernel_size=3)
        self.conv2 = nn.Conv1d(394, 30, kernel_size=4)
        self.conv3 = nn.Conv1d(394, 30, kernel_size=5)

        # large kernel to get good info despite padding
        self.mp = nn.MaxPool1d(kernel_size=50)
        # there are 12 candidates to choose from
        self.hidden = nn.Linear(450, 150)
        self.output = nn.Linear(150, 12)

        self.dropout = nn.Dropout(.5)

    # input is batch_size x #numwords in sentence (340) x dim_word_embedding (300)
    def forward(self, input):
        in_size = input.size(0)

        input1 = self.mp(nn.functional.relu(self.conv1(input)))
        input2 = self.mp(nn.functional.relu(self.conv2(input)))
        input3 = self.mp(nn.functional.relu(self.conv3(input)))

        # print(input1.size(), input2.size(), input3.size())
```

Results

Type of Model	**% of Test Set Correctly Classified**
RNN + Classifier	47.347% (705/1489)
CNN + Classifier	33.311% (496/1489)
Softmax Distribution Over # of Datapoints / Candidate	22.633% (337/1489)
Linear Distribution Over # of Datapoints / Candidate	10.208% (152/1489)

These results show that the networks are significantly better than baseline, but that the results aren't great. I mostly blame the small, somewhat inconsistent dataset that relied on Youtube Transcripts, which are imperfect and lack punctuation / other indicators. I'd love to try on more data!

Citations

- Title slide democrats Image - https://www.reuters.com/article/us-usa-election-debate-impeachment-factb/factbox-democratic-presidential-candidates-on-impeaching-donald-trump-idUSKBN1WT183
- Trump Image - https://www.whitehouse.gov/people/donald-j-trump/
- Andrew Yang Image - https://en.wikipedia.org/wiki/Andrew_Yang
- GRU Image - https://towardsdatascience.com/grus-and-lstm-s-741709a9b9b1
- CNN Architecture Inspiration - https://arxiv.org/pdf/1408.5882.pdf