# Real or Not?
# NLP with Disaster Tweets

W207 Final Group Project
Fall 2020, Section 6
Harvi Singh, Ciaran O’Connor, Omar Kapur

# The Challenge


Given

predict whether new tweets are about real disasters

Columns:

- text - raw text of the tweet
- keyword - a keyword from the tweet
- location - where the tweet was sent from
- target - 1 for about a real disaster, 0 for not (in train.csv only)

# EDA

Data provided by Kaggle:

- train.csv - 7,613 entries
- test.csv - 3,263 entries

Feature selection:

- Used only the text feature
- Location and keyword were not helpful

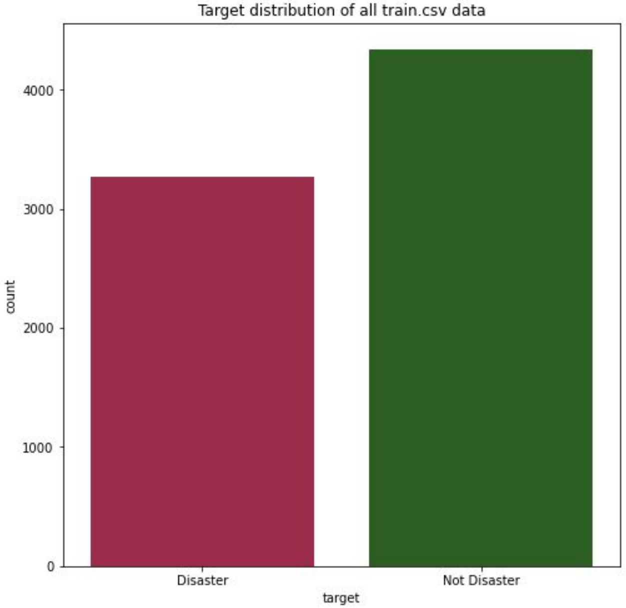
Cross-validation:

- For all models we split training data into train and dev sets (80/20 split)

Null counts for train.csv:

| Column | Null Count | Percent Null |
|---|---|---|
| location | 2533 | 33.3% |
| keyword | 61 | 0.8% |
| text | 0 | 0.0% |
| target | 0 | 0.0% |

| Dataset | Observations |
|---|---|
| train | 6090 |
| dev | 1523 |
| test | 3263 |



Tweets about disasters:

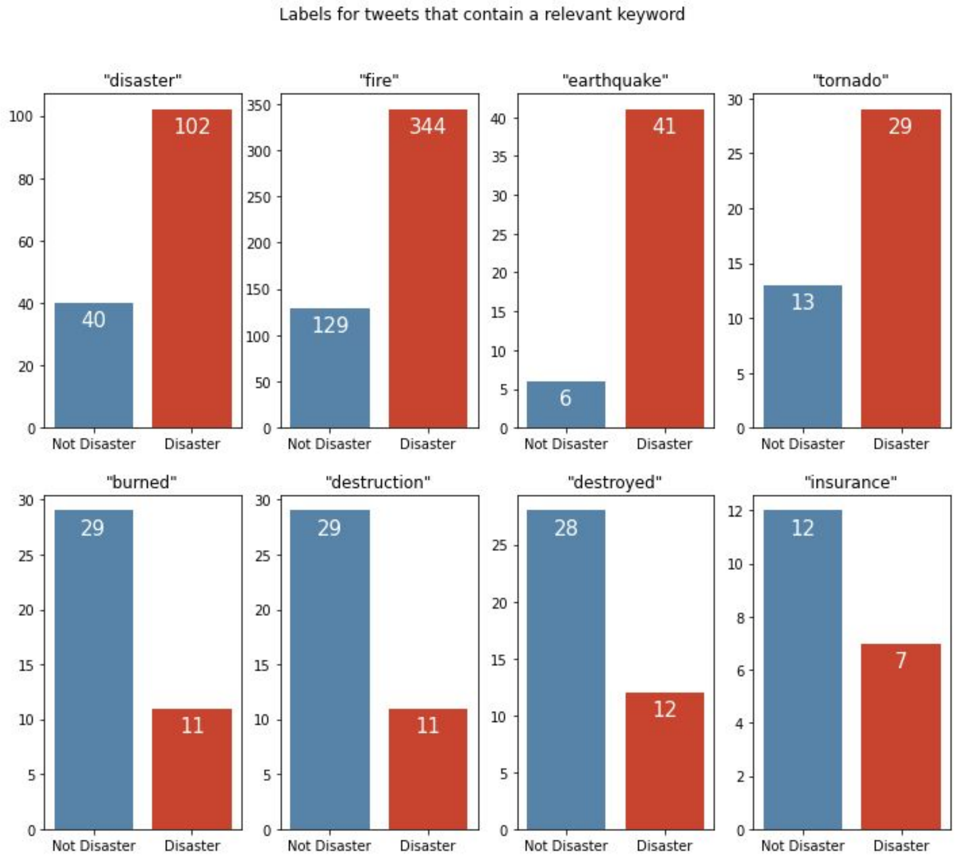

Tweets NOT about disasters:


(wordclouds have stopwords removed)

# EDA

Example tweets that could be easily confused:

> 'I forgot to bring chocolate with me major disaster'
>
> 'four technologies that could let humans survive environmental disaster [link]'
>
> 'our garbage truck really caught on fire lmfao'



# EDA

Some tweets appear to be poorly labeled in the training data:

> ‘Black Eye 9: A space battle occurred at Star M27329 involving 1 fleets totaling 1236 ships with 7 destroyed’ - labeled Disaster

> ‘Black Eye 9: A space battle occurred at Star O784 involving 2 fleets totaling 4103 ships with 50 destroyed’ - labeled Not Disaster

Other tweets have questionable classifications:

> ‘people with a #tattoo out there.. Are u allowed to donate blood and receive blood as well or not?’

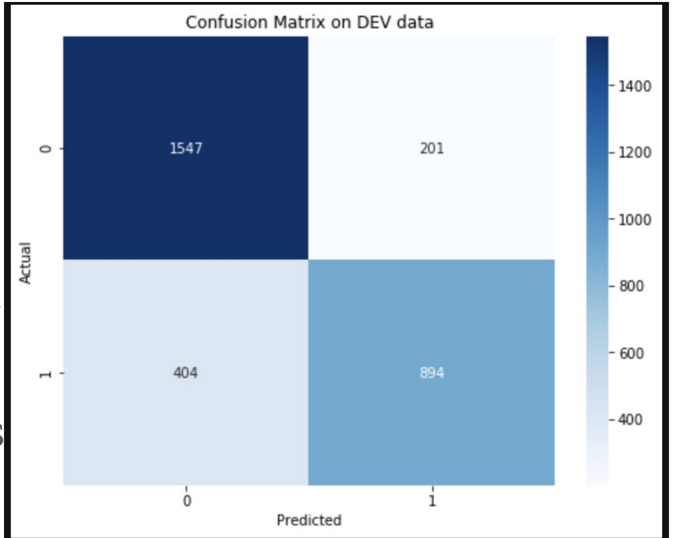
> ‘Bloody insomnia again! Grrrr!! #Insomnia’

> ‘@HopefulBatgirl went down I was beaten and blown up. Then next thing I know Ra Al Ghul brought me back to life and I escaped and for a---’

These tweets are labeled as being about Disasters

# Baseline Approach

1. Apply minimal preprocessing (make text lowercase, remove punctuation and numbers)
2. Transform tweet text using CountVectorizer
3. Predict label using Multinomial Naive Bayes




```
DEV classification report:

              precision    recall  f1-score   support

           0       0.79      0.89      0.84      1748
           1       0.82      0.69      0.75      1298

    accuracy                           0.80      3046
   macro avg       0.80      0.79      0.79      3046
weighted avg       0.80      0.80      0.80      3046
```

**Dev results**
Weighted average f1 score: 0.80
Accuracy: 0.80

**Test (submission) results**
Mean f1 score: 0.79
603rd out of 1,121 submissions

# Improved Approaches

- Better preprocessing
- Describe ‘bag of words’ approaches
- Tf-Idf
- SVM, Logistic Regression, Naive Bayes
- Ensemble Classifier - Using Voting
- Neural Network
- Word Embeddings
- Language Model (BERT)

# Better Preprocessing and TfIdf

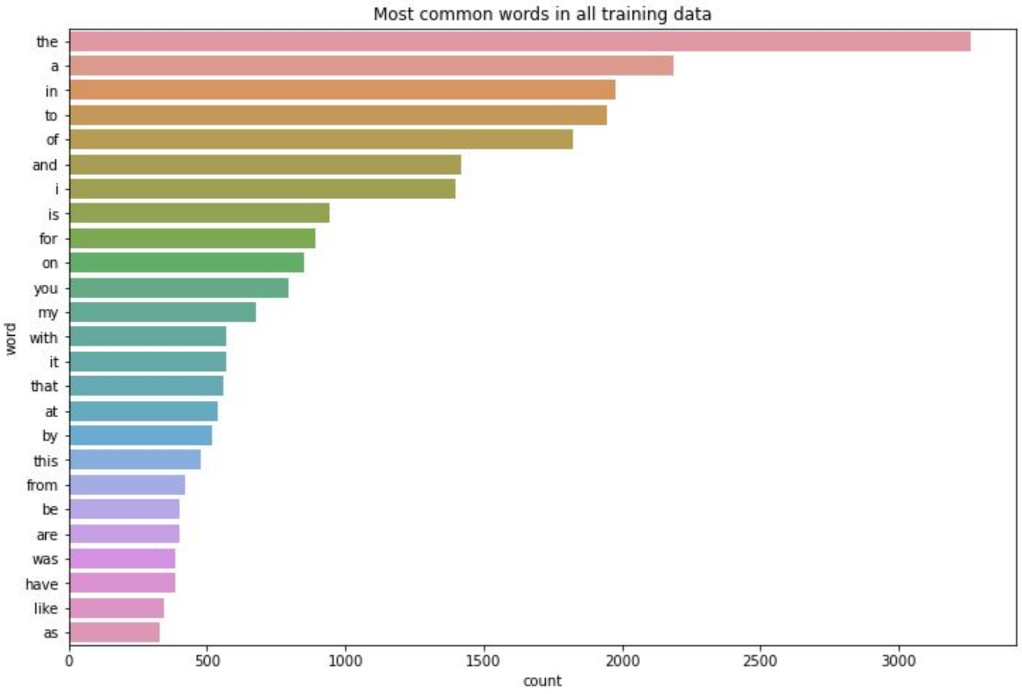
Improvements to the baseline model:

1. Tf-Idf - why does it perform better?

Term Frequency  Inverse Data Frequency

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}} \qquad idf(w) = \log\left(\frac{N}{df_t}\right)$$

2. Better preprocessing function -
   a. Stemming
   b. Special characters
   c. Formating
   d. Common words/stop words



# Simple machine learning and ensemble voting

1. GridSearchCV
2. ensemble - why does it work?
3. L1 vs L2 Regularization?

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

Loss function with L1 regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

Loss function with L2 regularisation

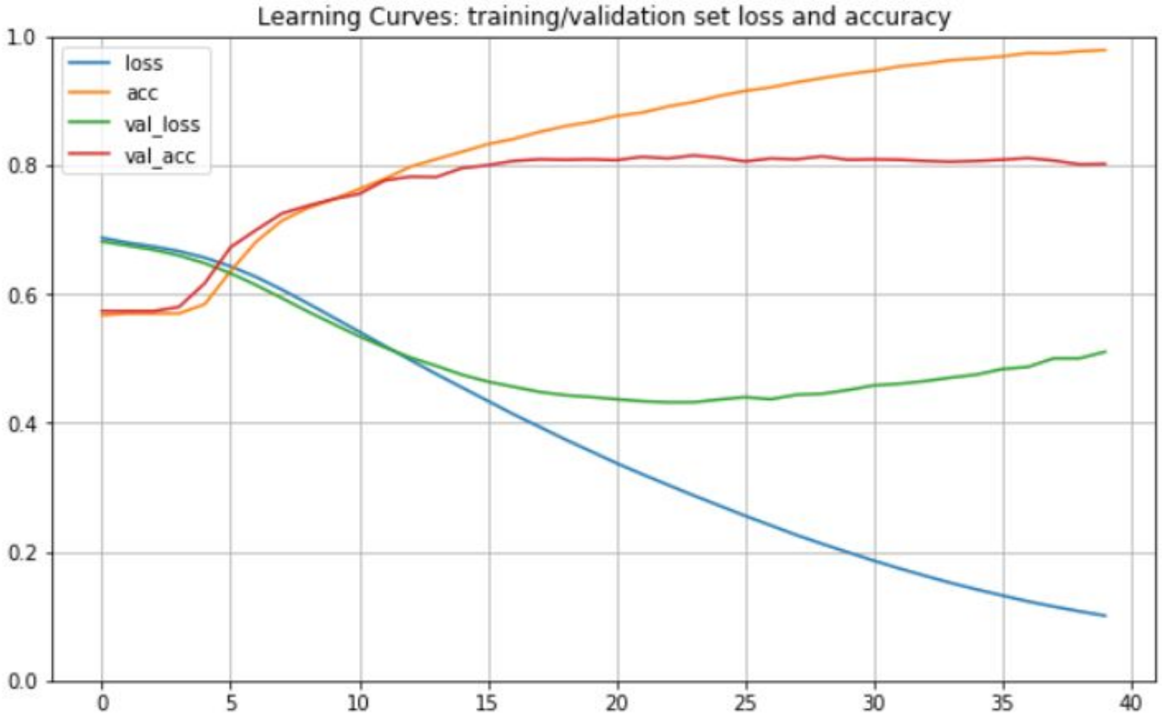
| | Training Set | Dev Set |
|---|---|---|
| multiNB_clf_preprocessed | 0.910 | 0.826 |
| log_clf_TfIdf | 0.836 | 0.825 |
| Baseline | 0.914 | 0.823 |
| Ensemble_voting_clf_TfIdf | 0.901 | 0.821 |
| log_clf_preprocessed_TfIdf | 0.889 | 0.820 |
| Ensemble_voting_clf_preprocessed_TfIdf | 0.906 | 0.820 |
| multiNB_clf_count_vect | 0.894 | 0.819 |
| Ensemble_voting_clf_preprocessed | 0.933 | 0.818 |
| Ensemble_voting_clf_count_vect | 0.933 | 0.817 |
| log_clf_preprocessed | 0.943 | 0.816 |
| svm_clf_preprocessed_TfIdf | 0.932 | 0.816 |
| svm_clf_TfIdf | 0.916 | 0.813 |
| multiNB_clf_TfIdf | 0.904 | 0.812 |
| svm_clf_preprocessed | 0.914 | 0.812 |
| svm_clf_count_vect | 0.921 | 0.811 |
| log_clf_count_vect | 0.952 | 0.810 |
| multiNB_clf_preprocessed_TfIdf | 0.891 | 0.803 |

# Neural Network (Using TensorFlow/Keras)

Highlights:

1. Started with a simple model with 2 hidden layers and 250 neurons each

2. Used learning curves to estimate the appropriate number of 'epochs' and also to identify overfitting



# Neural Network

3. Optimizing the NN utilizing GridSearchCV / RandomizedSearchCV

4. Optimized the number of epochs using the ‘early stopping’ option with a patience of 5

A.

```
random_search_cv.best_params_
```

```
{'n_neurons': 240, 'n_layers': 1, 'learning_rate': 0.01}
```

B. neurons

```
random_search_cv.best_params_
```

```
{'n_neurons': 320, 'n_layers': 1, 'learning_rate': 0.01}
```

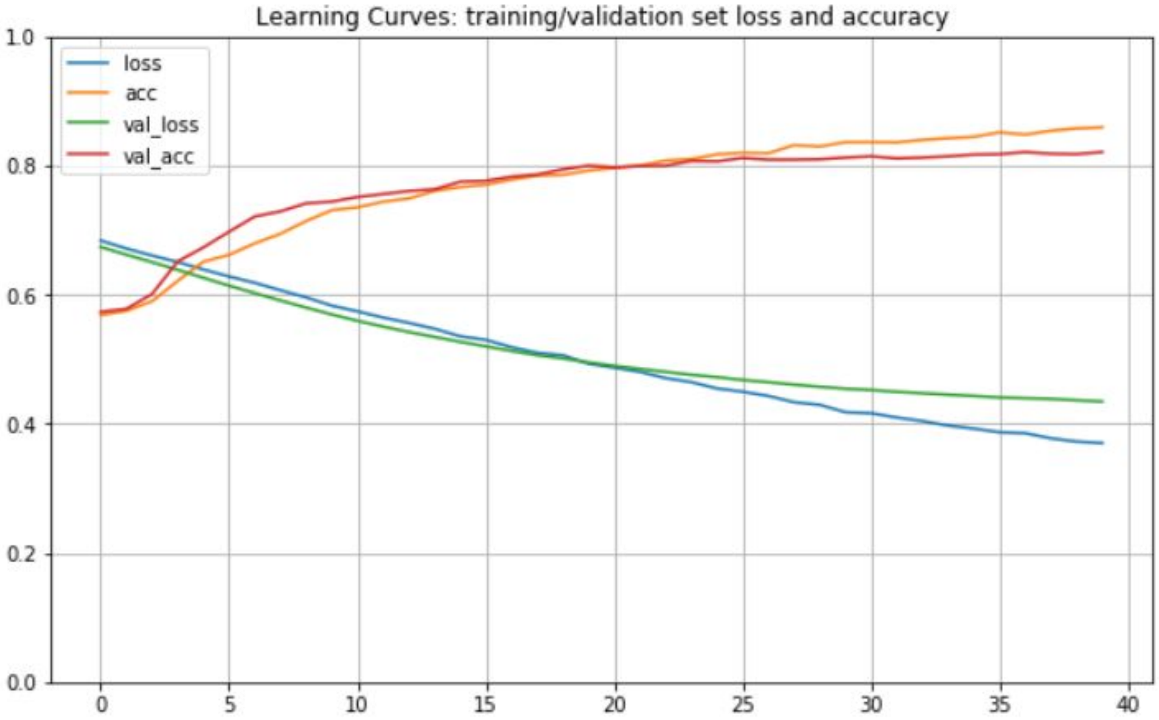
```
random_search_cv.best_score_
```

```
0.7950738867123922
```

# Neural Network

Finally, regularized the model using 2 approaches -

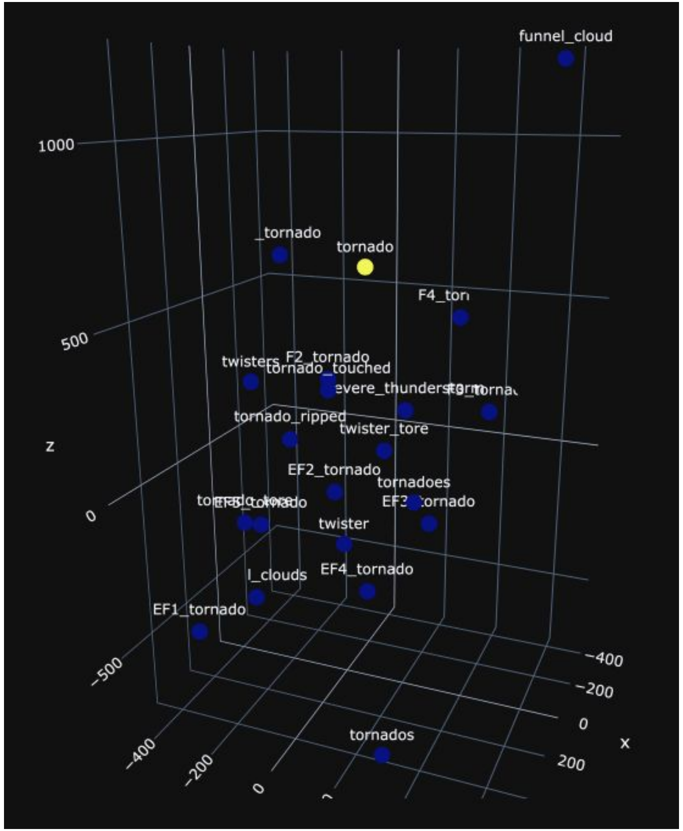
1. L2 on the single layered NN
2. dropout on the model with 2 hidden layers
3. Best NN result was for a 2 layered 320 neuron model with a dropout rate of 0.25 : 82.1% accuracy

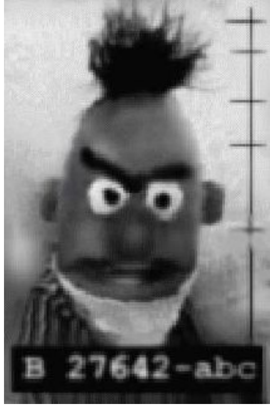


# Word Embeddings

- Numerical representations of text - each word is encoded in a vector
- **Word2Vec** is a popular framework for this that we used
- Words are trained against other words that neighbor them in the input corpus
- Embeddings can be used for text classification (among other things)

We trained word embeddings on our training data and used pre-trained embeddings (trained on 100 billion Google News documents)





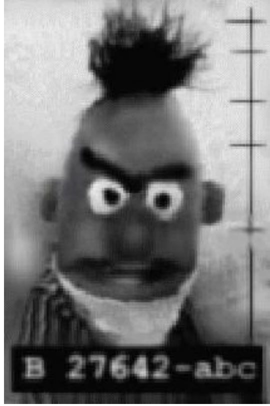

# **B**i-directional **E**ncoder **R**epresentations from **T**ransformers

BERT’s key innovation is applying bidirectional (all at once) training of Transformer; a popular attention model, to language modelling; the approach contrasts to L-R/R-L/both techniques.

BERT comes  pre-trained using a combination of masked language modeling objective and next sentence prediction on a large corpus comprising the Toronto Book Corpus and Wikipedia.

It uses TRANSFER LEARNING - taking a BERT pre-trained neural net model, which has been applied to a given task; which is then used  as the basis for new purpose specific modeling

BERT applications: Next sentence prediction, text classification, sentiment analysis, Named Entity Recognition and Question and Answer




**B**i-directional

**E**ncoder

**R**epresentations from

**T**ransformers

The TRANSFORMER - is an ATTENTION mechanism that learns the contextual relations between words in a text.

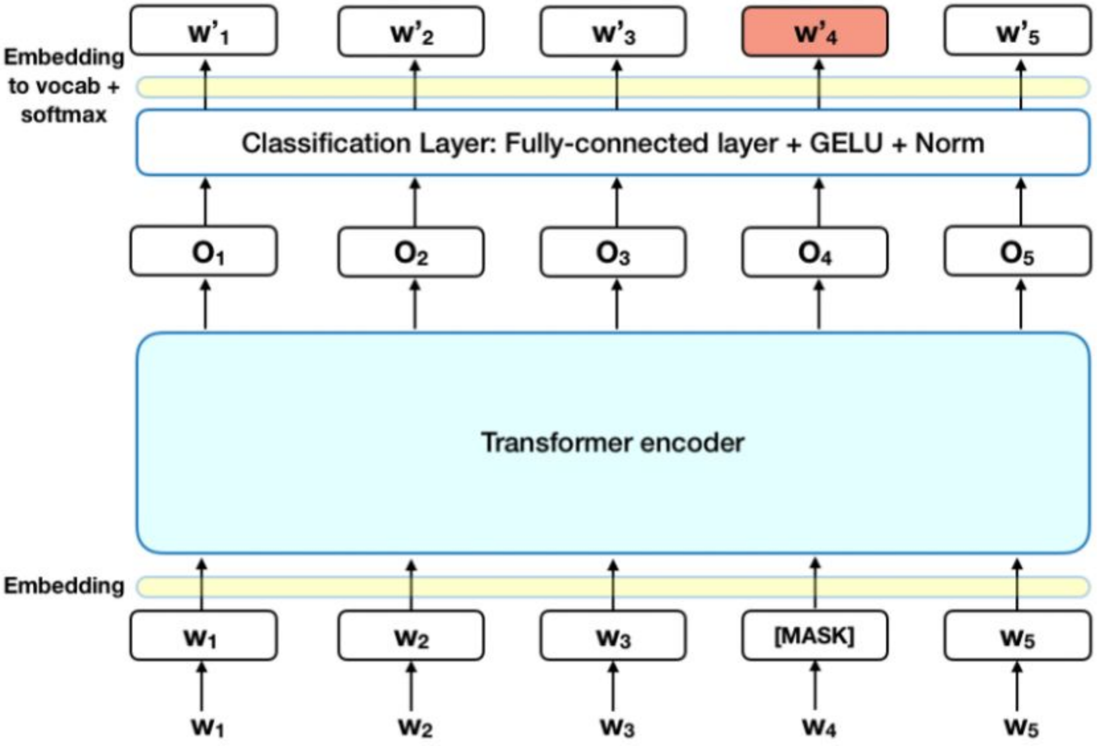
A TRANSFORMER is any task that transforms an input sequence to an output sequence.

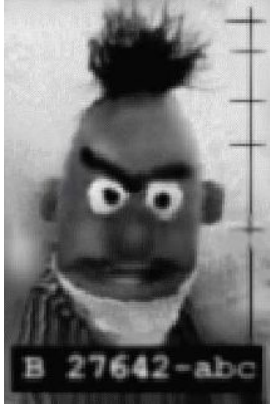
In TRANSFORMERS  a sequence of tokens are first embedded into vectors and then processed by the neural net.

ATTENTION  models focus on a certain region of the corpus with  “high resolution” while perceiving the surrounding text in “low resolution”, and then adjusting the focal point over time.

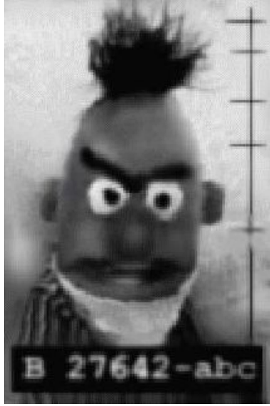
BERT generates a models model using an encoder mechanism.

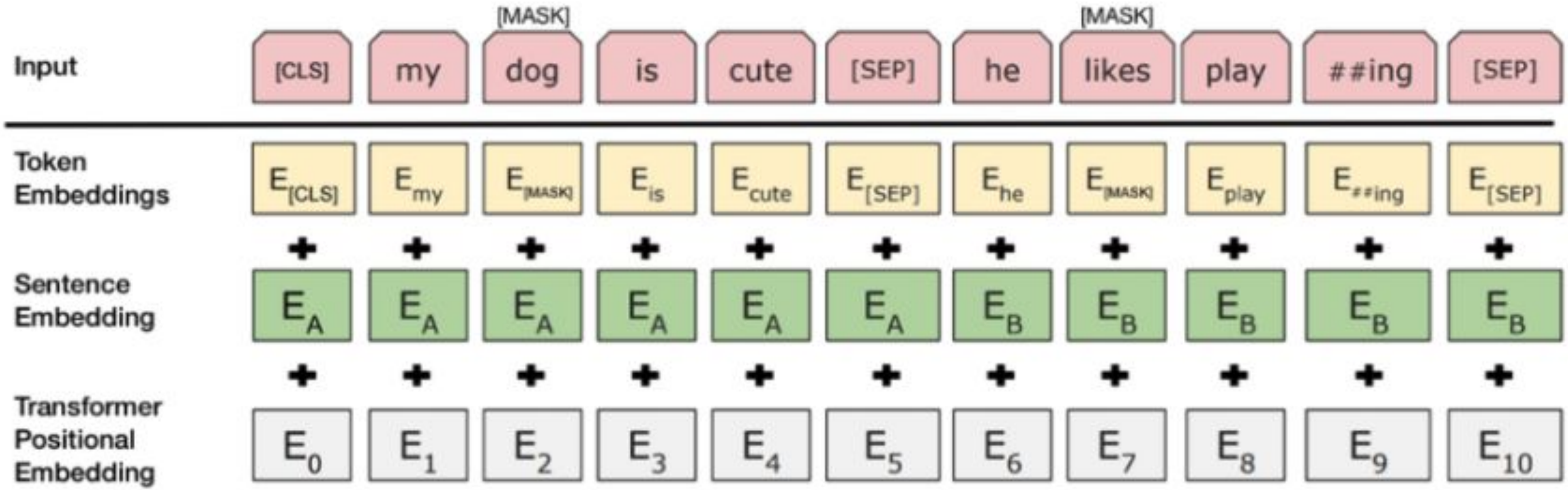
# **BERT** Transformer-encoder



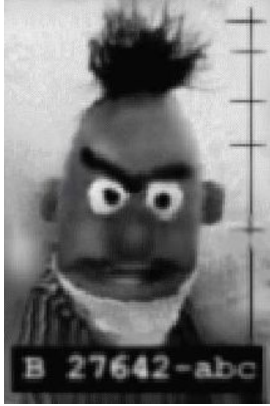
# BERT Transformer-encoder




Source: BERT [Devlin et al., 2018], with modifications

# **BERT** model fixed hyperparameters

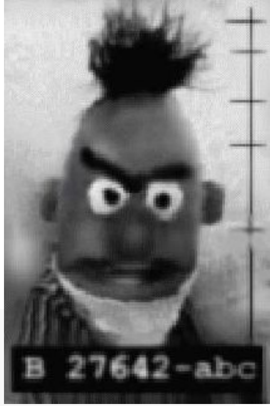



```
print(bert_config)
```

```
BertConfig {
  "attention_probs_dropout_prob": 0.1,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "type_vocab_size": 2,
  "vocab_size": 30522
}
```

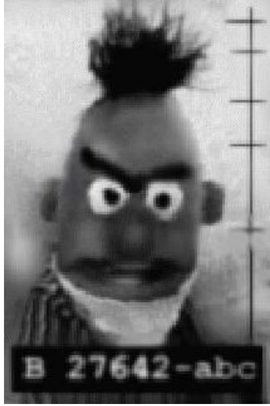
```
#
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=len(y_columns)).cuda()
#
```

# **BERT** little bit of code




```
model = model.train()
scaler = torch.cuda.amp.GradScaler()
tq = tqdm(range(EPOCHS))
for epoch in tq:
    batches = torch.utils.data.DataLoader(training_dataset, batch_size=batch_size, shuffle=True)
    avg_loss = 0.
    avg_accuracy = 0.0
    filterLoss = 0.0
    dataLoader = tqdm(enumerate(batches),total=len(batches),desc='batches progress',leave=False)
    optimizer.zero_grad()
    for i,(x_batch, y_batch) in dataLoader:
        with torch.cuda.amp.autocast():
            y_pred = model(x_batch.to(device), attention_mask=(x_batch>0).to(device), labels=None)[0]
            loss =  F.binary_cross_entropy_with_logits(y_pred,y_batch.to(device))
        scaler.scale(loss).backward()
        if (i+1) % accumulation_steps == 0:              # Wait for even several backward steps
            scaler.step(optimizer)
            scaler.update()
            optimizer.zero_grad()
        if filterLoss == 0.0:
            filterLoss = loss.item()
        else:
            filterLoss = (1.0-nF)*filterLoss + nF*loss.item()
        dataLoader.set_postfix(loss = filterLoss)
        avg_loss += loss.item() / len(batches)
```

# **BERT** how did we innovate here?




BERT is a very sophisticated model -> learning curve to get on top of it.

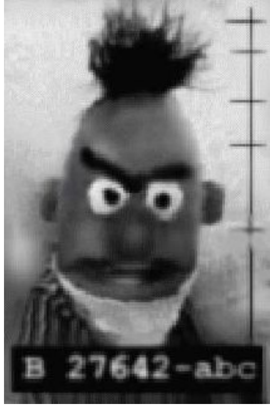
We applied Torch and CUDA with Automatic Mixed Precision processing of matrices.

CUDA - a computational framework for maximal GPU utilisation and performance

GPU calculation performed on Amazon Web Services instance of type p3.2xlarge, which is a Tesla V100-SXM2-16GB.

Fine-tuned Hyperparameters, changing learning rates, Epochs and batch sizes.

Validation prediction F1 Score of 0.84 (rounded) and AUC of 0.89.




# **B**i-directional
# **E**ncoder
# **R**epresentations from
# **T**ransformers

## How are we doing?

| | | | | |
|---|---|---|---|---|
| 264 | Sujith K Nair | 0.82868 | 13 | 2mo |
| 265 | oconnorcpj | 0.82868 | 2 | ~10s |

Your Best Entry 🡅

Your submission scored 0.82868 — Tweet this!

| | | | | |
|---|---|---|---|---|
| 266 | Artem Kuchumov | 0.82837 | 7 | 2mo |
| 267 | BestTUDSIChittana | 0.82837 | 4 | 19d |

# Conclusion

Kaggle Submission Performance

- Baseline - 79 %  average f1 (45 percentile on kaggle)
- Best one from ML - 79.7% average f1 (55 percentile on Kaggle)
- BERT (champ) - 82.7 % average f1 (83 percentile on Kaggle)

Takeaways:

- The relatively small size of dataset made modeling difficult
- Hardships in applying complex methods - lots of technology
- Getting a big improvement on our baseline was very difficulty - required a significant jump in complexity
- Competency