jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
bölümlerinde 25,200
##n           1,009
mezun         6,424
,                16
Veri         15,414
Bilimi       28,883
ve            1,992
makine        9,769
öğrenme      10,938
##si          2,462
alanında      5,114
min           4,883
.                18
2                22
yıl           2,180
```

In [19]:

```python
segments_ids = [1] * len(tokenized_text)

print (segments_ids)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

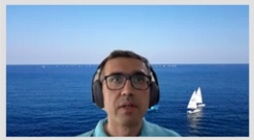
In [20]:

```python
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

Out[20]:

```
BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(32000, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (token_type_embeddings): Embedding(2, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
```

Recording



jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

File Edit View Insert Cell Kernel Widgets Help   Trusted   Python 3

```
bolumlerinde 25,260
##n           1,009
mezun         6,424
,                16
Veri         15,414
Bilimi       28,883
ve            1,992
makine        9,769
öğrenme      10,938
##si          2,462
alanında      5,114
min           4,883
.                18
2                22
yıl           2,180
```

In [19]:

```
segments_ids = [1] * len(tokenized_text)

print (segments_ids)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

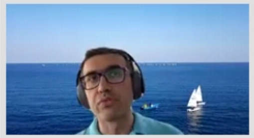
In [20]:

```
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

Out[20]:

```
BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(32000, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (token_type_embeddings): Embedding(2, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
```

Recording



```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [20]:

```python
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

Out[20]:

```
BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(32000, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (token_type_embeddings): Embedding(2, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
```

In [22]:

```python
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

In [23]:

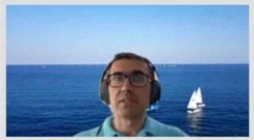
```python
encoded_layers.size()
```

Out[23]:

```
torch.Size([1, 211, 768])
```

In [24]:

```python
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# NEDEN BÖYLE? msa

#print ("Number of layers:", len(encoded_layers))
```



%69 15:23

Recording

localhost

using different decoding metho... | ipynb/ | TRANSFORMER ENCODING V5 - Jupyter Notebook

pyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Edit View Insert Cell Kernel Widgets Help

Run Code

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

n [20]:

```
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

```
    (position_embeddings): Embedding(512, 768)
    (token_type_embeddings): Embedding(2, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
```

n [22]:

```
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

n [23]:

```
encoded_layers.size()
```

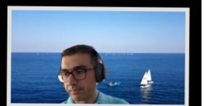
ut[23]:

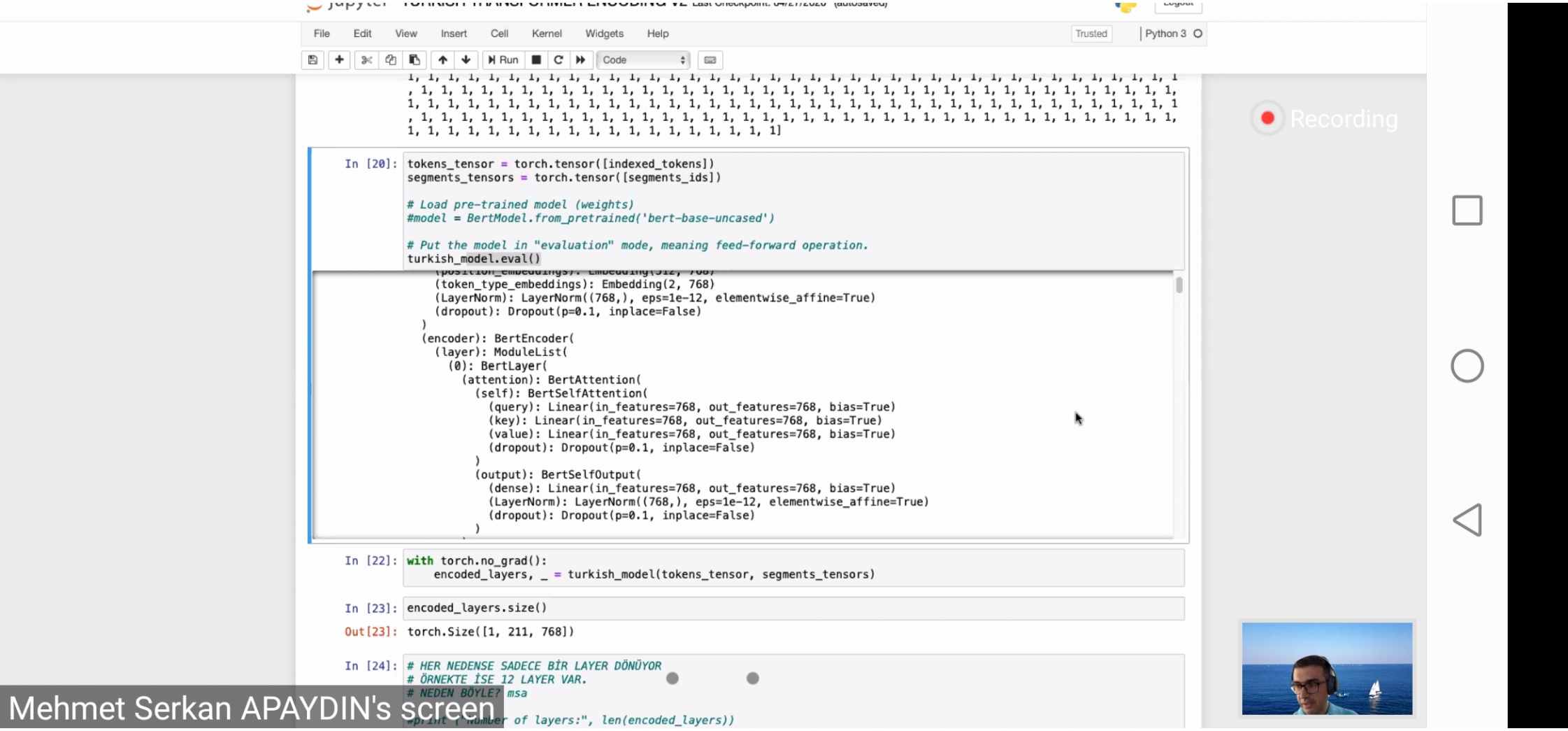
```
torch.Size([1, 211, 768])
```

n [24]:

```
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# NEDEN BÖYLE? msa

#print ("Number of layers:", len(encoded_layers))
#layer_i = 0

print ("Number of batches:", len(encoded_layers))
batch_i = 0
```

Mehmet Serkan APAYDIN's screen




File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Recording
In [20]:
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])
# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')
# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
(token_type_embeddings): Embedding(2, 768)
(LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
(dropout): Dropout(p=0.1, inplace=False)
(encoder): BertEncoder(
(layer): ModuleList(
(0): BertLayer(
(attention): BertAttention(
(self): BertSelfAttention(
(query): Linear(in_features=768, out_features=768, bias=True)
(key): Linear(in_features=768, out_features=768, bias=True)
(value): Linear(in_features=768, out_features=768, bias=True)
(dropout): Dropout(p=0.1, inplace=False)
(output): BertSelfOutput(
(dense): Linear(in_features=768, out_features=768, bias=True)
(LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
(dropout): Dropout(p=0.1, inplace=False)
In [22]:
with torch.no_grad():
encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
In [23]:
encoded_layers.size()
Out[23]:
torch.Size([1, 211, 768])
In [24]:
HER NEDENSE SADECE BİR LAYER DÖNÜYOR
ÖRNEKTE İSE 12 LAYER VAR.
NEDEN BÖYLE? msa
Mehmet Serkan APAYDIN's screen

jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [20]:

```
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

```
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
```

In [22]:

```
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

In [23]:

```
encoded_layers.size()
```

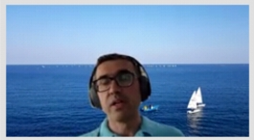
Out[23]:

```
torch.Size([1, 211, 768])
```

In [24]:

```
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# ...? msa
```

Recording



Mehmet Serkan APAYDIN's screen

jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [20]:

```
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

```
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
        )
```

In [22]:

```
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

In [23]:

```
encoded_layers.size()
```

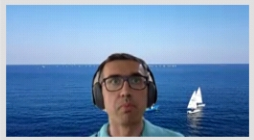
Out[23]:

```
torch.Size([1, 211, 768])
```

In [24]:

```
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# NEDEN BÖYLE? msa
```

Recording



jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

Recording

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [20]:
```
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

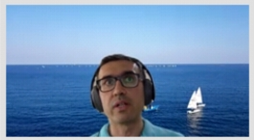
```
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (1): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
```

In [22]:
```
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

In [23]:
```
encoded_layers.size()
```

Out[23]:
```
torch.Size([1, 211, 768])
```

In [24]:
```
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# NEDEN BÖYLE? msa
```



jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [20]:

```
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

```
          )
          (1): BertLayer(
            (attention): BertAttention(
              (self): BertSelfAttention(
                (query): Linear(in_features=768, out_features=768, bias=True)
                (key): Linear(in_features=768, out_features=768, bias=True)
                (value): Linear(in_features=768, out_features=768, bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
              (output): BertSelfOutput(
                (dense): Linear(in_features=768, out_features=768, bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
            )
            (intermediate): BertIntermediate(
              (dense): Linear(in_features=768, out_features=3072, bias=True)
            )
            (output): BertOutput(
              (dense): Linear(in_features=3072, out_features=768, bias=True)
```

In [22]:

```
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

In [23]:

```
encoded_layers.size()
```

Out[23]:

```
torch.Size([1, 211, 768])
```

In [24]:

```
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# NEDEN BÖYLE? msa
```

Recording

jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

Recording

In [20]:
```
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

```
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
  )
  (pooler): BertPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
)
```

In [22]:
```
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

In [23]:
```
encoded_layers.size()
```

Out[23]:
```
torch.Size([1, 211, 768])
```

In [24]:
```
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# NEDEN BÖYLE? msa

#print ("Number of layers:", len(encoded_layers))
#layer_i = 0

print ("Number of batches:", len(encoded_layers))
batch_i = 0
```

jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

Recording

```
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

In [20]:

```
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])

# Load pre-trained model (weights)
#model = BertModel.from_pretrained('bert-base-uncased')

# Put the model in "evaluation" mode, meaning feed-forward operation.
turkish_model.eval()
```

```
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (11): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
```

In [22]:

```
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

In [23]:

```
encoded_layers.size()
```

Out[23]:

```
torch.Size([1, 211, 768])
```

In [24]:

```
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# NEDEN BÖYLE? msa

...ber of layers:", len(encoded_layers))
#layer_i = 0
```

Mehmet Serkan APAYDIN's screen

jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

```
            (output): BertOutput(
              (dense): Linear(in_features=3072, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
        )
      )
      (pooler): BertPooler(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (activation): Tanh()
      )
    )
```

Recording

In [22]:
```
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

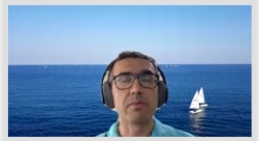
In [23]:
```
encoded_layers.size()
```

Out[23]:
```
torch.Size([1, 211, 768])
```

In [24]:
```
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# NEDEN BÖYLE? msa

#print ("Number of layers:", len(encoded_layers))
#layer_i = 0

print ("Number of batches:", len(encoded_layers))
batch_i = 0

print ("Number of tokens:", len(encoded_layers[batch_i]))
token_i = 0

print ("Number of hidden units:", len(encoded_layers[batch_i][token_i]))
```

```
Number of batches: 1
Number of tokens: 211
Number of hidden units: 768
```

In [29]:
```
token_i = 5
#layer_i = 5
vec = encoded_layers[batch_i][token_i]

# Plot the values as a histogram to show their distribution.
```



jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
```

In [22]:
```
with torch.no_grad():
    encoded_layers, _ = turkish_model(tokens_tensor, segments_tensors)
```

In [23]:
```
encoded_layers.size()
```

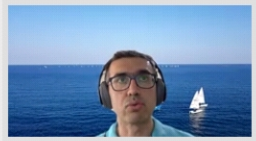
Out[23]:
```
torch.Size([1, 211, 768])
```

In [24]:
```
# HER NEDENSE SADECE BİR LAYER DÖNÜYOR
# ÖRNEKTE İSE 12 LAYER VAR.
# NEDEN BÖYLE? msa

#print ("Number of layers:", len(encoded_layers))
#layer_i = 0

print ("Number of batches:", len(encoded_layers))
batch_i = 0

print ("Number of tokens:", len(encoded_layers[batch_i]))
token_i = 0

print ("Number of hidden units:", len(encoded_layers[batch_i][token_i]))
```

```
Number of batches: 1
Number of tokens: 211
Number of hidden units: 768
```

In [29]:
```
token_i = 5
#layer_i = 5
vec = encoded_layers[batch_i][token_i]

# Plot the values as a histogram to show their distribution.
plt.figure(figsize=(10,10))
plt.hist(vec, bins=200)
plt.show()
```

Recording



jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
     Type of encoded_layers:  <class 'torch.Tensor'>
Tensor shape for each layer:  torch.Size([211, 768])
```

In [31]:
```
# `encoded_layers` has shape [1 x 211 x 768]

# `token_vecs` is a tensor with shape [211 x 768]
token_vecs = encoded_layers[0]

# Calculate the average of all 22 token vectors.
sentence_embedding = torch.mean(token_vecs, dim=0)
```

In [32]:
```
print ("Our final sentence embedding vector of shape:", sentence_embedding.size())
```

```
Our final sentence embedding vector of shape: torch.Size([768])
```

In [ ]:
```
--- VOILA! WE NOW HAVE A SENTENCE EMBEDDING FOR THE ENTIRE SENTENCE ---
--- THIS CAN BE USED TO ENCODE CV OR JOB ADVERTISEMENTS ---
--- AND COMPARE WITH TF-IDF OR COUNT-VECTORIZER ---
```

In [20]:
```
tokens_2 = turkish_tokenizer.tokenize("Koşucular koşmaya çıktıklarında koşu parkurunun koşuya uygun olmadığını görüp
print("Tokens: {}".format(tokens_2))
```

```
Tokens: ['Koş', '##ucu', '##lar', 'koş', '##maya', 'çıktık', '##larında', 'koşu', 'parkur', '##unun', 'koşu', '##ya
', 'uygun', 'olmadığını', 'görüp', 'koş', '##maktan', 'vazgeç', '##tiler']
```

In [26]:
```
tokens_3 = turkish_tokenizer.tokenize("Bu bir örnek metindir")
print("Tokens: {}".format(tokens_3))

# This is not sufficient for the model, as it requires integers as input,
# not a problem, let's convert tokens to ids.
tokens_ids_3 = turkish_tokenizer.convert_tokens_to_ids(tokens_3)
print("Tokens id: {}".format(tokens_ids_3))

# Add the required special tokens
tokens_ids_3 = turkish_tokenizer.build_inputs_with_special_tokens(tokens_ids_3) #??
print("After adding the special tokens, Tokens id: {}".format(tokens_ids_3))


# We need to convert to a Deep Learning framework specific format, let's use PyTorch for now.
tokens_pt_3 = torch.tensor([tokens_ids_3])
print("Tokens PyTorch: {}".format(tokens_pt_3))

# Now we're ready to go through BERT with out input
```

Recording

jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

Recording

In [20]:
```
tokens_2 = turkish_tokenizer.tokenize("Koşucular koşmaya çıktıklarında koşu parkurunun koşuya uygun olmadığını görüp
print("Tokens: {}".format(tokens_2))
```

```
Tokens: ['Koş', '##ucu', '##lar', 'koş', '##maya', 'çıktık', '##larında', 'koşu', 'parkur', '##unun', 'koşu', '##ya
', 'uygun', 'olmadığını', 'görüp', 'koş', '##maktan', 'vazgeç', '##tiler']
```

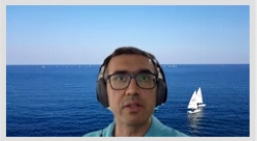
In [26]:
```
tokens_3 = turkish_tokenizer.tokenize("Bu bir örnek metindir")
print("Tokens: {}".format(tokens_3))

# This is not sufficient for the model, as it requires integers as input,
# not a problem, let's convert tokens to ids.
tokens_ids_3 = turkish_tokenizer.convert_tokens_to_ids(tokens_3)
print("Tokens id: {}".format(tokens_ids_3))

# Add the required special tokens
tokens_ids_3 = turkish_tokenizer.build_inputs_with_special_tokens(tokens_ids_3) #??
print("After adding the special tokens, Tokens id: {}".format(tokens_ids_3))


# We need to convert to a Deep Learning framework specific format, let's use PyTorch for now.
tokens_pt_3 = torch.tensor([tokens_ids_3])
print("Tokens PyTorch: {}".format(tokens_pt_3))

# Now we're ready to go through BERT with out input
outputs_3, pooled_3 = turkish_model(tokens_pt_3)
print("Token wise output: {}, Pooled output: {}".format(outputs.shape, pooled.shape))
```

```
Tokens: ['Bu', 'bir', 'örnek', 'metin', '##dir']
Tokens id: [2123, 1996, 4073, 8111, 2067]
After adding the special tokens, Tokens id: [2, 2123, 1996, 4073, 8111, 2067, 3]
Tokens PyTorch: tensor([[   2, 2123, 1996, 4073, 8111, 2067,    3]])
Token wise output: torch.Size([1, 7, 768]), Pooled output: torch.Size([1, 768])
```

In [27]:
```
input_ids_4 = torch.tensor([turkish_tokenizer.encode("Bu metin üzerindeki tüm dikkat vektörleri ve hidden state'leri
```

In [28]:
```
print(input_ids_4)
```

```
tensor([[    2,  2123,  8111,  5931,  2525,  2903, 31599,  2065,  1992, 10276,
          2025,  7489,  1025,    11, 17746, 27416,     3]])
```



jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
Tokens: ['Bu', 'bir', 'örnek', 'metin', '##dir']
Tokens id: [2123, 1996, 4073, 8111, 2067]
After adding the special tokens, Tokens id: [2, 2123, 1996, 4073, 8111, 2067, 3]
Tokens PyTorch: tensor([[   2, 2123, 1996, 4073, 8111, 2067,    3]])
Token wise output: torch.Size([1, 7, 768]), Pooled output: torch.Size([1, 768])
```

In [27]:
```
input_ids_4 = torch.tensor([turkish_tokenizer.encode("Bu metin üzerindeki tüm dikkat vektörleri ve hidden state'leri
```

In [28]:
```
print(input_ids_4)
```

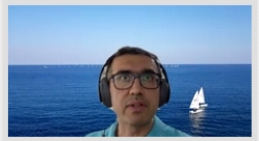
```
tensor([[    2,  2123,  8111,  5931,  2525,  2903, 31599,  2065,  1992, 10276,
          2025,  7489,  1025,    11, 17746, 27416,     3]])
```

In [29]:
```
# tokens = tokenizer.tokenize("This is an input example")
# tokens_ids = tokenizer.convert_tokens_to_ids(tokens)
# tokens_pt = torch.tensor([tokens_ids])

# This code can be factored into one-line as follow
tokens_pt2 = turkish_tokenizer.encode_plus("Bu bir örnek metindir", return_tensors="pt")

for key, value in tokens_pt2.items():
    print("{}:\n\t{}".format(key, value))

outputs2, pooled2 = turkish_model(**tokens_pt2)
print("Difference with previous code: ({}, {})".format((outputs2 - outputs_3).sum(), (pooled2 - pooled_3).sum()))
```

```
input_ids:
        tensor([[   2, 2123, 1996, 4073, 8111, 2067,    3]])
token_type_ids:
        tensor([[0, 0, 0, 0, 0, 0, 0]])
attention_mask:
        tensor([[1, 1, 1, 1, 1, 1, 1]])
Difference with previous code: (0.0, 0.0)
```

In [30]:
```
# Single segment input
single_seg_input = turkish_tokenizer.encode_plus("Bu örnek bir cümledir")

# Multiple segment input
multi_seg_input = turkish_tokenizer.encode_plus("Bu A cümlesidir", "Bu B cümlesidir")

print("Single segment token (str): {}".format(turkish_tokenizer.convert_ids_to_tokens(single_seg_input['input_ids'])
print("Single segment token (int): {}".format(single_seg_input['input_ids']))
```

Recording



jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
Single segment token (str): ['[CLS]', 'Bu', 'örnek', 'bir', 'cümle', '##dir', '[SEP]']
Single segment token (int): [2, 2123, 4073, 1996, 9446, 2067, 3]
Single segment type       : [0, 0, 0, 0, 0, 0, 0]

Multi segment token (str): ['[CLS]', 'Bu', 'A', 'cümlesi', '##dir', '[SEP]', 'Bu', 'B', 'cümlesi', '##dir', '[SEP]'
]
Multi segment token (int): [2, 2123, 37, 30773, 2067, 3, 2123, 38, 30773, 2067, 3]
Multi segment type       : [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
```

In [31]:

```
# Padding highlight
tokens = turkish_tokenizer.batch_encode_plus(
    ["Bu örnek bir cümledir", "Bu ikinci örnek bir cümledir"],
    pad_to_max_length=True  # First sentence will have some PADDED tokens to match second sequence length
)

for i in range(2):
    print("Tokens (int)      : {}".format(tokens['input_ids'][i]))
    print("Tokens (str)      : {}".format([turkish_tokenizer.convert_ids_to_tokens(s) for s in tokens['input_ids'][i
    print("Tokens (attn_mask): {}".format(tokens['attention_mask'][i]))
    print()
```

```
Tokens (int)      : [2, 2123, 4073, 1996, 9446, 2067, 3, 0]
Tokens (str)      : ['[CLS]', 'Bu', 'örnek', 'bir', 'cümle', '##dir', '[SEP]', '[PAD]']
Tokens (attn_mask): [1, 1, 1, 1, 1, 1, 1, 0]

Tokens (int)      : [2, 2123, 3557, 4073, 1996, 9446, 2067, 3]
Tokens (str)      : ['[CLS]', 'Bu', 'ikinci', 'örnek', 'bir', 'cümle', '##dir', '[SEP]']
Tokens (attn_mask): [1, 1, 1, 1, 1, 1, 1, 1]
```

In [40]:

```
#de_bert = turkish_model.from_pretrained("dbmdz/bert-base-german-cased")
#de_tokenizer = turkish_tokenizer.from_pretrained("dbmdz/bert-base-german-cased")
de_tokenizer = AutoTokenizer.from_pretrained("dbmdz/bert-base-german-cased",output_hidden_states=True,output_attenti



de_model = AutoModel.from_pretrained("dbmdz/bert-base-german-cased")
```

In [16]: #Let's load German BERT from the Bavarian State Library

Recording

jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run | Code

```
    print("{}:\n\t{}".format(key, value))

outputs2, pooled2 = turkish_model(**tokens_pt2)
print("Difference with previous code: ({}, {})".format((outputs2 - outputs_3).sum(), (pooled2 - pooled_3).sum()))
```

```
input_ids:
        tensor([[   2, 2123, 1996, 4073, 8111, 2067,    3]])
token_type_ids:
        tensor([[0, 0, 0, 0, 0, 0, 0]])
attention_mask:
        tensor([[1, 1, 1, 1, 1, 1, 1]])
Difference with previous code: (0.0, 0.0)
```

In [30]:

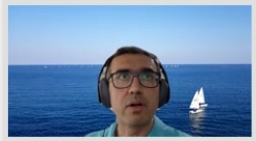
```
# Single segment input
single_seg_input = turkish_tokenizer.encode_plus("Bu örnek bir cümledir")

# Multiple segment input
multi_seg_input = turkish_tokenizer.encode_plus("Bu A cümlesidir", "Bu B cümlesidir")

print("Single segment token (str): {}".format(turkish_tokenizer.convert_ids_to_tokens(single_seg_input['input_ids'])))
print("Single segment token (int): {}".format(single_seg_input['input_ids']))
print("Single segment type       : {}".format(single_seg_input['token_type_ids']))

# Segments are concatened in the input to the model, with
print()
print("Multi segment token (str): {}".format(turkish_tokenizer.convert_ids_to_tokens(multi_seg_input['input_ids'])))
print("Multi segment token (int): {}".format(multi_seg_input['input_ids']))
print("Multi segment type       : {}".format(multi_seg_input['token_type_ids']))
```

```
Single segment token (str): ['[CLS]', 'Bu', 'örnek', 'bir', 'cümle', '##dir', '[SEP]']
Single segment token (int): [2, 2123, 4073, 1996, 9446, 2067, 3]
Single segment type       : [0, 0, 0, 0, 0, 0, 0]

Multi segment token (str): ['[CLS]', 'Bu', 'A', 'cümlesi', '##dir', '[SEP]', 'Bu', 'B', 'cümlesi', '##dir', '[SEP]'
]
Multi segment token (int): [2, 2123, 37, 30773, 2067, 3, 2123, 38, 30773, 2067, 3]
Multi segment type       : [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
```

In [31]:

```
# Padding highlight
tokens = turkish_tokenizer.batch_encode_plus(
    ["Bu örnek bir cümledir", "Bu ikinci örnek bir cümledir"],
    pad_to_max_length=True  # First sentence will have some PADDED tokens to match second sequence length
)

for i in range(2):
    print("Tokens (int)      : {}".format(tokens['input_ids'][i]))
```

Recording



jupyter TURKISH TRANSFORMER ENCODING V2 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run

Zoom

Leave

Recording

In [31]:
```
# `encoded_layers` has shape [1 x 211 x 768]

# `token_vecs` is a tensor with shape [211 x 768]
token_vecs = encoded_layers[0]

# Calculate the average of all 22 token vectors.
sentence_embedding = torch.mean(token_vecs, dim=0)
```

In [32]:
```
print ("Our final sentence embedding vector of shape:", sentence_embedding.size())
```

```
Our final sentence embedding vector of shape: torch.Size([768])
```

In [ ]:
```
--- VOILA! WE NOW HAVE A SENTENCE EMBEDDING FOR THE ENTIRE SENTENCE ---
--- THIS CAN BE USED TO ENCODE CV OR JOB ADVERTISEMENTS ---
--- AND COMPARE WITH TF-IDF OR COUNT-VECTORIZER ---
```

In [20]:
```
tokens_2 = turkish_tokenizer.tokenize("Koşucular koşmaya çıktıklarında koşu parkurunun koşuya uygun olmadığını görüp
print("Tokens: {}".format(tokens_2))
```

```
Tokens: ['Koş', '##ucu', '##lar', 'koş', '##maya', 'çıktık', '##larında', 'koşu', 'parkur', '##unun', 'koşu', '##ya
', 'uygun', 'olmadığını', 'görüp', 'koş', '##maktan', 'vazgeç', '##tiler']
```

In [26]:
```
tokens_3 = turkish_tokenizer.tokenize("Bu bir örnek metindir")
print("Tokens: {}".format(tokens_3))

# This is not sufficient for the model, as it requires integers as input,
# not a problem, let's convert tokens to ids.
tokens_ids_3 = turkish_tokenizer.convert_tokens_to_ids(tokens_3)
print("Tokens id: {}".format(tokens_ids_3))

# Add the required special tokens
tokens_ids_3 = turkish_tokenizer.build_inputs_with_special_tokens(tokens_ids_3) #??
print("After adding the special tokens, Tokens id: {}".format(tokens_ids_3))


# We need to convert to a Deep Learning framework specific format, let's use PyTorch for now.
tokens_pt_3 = torch.tensor([tokens_ids_3])
print("Tokens PyTorch: {}".format(tokens_pt_3))

# Now we're ready to go through BERT with out input
outputs_3, pooled_3 = turkish_model(tokens_pt_3)
print("Token wise output: {}, Pooled output: {}".format(outputs.shape, pooled.shape))
```

Unmute Start Video Share Participants More 1

jupyter turkish transformer encoding Last Checkpoint: 04/20/2020 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

Recording

In [27]:
```
input_ids_4 = torch.tensor([turkish_tokenizer.encode("Bu metin üzerindeki tüm dikkat vektörleri ve hidden state'leri
```

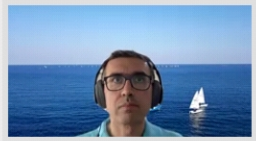
In [28]:
```
print(input_ids_4)
```

```
tensor([[    2,  2123,  8111,  5931,  2525,  2903, 31599,  2065,  1992, 10276,
          2025,  7489,  1025,    11, 17746, 27416,     3]])
```

In [29]:
```
# tokens = tokenizer.tokenize("This is an input example")
# tokens_ids = tokenizer.convert_tokens_to_ids(tokens)
# tokens_pt = torch.tensor([tokens_ids])

# This code can be factored into one-line as follow
tokens_pt2 = turkish_tokenizer.encode_plus("Bu bir örnek metindir", return_tensors="pt")

for key, value in tokens_pt2.items():
    print("{}:\n\t{}".format(key, value))

outputs2, pooled2 = turkish_model(**tokens_pt2)
print("Difference with previous code: ({}, {})".format((outputs2 - outputs_3).sum(), (pooled2 - pooled_3).sum()))
```

```
input_ids:
        tensor([[   2, 2123, 1996, 4073, 8111, 2067,    3]])
token_type_ids:
        tensor([[0, 0, 0, 0, 0, 0, 0]])
attention_mask:
        tensor([[1, 1, 1, 1, 1, 1, 1]])
Difference with previous code: (0.0, 0.0)
```

In [30]:
```
# Single segment input
single_seg_input = turkish_tokenizer.encode_plus("Bu örnek bir cümledir")

# Multiple segment input
multi_seg_input = turkish_tokenizer.encode_plus("Bu A cümlesidir", "Bu B cümlesidir")

print("Single segment token (str): {}".format(turkish_tokenizer.convert_ids_to_tokens(single_seg_input['input_ids'])
print("Single segment token (int): {}".format(single_seg_input['input_ids']))
print("Single segment type       : {}".format(single_seg_input['token_type_ids']))

# Segments are concatened in the input to the model, with
print()
print("Multi segment token (str): {}".format(turkish_tokenizer.convert_ids_to_tokens(multi_seg_input['input_ids'])))
print("Multi segment token (int): {}".format(multi_seg_input['input_ids']))
print("Multi segment type       : {}".format(multi_seg_input['token_type_ids']))
```



jupyter TURKISH TRANSFORMER ENCODING V5 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
Our final sentence embedding vector of shape: torch.Size([768])
```

In [ ]:
```
--- VOILA! WE NOW HAVE A SENTENCE EMBEDDING FOR THE ENTIRE SENTENCE ---
--- THIS CAN BE USED TO ENCODE CV OR JOB ADVERTISEMENTS ---
--- AND COMPARE WITH TF-IDF OR COUNT-VECTORIZER ---
```

In [33]:
```
for i, token_str in enumerate(tokenized_text):
  print (i, token_str)
```

```
0 [CLS]
1 Üniversiteler
2 ##in
3 ilgili
4 lisans
5 bölümlerinde
6 ##n
7 mezun
8 ,
9 Veri
10 Bilimi
11 ve
12 makine
13 öğrenme
14 ##si
15 alanında
16 min
17 .
18 2
```

In [41]:
```
word_order = [i for i in range(len(tokenized_text)) if tokenized_text[i] == 'öğrenim']

l = [x**2 for x in range(4)]
for i in range(len(word_order)):
    print (word_order[i],tokenized_text[word_order[i]])
```

```
72 öğrenim
76 öğrenim
80 öğrenim
84 öğrenim
```

In [47]:
```
kelime_indeks = [i for i in range(len(tokenized_text)) if tokenized_text[i] == 'hikaye']
for i in range(len(kelime_indeks)):
    print (kelime_indeks[i],tokenized_text[kelime_indeks[i]])
```

Recording

jupyter TURKISH TRANSFORMER ENCODING V5 Last Checkpoint: 04/27/2020 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Run Code

```
8 ,
9 Veri
10 Bilimi
11 ve
12 makine
13 öğrenme
14 ##si
15 alanında
16 min
17 .
18 2
```

Recording

In [41]:
```
word_order = [i for i in range(len(tokenized_text)) if tokenized_text[i] == 'öğrenim']

l = [x**2 for x in range(4)]
for i in range(len(word_order)):
    print (word_order[i],tokenized_text[word_order[i]])
```

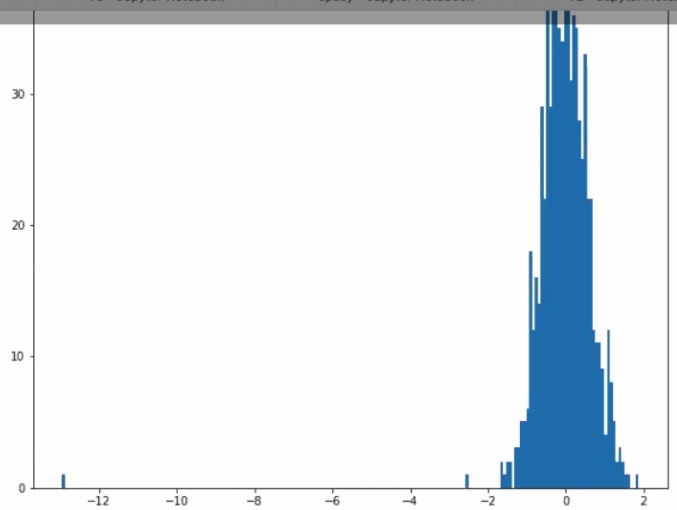
```
72 öğrenim
76 öğrenim
80 öğrenim
84 öğrenim
```

In [47]:
```
kelime_indeks = [i for i in range(len(tokenized_text)) if tokenized_text[i] == 'hikaye']
for i in range(len(kelime_indeks)):
    print (kelime_indeks[i],tokenized_text[kelime_indeks[i]])
```

```
111 hikaye
```

In [42]:
```
print('First 4 vector values for each instance of "öğrenim".')
print("denetim##li öğrenim", str(token_vecs[72][:5]))
print("denetim##siz öğrenim", str(token_vecs[76][:5]))
print("güçlendir##me öğrenim##i", str(token_vecs[80][:5]))
print("derin öğrenim", str(token_vecs[84][:5]))
```

```
First 4 vector values for each instance of "öğrenim".
denetim##li öğrenim tensor([ 0.2599, -0.1962, -0.7959,  0.5174, -0.6913])
denetim##siz öğrenim tensor([ 0.1796,  0.19[illegible], -0.[illegible]895,  0.2528, -0.7096])
güçlendir##me öğrenim##i tensor([-0.0602, -0.3918, -0.7981,  0.0927,  0.7299])
derin öğrenim tensor([ 0.1547, -1.2101, -1.2573,  0.3666, -0.5956])
```



Grouping the values by layer makes sense for the model, but for our purposes we want it grouped by token.

Current dimensions:

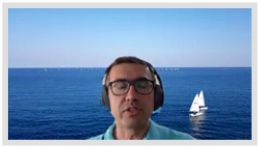
```
[# layers, # batches, # tokens, # features]
```

Desired dimensions:

```
[# tokens, # layers, # features]
```

Luckily, PyTorch includes the `permute` function for easily rearranging the dimensions of a tensor.

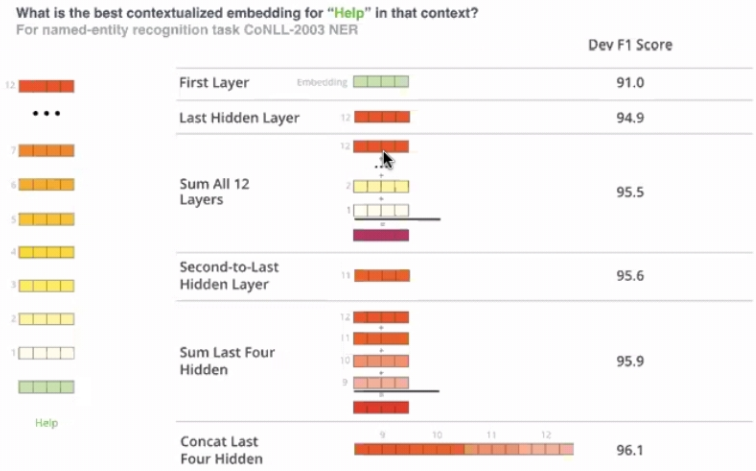
However, the first dimension is currently a Python list!



Now, what do we do with these hidden states? We would like to get individual vectors for each of our tokens, or perhaps a single vector representation of the whole sentence, but for each token of our input we have 12 separate vectors each of length 768.

In order to get the individual vectors we will need to combine some of the layer vectors...but which layer or combination of layers provides the best representation? The BERT authors tested this by feeding different vector combinations as input features to a BiLSTM used on a named entity recognition task and observing the resulting F1 scores.

(Image from Jay Allamar's blog)



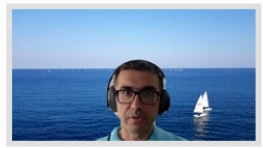
While concatenation of the last four layers produced the best results on this specific task, many of the other methods come in a close second and in general it

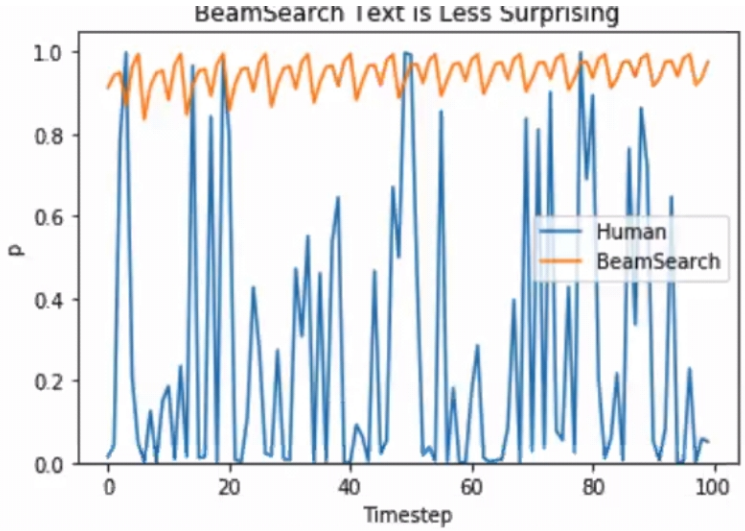
As an alternative method, let's try creating the word vectors by **summing** together the last four layers.

```
# Stores the token vectors, with shape [22 x 768]
token_vecs_sum = []

# `token_embeddings` is a [22 x 12 x 768] tensor.

# For each token in the sentence...
for token in token_embeddings:

    # `token` is a [12 x 768] tensor

    # Sum the vectors from the last four layers.
    sum_vec = torch.sum(token[-4:], dim=0)

    # Use `sum_vec` to represent `token`.
    token_vecs_sum.append(sum_vec)

print ('Shape is: %d x %d' % (len(token_vecs_sum), len(token_vecs_s
```

```
Shape is: 22 x 768
```

Sentence Vectors





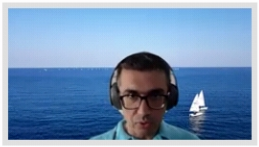
So let's stop being boring and introduce some randomness 🤪.

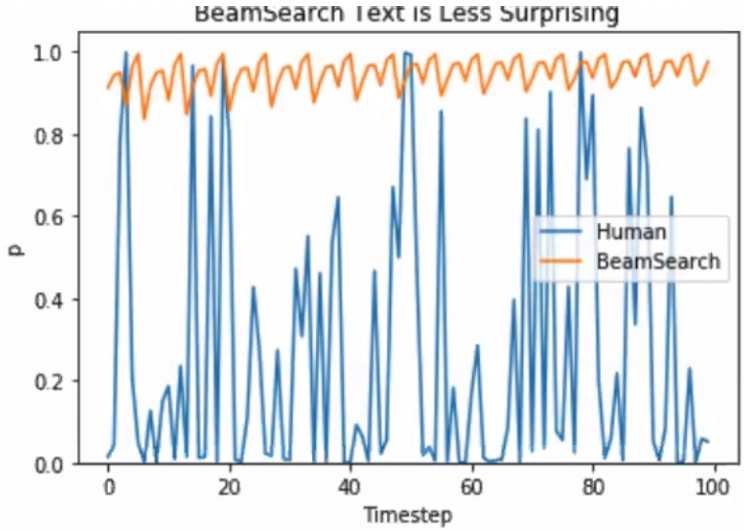
Sampling

In its most basic form, sampling means randomly picking the next word w_t according to its conditional probability distribution:

$$w_t \sim P(w|w_{1:t-1})$$

Taking the example from above, the following graphic visualizes language generation when sampling.





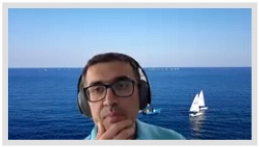
So let's stop being boring and introduce some randomness 🤪.

Sampling

In its most basic form, sampling means randomly picking the next word w_t according to its conditional probability distribution:

$$w_t \sim P(w|w_{1:t-1})$$

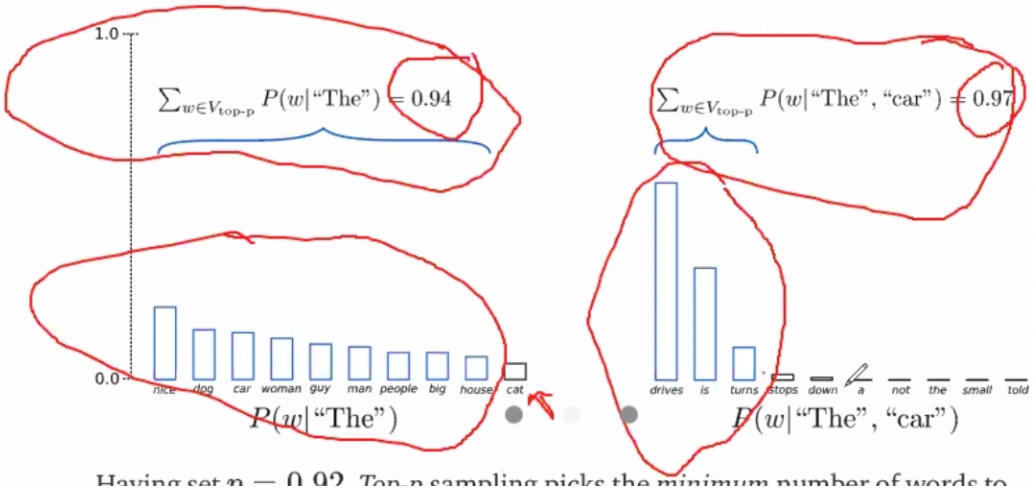
Taking the example from above, the following graphic visualizes language generation when sampling.



pool to a fixed size K could endanger the model to produce gibberish for sharp distributions and limit the model's creativity for flat distribution. This intuition led Ari Holtzman et al. (2019) to create ***Top-p***- or ***nucleus***-sampling.

Top-p (nucleus) sampling

Instead of sampling only from the most likely K words, in *Top-p* sampling chooses from the smallest possible set of words whose cumulative probability exceeds the probability p. The probability mass is then redistributed among this set of words. This way, the size of the set of words (*a.k.a* the number of words in the set) can dynamically increase and decrease according to the next word's probability distribution. Ok, that was very wordy, let's visualize.



Having set $p = 0.92$, Top-p sampling picks the *minimum* number of words to

%60 16:00

Zoom

Leave

Recording

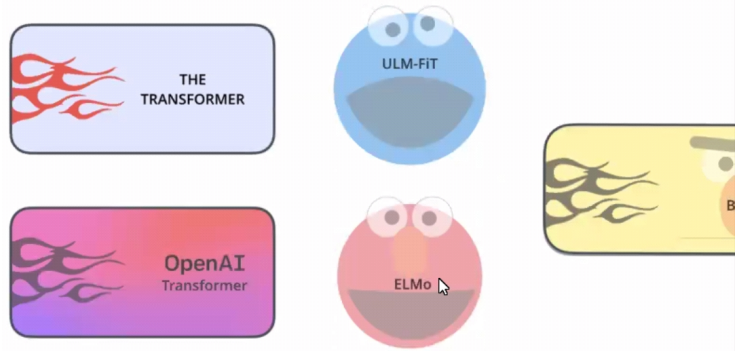
.github.io/illustrated-bert/

Arama

Şehir University Workstream Classes eecs 202 Natural Language P... uzaktan eğitim

nslations: Chinese (Simplified), Japanese, Korean, Persian, Russian

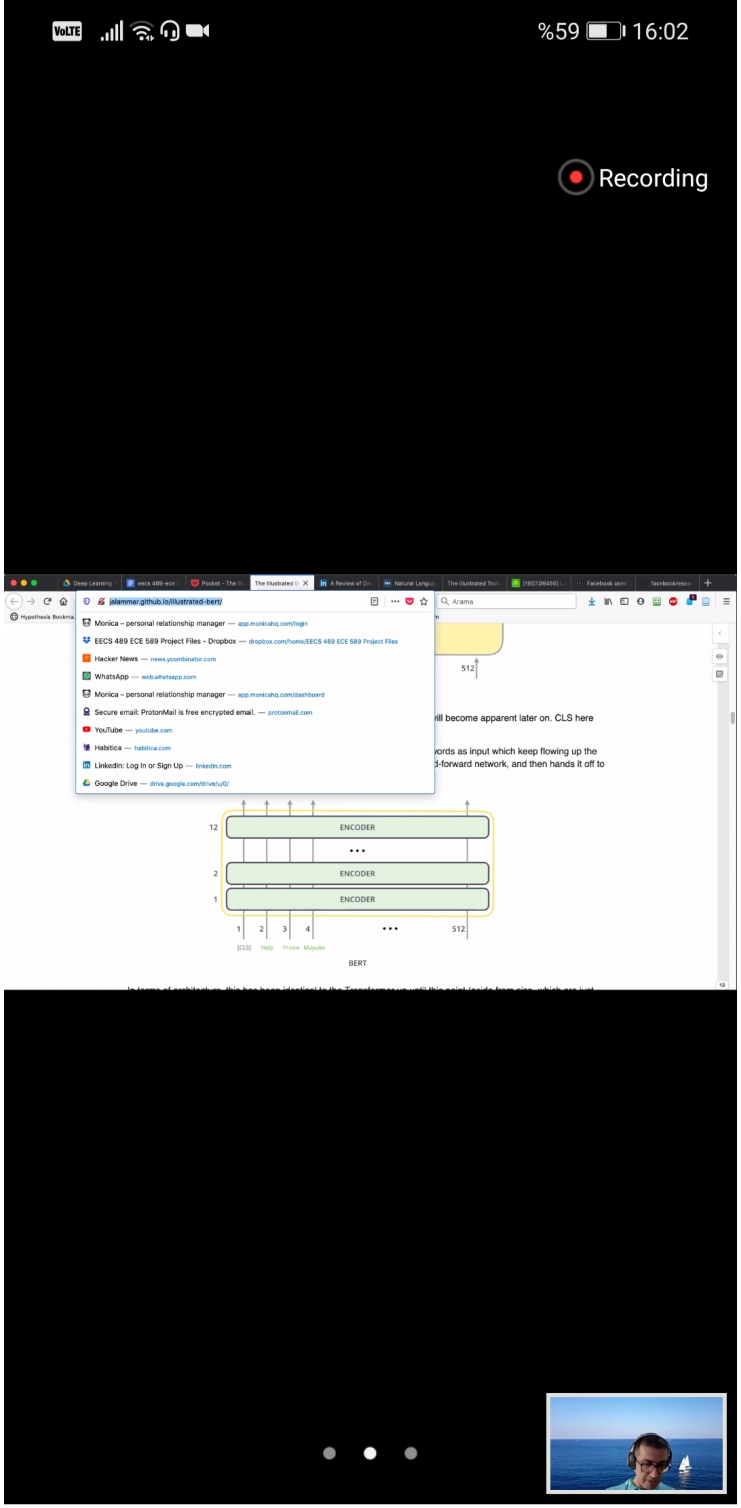
e year 2018 has been an inflection point for machine learning models handling text (or more accura
nguage Processing or NLP for short). Our conceptual understanding of how best to represent words
a way that best captures underlying meanings and relationships is rapidly evolving. Moreover, the N
s been putting forward incredibly powerful components that you can freely download and use in you
d pipelines (It's been referred to as NLP's ImageNet moment, referencing how years ago similar dev
celerated the development of machine learning in Computer Vision tasks).

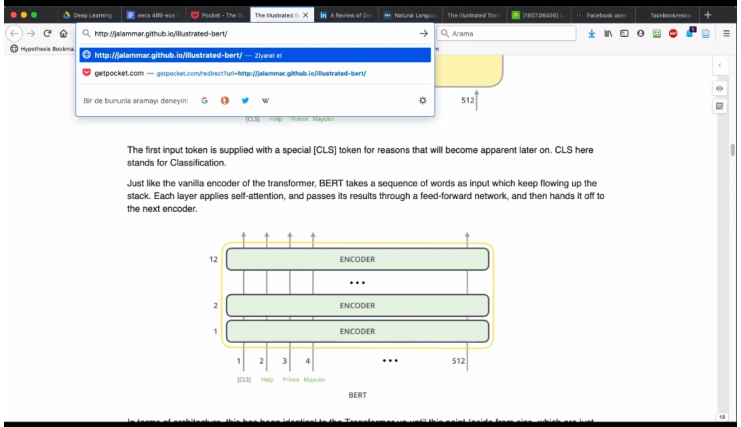


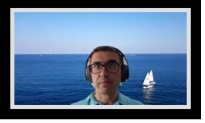
.M-FiT has nothing to do with Cookie Monster. But I couldn't think of anything else..)

e of the latest milestones in this development is the release of BERT, an event described as markin

Unmute Start Video Share Participants More 6


%59 16:02
Recording
jalammar.github.io/illustrated-bert/
Monica – personal relationship manager — app.monicahq.com/login
EECS 489 ECE 589 Project Files - Dropbox — dropbox.com/home/EECS 489 ECE 589 Project Files
Hacker News — news.ycombinator.com
WhatsApp — web.whatsapp.com
Monica – personal relationship manager — app.monicahq.com/dashboard
Secure email: ProtonMail is free encrypted email. — protonmail.com
YouTube — youtube.com
Habitica — habitica.com
LinkedIn: Log In or Sign Up — linkedin.com
Google Drive — drive.google.com/drive/u/0/
512
ENCODER
ENCODER
ENCODER
12
2
1
1
2
3
4
512
[CLS]
Help
Prince
Mayuko
BERT



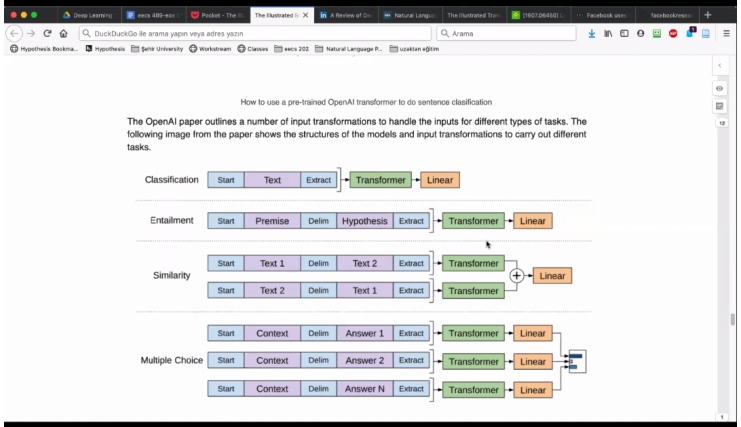


%58 16:09

Zoom

Leave

Recording



Unmute

Start Video

Share

Participants

10

More

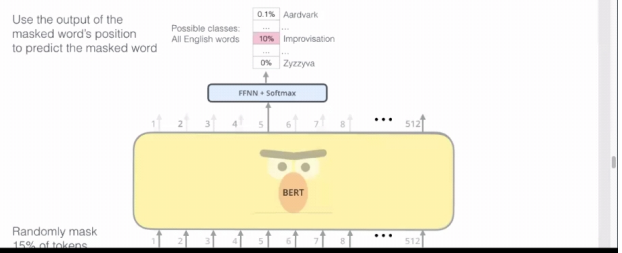
"Hold my beer", said R-rated BERT.

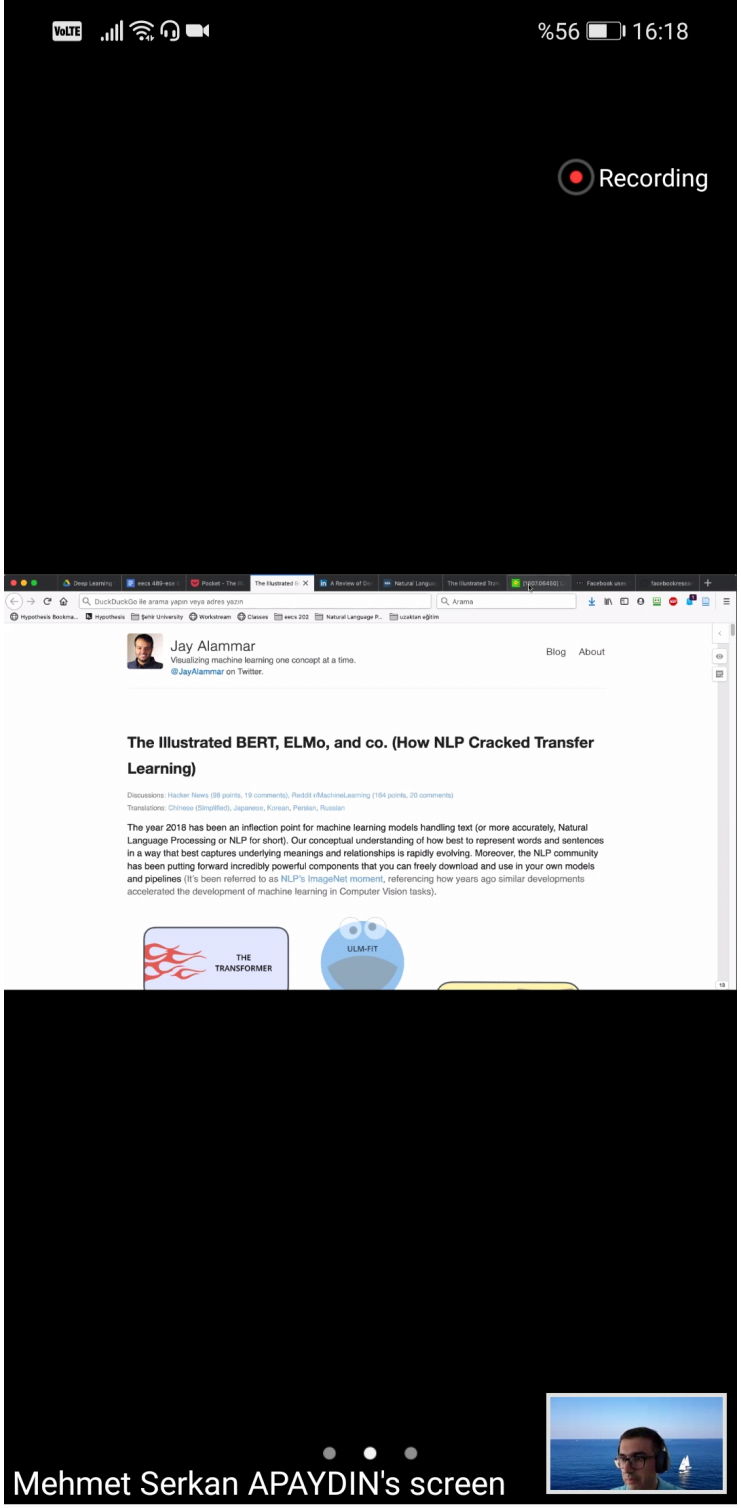
Masked Language Model

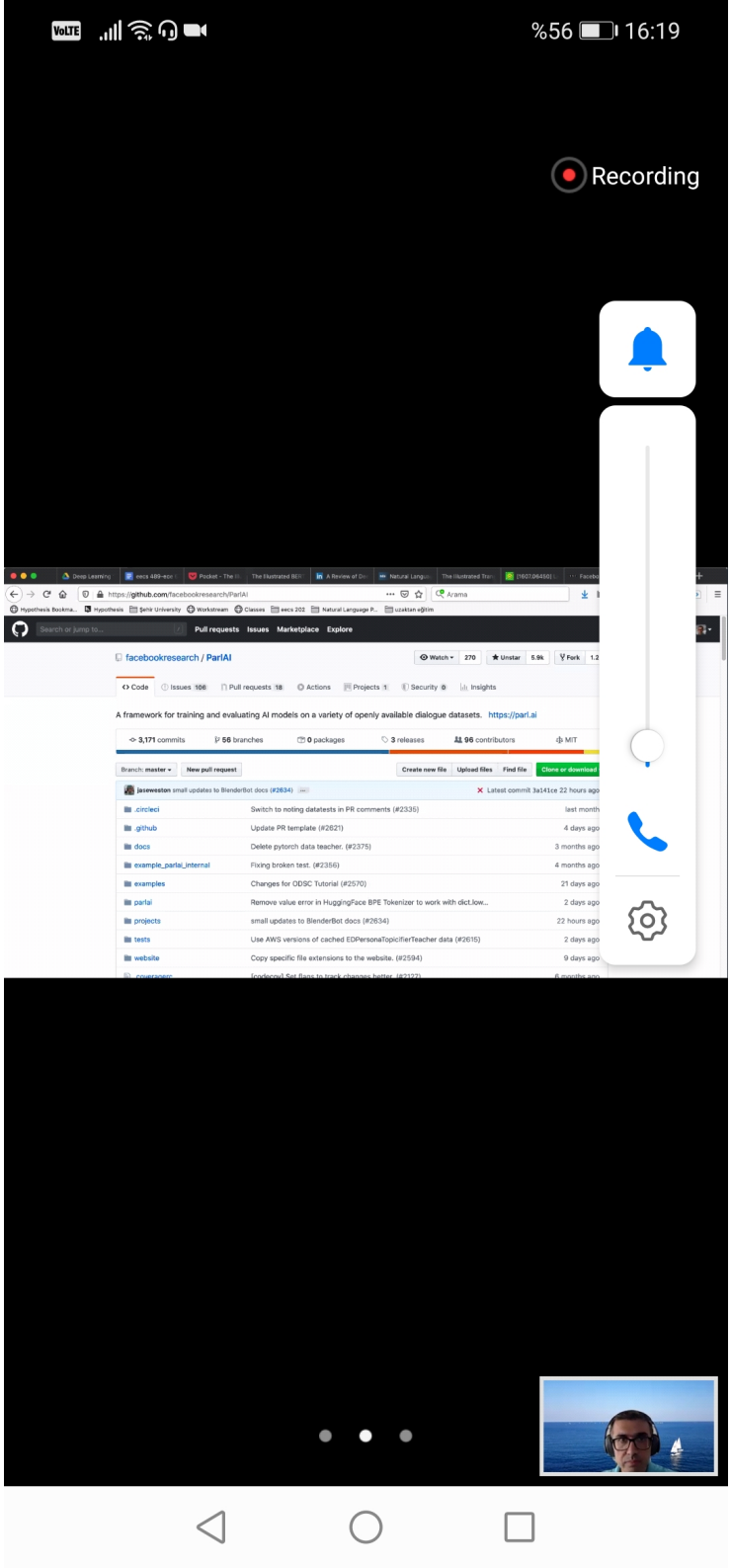
"We'll use transformer encoders", said BERT.

"This is madness", replied Ernie, "Everybody knows bidirectional conditioning would allow each word to indirectly see itself in a multi-layered context."

"We'll use masks", said BERT confidently.

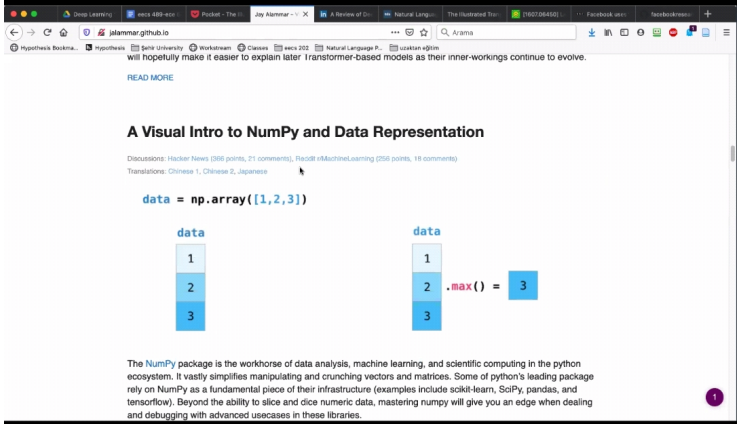



%56 16:18
Recording
Jay Alammar
Visualizing machine learning one concept at a time.
@JayAlammar on Twitter.
Blog About
The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)
The year 2018 has been an inflection point for machine learning models handling text (or more accurately, Natural Language Processing or NLP for short). Our conceptual understanding of how best to represent words and sentences in a way that best captures underlying meanings and relationships is rapidly evolving. Moreover, the NLP community has been putting forward incredibly powerful components that you can freely download and use in your own models and pipelines (It's been referred to as NLP's ImageNet moment, referencing how years ago similar developments accelerated the development of machine learning in Computer Vision tasks).
THE TRANSFORMER
ULM-FiT
Mehmet Serkan APAYDIN's screen


%56 16:19
Recording
facebookresearch / ParlAI
A framework for training and evaluating AI models on a variety of openly available dialogue datasets. https://parl.ai

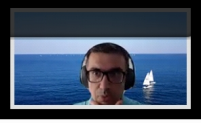
%56 16:19

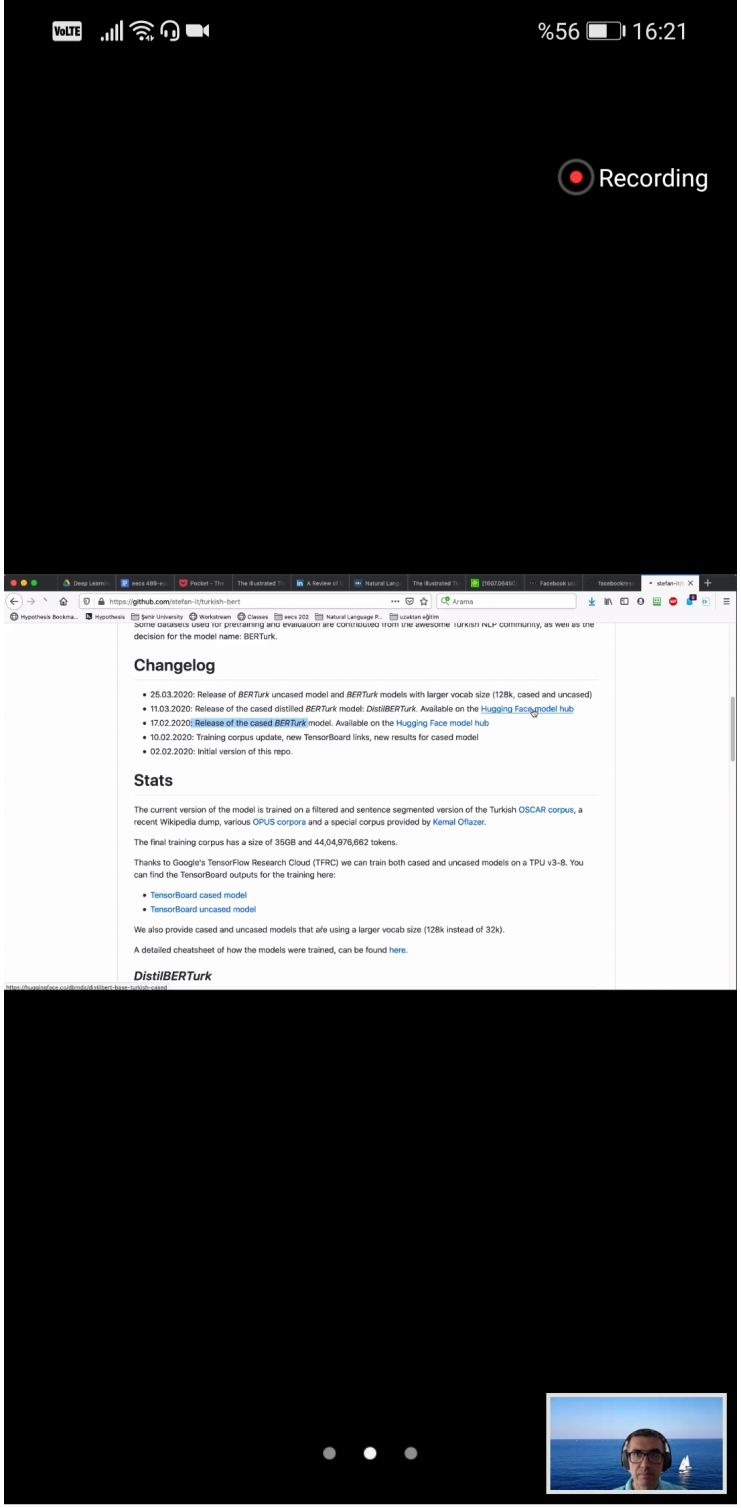
Recording

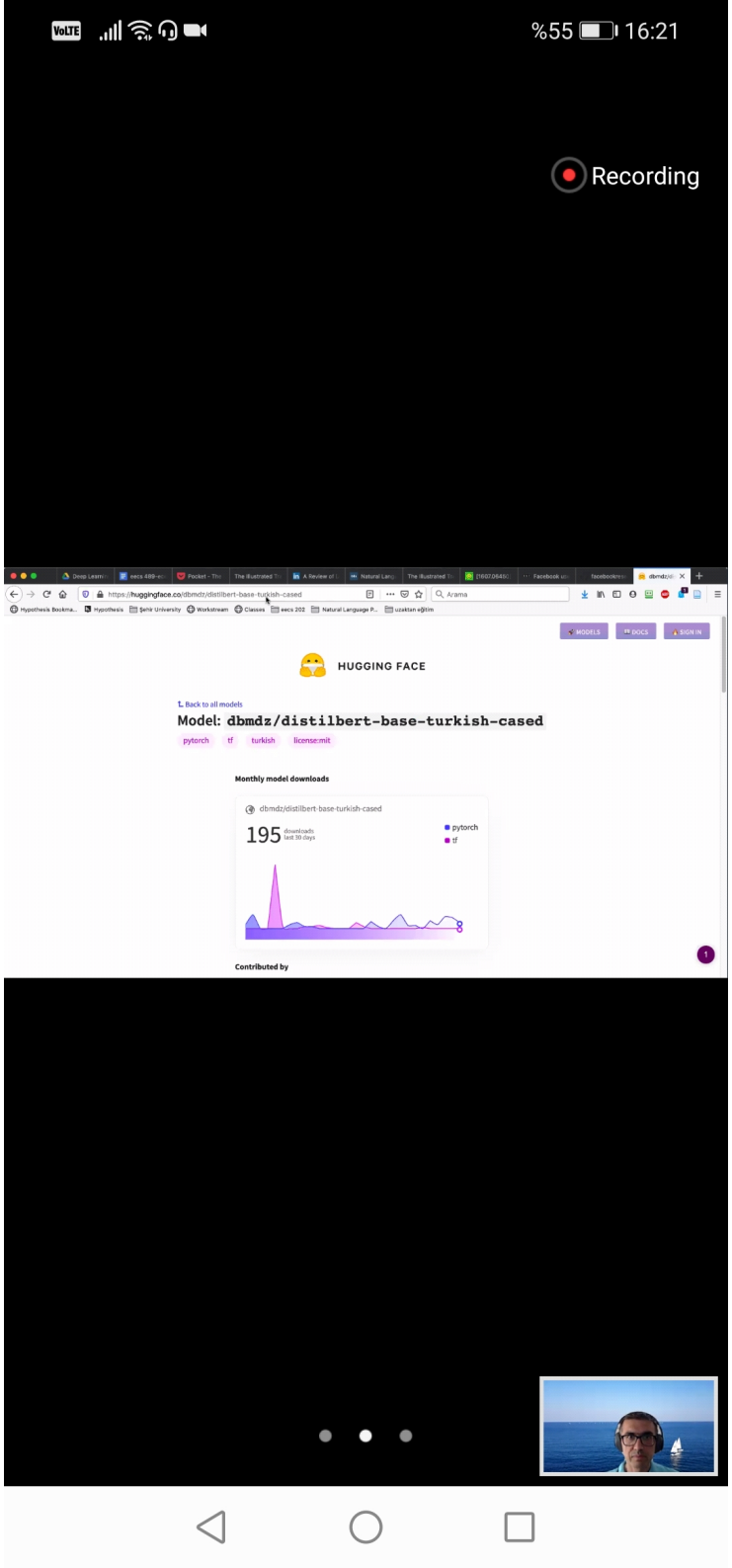


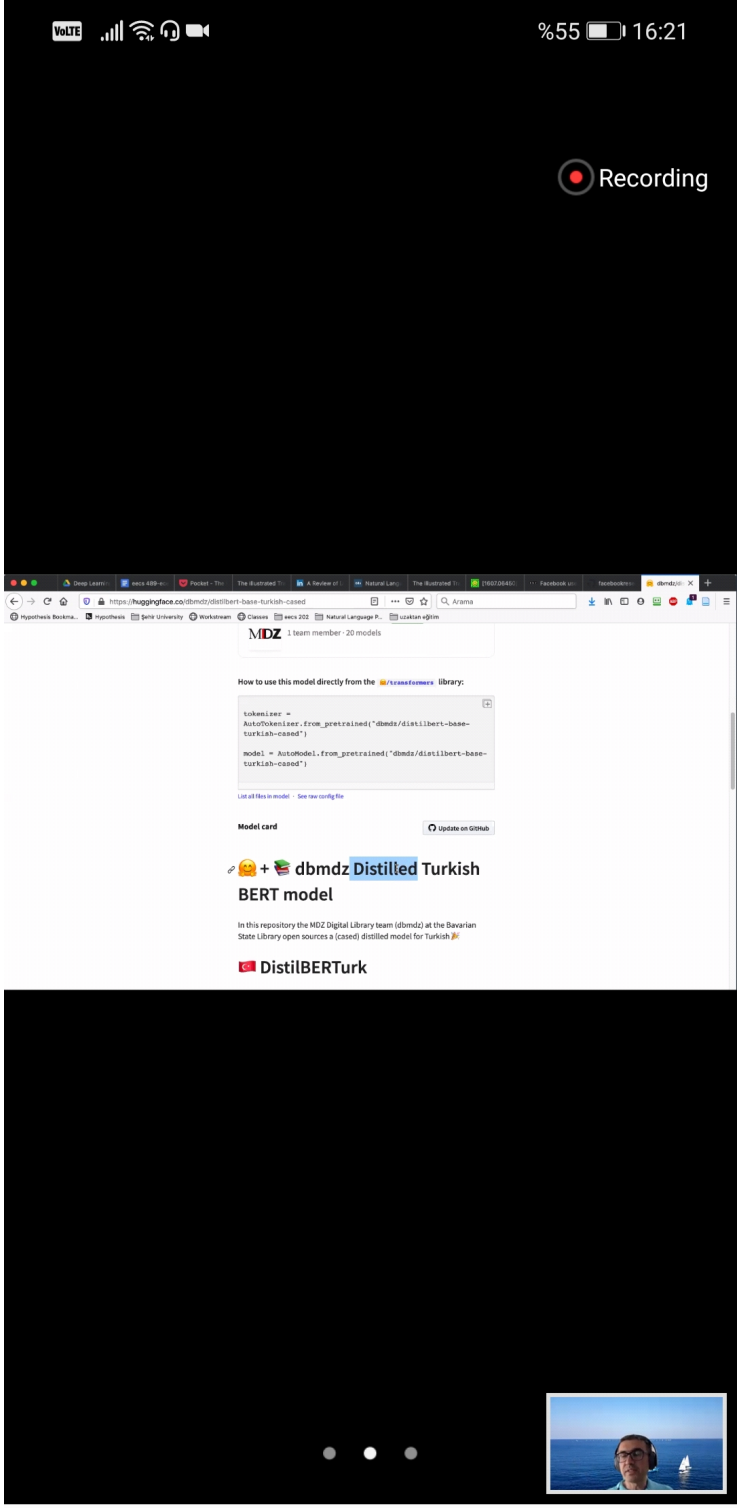
From Muhlis Sariyer to All

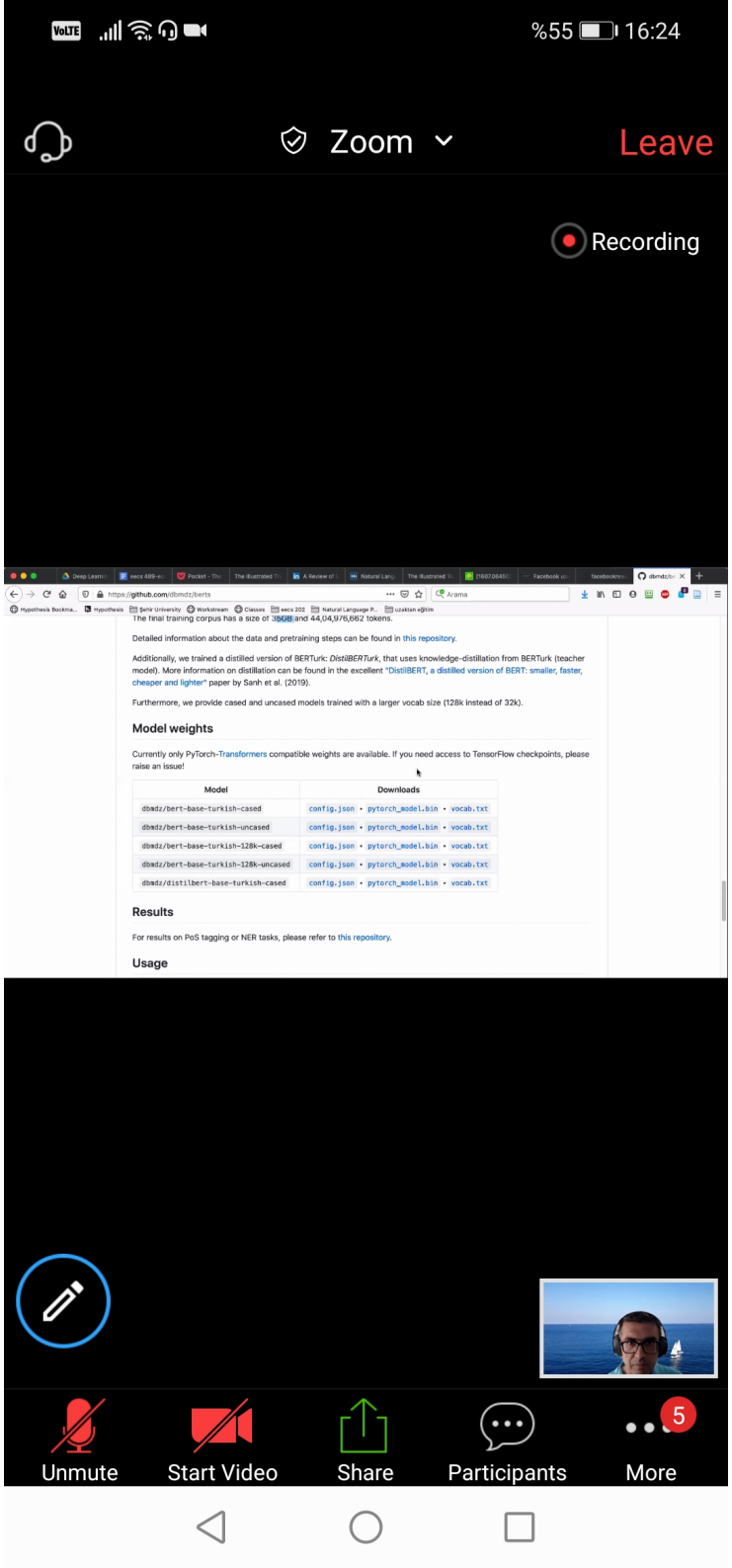
will you be sharing the codes you used today on lms?




%56 16:21
Recording
https://github.com/stefan-it/turkish-bert
Changelog
25.03.2020: Release of BERTurk uncased model and BERTurk models with larger vocab size (128k, cased and uncased)
11.03.2020: Release of the cased distilled BERTurk model: DistilBERTurk. Available on the Hugging Face model hub
17.02.2020: Release of the cased BERTurk model. Available on the Hugging Face model hub
10.02.2020: Training corpus update, new TensorBoard links, new results for cased model
02.02.2020: Initial version of this repo.
Stats
The current version of the model is trained on a filtered and sentence segmented version of the Turkish OSCAR corpus, a recent Wikipedia dump, various OPUS corpora and a special corpus provided by Kemal Oflazer.
The final training corpus has a size of 35GB and 44,04,976,662 tokens.
Thanks to Google's TensorFlow Research Cloud (TFRC) we can train both cased and uncased models on a TPU v3-8. You can find the TensorBoard outputs for the training here:
TensorBoard cased model
TensorBoard uncased model
We also provide cased and uncased models that are using a larger vocab size (128k instead of 32k).
A detailed cheatsheet of how the models were trained, can be found here.
DistilBERTurk


%55 16:21
Recording
https://huggingface.co/dbmdz/distilbert-base-turkish-cased
HUGGING FACE
Back to all models
Model: dbmdz/distilbert-base-turkish-cased
pytorch
tf
turkish
license:mit
Monthly model downloads
dbmdz/distilbert-base-turkish-cased
195
downloads
last 30 days
pytorch
tf
Contributed by


%55 16:21
Recording
https://huggingface.co/dbmdz/distilbert-base-turkish-cased
MDZ 1 team member · 20 models
How to use this model directly from the transformers library:
tokenizer = AutoTokenizer.from_pretrained("dbmdz/distilbert-base-turkish-cased")
model = AutoModel.from_pretrained("dbmdz/distilbert-base-turkish-cased")
List all files in model · See raw config file
Model card
Update on GitHub
🤗 + 📚 dbmdz Distilled Turkish BERT model
In this repository the MDZ Digital Library team (dbmdz) at the Bavarian State Library open sources a (cased) distilled model for Turkish 🎉
🇹🇷 DistilBERTurk


%55 16:24
Zoom
Leave
Recording
https://github.com/dbmdz/berts
The final training corpus has a size of 35GB and 44,04,976,662 tokens.
Detailed information about the data and pretraining steps can be found in this repository.
Additionally, we trained a distilled version of BERTurk: DistilBERTurk, that uses knowledge-distillation from BERTurk (teacher model). More information on distillation can be found in the excellent "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter" paper by Sanh et al. (2019).
Furthermore, we provide cased and uncased models trained with a larger vocab size (128k instead of 32k).
Model weights
Currently only PyTorch-Transformers compatible weights are available. If you need access to TensorFlow checkpoints, please raise an issue!
Model
Downloads
dbmdz/bert-base-turkish-cased
config.json · pytorch_model.bin · vocab.txt
dbmdz/bert-base-turkish-uncased
config.json · pytorch_model.bin · vocab.txt
dbmdz/bert-base-turkish-128k-cased
config.json · pytorch_model.bin · vocab.txt
dbmdz/bert-base-turkish-128k-uncased
config.json · pytorch_model.bin · vocab.txt
dbmdz/distilbert-base-turkish-cased
config.json · pytorch_model.bin · vocab.txt
Results
For results on PoS tagging or NER tasks, please refer to this repository.
Usage
Unmute
Start Video
Share
Participants
5
More