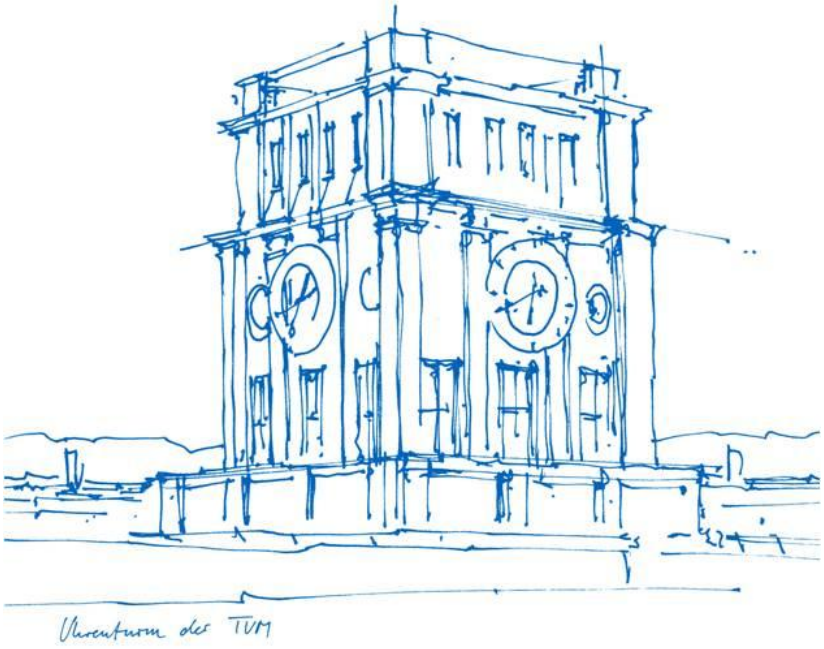
Comparing **Sentiment Analysis** of Discourse Units and Sentences

Group 9: Li Canchen, Hendrik Pauthner, Tim Pfeifle
Technische Universität München
Informatics
Research Group Social Computing
Munich, 19, June, 2019



Progress Report

- Preprocessing code finished.
  - lemmatize
  - stopwords remove
  - length padding
- MilNet model running.
  - XLING feature based
  - word2vec feature based

➔ Memory usage not suitable for Colab

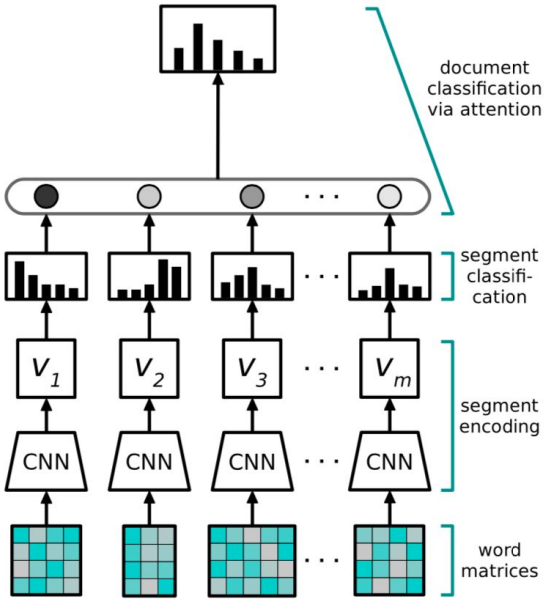
➔ Model accuracy not satisfiable

Embedding Layer

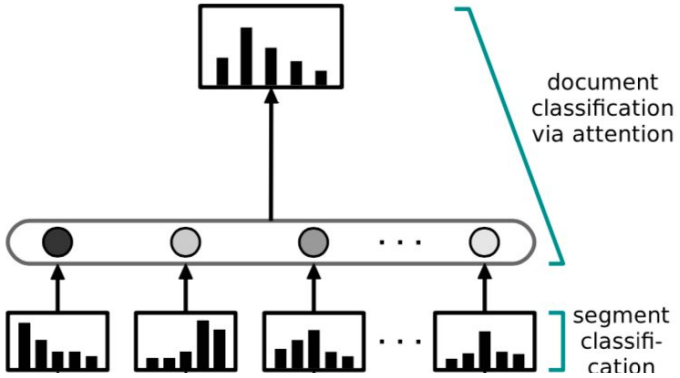
(None, 20, 30, 300) -> 272.8 GB in total !

```
embedding_layer = tf.keras.layers.Embedding(
      input_dim=w2v.shape[0],
      output_dim=w2v_len,
      weights=[w2v],
      input_length=max_word,
      trainable=False
)(model_input)
```

(None, 20, 30) -> Less than 2 GB .



Parameter not Found ?



classification_of_segments

(None, 20, 5)

attention_weights

(None, 20, 1)

tf.matmul

(None, 5)

Parameter not Found ?

classification_model = …

attention_model = …

Option 1: tf.matmul

```
weighted_layer = tf.keras.layers.Lambda(tf.matmul,
                                        arguments={'transpose_a': True,
                                                   'b':attention_model})(classification_model)
squeeze_layer = tf.keras.layers.Lambda(tf.squeeze, arguments={'axis':
-1})(weighted_layer)
```

Option 2: tf.keras.layers.Multiply

```
weighted_layer = tf.keras.layers.Multiply()([attention_model, classification_model])
reduce_layer = tf.keras.layers.Lambda(tf.reduce_mean, arguments={'axis':1})
                                                   (weighted_layer)
```

Parameter not Found ?

```
Constructing Model ...
Model Constructed. Compiling ...
Model Compiled.
Model: "model_95"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_73 (InputLayer)        [(None, 20, 30)]          0
_________________________________________________________________
embedding_3 (Embedding)      (None, 20, 30, 300)       58749600
_________________________________________________________________
model_92 (Model)             (None, 20, 210)           5061000
_________________________________________________________________
model_94 (Model)             (None, 20, 5)             21100
_________________________________________________________________
lambda_404 (Lambda)          (None, 5, 1)              0
_________________________________________________________________
lambda_405 (Lambda)          (None, 5)                 0
=================================================================
Total params: 63,831,700
Trainable params: 5,073,700
Non-trainable params: 58,758,000
_________________________________________________________________
```

```
Constructing Model ...
Model Constructed. Compiling ...
Model Compiled.
Model: "model_47"
_____________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=============================================================================================
input_25 (InputLayer)        [(None, 20, 30)]          0
_____________________________________________________________________________________________
embedding_1 (Embedding)      (None, 20, 30, 300)       58749600    input_25[0][0]
_____________________________________________________________________________________________
model_44 (Model)             (None, 20, 210)           5061000     embedding_1[0][0]
_____________________________________________________________________________________________
sequential_161 (Sequential)  (None, 20, 300)           325800      model_44[1][0]
_____________________________________________________________________________________________
model_45 (Model)             (None, 20, 1)             1812000     sequential_161[0][0]
_____________________________________________________________________________________________
model_46 (Model)             (None, 20, 5)             21100       model_44[1][0]
_____________________________________________________________________________________________
multiply_1 (Multiply)        (None, 20, 5)             0           model_45[1][0]
                                                                   model_46[1][0]
_____________________________________________________________________________________________
lambda_201 (Lambda)          (None, 5)                 0           multiply_1[0][0]
=============================================================================================
Total params: 65,969,500
Trainable params: 7,211,500
Non-trainable params: 58,758,000
_____________________________________________________________________________________________
```

SparseCategoricalEntropy

```
model.compile(
    […]
    loss=tf.keras.losses.CategoricalCrossentropy(),
    […]
)
```

ValueError: You are passing a target array of shape (1000, 1) while using as loss `categorical_crossentropy`. `categorical_crossentropy` expects targets to be binary matrices (1s and 0s) of shape (samples, classes)

→ **from keras.utils import to_categorical**

y_binary = to_categorical(y_int)

Alternatively, you can use the loss function `*sparse_categorical_crossentropy*` instead, which does expect integer targets.

SparseCategoricalEntropy

tf.keras.losses.SparseCategoricalEntropy

- Available after version 1.14-rc
- Compute the categorical cross entropy between feature and integer labels

```
cce = tf.keras.losses.SparseCategoricalCrossentropy()
loss = cce(
  [0, 1, 2],
  [[.9, .05, .05], [.5, .89, .6], [.05, .01, .94]]) # Loss: 0.3239
```

tf.keras.losses.CategoricalEntropy

- Compute the categorical cross entropy between feature and one-hot labels
- Compute the categorical cross entropy between feature and integer labels (before version 1.14-rc)

```
cce = tf.keras.losses.CategoricalCrossentropy()
loss = cce(
  [[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]],
  [[.9, .05, .05], [.5, .89, .6], [.05, .01, .94]]) # Loss: 0.3239
```

Solved Problems

- Appending to `np.array` too slow: Do not use `np.append` frequently, use `np.zeros` to allocate the array first
- Dumping lots of `npy` files requires lots of time: Use hdf5 file to store bunch of `np.array` into a single file
- Cannot load dataset at once: Use `fit_generator` instead of `fit`.

Still Existing Problems

- Model accuracy: Barely higher than 20% for word2vec features

XLING Feature Based Model

- On 1000 reviews
- 5 Balanced classes
- Accuracy:
  - Validation: 75%
- Macro F1-Score: 0.67

Next: Integrate XLING into Dataloader

→ Train on whole dataset

Next Steps

- Keep debugging the model with small dataset
- Train on full dataset with sentence/EDU features
- Integrate data loading / embeddings process into a continuous **df.data** pipeline
- Use different segment embeddings e.g. BERT, ELMo

Muchas Gracias!

Li Canchen, Hendrik Pauthner, Tim Pfeifle

Munich, 22. May 2019