GROUP 7 ML FINAL PROJECT: SPELLING CORRECTION

Clarissa Cheam, Tianchang Li, Ayushi Mishra and Geethanjali Vasudevan

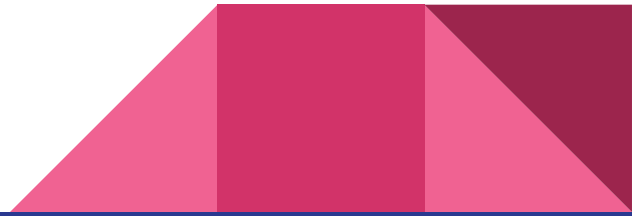
Task description and Motivation:

Task:

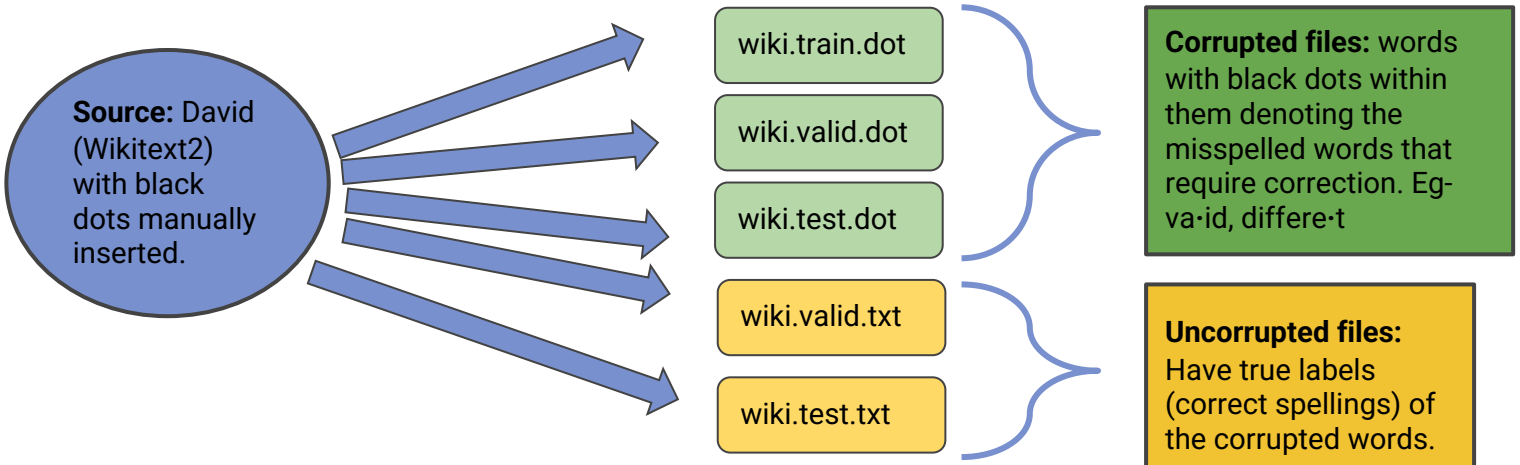
The task we intend to address involves **predicting the correct spelling** of the identified misspelled words based on the data provided.

Motivation:

- Importance of written communication -> **clearly getting our message across**.
- Grammarly - impactful, meaningful and mistake-free
- Downstream applications being beneficial like **smart typing in iphones.**
- Custom-specific words, old-english conversion, etc



Dataset description and source:



Note- that the black dot is actually ch(183) and is therefore different from full stops.

For illustration purposes here's how our wiki.train.dot file looks like-

```
 = Valkyria Chronicles III =

 Senjɔç no Valkyria 3 : <unk> Chronicles ( Japanese :
Êà¶Â†¥„ÅÆ„É¥„Ç°„É´„Ç≠„É•„É™„Ç¢3 , lit . Valkyria of the Battlefield 3 ) ,
commonly referred to as Valkyria Chronicles III outside Japan , is a
tactical role @-@ playing video game developed by Sega and Media.Vision
for the PlayStation Portable . Released in January 2011 in Japan , it is
the third game in the Valkyria series . <unk> the same fu¬∑ion of
tactical and real @-@ time gameplay as its predecessors , the story runs
parallel to the first game and follows the " Nameless " , a penal
military unit serving the nation of Gallia during the Second Europan War
who perform secret black operations and are pitted against the Imperial
unit " <unk> Raven " .
 The game began development in 2010 , carrying over a large portion of
the work done on Valkyria Chronicles II . While it retained the standard
```

Dataset summary statistics:

Preprocessing steps:

- We removed **punctuation** and **new lines** within the dataset.
- We then removed **custom created stopwords** like unk, usp, etc. This was done to remove unnecessary information like html related keywords present in the dataset in order to get the correct accuracy.
- Finally the capitalization within the dataset was removed by lowercasing everything.

Summary statistics of the data:

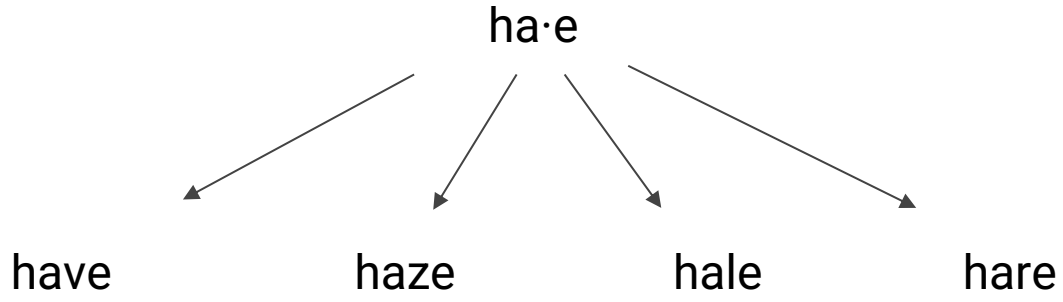
S.No.	Data type	File name	No. of lines
1	Train data	wiki.train.dot	10780437
2	Valid data	wiki.valid.dot	1120192
		wiki.valid.txt	1120192
3	Test data	wiki.test.dot	1255018
		wiki.test.txt	1255018

In our corrupted dataset (files with dot extension), **around 1% of words have the black dot** (corrupted words) in them.

Model Specification (non-NN model 1):

1. **Character-based Naive Bayes**

$$argmax(c) \in candidates\ P(c)\ P(w|c)$$



Model Specification (non-NN model 1-contd.):

1. **Character-Naive Bayes (contd.)**

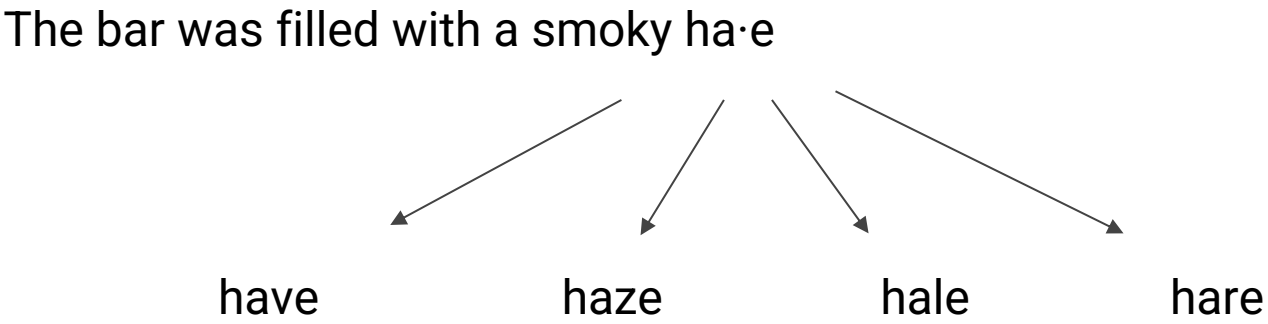
$$argmax(c) \in candidates\ P(c)\ P(w|c)$$

Specifically,

- *Candidate Model:* c ∈ candidates, if c is an uncorrupted word with the same length as the corrupted word w and matches the rest of the letter combination (around the black dot).
- *Selection Mechanism:* Since random 1% of the words are corrupted, P(*w*|*c*) is equal among all candidates. Thus, we chose the candidate with the highest term frequency P(*c*).
- *Language Model:* P(c)
  The probability that c appears as a word in the document.

Model Specification (non-NN model 2):

2. Character-based NB + n-gram context - JamSpell



Model Specification (non-NN model 2- contd.):

2. Character-based NB + n-gram context - JamSpell

Additionally depends on the joint probability of this candidate showing up together with the few words preceding it – calculated as a product of probabilities of all lower order grams

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

Each gram is calculated from.

$$P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}) = \frac{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(n-1)}, \ldots, w_{i-1})}$$

Note: to better work with the nature of JamSpell (trained on real misspelled words), we replace the black dots with the least-frequent letter, ‘q’.

Model Evaluation (non-NN models 1 & 2):

Evaluation Metrics:

- Accuracy = ***# of correctly predicted words / # of all corrupted words***
- Run time in seconds

Model	Accuracy	Runtime (sec)
Random guessing - baseline	3.85%	0
Basic Naïve Bayes	84.31%	338
JamSpell - NB + context	91.33%	764

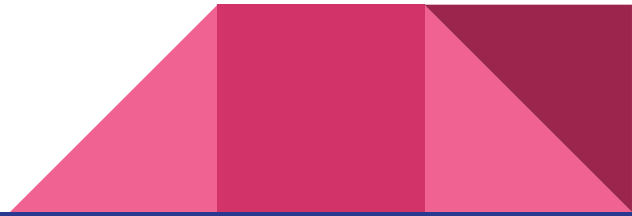
Result analysis:

- Have significantly surpassed the baseline model.
- Character-based NB took a little over 5 minutes to predict 1154 words, which is sufficient given that we manually implemented the algorithm.
- JamSpell further boosted the accuracy up by 7 percents. It doubled the run time.

Model Evaluation (non-NN models 1 & 2-contd.):

Assumptions and Limitations:

- Oversimplified the task.
- Assumed misspelled words had been perfectly identified.



Model Specification (non-NN model 3):

3. Char2vec + nearest neighbor

Fasttext char2vec:

1. **Sub-word generation**: For a word, we generate *character n-grams* and then *apply hashing to bound the memory requirements.*
2. **Skip-gram with negative sampling**: The embedding for the center word is calculated followed by *collecting negative samples randomly.*

NearestNeighbors: We implement ***unsupervised nearest neighbors***. The dataset is structured such that ***points nearby in index order are nearby in parameter space***.

For distance metrics, we will use the Euclidean metric.

$$d(x, x') = \sqrt{(x_1 - x_1')^2 + \ldots + (x_n - x_n')^2}$$

Finally, the input x gets assigned to the class with the largest probability.

$$P(y = j|X = x) = \frac{1}{K} \sum_{i \in \mathcal{A}} I(y^{(i)} = j)$$

Results and Analysis (non-NN model 3):

The char2vec model was able to overcome the problems of one-hot encoding such as the

- ***Sparse representation problem***
  This model was able to capture relationship between letters due to sub-letter training and skip-gram.
- ***Scalability issues***

Char2vec also helped with the **word2vec out-of-vocabulary** and morphology problem too.

```
words_input = ['wall·nia', 'hel·', 'line·', '·ill', '·alestro', 'pounde·','chron.cle',"temper.ture","co.t"]


wall·nia
--> ['Wallonia' 'Wallonia' 'Wallonia' 'Wallonia' 'Wallonia']
hel·
--> ['heel' 'heel' 'heel' 'heel' 'heel']
line·
--> ['line' 'line' 'line' 'line' 'line']
·ill
--> ['ill' 'ill' 'ill' 'ill' 'ill']
·alestro
--> ['alters' 'alters' 'alters' 'alerts' 'alerts']
pounde·
--> ['pounder' 'pounder' 'pounder' 'pounder' 'pounder']
chron.cle
--> ['chronicle' 'chronicle' 'chronicle' 'chronicle' 'chronicle']
temper.ture
--> ['temperature' 'temperature' 'temperature' 'temperature' 'temperature']
co.t
--> ['cost' 'cost' 'cost' 'cost' 'cost']
```

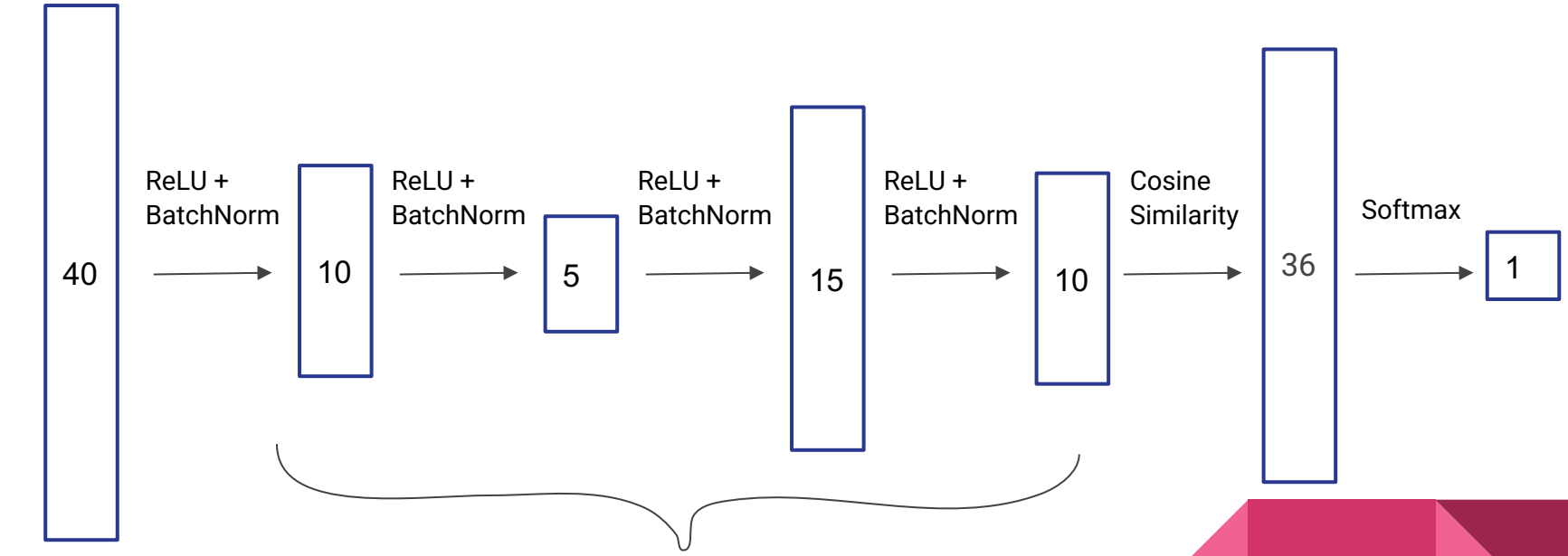
Result and Analysis (non-NN model 3- contd.)

Limitations and Assumptions:

- Char2vec model does not take into consideration the order of words in which they appear which leads to loss of syntactic and semantic understanding of the sentence.
- For the char2vec model, it was important to remove the corrupted word from the corpus, as the model could learn the corruption pattern and learn the embedding.

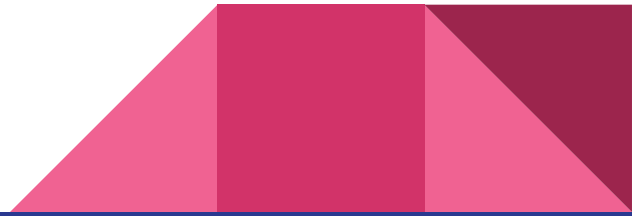
Model Specification (for Neural Networks):

1. **NN model 1: Feed Forward Model - Approach A**



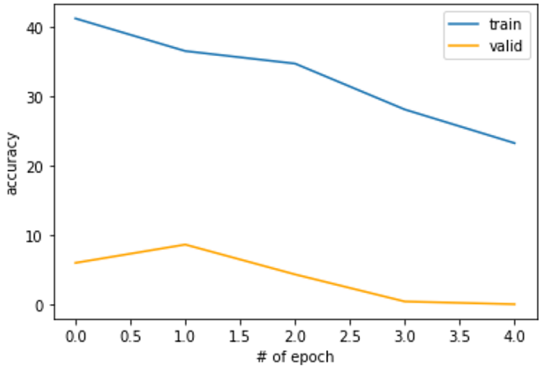
Model Specification - NN model 1 (contd.):

- Training set - wiki.valid.txt; validation set - wiki.test.dot
- Window size - 4
- Loss function - cross entropy
- Gradient descent function - Adam
- Regularization - L1
- Batch size - 256
- Learning rate - 0.01, 0.005, 0.001

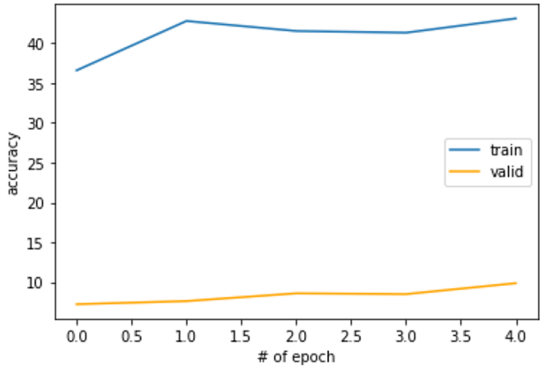


Model Evaluation - NN model 1 (Approach A):

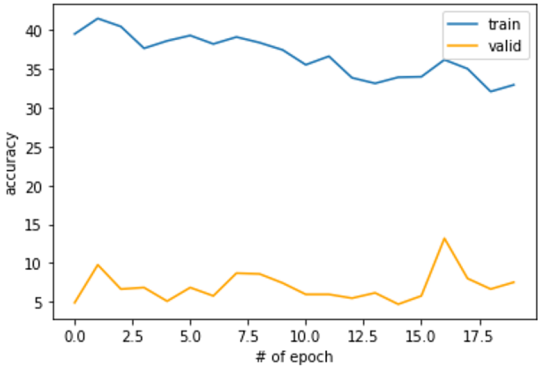
Main results:



Learning rate = 0.01, 5 epochs



Learning rate = 0.001, 5 epochs



Learning rate = 0.001, 20 epochs

Model Evaluation - NN model 1(Approach A-contd.):

Analysis of results:

- Model beats the baseline accuracy (2.77%).
- Model overshoots with a larger learning rate.
- Model might be overfitted to the training set.

Limitations:

- Low and unstable validation accuracy - may need larger training and test sets.
- Accuracy goes down in the first few epochs - may need more iterations.

Model Specification (for NN model 1- contd.):

2. NN model 1: Feed Forward Model- Approach B

Training set - wiki.training.dot; validation set - wiki.valid.dot, wiki.valid.txt

NN architecture and hyperparameters: Combinations attempted-

- **Activation Function tried:** Sigmoid, Tanh, ReLU
  - Findings: Not much difference in terms of its impact accuracies yield in training loop
- **Hidden Layers tried:** 1 layer (hidden_dim=1000), 2 layer (hidden_dim1=1000, hidden_dim2=100)
  - Findings: 2 layers observed to perform better
- **Output dimension:** 35
  - Mapped to 35 possible classes
- **Input dimension:** flatten tensor of 16x35, 64x35
  - The content of the input will be discussed the following slides
- **Loss Function:** Multi-class cross entropy
  - Weight Decay: 0.01
- **Optimizer:** Stochastic Gradient Descent
  - Learning rate: 0.01
  - Momentum: 0.8

Model Specification (for NN model 1- contd.):

Regarding feeds to loss function: Approaches attempted-

- Direct feed to loss function
- Implement cosine similarity pytorch nn module and append to the nn.Sequential block.

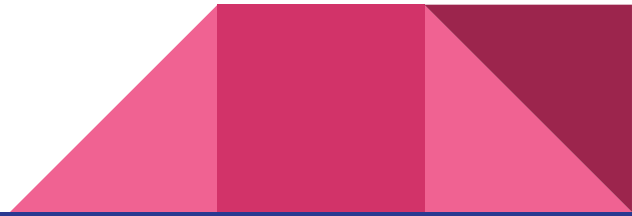
```
class CosineSimilarity(nn.Module):
    def __init__(self):
        super(CosineSimilarity, self).__init__()
        self.charset = validchar()
        self.charsetlookup = {}
        for k, v in self.charset.items():
            l = [0]*35
            l[v] = 1
            self.charsetlookup[v] = torch.Tensor(l)

    def forward(self, input: torch.Tensor) -> torch.Tensor:
        for k, v in self.charset.items():
            input[0][v] = torch.dot(input[0], self.charsetlookup[v])
        return input
```

```
self.layers = nn.Sequential(
    nn.Linear(in_dim, hidden_dim1),
    nn.ReLU(),
    nn.Linear(hidden_dim1, hidden_dim2),
    nn.ReLU(),
    nn.Linear(hidden_dim2, out_dim),
    CosineSimilarity(),
    nn.Softmax(dim=1)
)
```

Model Specification (for NN model 1- contd.):

Character embedding:

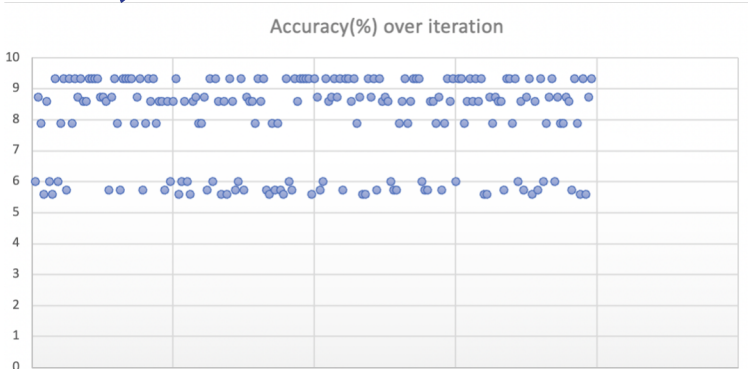
- Randomly initialized 16x35 and 64x35 representation
- 3 characters windows of character one-hot representation (previous-current-next) with ***rand*** as padding to form 16x35 embeddings
- 3 characters windows of character one-hot representation (previous-current-next) with zeros as padding to form 16x35 embeddings

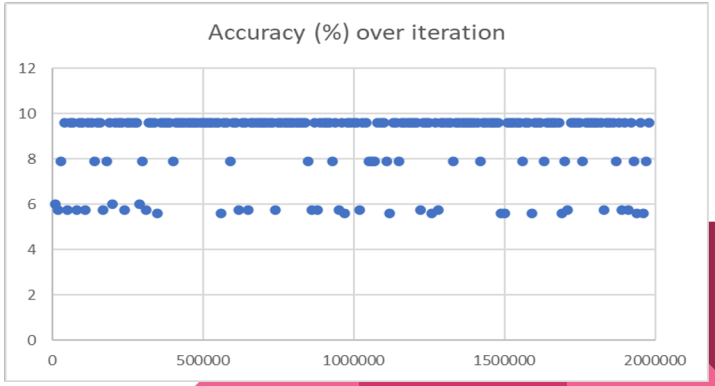


Results for NN model 1(Approach B) :

A few working attempts:

- FFNN, 16x35 flatten input dim, 3 characters windows of character one-hot representation (previous-current-next) with rand as padding, learning rate=0.01, momentum=0.8, weight decay=0.01, epoch=1, direct feed to loss function
  - **Test set accuracy : 9.475**
- FFNN, 16x35 flatten input dim, 3 characters windows of character one-hot representation (previous-current-next) with zeros as padding, learning rate=0.01, momentum=0.8, weight decay=0.01, epoch=1, direct feed to loss function
  - **Test set accuracy : 11.395**





Analysis of results for NN model 1 (Approach B):

Implementation challenges encountered:

- **Any custom significant loops introduced to compute embeddings or probabilities yield extreme slowness in training.**
  - Either need to be better in ***Python "vectorized operations"*** provided such as numpy, panda, etc for preprocess.
  - If no libraries are available for practical usage, probably need to ***implement in C/C++*** for performance.
- **"Stagnant / Bad accuracy / loss between training iterations"**
  - In a lot of cases of our attempt, we encounter a lot of cases in which the ***NN is just not learning.***
  - After lots of trial and error, reviews of NN and probability theory, it is either usually related to ***high level Math abstraction*** we are attempting to have NN perform for us to solve problem or the changes in accuracy is too small to be observed in large iterations, ***scaling accuracy computation to smaller iteration*** usually will shows the changes.

Model Specification (for Neural Networks-contd.):

3. NN model 3: Sequences using LSTM

Aim: We will develop a model of the text that we can then use to **generate new sequences of text.** The language model will be statistical and will predict the probability of each word given an input sequence of text. The predicted word will be fed in as input to in turn generate the next word.

A key design decision is how long the input sequences should be (need to be long enough to allow the model to learn the context for the words to predict). This input length will also define the length of seed text used to generate new sequences when we use the model.

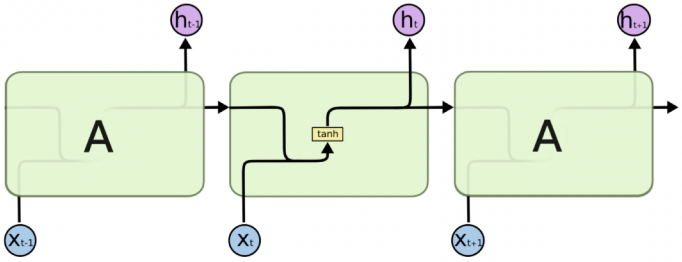
Model trained: Neural language model

- It uses a distributed representation for words so that different words with similar meanings will have a similar representation.
- It learns the representation at the same time as learning the model.
- It learns to predict the probability for the next word using the context of the last 100 words.

Model Specification (for NN model 3-contd.):

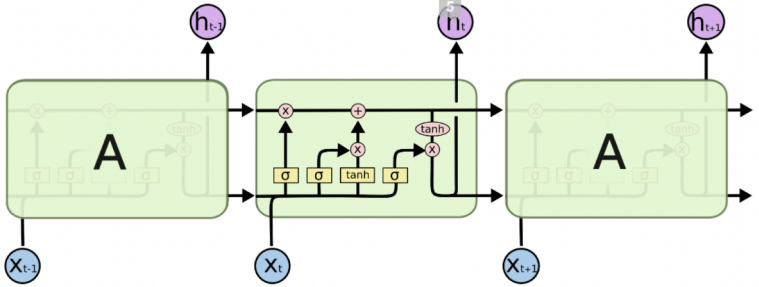
Embedding layer: To learn the representation of words (expects input sequences to be comprised of integers)

Long Short-Term Memory (LSTM) recurrent neural network: To learn to predict words based on their context. LSTMs are a special kind of RNN, capable of learning long-term dependencies.



The repeating module in a standard RNN contains a single layer.

In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



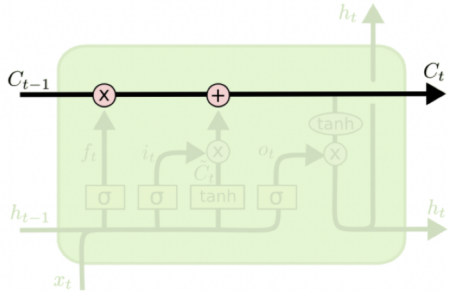
The repeating module in an LSTM contains four interacting layers.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four.

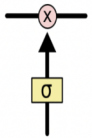
Image source for LSTM: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Model Specification (for NN model 3-contd.):

More about LSTMs:

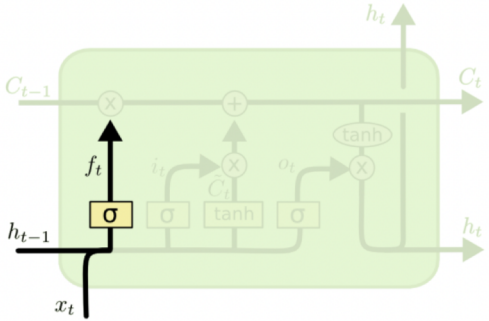


*The **cell state** lets information just flow along it unchanged.*



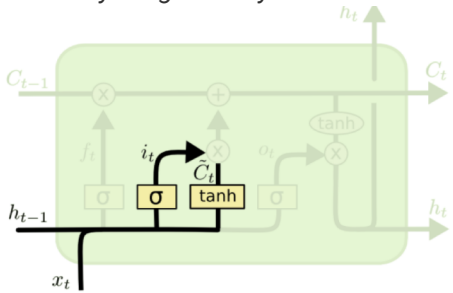
*The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called **gates.** An LSTM has three gates, to protect and control the cell state.*

Image source for LSTM: https://colah.github.io/posts/2015-08-Understanding-LSTMs/



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

***Stage 1:** What information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "**forget gate layer**."*

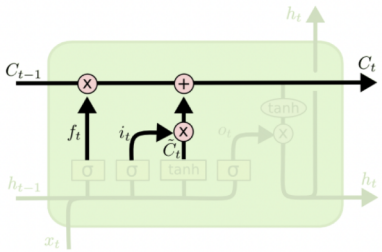


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

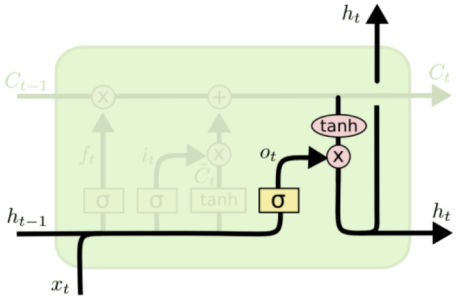
***Stage 2:** Decide what new information we're going to store in the cell state. First, a **sigmoid layer called the "input gate layer"** decides which values we'll update. Next, a **tanh layer creates a vector of new candidate values** that could be added to the state.*

Model Specification (for NN model 3-contd.):



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Stage 3: *Update the old cell state into the new cell state*



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Stage 4: *We need to decide what we're going to output. First, we run a* ***sigmoid layer*** *which decides what parts of the cell state we're going to output. Then, we put the output of sigmoid gate through* ***tanh*** *(to push the values to be between -1 and 1) to get desired filtered version of our cell state.*

So using LSTM we can map each word in our vocabulary to a unique integer and encode our input sequences. Later, when we make predictions, we can convert the prediction to numbers and look up their associated words in the same mapping.

Image source for LSTM: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Results and Analysis of NN model 3:

The output here show the NN model that has been built (right) and the tokens and sequences generated (bottom).

- With so many high trainable model parameters the NN takes hours to train.
- Look into advanced input character embedding.
- Increased GPU support required.
- Knowledge of extensive libraries.

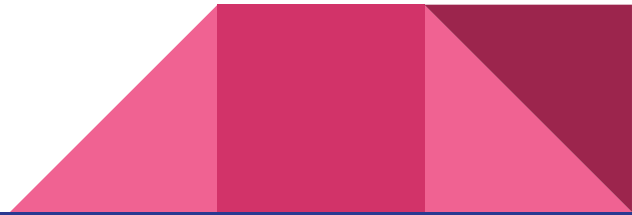
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 50, 50)            1365900
_________________________________________________________________
lstm (LSTM)                  (None, 100)               60400
=================================================================
Total params: 1,426,300
Trainable params: 1,426,300
Non-trainable params: 0
_________________________________________________________________
None
```

```
len of lines: 3030
number of words: 133787
examples of processed words: ['written', 'by', 'mak', 'unk', 'he', 'appeared', 'on', 'episode', 'of', 'the', 'television']

 = Valkyria Chronicles III =

 Senjō no Valkyria 3 : <unk> Chronicles ( Japanese : 戦場のヴァルキュリア3 , lit . Valkyria of the Battlefield 3 ) , commonly referred to as Valkyria
['valkyria', 'chronicles', 'iii', 'senjō', 'no', 'valkyria', 'chronicles', 'japanese', 'lit', 'valkyria', 'of', 'the', 'battlefield', 'commonly',
Total Tokens: 1621447
Unique Tokens: 27317
Total Sequences: 1621396
```

Summary:

- Text analysis is more complicated than we anticipated.
- For this simplified task, implementation of non-neural models on character base yielded good results but for further increase in accuracy deep learning methods need to be explored.
- Character level model turned out to be better than word level models.
- Industrial level application could require a much more intricate mix of models with more GPU and RAM space.

In 7 words: Best of luck for NLP next quarter. :P



Thank you!