Prediction

Damien Lopez
Alexander Mccaffrey

CITY UNIVERSITY
LONDON

1 Abstract

Conversational agents are limited by their lack of understanding of emotional subtext. The underlying sentiment of an utterance is an important dimension of any human conversation and being able to modify responses given these sentiments makes conversations more engaging. The following report describes a model which can classify the sentiment of an input sequence and modify its response accordingly. This is achieved by training two different transformer based models, one to classify sequences as being one of five emotional classes and one to generate text using this classification as context.

2 Introduction

The application of deep learning techniques to natural language processing (NLP) systems has led to the development of highly adaptive conversational agents (Young et al. 2017). Whereas conventional chatbots use a set of hand-crafted rules to answer a limited set of questions, transformer and recurrence-based models can learn features of language without these features being explicitly represented. The user is not constrained to input contextually correct responses and these models can generate varied outputs in a wide range of tasks.

The underlying sentiment of an utterance is an important aspect of human conversations. In order to make conversations with artificial agents more engaging, their output should adapt to the sentiment of the human utterances they respond to. We present a model in which the sentiment of an utterance is represented explicitly, a generative model then uses this representation to output a response of the same sentiment.

This is achieved by training a transformer-based model, GPT-2, to generate responses of a particular sentiment. The model is trained with different 'personas', each of which correspond to one of the sentiment classes the model learns to generate. The persona acts as a type of context when generating a reply. A sentiment classification model is trained to classify the sentiment of an input sequence as belonging to one of the following classes: "happy", "sad", "surprised", "angry" or "no emotion".

The output of the classification model is used as the 'persona' for GPT-2, which partly determines the reply generated. The persona is combined with the original utterance before being input to the generative model, which outputs a reply of the same sentiment. Both the generative and classification models are trained using the following datasets:

- DailyDialog: A high quality dataset of 13118 multi-turn dialogues. Each utterance has a corresponding emotional label indicating it belongs to one of seven sentiment classes
- EmotionLines Dataset: 1000 dialogues taken from the TV series Friends, each utterance is labelled as being of a particular emotion.

3 The Dataset

Both the DailyDialog and EmotionLines datasets contain annotated dialogues in which each utterance is labelled as belonging to one of seven emotional classes.

In the Daily Dialog dataset there are 13118 individual dialogues, each with two speakers. For each dialogue in the dataset, the utterance of each speaker is labelled with a particular emotion.

In the EmotionLines dataset there are approximately 29000 labelled utterances from 2000 dialogues. The information is stored in a JSON file and Ekman emotions (Ekman et al. 1987) are used to label each utterance as follows: 0: no emotion, 1: anger, 2: disgust, 3: fear, 4: happiness, 5: sadness, 6: surprise.

Dialogue Utterance one	Dialogue Utterance two
The kitchen stinks .	I'll throw out the garbage .
Emotion Utterance one	**Emotion Utterance two**
2	0

There were 96117 dialogue pairs produced with an equal amount of emotion pairs. To teach the chatbot to respond with concise sentences, any dialogue pair with more than 51 tokens was removed from the dataset. However there was a class imbalance as shown in table 1

	No emotion	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Original Class makeup	78.44	1.82	0.70	0.51	13.36	1.66	3.49
Used Class makeup	19.01	16.69	6.40	4.66	19.01	15.19	19.01

Table 1: Class Imbalance for Chatbot

To prevent any problems in training as a result of this class imbalance, up to 2000 samples of each class were included in the final dataset, resulting in a dataset of 10519 dialogue pairs, albeit with fear and disgust grossly underrepresented. As the dataset is approximately a 10th of the the original size, there is a significant risk of overfitting.

To train the sentiment analysis tool, a combination of the DailyDialog and EmotionLines dataset was used. The data was used differently with the inputs to training being one utterance and its emotion label. The utterances with the emotion labels of fear and disgust were removed as there were too little of them to be used. This resulted in a dataset comprising of 9830 samples out of an initial 117310 samples.

	No emotion	Anger	Disgust	Fear	Happiness	Sadness	Surprise
Original Class makeup	79.62	1.81	0.67	0.49	12.66	1.55	3.17
Used Class makeup	20.34	20.34	0	0	20.34	18.61	20.34

Table 2: Class Imbalance for Sentiment Analysis

4 GPT-2 Architecture

GPT-2 is a generative Transformer architecture that can be adapted to conversational tasks. Unlike the original Transformer architecture proposed by Vaswani et al. 2017, GPT-2 does not use any encoder blocks. The small version of GPT-2 used contains 12 identical decoder blocks. Given an input sequence, GPT-2 generates prediction scores for each word in its vocabulary. These prediction scores can be used to generate text at each time step using a decoding strategy.

4.1 The Decoder Block

As illustrated in figure 1 each decoder block is composed of a masked multi self-attention layer and a feed forward network. Each of these are followed by a layer normalization operation. Residual connections around each of the sub-layers are included to improve gradient flow throughout training (Zaeemzadeh, Rahnavard, and Shah 2018).

4.1.1 Masked Multi Self Attention

In order to better understand the meaning of each word in an input sentence, the model uses the immediate context in which each word was said. Specifically, for a given word, the model uses a weighted sum of the previous words to help determine the new representation of the given words. The masked multi self-attention layer solves this problem by assigning each of the previous words in a sequence a score when a new word is input.
These scores are determined by first multiplying the input vector of a given word by predefined Key, Query and Value matrices. This generates Key, Query and Value vectors for each word, the dimensions of which are equal to the original vector, (1, 768). These vectors are then split into twelve (1, 64) vectors so that self-attention can be computed multiple times on different parts of the Key, Query and Value vectors by different attention heads. Attention scores are computed by taking the dot product of the Query and Key vectors: the attention given to the word at position j when determining the representation of the word at position i depends on the dot product of Q_i and K_j. The scores for each of the words up to position i are computed and a softmax function is applied. The softmax scores are used to multiply the corresponding Value vectors V_j, which are summed for all j. The output vector, a weighted average of the Value vectors for each word depending on their relevance is the output of the attention head. The output vectors for each attention head are concatenated and linearly projected before being passed to the feed forward network.

4.1.2 Layer Normalization

Layer Normalization computes the "mean and variance used for normalization from all of the summed inputs to the neurons in a layer on a single training case" (Ba, Kiros, and Hinton 2016). This stabilises the inputs to a neuron in each forward pass, ensuring changes in the output of one layer do not cause highly correlated changes in the weights of the next layer. This addresses the problem of internal covariate shift (Ioffe and Szegedy 2015), which slows down learning as the network has to adjust to the changing distribution of network activations.

4.1.3 The Feed Forward Network

The feed forward network in each decoder block is contains two 1D convolutional layers followed by a GELU activation (Hendrycks and Gimpel 2016).

4.2 Adapting GPT-2 for dialogue

We use a pre-trained GPT-2 model from OpenAI and adapt it to generate dialogue. The input to the model contains three types of context:

- Persona: This is the model personality. In our case each personality permutation corresponds to one of the emotions in the dataset labels.
- History: The utterances that have been said so far in the conversation.
- Reply: The output of the model that has been generated so far.

A naive approach to applying pretrained GPT-2 to conversational tasks would be to simply concatenate these three types of contexts and input this sequence to the model. This approach raises two problems:

1. GPT-2 would not know which parts of the input sequence can be attributed to which speaker. This information is needed to ensure the output of the model is consistent.
2. GPT-2 would not know the position of each word in the input sequence.

In order to solve the first problem special tokens are added to the model vocabulary: "bos", "eos", "speaker1", "speaker2", "pad". The tokens are used to build a segment embedding, indicating the type of context a part of the input sequence belongs to. Firstly, a string of "speaker1" and "speaker2" tokens are generated to map each token in the input sequence to a particular speaker. Next, a positional embedding is generated, this is simply a list of numbers from 1 to the length of the input sequence indicating the position of each word in the sequence.

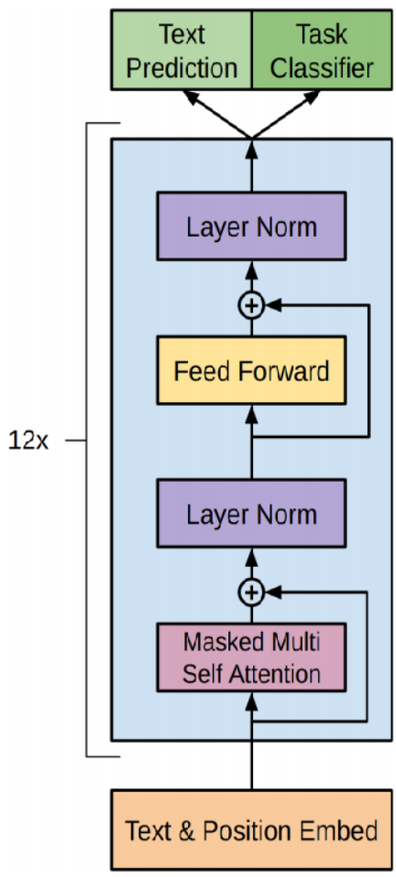


Figure 1: Illustrated Decoder Block from Radford et al. 2018

4.3 The Double Head Model

The GPT-2 model is trained to minimise loss on two tasks: a language modelling task and Next Sentence Prediction (NSP).
The output of the model is a probability distribution over the model vocabulary. The Language modelling loss is calculated as the cross-entropy loss between this distribution and the tokens of the words in the correct reply. However, many of the tasks the model needs to solve to converse in an engaging way (e.g. inference, question answering) require the model to understand the relationship between two sentences. This understanding is not directly captured through the language modelling task. To account for this, the model uses a next sentence prediction head.
Along with the correct response, responses of a different emotion are passed to the model during training. The hidden states output from the model is then passed to both the language modelling and NSP heads. The model loss is a weighted sum of the loss from both heads. Including the NSP loss trains the model to consider the global context, including the input 'persona'. The input to the language modelling head is therefore more likely to take account of the global context, meaning

'persona' will influence the next word predictions.

5 Training GPT-2

The data was split into training, validation and test datasets of size 9000, 1000 and 519 samples respectively . Thereafter the datasets were loaded with a dataloader, with training data being sampled randomly and validation and test data being sampled sequentially each with a batch size of 2.

The final model was trained for 3 epochs, training beyond this point resulted in overfitting given the relatively small dataset and the large number of parameters. The model was trained using the following hyper parameters:

Hyperparameter	Value
Batch size	2
Learning rate	$6.25e^{-5}$
LM coefficient	1
MC coefficient	1

A batch size of 2 for both the training and the validation set was used as the CUDA memory capacity was insufficient to run higher batch values. Performance on the validation set was evaluated regularly using the following metrics; language modelling loss, multiple-choice classification loss, overall loss, and perplexity were measured for both training and validation sets.

Perplexity was chosen as a particularly important metric to calculate as it is a measure of how well a probability model predicts a sample. The lower the perplexity value, the better the model is at predicting the next sample. Perplexity measures how surprising the next sample is given the current probability model.

In this case we train the model to predict the language modelling labels of the reply of the appropriate emotion. If the labels are surprising given the probability distribution of the model i.e. a high language modelling loss, then perplexity will be high.

In the GPT-2 double heads model, language modelling loss is calculated by taking the cross-entropy loss between the language modelling labels and the prediction scores for each token of the correct reply.

Perplexity is defined as:

$$\text{Perplexity} = e^{-\frac{1}{N}\sum_{i=1}^{N}\log_e q(x_i)} \tag{1}$$

Where N is the number of tokens in the vocabulary, x_i is the i^{th} token in the model vocabulary. The exponent is equivalent to the language modelling loss.

The data is loaded as follows:

Dataset

Sentence	Reply
The kitchen stinks .	I'll throw out the garbage .

Emotion Sentence	Emotion Reply
2	0

As loaded

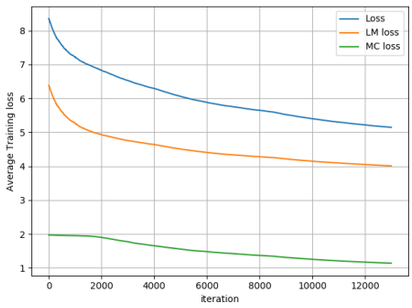
Emotion	Sentence	Reply
I have no Emotion.	The kitchen stinks.	I'll throw out the garbage .
I have no Emotion.	The kitchen stinks.	Angry Reply
I have no Emotion.	The kitchen stinks.	Disgusted Reply
I have no Emotion.	The kitchen stinks.	Afraid Reply
I have no Emotion.	The kitchen stinks.	Happy Reply
I have no Emotion.	The kitchen stinks.	Sad Reply
I have no Emotion.	The kitchen stinks.	Surprised Reply

Token Type IDs

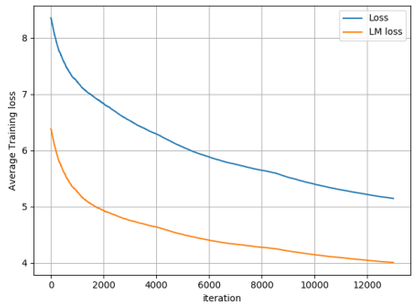
Emotion	Sentence	Reply
Speaker1	Speaker2	Speaker1

5.1 Results of Training

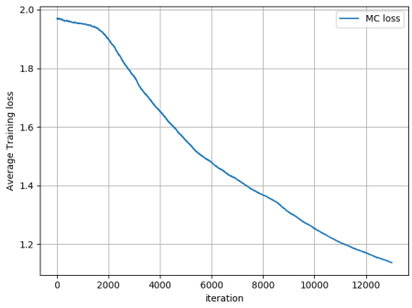
The model was trained for 4 epochs. Figure 2 illustrates the average Total, language modelling and classification losses throughout training



(a) Averaged Training Losses

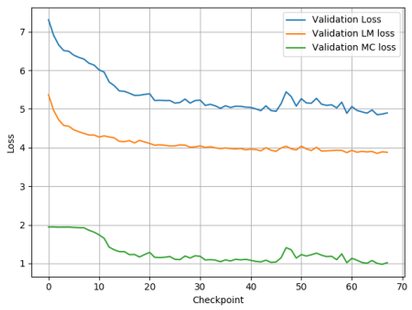


(b) Average Total and Language Modelling loss

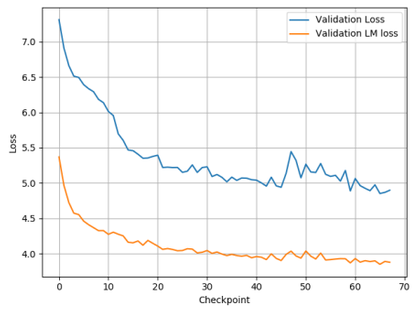


(c) Average MC Losses

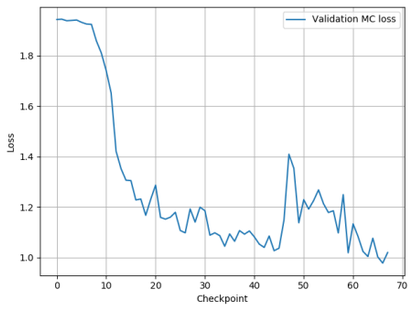
Every 200 iterations, the model was tested on the validation dataset. The total, language modelling and classification losses were recorded as well as the *Perplexity* of the text in the validation set according to the language model.



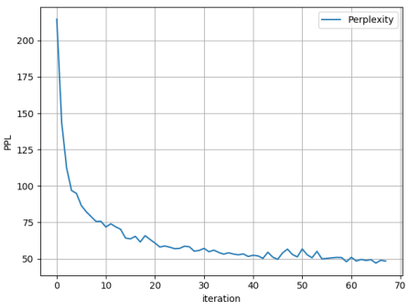
(a) Validation Losses



(b) Total and Language Modelling loss



(c) MC Loss



(d) Perplexity

The Validation and Training LM loss decreased to equal values of around 3.8. Training beyond 3 epochs was found to result in overfittng, this is unsurprising given the large number of trainable parameters in the GPT-2 model and the relatively small dataset. The saved model achieved a perplexity score of 48 on the validation set.

6 BERT Architecture

The BERT classification model is composed of an embedding layer, 12 encoder layers and finally a linear layer for classification. For each input token, 768-long vector embedding is generated. This embedding vector is transformed progressively by each of the encoder blocks, each of which contains a Multi Self-attention layer and a feed-forward layer. The BERT model also includes residual connections around each of these sub-layers as employed in the original Transformer encoder blocks from

Vaswani et al. 2017. The output of the final encoder block is passed to a linear layer which learns to classify the input. The encoder blocks use weights from BERT pre-training, in which BERT is trained to predict missing words using a large corpus (Devlin et al. 2018).

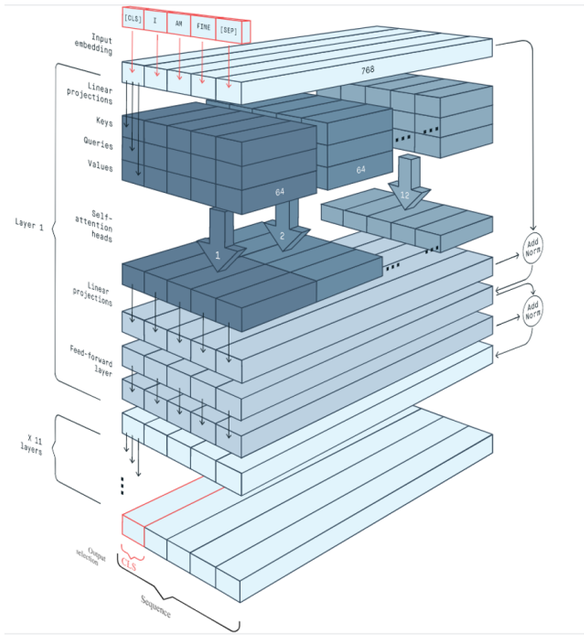


Figure 4: Basic Bert architecture

6.1 Input Embedding

The input sequence is wrapped in 'CLS' and 'SEP' tokens. The 'CLS' token indicates the beginning of an input sequence, while 'SEP' tokens are used to indicate the beginning of a new sentence. The input embedding is then constructed as the sum of the token, segmentation and position embeddings. The segmentation embedding maps each token in the input sequence to a particular sentence. For example, the input sequence in figure 4 is composed of two sentences. The segment embedding for each token can therefore take one of two values, depending on which sentence the token belongs to. The segment embeddings and the 'SEP' tokens are how the model differentiates sentences within the input sequence.
The WordPiece model (Wu et al. 2016) is used to generate the token embeddings. The model has a vocabulary size of 30000 and maps each token to a 768-long vector. Finally, the position embeddings encode each token's position in the input sequence.
The sum of these three embeddings is the input to the first encoder block.

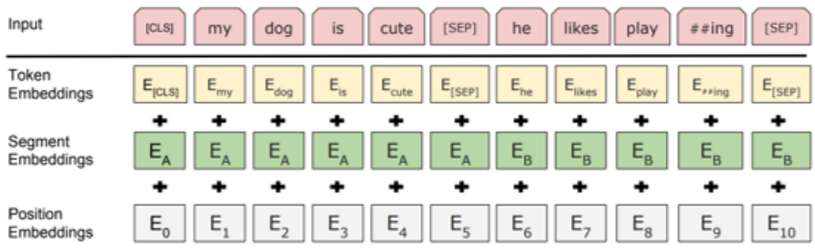


Figure 5: Input representation Devlin et al. 2018

6.2 The Encoder Block

Each of the 12 encoder blocks contains a Multi-Head attention layer and a feed forward network. This attention layer is similar to the one used in the GPT-2 decoder blocks, it uses 12 attention heads and generates Key, Query and Value vectors for each token using predefined matrices. However, unlike GPT-2, BERT uses bidirectional self-attention, meaning it is not constrained in that each token can only attend tokens to its left (that come before it in the input sequence). By processing he entire sequence at once, tokens later on in the input sequence inform the representation of earlier tokens, this is ideal for sentence level tasks such as sentiment classification.

The output of the self-attention layers is linearly projected and passed to a feed-forward neural network. The network is composed of two linear layer that have input and output dimensions (768, 3072) and (3072, 768). A GELU activation function is applied after the first layer. Increasing the size

of the embedding by a factor of 4 increases the representation capacity and achieved state-of-the-art results when implemented in the original transformers architecture (Vaswani et al. 2017)

6.3 The Classification Head

The final encoder output for the 'CLS' token is passed to a pretrained pooling layer with a tanh activation function. The ouput of the pooling layer is fed to a linear layer with 5 output features. The output of this layer is passed through a softmax function, to give the predicted probability of the input sequence belonging to a particular class. Note the outputs of all input tokens can be averaged and then passed to the linear layer but using the 'CLS' token alone was found to be sufficient for this classification task.

7 Training the BERT Classification model

The data was split into training, validation and test datasets of size 8000, 1000 and 830 samples respectively. Thereafter the datasets were loaded with a dataloader, with training data being sampled randomly and validation and test data being sampled sequentially each with a batch size of 32.

The AdamW optimizer was used with a learning rate of 2e-5 and a weight_decay of 0.4. Dropout of hidden layers was implemented with a value of 0.4 to prevent overfitting.

Hyperparameter	Value
Batch size	32
Learning rate	$2e^{-5}$
Hidden layer dropout	0.4
Weight Decay	0.4

The model was evaluated after every epoch and saved if the validation loss achieved was lower than the minimum validation loss in the run.

In addition, a scheduler was used to decrease learning rate over time. This is used to ensure that as training progresses the learning rate will not become too high relative to the gradient causing divergence and therefore increasing stability (Ng n.d.).

7.1 Results of Training Bert

With the scheduler activated training performed well with very little overfitting as seen in the loss graphs. The loss reached a plateau as a result of the scheduler however the validation loss was quite low and an accuracy of 76.8 % achieved.

With the scheduler deactivated, the training and validation loss fell. The model was saved at epoch 4 and did not overfit up until epoch 6. Thereafter the model validation loss increased, and training loss decreased at higher rates.

	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy	Test Accuracy
Scheduler	0.55	0.68	0.81	0.78	0.768
No Scheduler	0.63	0.64	0.773	0.778	0.802

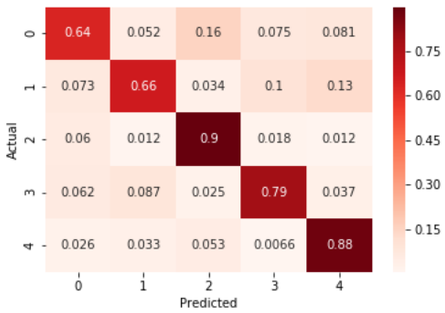
Table 3: Results of Sentiment Analysis Training

As the emotions for afraid and disgust were removed the labels for the emotions are as follows:

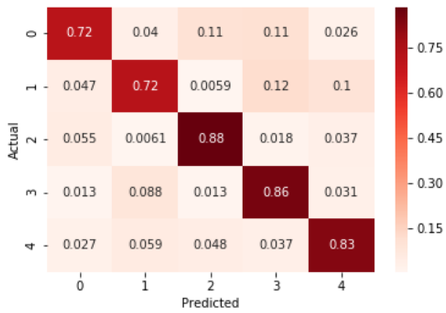
0: No Emotion, 1: Anger, 2, Happiness, 3: Sadness, 4: Surprise

As can be seen the results with the scheduler had decreased performance than without. The confusion matrices for both models show a good spread of results for each class with the model trained without the scheduler having slightly better average results. In general the model has a harder time predicting instances of no emotion and anger, but performs well on instances of surprise, happiness and sadness.

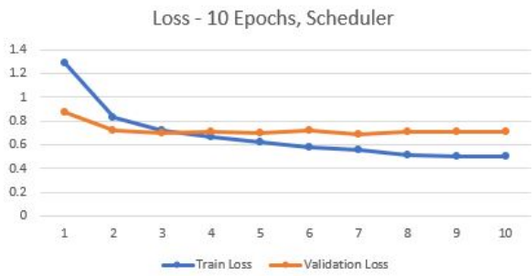
With the model achieving an accuracy of 80.2 % on the test set, the model with no scheduling was decided to be used.



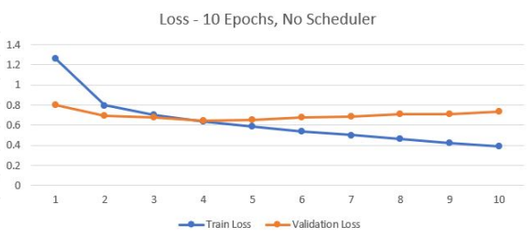
(a) Confusion Matrix: Scheduler Activated



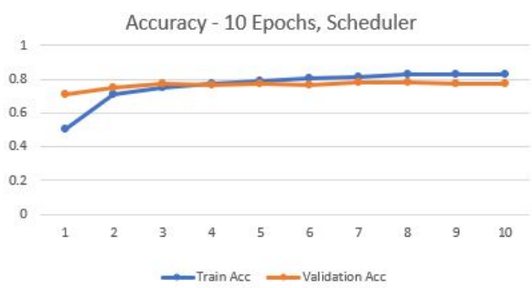
(b) Confusion Matrix: Scheduler Deactivated



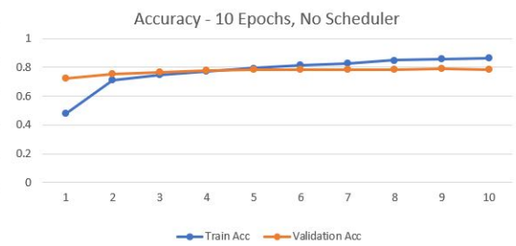
(c) Loss Graphs: Scheduler Activated



(d) Loss Graphs: Scheduler Deactivated



(e) Loss Graphs: Scheduler Activated

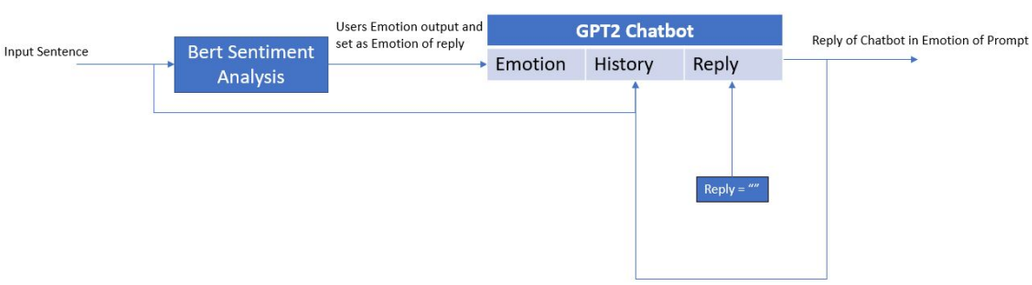


(f) Loss Graphs: Scheduler Deactivated

Figure 6: Training Metrics

8 Combining BERT and GPT-2

An input sequence is classified using the trained BERT model. The output class value and original sequence are input to GPT-2 as the persona and history segments, respectively. Thereafter the reply is generated and appended to the history so that the model can build a conversation history.



9 Text Generation

Having trained the GPT-2 model to predict the next word of a sequence, a decoding strategy is used to generate a coherent continuation from a given context. The probability of a generated sequence of a words given an input sequence of b words is measured as the product of the conditional probabilities of each word given the input sequence:

$$P(x_{1:a+b}) = \prod_{i=1}^{a=b} P(x_i|x_1...x_{i-1}) \tag{2}$$

Assuming the GPT-2 language model has learnt to predict the next word of a sentence well, 'coherent continuations' correspond to sequences with a high probability. However, finding the sequence with the highest probability is not tractable using transformer models (Chen et al. 2018). We evaluate three decoding strategies for open-ended text generation using the trained GPT-2 model.

9.1 Greedy search

Given an input the model generates prediction scores over its vocabulary. A simple approach to generating text would be to choose the most likely word according to this distribution. Having selected a word, it is appended to the original input. Scores are recalculated to determine the next word.

9.2 Top-k sampling

Top-k sampling samples from a truncated version of the generated probability distribution. At each time step, the top-k vocabulary V^k of the distribution is defined as the set of k words that maximise:

$$\sum_{x \in V^k} P(x|x_1...x_{i-1}) \tag{3}$$

Defining $p' = \sum_{x \in V^k} P(x|x_1...x_{i-1})$, the original distribution is re-scaled:

$$P'(x|x_{1:x_{i-1}}) = \begin{cases} P'(x|x_{1:x_{i-1}})/p' & \text{if } x \in V^k \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

The next word is then sampled from this distribution.

9.3 Nucleus sampling

Like top-k sampling, nucleus sampling generates new words using a truncated version of the probability distribution output by the model. Given a distribution $P'(x|x_{1:i-1})$ the top-p vocabulary V^p is defined as the smallest set such that:

$$\sum_{x \in V^p} P(x|x_1...x_{i-1}) \geq p \tag{5}$$

Defining $p' = \sum_{x \in V^p} P(x|x_1...x_{i-1})$, the original distribution is re-scaled:

$$P'(x|x_{1:x_{i-1}}) = \begin{cases} P'(x|x_{1:x_{i-1}})/p' & \text{if } x \in V^p \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Top-k and Nucleus sampling differ only in how they truncate the original distribution. Choosing where to truncate "can be interpreted as determining the generative model's trustworthy prediction zone" (Holtzman et al. 2019).

For high-entropy distributions, the 'trust worthy zone' may include more than k words, while for low-entropy distributions the it may include less. Using Nucleus sampling, "number of candidates considered rises and falls dynamically, corresponding to the changes in the model's confidence region over the vocabulary" (Holtzman et al. 2019).

10 Evaluation

To evaluate the combined model using each decoding strategy we measure the proportion of prompt/response pairs classified as belonging to the same sentiment class. The prompts in the test set are fed to the combined model, and the BERT model output is recorded. This output and the original prompt are passed to the GPT-2 model which generates a reply. The BERT classification model then classifies the sentiment of the generated reply.

The following hyperparameters were used for each decoding strategy:

	Top-k	Top-p	Temperature
Greedy	NA	NA	NA
Top-k	20	0.0	0.7
Nucleus	0.0	0.9	0.7

Table 4: Decoding Hyperparameters

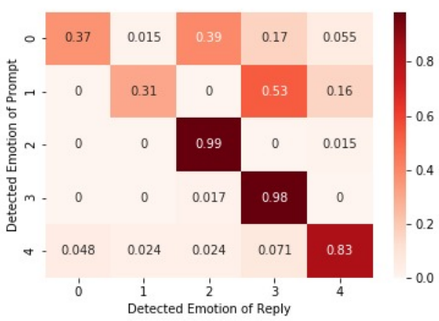
As illustrated in figure 7 Nucleus sampling performed the best, with an overall accuracy of 72.2% versus 70.3% and 61.7% for Top-K and Greedy sampling respectively.

With Nucleus sampling, the model was able to generate replies which matched the emotion of the prompt more than half the time for each emotion, with the exception of anger. In the case of

anger, the model generated a sentence which could be classified as either anger or sadness. As these emotions are admittedly very close in tone this could be seen as a good result, nonetheless.

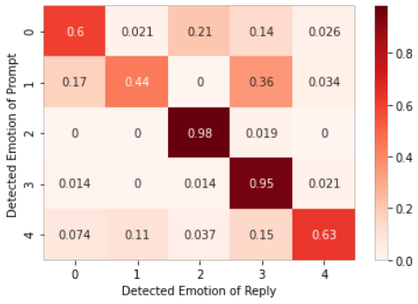
With all decoding strategies the model was able to produce happy and sad responses on request a high proportion of the time(>94%). As these are two particularly strong emotions this is not a surprising result, although a welcome one.

A particularly interesting result is that with greedy search, when requested to produce a 'surprised' response, the model was able to do so 83 % of the time. This could be attributed to the use of exclamation marks and words like 'really', 'oh' being indicative of surprise, meaning the greedy decoding strategy is well suited to generate sequences that will be classified as 'surprise' by BERT.



(a) Confusion Matrix: Greedy Decoding

(b) Confusion Matrix: Top-k Decoding



(c) Confusion Matrix: Nucleus Decoding

Figure 7: Final Performance, 0: No Emotion, 1: Anger, 2, Happiness, 3: Sadness, 4: Surprise

Decoding Strategy	Greedy	Top-k	Nucleus
Accuracy	61.7	70.3	72.2

Table 5: Accuracy using different decoding strategies

A transcript of a conversation using Nucleus Sampling is shown below:

As can be seen the model put out reasonably coherent text that was on topic. In four of the six dialogue pairs, the agent was able to respond with the same emotion as it was given. It must be recognised that as a result of the BERT classifier having an accuracy of 80.2 %, the results of the evaluation could be better or worse in reality.

11 Results and Future Work

The final model was able to recognise the underlying sentiment of prompts with an 80.2 % accuracy. The generated text using the nucleus decoding strategy was able to generate coherent responses classified as having the same predicted emotion the prompt 72.2% of the time.
A key constraint throughout the project was limited amount of labelled conversational data. This was exacerbated by the fact both datasets had to be refined to deal with a large class imbalance. Given the large number of parameters in both BERT and GPT-2 models used, this meant each model was trained for a relatively short period to avoid overfitting. We believe the accuracy of the combined model and the coherence of generated responses would improve dramatically given more data.
Having trained a model to recognise the sentiment of an input sequence, and to generate a reply of a specific sentiment, simply mirroring the sentiment of the input sequence is by no means the only use case. For example, the model could be also used to generate replies of different sentiments until it elicits a desired emotional response from the user.

References

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). *Layer Normalization*. arXiv: 1607.06450 [stat.ML].

Chen, Yining et al. (June 2018). "Recurrent Neural Networks as Weighted Language Recognizers". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2261–2271. DOI: 10.18653/v1/N18-1205. URL: https://www.aclweb.org/anthology/N18-1205.

Devlin, Jacob et al. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL].

Ekman et al. (1987). "Universals and cultural differences in the judgments of facial expressions of emotion". In: *Journal of personality and social psychology*.

Hendrycks, Dan and Kevin Gimpel (2016). *Gaussian Error Linear Units (GELUs)*. arXiv: 1606.08415 [cs.LG].

Holtzman, Ari et al. (2019). *The Curious Case of Neural Text Degeneration*. arXiv: 1904.09751 [cs.CL].

Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv: 1502.03167 [cs.LG].

Ng, Ritchie (n.d.). *Learning Rate Scheduling*¶. URL: https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/lr_scheduling/#citation.

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].

Wu, Yonghui et al. (2016). *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. arXiv: 1609.08144 [cs.CL].

Young, Tom et al. (2017). *Recent Trends in Deep Learning Based Natural Language Processing*. arXiv: 1708.02709 [cs.CL].

Zaeemzadeh, Alireza, Nazanin Rahnavard, and Mubarak Shah (May 2018). *Norm-Preservation: Why Residual Networks Can Become Extremely Deep?*