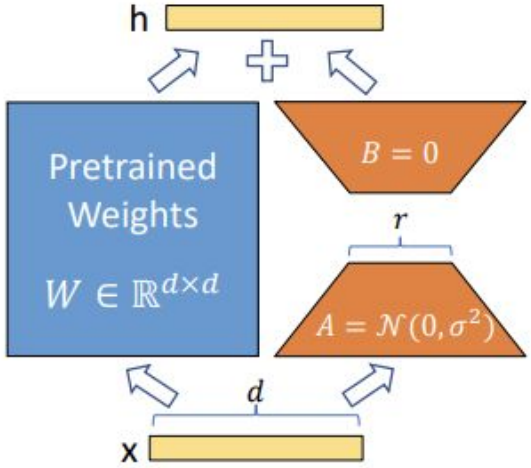
1, 2주차 (10/23~11/3)

3L

이론 배경

- LoRA
- Normalized subspace similarity
- Main idea

이론 배경 (LoRA)



ΔW ≈ BA (compressed, truncated(?))

W, ΔW = [m,n], B = [m,r], A = [r,n]

초기 가설: ΔW = BA

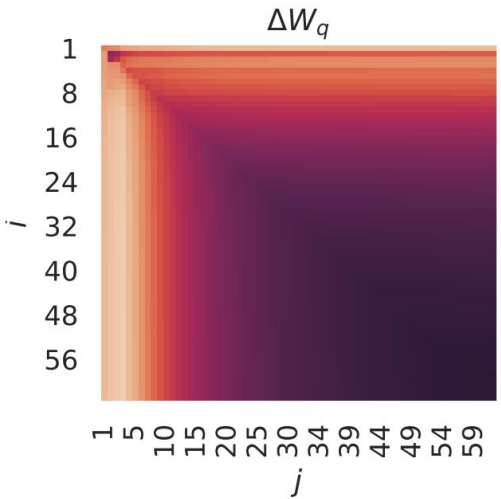
변경된 가설: ΔW ≈ BA

ΔW가 가능하다면, W도 compression 가능할 것이다

성능을 해치지 않고, weight compression의 방법으로 LoRA를 사용하자!

이론 배경 (Normalized subspace similarity)

$$\phi(A_{r=8}, A_{r=64}, i, j) = \frac{||U^{i\top}_{A_{r=8}} U^{j}_{A_{r=64}}||^2_F}{\min(i,j)} \in [0,1]$$



LoRA 논문 11p. https://arxiv.org/pdf/2106.09685.pdf

A1,2 = 각기 다른 랜덤시드로 LoRA 했을때의 행렬 A

U: A를 svd해서 얻은 right singular vectors

분자: Grassmann distance(기저 차원 유사도 계산)

분모: normalization(차원수 증가에 따른 페널티)

같은 과정을 B에 대한 svd의 left singular vectors를 통해서도 같은 효과를 볼 수 있다

의의: 각기 다른 랜덤시드에 대해서 학습할때 동일하게 주목하는 ΔW의 subspace의 rank 수를 추정할 수 있다.

이론 배경 - Main idea

배경: LoRA 는 ‘다양한 데이터’로 학습된 ‘pre-train model’을 ‘내' 데이터에 ‘적응' 시키는 방법 중 하나이다.

착안점: 모델의 학습으로 인한 변화량을 저차원의 adapter에 compress하는 LoRA 방식으로부터 pre-train model 또한 저차원으로 compress 가능 하리라는 생각을 가질 수 있다.

실험 설계: 그 이론을 증명하기 위해 잘 훈련된 모델(pre-train model or fine tuned model)의 weights들에 0.9(scale factor or transform factor)를 scaling 해주고, fine tuning에 사용한 data 로 LoRA 학습

실험 설계 심화: 학습이 진행 됨에 따라 original 에 페널티를 가하는 스케쥴링을 통해 original weight의 핵심 성능을 adapter에 transfer가 가능할 수 있다

진행 사항

- Toy Project(MNIST)
- Our LoRA layer
- GPT2 with LoRA layer

진행사항 - Toy Project (MNIST)

- 공식 Keras Code Example 에 있는 LoRA layer 는 논문의 이론에 부합하지 않음

https://keras.io/examples/nlp/parameter_efficient_finetuning_of_gpt2_with_lora/

- 오른쪽의 LoRA Layer Class 내에서 A,B 를 구현한 부분을 살펴 보면, Weights 가 아니라 Dense Layer 그 자체로 구현해둠
- 공식 Microsoft LoRA Pytorch 코드를 참조하여, Keras Layer 에 적용할 수 있는 형태로 변경

```
        # LoRA dense layers.
        self.A = keras.layers.Dense(
            units=rank,
            use_bias=False,
            # Note: the original paper mentions that normal distribution was
            # used for initialization. However, the official LoRA implementation
            # uses "Kaiming/He Initialization".
            kernel_initializer=keras.initializers.VarianceScaling(
                scale=math.sqrt(5), mode="fan_in", distribution="uniform"
            ),
            trainable=trainable,
            name=f"lora_A",
        )
        # B has the same `equation` and `output_shape` as the original layer.
        # `equation = abc,cde->abde`, where `a`: batch size, `b`: sequence
        # length, `c`: `hidden_dim`, `d`: `num_heads`,
        # `e`: `hidden_dim//num_heads`. The only difference is that in layer `B`,
        # `c` represents `rank`.
        self.B = keras.layers.EinsumDense(
            equation=original_layer_config["equation"],
            output_shape=original_layer_config["output_shape"],
            kernel_initializer="zeros",
            trainable=trainable,
            name=f"lora_B",
        )

    def call(self, inputs):
        original_output = self.original_layer(inputs)
        if self.trainable:
            # If we are fine-tuning the model, we will add LoRA layers' output
            # to the original layer's output.
            lora_output = self.B(self.A(inputs)) * self._scale
            return original_output + lora_output

        # If we are in inference mode, we "merge" the LoRA layers' weights into
        # the original layer's weights - more on this in the text generation
        # section!
        return original_output
```

Our LoRA Layer (for Dense Layer)

- Weights 로만, A,B 구현
- 논문과 똑같은 형태로 처리
- Wx + BAx

```
def build(self, input_shape):
    # LoRA weights.
    self.A_weight = self.add_weight(
        name="lora_A_weight",
        shape=(self.rank, input_shape[-1]),
        initializer=keras.initializers.VarianceScaling(
            scale=math.sqrt(5), mode="fan_in", distribution="uniform"
        ),
        trainable=self.trainable,
    )

    self.B_weight = self.add_weight(
        name="lora_B_weight",
        shape=(self.original_layer.units, self.rank),
        initializer="zeros",
        trainable=self.trainable,
    )

    super().build(input_shape)

def call(self, inputs):
    original_output = self.original_layer(inputs)
    if self.trainable:
        # Matrix multiplication for A and B weights with inputs
        lora_A_output = tf.matmul(self.A_weight, tf.transpose(inputs)) #Ax
        lora_output = tf.transpose(tf.matmul(self.B_weight, lora_A_output) * self._scale) # BAx Transpose back to [batch_size, original_layer.units]
        return original_output + lora_output

    return original_output
```

Our LoRA Layer (for EinsumDense Layer)

```
def build(self, input_shape):
    # LoRA weights.
    self.A_weight = self.add_weight(
        name="lora_A_weight",
        shape=(self.rank, input_shape[-1]),
        initializer=keras.initializers.VarianceScaling(
        scale=math.sqrt(5), mode="fan_in", distribution="uniform"
        ),
        trainable=self.trainable,
)
    kernel_shape = self.original_layer.kernel.shape
    self.B_weight = self.add_weight(
            name="lora_B_weight",
            shape=(self.rank,) + kernel_shape[1:],
            initializer="zeros",
            trainable=self.trainable,
    )

    super().build(input_shape)

def call(self, inputs):
    original_output = self.original_layer(inputs)
    if self.trainable:
        # Matrix multiplication for A and B weights with inputs
        lora_A_output = tf.matmul(self.A_weight, tf.transpose(inputs)) #Ax
        lora_output = tf.einsum(self.original_layer.equation, tf.transpose(lora_A_output) , self.B_weight) * self._scale  # BAx Transpose back to [batch_size, original_layer.units]
        return original_output + lora_output
```

Toy project Original model v1

```
[ ]  import tensorflow as tf
     from tensorflow.keras.layers import Input, Flatten, Dense
     from tensorflow.keras.models import Model

     # 1. 기존 모델 정의 및 학습
     inputs = Input(shape=(28, 28))
     x = Flatten()(inputs)
     x = Dense(128, activation='relu')(x)
     x = Dense(128, activation='relu')(x)
     outputs = Dense(10)(x)
     model = Model(inputs=inputs, outputs=outputs)
     model.compile(optimizer='adam',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])
     model.summary()
     model.fit(train_images, train_labels, epochs=10)
```

```
Model: "model_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_27 (InputLayer)       [(None, 28, 28)]          0

 flatten_26 (Flatten)        (None, 784)               0

 dense_72 (Dense)            (None, 128)               100480

 dense_73 (Dense)            (None, 128)               16512

 dense_74 (Dense)            (None, 10)                1290

_________________________________________________________________
```

Fashion MNIST classification model

target layer: 128 * 128

(0.9 scaling이 예정된 layer)

```
Test accuracy: 0.8863999843597412
```

Toy project - LoRA

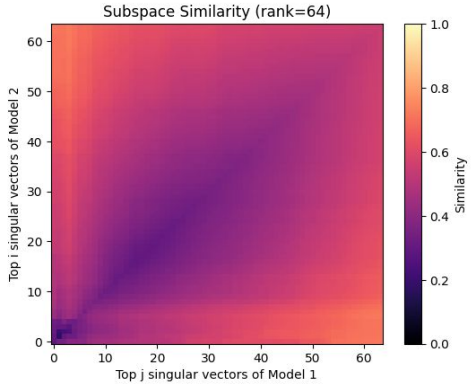
Fashion MNIST classification model

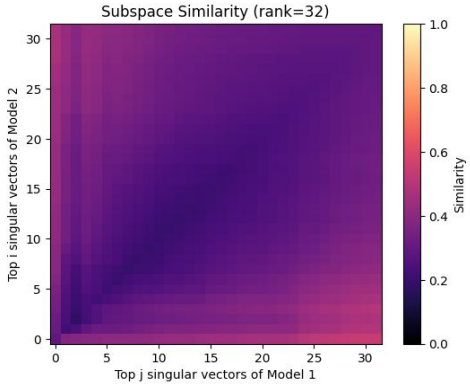
	rank = 64	rank=32	rank=80	rank=5
test acc	0.8762999773025513	0.8802999855408783	0.8808000087738037	0.8687000274658203

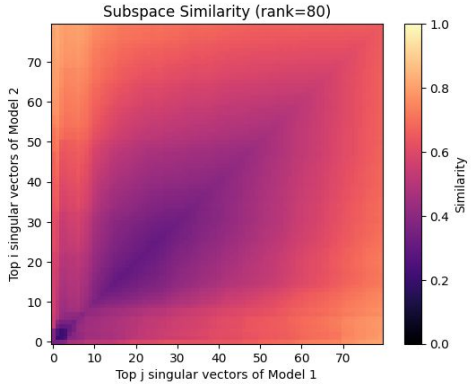
target layer weight에 0.9를 scaling 후 2번의 독립적인 LoRA 시도

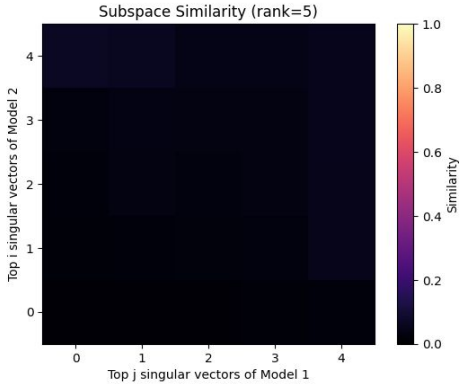
rank에 변화를 주며 두 adapter의 A 행렬의 subspace similarity 비교

현재 해석: 0.1W에서 변화된 BA의 주요 subspace 성분의 rank는 대략 8에서 12정도의 값을 가지는듯 함.









0.9W(freeze) + BA로 학습시, BA는 0.1W을 복원할까?에 대한 내용

[0.1W] vs [0.1W(truncated)*] vs [BA]

0.1W(truncated)*: W의 SVD를 통해 BA에서 사용된 rank 만큼의 성분에 대한 것만 남기고 다시 복원

```
✓ 0초 [30] np.linalg.norm(W_U @ tf.transpose(BA_09_U), 'fro')**2

       64.0

✓ 0초 [31] np.linalg.norm(W_U @ tf.transpose(BA_o_U), 'fro')**2

       63.999992370605696

   [ ] np.linalg.norm(W_U @ tf.transpose(BA_o_U), 'fro')**2
```

진행사항 - GPT-2에 LoRA 적용

Keras tutorial task로 진행 (Torch로 전환하기 전);

dataset: reddit

task: text generation

- https://colab.research.google.com/drive/1us0wQFaLVpzotvnHWkR7YTDOQXy6UuvU?usp=sharing

```
gpt2_lm.fit(train_ds, epochs=EPOCHS, callbacks=[gpu_memory_callback])
gpt2_lm_memory_usage = gpu_memory_callback.memory_usage
```

```
Epoch 1/5
500/500 [==============================] - 593s 882ms/step - loss: 3.2987 - accuracy: 0.3268
Epoch 2/5
500/500 [==============================] - 441s 882ms/step - loss: 3.1311 - accuracy: 0.3482
Epoch 3/5
500/500 [==============================] - 427s 854ms/step - loss: 3.0391 - accuracy: 0.3598
Epoch 4/5
500/500 [==============================] - 428s 855ms/step - loss: 2.9696 - accuracy: 0.3684
Epoch 5/5
500/500 [==============================] - 426s 852ms/step - loss: 2.9089 - accuracy: 0.3756
```

문제점: 정확도가 떨어짐

기존 dataset의 noise 등의 우려로 다른 dataset과 task로 변경;

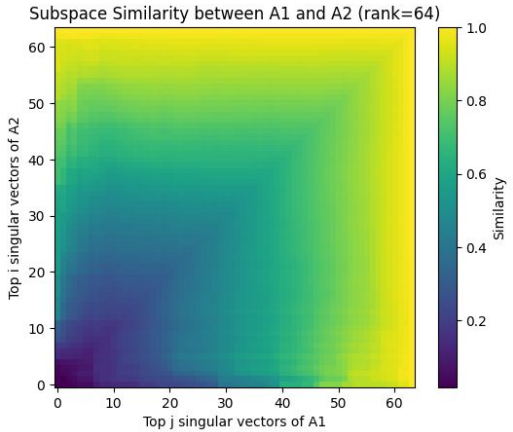
dataset: Large Movie Review

task: gpt2 classification

- https://gmihaila.github.io/tutorial_notebooks/gpt2_finetune_classification/

진행사항 - GPT-2에 LoRA 적용

Keras tutorial task 진행하며 subspace similarity 시각화



- https://colab.research.google.com/drive/1us0wQFaLVpzotvnHWkR7YTDOQXy6UuvU?usp=sharing

Toy Project 진행 중 적용된 수정 사항 반영

- 논문 구현을 위한 LoRA layer 코드 수정
- Subspace similarity 공식 수정