Introduction to Machine Learning

Homework Solution - 5

Instructor: Prof Chinmay Hegde *TA:* Devansh Bisla, Maryam Majzoubi

Problem 1
Consider a one-hidden layer neural network (without biases for simplicity) with sigmoid activations trained with squared-error loss. Draw the computational graph and derive the forward and backward passes used in backpropagation for this network.

$$\hat{y} = W_2\sigma(W_1 x), \quad \mathcal{L} = \|\hat{y} - y\|_2^2$$

Qualitatively compare the computational complexity of the forward and backward passes. Which pass is more expensive and by roughly how much?

(Solution) The computational graph is shown in Figure 1.

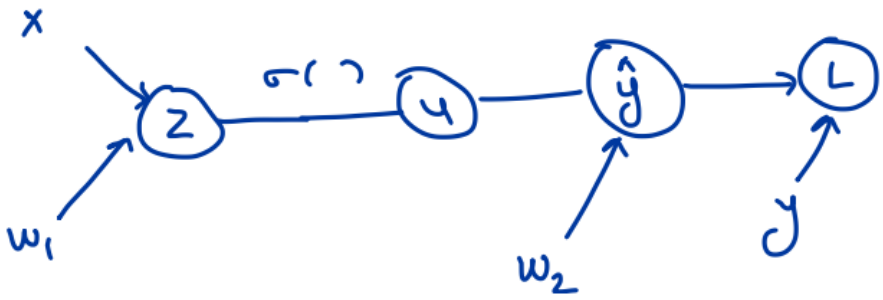

Figure 1: Computational graph.

The following equation show the forward propagation of the model:

$$Z = XW_1 \tag{0.1}$$
$$u = \sigma(Z) \tag{0.2}$$
$$\hat{y} = uW_2 \tag{0.3}$$
$$L = (\hat{y} - y)^2 \tag{0.4}$$

The backward propagation requires computation of gradient of the loss function. The following equation show the backward propagation:

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial W_2} = 2(\hat{y} - y)^T u \tag{0.5}$$
$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial u}\frac{\partial u}{\partial Z}\frac{\partial Z}{\partial W_1} = 2(\hat{y} - y)W_2\sigma(Z)(1 - \sigma(Z))X \tag{0.6}$$

It can be easily observed from above equations that backward propagation requires much more computation (multiplication) than forward propagation, roughly 2 times.

Problem 2
Suppose that a convolutional layer of a neural network has an input tensor $X[i, j, k]$ and computes an output as follows:

$$Z[i, j, m] = \sum_{k_1} \sum_{k_2} \sum_{n} W[k_1, k_2, n, m] X[i + k_1, j + k_2, n] + b[m]$$
$$Y[i, j, m] = \text{ReLU}(Z[i, j, m])$$

for some kernel W and bias b. Suppose X and W have shapes $(48, 64, 10)$ and $(3, 3, 10, 20)$ respectively.

(a) What are the shapes of Z and Y?

(b) What are the number of input and output channels?

(c) How many multiply- and add- operations are required to perform a forward pass through this layer? Rough calculations are OK.

(d) What are the total number of trainable parameters in this layer?

(Solution)

(a) The shape of both Z and Y is $(48 - 3 + 1, 64 - 3 + 1, 20) = (46, 62, 20)$.

(b) The number of input channels is 10 and the number of output channels is 20.

(c) There are $(46 * 62 * 20) * (3 * 3 * 10) = 5,133,600$ multiply- and add- operations are required.

(d) There are $(3 * 3 * 10 * 20) + 20 = 1820$ trainable parameters in total.

Problem 3
The use of ℓ_2 regularization for training multi-layer neural networks has a special name: *weight decay*. Assume an arbitrary dataset $\{(x_i, y_i)\}_{i=1}^n$ and a loss function $\mathcal{L}(w)$ where w represents the trainable weights (and biases).

(a) Write down the ℓ_2 regularized loss, using a weighting parameter λ for the regularizer.

(b) Derive the gradient descent update rules for this loss.

(c) Conclude that in each update, the weights are "shrunk" or "decayed" by a multiplicative factor before applying the descent update.

(d) What does increasing λ achieve algorithmically, and how should the learning rate be chosen to make the updates stable?

(Solution)

(a) $Loss = \mathcal{L}(w) + \lambda||w||_2^2$

(b) $w^{t+1} = w^t - \alpha\frac{\partial Loss}{\partial w} = w^t - \alpha(\nabla\mathcal{L}(w^t) + 2\lambda w^t)$

(c) We can re-write the above as: $w^{t+1} = w^t(1 - 2\alpha\lambda) - \alpha\nabla\mathcal{L}(w^t)$. As it is observed, in each update the weights are decayed by a multiplicative factor of $(1 - 2\alpha\lambda)$.

(d) Increasing λ will encourage more regularization and hence results in smaller weights. In order to have a stable update the learning rate should not be too high. More specifically, we need $0 < (1 - 2\alpha\lambda) < 1$. This results the learning rate to be $0 < \alpha < \frac{1}{2\lambda}$.

Problem 4
In this exercise, we will fine-tune a pre-trained deep network (ResNet34) for a particular two-class dataset which can be downloaded from the attached zip file. Code for preprocessing the dataset, and for loading ResNet34, can be found below. Since ResNet34 is for 1000 output classes, you will need to modify the last layer to reduce two classes. Train for 20 epochs and plot train and validation loss curves. Report your final train and validation accuracies.

The code is defined below:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt

# data transforsm
dset_transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])])


# Use the image folder function to create datasets
dsets = {x: datasets.ImageFolder(f"data/{x}", dset_transform)
         for x in ['train', 'val']}

# create data loader
dataloaders = {x: torch.utils.data.DataLoader(dsets[x], batch_size=16,
                                              shuffle=(x == "train"), ...
                                                  num_workers = 0)
               for x in ['train', 'val']}

# intialize model
model = models.resnet34(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 2)

# define loss funtion
criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
save_loss = {'train':[], 'val':[]}
save_acc = {'train':[], 'val':[]}

for epoch in range(10):

    # Each epoch has a training and validation phase
    for phase in ['train', 'val']:
        if phase == 'train':
            model.train()  # Set model to training mode
        else:
            model.eval()   # Set model to evaluate mode
```

```
46
47          current_loss = 0.0
48          current_corrects = 0
49
50          for batch_idx, (inputs, labels) in enumerate(dataloaders[phase], 1):
51              # We need to zero the gradients, don't forget it
52              optimizer.zero_grad()
53
54              # Time to carry out the forward training poss
55              with torch.set_grad_enabled(phase == 'train'):
56                  outputs = model(inputs)
57                  _, preds = torch.max(outputs, 1)
58                  loss = criterion(outputs, labels)
59
60                  # backward + optimize only if in training phase
61                  if phase == 'train':
62                      loss.backward()
63                      optimizer.step()
64
65              # We want variables to hold the loss/acc statistics
66              current_loss += loss.item() * inputs.size(0)
67              current_corrects += torch.sum(preds == labels.data)
68          # saving variable for plottin
69          save_loss[phase] += [current_loss / len(dataloaders[phase].dataset)]
70          save_acc[phase] += [current_corrects.float() / ...
                len(dataloaders[phase].dataset)]
71
72          # pretty print
73          print(f"Epoch:{epoch} -- Phase:{phase} -- ...
                Loss:{save_loss[phase][-1]:.2f} -- ...
                Acc:{save_acc[phase][-1]*100:.2f}")
74
75  # pretty plots
76  plt.plot(save_loss['train'])
77  plt.plot(save_loss['val'])
78  plt.legend(["train", "val"])
79  plt.title("Loss")
80  plt.savefig('loss.png')
81  plt.close()
82  plt.plot(save_acc['train'])
83  plt.plot(save_acc['val'])
84  plt.legend(["train", "val"])
85  plt.title("Accuracy")
86  plt.savefig('acc.png')
87  plt.close()
```

The corresponding output is shown below:

```
Epoch:0 -- Phase:train -- Loss:1.06 -- Acc:38.33
Epoch:0 -- Phase:val -- Loss:0.80 -- Acc:41.67
Epoch:1 -- Phase:train -- Loss:0.51 -- Acc:78.33
Epoch:1 -- Phase:val -- Loss:0.36 -- Acc:87.50
Epoch:2 -- Phase:train -- Loss:0.18 -- Acc:100.00
Epoch:2 -- Phase:val -- Loss:0.12 -- Acc:100.00
Epoch:3 -- Phase:train -- Loss:0.09 -- Acc:100.00
Epoch:3 -- Phase:val -- Loss:0.04 -- Acc:100.00
Epoch:4 -- Phase:train -- Loss:0.04 -- Acc:100.00
```

```
Epoch:4 -- Phase:val -- Loss:0.02 -- Acc:100.00
Epoch:5 -- Phase:train -- Loss:0.03 -- Acc:100.00
Epoch:5 -- Phase:val -- Loss:0.01 -- Acc:100.00
Epoch:6 -- Phase:train -- Loss:0.02 -- Acc:100.00
Epoch:6 -- Phase:val -- Loss:0.01 -- Acc:100.00
Epoch:7 -- Phase:train -- Loss:0.02 -- Acc:100.00
Epoch:7 -- Phase:val -- Loss:0.01 -- Acc:100.00
Epoch:8 -- Phase:train -- Loss:0.01 -- Acc:100.00
Epoch:8 -- Phase:val -- Loss:0.01 -- Acc:100.00
Epoch:9 -- Phase:train -- Loss:0.02 -- Acc:100.00
Epoch:9 -- Phase:val -- Loss:0.01 -- Acc:100.00
```

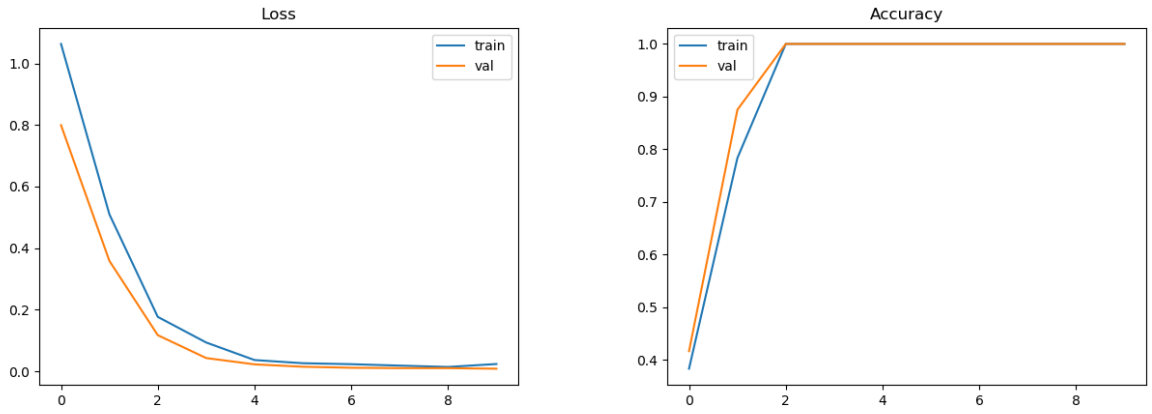

Figure 2: Training and validation **(left)**: Loss and **(Right):** Accuracy.