Hyperparameter optimization for transformer models using practical metrics

By

Pramod Gupta (pramodgupta@microsoft.com)

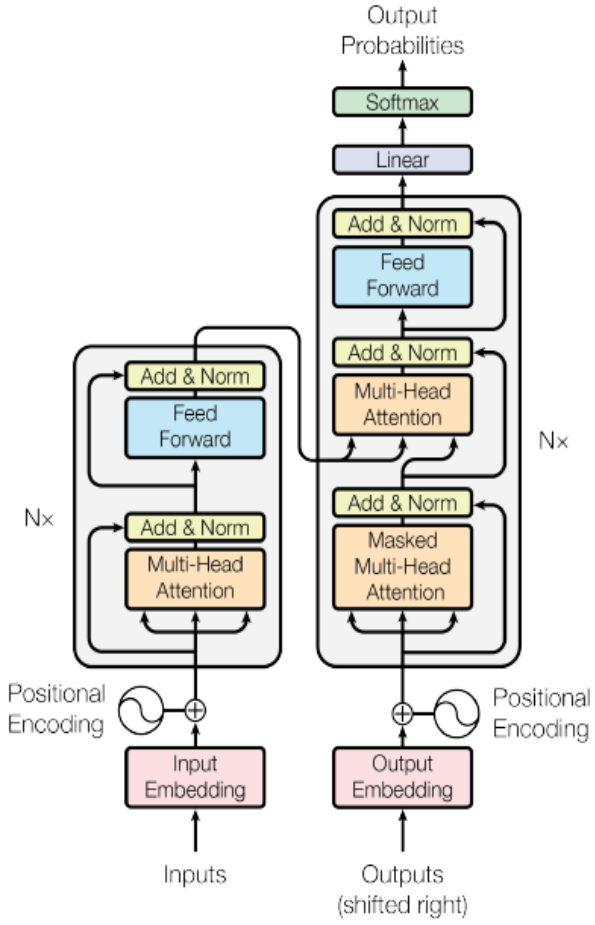
James Wither (jawithe@microsoft.com)

Agenda

- Introduction to Transformer Model
- What is Transfer Learning
- Hyperparameter Optimization
- Hyperparameter Strategies
- Tool & API for Hyperparameter Optimization
- Huggingface Transformer Training API
- Some Code and Results

Transformer Model

- Transformers are Attention based model
- It is primarily self-supervised model trained on huge data
- Transformers model does not suffer with gradient vanishing problems
- It is easily parallelized
- Transformers model has many components in its architecture
- Major architectural components in Transformer models
  - Multi Head Attention
  - Feed Forward Layers
  - Linear Layers
  - Embedding Dimension
- Due to huge domain/language corpus involved in Transformers enables it to learn all syntactic and semantic feature of the language
- Transformers make Transfer Learning very popular in NLP because of this property

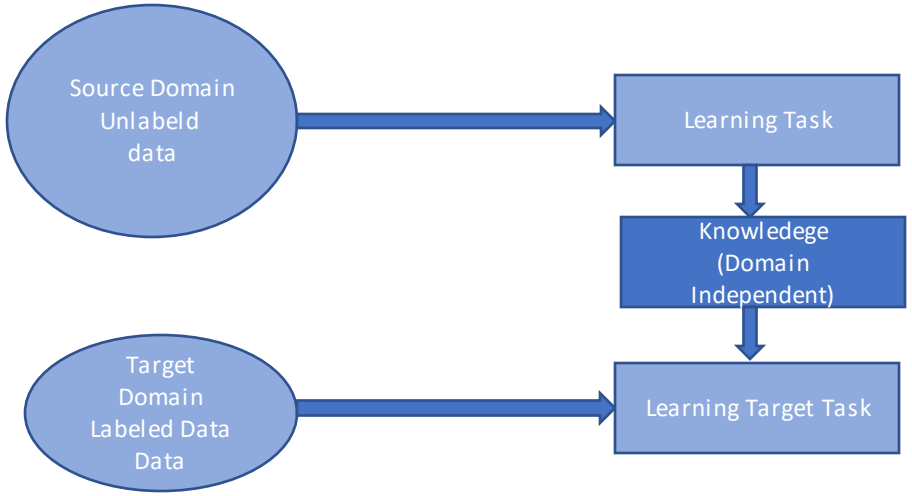


https://www.aclweb.org/anthology/N19-1423/

Transfer Learning

- Transfer learning aims to leverage the knowledge
- Learned from resource-rich domain/task to help learning a task where we don't have much labeled data
- The resource rich domain is called Source domain and resource scarce task is called Target domain

- The layers of Transfers model are considered as feature extractor
- Learned weights of source domain share for target task
- Shared weights are fine-tuned on the target labeled data to get adapted accordingly
- Due to lack of huge labeled data we need to be very careful during fine tuning of model according to downstream tasks
- Fine Tuning of Transformers involves many **hyperparameters** such as :
  - Batch Size
  - Learning Rate
  - Warmup
  - Weight Decay



Why Hyperparameter Optimization

- **Transformer Models can be used two ways for solving any business problem:**
  a) Trained Transformer model on own data
     I. Because Pretrained models are huge and require huge storage
     II. Inference time is large
     III. Many time we don't need all deep knowledge of Language which big Transformer models learned
     IV. Due to not very semantic downstream tasks we want to avoid large pre trained models
     V. Transfer model and downstream tasks are from very orthogonal domain

  b) Fine Tuned pre-trained Transformer Model for down-stream Task with small set of labeled data
     i. Bigger pre trained model and downstream task are from similar domain
     ii. Non availability of huge Labeled data even unlabeled domain/language resource is not available

Hyperparam	$\text{RoBERTa}_{\text{LARGE}}$	$\text{RoBERTa}_{\text{BASE}}$
Number of Layers	24	12
Hidden size	1024	768
FFN inner hidden size	4096	3072
Attention heads	16	12
Attention head size	64	64
Dropout	0.1	0.1
Attention Dropout	0.1	0.1
Warmup Steps	30k	24k
Peak Learning Rate	4e-4	6e-4
Batch Size	8k	8k
Weight Decay	0.01	0.01
Max Steps	500k	500k
Learning Rate Decay	Linear	Linear
Adam ϵ	1e-6	1e-6
Adam β_1	0.9	0.9
Adam β_2	0.98	0.98
Gradient Clipping	0.0	0.0

Table 9: Hyperparameters for pretraining $\text{RoBERTa}_{\text{LARGE}}$ and $\text{RoBERTa}_{\text{BASE}}$.

- In both the scenarios there are many hyperparameters with large range of possible values
- Hence, identifying optimal hyperparameters for given tasks from such a large space is very **time-consuming** and **resource intensive**

HPO Strategies in Transformers

	Random	Adaptive	Evolutionary
Sequential	Grid/Random Search	Bayesian Optimization	Genetic Algorithm
Parallel	Asynchronous Successive Halving Algorithm (ASHA)	Bayesian Optimization With Hyperband	Population Based Training

Pros Cons of HPO Strategies

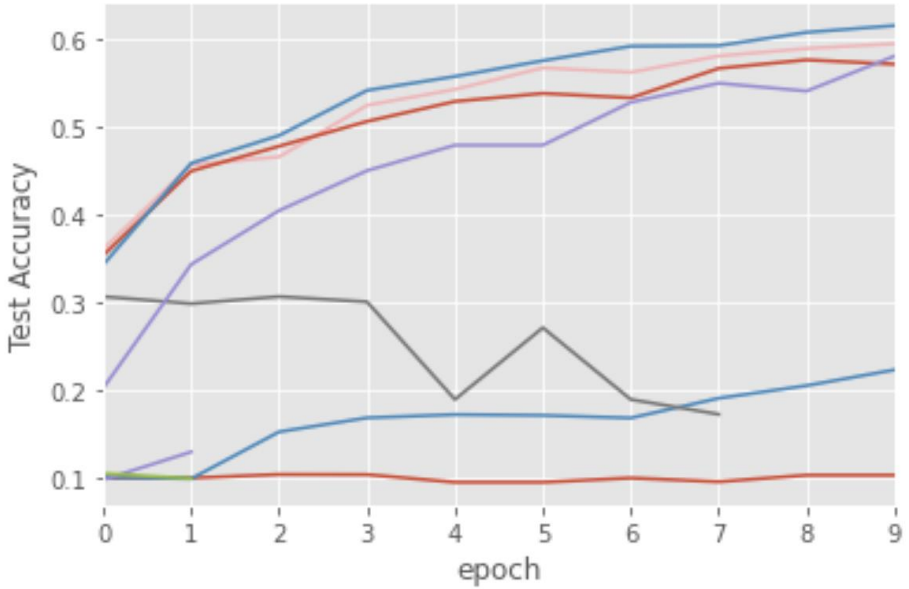
- **Grid Search**
  - Highly inefficient
  - Not good for large search space
  - Easy to use
  - Easy to parallelize
- **Random Search**
  - Good for large search space
  - Better than Grid search
  - Easy to parallelize
  - Stochastic approach
- **Bayesian Optimization**
  - Uses prior information with random Serach
  - Initial run of BO is very similar to Random Search
  - Not easy to parallelize
- **BO + Eary Stopping (ASHA)**
  - To avoid it BO combine with ASHA
  - ASHA helps to avoid running early trials of runs
- **Population Based Training**
  - Proposed by DeepMind
  - Its uses evolutionary strategy for searching Hyperparameter in Search Space
  - https://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf

Hyperparameter Optimization API

- There are many HPO API available for Deep Learning such as:
  - Keras Tuner
  - Ray Tune
  - Optuna
  - HOpt
  - Microsoft NNI

HPO with Ray Tune for Deep Learning

- **Objective :** We try to get optimal configuration for classifying images into 10 classed
- **Dataset :** CIFAR10




```
#ray tune format for defining search space

config = {
    "l1": tune.sample_from(lambda _: 2 ** np.random.randint(2, 9)),
    "l2": tune.sample_from(lambda _: 2 ** np.random.randint(2, 9)),
    "lr": tune.loguniform(1e-4, 1e-1),
    "batch_size": tune.choice([2, 4, 8, 16])
}
```

HP Search Space

```
#defining scheduler to be used for selecting optimal hyperparmeter configiration
#for next run, which helps to obtain optimal configuration for Hyperparmeter efficiently

scheduler = ASHAScheduler(
    max_t=max_num_epochs,
    grace_period=1,
    reduction_factor=2)
```

Scheduler

```
#running the hyperparameter optimzation process by using Ray Tune as backend for hyperparameter optimization
result = tune.run(
    tune.with_parameters(train_cifar, data_dir=data_dir),
    resources_per_trial={"cpu": 2, "gpu": gpus_per_trial},
    config=config,
    metric="loss",
    mode="min",
    num_samples=num_samples,
    scheduler=scheduler
)
```

Optimization

- Above Graph shows there are many epochs run for HPO
- But due to ASHA Scheduler many epochs got terminated early on.
- Because these epochs are not showing potential to get optimal performance

Huggingface Transformer Training API

- There are many API for HPO for Deep Learning but there is no explicit API for transformer training
- Huggingface provides Transformer training API which is **Trainer API**
- Training API **simplifies** Transformer Training and Fine Tuning very easy
- Major building block for Training API:
  - Dataset
  - Compute_metrics
  - TrainingArguments
  - Trainer

1

```python
# Create torch dataset
class Dataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        if self.labels:
            item["labels"] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.encodings["input_ids"])
```

```python
train_dataset = Dataset(X_train_tokenized, y_train)
val_dataset = Dataset(X_val_tokenized, y_val)
```

2

```python
# Define Trainer parameters
def compute_metrics(p):
    pred, labels = p
    pred = np.argmax(pred, axis=1)

    accuracy = accuracy_score(y_true=labels, y_pred=pred)
    recall = recall_score(y_true=labels, y_pred=pred)
    precision = precision_score(y_true=labels, y_pred=pred)
    f1 = f1_score(y_true=labels, y_pred=pred)

    return {"accuracy": accuracy,
            "precision": precision,
            "recall": recall,
            "f1": f1}
```

3

```python
# Define Training Arguments
args = TrainingArguments(
    output_dir="imdb_small_model",
    evaluation_strategy="steps",
    eval_steps=400,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=2,
    save_steps=3000,
    seed=0,
    load_best_model_at_end=True,
)
```

4

```python
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics,
    #callbacks=[EarlyStoppingCallback(early_stopping_patience=3)],
)
```

HPO with Huggingface Trainer API

- Trainer API provides functionalities to provide optimizing hyperparameters in Transformer
- We can optimizing any parameters/hyperparameters available in TrainingArgumnents directly
- Indirectly one can optimize any hyperparameter by subclassing TrainingArguments if its not there in TraningArguments explicitly
- API uses **Ray Tune** or **Optuna** as hyperparameter optimization backend
- All SOTA HPO techniques are available in the API

Steps Involved in HPO with Huggingface API

1. Defining Dataset (Training and Validation)
2. Model definition
3. Defining Metrics
4. Defining Objective (to be optimized)
5. Declaring Hyperparameter Search Space
6. Declaring TrainingArguments
7. Defining Trainer
8. Optimization

Lab Repository and Presentation

https://github.com/pramod05/Transformer_HPO

References

- https://huggingface.co/transformers/
- https://huggingface.co/transformers/main_classes/trainer.html