به نام خدا

گزارش Soft Prompt Tuning

غزل زمانی‌نژاد

در ابتدا کتابخانه‌های مورد نیاز را نصب و سپس import می‌کنیم. مقادیر ثابت را تعریف کرده و مدل، توکنایزر و دیتاست را لود می‌کنیم.

برای تبدیل برچسب داده‌ها به شماره کلاس و بالعکس، کلاس Labels را تعریف می‌کنیم:

```
class Labels:
    def __init__(self, labels):
        self.labels = labels
        self.label_to_id_dict = {l:i for (i, l) in enumerate(self.labels)}
        self.id_to_label_dict = {i:l for (i, l) in enumerate(self.labels)}

    def convert_labels_to_ids(self, labels):
        return [self.label_to_id_dict.get(l, 2) for l in labels]

    def convert_ids_to_labels(self, ids):
        return [self.id_to_label_dict.get(i, 'other') for i in ids]

imdb_labels = Labels(['negative', 'positive'])
```

سپس با استفاده از تابع preprocess_function، داده‌ها را پیش‌پردازش می‌کنیم. برای این کار به تعداد توکن‌های سافت پرامپت، به ابتدای متن ورودی پد اضافه و سپس آن را توکنایز می‌کنیم. همچنین پس از تبدیل برچسب کلاس‌ها به حالت رشته، آن‌ها را نیز توکنایز می‌کنیم. بجز داده آزمون، از ۲۰٪ داده آموزش به عنوان داده ارزیابی استفاده می‌کنیم.

```
def split_data(data, train_size, seed=42, shuffle=True):
    if shuffle:
        # Shuffle dataset
        data = data.shuffle(seed)
    data_idx = int(len(data) * train_size)
    train_dataset = data.select(range(data_idx))
    eval_dataset = data.select(range(data_idx, len(data)))
    return train_dataset, eval_dataset

def preprocess_function(example):
    # Prepend padding to input with the length same as soft prompts
    pad_token = tokenizer.pad_token
    padded = [pad_token * N_SOFT_PROMPT + t for t in example['text']]
    # Tokenize input text
    model_input = tokenizer(padded, max_length=256, truncation=True)
    # Convert labels to their string format
    label_str = imdb_labels.convert_ids_to_labels(example['label'])
    # Tokenize converted label
    label_tokenized = tokenizer(label_str, max_length=10, truncation=True)
    model_input['labels'] = label_tokenized.input_ids
```

```
    return model_input

# Preprocess data
train_dataset = dataset['train'].map(preprocess_function, batched=True)
test_dataset = dataset['test'].map(preprocess_function, batched=True)

# Keep 20% of data as validation set
train_size = 0.8
train_dataset, eval_dataset = split_data(train_dataset, train_size)

train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])
eval_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])
test_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'labels'])
```

با استفاده از data loader، داده را در batch های ۳۲ عددی تقسیم می‌کنیم.

```
col_fn = DataCollatorForSeq2Seq(
    tokenizer, return_tensors='pt', padding='longest',
)

train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    collate_fn=col_fn,
    shuffle=True
)
eval_loader = torch.utils.data.DataLoader(
    eval_dataset,
    batch_size=BATCH_SIZE,
    collate_fn=col_fn,
    shuffle=False
)
test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=BATCH_SIZE,
    collate_fn=col_fn,
    shuffle=False
)
```

اکنون طبق مقاله یک لایه سافت پرامپت تعریف کرده و آن را با لایه امبدینگ مدل T5 جابجا می‌کنیم. برای این کار کلاس CustomEmbeddingLayer را تعریف می‌کنیم. در constructor آن لایه سافت پرامپت را مقداردهی می‌کنیم. در اینجا

تعداد توکن‌های سافت پرامپت را ۱۰ در نظر گرفته‌ایم. در forward، ابتدا سافت پرامپت را تکرار می‌کنیم تا به تعداد batch size برسد. سپس آن را به ابتدای امبدینگ‌های اصلی (بدون بد) اضافه می‌کنیم.

```
class CustomEmbeddingLayer(nn.Module):
    def __init__(self, original_embedding, n_prompts):
        super(CustomEmbeddingLayer, self).__init__()
        self.original_embedding = original_embedding
        self.n_prompts = n_prompts # Number of tokens added as soft prompt
        # Initialize soft prompts
        self.soft_prompts = nn.parameter.Parameter(self.initialize_prompt())

    def initialize_prompt(self, initialize_from_vocab=True):
        if initialize_from_vocab:
            return self.original_embedding.weight[:self.n_prompts]
        else:
            return torch.randn(self.n_prompts, self.original_embedding.embedding_dim)

    def forward(self, input_ids):
        original_embeds = self.original_embedding(input_ids)
        batch_size = input_ids.size(0)
        # Expand soft prompts size to batch size
        soft_prompt_embeds = self.soft_prompts.unsqueeze(0).expand(batch_size, -1, -1)
        embeds_without_pad = original_embeds[:, self.n_prompts:]
        return torch.cat([soft_prompt_embeds, embeds_without_pad], dim=1)
```

با استفاده از تابع change_model_embedding، لایه امبدینگ T5 را با لایه‌ای که تعریف کردیم جابجا می‌کنیم.

```
def change model embedding(model):
    encoder = model.get_encoder()
    embedding_layer = encoder.get_input_embeddings()
    new_embedding_layer = CustomEmbeddingLayer(embedding_layer, N_SOFT_PROMPT)
    encoder.set input embeddings(new embedding layer)

change_model_embedding(model)
```

مطابق مقاله، تنها باید توکن‌های سافت پرامپت مورد آموزش قرار بگیرند و بقیه وزن‌های شبکه بدون تغییر باقی بماند. با استفاده از تابع freeze_params این کار را انجام می‌دهیم.

```
def freeze_params(model, trainable_params):
    for name, param in model.named_parameters():
        param.requires_grad = any(trainable_param in name for trainable_param in
trainable_params)
```

```
# Freeze all parameters except soft prompts
freeze_params(model, ['soft_prompts'])
```

در تابع train_one_epoch کد مربوط به آموزش مدل برای ۱ ایپاک پیاده‌سازی شده است.

```
def train_one_epoch(loader, model, optimizer):
    running_loss = 0

    model.train()
    for batch, data in enumerate(tqdm(loader)):
        optimizer.zero_grad()

        data = data.to(DEVICE)
        output = model(**data)
        batch_loss = output.loss

        batch_loss.backward()
        optimizer.step()

        running_loss += batch_loss.item()

        if (batch+1) % TRAIN_LOG == 0:
            print(f'  batch {batch+1} loss: {batch_loss.item()}')

    running_loss /= len(loader)
    return running_loss
```

در تابع val_one_epoch، کد مربوط به ارزیابی مدل برای ۱ ایپاک پیاده‌سازی شده است.

```
def post_process(data):
    # Convert token ids to words
    data = tokenizer.batch_decode(data, skip_special_tokens=True)
    # Convert words to class labels
    data = imdb_labels.convert_labels_to_ids(data)
    return data

def val_one_epoch(loader, model, optimizer):
    predictions = []
    labels = []
    running_loss = 0

    model.eval()
```

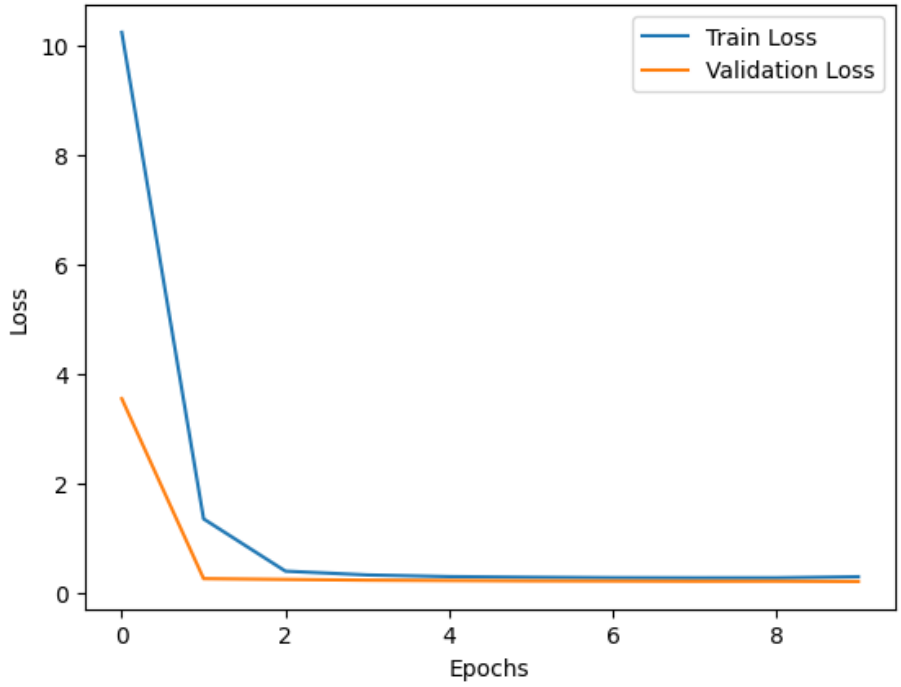
```
    with torch.no_grad():
        for batch, data in enumerate(tqdm(loader)):
            data = data.to(DEVICE)
            output = model(**data)
            # Calculate loss
            batch_loss = output.loss
            running_loss += batch_loss.item()
            # Generate model prediction
            pred = model.generate(input_ids=data.input_ids, attention_mask=data.attention_mask,
max_new_tokens=10)
            predictions.extend(pred)
            labels.extend(data.labels)

    predictions = post_process(predictions)
    labels = post_process(labels)

    acc = accuracy_score(y_true=labels, y_pred=predictions)
    running_loss /= len(loader)
    return {"loss": running_loss, "accuracy": acc}
```

با استفاده از تابع train_val_loop مدل را در ۱۰ ایپاک آموزش داده و ارزیابی می‌کنیم. نتایج بدست آمده به صورت زیر است:

شماره ایپاک	مقدار خطای آموزش	مقدار خطای ارزیابی	مقدار دقت ارزیابی
۱	۱۰,۲۳	۳,۵۵	۰
۲	۱,۳۵	۰,۲۶	۰,۷۶
۳	۰,۴۰	۰,۲۵	۰,۷۵
۴	۰,۳۳	۰,۲۳	۰,۷۶
۵	۰,۳۰	۰,۲۳	۰,۷۸
۶	۰,۲۹	۰,۲۲	۰,۷۸
۷	۰,۲۸	۰,۲۲	۰,۷۹
۸	۰,۲۸	۰,۲۱	۰,۷۹
۹	۰,۲۸	۰,۲۲	۰,۸۰
۱۰	۰,۳۰	۰,۲۱	۰,۸۰

نکته قابل توجه آن است که در ایپاک اول دقت ارزیابی ۰ درصد است. برای بررسی بیشتر، پیش‌بینی مدل را پس از پایان ایپاک اول چاپ کردیم که به صورت ‘ ’ و ‘.’ بود. برای جلوگیری از این اتفاق، می‌توانیم logit ها را به گونه‌ای پردازش کنیم که مدل تنها بین برچسب‌های مورد نظر حق انتخاب داشته باشد.

همچنین نمودار مقدار خطای آموزش و ارزیابی مدل به صورت زیر است:



مطابق این نمودار، آموزش در وضعیت مناسبی قرار دارد و مدل دچار overfit یا underfit نشده است. با ادامه فرآیند آموزش می‌توانیم به عملکرد بهتری از مدل دست یابیم.

در آخر عملکرد مدل را روی داده آزمون ارزیابی می‌کنیم. طی این آموزش مدل توانسته به دقت ۸۱ درصد بر روی داده آزمون برسد.

همچنین بنا به نیاز می‌توانیم از Wandb برای لاگ کردن مواردی از قبیل میزان خطا استفاده کنیم.