



# AI Hackathon

Team: bella\_ciao



## Approach

1. Web Scraping, Tokenization and Numericalization (Preprocessing)
2. Language Model (AWD\_LSTM)
3. Text Classifier
4. Fine tuning of hyperparameters



## Preprocessing

- Library pypatent and BeautifulSoup used for web-scraping the training and test data
- The sentences are split into words based on spaces and punctuation using FastAi Library Tokenizer and Numericalizer.



## Language Model

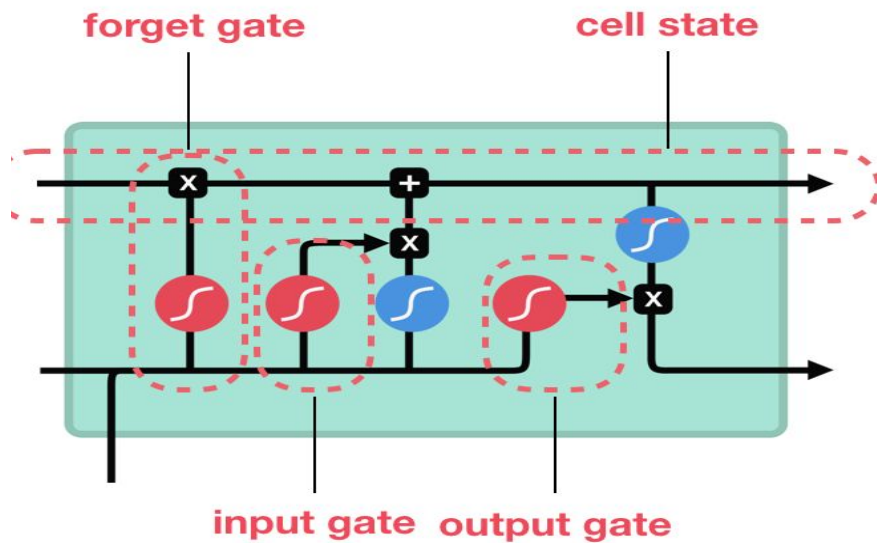
- Language Model is based on the recurrent neural neural network: AWD LSTM
- Pre-trained model based on Wiki-103 dataset used after fine-tuning.
- Aim of the language model: to predict the next word in the given natural language sentence.



## Why AWD LSTM?

The AWD-LSTM stands for ASGD Weight-Dropped LSTM. It uses DropConnect and a variant of Average-SGD (NT-ASGD) along with several other well-known regularization strategies

## LSTM



sigmoid



tanh



pointwise  
multiplication



pointwise  
addition



vector  
concatenation



## Forget Gate

First, we have the forget gate. This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.



## Input Gate

To update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.





## Cell State

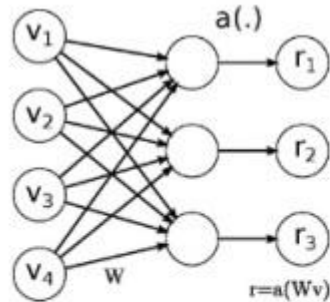
Now we should have enough information to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.



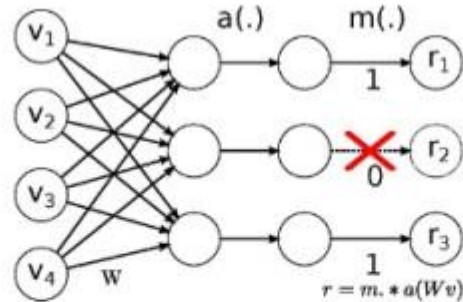
## DropConnect

We know that in Dropout, a randomly selected subset of activations is set to zero within each layer. In DropConnect, instead of activations, a randomly selected subset of weights within the network is set to zero. Each unit thus receives input from a random subset of units in the previous layer.

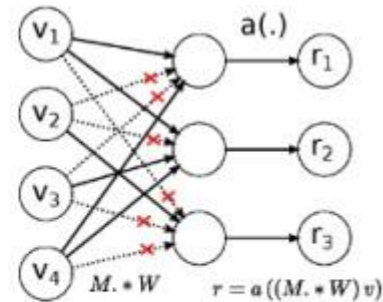
# Comparing NO-Drop Network, DropOut Network and DropConnect Network



No-Drop Network



DropOut Network



DropConnect Network



## ASGD

It has been found that for the particular task of language modeling, traditional SGD without momentum outperforms other algorithms such as momentum SGD, Adam, Adagrad, and RMSProp. Therefore investigated a variant of the traditional SGD algorithm known as ASGD (Average SGD).



### Traditional SGD update –

$W_t = W_{\text{prev}} - \text{lr}_t * \text{grad}(W_{\text{prev}})$

### ASGD update –

$\text{Avg\_fact} = 1/\max(t-K, 1)$

if( $\text{avg\_fact} \neq 1$ ):

$W_t = \text{avg\_fact} * (\text{sum}(w_{\text{prevs}}) + (w_{\text{prev}} - \text{lr}_t * \text{grad}(w_{\text{prev}})))$

Else:

$W_t = w_{\text{prev}} - \text{lr}_t * \text{grad}(w_{\text{prev}})$

where,

**K** is the minimum number of iterations run before weight averaging starts. So before **K** iterations, the ASGD will behave similarly to a traditional SGD. **t** is the current number of iterations done, **sum(w\_prevs)** is the sum of weights from iteration **K** to **t** and **lr<sub>t</sub>** is the learning rate at iteration **t** decided by a learning rate scheduler.



## NT-ASGD

We used variant of ASGD (NT-ASGD) in which-

- Averaging is triggered only when the validation metric fails to improve for multiple cycles. This is ensured by the non-monotone interval hyperparameter  $n$ . So whenever the validation metric does not improve for  $n$  cycles the algorithm switches to use ASGD. The authors found that setting  $n=5$  works well.
- A constant learning rate is used throughout the experiment and hence no further tuning is required.



## **Extended Regularization Techniques**

In addition to the two techniques discussed above, AWD-LSTM uses additional regularization techniques that prevent overfitting and improve data efficiency.

- Variable Length Backpropagation Sequences
- Variational Dropout
- Embedding Dropout
- Reduction in Embedding Size
- Temporal Activation Regularization

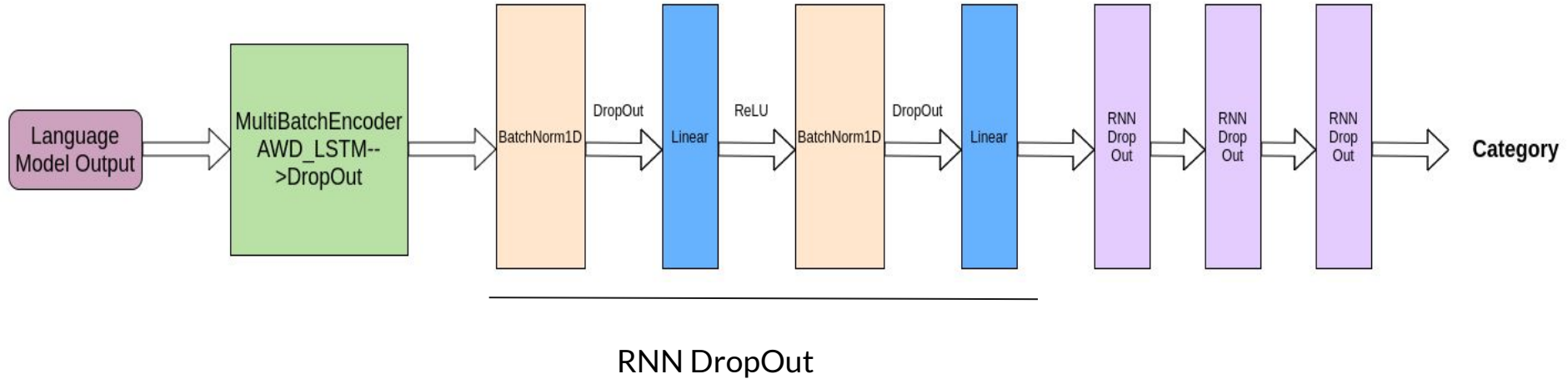


## Text Classifier

- The language model described is used for getting the encoded natural language sentences
- Encoded sentence used as an input to the text classifier
- The text classifier is again based on the AWD LSTM model




# The Classifier Model





## Fine Tuning and Reaching a Best Possible Minima

- The language model was fine tuned with a learning rate of  $1e-3$  yielding an accuracy of approximately 33 % (significance)
- The language model embeddings are then used as input to the classifier
- The classifier used Adam optimizer as a variant of gradient descent.

- 
- The pretrained weights of the classifier were randomized for the last 3 layers
  - These weights are learnt for our dataset initially with a learning rate of  $1e-1$  and momentum parameters of 0.7
  - Once the model reaches a local minimum, the learning rate is decreased to  $1e-3$  and later to  $1e-4$  and the momentum parameters are increased to 0.9

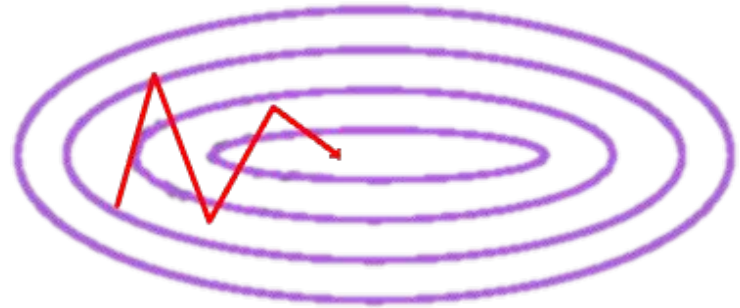
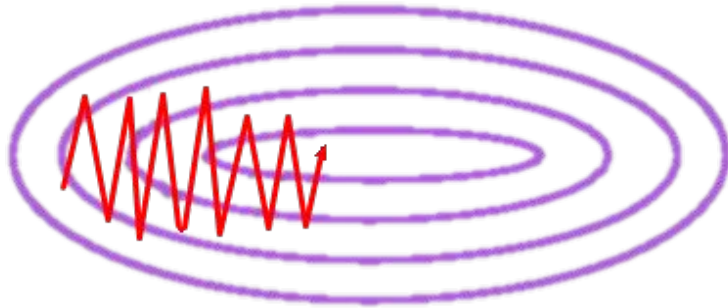


Diagram in order to explain the fine-tuning steps



## GUI for the Application

- Using AppJar Library and the Predictions being made by our model for the assigned test dataset.
- Voice Replies and Text Responses made available.



**Thank you**