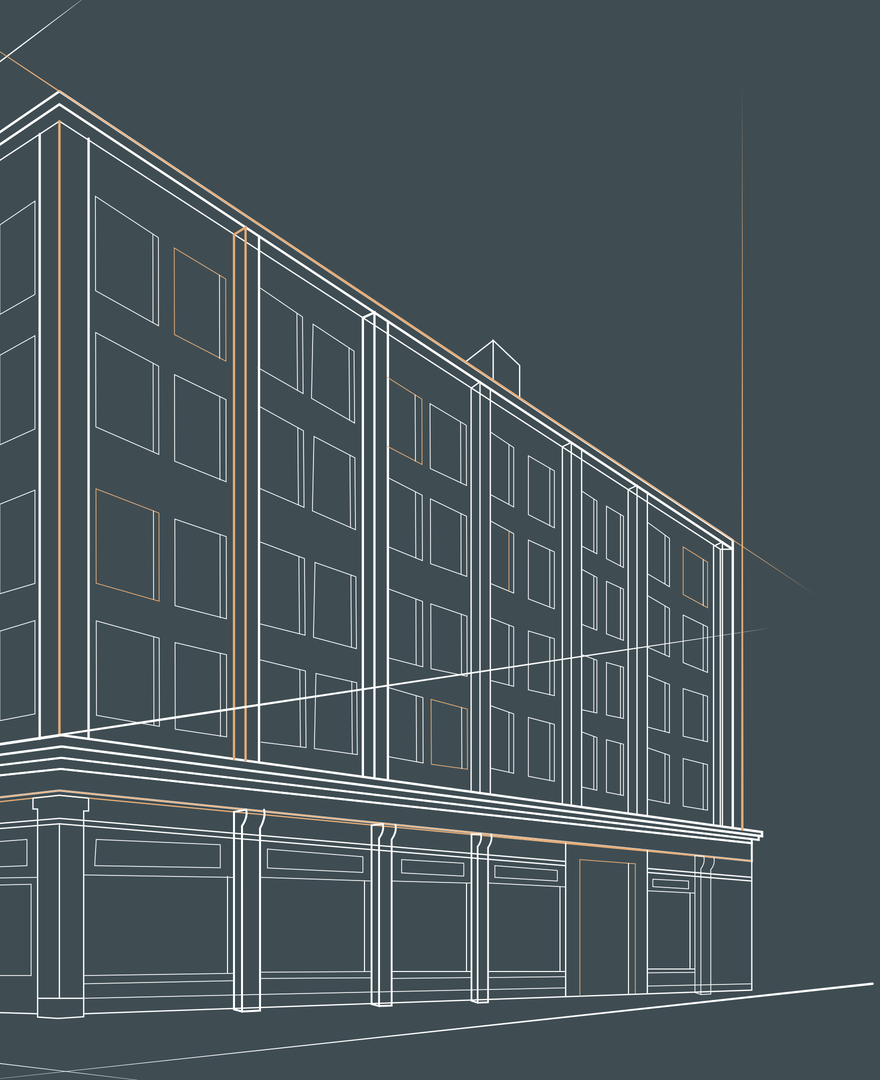
Generating Sequences With Recurrent Neural Networks

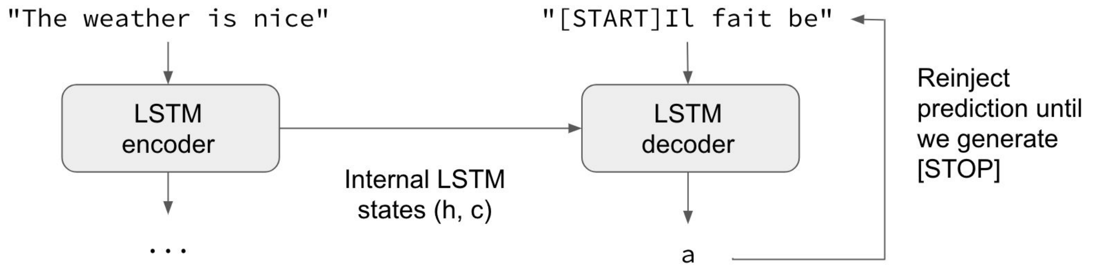
Dealers choice:
Sanchit Arora (2019101047)
Jatin Gupta (2021201048)
Sashank S (2021701038)



01

Introduction & Proposal

Text Generation


"The weather is nice"
LSTM encoder
...
Internal LSTM states (h, c)
"[START]Il fait be"
LSTM decoder
a
Reinject prediction until we generate [STOP]

Problem Statement

Long Short-term Memory recurrent neural networks can be used to generate complex sequences with long-range structure, simply by predicting one data point at a time.

RNNs can be trained for sequence generation by processing real data sequences one step at a time and predicting what comes next.

Standard RNNs are unable to store information about past inputs for very long. The problem is that if the network's predictions are only based on the last few inputs, and these inputs were themselves predicted by the network, it has little opportunity to recover from past mistakes.

Long Short-term Memory (LSTM) is an RNN architecture designed to be better at storing and accessing information than standard RNNs.

The task we wish to demonstrate is that LSTM can use its memory to generate complex, realistic sequences containing long-range structure.

Goals

- Define a ‘deep’ RNN composed of stacked LSTM layers, and apply the prediction network to text from the Penn Treebank and Hutter Prize Wikipedia datasets. Use the network to predict one character at a time as well as predict one word at a time.

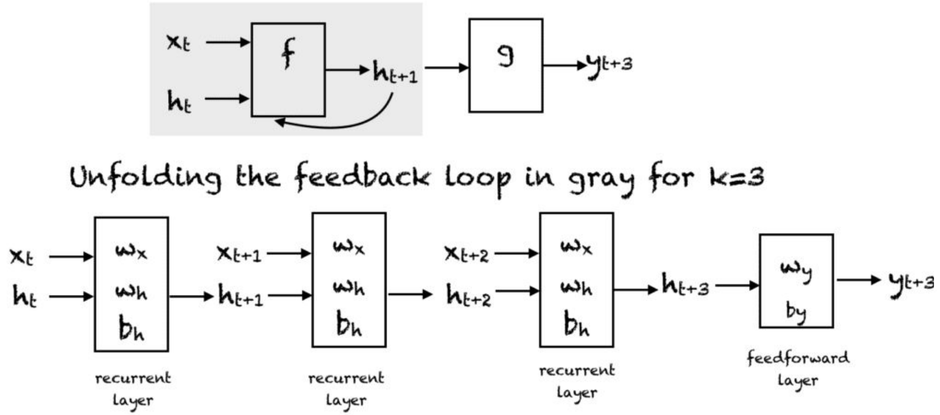
- Generate samples of text to show the network’s ability to model long-range dependencies.

- Apply the prediction network to real-valued data, IAM Online Handwriting Database, through the use of a mixture density output layer.

- Build a synthesis network that is trained on the IAM database, and use it to generate cursive handwriting samples given a user input text.

Recurrent Neural Network

Neural Network where the output from the previous step is fed as input to the current step.



$$h_t^1 = \mathcal{H}\left(W_{ih^1}x_t + W_{h^1h^1}h_{t-1}^1 + b_h^1\right) \tag{1}$$

$$h_t^n = \mathcal{H}\left(W_{ih^n}x_t + W_{h^{n-1}h^n}h_t^{n-1} + W_{h^nh^n}h_{t-1}^n + b_h^n\right) \tag{2}$$

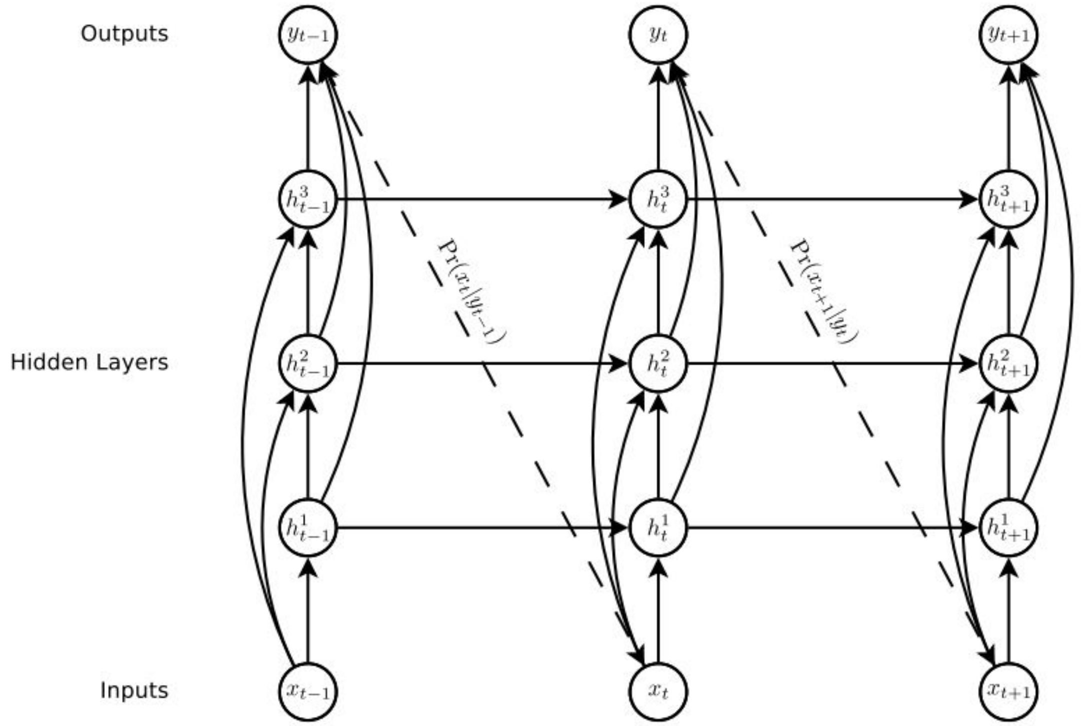
$$\hat{y}_t = b_y + \sum_{n=1}^{N} W_{h^n y} h_t^n \tag{3}$$

$$y_t = \mathcal{Y}(\hat{y}_t) \tag{4}$$

$$\Pr(\mathbf{x}) = \prod_{t=1}^{T} \Pr(x_{t+1}|y_t) \tag{5}$$

$$\mathcal{L}(\mathbf{x}) = -\sum_{t=1}^{T} \log \Pr(x_{t+1}|y_t) \tag{6}$$

Network Architecture



Long Short Term Memory

- Long Short-Term Memory (LSTMs) uses purpose-built memory cells to store information, is better at finding and exploiting long range dependencies in the data.

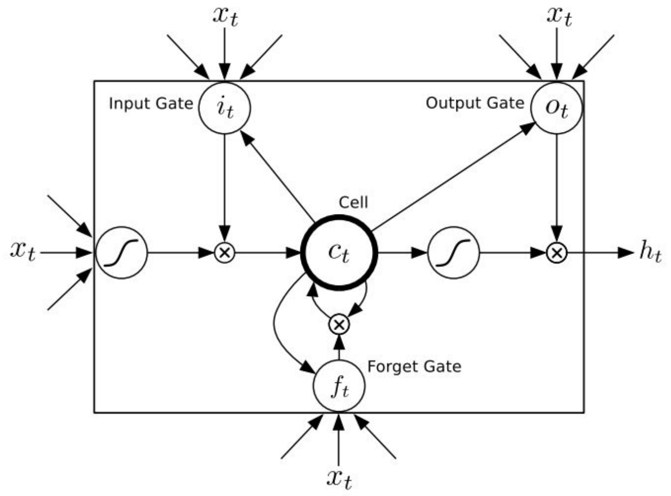
$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \quad (7)$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \quad (8)$$

$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \quad (9)$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \quad (10)$$

$$h_t = o_t \tanh(c_t) \quad (11)$$



Text Prediction

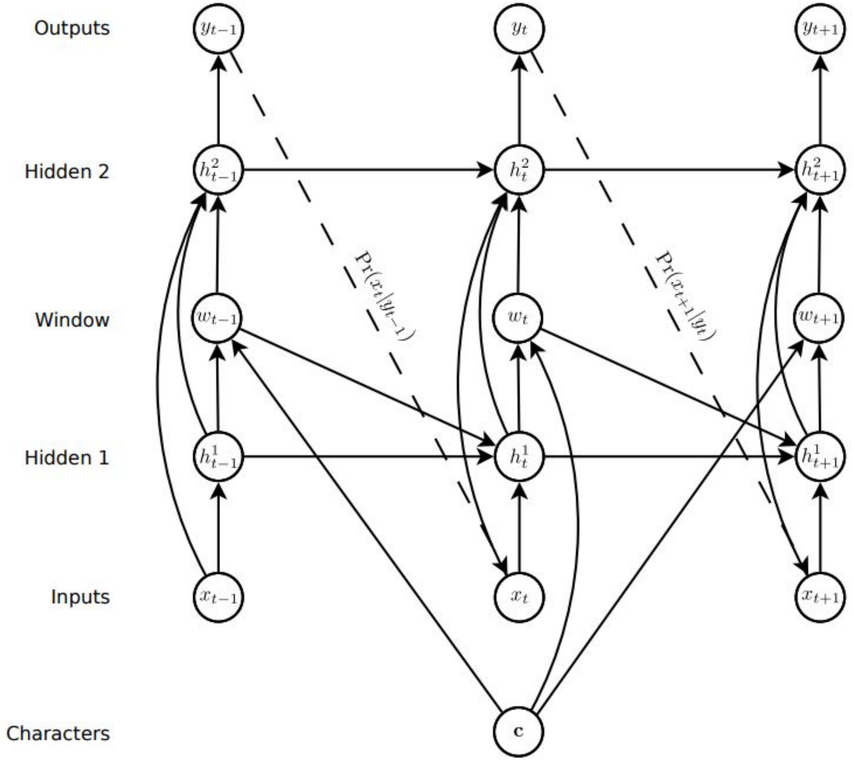
- Text data is discrete, and is typically presented to neural networks using 'onehot' input vectors.
- That is, if there are K text classes in total, and class k is fed in at time t, then xt is a length K vector whose entries are all zero except for the k th, which is one.
- Text Prediction occurs in two levels : word and character
- Text prediction (usually referred to as language modelling) when performed at the word level, has K representing the number of words in the dictionary.
- For realistic tasks, the number of words exceed 100,000. Having so many classes demands a huge amount of training data to adequately cover the possible contexts for the words.
- Word-level models are also not applicable to text data containing non-word strings, such as multi-digit numbers or web addresses.
- Predicting one character at a time is more interesting from the perspective of sequence generation, because it allows the network to invent novel words and strings.

Handwriting Prediction

- Generate convincing real-valued sequences, using online handwriting data (the writing is recorded as a sequence of pen-tip locations).
- Determine a predictive distribution suitable for real-valued inputs.
- The idea of mixture density networks is to use the outputs of a neural network to parameterise a mixture distribution.
- A subset of the outputs are used to define the mixture weights, while the remaining outputs are used to parameterise the individual mixture components.
- Mixture density outputs can also be used with RNNs. In this case the output distribution is conditioned not only on the current input, but on the history of previous inputs.
- Each input vector xt consists of a real-valued pair x1, x2 that defines the pen offset from the previous input, along with a binary x3 that has value 1 if the vector ends a stroke and value 0 otherwise.
- A mixture of bivariate Gaussians can be used to predict x1 and x2, while a Bernoulli distribution can be used for x3.

Synthesis Network

- Similar to the prediction network, with an added input from the character sequence, mediated by the window layer.



Expected Results

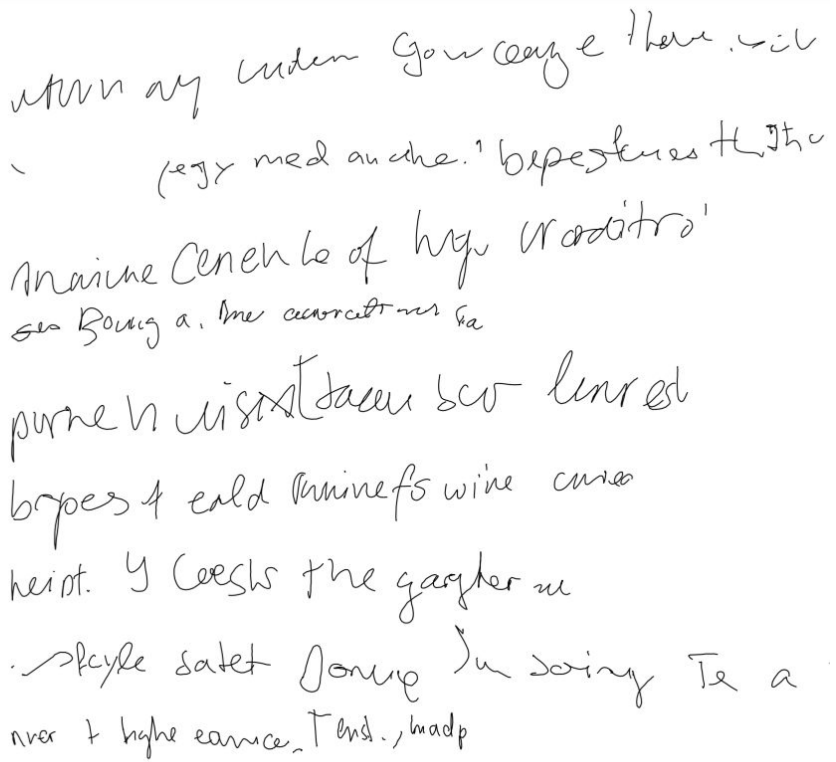


Fig. Online handwriting samples generated by the prediction network.

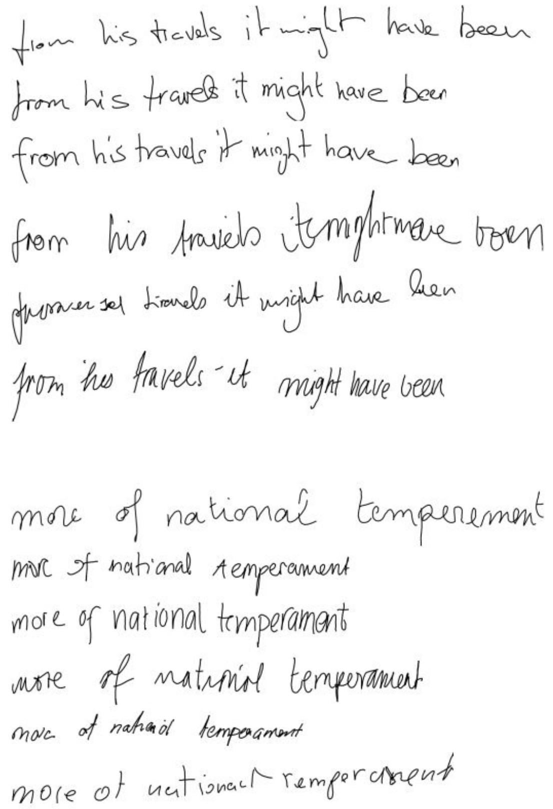
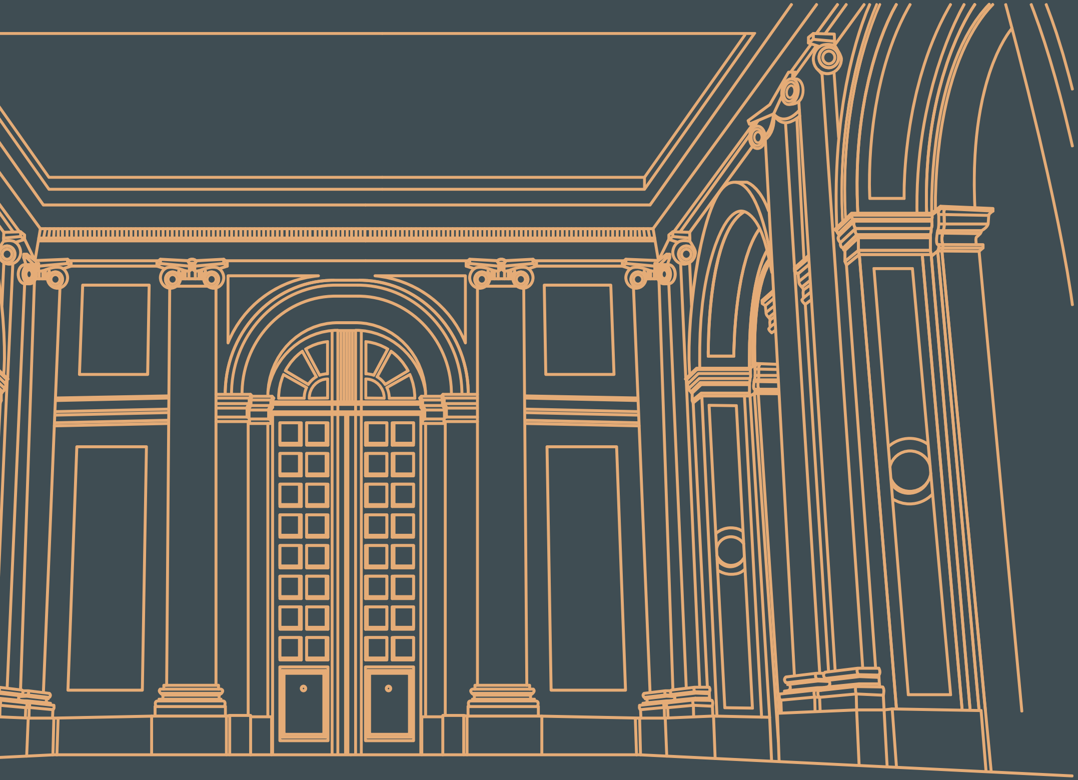

from his travels it might have been
more of national temperament

Fig. The top line in each block is real, the rest are unbiased samples from the synthesis network.

Dataset

- The first set of text prediction experiments focuses on the Penn Treebank portion of the Wall Street Journal corpus.
    - Although a relatively small text corpus (a little over a million words in total), the Penn Treebank data is widely used as a language modelling benchmark. The training set contains 930,000 words, the validation set contains 74,000 words and the test set contains 82,000 words.
- The second set of focuses on the Hutter Wikipedia Dataset.
    - Wikipedia data is interesting from a sequence generation perspective because 8 it contains not only a huge range of dictionary words, but also many character sequences that would not be included in text corpora traditionally used for language modelling.
- The handwriting prediction and synthesis focuses on the IAM online handwriting database (IAM-OnDB).
    - IAM-OnDB consists of handwritten lines collected from 221 different writers using a ‘smart whiteboard’. The writers were asked to write forms from the Lancaster-Oslo-Bergen text corpus, and the position of their pen was tracked using an infra-red device in the corner of the board.



02

Project Expectations

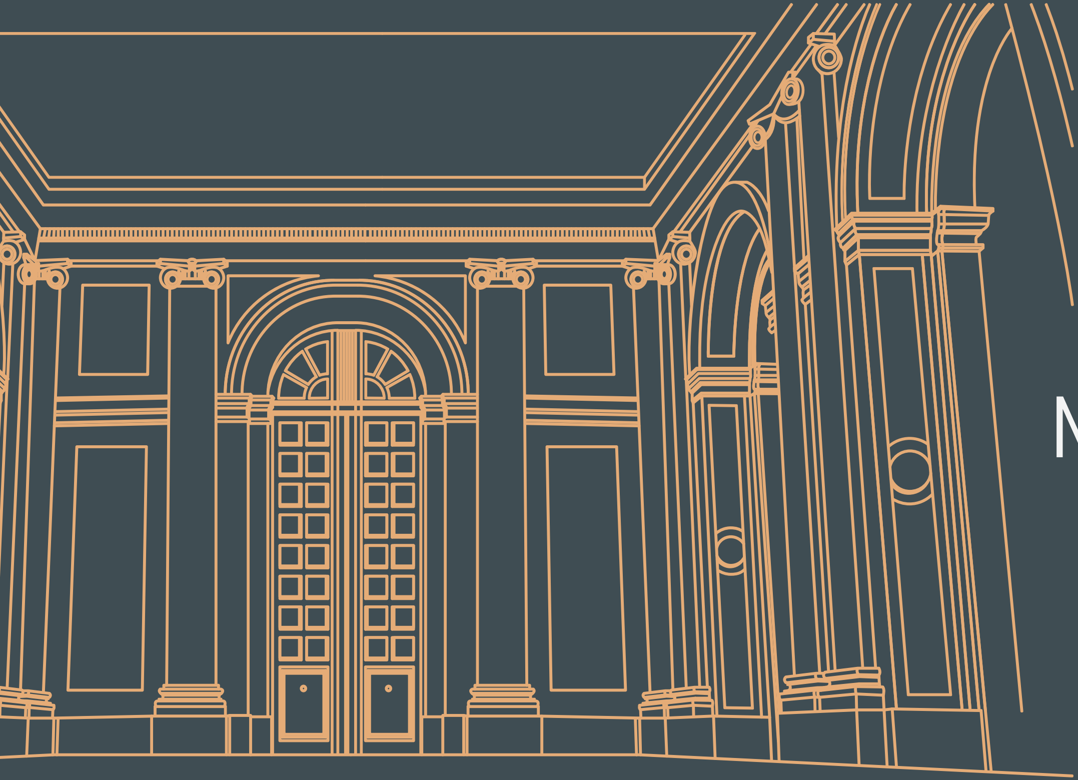
Expected Deliverables

The overall expectation from this project is the implementation of sequence models for prediction and generation of text. To outline the tasks in milestones, we shall follow the below deliverables:

1. Build a prediction network for Penn Treebank dataset and perform character and word level experiments.
2. Build a prediction network for Hutter prize Wiki dataset and generate a sample wikipedia document.
3. Build a prediction network can be applied to real-valued data through the use of a mixture density output layer. Generate handwriting samples from the IAM Online Handwriting Database.
4. Extend the prediction network that allows it to condition its outputs on a short annotation sequence whose alignment with the predictions is unknown, making it suitable for handwriting synthesis.

Expected Timeline

- **Week 1**
  - Read the paper thoroughly and get detailed understanding of the idea.
  - Download the Penn Tree bank, Hutter Prize Wikipedia and IAM-OnDB dataset on servers.
  - Prepare preprocessed datasets from the instructions of paper.
  - Apply additional processing on dataset and prepare PyTorch dataloaders for character level and word level predictions.
- **Week 2**
  - Prepare the RNN encoder models and setup structure for entire experiments.
  - Write complete encoder-decoder architecture in PyTorch.
  - Prepare loss functions and final training pipeline.
  - Run the character level and word level experiments for Penn-Treebank and Hutter Prize Dataset.
- **Week 3**
  - Setup the IAM-OnDB. Dataset.
  - Set up Dataloader to load and view the dataset.
  - Build a prediction model that sequentially generates a predictive probability distribution, given a word and surrounding context.
  - Train the network to generate accurate distributions of the future given the historical past, sample from the probability distribution to generate our handwriting sample.
- **Week 4**
  - Build synthesis model with an additional window layer.
  - Generate samples conditionally and unconditionally.



03

Mid-Evaluation Progress

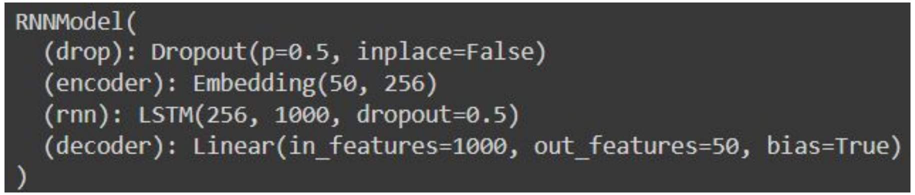
Penn Treebank Dataset

Character Level	Word Level
● Number of tokens	● Number of tokens
Train: 5017483	Train: 929589
Valid: 393043	Valid: 73760
Test: 442424	Test: 82430

The vocabulary is limited to 10,000 words, with all other words mapped to a special ‘unknown word’ token. The end-of-sentence token was included in the input sequences, and was counted in the sequence loss.

Encoder-Decoder Network

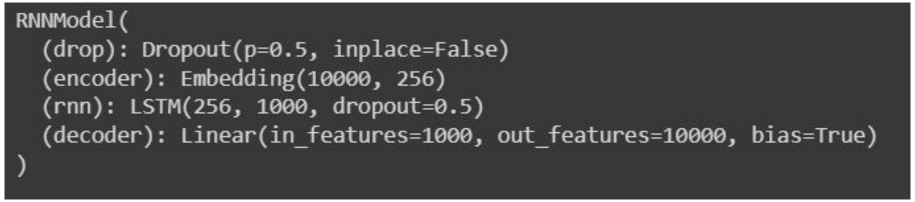
Character Level



```
RNNModel(
  (drop): Dropout(p=0.5, inplace=False)
  (encoder): Embedding(50, 256)
  (rnn): LSTM(256, 1000, dropout=0.5)
  (decoder): Linear(in_features=1000, out_features=50, bias=True)
)
```

- Single hidden layer with 1000 LSTM units.
- The input and output layers were size 50, giving approximately 4.3M weights in total

Word Level



```
RNNModel(
  (drop): Dropout(p=0.5, inplace=False)
  (encoder): Embedding(10000, 256)
  (rnn): LSTM(256, 1000, dropout=0.5)
  (decoder): Linear(in_features=1000, out_features=10000, bias=True)
)
```

- Single hidden layer with 1000 LSTM units.
- 10,000 inputs and outputs and around 54M weights.

- All networks were trained with stochastic gradient descent, using a learn rate of 0.0001 and a momentum of 0.99.
- The LSTM derivates were clipped in the range [-1, 1]

Regularization

Method 1: Weight noise with a std. deviation of 0.075 applied to the network weights at the start of each training sequence.

Method 2: Adaptive weight noise, where the variance of the noise is learned along with the weights using a Variational inference loss function.

When weight noise was used, the network was initialised with the final weights of the unregularised network. Similarly, when adaptive weight noise was used, the weights were initialised with those of the network trained with weight noise.

Evaluation

Perplexity (PPL): Tells us how well our models are at predicting the next word in a text.

$$\mathrm{PPL} = e^{L_e \cdot \overline{n}}$$

We calculate it by using the output of our loss function, the negative log likelihood Le. n is the average number of atomic units that the words in the test set are split into.

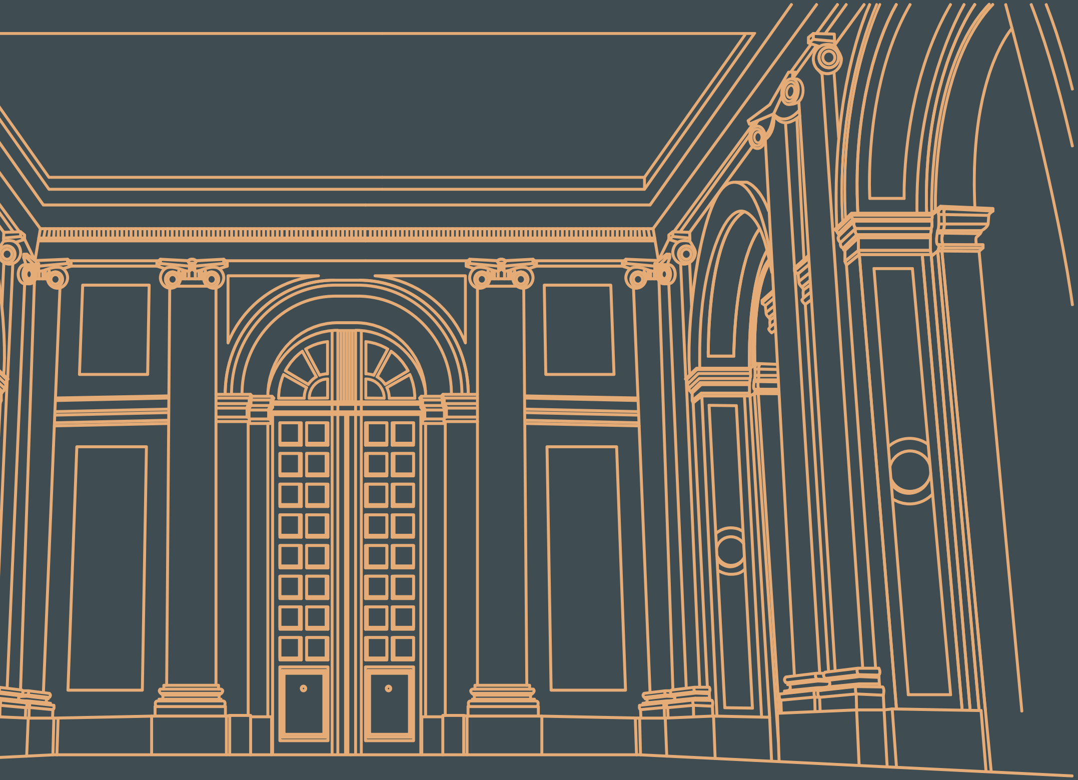
Bits-per-character (BPC): Describes the average character-wise cross-entropy. For the character-based model this is equal to the negative log likelihood L2 to the base 2. For the word-based models, we need to break this value down to apply to single characters.

$$\mathrm{BPC} = \frac{L_2 \cdot \overline{n}}{\overline{w} + 1}$$

Where w is the average word-length in the testset.

Results

Input	Regularization	BPC	Perplexity	Epochs
Char	None	4.343	20.29	10
Char	Weight Noise	4.426	21.50	25
Char	Adaptive Weight Noise	4.412	21.28	10
Word	None	9.644	800.29	10
Word	Weight Noise	9.572	761.17	25



04

Future Steps

Next Steps

- As per the tasks outlined in the expected deliverables and the timeline, the next tasks comprise of preparing the end-to-end training pipeline for the Wiki dataset.
- Additionally, process the IAM-OnDB. Dataset.
- Build a prediction model that sequentially generates a predictive probability distribution, given a word and surrounding context.
- Train the network to generate accurate distributions of the future given the historical past, sample from the probability distribution to generate our handwriting sample.
- Build synthesis model with an additional window layer and generate samples conditionally and unconditionally.

Thanks!

...