Lab 5
Pytorch Common Mistakes

Prepared by:
Omar Samir

Common Mistakes

1. Didn’t overfit a single batch
2. Forgot to set training or eval
3. Forgot optimizer.zero_grad()
4. Using Softmax with Cross Entropy Loss
5. Using view() as permute()
6. Not shuffling the data
7. Not Clipping Gradients

Didn’t overfit a single batch

After creating your neural network, you need to make sure that it has the ability to learn your problem and that your training loop has no bugs. To do so, make sure you do the following before the actual training:

1. Select a small batch from your dataset (say 64 training example)
2. Train your network on this small batch only
3. Make sure that the loss on this small batch is nearly Zero

Forgot to set training or eval

Pytorch supports two functions to be applied on your model

1. model.train(): This function is used to put your model in the training mode
2. model.eval(): This function is used to put your model in evaluation mode

Some layers are used while training that we need to remove in the testing mode like dropout layers

Forgot optimizer.zero_grad()

You will need to calculate your gradients with respect to the current batch. The default pytorch behaviour is accumulating the gradients. So if you forgot to use it, you will end up optimizing your model at each step with all the gradients of previous batches.

Using Softmax with Cross Entropy Loss

- Softmax is already done in the cross entropy loss.
- Doing softmax in your forward function will end up with applying two softmaxes one by you and one by the cross entropy loss function

Using view() as permute()

```python
import torch

x = torch.tensor([[1,2,3],[4,5,6]])

print(x)

print(x.view(3, 2))
print(x.permute(1, 0))
```

```
tensor([[1, 2, 3],
        [4, 5, 6]])
tensor([[1, 2],
        [3, 4],
        [5, 6]])
tensor([[1, 4],
        [2, 5],
        [3, 6]])
```

Not shuffling the data

- As we saw in the previous tutorial, shuffling can be easily done using pytorch dataloader
- Shuffling is very important as you need your mini-batches to contain nearly all the different examples from your dataset
- Forgetting to shuffle the data may cause the batches to be biased towards specific classes (some batches are positive, and the other are negative)
- This will cause your optimization loop to go in a biased direction based on the batch type

Not Clipping Gradients

- Clipping gradients is important when useing RNNS, LSTMs and GRUs
- It helps to solve the exploding gradients problem
- The exploding gradients problem happen when the gradients are large and the backpropagation will result in multiplying the partial gradients resulting in a very large step for your wights
- If the data is noisy or the current batch has a problem, this will cause your network to go fast in a direction that we want to avoid
- Overall exploding gradients will cause your training process to be unstable

Not Clipping Gradients

Can be simply done in pytorch using one line

```
# backward
optimizer.zero_grad()
loss.backward()
torch.nn.utils.clip_grad_norm(model.parameters(), max_norm=1)
```