Causal New Corpus

- Advanced NLP Project

Team - Thunder Bolts

Nithil Konduru - 2020101104

Greeshma Amaraneni - 2020101035

Motivation:

Event Identification is one of the important downstream tasks like the question answering and summarization. Finding causality between events is one of the important challenge as causality is more psychological than a linguistic concept

Goal:

Implement a Language Model that identifies if a sentence is causal or not, having accuracy more than the model mentioned in research paper by pre-training on datasets like Because 2.0, EventStoryLine

Data Sets

1. Causal New Corpus
2. Because 2.0
3. CTB
4. Event Story Line
5. PDTB

Data Set Characteristics

Because 2.0

Because 2.0 has approximately 5030 Causal sentences which is very huge and Because 2.0, a new version of the Because corpus with exhaustively annotated expressions of causal language.

EventStoryLine

Eventstory line is Annotated data for the identification of storylines. Data collected via crowdsourcing, in collaboration with the VU Amsterdam. Evenstory line,which has 1770 Causal sentences and 1500 Non causal sentences

CTB

CTB is the dataset consists of 1736 examples in which 318 Causal and 1418 are non causal and with its own semantic rules

Models Implemented

1. BERT Model
2. LSTM Baseline Model
3. BERT Baseline + LSTM Approach

Overview of Models

BERT Baseline Approach

Fine-Tuned pre-trained BERT model by adding 2 fully connected layers followed by softmax layer to the pooled-output obtained to do 2-way text classification

LSTM Approach

Experimented LSTM Baseline model by embedding words using FastText and added hidden layers followed by sigmoid activation.

BERT Baseline + LSTM Approach

The word embeddings generated by BERT are taken and given as input to LSTM baseline and followed the same LSTM approach.

BERT Baseline

- Model Architecture
- Architecture Diagram
- Code Snippets
- Hyper Parameters

Model Architecture

BERT Baseline

Input to this model is the tokenized words along with the special token CLS indicating the start of the sentence that BERT understands. BERT returns two outputs, one is output embedding of each of the tokens and other is that pooled output of all the tokens together of 768 embedding dimension

Fine Tuning BERT

Now a dropout layer is added to the pooled output inorder to avoid overfitting and this output is passed to fully connected linear layers and again a dropout layer is added and finally one more layer is added to this network.

Loss Backpropogation

The output is a two-dimensional vector which indicates probabiltiy for causal and non-casual after applying a softmax layer and loss is calculated with predicted value and is back propogated with cross entropy as loss criterion and Adam optimizer.

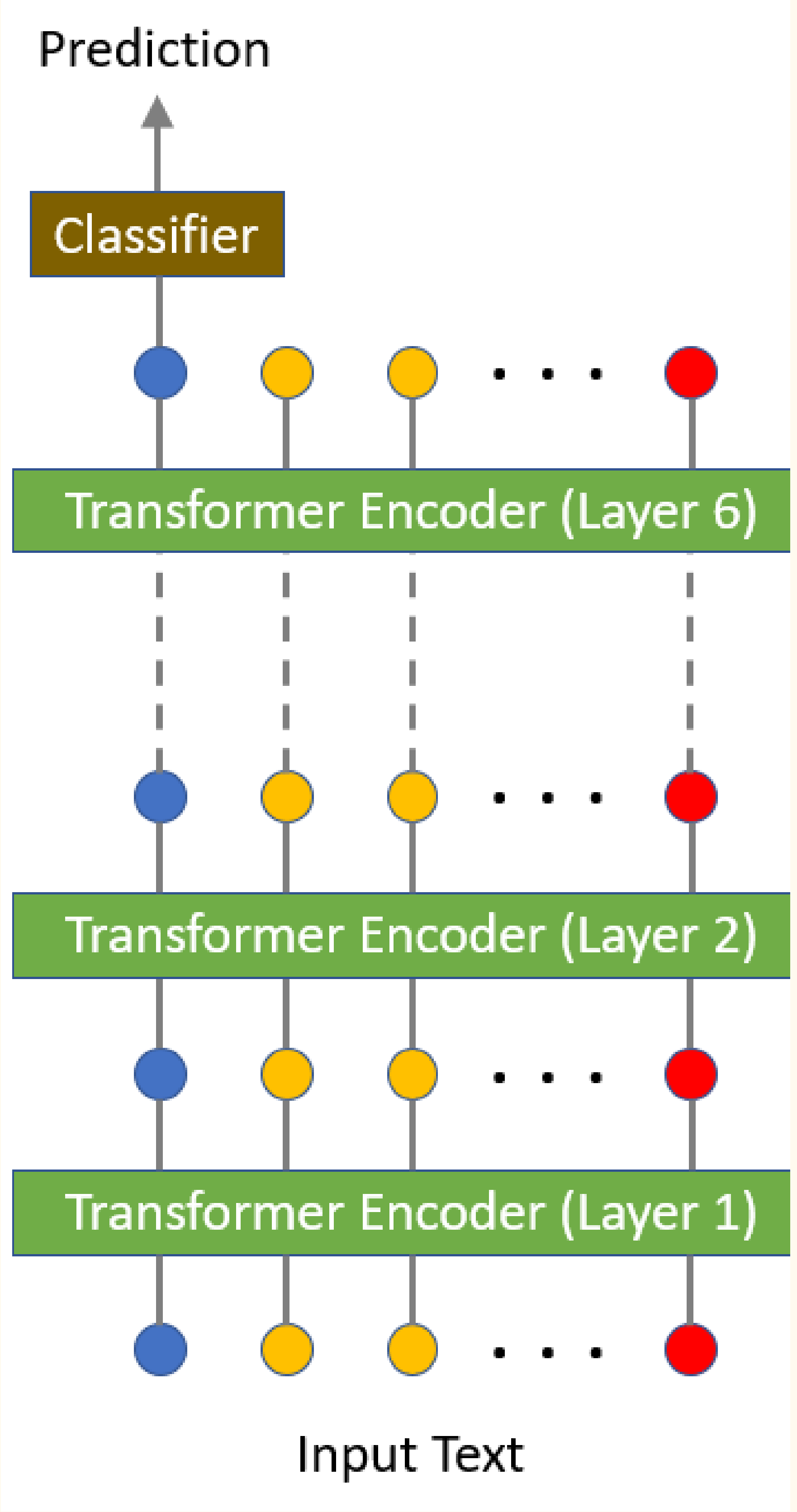
Related Code snippets

The adjacent images of code represent the BERT Baseline Approach that we have discussed

```python
class CausalClassifier(nn.Module):

    def __init__(self, hidden_dim, n_classes=2):
        super(CausalClassifier, self).__init__()
        # Layer to load the BERT Model
        self.bert = BertModel.from_pretrained(model_name)
        # Drop out layer added to avoid overfitting
        self.drop = nn.Dropout(p=0.3)

        # Linear layers added to pass bert pooled output to 2-way classifier network
        self.fc1 = nn.Linear(self.bert.config.hidden_size, 2*hidden_dim)
        self.fc2 = nn.Linear(2*hidden_dim, hidden_dim)
        self.out = nn.Linear(hidden_dim, n_classes)

        # Drop out layer in between thse hidden linear layers
        self.drop_fc = nn.Dropout(p=0.1)

    def forward(self, input_ids, attention_mask):
        #Passing in the input ids and attention mask to bert model
        _, pooled_output = self.bert(input_ids=input_ids,attention_mask=attention_mask)
        #sending pooled output to drop out layer
        output = self.drop(pooled_output)

        #output passed to linear layers and a dropout layer
        output_layer1 = self.fc1(output)
        output_layer2 = self.fc2(output_layer1)
        output = self.drop_fc(output_layer2)
        output = self.out(output)

        return output
```

```python
# Printing the stats in terminal
logger.info("***** Causal News Corpus(Team :Thunderbolts) *****")
print(classification_report(y_test, y_pred, target_names=class_names))

# Displaying the confusion matrix for the data we have used so far in evaluation phase
def show_confusion_matrix(confusion_matrix):
    hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
    hmap.yaxis.set_ticklabels(
        hmap.yaxis.get_ticklabels(), rotation=0, ha='right')
    hmap.xaxis.set_ticklabels(
        hmap.xaxis.get_ticklabels(), rotation=30, ha='right')
    plt.ylabel('True')
    plt.xlabel('Predicted')


# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)
```

Architecture Diagram



BERT Baseline

Hyper Parameters

01 Number of Epochs - 15

02 Loss Function - Cross Entropy Loss

03 Optimizer - Adam

04 Batch Size -16

05 Learning Rate - 0.01

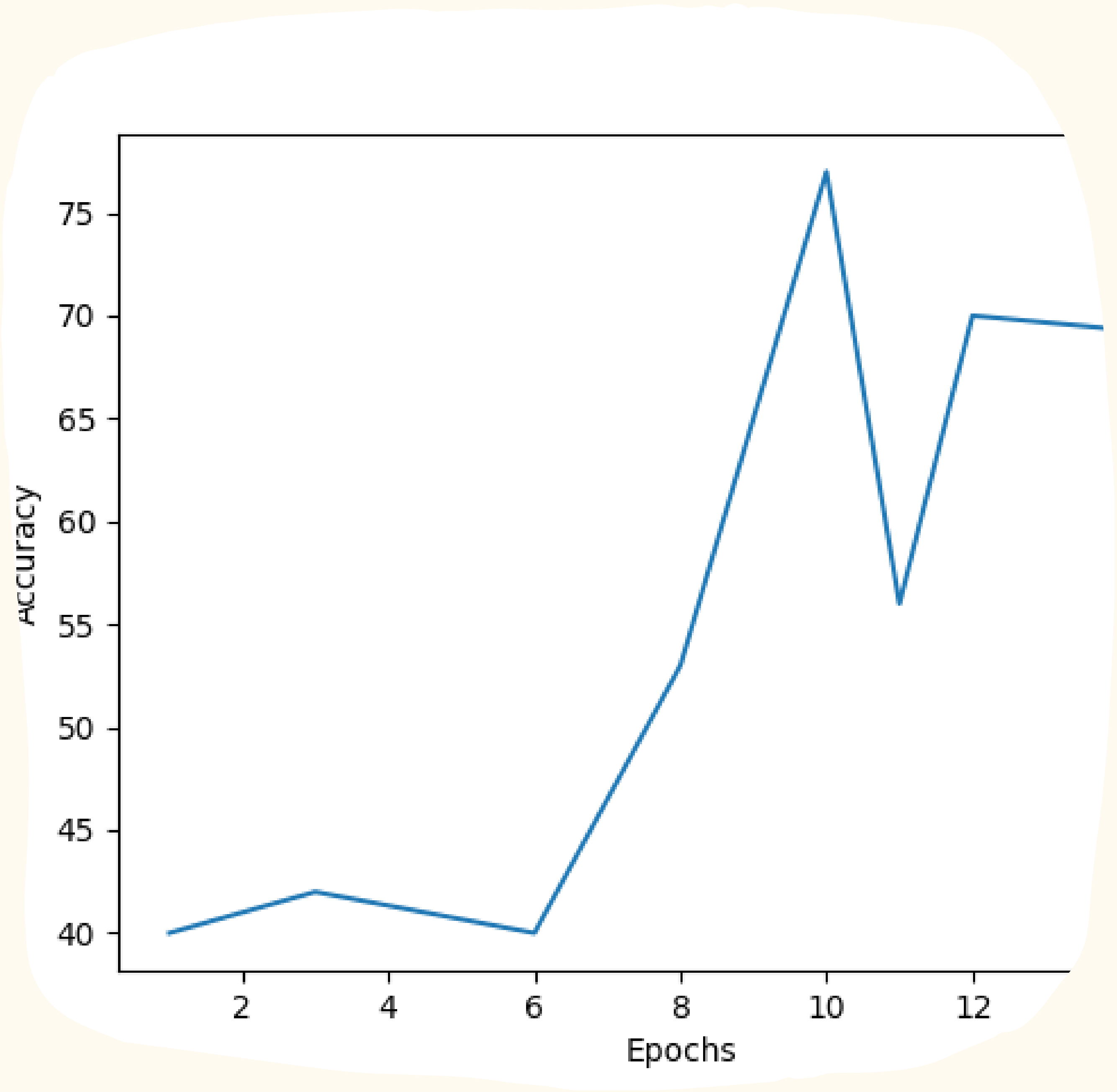
06 Hidden Layers - 2

Performance Over Metrics

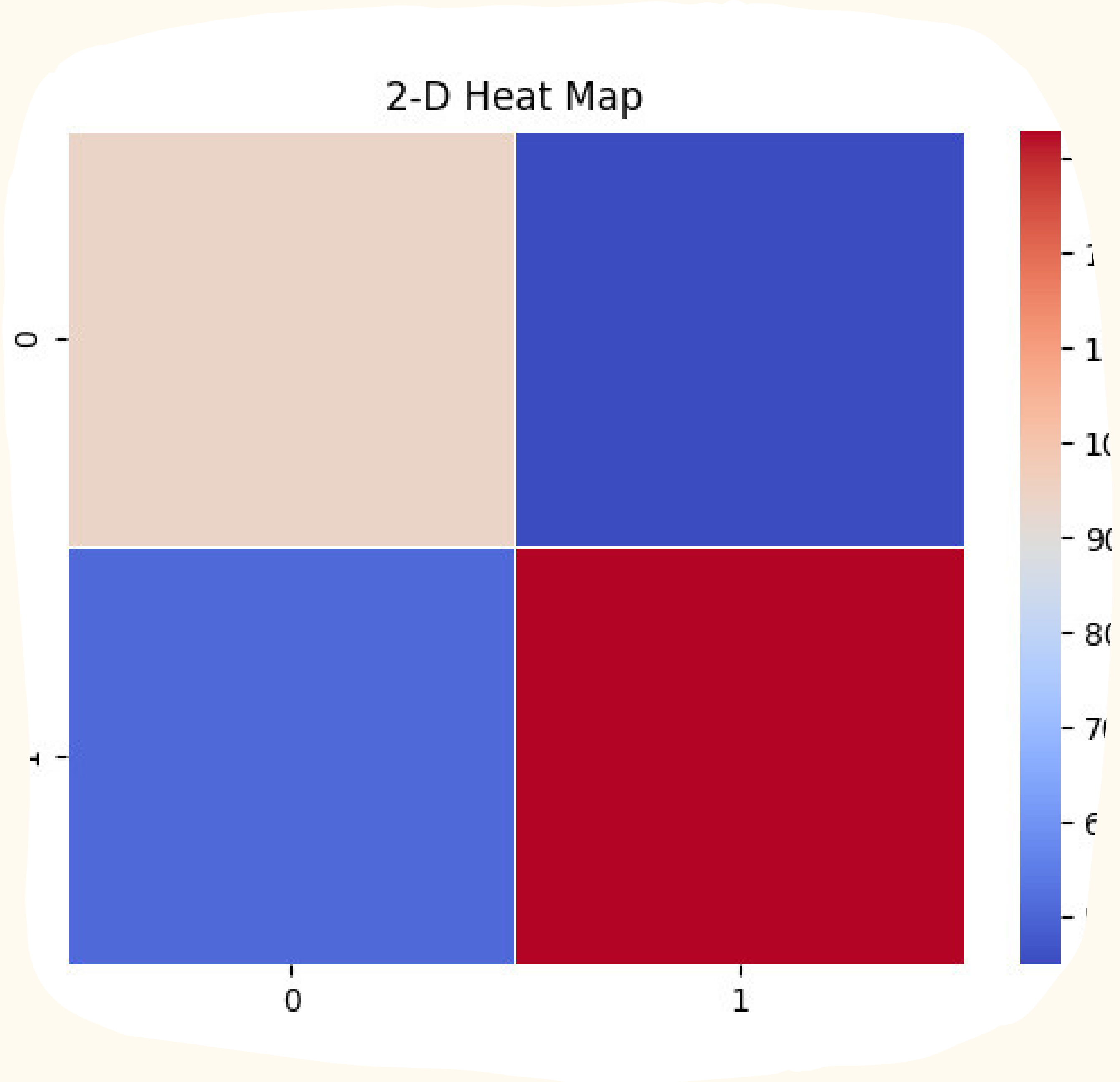
Accuracy - 77.12
F1 Score - 77.01
Precision - 81.78
Recall - 82.52
MCC - 42.43

Results

Accuracy vs Epochs



Confusion Matrix



LSTM Baseline

- Model Architecture
  - Architecture Diagram
    - Code Snippets
      - Hyper Parameters

Model Architecture

LSTM Baseline

LSTM model expects embeddings to be passed for the words. For this purpose, pre-trained FastText embeddings are used to get embeddings for each word and they are passed to the LSTM model.

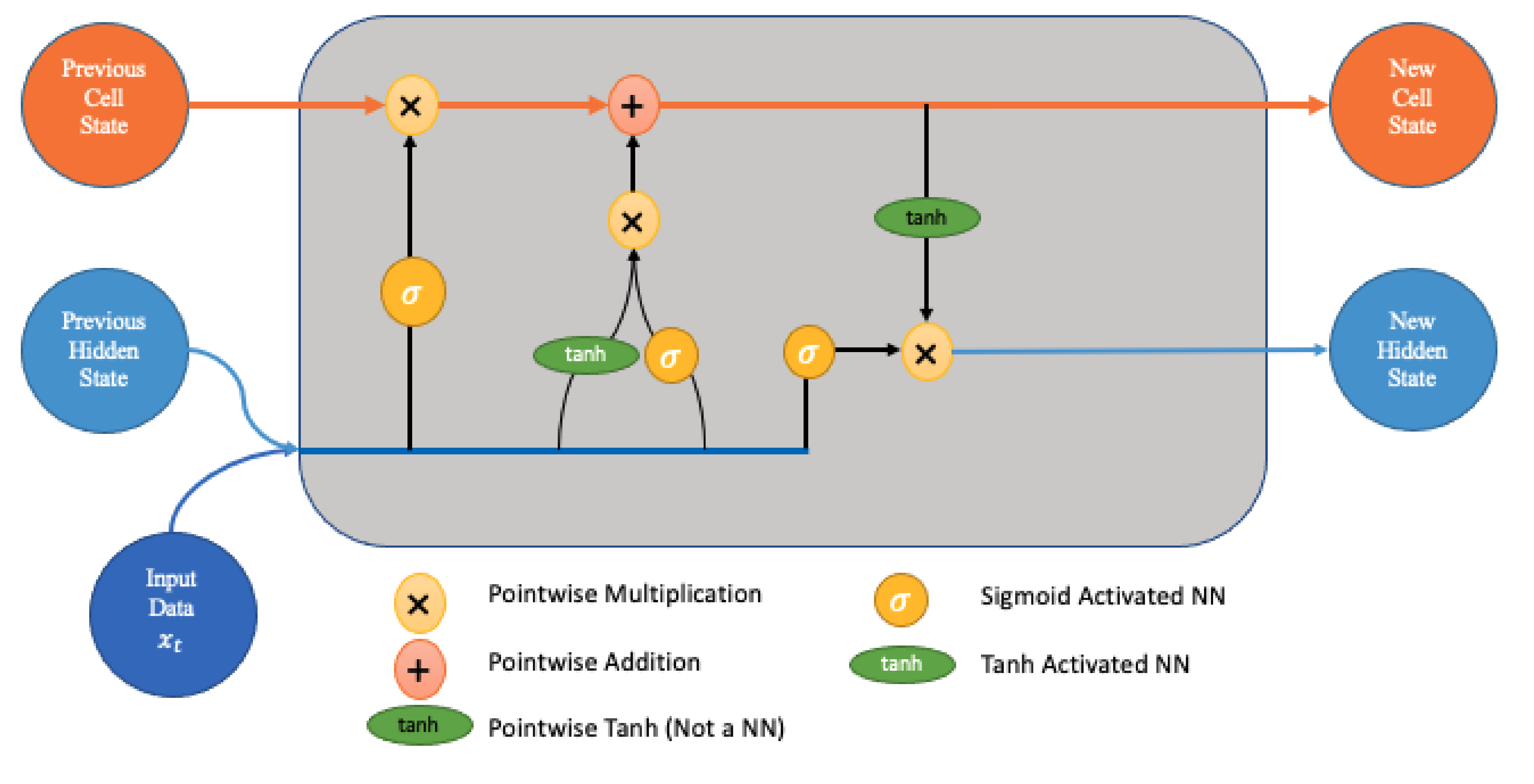
Fine Tuning LSTM

After passing through the stack of LSTM layers, the output is passed to fully connected layers where the last layer outputs a single value and a sigmoid activation is applied to generate probability of being causal.

Loss Backpropogation

Using the loss criterion as Binary cross entropy loss, loss is calculated in the predicted output and the loss is back propagated and then an optimizer is called to tweak the network and increase efficiency.

Architecture Diagram



LSTM Baseline

Related Code snippets

The adjacent images of code represent the LSTM baseline architecture that we have discussed

```
def evaluation(self):
    logger.info("***** eval metrics *****")

    predictions = []
    self.model.eval()
    with torch.no_grad():
        for x_batch, y_batch in self.loader_test:
            x = x_batch.type(torch.LongTensor)
            y = y_batch.type(torch.FloatTensor)

            ## preidicting the results

            y_pred = self.model(x)
            predictions += list(y_pred.detach().numpy())

    return predictions


def calculate_accuray(grand_truth, predictions):
    logger.info("***** Predict *****")
    true_positives = 0
    true_negatives = 0

    for true, pred in zip(grand_truth, predictions):

        ## calulating the accuracy by analying the predictions
        if (pred > 0.5) and (true == 1):
            true_positives += 1
        elif (pred < 0.5) and (true == 0):
            true_negatives += 1
        else:
            pass

    return (true_positives+true_negatives) / len(grand_truth)
```

ⓧ 0 ⚠ 1 Live Share

```
class causalmodel(nn.ModuleList):

    def __init__(self, args):
        super(causalmodel, self).__init__()

        # Initilasing the paarmeters used in the LSTM Network
        self.batch_size = args.batch_size
        self.hidden_dim = args.hidden_dim
        self.LSTM_layers = args.lstm_layers
        self.input_size = args.max_words # embedding dimention

        # Adding droput layer
        self.dropout = nn.Dropout(0.5)
        self.embedding = nn.Embedding(self.input_size, self.hidden_dim, padding_idx=0)
        # Main LSTM Layer added with argument pased parameters for the model
        self.lstm = nn.LSTM(input_size=self.hidden_dim, hidden_size=self.hidden_dim, num_layers=self.LSTM_layers,
        # Two fully connected layers added to output a 1d vector finally
        self.fc1 = nn.Linear(in_features=self.hidden_dim, out_features=256)
        self.fc2 = nn.Linear(256, 1)


    def forward(self, x):

        # Intilaising with zeros for hidden and cell states
        h = torch.zeros((self.LSTM_layers, x.size(0), self.hidden_dim))
        c = torch.zeros((self.LSTM_layers, x.size(0), self.hidden_dim))

        # FasText embedding of the input is considered
        # Tokenized word embeddings input is passed to LSTM
        out, (hidden, cell) = self.lstm(out, (h,c))
        # Drop out layer is added
        out = self.dropout(out)
        # Relu activaion layer is added
        out = torch.relu_(self.fc1(out[:,-1,:]))
        # Drop out layer along with sigmoid activation
        out = self.dropout(out)
        out = torch.sigmoid(self.fc2(out))

        return out
```

Hyper Parameters

01 Number of Epochs - 15

02 Loss Function - Binary Cross Entropy

03 Optimizer - RMSProp

04 Batch Size -16

05 Learning Rate - 0.04

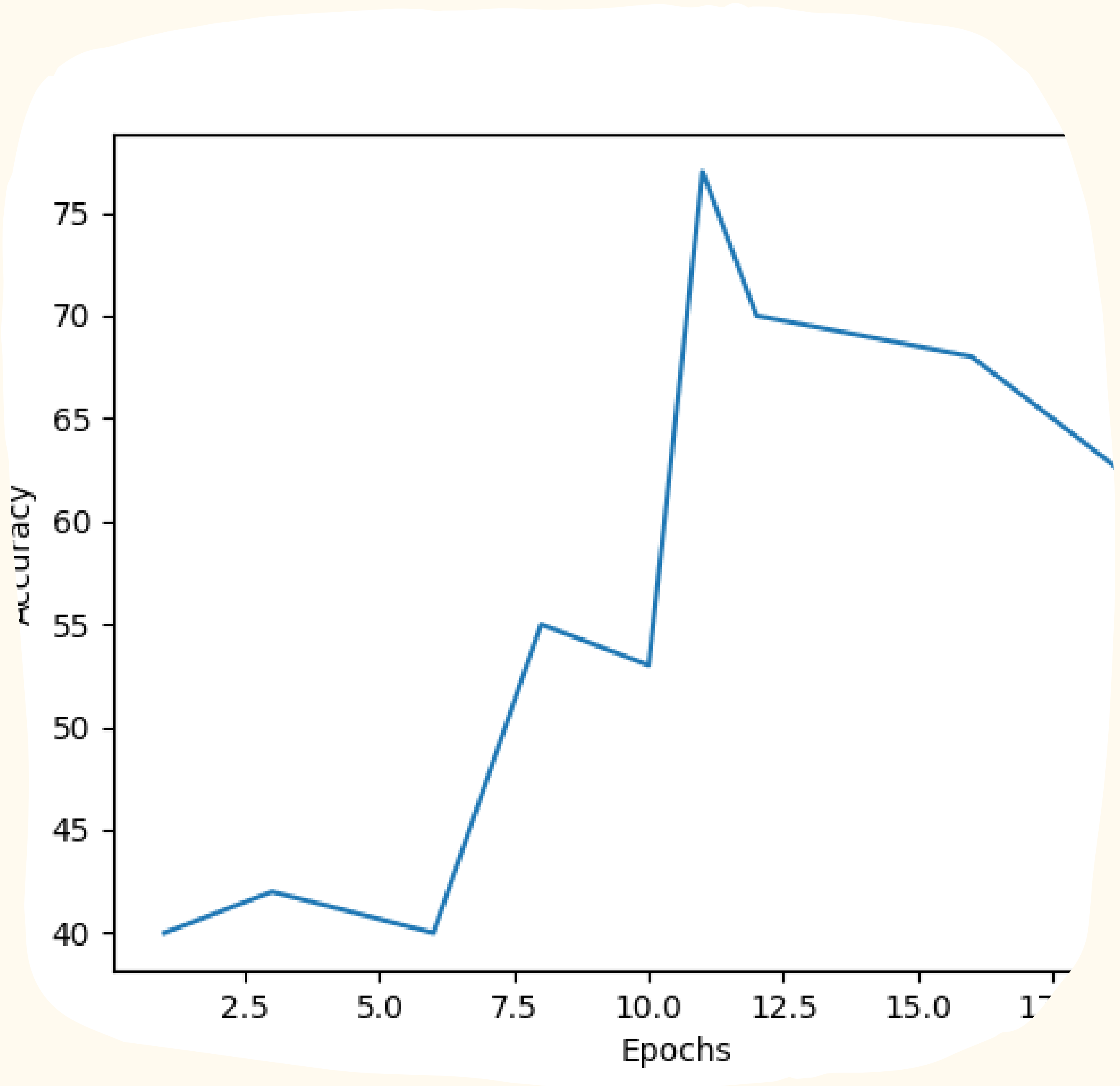
06 Stacked Bi-LSTM Layers - 5

Performance Over Metrics

Accuarcy - 74.19
F1 Score - 81.28
Precision - 71.97
Recall - 81.52
MCC - 49.43

Results

Accuracy vs Epochs



LSTM Baseline

BERT Baseline+LSTM Approach

- Model Architecture
  - Architecture Diagram
    - Code Snippets
      - Hyper Parameters

Model Architecture

BERT Baseline

Input to this model is the tokenized words along with the special token CLS indicating the start of the sentence that BERT understands. BERT returns two outputs, one is output embedding of each of the tokens and other is that pooled output of all the tokens together of 768 embedding dimension

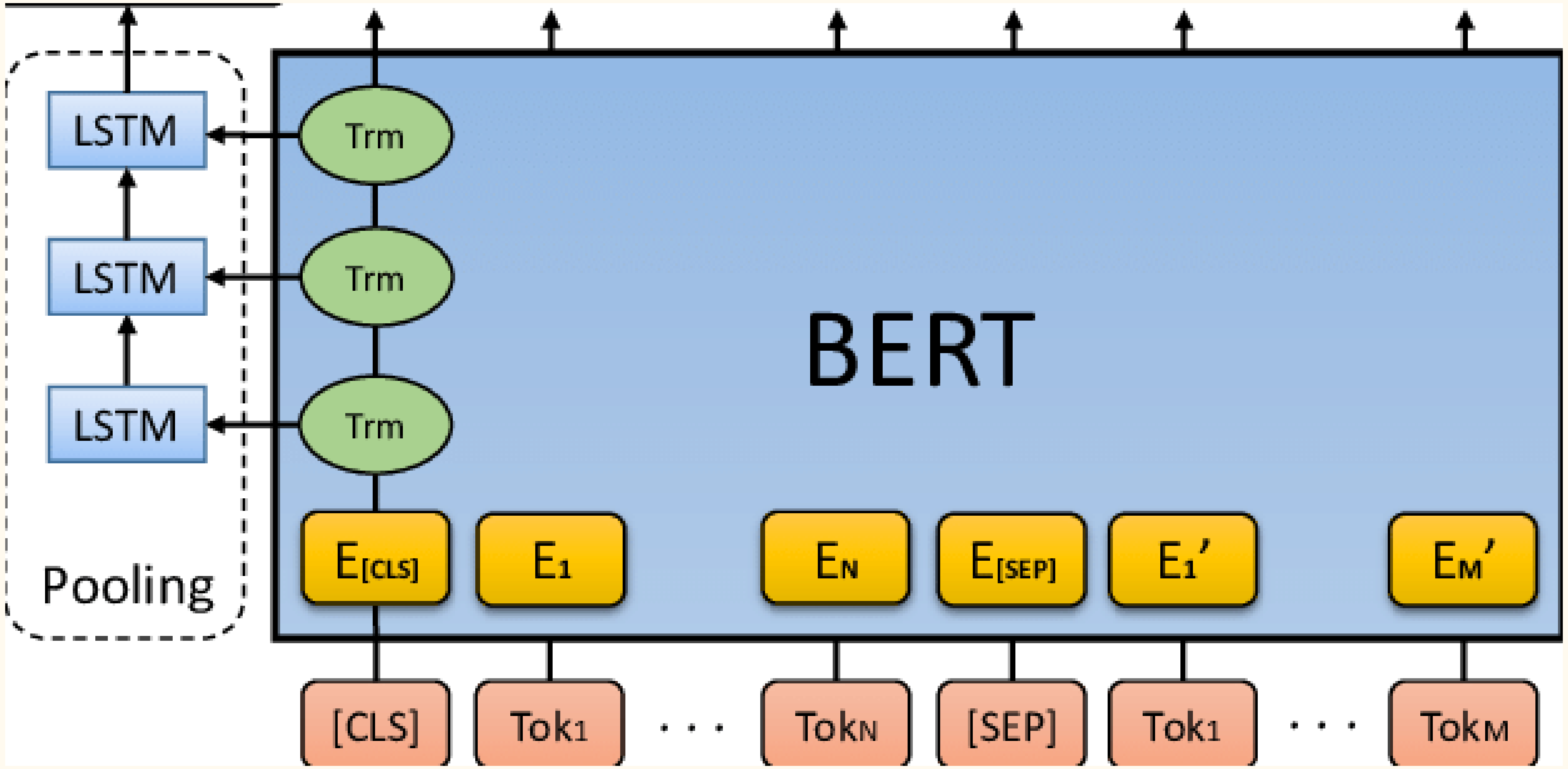
BERT Output to LSTM

This embedding is fed to LSTM network after passing through two fully connected layers and gives an embedding of argumented hidden size dimension for each word and a dropout layer and then two other linear layers are added which result in a one-dimensional vector for each word.

Loss Backpropogation

Sigmoid activation is applied which essentially borrows the same idea from LSTM baseline approach. Loss is calculated using Binary cross entropy loss in the predicted output and the loss is back propagated and then an optimizer is called to tweak the network and increase efficiency

Architecture Diagram



BERT Baseline+ LSTM Approach

Related Code snippets

The adjacent images of code represent the BERT baseline+ LSTM approach that we have discussed

```
# Printing the Evaluation metrics
def evaluation(self):
    logger.info("***** eval metrics *****")
    # As we don't backpropogate during this eval phase
    predictions = []
    self.model.eval()
    with torch.no_grad():
        for x_batch, y_batch in self.loader_test:
            x = x_batch.type(torch.LongTensor)
            y = y_batch.type(torch.FloatTensor)
            ## here we are predicting the results
            y_pred = self.model(x)
            predictions += list(y_pred.detach().numpy())

    return predictions

# Calculating the accuracy
def calculate_accuray(grand_truth, predictions):
    logger.info("***** Predict *****")
    true_positives = 0
    true_negatives = 0

    ## checking the porbabilites to calculate the the truepositives a

    for true, pred in zip(grand_truth, predictions):
        if (pred > 0.5) and (true == 1):
            true_positives += 1
        elif (pred < 0.5) and (true == 0):
            true_negatives += 1
        else:
            pass

    return (true_positives+true_negatives) / len(grand_truth)
```

```
class CausalClassifier(nn.Module):

    def __init__(self, hidden_dim, n_classes=2):
        super(CausalClassifier, self).__init__()
        # Intialising the pre-trained BERT Model
        self.bert = BertModel.from_pretrained(model_name)
        # Drop out layers added to avoid overfitting
        self.drop = nn.Dropout(p=0.3)
        self.dropout = nn.Dropout(0.5)
        self.embedding = nn.Embedding(
            self.input_size, self.hidden_dim, padding_idx=0)
        # LSTM layer to take in the embeddings generated by BERT
        self.lstm = nn.LSTM(input_size=self.hidden_dim, hidden_size=self.hidden_dim,
                            num_layers=self.LSTM_layers, batch_first=True)
        # Two linear layers added to output a 1d vector represneting probability belonging to 1 class
        self.fc1 = nn.Linear(in_features=self.hidden_dim, out_features=256)
        self.fc2 = nn.Linear(256, 1)

    def forward(self, input_ids, attention_mask):

        # Word embeddings of tokens are generated by bert along with pooled output
        embeddings, pooled_output = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask
        )

        # Initialising the hidden and cell states with zeros
        h = torch.zeros(
            (self.LSTM_layers, embeddings.size(0), self.hidden_dim))
        c = torch.zeros(
            (self.LSTM_layers, embeddings.size(0), self.hidden_dim))

        # Adding dropout layers
        out = self.drop(embeddings)

        # output of BERT Genreated embeddings passed to LSTM layer
        out, (hidden, cell) = self.lstm(out, (h, c))

        # Dropoped out some neurons from network randomly and applied Relu non-linear layer
        # Followed by sigmoid activation
        out = self.dropout(out)
        out = torch.relu_(self.fc1(out[:, -1, :]))
        out = self.dropout(out)
        out = torch.sigmoid(self.fc2(out))

        return out
```

Hyper Parameters

01 Number of Epochs - 20

02 Loss Function - Binary Cross Entropy

03 Optimizer - RMSProp

04 Batch Size -16

05 Learning Rate - 0.04

06 Stacked Bi-LSTM Layers - 5

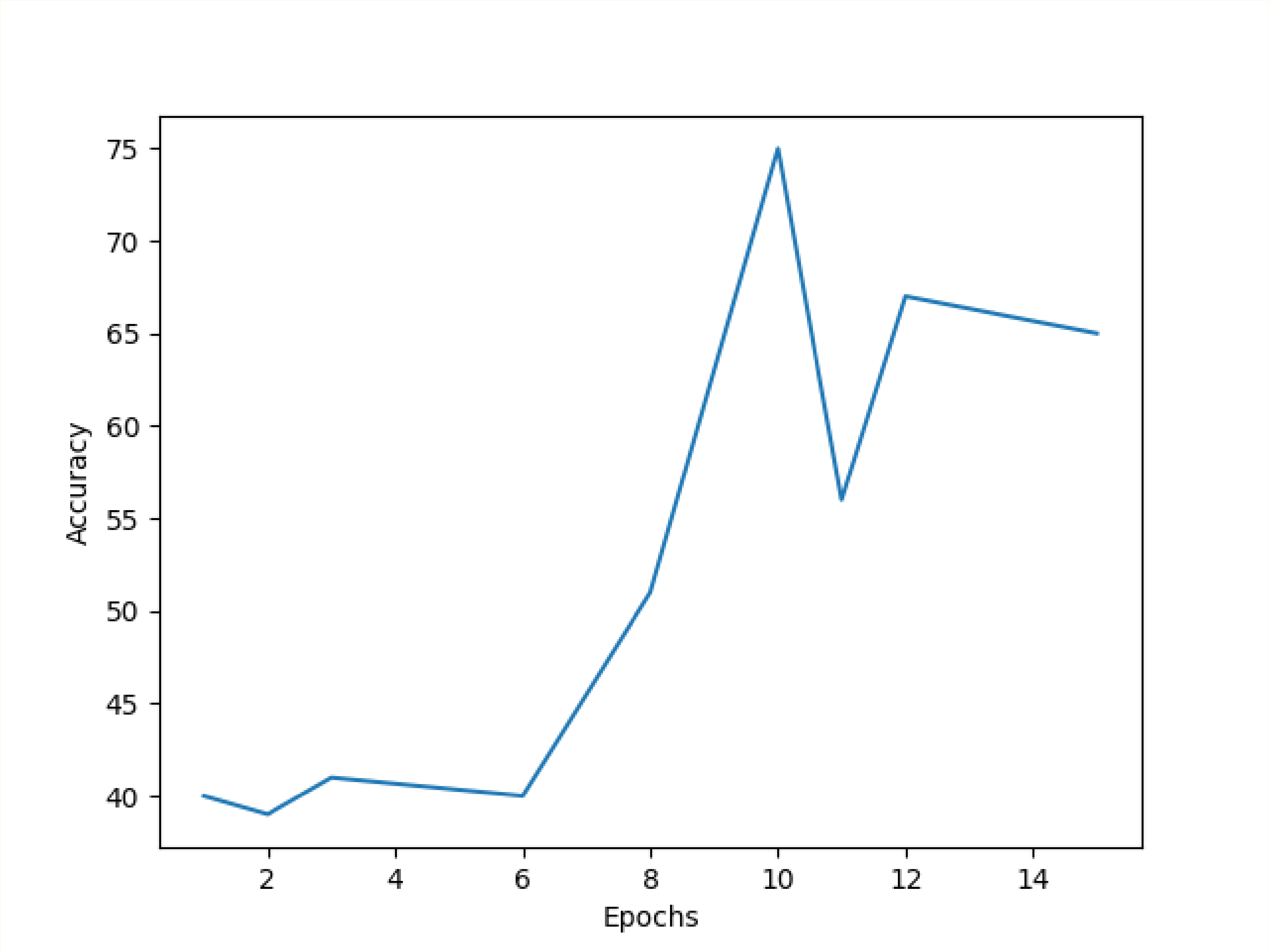
07 BERT embedding dimension - 768

Performance Over Metrics

Accuarcy - 78.72
F1 Score - 82.78
Precision - 78.57
Recall - 85.62
MCC - 54.93

Results

Accuracy vs Epochs



Overall Analysis

In CTB there are 1736 examples in which 318 Causal and 1418 are non causal .Here in this data set non causal sentences are single sentences but in the CNC dataset it is not like that it has causal and noncausal of irrespective lengths so for the model trained on CTB it has less probability to predict the non causal sentences with more than single sentences because of that we can observe the worsen scores when trained on the CTB and it is generated by Computer Aided translation.

Eventstory lines is Annotated data for the identification of storylines. Data collected via crowdsourcing, in collaboration with the VU Amsterdam.It has similar characteristics with the CNC but there are some restrictions the sentence length is not much in the Event storyline and we found some sentences it is annotating as Causal even if it non causal when trained on the EventStoryLine it has annotation rule to mark some words as causal which CNC doesn’t make causal. We have seen a sentence

“The criticism comes as the city prepared on Sunday for its third consecutive day of mass civil dissent , following Saturday ’ s rally in Yuen Long and an 11-hour-sit-in at the Hong Kong airport on Friday.”This sentence has identifies as causal by the Even storyline whereas CNC doesn’t.it might be because of the High non causal sentences of the Event storyline and its annotation rules of identifying phrases as Causal.We have worked out this with some examples.

BECauSE 2.0, a new version of the BECauSE corpus with exhaustively annotated expressions of causal language, but also seven semantic relations that are frequently co-present with causation. The new corpus shows high inter-annotator agreement, and yields insights both about the linguistic expressions of causation and about the process of annotating co-present semantic relations

Model has increased its
Accuracy because of the similar semantic relationship and annotation relations has increased its accuracy and also because of its huge size. We implemented three approaches and we could increase in 1.2% accuracy compared to the model implemented in the paper. We have achieved this through improving the baseline model by fine tuning the pre-trained BERT model with LSTM layer to do a downstream task of text classification and pre-training the model on dataset Because 2.0

Future Work

For a sentence to be a causal in the first place, there should be a cause followed by an effect and a signal indicating the relation between both of them and in some cases signal is implicit in nature and in majority of the cases there is an explicit signal. So, after understanding the research paper and implementing the same, we could improve this work further by introducing another downstream task of identifying the cause, effect and signal for a casual sentence.

Thank you!

Do you have any questions for us?