Week 7: Accounting for Context

Substantive topic: Ethnic politics in China

Naiyu Jiang

Text classification – my tiny corpus

- Social media posts (Chinese) on ethnic minority forum
- Eight categories: networking, job, culture, language, religion, admin, policy/politics, other
- Seven ethnic groups: Tujia, Hui, Zhuang, Man, Miao, Zhuang, Yi

```
[9] path_ex = 'drive/MyDrive/Colab Notebooks/week-7/label2.csv'
    df_ex = pd.read_csv(path_ex, header = None, names=['sentence', 'category_id', 'category', 'ethnic'])
```

```
df_ex.sample(10)
```

	sentence	category_id	category	ethnic
434	水西简史，，，，，，，，，，，，，，，	3	culture	Yi
460	如果说彝族曾经强大过 如果说彝族曾经强大过，肯定是在云贵，而不是在四川。当南诏称雄南亚东南亚...	3	culture	Yi
663	55首好听的壮语歌曲 55首好听的壮语歌曲，，，，，，	3	culture	Zhuang
339	【工作】求 会苗语+英语 的朋友，有的话didi我呀 【工作】求 会苗语+英语 的朋友，有的...	2	job	Miao
631	只会说壮语算不算文明人？近段时间看到一些地方和学校常有这样的标语 “请说普通话，做个文明人...	4	language	Zhuang
408	星星用彝语怎么说？ “改” 第一声（给）（弱）第三声 我们这里是这样叫的	4	language	Yi
232	专家解读最早的辽河流域居民 专家解读最早的辽河流域居民	3	culture	Man
376	雷山苗族医药传人 他叫王增世，今年57岁，苗族，家住贵州省黔东南州雷山县望丰乡公统村，是家族...	3	culture	Miao
693	基础词汇语音 13" 9"\r\n \r\n ...	4	language	Zhuang
119	厦门艾梵咖啡招聘 厦门艾梵咖啡招聘，	2	job	Hui

Text classification – my tiny corpus

- Social media posts (Chinese) on ethnic minority forum
- Eight categories: networking, job, culture, language, religion, admin, policy/politics, other
- Seven ethnic groups: Tujia, Hui, Zhuang, Man, Miao, Zhuang, Yi

```
[9] path_ex = 'drive/MyDrive/Colab Notebooks/week-7/label2.csv'
    df_ex = pd.read_csv(path_ex, header = None, names=['sentence', 'category_id', 'category', 'ethnic'])
```

```
df_ex.sample(10)
```

	sentence	category_id	category	ethnic
434	水西简史，，，，，，，，，，，，，，，	3	culture	Yi
460	如果说彝族曾经强大过 如果说彝族曾经强大过，肯定是在云贵，而不是在四川。当南诏称雄南亚东南亚...	3	culture	Yi
663	55首好听的壮语歌曲 55首好听的壮语歌曲，，，，，，	3	culture	Zhuang
339	【工作】求 会苗语+英语 的朋友，有的话didi我呀 【工作】求 会苗语+英语 的朋友，有的...	2	job	Miao
631	只会说壮语算不算文明人？近段时间看到一些地方和学校常有这样的标语 “请说普通话，做个文明人...	4	language	Zhuang
408	星星用彝语怎么说？ “改” 第一声（给）（弱）第三声 我们这里是这样叫的	4	language	Yi
232	专家解读最早的辽河流域居民 专家解读最早的辽河流域居民	3	culture	Man
376	雷山苗族医药传人 他叫王增世，今年57岁，苗族，家住贵州省黔东南州雷山县望丰乡公统村，是家族...	3	culture	Miao
693	基础词汇语音 13" 9"\r\n \r\n ...	4	language	Zhuang
119	厦门艾梵咖啡招聘 厦门艾梵咖啡招聘，	2	job	Hui

Text classification using LSTM

Why LSTM?

- It is effective in memorizing important information;
- We can use a multiple word string to find out the class to which it belongs;
- To find out the actual meaning in input string and will give the most accurate output class.

```
# delete components other than Chinese characters
def remove_punctuation(line):
    line = str(line)
    if line.strip()=='':
        return ''
    rule = re.compile(u"[^a-zA-Z0-9\u4E00-\u9FA5]")
    line = rule.sub('',line)
    return line

def stopwordslist(filepath):
    stopwords = [line.strip() for line in open(filepath, 'r', encoding='utf-8').readlines()]
    return stopwords

# load Chinese stopwords
stopwords = stopwordslist("drive/MyDrive/Colab Notebooks/week-7/stopwords.txt")
```

```
[ ] df_ex['clean_sentence'] = df_ex['sentence'].apply(remove_punctuation)
    df_ex.sample(10)
```

```
# define model
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(8, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
3/3 [==============================] - 0s 102ms/step - loss: 1.4479 - accuracy: 0.4857
Test set
  Loss: 1.448
  Accuracy: 0.486
```

Text classification using LSTM

- Pre-process Chinese texts
- Cut words and remove stop words
- Split dataset
- Define model
- Train model
- Evaluation

```
# delete components other than Chinese characters
def remove_punctuation(line):
    line = str(line)
    if line.strip()=='':
        return ''
    rule = re.compile(u"[^a-zA-Z0-9\u4E00-\u9FA5]")
    line = rule.sub('',line)
    return line

def stopwordslist(filepath):
    stopwords = [line.strip() for line in open(filepath, 'r', encoding='utf-8').readlines()]
    return stopwords

# load Chinese stopwords
stopwords = stopwordslist("drive/MyDrive/Colab Notebooks/week-7/stopwords.txt")
```

```
[ ] df_ex['clean_sentence'] = df_ex['sentence'].apply(remove_punctuation)
    df_ex.sample(10)
```

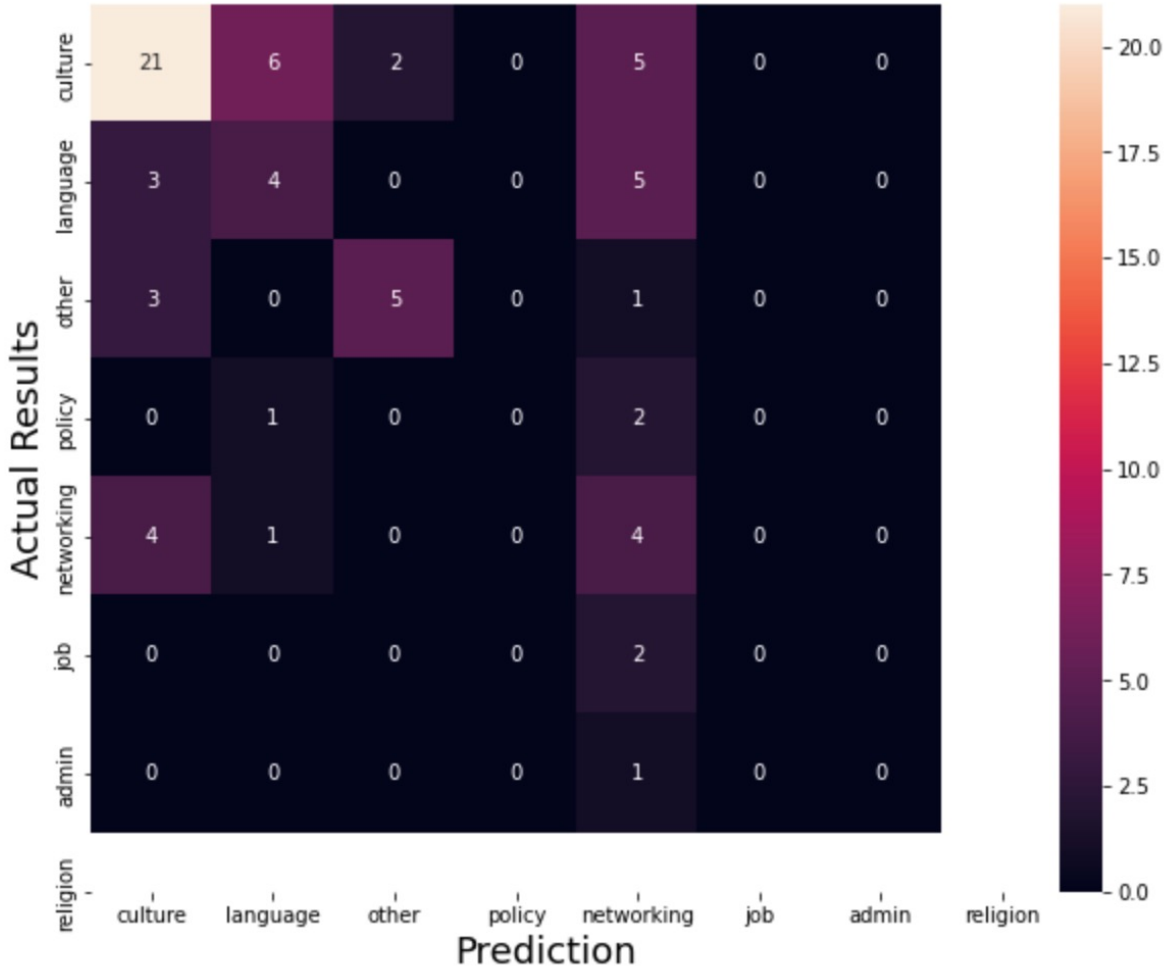
```
# define model
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(8, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
3/3 [==============================] - 0s 102ms/step - loss: 1.4479 - accuracy: 0.4857
Test set
  Loss: 1.448
  Accuracy: 0.486
```

Text classification using LSTM

```
Text(0.5, 51.0, 'Prediction')
```




```
[ ]  from  sklearn.metrics import classification_report

     print('accuracy %s' % accuracy_score(y_pred, Y_test))
     print(classification_report(Y_test, y_pred, target_names=cat_i
```

```
accuracy 0.4857142857142857
              precision    recall  f1-score   support

     culture       0.68      0.62      0.65        34
    language       0.33      0.33      0.33        12
       other       0.71      0.56      0.63         9
      policy       0.00      0.00      0.00         3
  networking       0.20      0.44      0.28         9
         job       0.00      0.00      0.00         2
       admin       0.00      0.00      0.00         1

    accuracy                           0.49        70
   macro avg       0.28      0.28      0.27        70
weighted avg       0.50      0.49      0.49        70
```

Text classification using BERT

- Convert all data into torch tensors
- Choose a BERT model to fine-tune: **“bert-base-chinese”**
- Define model
- Model training: loss function + optimizer
- Evaluation

```
# Select a batch size for training. For fine-tuning BERT on a specific task, the authors recommend a batch size of 16 or 32
batch_size = 16

# Create an iterator of our data with torch DataLoader. This helps save on memory during training because, unlike a for loop,
# with an iterator the entire dataset does not need to be loaded into memory

train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=batch_size)

validation_data = TensorDataset(validation_inputs, validation_masks, validation_labels)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_sampler, batch_size=batch_size)
```

```
model = BertForSequenceClassification.from_pretrained("bert-base-chinese", num_labels=9)
model.cuda()
```

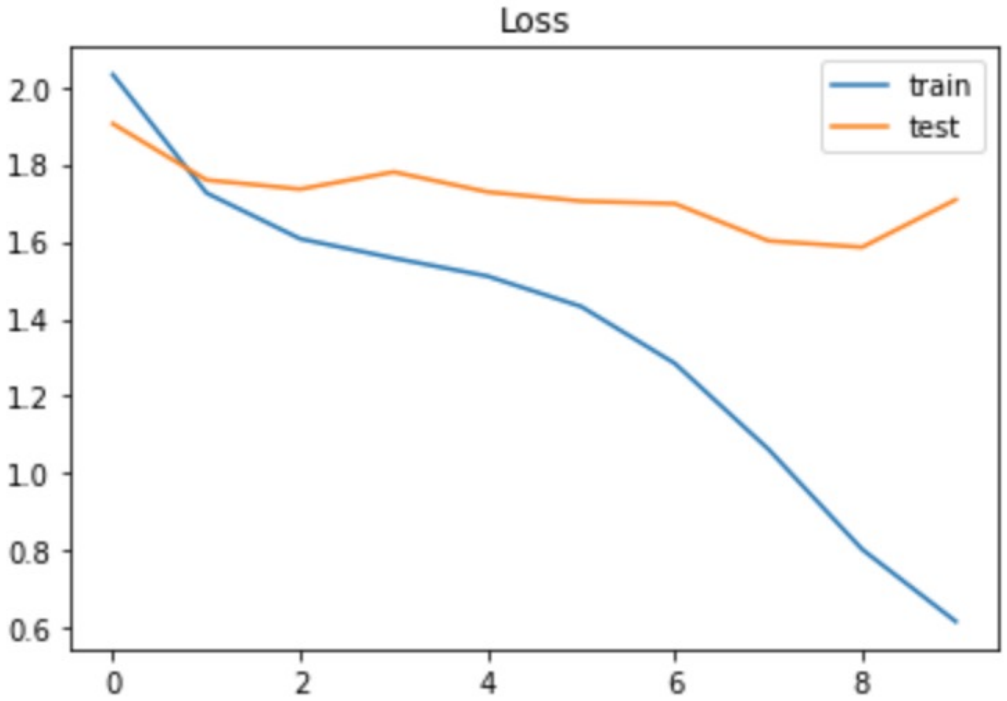
Text classification using BERT

```
======== Epoch 3 / 4 ========
Training...

  Average training loss: 0.90
  Training epcoh took: 0:00:14

Running Validation...
  Accuracy: 0.70
  Validation took: 0:00:01

======== Epoch 4 / 4 ========
Training...

  Average training loss: 0.74
  Training epcoh took: 0:00:14

Running Validation...
  Accuracy: 0.71
  Validation took: 0:00:01

Training complete!
```

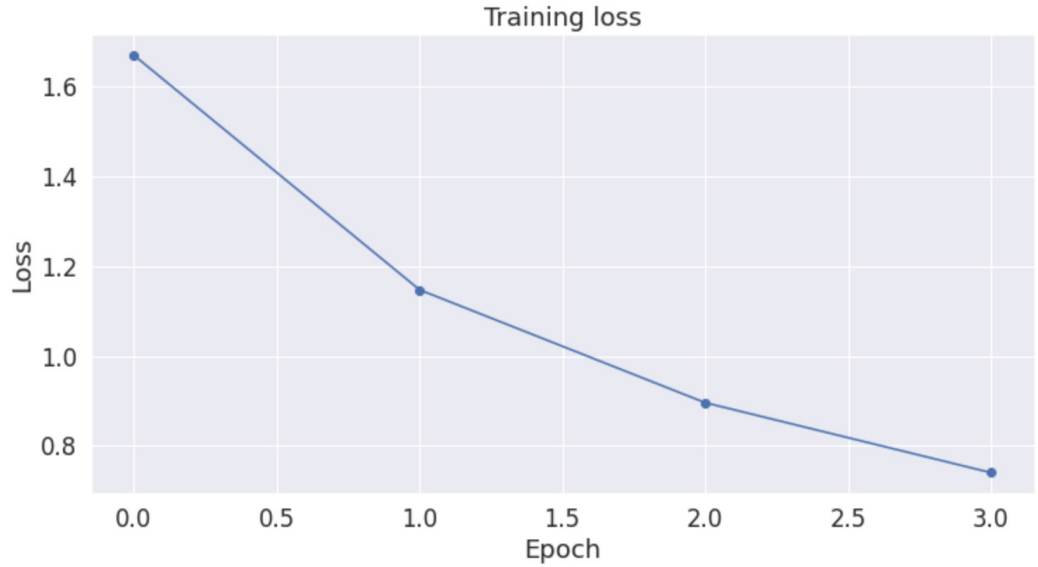
Transformers?

- It randomly masks words in a particular context, and predicts them;
- Learn from words in all positions, meaning the entire sentence;
- Process an input sequence of words all at once.

LSTM vs. BERT

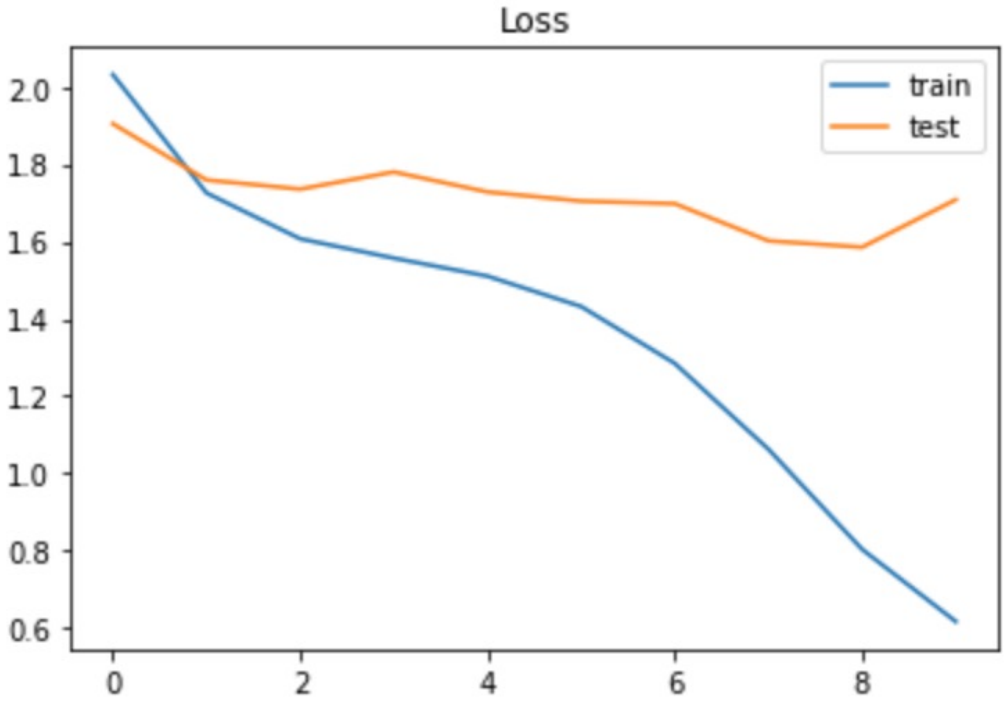



```
# Label the plot.
plt.title("Training loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")

plt.show()
```

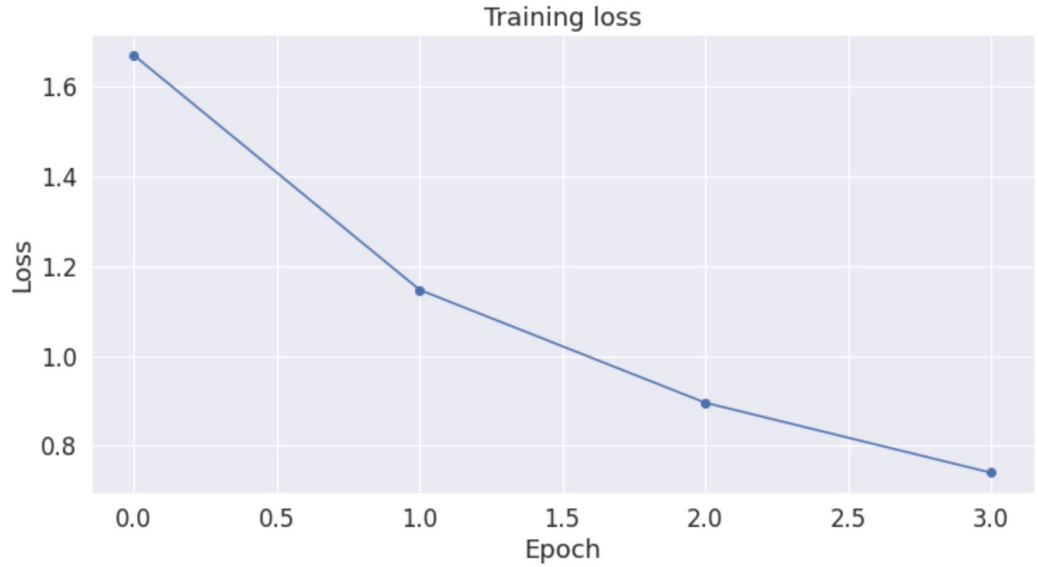


- BERT outperforms LSTM in this case, but is BERT always better than LSTM?
- Given a small dataset, can we use a large pre-trained model like BERT and get better results than simple models?

LSTM vs. BERT



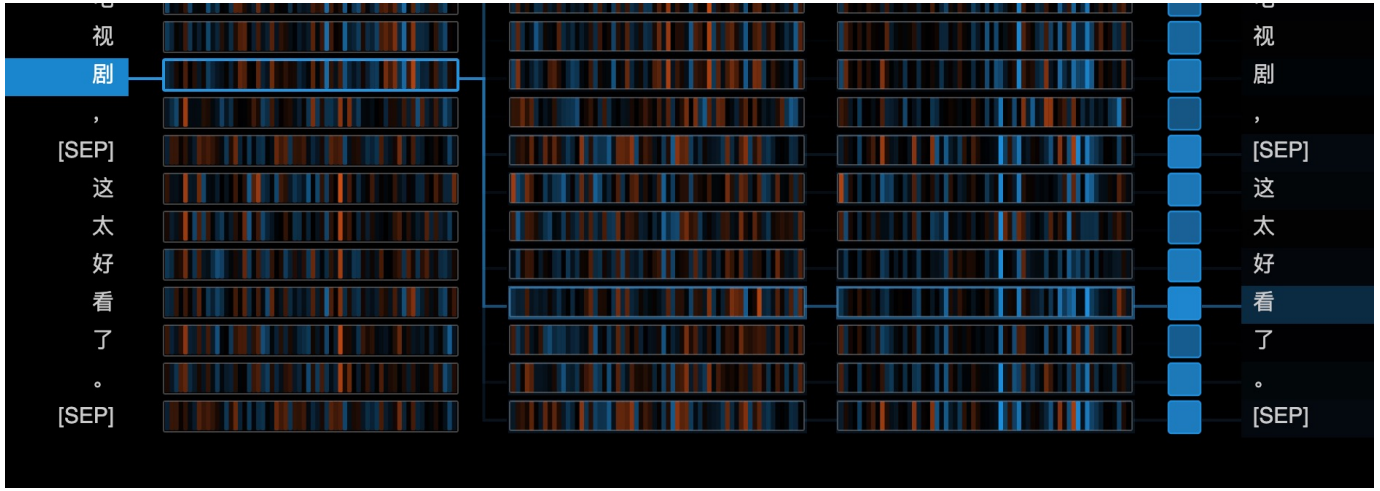

```
# Label the plot.
plt.title("Training loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")

plt.show()
```



- BERT outperforms LSTM in this case, but is BERT always better than LSTM?
- Given a small dataset, can we use a large pre-trained model like BERT and get better results than simple models?

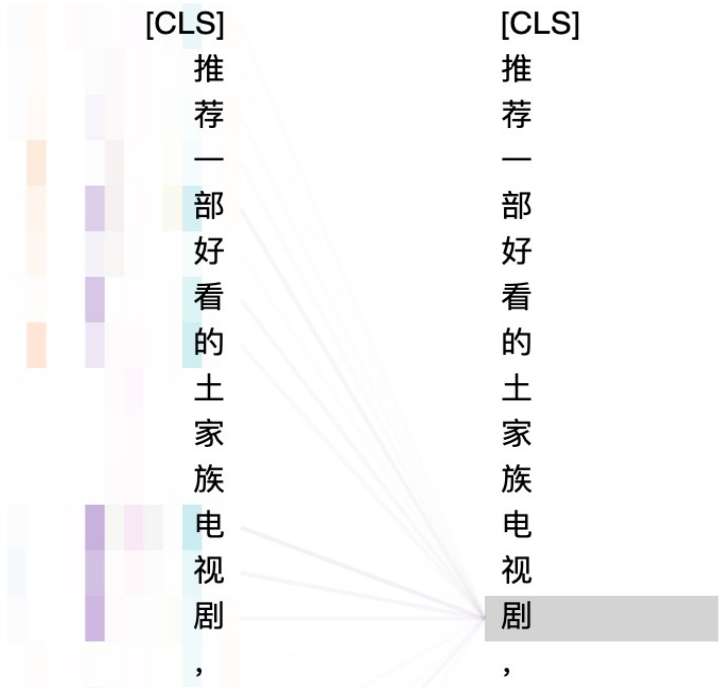
Visualize BERT using the bertviz package

```
[16] sentence_a = "推荐一部好看的土家族电视剧，"  Recommend a good Tujia TV series
     sentence_b = "这太好看了。" This is so good!

     inputs = tokenizer.encode_plus(sentence_a, sentence_b, return_tensors='pt', add_specia
     input_ids = inputs['input_ids']
     token_type_ids = inputs['token_type_ids']
     attention = model(input_ids, token_type_ids=token_type_ids)[-1]
     sentence_b_start = token_type_ids[0].tolist().index(1)
     input_id_list = input_ids[0].tolist() # Batch index 0
     tokens = tokenizer.convert_ids_to_tokens(input_id_list)
```




```
head_view(attention, tokens, sentence_b_start)
```

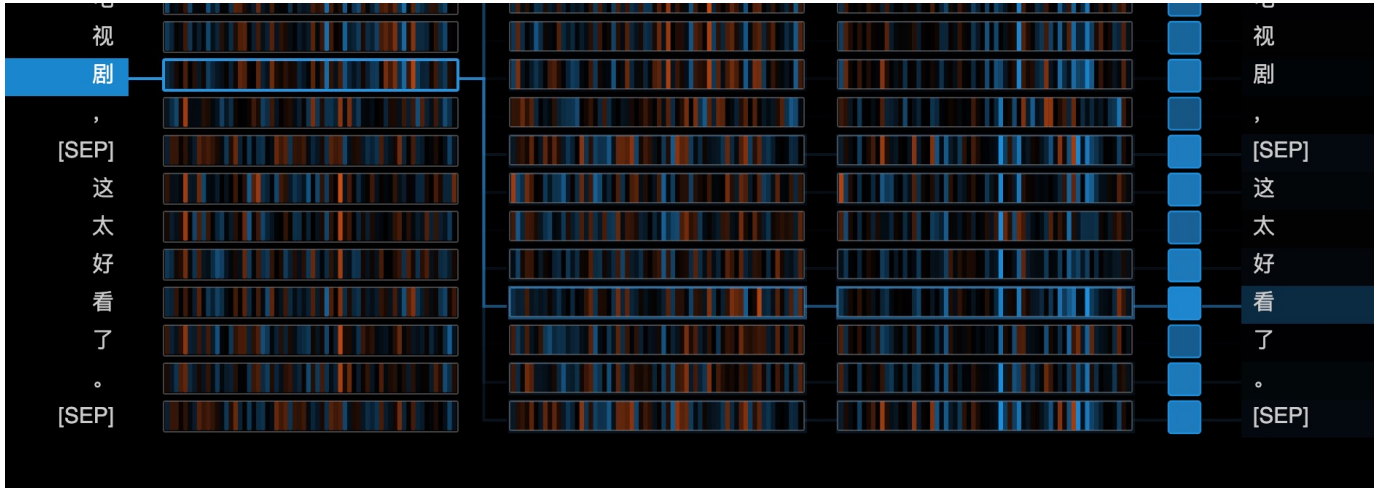
Layer: 9 Attention: All



BERT can learn the “context”.

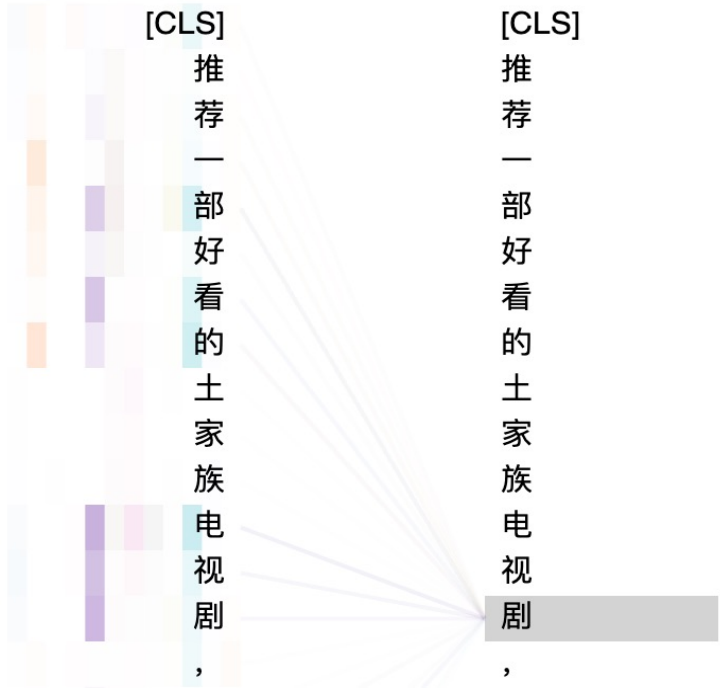
Visualize BERT using the bertviz package

```
[16] sentence_a = "推荐一部好看的土家族电视剧，"  Recommend a good Tujia TV series
     sentence_b = "这太好看了。" This is so good!

     inputs = tokenizer.encode_plus(sentence_a, sentence_b, return_tensors='pt', add_specia
     input_ids = inputs['input_ids']
     token_type_ids = inputs['token_type_ids']
     attention = model(input_ids, token_type_ids=token_type_ids)[-1]
     sentence_b_start = token_type_ids[0].tolist().index(1)
     input_id_list = input_ids[0].tolist() # Batch index 0
     tokens = tokenizer.convert_ids_to_tokens(input_id_list)
```




```
head_view(attention, tokens, sentence_b_start)
```

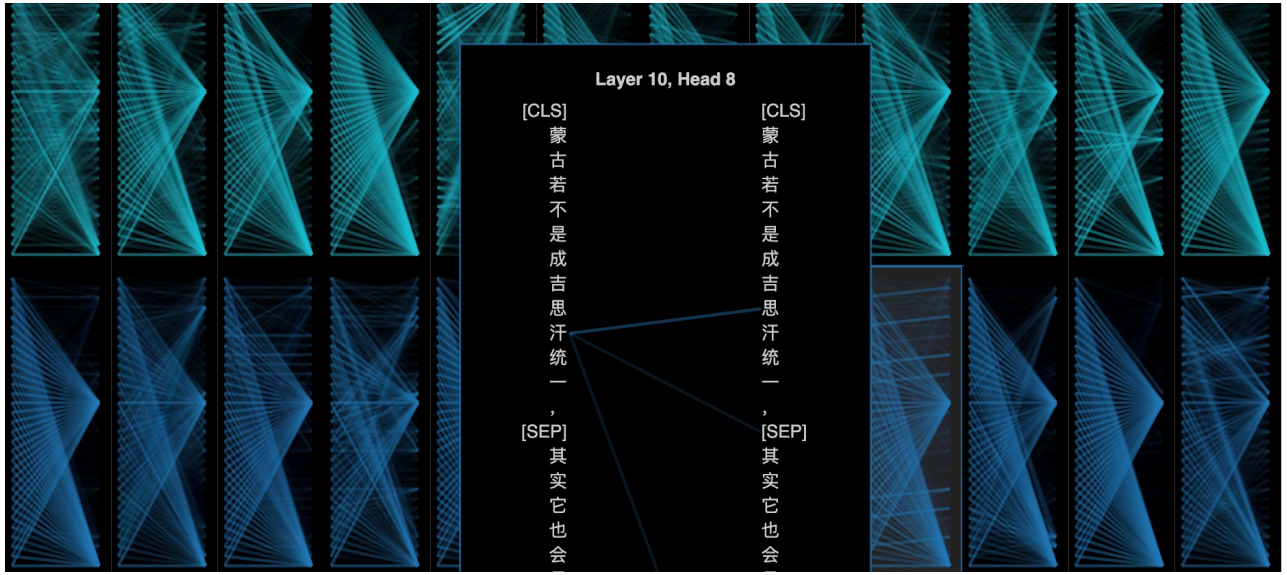
Layer: 9 Attention: All



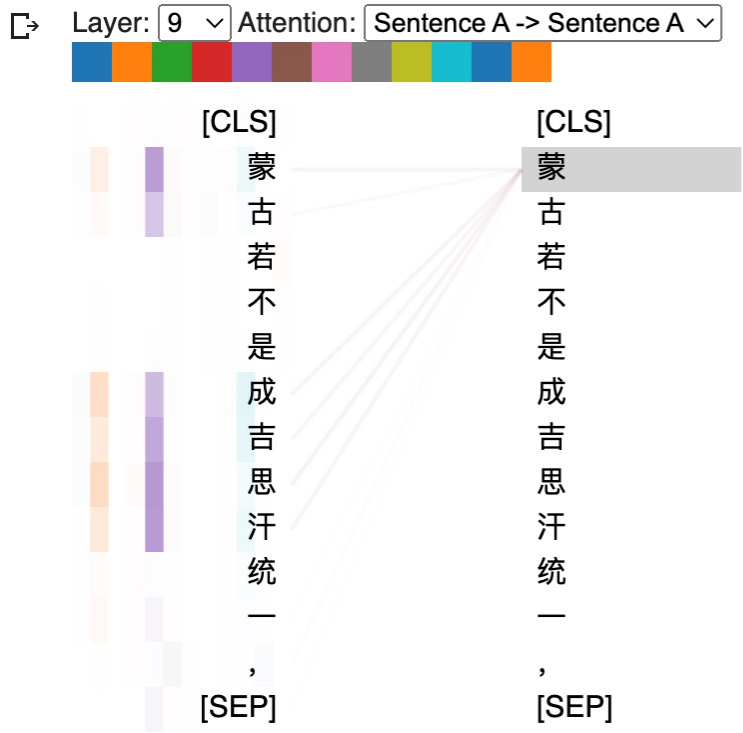
BERT can learn the “context”.

Visualize BERT using the bertviz package

```
sentence_a = "蒙古若不是成吉思汗统一，" If Mongolia was not unified by Genghis Khan,
sentence_b = "其实它也会是一个散居分布的民族。" it will also be a diaspora.

inputs = tokenizer.encode_plus(sentence_a, sentence_b, return_tensors='pt', add_spe
input_ids = inputs['input_ids']
token_type_ids = inputs['token_type_ids']
attention = model(input_ids, token_type_ids=token_type_ids)[-1]
sentence_b_start = token_type_ids[0].tolist().index(1)
input_id_list = input_ids[0].tolist() # Batch index 0
tokens = tokenizer.convert_ids_to_tokens(input_id_list)
```



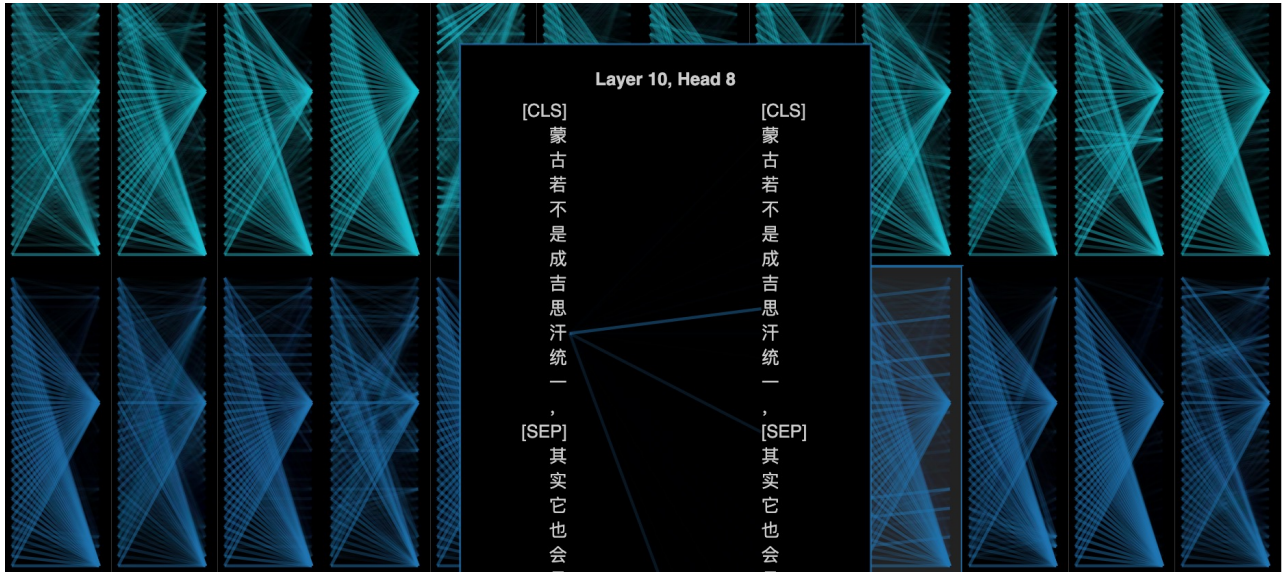

```
head_view(attention, tokens, sentence_b_start)
```



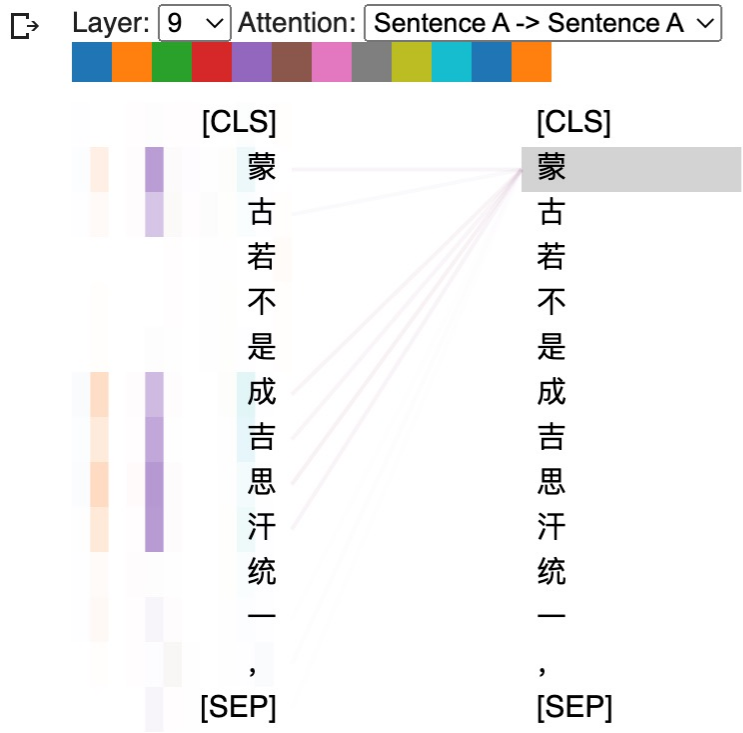
BERT can learn the “context”.

Visualize BERT using the bertviz package

```
sentence_a = "蒙古若不是成吉思汗统一，" If Mongolia was not unified by Genghis Khan,
sentence_b = "其实它也会是一个散居分布的民族。" it will also be a diaspora.

inputs = tokenizer.encode_plus(sentence_a, sentence_b, return_tensors='pt', add_spe
input_ids = inputs['input_ids']
token_type_ids = inputs['token_type_ids']
attention = model(input_ids, token_type_ids=token_type_ids)[-1]
sentence_b_start = token_type_ids[0].tolist().index(1)
input_id_list = input_ids[0].tolist() # Batch index 0
tokens = tokenizer.convert_ids_to_tokens(input_id_list)
```




```
head_view(attention, tokens, sentence_b_start)
```



BERT can learn the “context”.

Language Modeling with pre-trained 'pipelines'

```
pprint(nlp(f"少数民族政策{nlp.tokenizer.mask_token}实施."))
```

```
[{'score': 0.3078611493110657,
  'sequence': '少数民族政策的实施.',
  'token': 44574,
  'token_str': '的'},
 {'score': 0.10741521418094635,
  'sequence': '少数民族政策大实施.',
  'token': 48262,
  'token_str': '大'},
 {'score': 0.06714530289173126,
  'sequence': '少数民族政策者实施.',
  'token': 49337,
  'token_str': '者'},
 {'score': 0.06547900289297104,
  'sequence': '少数民族政策中实施.',
  'token': 47643,
  'token_str': '中'},
 {'score': 0.038725387305021286,
  'sequence': '少数民族政策士实施.',
  'token': 49527,
  'token_str': '士'}]
```

Language Modeling with 'pipelines'

```
[35] from transformers import pipeline
     clf = pipeline('sentiment-analysis')
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english (https://huggingface.co/distilbert-base-u
Downloading: 100% 629/629 [00:00<00:00, 5.36kB/s]
Downloading: 100% 255M/255M [00:13<00:00, 39.2MB/s]
Downloading: 100% 48.0/48.0 [00:00<00:00, 960B/s]
Downloading: 100% 226k/226k [00:00<00:00, 1.20MB/s]
```

```
clf('突然释然了，我的民族 突然看到一句话，说蒙古若不是成吉思汗统一，其实蒙古族也会是一个散居分布的民族，只不过藏族统不统一都是一个超级团结的民族')
```

```
[{'label': 'NEGATIVE', 'score': 0.9441806674003601}]
```

Sentiment Analysis: binary-class classification

Language Modeling with 'pipelines'

```
[37] from transformers import pipeline
     generator = pipeline("text-generation", model="uer/gpt2-chinese-poem")
```

```
Downloading: 100% 577/577 [00:00<00:00, 12.0kB/s]
Downloading: 100% 406M/406M [00:11<00:00, 40.7MB/s]
Downloading: 100% 271/271 [00:00<00:00, 4.17kB/s]
Downloading: 100% 113k/113k [00:00<00:00, 1.40MB/s]
Downloading: 100% 112/112 [00:00<00:00, 2.06kB/s]
```

```
results = generator(
    "[CLS] 禅 边 风 味 客 边 愁 , ",
    max_length=40,
    num_return_sequences=2,
)
print(results)
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
[{'generated_text': '[CLS] 禅 边 风 味 客 边 愁 , 好 梦 依 稀 到 旧 游 。 知 否 乡 关 重 回 首 , 茫 茫 衰 柳 白 云 秋 。 名 浪 得 号 虚 空'},
```

Text generation: Chinese ancient poem

Language Modeling with 'pipelines'

```
from transformers import pipeline, AutoModelWithLMHead, AutoTokenizer
model = AutoModelWithLMHead.from_pretrained("Helsinki-NLP/opus-mt-en-zh")
tokenizer = AutoTokenizer.from_pretrained("Helsinki-NLP/opus-mt-en-zh")
translation = pipeline("translation_en_to_zh", model=model, tokenizer=tokenizer)
```

```
Downloading: 100% 1.37k/1.37k [00:00<00:00, 33.4kB/s]
Downloading: 100% 298M/298M [00:07<00:00, 45.8MB/s]
Downloading: 100% 44.0/44.0 [00:00<00:00, 990B/s]
Downloading: 100% 788k/788k [00:00<00:00, 1.07MB/s]
Downloading: 100% 786k/786k [00:00<00:00, 1.04MB/s]
Downloading: 100% 1.54M/1.54M [00:00<00:00, 1.15MB/s]
```

```
[40] text = "I like to study Data Science and Machine Learning"
translated_text = translation(text, max_length=40)[0]['translation_text']
print(translated_text)
```

```
我喜欢学习数据科学和机器学习
```

```
text = "Rapidly developing economies such as China and India, as well as other industrial countries in Europe and Asia, continue to encourage
translated_text = translation(text, max_length=40)[0]['translation_text']
print(translated_text)
```

```
中国和印度等快速发展中经济体以及欧洲和亚洲其他工业国家继续鼓励和推进工程教学。
```

Translation: From English to Chinese

Language Modeling with 'pipelines'

```python
from transformers import AutoModelWithLMHead, AutoTokenizer, top_k_top_p_filtering
import torch
from torch.nn import functional as F
tokenizer = AutoTokenizer.from_pretrained("uer/gpt2-chinese-couplet")
model = AutoModelWithLMHead.from_pretrained("uer/gpt2-chinese-couplet")
sequence = f"中国的少数民族主要有壮族，苗族和回族"
input_ids = tokenizer.encode(sequence, return_tensors="pt")
# get logits of last hidden state
next_token_logits = model(input_ids).logits[:, -1, :]
# filter
filtered_next_token_logits = top_k_top_p_filtering(next_token_logits, top_k=50, top_p=1.0)
# sample
probs = F.softmax(filtered_next_token_logits, dim=-1)
next_token = torch.multinomial(probs, num_samples=1)
generated = torch.cat([input_ids, next_token], dim=-1)
resulting_string = tokenizer.decode(generated.tolist()[0])
```

Causal language modeling

```
Downloading: 100% 414/414 [00:00<00:00, 4.44kB/s]
Downloading: 100% 576/576 [00:00<00:00, 4.05kB/s]
Downloading: 100% 107k/107k [00:00<00:00, 1.24MB/s]
Downloading: 100% 112/112 [00:00<00:00, 692B/s]
Downloading: 100% 401M/401M [00:12<00:00, 44.6MB/s]
```

```python
print(resulting_string)
```

```
[CLS] 中 国 的 少 数 民 族 主 要 有 壮 族 ， 苗 族 和 回 族 [SEP] 志
```

Prior methods?

- Context: context-independent vs. context-dependent
- Word order: whether taking into account the word position
- Embeddings: input as a single word or a sentence?