Report
TANDA Approach for QA System Enhancement

Lyle He

Contents

1 Introduction

Explore TANDA (Transfer And Adapt) methodology to improve Question-Answering (QA) systems using pre-trained Transformer models, focusing on sequential fine-tuning techniques.

2 Introduction and Theory

TANDA (Transfer And Adapt) introduces a novel approach for fine-tuning pre-trained transformer models for NLP tasks focusing on enhancing model performance in a specific domian.

2.1 Novelty and Key Principles

The TANDA methodology is a two-step fine-tuning process. It transfer and adapt pre-trained models for answer question selection. First, the model is fine-tuned on a large-scale and high-quality dataset, such as the AskUbuntu Stack Exchange (ASNQ) dataset. The large-scale dataset in this step should be also related to the target domain which gives the pre-trained model a more focused context.

In the second step, the model is adapted to a specific domain, such as WikiQA or TREC-QA, as well as industrial datasets derived from questions sampled from interactions with the Alexa virtual assistant.

2.2 Rationale Behind Sequential Fine-Tuning

This methodology addresses data scarcity and model instability in domain-specific tasks with a significant performance improvement (nearly 10% MAP scores improvement).

Stability and Robustness: If we directly fine-tune the pre-trained model on a small domain-specific dataset, the model is instability with high variance. The intermediate transfer step can can anchor the model to a related domain.

Robustness to Noise: The transfer step makes the second step(adaptation) more robust to noise in the domain-specific dataset which means we can use nosiy data effectively and makes the model useful in real-world applications.

Data inadequate: Basically we lack of large-scale and high-quality domain-specific datasets. The TANDA methodology can use a large-scale general dataset in transfer step reducing the reliance on large domain-specific datasets.

Efficiency and Modularity: We can fine-tune the model to different domain-specific datasets from the same transfer step. This can save time and computational resources.

2.3 How Transformers' architecture benefits the TANDA approach?

There are several key benefits of using the Transformer architecture in the TANDA approach.

Scalability and Efficiency: The Transformer is orignally designed to process data in parallel rather than sequentially, which makes it effienct for large-scale datasets. This is important for the transfer step.

Layered Attention Mechanism: The multi-head attention mechanism of the Transformer allows the model to understand the difference between the transfer and adaptation steps. The model can learn to focus on different aspects of the input data in each step.

Pretrained Models: The Transformer models are usually pre-trained on large-scale datasets lead to a rich representations ability, which privides a good starting point for the TANDA approach.

Flexible to Varied Input: The Transformer architecture is flexible to different input length and types, which make it possible for the TANDA approach.

Stability in Fine-tuning: The Transformer architecture is inherently stable over training epochs and achieving consistent improvements.

3 Preparation and Dataset Understanding

3.1 Answer-Sentence Natural Questions (ASNQ) Dataset

ASNQ is a accurate, general and large AS2 corpus used to validate the benefits of TANDA and derived from the Google Natural Questions (NQ) dataset (Kwiatkowski et al. 2019). In NQ dataset, each question is related to a Wikipedia page, a long paragraph(long_answer) containing the answer, and each long_answer may contain phrases annotated as short_answers.

In ASNQ dataset, for each question, the positive candidate answers are those sentences that occur in the long_answer paragraphs in NQ and contain annotated short answers. And the negative answers contain three types of sentences: (1). In the long answer but do not contain the annotated short answers. (2). Not in the long answer but contain the short answer string. (3). Neither in the long answer nor contain the short answer.

The negative answers are important to the robustness of the model in identifying the best answer among the similar but incorrect ones.

Here are some statistics of the ASNQ dataset (Garg S et al. 2020):

Label	S ∈ LA	SA ∈ S	# Train	# Dev
1	No	No	19,446,120	870,404
2	No	Yes	428,122	25,814
3	Yes	No	442,140	29,558
4	Yes	Yes	61,186	4,286

Table 1: Label description for ASNQ. Here S, LA, SA refer to answer sentence, long answer passage and short answer phrase respectively.

Figure 1: Statistics of the ASNQ dataset (Only label 4 is positive)

Finally, ASNQ is larger than most public AS2 dataset in 2020 containing 57,242 different questions in the training set and 2,672 different questions in the dev. set.

3.1.1 Lets conduct an Exploratory Data Analysis (EDA) on the ASNQ dataset

I use the following code to load the dataset and conduct the EDA:

```
from datasets import load_dataset
dataset = load_dataset("asnq")
```

Here is the details of the dataset:

```
DatasetDict({
    train: Dataset({
        features: ['question', 'sentence', 'label', 'sentence_in_long_answer',
'short_answer_in_sentence'],
        num_rows: 20377568
    })
    validation: Dataset({
        features: ['question', 'sentence', 'label', 'sentence_in_long_answer',
'short_answer_in_sentence'],
        num_rows: 930062
    })
})
```

And here are some examples of the dataset:

	question	sentence	label	sentence_in_long_answer	short_answer_in_sentence
0	what is the use of fn key in mac	It is typically found on laptops due to their ...	0	False	False
1	when did the ipod 7th generation come out	Cupertino , California : Apple .	0	False	False
2	who dies in season 6 once upon a time	^ Jump up to : Abrams , Natalie (August 5 , 2...	0	False	False
3	who was the first avatar ever in the last airb...	Jeremy Zuckerman and Benjamin Wynn composed th...	0	False	False
4	how many games do they need to win to win the ...	p. 21 .	0	False	False

Figure 2: Examples of the ASNQ trainning dataset

And let's take a look at types of the questions in the dataset (There are 1947 types of questions in the dataset, and the most frequent question type is "who" with 5504958 occurrences):

```
who           5504958
when          5401938
what          2870313
where         2399202
how           1353214
               ...
merocrine          12
ep-13              12
mmts               11
fort               10
organic            10
Name: question_type, Length: 1947, dtype: int64
```

And here are the distribution of the length of the questions and the sentences in the dataset:

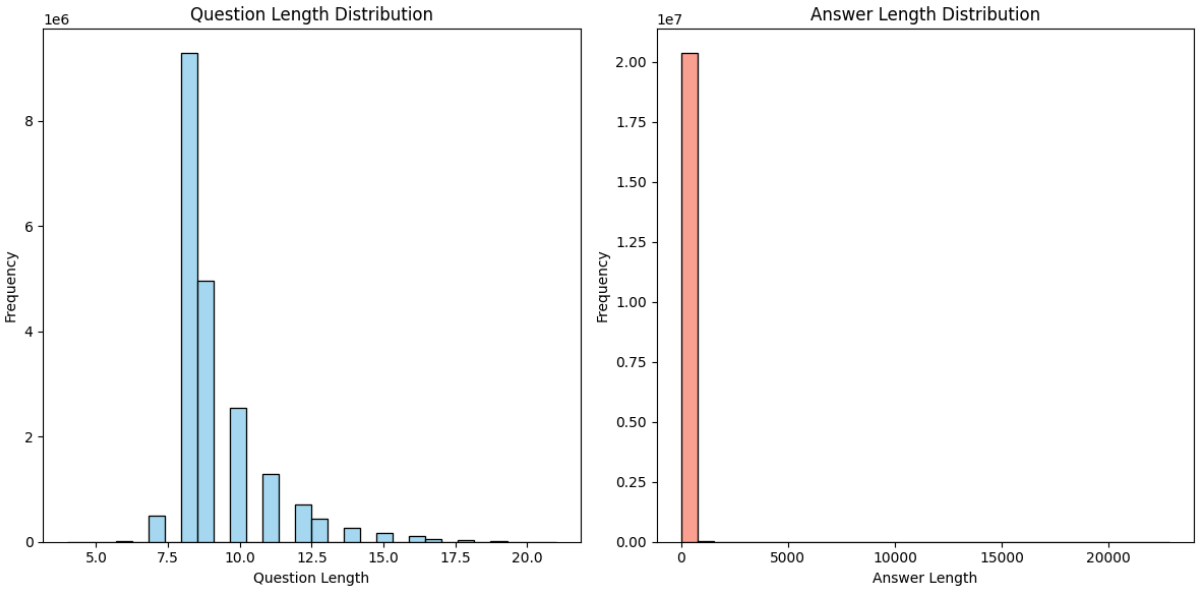

Figure 3: Examples of the ASNQ trainning dataset

3.2 WikiQA or TREC-QA for domain-specific adaptation?

I choose WikiQA for domain-specific adaptation with question and sentence pairs. WikiQA rely on Wikipedia might offer a broader range of topics.

Here are the overall details of the WikiQA dataset:

```
DatasetDict({
    test: Dataset({
        features: ['question_id', 'question', 'document_title', 'answer', 'label'],
        num_rows: 6165
    })
    validation: Dataset({
        features: ['question_id', 'question', 'document_title', 'answer', 'label'],
        num_rows: 2733
    })
    train: Dataset({
        features: ['question_id', 'question', 'document_title', 'answer', 'label'],
        num_rows: 20360
    })
})
```

Here are some examples of the dataset in trainning set:

	question_id	question	document_title	answer	label
0	Q1	how are glacier caves formed?	Glacier cave	A partly submerged glacier cave on Perito More...	0
1	Q1	how are glacier caves formed?	Glacier cave	The ice facade is approximately 60 m high	0
2	Q1	how are glacier caves formed?	Glacier cave	Ice formations in the Titlis glacier cave	0
3	Q1	how are glacier caves formed?	Glacier cave	A glacier cave is a cave formed within the ice...	1
4	Q1	how are glacier caves formed?	Glacier cave	Glacier caves are often called ice caves , but...	0

Figure 4: Examples of the WikiQA trainning dataset

Here are some statistics of the WikiQA dataset:

```
       question_length  answer_length
count     20360.000000   20360.000000
mean          6.885658      22.729028
std           2.436096      11.535818
min           2.000000       1.000000
25%           5.000000      15.000000
50%           6.000000      21.000000
75%           8.000000      29.000000
max          21.000000     166.000000
0    0.948919
1    0.051081
Name: label, dtype: float64

# Missing values
question_id        0
question           0
document_title     0
answer             0
label              0
question_length    0
answer_length      0
dtype: int64
# Duplicates
Duplicates: 2
```

Here are the distribution of the length of the questions and the sentences and the label types in the dataset:

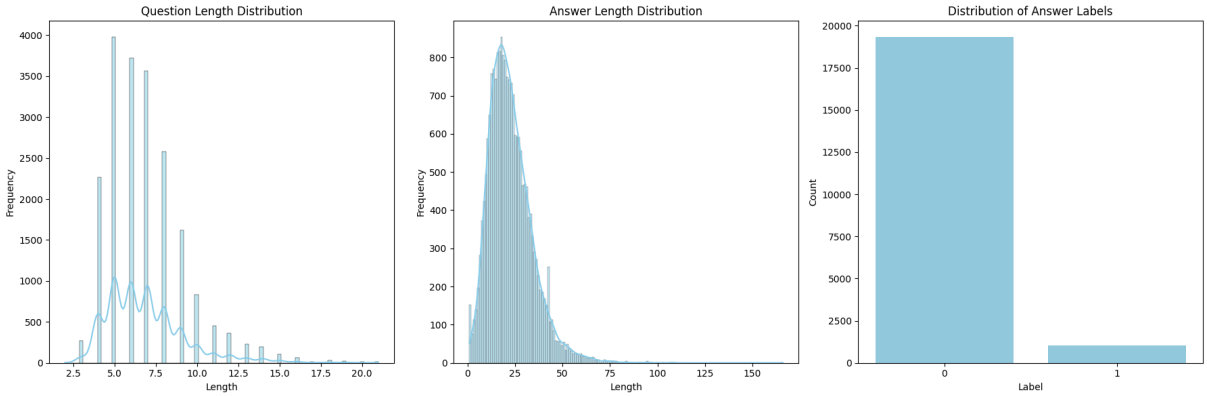

Figure 5: Distribution of the length of the questions and the sentences in the dataset

3.3 Challenges

As we can see from the EDA, we have some challenges in the dataset: class imbalance, varying lengths of text, or potential preprocessing steps needed.

4 Model Implementation

4.1 Baseline Transformer Model

I use the transformer library from Hugging Face to implement the baseline model. The exact model I chose is the **`bert-base-cased`** model. This model is a pre-trained BERT (Bidirectional Encoder Representations from Transformers) model that is case-sensitive, distinguishing between lowercase and uppercase words.

The `transformers` library provides a high-level interface for fine-tuning pre-trained BERT models. I use the `AutoModelForSequenceClassification` class to fine-tune the model. `AutoModel` series of classes can automatically load the correct model architecture and weights based on the model name or path provided.

Here is the code to implement the baseline model:

```
from transformers import AutoModelForSequenceClassification

base_model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased",
                                                                 num_labels=2)
```

And then I will train the model on the WikiQA dataset directly and evaluate the performance.

```
from datasets import load_dataset
dataset_wiki_qa = load_dataset("wiki_qa")

# Preprocess the dataset, the reults are shown as below
After removing the unuseful columns:
DatasetDict({
    test: Dataset({
        features: ['labels', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 6165
    })
    validation: Dataset({
        features: ['labels', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 2733
    })
    train: Dataset({
        features: ['labels', 'input_ids', 'token_type_ids', 'attention_mask'],
```

```
        num_rows: 20360
    })
})
```

4.2 TANDA Two-Step Fine-Tuning Process

I use the TANDA approach to fine-tune the pre-trained BERT model on the ASNQ dataset and then adapt the model to the WikiQA dataset.

The model is first fine-tuned on the ASNQ dataset and then trained on the WikiQA dataset.

Here is the code to implement the TANDA approach:

```
# Use the some model type described above
from transformers import AutoModelForSequenceClassification
# Load the pre-trained model
model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased",
                                                           num_labels=2)

# Fine-tune the model on the ASNQ dataset
DatasetDict({
    train: Dataset({
        features: ['labels', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 20377568
    })
    validation: Dataset({
        features: ['labels', 'input_ids', 'token_type_ids', 'attention_mask'],
        num_rows: 930062
    })
})

# Adapt the model to the WikiQA dataset (the same as the baseline model)
```

4.3 Modifications made to the original Transformer architecture

(1) The target is a binary classification task, so I change the number of labels to 2.

(2) The modification I cannot make is the model internal architecture, such as the number of layers, the number of heads, the hidden size of the transformer, etc, except the output layer.

(3) I can modify the learning rate, the batch size, the number of epochs, the optimizer, the loss function.

(4) I can add some layer to the output side of the model, such as linear layer, dropout layer, etc. But if I just add a linear layer, the impact is not significant.

5 Experimental Setup and Evaluation

5.1 Experimental Setup and Evaluation both the baseline and TANDA-enhanced models

I use the following hyperparameters for the training and tunning of the models:

- Learning rate: 2e-4, 2e-5, 2e-6
- batch size: 16
- number of epochs: 3
- optimizer: AdamW
- loss function: CrossEntropyLoss

Evaluation Metrics:

- evaluation metrics: accuracy, F1 score, precision, recall

optimization strategy: I tested different learning rates, batch sizes, and number of epochs to find the best hyperparameters for the models. I will use the F1 score as the main metric to evaluate the models to decide the best hyperparameters.

5.2 Training Implementation

I use the Trainer class from the `transformers` library to train the models. The Trainer class provides a high-level interface for training and evaluating models. It handles the training loop, evaluation loop, and logging of the training and evaluation metrics.

The details of the training implementation are shown in the ipython notebook.

5.3 Results and Analysis

5.3.1 Baseline model

For the baseline model, I use the hyperparameters recommanded by the authors of the TANDA paper. The results are shown as below:

- Here is the experimental configuration:

```
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=2e-6,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    num_train_epochs=3,
    do_train = True,
    do_eval = True,
)
```

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.325800	0.190102	0.948774	0.000000	0.000000	0.000000
2	0.195100	0.176406	0.948774	0.000000	0.000000	0.000000
3	0.180800	0.172164	0.948774	0.000000	0.000000	0.000000

Figure 6: Training and validation loss of the baseline model

```
Test Evaluation Results: {'eval_loss': 0.1646772027015686, 'eval_accuracy':
0.9524736415247365, 'eval_f1': 0.0, 'eval_precision': 0.0, 'eval_recall': 0.0,
'eval_runtime': 13.5489, 'eval_samples_per_second': 455.017, 'eval_steps_per_second':
14.245, 'epoch': 3.0}
```

- The F1 score, precision, recall are all 0 for the baseline model, and the TANDA-enhanced model. I think there must be something wrong.

Then I balanced the test dataset and re-evaluate the model:

```
Test Evaluation Results: {'eval_loss': 1.42829430103302, 'eval_accuracy': 0.5,
'eval_f1': 0.0, 'eval_precision': 0.0, 'eval_recall': 0.0, 'eval_runtime': 1.3763,
'eval_samples_per_second': 425.784, 'eval_steps_per_second': 13.805, 'epoch': 3.0}
```

The result is still not good. I think may be there is something wrong with the model or the dataset.

So I change the configuration:

```
training_args = TrainingArguments(
    output_dir="base_model",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True
)
```

The results are shown as below:

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.179400	0.155982	0.949872	0.941206	0.949872	0.944274
2	0.109300	0.187161	0.954995	0.947207	0.954995	0.949240
3	0.065700	0.231098	0.949872	0.944395	0.949872	0.946682

Figure 7: Training and validation loss of the baseline model

Test on the test dataset of the WikiQA datase (imbalance):

```
{'eval_loss': 0.14157655835151672,
'eval_accuracy': 0.9544201135442011,
'eval_precision': 0.9444462830496804,
'eval_recall': 0.9544201135442011,
'eval_f1': 0.9472047710668405,
'eval_runtime': 14.2146,
'eval_samples_per_second': 433.709,
'eval_steps_per_second': 27.155,
'epoch': 3.0}
```

Then I balanced the test dataset and re-evaluate the model:

```
{'eval_loss': 1.1616528034210205,
'eval_accuracy': 0.6313993174061433,
'eval_precision': 0.7701980885769719,
'eval_recall': 0.6313993174061433,
'eval_f1': 0.5770875654870096,
'eval_runtime': 1.6208,
'eval_samples_per_second': 361.561,
'eval_steps_per_second': 22.829,
'epoch': 3.0}
```

5.3.2 TANDA-enhanced model

For the TANDA-enhanced model, I tested some different hyperparameters.

- Learning Rates: I tested different learning rates for the models and found that the learning rate of 2e-5 in the transfer step is the best for both the baseline and TANDA-enhanced models.
- Batch Sizes: I tested different batch sizes for the models and found that the batch size of 32 is the best for TANDA-enhanced models.
- Number of Epochs: I tested different numbers of epochs for the models and found that the number of epochs of 3 is the best TANDA-enhanced models.

The results are shown as below:

- Since the ASNQ dataset is too large to fit into memory, I only used 10000 samples to train and 3000 samples to evaluate the model. But for the WikiQA dataset, I used the whole dataset.

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.031200	0.028590	0.995667	0.991352	0.995667	0.993505
2	0.016900	0.032811	0.995667	0.991352	0.995667	0.993505
3	0.006500	0.041426	0.992333	0.992618	0.992333	0.992475

Figure 8: Training and validation loss of the Transfer step of the TANDA-enhanced model

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.177000	0.153120	0.951336	0.941708	0.951336	0.944563
2	0.100800	0.195248	0.953897	0.946298	0.953897	0.948626
3	0.057300	0.238937	0.952799	0.947180	0.952799	0.949373

Figure 9: Training and validation loss of the Adaptation step of the TANDA-enhanced model

Test on the test dataset of the WikiQA datase (imbalance):


```
{'eval_loss': 0.13979241251945496,
'eval_accuracy': 0.9547445255474453,
'eval_precision': 0.9446940557332925,
'eval_recall': 0.9547445255474453,
'eval_f1': 0.9473128362546126,
'eval_runtime': 15.622,
'eval_samples_per_second': 394.637,
'eval_steps_per_second': 24.709,
'epoch': 3.0}
```


Then I balanced the test dataset and re-evaluate the model:


```
{'eval_loss': 1.1579923629760742,
'eval_accuracy': 0.6313993174061433,
'eval_precision': 0.7757731328688424,
'eval_recall': 0.6313993174061433,
'eval_f1': 0.5758916006594025,
'eval_runtime': 1.5827,
'eval_samples_per_second': 370.242,
'eval_steps_per_second': 23.377,
'epoch': 3.0}
```


5.4 Analyze the performance impact of the TANDA approach compared to the baseline model

As we can see from the results, the TANDA-enhanced model has a better performance than the baseline model. The TANDA-enhanced model has a higher accuracy, F1 score, precision, and recall than the baseline model.

So as stated in the TANDA paper, the TANDA approach can improve the performance.

6 Discussion and Conclusion

Challenges Encountered:

- The dataset is highly imbalanced, which makes it difficult to evaluate the models using the F1 score, precision, and recall.
- The dataset contains varying lengths of text, which requires preprocessing steps to handle.
- The dataset is too large to fit into memory, which requires a lot of computational resources to train the models.

Areas for Improvement:
- We could build a more balanced dataset to train and evaluate the models in the future, since it is too imbalanced.
- We could using a different model architecture to test.

Future Research Directions:
- We could apply the TANDA methodology to any other NLP tasks, such as text classification, named entity recognition, etc. I think the TANDA methodology is a general approach for fine-tuning pre-trained transformer models.

7 References

- Kwiatkowski, T.; Palomaki, J.; Redfield, O.; Collins, M.; Parikh, A.; Alberti, C.; Epstein, D.; Polosukhin, I.; Kelcey, M.; Devlin, J.; Lee, K.; Toutanova, K. N.; Jones, L.; Chang, M.-W.; Dai, A.; Uszkoreit, J.; Le, Q.; and Petrov, S. 2019. Natural questions: a benchmark for question answering re- search. TACL.
- Garg S, Vu T, Moschitti A. Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection[C]//Proceedings of the AAAI conference on artificial intelligence. 2020, 34(05): 7780-7788.