پیش پردازش

کلاس TextPreProc به طور کلی وظیفه تمیز کردن متن ورودی شامل پاک کردن stop word ها پاک کردن لینک ها و white space ها را دارد.

Embedding

مدل fast text: برای اینکه مدل CNN بتواند متن ها را پردازش کند باید داده های متنی ابتدا به داده های عددی تبدیل شود که برای اینکار از مدل fast text استفاده شده. این مدل از قبل روی زبان فارسی آموزش داده شده و هر کلمه فارسی را به یک بردار عددی به طول ۳۰۰ تبدیل می کند. این مدل کلماتی که از نظر معنایی نزدیک هستند را طوری در این فضای برداری قرار می دهد که بهم دیگر نزدیک باشند این باعث می شود تا مدل CNN بتواند روی کلماتی که در داده آموزش قبلا نبوده هم عملکرد نسبتا خوبی داشته باشد

طول جمله: یکی از مشکلاتی که برای پردازش جملات وجود دارد این است که ابعاد ورودی در شبکه باید مشخص باشد در حالی که طول جملات در دیتاست برابر نیست بنابراین ابتدا باید بتوانیم طول جملات را به طور بهینه مشخص کنیم. هدف ایجاد تعادل بین Truncation و Padding است. اگر طول بهینه را کم در نظر بگیریم اطلاعات جملات را از دست می دهیم و اگر این طول خیلی زیاد باشد منجر به افزایش توان پردازشی و هدر رفتن حافظه می شود. بنابراین یک روش منطقی این است که توزیع طول جملات در مجموعه داده آموزش را تحلیل کنیم:

```
for p in [90, 95, 99]:
    print(f"{p}th percentile:", np.percentile(sentence_lengths, p))
```

✓ 0.0s

```
90th percentile: 38.0
95th percentile: 49.0
99th percentile: 78.0
```

این آمار نشان می دهد که ۹۵ درصد جملات دیتاست طولی کمتر یا مساوی ۴۹ دارند. بر این اساس طول بهینه را ۵۰ در نظر گرفتیم که باعث می شود ۹۵ درصد جملات بدون حذف هیچ کلمه ای وارد شبکه شوند و همینطور مدل روی داده های دور افتاده و نویزی آموزش نمی بیند.

Padding: در این روش اگر طول ورودی کمتر از حد تعیین‌شده باشد، انتهای آن با بردارهای صفر پر می‌شود تا به ابعاد استاندارد برسد و در شبکه رفتاری خنثی داشته باشد.

Truncation: برای ورودی‌هایی که طولشان بیشتر از حد تعیین‌شده است، بخش اضافی از انتهای آن‌ها حذف (برش) می‌شود تا اندازه آن‌ها با ورودی شبکه فیت شود.

Dataset:

این کلاس برای کپسوله سازی فرآیند بارگذاری داده ها نوشته شده است. هر نظر را ابتدا با استفاده از کلاس embedder به بردار عددی تبدیل می کند و به همراه برچسب آن کلاس یک فرمت تانسور خروجی می دهد نکته مهم این است که نوع داده ها برای سازگاری با توابع هزینه باید float 32 و long باشد

CNN classifier:

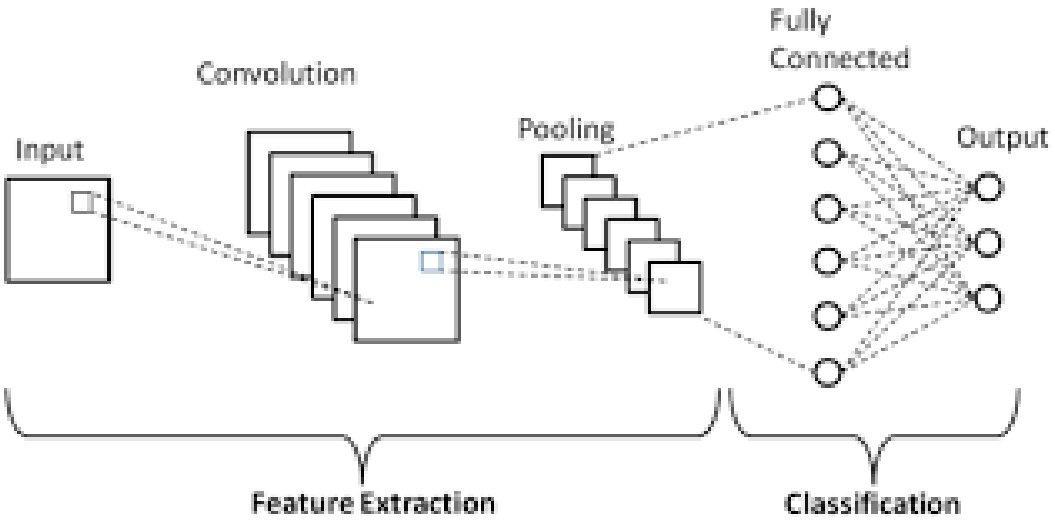
```
class CNNClassifier(torch.nn.Module):
    def __init__(self, input_dim = FAST_TEXT_DIM, hidden_dim = 128 , dropout_prob=0.5):
        super(CNNClassifier, self).__init__()
        linier_dim = int((MAX_LEN / 2) * hidden_dim)
        self.conv1 = nn.Conv1d(input_dim, hidden_dim, kernel_size=3, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool1d(2)
        self.dropout = nn.Dropout(dropout_prob)
        self.classifier = nn.Linear(linier_dim , 2)

    def forward(self , x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.dropout(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)

        return x
```

توضیح لایه های شبکه:

لایه کانولوشن یک‌بعدی: در این لایه، اندازه هسته برابر با ۳ تنظیم شده است. این یعنی مدل در هر لحظه یک پنجره‌ی ۳ کلمه‌ای را روی متن حرکت می‌دهد تا ارتباط معنایی میان کلمات متوالی را درک کند. تعداد فیلترهای خروجی برابر با ۱۲۸ در نظر گرفته شده است؛ به این معنا که شبکه تلاش می‌کند در هر گام، ۱۲۸ ویژگی یا الگوی مختلف را از ترکیب کلمات استخراج کند. پارامترهای این فیلترها (وزن‌ها و بایاس‌ها) در طول فرایند آموزش (Backpropagation) به‌روزرسانی می‌شوند تا مدل بهترین ویژگی‌ها را برای تشخیص احساسات یاد بگیرد.

- تابع فعال‌ساز (ReLU)**:** بلافاصله پس از عملیات کانولوشن، از تابع فعال‌ساز **ReLU** استفاده شده است. هدف اصلی این تابع، اعمال **Non-linearity** به شبکه است. بدون این تابع، تمام عملیات شبکه تبدیل به یک ترکیب خطی ساده می‌شد و مدل قدرت یادگیری الگوهای پیچیده را ندارد.

لایه کاهش ابعاد: این لایه با انتخاب بیشتری مقدار در بازه های دوتایی ابعاد ورودی برای لایه بعد را نصف می کند و حجم محاسبات را کاهش می دهد ضمن استخراج برجسته ترین ویژگی ها.
لایه Dropout: برای جلوگیری از overfit شدن مدل روی داده های آموزش از تکنیک حذف تصادفی استفاده می شود. در این حالت هر نورون از لابه قبل به یک احتمالی خروجی صفر می دهد این باعث می شود که شبکه برای پیش بینی به تعداد محدودی از نورون ها و ترکیب آنها تکیه نکند. نکته مهم این است که این لایه تنها در زمان آموزش فعال است و در زمان ارزیابی از همه نورون ها برای پیش بینی استفاده می شود
لایه خروجی: این یک لایه خطی تمام متصل است که ابعاد ورودی آن به اندازه ابعاد خروجی لایه کاهش ابعاد یعنی (نصف طول جملات * تعداد فیلتر ها یا نورون های لایه کانولوشن) است. ابعاد خروجی آن نیز به اندازه همان تعداد نورون های خروجی است که برای این مسئله چون یک classifier باینری است برابر ۲ می شود. خروجی اول امتیاز کلاس صفر و خروجی دوم امتیاز کلاس یک است. به طور کل عملکرد این لایه ضرب ماتریسی ساده برای نگاشت کردن فضای ویژگی ها به فضای کلاس ها است



پیدا کردن هایپر پارامتر ها:

پیدا کردن مقدار مناسب برای فرا پارامتر هایی مانند طول جملات,نرخ dropout , نرخ یادگیری می تواند در نحوه یادگیری مدل تاثیر زیادی بگذارند اما به دلیل محدودیت های سیستمی برای آموزش طولانی مدت مدل صرفا دوتا از این هایپر پارامتر ها با دامنه بسیار محدود مورد تحلیل قرار گرفته

```
learning_rates = [1e-3, 5e-4, 1e-4]
dropout_rates  = [0.3, 0.5]

BATCH_SIZE = 32
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
dev_loader = DataLoader(dev_dataset, batch_size=BATCH_SIZE, shuffle=False)

for lr in learning_rates:
    for do in dropout_rates:
        model = CNNClassifier(dropout_prob=do)

        loss = torch.nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr=lr)

        trained_model = train_model(
            model=model,
            optimizer=optimizer,
            train_loader=train_loader,
            val_loader=dev_loader,
            loss_module=loss,
            num_epochs=10,
        )

        print(f"Done with LR={lr}, DO={do}")

print("finish")
```

برای پیدا کردن مقدار مناسب learning rate , dropout rate مدل روی ترکیب های مختلف آموزش دیده شده و دقت آن روی دیتاست dev مورد ارزیابی قرار گرفته و نتیجه به این شکل بوده است:

```
Start training ...


Training Finished. Best Validation Accuracy: 81.46%
Done with LR=0.001, DO=0.3
Start training ...


Training Finished. Best Validation Accuracy: 81.62%
Done with LR=0.001, DO=0.5
Start training ...


Training Finished. Best Validation Accuracy: 82.06%
Done with LR=0.0005, DO=0.3
Start training ...


Training Finished. Best Validation Accuracy: 81.84%
Done with LR=0.0005, DO=0.5
Start training ...


Training Finished. Best Validation Accuracy: 81.37%
Done with LR=0.0001, DO=0.3
Start training ...


Training Finished. Best Validation Accuracy: 81.29%
Done with LR=0.0001, DO=0.5
finish
```

با توجه به این نتایج مقادیر هایپر پارامتر هارو تعیین کرده و مدل را با استفاده از آنها مجدد آموزش می دهیم

دقت مدل روی داده های تست:

```python
def evaluate(model, test_loader , device='cpu'):
    model.eval()
    total_correct_pred = 0
    total_data = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            total_correct_pred += (preds == labels).sum().item()
            total_data += labels.size(0)

    accuracy = (total_correct_pred / total_data) * 100
    return accuracy
```

✓ 0.0s

```python
BATCH_SIZE = 32

df_test = pd.read_csv("./snappfood/test.csv", sep="\t")
test_dataset = SentimentDataset(dataframe=df_test, embedder=embedder)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE, shuffle=False)

print(f"Accuracy on test set: {evaluate(trained_model, test_loader):.2f}%")
```

✓ 3.1s

```
Accuracy on test set: 82.19%
```