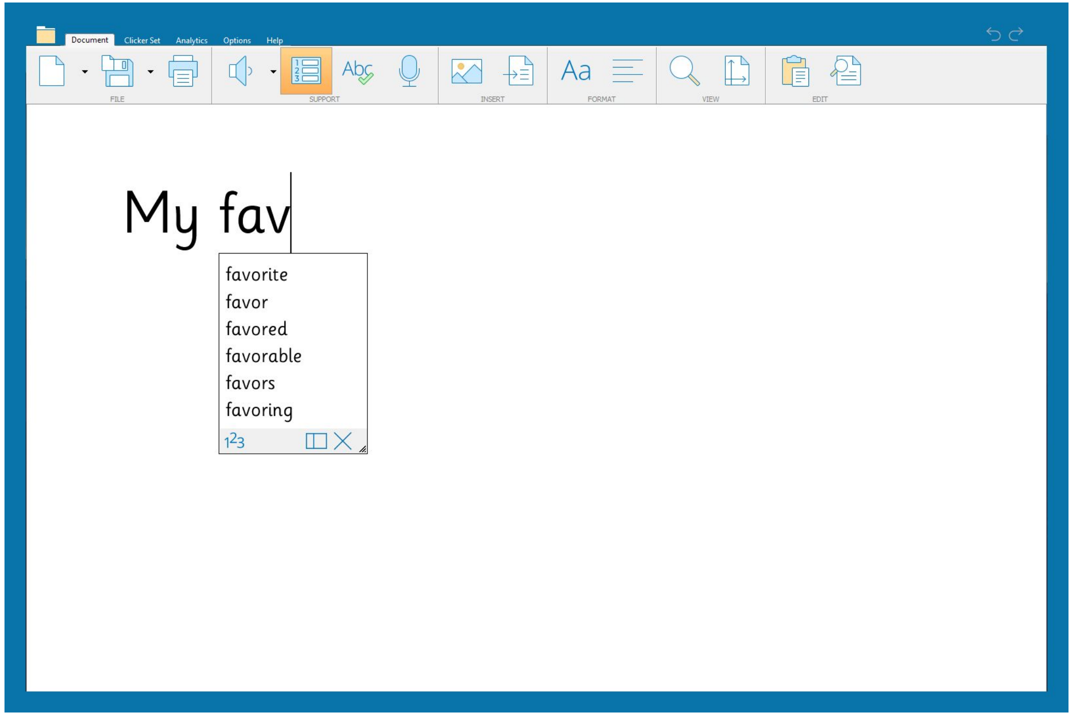
Word Predictor

Adam Ezzaim - Houcine Hassani Idrissi - Kaan Uçar

Task : Predict next word in a sequence



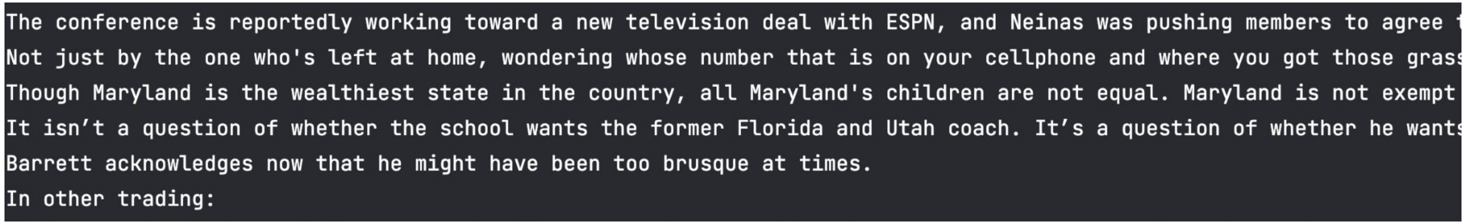
My fav

favorite
favor
favored
favorable
favors
favoring

image source

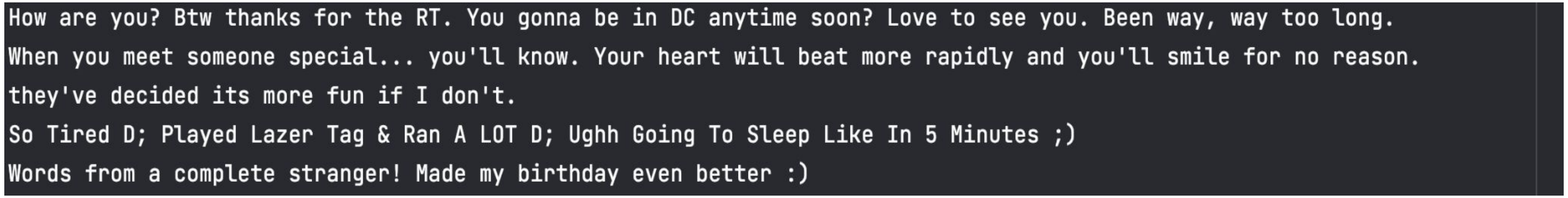
Data: Two sets

- Eng news: 1M lines - 34M words



```
The conference is reportedly working toward a new television deal with ESPN, and Neinas was pushing members to agree t
Not just by the one who's left at home, wondering whose number that is on your cellphone and where you got those grass
Though Maryland is the wealthiest state in the country, all Maryland's children are not equal. Maryland is not exempt
It isn't a question of whether the school wants the former Florida and Utah coach. It's a question of whether he wants
Barrett acknowledges now that he might have been too brusque at times.
In other trading:
```

- Eng Twitter: 2M lines - 30M words



```
How are you? Btw thanks for the RT. You gonna be in DC anytime soon? Love to see you. Been way, way too long.
When you meet someone special... you'll know. Your heart will beat more rapidly and you'll smile for no reason.
they've decided its more fun if I don't.
So Tired D; Played Lazer Tag & Ran A LOT D; Ughh Going To Sleep Like In 5 Minutes ;)
Words from a complete stranger! Made my birthday even better :)
```

- Random split : 90% train - 10% test at paragraph level

Background : N-gram method (trigram)

N-gram model takes the n − 1 preceding words into account:

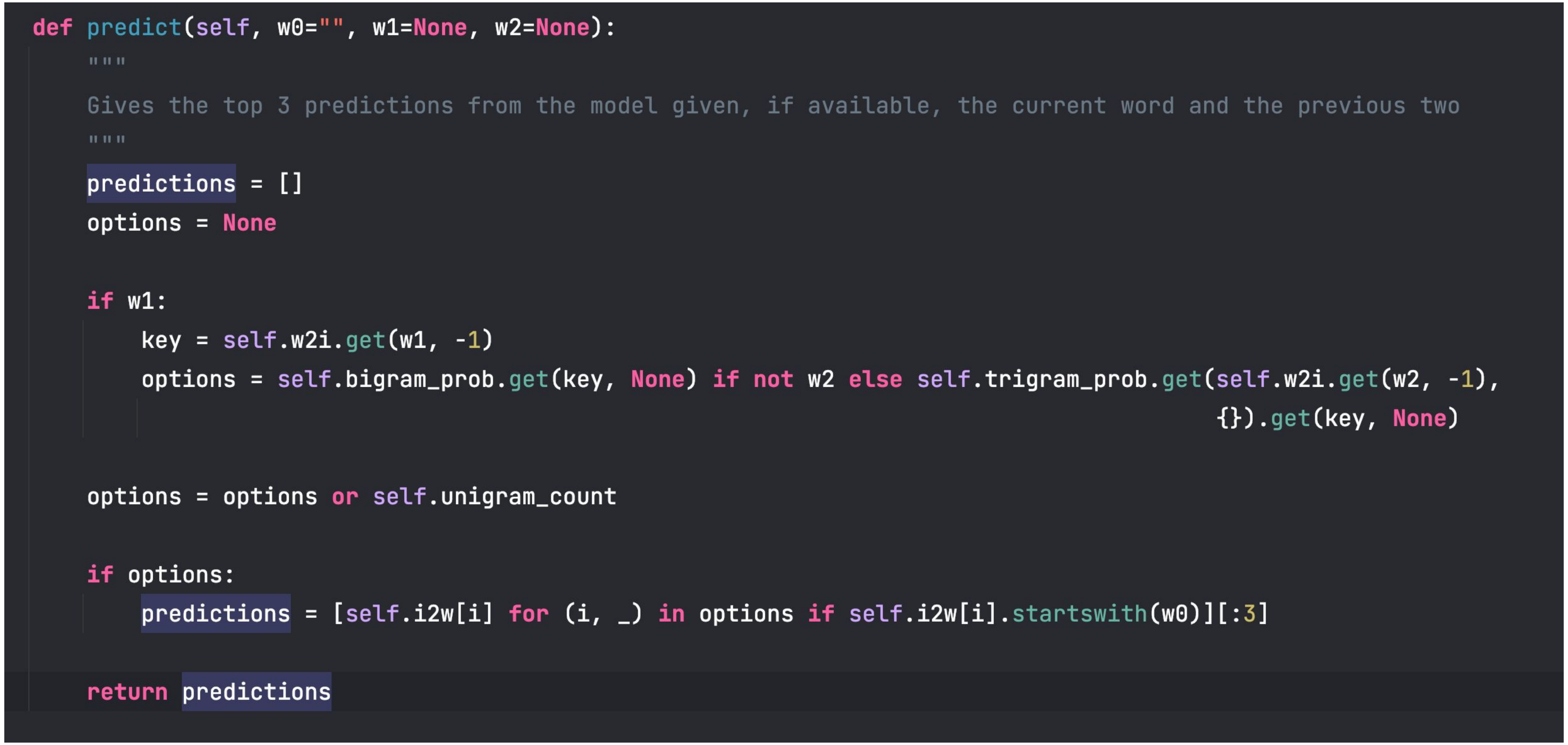
$$for\ n = 3 : \ P(w_i|w_1 \ldots w_{i-1}) = P(w_i|w_{i-n+1} \ldots w_{i-1}) = P(w_i|w_{i-2}w_{i-1})$$

Count word strings in a text corpus, compute fraction: $P(w_i|w_{i-2}w_{i-1}) = \frac{c(w_{i-2},w_{i-1},w_i)}{c(w_{i-2},w_{i-1})}$.

We add Laplace smoothing to ensure that no N-gram has a zero probability even if it was unseen in the training data :

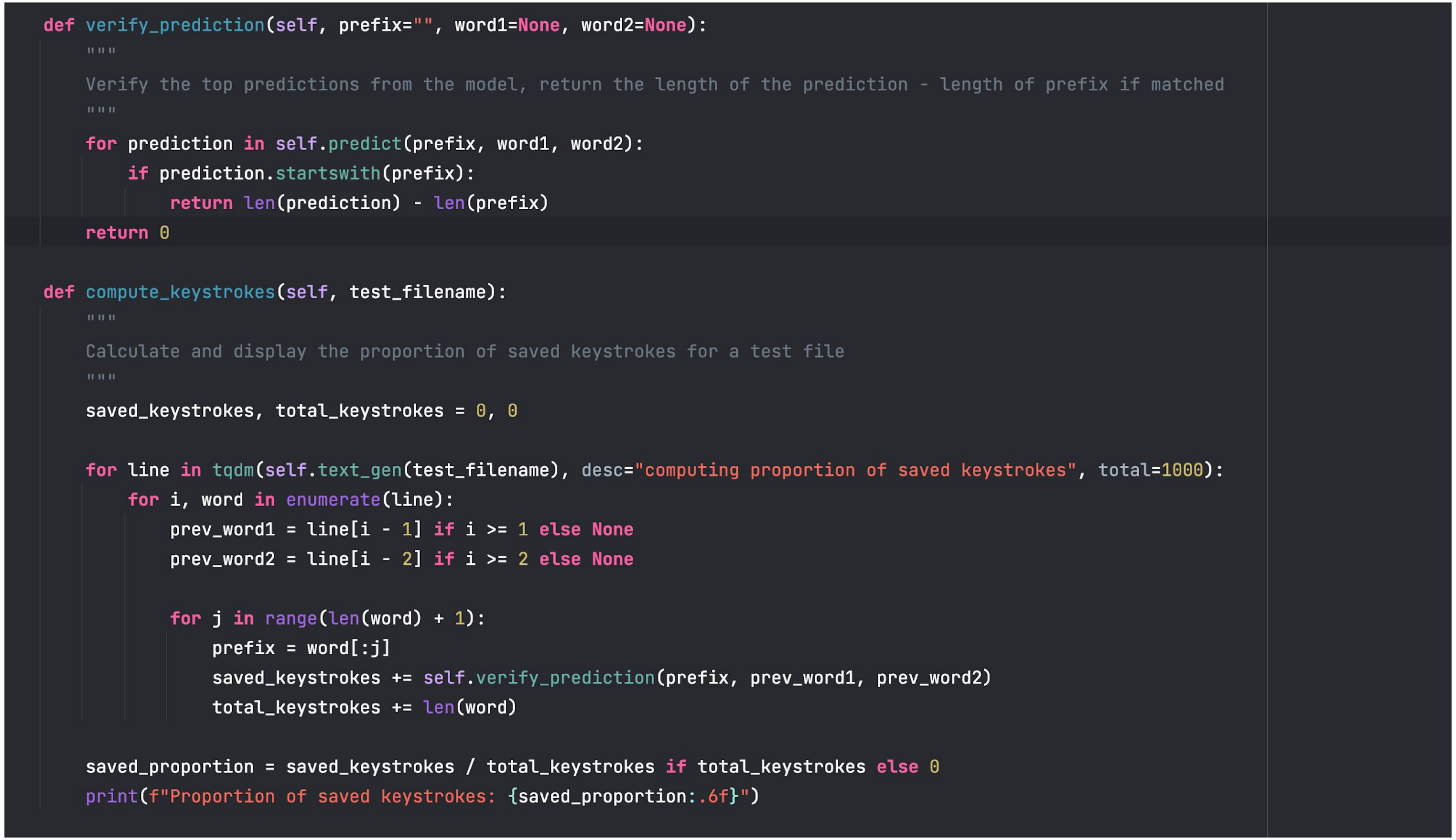
$$P(w_i|w_{i-3}w_{i-2}w_{i-1}) = \frac{c(w_{i-2}, w_{i-1}, w_i) + 1}{c(w_{i-2}, w_{i-1}) + V}$$

Implementation : prediction

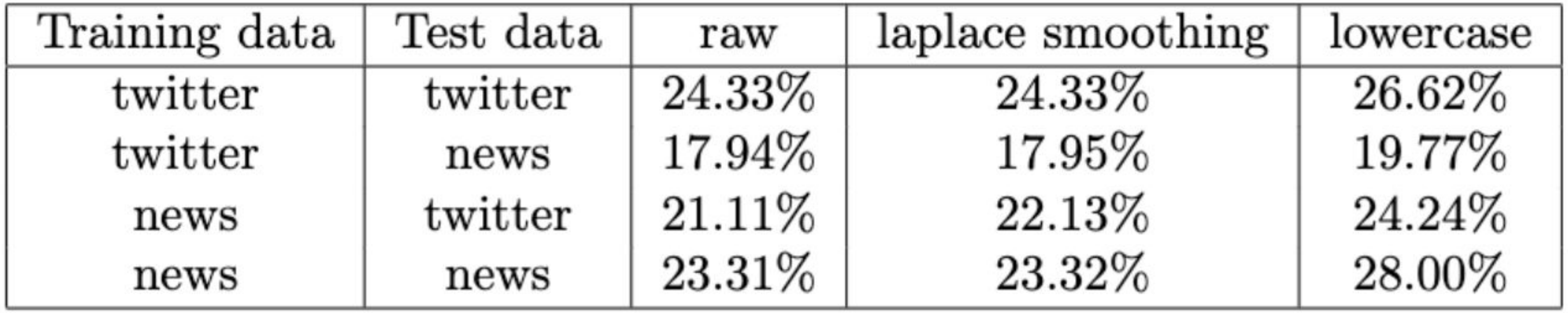


```python
def predict(self, w0="", w1=None, w2=None):
    """
    Gives the top 3 predictions from the model given, if available, the current word and the previous two
    """
    predictions = []
    options = None

    if w1:
        key = self.w2i.get(w1, -1)
        options = self.bigram_prob.get(key, None) if not w2 else self.trigram_prob.get(self.w2i.get(w2, -1),
                                                                                       {}).get(key, None)

    options = options or self.unigram_count

    if options:
        predictions = [self.i2w[i] for (i, _) in options if self.i2w[i].startswith(w0)][:3]

    return predictions
```

Implementation : test and evaluation


```python
def verify_prediction(self, prefix="", word1=None, word2=None):
    """
    Verify the top predictions from the model, return the length of the prediction - length of prefix if matched
    """
    for prediction in self.predict(prefix, word1, word2):
        if prediction.startswith(prefix):
            return len(prediction) - len(prefix)
    return 0

def compute_keystrokes(self, test_filename):
    """
    Calculate and display the proportion of saved keystrokes for a test file
    """
    saved_keystrokes, total_keystrokes = 0, 0

    for line in tqdm(self.text_gen(test_filename), desc="computing proportion of saved keystrokes", total=1000):
        for i, word in enumerate(line):
            prev_word1 = line[i - 1] if i >= 1 else None
            prev_word2 = line[i - 2] if i >= 2 else None

            for j in range(len(word) + 1):
                prefix = word[:j]
                saved_keystrokes += self.verify_prediction(prefix, prev_word1, prev_word2)
                total_keystrokes += len(word)

    saved_proportion = saved_keystrokes / total_keystrokes if total_keystrokes else 0
    print(f"Proportion of saved keystrokes: {saved_proportion:.6f}")
```

Results : trigram method

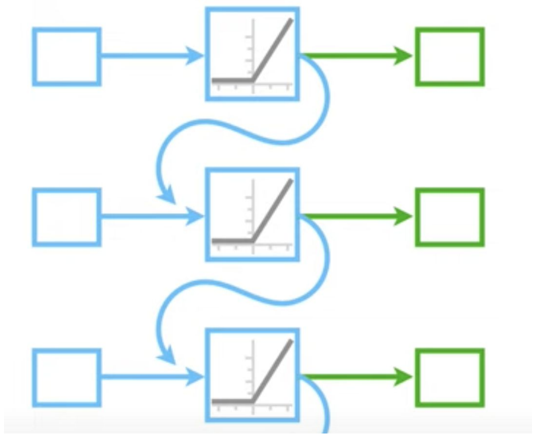


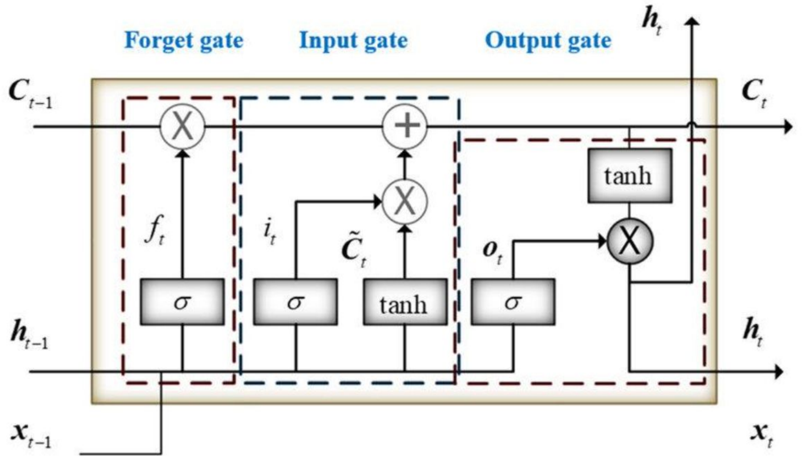
Training data	Test data	raw	laplace smoothing	lowercase
twitter	twitter	24.33%	24.33%	26.62%
twitter	news	17.94%	17.95%	19.77%
news	twitter	21.11%	22.13%	24.24%
news	news	23.31%	23.32%	28.00%

Table 1: Proportion of saved keystrokes using different train/test data and hyperparameter combinations

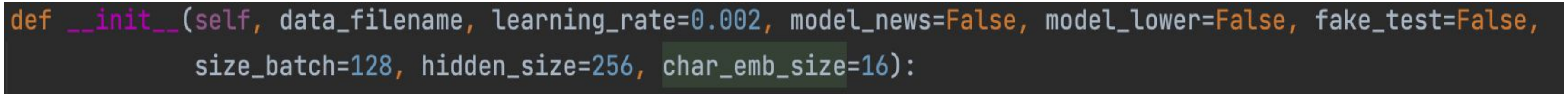
Background : Neural Network method

- Classic feedforward network limited when dealing with varying length of sequences
- RNN uses feedback, can be seen as NN with multiple inputs
- LSTM for the vanishing gradient problem

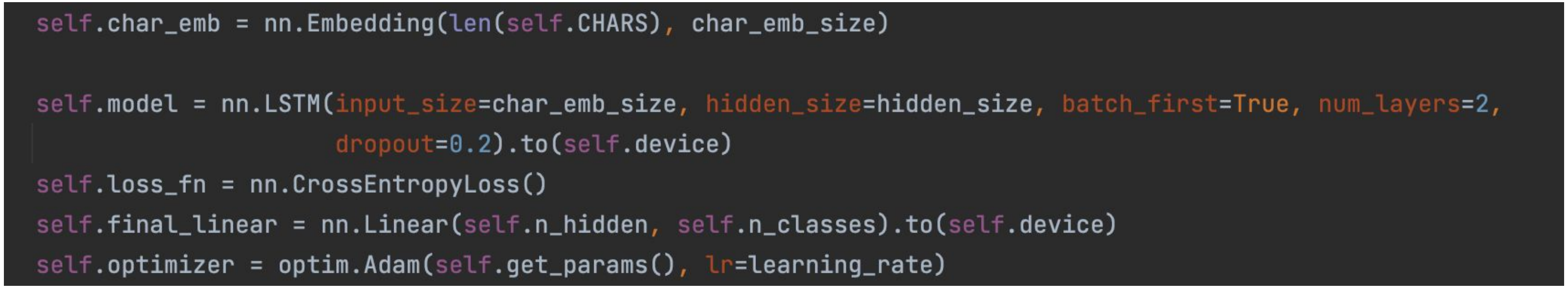




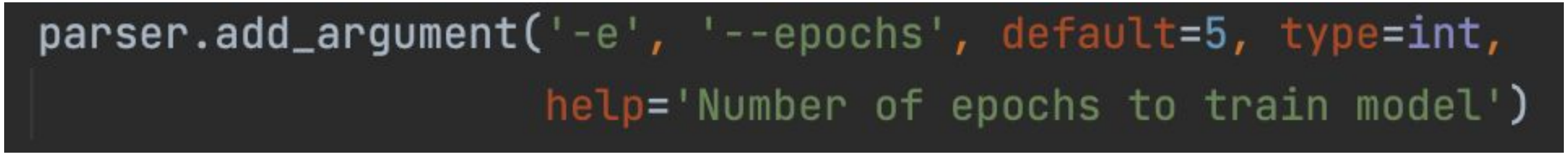
Implementation : using PyTorch’s LSTM class



```
def __init__(self, data_filename, learning_rate=0.002, model_news=False, model_lower=False, fake_test=False,
             size_batch=128, hidden_size=256, char_emb_size=16):
```

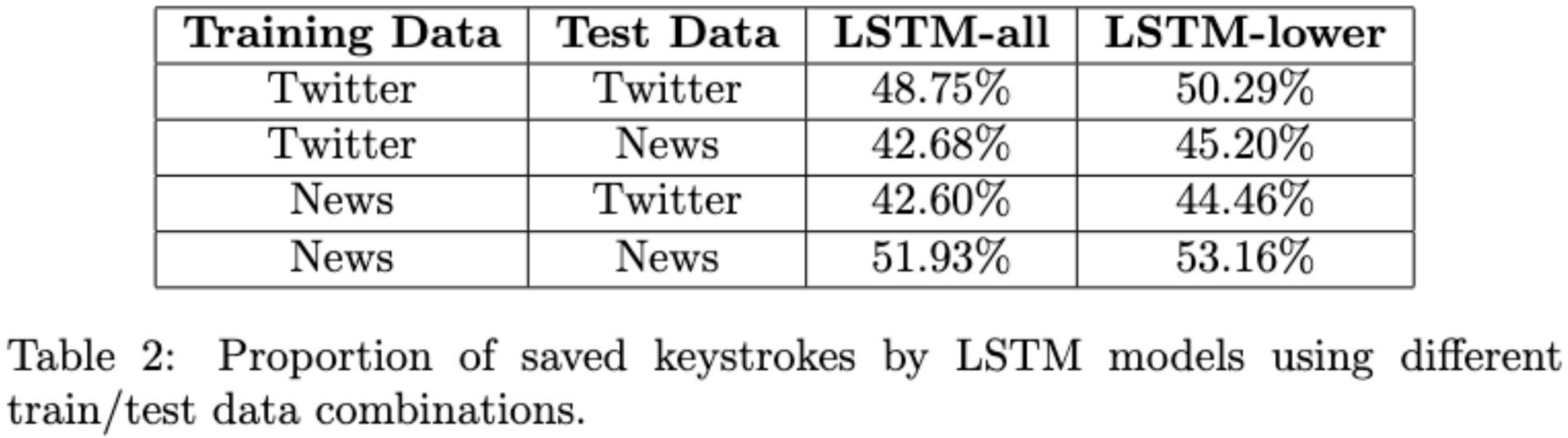


```
self.char_emb = nn.Embedding(len(self.CHARS), char_emb_size)

self.model = nn.LSTM(input_size=char_emb_size, hidden_size=hidden_size, batch_first=True, num_layers=2,
                     dropout=0.2).to(self.device)
self.loss_fn = nn.CrossEntropyLoss()
self.final_linear = nn.Linear(self.n_hidden, self.n_classes).to(self.device)
self.optimizer = optim.Adam(self.get_params(), lr=learning_rate)
```



```
parser.add_argument('-e', '--epochs', default=5, type=int,
                    help='Number of epochs to train model')
```

Results: LSTM



Training Data	Test Data	LSTM-all	LSTM-lower
Twitter	Twitter	48.75%	50.29%
Twitter	News	42.68%	45.20%
News	Twitter	42.60%	44.46%
News	News	51.93%	53.16%

Table 2: Proportion of saved keystrokes by LSTM models using different train/test data combinations.

Conclusion

- Neural Networks seem better suited for the word prediction task