자연어 처리 Assignment1

2019312014 박병준

담당교수: 최윤석

Learning rate가 크면 학습이 안되는 이슈로 인해 두 가지 LR 값(5, 0.001)으로 각각 학습시켰다.

```
# Hyperparameters
EPOCHS = 10  # epoch
LR = 0.001  # learning rate
BATCH_SIZE = 64  # batch size for training
```

1. Conduct experiments with different op□mizers: SGD, Adam, Adagrad, RMSProp

```
optimizer = torch.optim.Adam(model.parameters(), lr=LR)
```

optimizer	test accuracy(lr=0.001)	train accuracy	valid accuracy	predict golf news	test accuracy(lr=5)
SGD	0.258	0.255	0.26	World news	0.907
Adam	0.918	0.968	0.918	Sports news	0.897
Adagrad	0.814	0.825	0.821	Sports news	0.905
RMSProp	0.919	0.953	0.917	Sports news	0.894

Lr=5인 경우 accuracy가 optimizer에 관계없이 90%에 근사하므로 분석에 적당하지 않아 lr=0.001인 경우를 중심으로 분석해보고자 한다. SGD는 batch 단위로 weight를 수정하는 기본적인 gradient descent 방식이다. Momentum 등의 방식과 달리 weight가 update되는 정도에 대한 가중치가 없어서, lr=0.001일 때의 accuracy를 보면 25%정도에 불과해 학습이 거의 진행되지 않았음을 알 수 있다(class가 4개뿐인 dataset이므로). SGD 다음으로 Adagrad가 낮은 accuracy를 기록했다. Adagrad는 학습을 통해 덜 update된 weight는 더 update하고 많이 update된 weight는 덜 update되도록 하는데, 학습이 오래되면 weight parameter가 지금까지 update된 총량의 값이 커져 학습이 되지 않는 문제가 발생할 수 있다. 이 때문에 optimizer가 Adam이나 RMSProp일 때보다 accuracy가 낮게 나왔다고 생각한다. Adam과 RMSProp의 test accuracy는 별 차이가 없는데, test accuracy가 0.92 이상으로 더 높아지지 않아 둘 간의 차이는 비교하기 어렵다고 볼 수 있다. RMSProp은 Adagrad의 단점(학습이 오래되면 weight가 update되지 않는 문제)을 보완한 방식으로, 지금까지 weight가 update된 총량을 보지 않고 대신 최근에 얼마나 weight가 update되었는지를 반영한다. 이로 인해 RMSProp이 Adagrad보다 높은 성능을 보일 수 있다. Adam은 RMSProp에 momentum을 융합한 방법으로, 가장 성능이 좋을 것으로 예상되었으나 마지막 epoch batch당 train accuracy에서만 1.5% 정도로 유의미하게 높은 성능을 보였다. Optimizer를 바꿔도 accuracy가 개선되지 않는 이유로는 model이 training dataset에 overfitting되거나 dataset이 부족한 이유

등 여러 가지가 있다.

2. Use Adam optimizer, conduct experiments with different number of epochs: 5, 10, 20, 50

epoch	test accuracy(lr=0.001)	train accuracy	valid accuracy	test accuracy(lr=5)
5	0.918	0.967	0.917	0.887
10	0.919	0.968	0.918	0.9
20	0.916	0.975	0.918	0.898
50	0.92	0.977	0.92	0.901

동일한 hyper parameter와 model을 가지고 epoch만 달리하여 학습을 진행했기 때문에, 통상적으로 epoch가 증가하여 학습이 진행될수록 accuracy가 증가한다. Test accuracy의 경우 epoch에 관계없이 비슷한 수치를 나타내기 때문에 비교가 어렵지만, train accuracy의 경우 epoch가 증가할수록 accuracy가 증가하는 유의미한 결과를 보여주었다. 다만 모든 accuracy 값이 epoch에 관계없이 일정한 값에 수렴하는 결과를 보여주었다. Accuracy가 개선되지 않는 이유는 앞서 언급한 바 있다.

3. Use Adam optimizer, 50 epochs and run the experiments with the following models

model	test accuracy	train accuracy	valid accuracy
1 layer, hidden dimension 128	0.917	0.977	0.916
2 layer, hidden dimension 128	0.916	0.977	0.92
2 layer, hidden dimension 64	0.918	0.967	0.923
3 layer, hidden dimension 128	0.919	0.969	0.92
3 layer, hidden dimension 64	0.916	0.968	0.92
3 layer, hidden dimension 32	0.918	0.965	0.923

```
emsize = 128
model = TextClassificationModel(vocab_size, emsize, num_class).to(device)
```

```
class TextClassificationModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=False)
        self.fc = nn.Linear(embed_dim, embed_dim)
        self.fc = nn.Linear(embed_dim, embed_dim)
        self.fc = nn.Linear(embed_dim, num_class)
        self.init_weights()
```

Test accuracy에선 유의미한 차이를 찾기 어려웠으나, train accuracy에선 layer 수가 같으면 hidden dimension 값이 높을수록 accuracy가 증가하는 경향을 보였다. hidden dimension 수가 많을수록 연산이 더 많이 이루어져서 더 높은 accuracy를 보인다고 생각한다. 하지만 모든 경우에서 epoch가 4일 때부터 accuracy가 특정 값에 도달하여 더 이상 증가하지 않았기 때문에, layer 수나 hidden dimension 수에 따른 accuracy 차이를 정확히 비교 분석할 수 없었다.