Deep learning in Bioinformatics

DATA MINING & BIOINFORMATICS LAB.

Index

1. Transfer learning
2. Text Classification
3. Datasets
4. Pre-trained model
5. Environment
6. Example
7. Practice

Transfer Learning

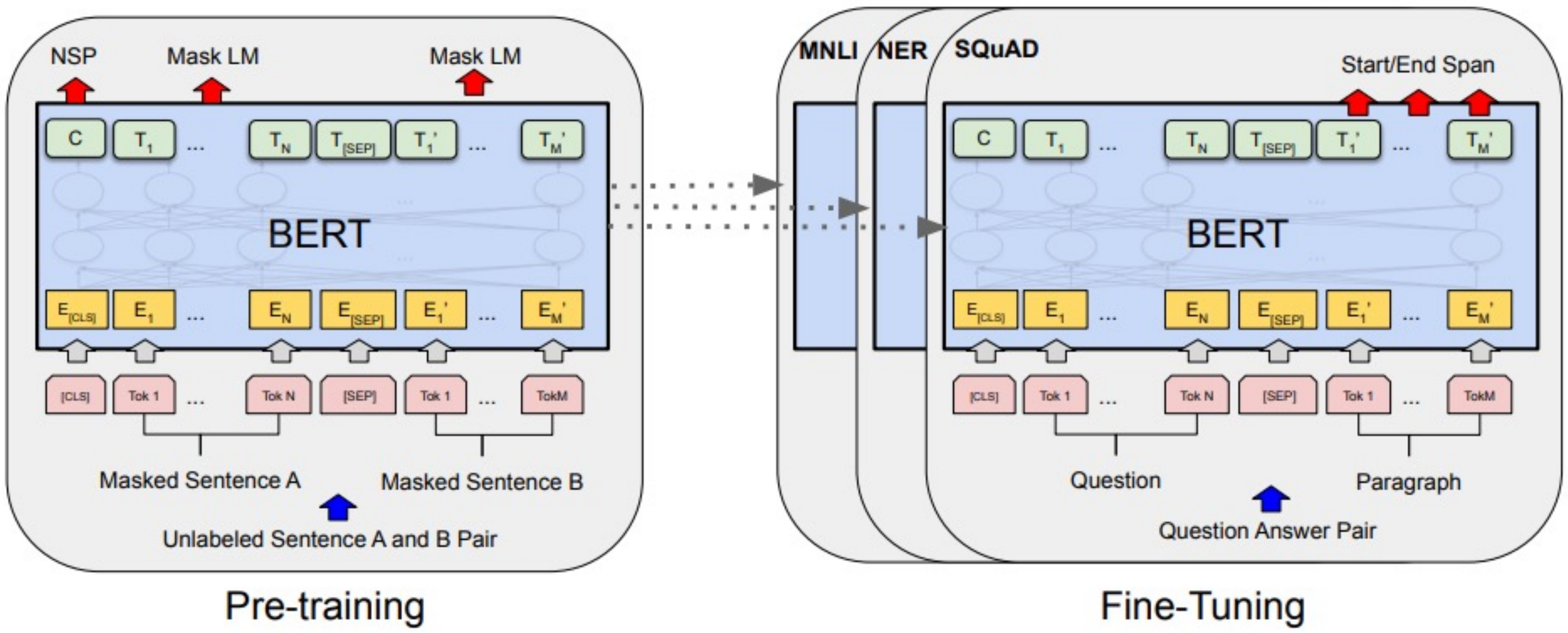
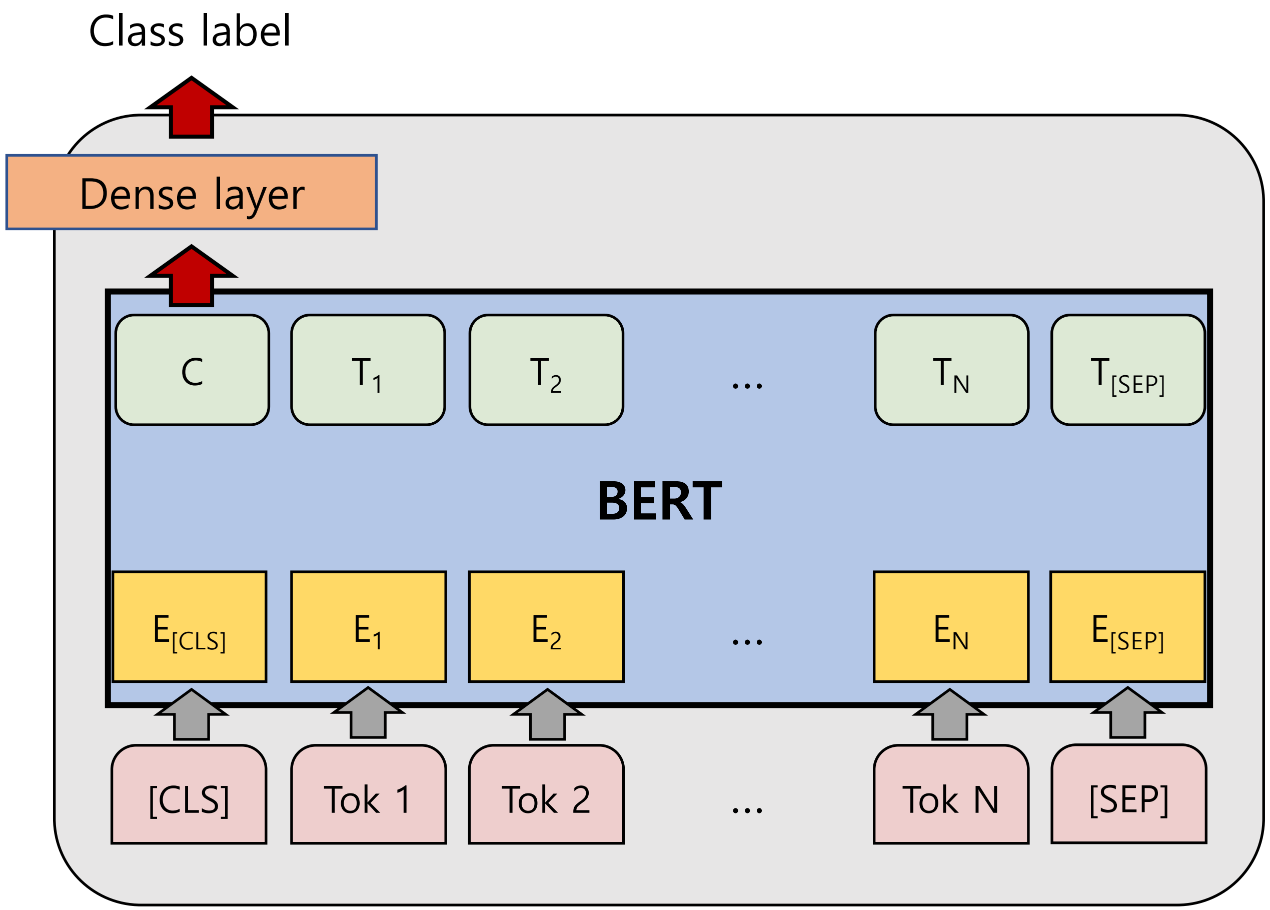


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Text Classification as transfer learning

- Text Classification is the task of assigning a label or class to a given text.
- Some use cases are sentiment analysis, natural language inference, assessing grammatical correctness.



Dataset – DeepLoc2.0

- These datasets provide proteins categorized into one or multiple of these ten locations: Cytoplasm, Nucleus, Extracellular, Cell membrane, Mitochondrion, Plastid, Endoplasmic reticulum, Lysosome/Vacuole, Golgi apparatus, Peroxisome.

ACC	Kingdom	Partition	Membrane	Cytoplasm	Nucleus	...	Sequence
Q28165	Metazoa	4	0	1	1	...	MAAAAAAAAAAGAAG...
Q86U42	Metazoa	4	0	1	1	...	MAAAAAAAAAAGAAG...
Q0GA42	Metazoa	3	1	0	0	...	MAAAAAAAAALGVRL...
P82349	Metazoa	1	1	1	0	...	MAAAAAAAAATEQQG...
Q7L5N1	Metazoa	1	0	1	1	...	MAAAAAAAAATNGTG...
Q96S94	Metazoa	0	0	0	1	...	MAAAAAAAGAAGSAA...
Q9CQ25	Metazoa	0	0	1	0	...	MAAAAAAAGGAALAV...
Q96P70	Metazoa	4	0	1	1	...	MAAAAAAGAASGLPG...
P63086	Metazoa	1	1	1	0	...	MAAAAAAGPEMVRGQ...
Q9UID3	Metazoa	3	0	0	0	...	MAAAAAAGPSPGSGP...
Q9SE42	Viridiplantae	3	0	1	0	...	MAAAAAAKIAPSMLS...

DeepLoc 2.0: multi-label subcellular localization prediction using protein language models

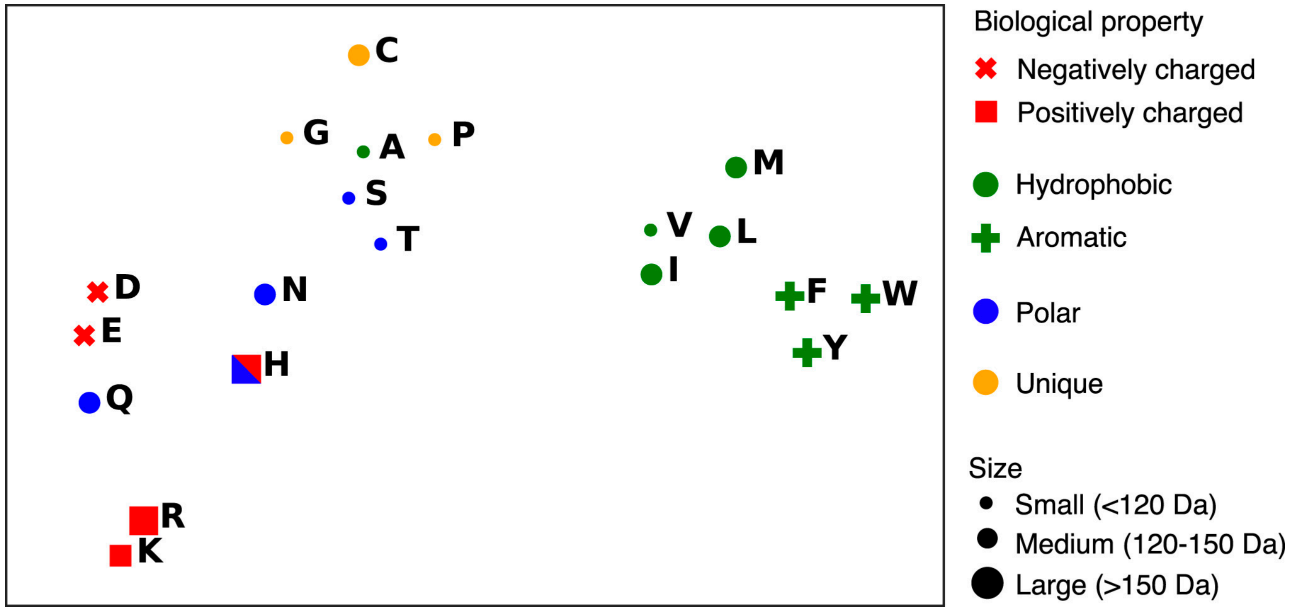
Dataset – DeepLoc2.0

- Toy dataset – Membrane (label)

index	acc	sequence	label
0	P63086	MAAAAAAGPEMVRGQ...	1
1	Q9UID3	MAAAAAAGPSPGSGP...	0
2	P51788	MAAAAAEEGMEPRAL...	1
3	Q3UCQ1	MAAAAALSGAGAPPA...	0
4	Q9NVF7	MAAAAEERMAEEGGG...	0
5	Q8CD10	MAAAAGRSAWLAAWG...	1
6	Q5VY80	MAAAAIPALLLCLPL...	1
7	Q96IV0	MAAAALGSSSGSASP...	0
8	O35094	MAAAALRGGWCRCPR...	1
9	Q9Y584	MAAAAPNAGGSAPET...	1
10	Q9NXG6	MAAAAVTGQRPETAA...	1

Pre-trained model – ESM

- Transformer protein language models from Meta AI's Fundamental AI Research Team.
- An unsupervised learning model to train a deep contextual language model on 86 billion amino acids across 250 million protein sequences spanning evolutionary diversity.



Biochemical properties of amino acids are represented in the Transformer model's output embeddings, visualized here with t-SNE. Through unsupervised learning, residues are clustered into hydrophobic, polar, and aromatic groups and reflect overall organization by molecular weight and charge. Visualization of 36-layer Transformer trained on UniParc.

https://github.com/facebookresearch/esm

(Proceedings of the National Academy of Sciences, 118(15), e2016239118.)

Environment

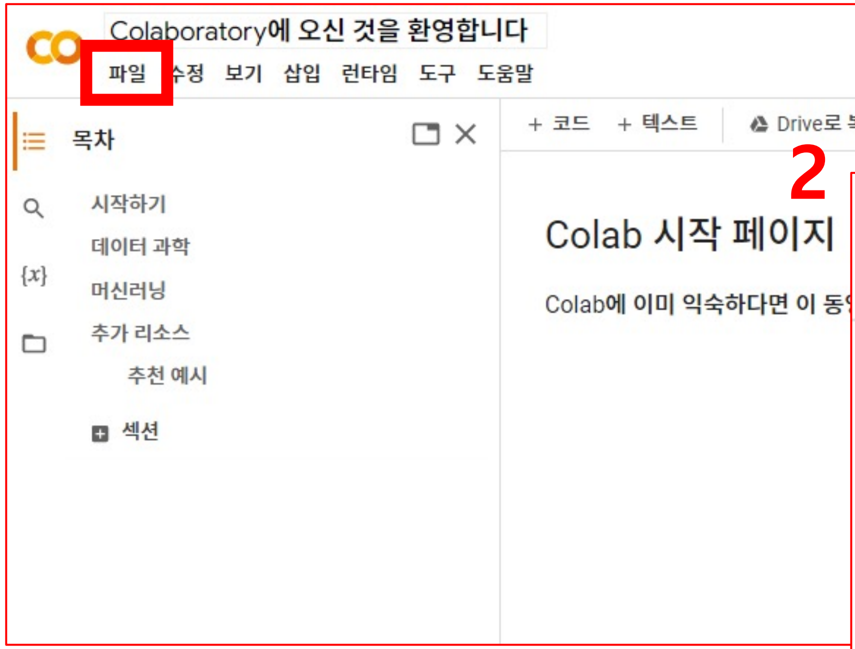
- Colab
    - Python 3.7.13
    - CUDA 11.2
    - Numpy 1.21.6
    - Scikit-learn 1.0.2
- Pytorch 1.12.0
- Transformers 4.24.0
- Datasets 2.7.1
- Evaluate 0.3.0

How to setup the environment

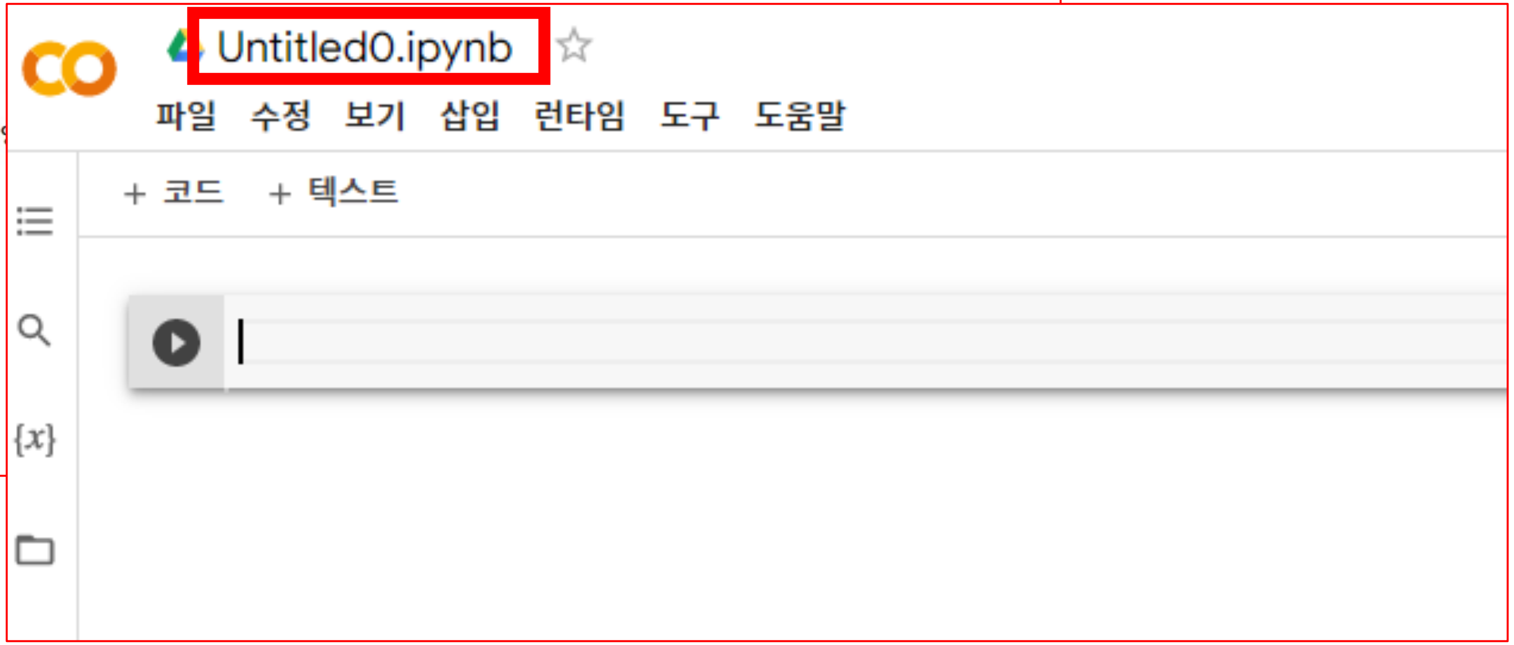
Colab

- https://colab.research.google.com/?hl=ko
- [파일]-[새노트]

1



2

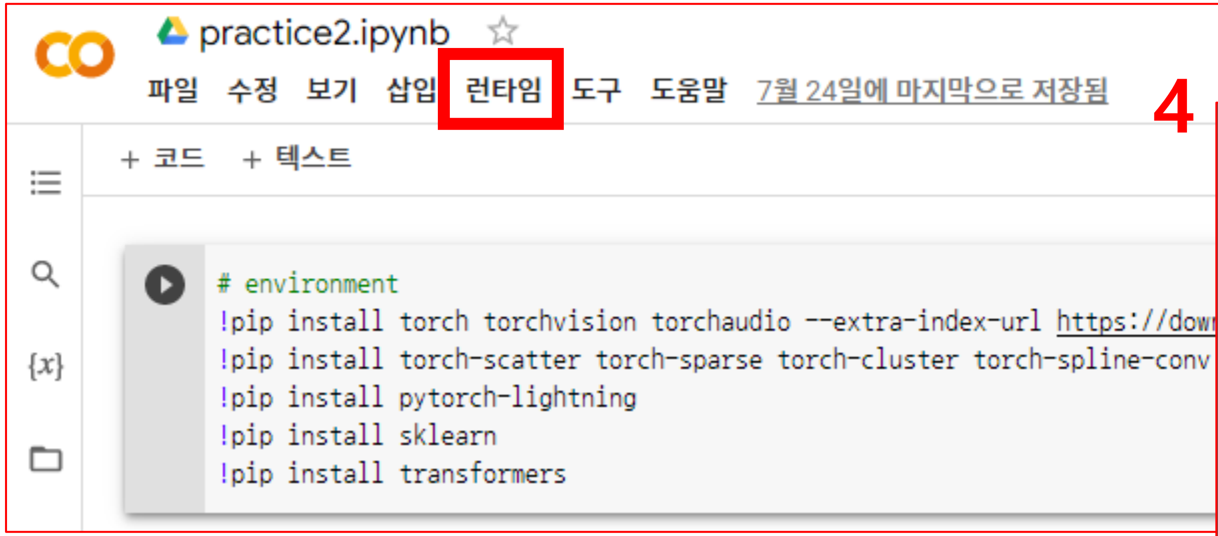


How to setup the environment

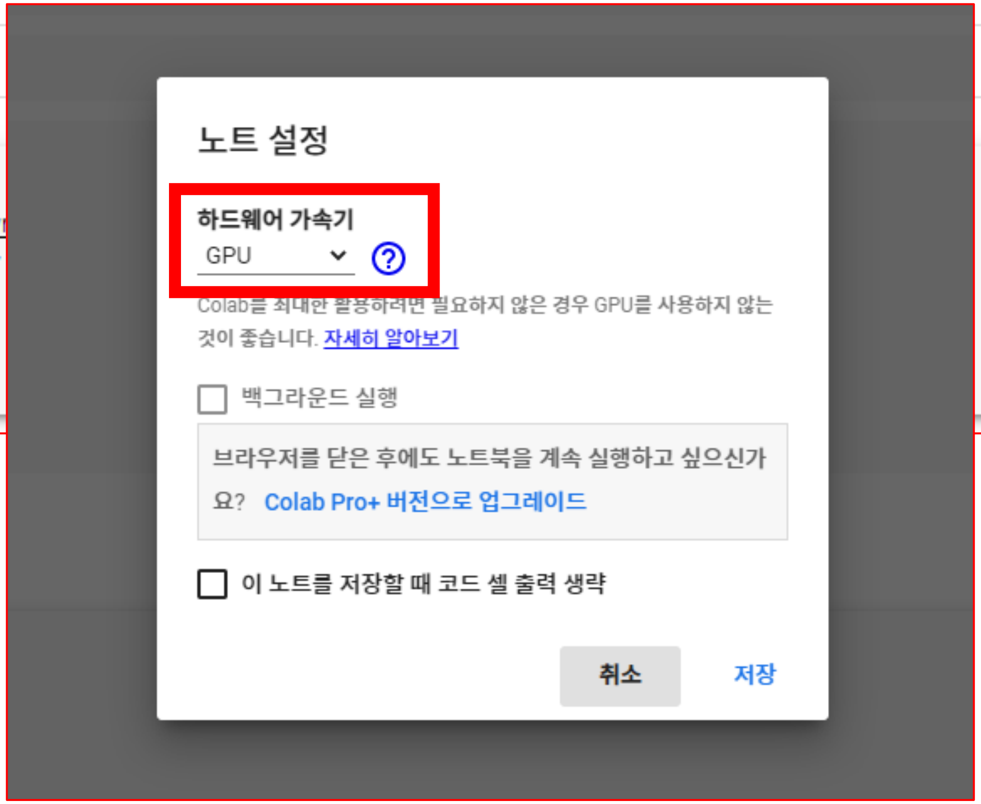
Colab

- [런타임]-[런타임 유형 변경]

3



4



How to setup the environment

Colab

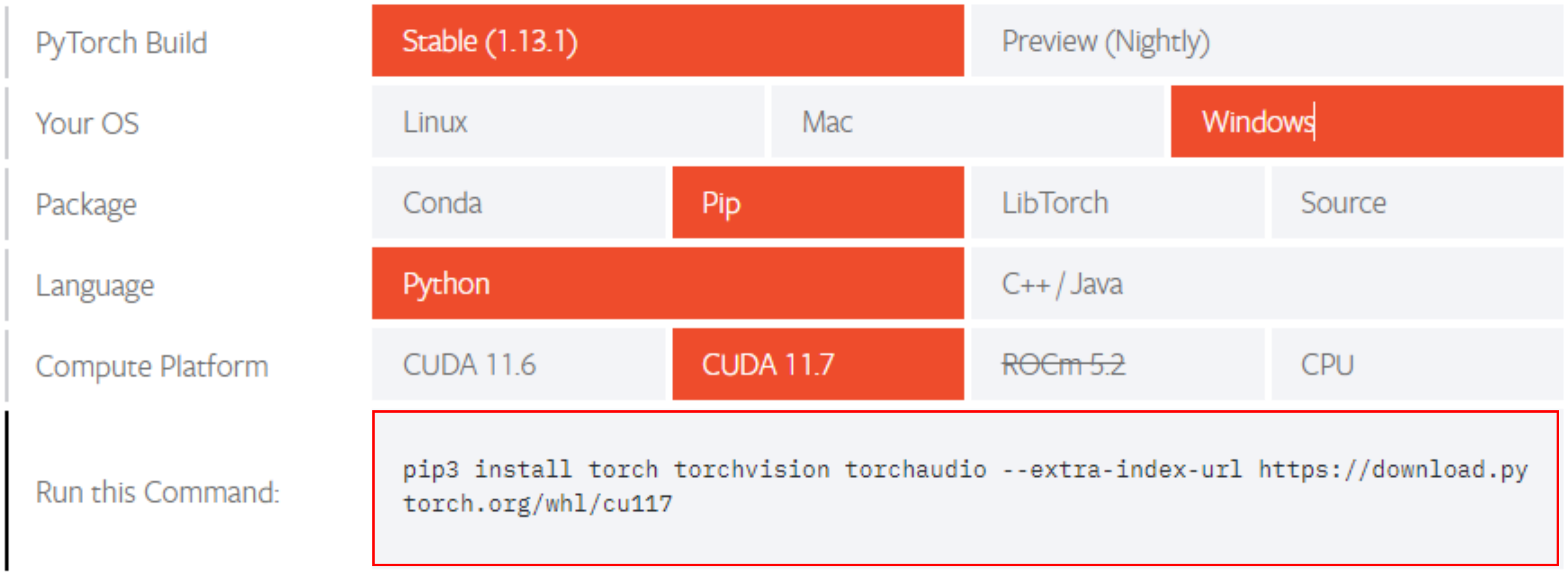
- IPython
  - Shell 명령어: ! 필요 (e.g. !pip)

```
!pip install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cu117
!pip install transformers
!pip install datasets
!pip install evaluate
!pip install scikit-learn
```

How to setup the environment

Pytorch

- https://pytorch.org/
- 1.13.1
- Linux
- Pip
- Python
- CUDA 11.7



Example - How to train Sequence Classification Model with transformers

Load datasets

```
data_dir = "KSBi-BIML2023/dataset"
train_data_file = data_dir + "/toy_train.csv"
valid_data_file = data_dir + "/toy_valid.csv"
```

```
data_files = {"train": train_data_file, "valid": valid_data_file}

raw_datasets = load_dataset(
    "csv",
    data_files=data_files,
    cache_dir=data_dir,
    use_auth_token=None,
)

print(raw_datasets)
```

Example - How to train Sequence Classification Model with transformers

Load datasets

```
data_dir = "KSBi-BIML2023/dataset"
train_data_file = data_dir + "/toy_train.csv"
valid_data_file = data_dir + "/toy_valid.csv"
```

```
data_files = {"train": train_data_file, "valid": valid_data_file}

raw_datasets = load_dataset(
    "csv",
    data_files=data_files,
    cache_dir=data_dir,
    use_auth_token=None,
)

print(raw_datasets)
```

```
DatasetDict({
    train: Dataset({
        features: ['index', 'acc', 'sequence', 'label'],
        num_rows: 2700
    })
    valid: Dataset({
        features: ['index', 'acc', 'sequence', 'label'],
        num_rows: 300
    })
})
```

Example - How to train Sequence Classification Model with transformers

Load pretrained model

```
pretrained_model_name = "facebook/esm2_t6_8M_UR50D"
```

```
config = AutoConfig.from_pretrained(
    pretrained_model_name,
    num_labels=num_labels,
)
tokenizer = AutoTokenizer.from_pretrained(
    pretrained_model_name,
    config=config,
)
model = EsmForSequenceClassification.from_pretrained(
    pretrained_model_name,
    config=config
)
```

https://huggingface.co/models

Example - How to train Sequence Classification Model with transformers

Load pretrained model

```
pretrained_model_name = "facebook/esn
```

```
config = AutoConfig.from_pretrained(
    pretrained_model_name,
    num_labels=num_labels,
)
tokenizer = AutoTokenizer.from_pretrai
    pretrained_model_name,
    config=config,
)
model = EsmForSequenceClassification.f
    pretrained_model_name,
    config=config
)
```

```
outputs = self.esm(
    input_ids,
    attention_mask=attention_mask,
    position_ids=position_ids,
    head_mask=head_mask,
    inputs_embeds=inputs_embeds,
    output_attentions=output_attentions,
    output_hidden_states=output_hidden_states,
    return_dict=return_dict,
)
sequence_output = outputs[0]
logits = self.classifier(sequence_output)
```

```
loss_fct = CrossEntropyLoss()
loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
```

```
output = (logits,) + outputs[2:]
return ((loss,) + output) if loss is not None else output
```

https://huggingface.co/models

Example - How to train Sequence Classification Model with transformers

Default pooling layer – use [CLS] token

```python
class EsmClassificationHead(nn.Module):
    """Head for sentence-level classification tasks."""

    def __init__(self, config):
        super().__init__()
        self.dense = nn.Linear(config.hidden_size, config.hidden_size)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.out_proj = nn.Linear(config.hidden_size, config.num_labels)

    def forward(self, features, **kwargs):
        x = features[:, 0, :]  # take <s> token (equiv. to [CLS])
        x = self.dropout(x)
        x = self.dense(x)
        x = torch.tanh(x)
        x = self.dropout(x)
        x = self.out_proj(x)
        return x
```

Example - How to train Sequence Classification Model with transformers

Set trainer (1)

```
def preprocess_function(examples):
  result = tokenizer(
      examples["sequence"],
      padding="longest",
      truncation=True
  )
  return result

train_dataset = raw_datasets["train"].map(preprocess_function, batched=True)
valid_dataset = raw_datasets["valid"].map(preprocess_function, batched=True)

metric = evaluate.load("accuracy")

def compute_metrics(p):
    preds = p.predictions[0] if isinstance(p.predictions, tuple) else p.predictions
    preds = np.argmax(preds, axis=1)

    result = metric.compute(predictions=preds, references=p.label_ids)
    return result
```

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Example - How to train Sequence Classification Model with transformers

Set trainer (2)

```
training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=num_epochs,     # total number of training epochs
    per_device_train_batch_size=1,   # batch size for evaluation
    do_train=True,                   # perform training
    save_strategy="no"               # checkpoint save strategy
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=compute_metrics,
    tokenizer=tokenizer,
    data_collator=None,
)
```

Example - How to train Sequence Classification Model with transformers

Train the model

```
train_result = trainer.train()
metrics = train_result.metrics

print(metrics)

trainer.save_model()  # Saves the tokenizer too for easy upload

trainer.log_metrics("train", metrics)
trainer.save_metrics("train", metrics)
trainer.save_state()
```

Example - How to train Sequence Classification Model with transformers

Train the model

```
train_result = trainer.train()
metrics = train_result.metrics

print(metrics)

trainer.save_model()  # Saves the toke

trainer.log_metrics("train", metrics)
trainer.save_metrics("train", metrics)
trainer.save_state()
```

```
***** Running training *****
  Num examples = 2700
  Num Epochs = 1
  Instantaneous batch size per device = 1
  Total train batch size (w. parallel, distributed & accumulation) = 1
  Gradient Accumulation steps = 1
  Total optimization steps = 2700
  Number of trainable parameters = 7840763
```

[2700/2700 03:27, Epoch 1/1]

Step	Training Loss
500	0.738900
1000	0.816600
1500	0.808200
2000	0.744200
2500	0.797800

Example - How to train Sequence Classification Model with transformers

Evaluate the model

```python
print("*** Evaluation ***")
metrics = trainer.evaluate(eval_dataset=valid_dataset)
trainer.log_metrics("eval", metrics)
trainer.save_metrics("eval", metrics)
```

```
***** Running Evaluation *****
  Num examples = 300
  Batch size = 8
*** Evaluation ***
[38/38 00:06]
***** eval metrics *****
  epoch                   =        1.0
  eval_accuracy           =     0.8267
  eval_loss               =     0.6252
  eval_runtime            = 0:00:07.17
  eval_samples_per_second =     41.815
  eval_steps_per_second   =      5.297
```

Example - How to train Sequence Classification Model with transformers

Predict the location of sequences

```
print("*** Prediction ***")
predict_dataset = valid_dataset
predictions, labels, metrics = trainer.predict(predict_dataset, metric_key_prefix="predict")
predictions = np.argmax(predictions, axis=1)

print("ACC\tSequence\tLabel\tPredictions")
for index, (pred, label) in enumerate(zip(predictions[:10], labels[:10])):
    print(predict_dataset["acc"][index],
          predict_dataset["sequence"][index][:15]+"...",
          label,
          pred)
```

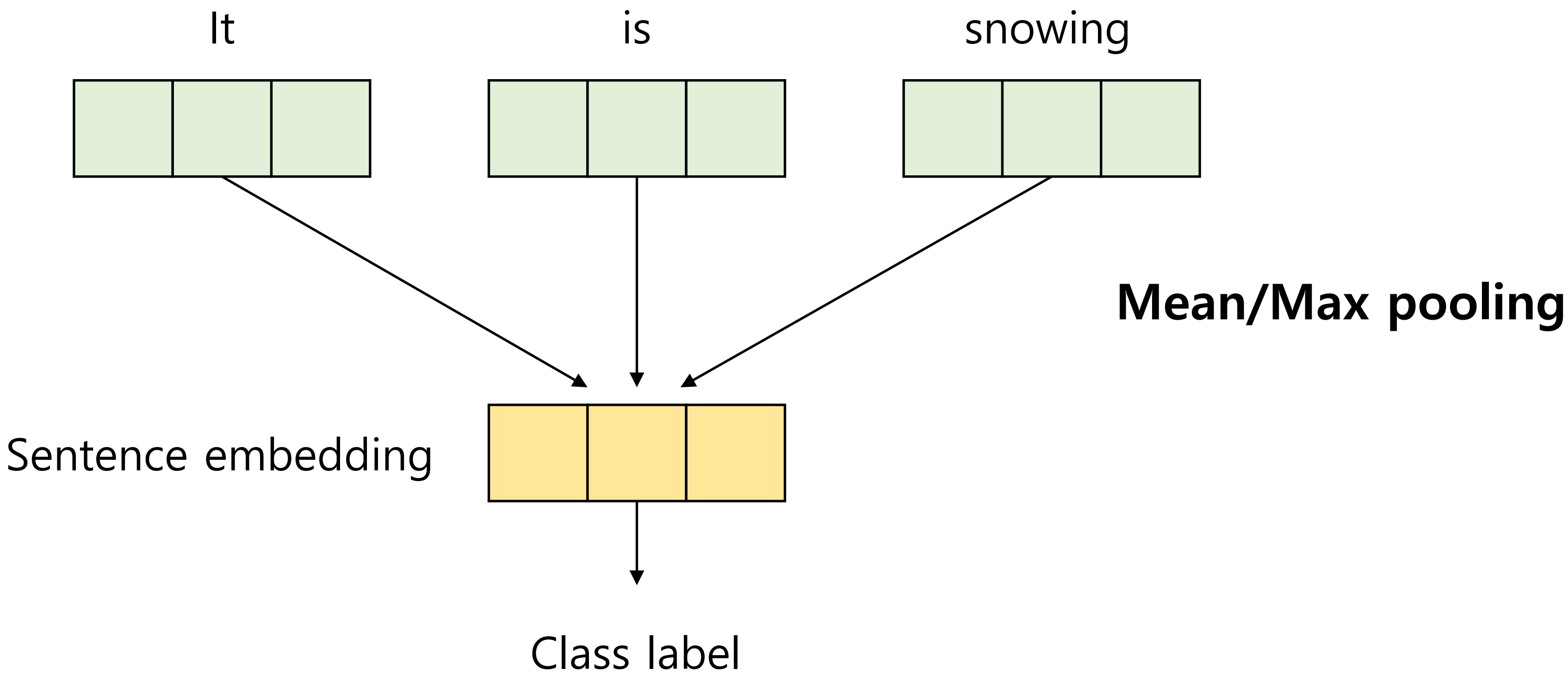
Example - How to train Sequence Classification Model with transformers

Predict the location of sequences

```python
print("*** Prediction ***")
predict_dataset = valid_dataset
predictions, labels, metrics = trainer.predict(predict_dataset, metric_key_prefix="predict")
predictions = np.argmax(predictions, axis=1)

print("ACC\tSequence\tLabel\tPredictions")
for index, (pred, label) in enumerate(zip(predictions[:10], lak
    print(predict_dataset["acc"][index],
          predict_dataset["sequence"][index][:15]+"...",
          label,
          pred)
```

```
***** Running Prediction *****
  Num examples = 300
  Batch size = 8
*** Prediction ***
ACC       Sequence         Label   Predictions
Q0GA42 MAAAAAAAAALGVRL... 1 1
Q96BY9 MAAACGPGAAGYCLL... 1 1
Q60415 MAAAEEGCDAGVEAD... 1 0
B5DEL3 MAAAIGVRGRFELLS... 0 1
P26453-2 MAAALLLALAFTFLS... 1 1
Q8BKJ9 MAAGGGLSRSERKAA... 0 0
O88951 MAALVEPLGLERDVS... 1 0
Q6AVT2 MAAMDLRVAAPASVA... 0 0
P16298 MAAPEPARAAPPPPP... 0 0
Q8N2H3 MAASGRGLCKAVAAS... 0 0
```

Practice – adopt other pooling layer (Mean/Max Pooling)



Practice – adopt other pooling layer (Mean Pooling)

torch.mean(*input*, *dim*, *keepdim=False*, *, *dtype=None*, *out=None*) → Tensor

https://pytorch.org/docs/stable/generated/torch.mean.html

Example:

```
>>> a = torch.randn(4, 4)
>>> a
tensor([[-0.3841,  0.6320,  0.4254, -0.7384],
        [-0.9644,  1.0131, -0.6549, -1.4279],
        [-0.2951, -1.3350, -0.7694,  0.5600],
        [ 1.0842, -0.9580,  0.3623,  0.2343]])
>>> torch.mean(a, 1)
tensor([-0.0163, -0.5085, -0.4599,  0.1807])
>>> torch.mean(a, 1, True)
tensor([[-0.0163],
        [-0.5085],
        [-0.4599],
        [ 0.1807]])
```

Practice – adopt other pooling layer (Max Pooling)

torch.max(*input*, *dim*, *keepdim=False*, *, *out=None*)

https://pytorch.org/docs/stable/generated/torch.max.html

Example:

```
>>> a = torch.randn(4, 4)
>>> a
tensor([[-1.2360, -0.2942, -0.1222,  0.8475],
        [ 1.1949, -1.1127, -2.2379, -0.6702],
        [ 1.5717, -0.9207,  0.1297, -1.8768],
        [-0.6172,  1.0036, -0.6060, -0.2432]])
>>> torch.max(a, 1)
torch.return_types.max(values=tensor([0.8475, 1.1949, 1.5717, 1.0036]),
indices=tensor([3, 0, 0, 1]))
```

Practice – adopt other pooling layer

Create the custom model (1)

```
class CustomModelForSequenceClassification(EsmPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels
        self.config = config

        self.esm = EsmModel(config, add_pooling_layer=False)
        if config.pooling == "max":
            self.pooling = MaxPooling()
        elif config.pooling == "mean":
            self.pooling = MeanPooling()
        else:
            self.pooling = CLSPooling()

        self.classifier = nn.Sequential(
            nn.Dropout(config.hidden_dropout_prob),
            nn.Linear(config.hidden_size, config.hidden_size),
            nn.Tanh(),
            nn.Dropout(config.hidden_dropout_prob),
            nn.Linear(config.hidden_size, config.num_labels),
        )
        self.init_weights()
```

Practice – adopt other pooling layer

Create the custom model (2)

```
def forward(
        self,
        input_ids: Optional[torch.LongTensor] = None,
        attention_mask: Optional[torch.Tensor] = None,
        position_ids: Optional[torch.LongTensor] = None,
        head_mask: Optional[torch.Tensor] = None,
        inputs_embeds: Optional[torch.FloatTensor] = None,
        labels: Optional[torch.LongTensor] = None,
        output_attentions: Optional[bool] = None,
        output_hidden_states: Optional[bool] = None,
        return_dict: Optional[bool] = None,
) -> Union[Tuple, SequenceClassifierOutput]:

    return_dict = return_dict if return_dict is not None else self.config.use_ret

    outputs = self.esm(
        input_ids,
        attention_mask=attention_mask,
        position_ids=position_ids,
        head_mask=head_mask,
        inputs_embeds=inputs_embeds,
        output_attentions=output_attentions,
        output_hidden_states=output_hidden_states,
        return_dict=return_dict,
```

Practice – adopt other pooling layer

Create the custom model (3)

```
sequence_output = outputs[0]
sequence_output = self.pooling(sequence_output)
logits = self.classifier(sequence_output)

loss = None
if labels is not None:
    loss_fct = CrossEntropyLoss()
    loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))

if not return_dict:
    output = (logits,) + outputs[2:]
    return ((loss,) + output) if loss is not None else output

return SequenceClassifierOutput(
    loss=loss,
    logits=logits,
    hidden_states=outputs.hidden_states,
    attentions=outputs.attentions,
)
```

Practice – adopt other pooling layer

Run new model (1)

```
pooling_list = ["mean", "max"]

for pooling in pooling_list:
  config.pooling = pooling

  training_args = TrainingArguments(
      output_dir=f'./results',          # output directory
      num_train_epochs=num_epochs,      # total number of training epochs
      per_device_train_batch_size=1,    # batch size for evaluation
      do_train=True,                    # perform training
      save_strategy="no"                # checkpoint save strategy
  )

  model = CustomModelForSequenceClassification.from_pretrained(
      pretrained_model_name,
      config=config
  )

  trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=valid_dataset,
    compute_metrics=compute_metrics,
    tokenizer=tokenizer,
    data_collator=None,
  )
```

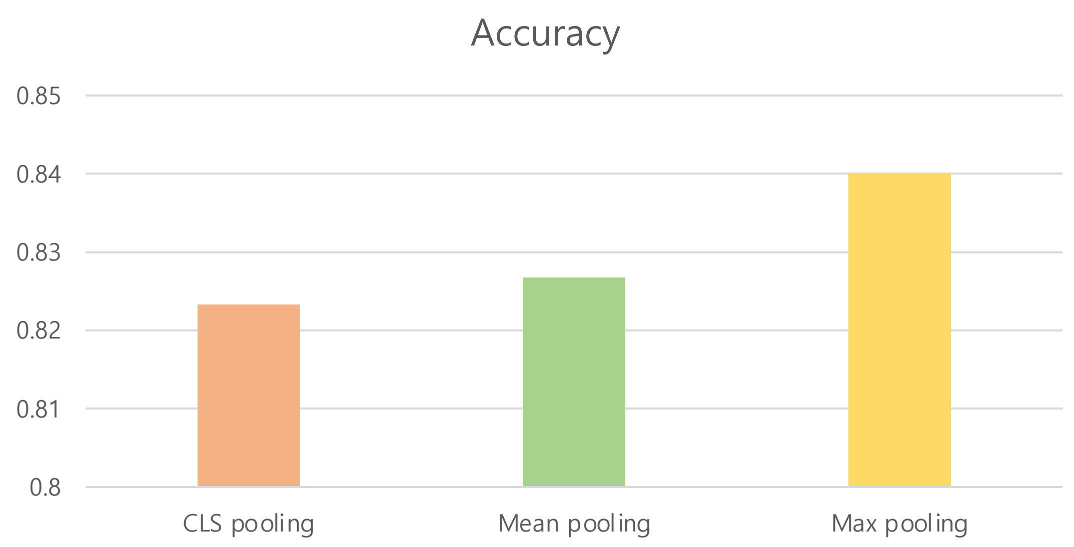
Practice – adopt other pooling layer

Run new model (2)

```
## Train
train_result = trainer.train()
metrics = train_result.metrics

print(metrics)

trainer.save_model()  # Saves the tokenizer too for easy upload

trainer.log_metrics("train", metrics)
trainer.save_metrics("train", metrics)
trainer.save_state()

## Prediction
print("*** Prediction ***")
predict_dataset = valid_dataset
predictions, labels, metrics = trainer.predict(predict_dataset, metric_key_prefix="predict")
predictions = np.argmax(predictions, axis=1)

print("ACC\tSequence\tLabel\tPredictions")
for index, (pred, label) in enumerate(zip(predictions[:10], labels[:10])):
    print(predict_dataset["acc"][index],
          predict_dataset["sequence"][index][:15]+"...",
          label,
          pred)
print(metrics)
```

Practice – adopt other pooling layer

Result



	Acc
CLS pooling	0.8233
Mean pooling	0.8267
Max pooling	0.84