Dialogue Summarization I 일상 대화 요약

학교 생활, 직장, 치료, 쇼핑, 여가, 여행 등 광범위한 일상 생활 중 하는 대화들에 대해 요약합니다.

#비공개대회 #AI부트캠프14기 #NLPAdvanced

[공유] Hyperparameter Tuning으로 최적의 조합 찾기

크리스(운영진) · 2025.06.18 14:34 · 조회수 13 3 0

Hyperparameter Tuning

1. **하이퍼파라미터 튜닝이란?**

머신러닝에서 하이퍼파라미터란 사용자의 입력값, 혹은 설정 가능한 옵션이라고 볼 수 있습니다. 즉, 모델이 학습하면서 자체적으로 최적의 값으로 정해지는 것이 아닌 사람이 직접 지정해 주어야 하는 변수를 말합니다. 따라서, 여러 번의 실험을 통해 모델의 성능을 최대화 시켜주는 최적의 하이퍼파라미터 값을 찾아야 합니다. 이렇게 수동적으로 하이퍼파라미터에 다양한 값을 입력하여 최적의 모델의 조건을 찾는 과정을 하이퍼파라미터 튜닝이라고 합니다.

2. **하이퍼파라미터의 종류에는 무엇이 있을까?**

하이퍼파라마터는 모델의 구조나 모델의 복잡도, 혹은 모델이 얼마나 빠르게 학습을 할 수 있는지를 결정할 수 있습니다. 대표적으로 인공 신경망 구조와 관련된 하이퍼파라미터에는 hidden layer 개수, dropout 비율, activation function, weight initialization 등이 있고, 학습 알고리즘과 관련된 하이퍼파라미터에는 epoch, iteration, batch size, optimizer, momentum 등이 있어 각 값들을 조절하여 상황마다 최대 성능을 발휘하는 모델을 설정할 수 있습니다.

이러한 여러 옵션들 중 우리의 베이스 라인 모듈의 Seq2SeqTrainingArguments를 구성하는 하이퍼파라미터들에 대해 알아보고 최적의 값을 찾는 방법을 알아봅시다.

[Seq2SeqTrainingArguments의 주요 하이퍼파라미터]

- **optim**: default=adamw_torch: 모델 훈련 시 최적의 가중치 파라미터를 결정하는 알고리즘을 설정합니다. 최적화 알고리즘 선택 옵션에는 adamw_hf, adamw_torch, adamw_torch_fused, adamw_apex_fused, adamw_anyprecision or adafactor 등이 있습니다.

- **learning_rate**: default=5e-05 (for AdamW): 각 batch/epoch에서 모델의 가중치 파라미터를 조절하는 비율을 설정합니다. 작을수록 학습 속도가 느려지고, 값이 크면 학습 중 예측할 수 없는 동작이 발생할 수 있습니다. 일반적으로 AdamW 최적화 알고리즘을 사용할 경우에는 5e-05를 기본값으로 설정합니다.
- **per_device_train_batch_size**: default=8: 가중치 파라미터가 갱신되기 전 신경망을 통해 전파된 데이터 샘플의 수 입니다. 적절한 배치 크기는 학습 속도와 학습 성능 모두에 영향을 미치기 때문에 적합한 배치 크기를 설정해야 합니다.
- **num_train_epochs**: default=20: 전체 학습 데이터셋을 모델이 몇 번 학습하는지 설정합니다. 모델의 정확도를 높이기 위해서 동일한 훈련 데이터를 사용하여 여러번 학습시키지만 지나치게 epoch를 높이게 되면, 그 학습 데이터셋에 과적합 될 수 있습니다.
- **gradient_accumulation_steps**: default=1: 학습 과정 중 얻어진 gradient를 n-step마다 누적시킨 후 가중치를 업데이트하도록 설정합니다. 모델의 메모리 사용량에 제한이 있거나, 현재 배치 크기를 더 늘릴 수 없는 경우 사용할 수 있습니다.
- **warmup_ratio**: default=0.0: 모델 학습 초반에 learning rate 값을 0에 가까운 값부터 시작하여 최종 값까지 도달할 때 증가시키는 비율을 설정합니다. learning rate를 점진적으로 늘려주므로써 가중치가 급격히 업데이트 되어 과적합이 되는 현상을 줄여주어 안정적인 학습을 할 수 있습니다.
- **lr_scheduler_type**: default=linear: 학습 과정 중 learning rate을 조정하는 방식을 설정합니다. 학습 초기에는 큰 learning rate으로 최적화 시킨 후 최적값에 가까워질수록 learning rate을 줄여갈 수 있습니다. 또는 learning rate을 줄였다 늘리기를 반복하여 가중치를 업데이트하며 최적값을 찾아갈 수 있습니다. 대표적인 scheduler type에는 linear, cosine 등이 있습니다.
- **fp16 / bf16**: default=False: 메모리를 적게 사용하는 데이터 형식으로 설정하여 기본 데이터 형식인 fp32 대비 연산 속도가 빨라 딥러닝 추론 분야에서 많이 사용됩니다. 하지만 정밀도가 낮아서 모델의 정확도가 떨어질 수 있어 일반적으로 모델 학습 시에는 fp32를 사용하고, 추론 단계에서는 fp16을 사용하여 연산 속도를 높이는 경우가 많습니다. bf16는 fp16보단 표현할 수 있는 수의 범위가 더 넓은 데이터 형식이지만 표현 정밀도가 떨어집니다.

3. **대표 하이퍼파라미터 최적화 알고리즘**

1) Grid Search 사용자가 미리 하이퍼파라미터 값들을 정의한 후 그 범위 내에서 알고리즘이 가능한 한 모든 조합을 측정한 후 가장 우수한 성능을 보이는 조합을 최종 하이퍼파라미터 값으로 지정합니다. 전체 탐색 대상 구간을 어떻게 설정할지, 간격은 어떻게 설정할지 등을 결정하는 것은 여전히 사람의 개입이 필요합니다.

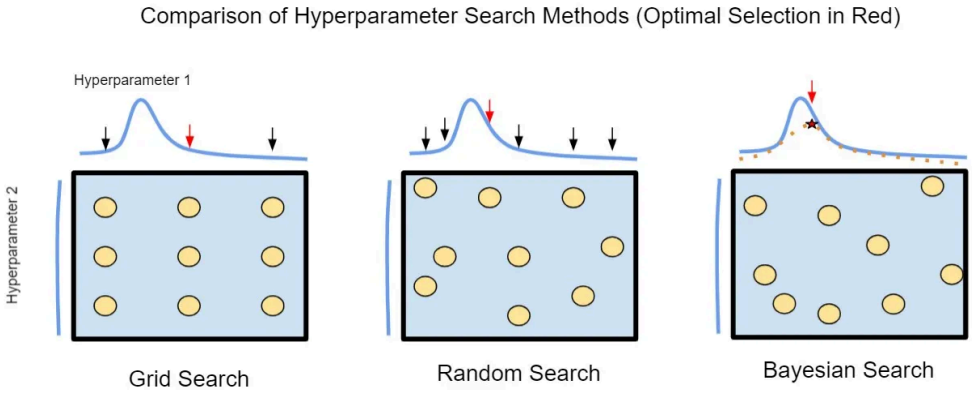
- 장점
  - 범위만 선정하여 여러 값들을 차례대로 입력하면 되기 때문에 직관적이며 쉽고 간단합니다.
  - 각 하이퍼파라미터의 조합들을 독립적으로 모델에 입력할 수 있기 때문에 병렬적으로 실험하여 성능을 산출할 수 있습니다.
- 단점
  - 하이퍼파라미터의 개수가 많은 모델에서는 적합하지 않습니다. 하이퍼파라미터의 개수에 따라서 실험해야 하는 경우의 수가 기하급수적으로 늘어나기 때문에 계산 비용이 많이 듭니다.
  - 미리 정의한 하이퍼파라미터 값들만으로 탐색하기 때문에 그 외에 존재할 수 있는 global optimum을 놓칠 수 있습니다.
  - 단순히 정해진 범위 안에서의 모든 경우의 수를 실험하는 것이기 때문에 이전 실험에서 얻은 결과의 의미가 다음 실험에 반영되지 않아 하이퍼파라미터의 개수가 적은 경우에 사용하는 것이 최적의 하이퍼파라미터 조합을 얻을 수 있는 알고리즘입니다.

2) Random Search 미리 정해진 하이퍼파라미터 값들의 범위에서 무작위로 선택하여 최적의 값을 탐색하는 방식입니다. 차례대로 모든 하이퍼파라미터 조합들을 실험하지 않기 때문에 효율적이고 확장 가능한 최적의 하이퍼파라미터 값의 탐색이 가능합니다. 또한, 사용자가 각 하이퍼파라미터에 대한 확률 분포를 임의로 정의하여 무작위로 선택되는 하이퍼파라미터 조합의 확률을 조절할 수도 있습니다.

- 장점
  - 더 적은 실험 횟수로도 높은 성능을 보여주는 하이퍼파라미터 조합을 찾아냅니다.
  - 각 하이퍼파라미터 조합들은 독립적으로 실험이 가능하기 때문에 병렬적인 실험이 가능합니다.
  - 많은 수의 하이퍼파라미터를 가지는 모델이나 모델 성능에 영향을 끼치는 하이퍼파라미터의 개수가 적은 경우에 용이합니다.
- 단점
  - 진행된 이전 실험 결과들의 의미가 다음 실험에 반영되지 않기 때문에 높은 성능을 내는 하이퍼파라미터 값의 범위를 좁혀나가며 최적화 시키기 힘듭니다.
  - 어떻게 해서 최종적으로 선택된 하이퍼파라미터 조합이 나오게 됐는지 알 수 없기 때문에 최종 결과를 최적의 조합으로 수용하기 힘듭니다.

3) Bayesian Optimization 베이즈 정리(Bayes' theorem)에 기반한 기법으로, 이전 실험을 통해 얻어진 결과를 다음 하이퍼파라미터 값 선택에 반영하여 최적의 하이퍼파라미터 조합을 탐색합니다. 특정 목적 함수를 최대화 시키는 방향으로 하이퍼파라미터 조합과 모델의 성능 사이의 관계를 확률적으로 정의하며 다음 단계에서 평가될 하이퍼파라미터 조합을 선정합니다.

- 장점
  - 많은 수의 하이퍼파라미터를 가지는 모델에 적용하기 용이합니다.
  - 이전 실험에서 선택된 하이퍼파라미터 조합 결과의 정보를 이용하여 다음으로 선택될 하이퍼파라미터 조합을 구성할 수 있어 검색 범위를 점점 의미있게 좁혀가며 좋은 성능을 보여주는 조합을 탐색합니다.
- 단점
  - 복잡한 계산 과정에 의해 구현하기가 까다롭습니다.
  - 이전 결과가 다음 결과에 영향을 끼치므로 동시에 여러 하이퍼파라미터 조합들에 대한 성능을 측정하기 어렵습니다.



4. **Optuna 라이브러리를 사용하여 하이퍼파라미터 튜닝하기**

하이퍼파라미터를 최적화 시키기 위한 다양한 framework들 중 **Optuna** 라이브러리를 사용하여 효율적인 하이퍼파라미터 튜닝 방법을 소개합니다. Optuna 라이브러리는 python 문법과 쉽게 통합될 수 있고, Tensorflow, Pytorch, Sklearn을 포함한 다양한 ML framework와 같이 사용이 가능합니다. Huggingface에서 Tranier 클래스의 API를 통해 Optuna 라이브러리를 사용하여 최적의 파라미터를 찾는 과정을 알아봅니다.

1. **튜닝 파이프라인을 설정**

Huggingface는 다양한 트랜스포머 모델과 학습 스크립트를 제공해주는 모듈입니다. 딥러닝 학습 및 평가와 관련된 작업들을 한번에 해결할 수 있도록 Trainer 클래스에서 API를 제공합니다.

```
# 하이퍼파라미터 튜닝을 하기 위한 Trainer 클래스의 주요 함수
training_args = TrainingArguments(
                optim= 'adamw_torch'
                learning_rate=1e-5,
                per_device_train_batch_size=16,
                num_train_epochs=20,
                gradient_accumulation_steps=1,
                warmup_ratio=0.1,
                lr _scheduler_type= 'cosine' ,
                fp16=True,
                bf16=True
)
```

- TrainingArguments 함수 Trainer를 정의할 때, Trianer가 학습 및 평가에 사용할 모든 하이퍼파라미터(hyperparameters)를 포함하는 TrainingArguments 클래스를 정의해줍니다. 이를 통해, Optimizer의 종류, learning rate, epoch 수, scheduler, half precision 사용여부 등을 지정할 수 있으며, 모든 파라미터의 목록은 여기에서 확인할 수 있습니다.

```
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = predictions.argmax(aixs=-1)
    return metric.compute(predictions = predictions, references = labels)

def model_init(trial):
    return MODEL.from_pretrained()

trainer = Trainer(
                args = training_args,
                compute_metrics = compute_metrics
                model_init = model_init
)
```

- compute_metrics 함수 사용자가 정의한 평가 방법 혹은 평가 척도에 대한 compute_metrics 함수를 Trainer에 지정하여 학습 시 하이퍼파라미터 조합들의 성능을 비교할 수 있도록 설정합니다.
- model_init 함수 최적의 하이퍼파라미터 조합을 탐색하기 위해 사용자가 정의한 기본 모델을 설정해줍니다.

2. **Optuna 라이브러리를 설치**

```
! pip install optuna
```

3. **Trainer의 hyperparameter_search API 모듈 설정**

Optuna 라이브러리를 사용하기 위해 필요한 설정과 탐색할 각 하이퍼파라미터 값들의 범위를 정해줍니다. 가장 좋은 파라미터 조합을 best_trials에 저장합니다.

```
import optuna

def optuna_hp_space(trial):
    return {
         "optim" : trial.suggest_categorical(name = 'optimizer',
         choices=[ 'sgd' , 'adamw_torch', 'rmsprop']) ,
         "learning_rate" : trial.suggest_loguniform(
                  name= 'learning_rate' , low=1e-04, high=5e-03),
         "per_device_train_batch_size": trial.suggest_categorical(
                  name = 'batch_size', choices = [16, 32]),
         "num_train_epochs": trial.suggest_int(name= 'epoch', low = 10,
```

```
                high=50, step=10),
        "gradient_accumulation_steps" : trial.suggest_int(name = 'epoch',
                low=1, high=5, step=1),
        "warmup_ratio" : trial.suggest_categorical(name= 'warm_ratio',
               choices=[0, 0.001]),
        "lr_scheduler_type" : trial.suggest_categorical(
                name= 'lr_schedule',
                choices=['linear' , 'cosine']),
        "fp16": trial.suggest_categorical(name= 'fp16' ,
                choices=[True, False]),
        "bf16": trial.suggest_categorical(name='bf16' ,
                choices=[True, False])
}

best_trials = trainer.hyperparameter_search(
        direction "maximize"
        backend = "optuna" ,
        hp_space = optunua_hp_space,
        n_trials = 20
)
```

5. **토론**

상기한 하이퍼파라미터 튜닝 방법을 활용하여 최적의 성능을 보여주는 하이퍼파라미터 조합을 찾아볼 수 있습니다. 대화 요약 모델의 성능을 최대한으로 높이기 위해, 다양한 실험을 통해 여러분들만의 하이퍼파라미터 조합을 찾아봅시다.

댓글