Chapter 4

1. How is a greyscale image represented on a computer? How about a color image?

Greyscale images are represented as 2-dimensional arrays, with each value with a range of 256 corresponding to the pixel value of the content of the image. 0 represents white, while 255 black, and numbers inbetween represent shades of grey.

Color images consist of three channels: red, green and blue. The images are represented by 256-value 2-dimensional arrays for each of the channels. 0 represents white, while 255 represents solid red, green, or blue, and numbers in between represent shades of the channel color. The three 2-dimensional arrays form a final 3-dimensional array which represents the color image.

2. How are the files and folders in the MNIST_SAMPLE dataset structured? Why?

There are two folders in `MNIST_SAMPLE`: `train` and `valid`. They respectively contain the training and validation dataset. This allows the user to use a uniform dataset for validating their model performance after each training step. Both the folders are further split into two folders: `3` and `7`. These provide the labels for the data. These folders contain the respective `.jpg` files for the respective class of images.

3. Explain how the “pixel similarity” approach to classifying digits works.

In the *pixel similarity* approach, we generate the "ideal" 3 and 7 from the training data by taking the average pixel value for every pixel for all the 3s and 7s in the training data respectively. Then, for an unseen image, we calculate its distance from the ideal 3 and 7 (pixel-wise absolute difference). If the distance is less from 3, the image is classified as a 3, else a 7.

4. What is a list comprehension? Create one now that selects odd numbers from a list and doubles them.

List comprehension is a pythonic way of iterating over a list or an iterator without using a `for`-loop.

```
def return_odd_double(input_list):
  odd_double = [2*i for i in input_list if i%2==1]
  return odd_double
```

5. What is a rank-3 tensor?

A rank-3 tensor is a tensor with 3 dimensions, eg a tensor of shape 4×5×6.

6. What is the difference between tensor rank and shape? How do you get the rank from the shape?

Rank is the number of dimensions in a tensor. Shape is the size of all dimensions of a tensor.

```
def get_rank(sample_tensor):
  return len(sample_tensor.shape)
```

7. What are RMSE and L1 norm?

L1 norm and RMSE are two commonly used methods of measuring distances. Simple differences do not work because positives and negatives cancel each other out, so alternatives have to be used:

- *absolute value* of differences (absolute value is the function that replaces negative values with positive values). This is called the *mean absolute difference* or *L1 norm*
- mean of the *square* of differences (which makes everything positive) and then take the *square root* (which undoes the squaring). This is called the *root mean squared error* (RMSE) or *L2 norm.*

8. How can you apply a calculation on thousands of numbers at once, many thousands of times faster than a Python loop?

Loops are slow in Python. The alternative is to express the same operation as array operations. Array operations are executed by NumPy or PyTorch, which allws them to be executed in C, which makes them much faster than Python. In the presence of a GPU, these operations will be executed in CUDA (again, much faster than Python), which also be parallelizable, allowing for the calculation of thousands of numbers at once.

9. Create a 3x3 tensor or array containing the numbers from 1 to 9. Double it. Select the bottom right 4 numbers.

```
a = tensor(range(1,10)).view(3,3)
print(a)
OUTPUT: tensor([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

b = 2*a
print(b)
OUTPUT: tensor([[ 2,  4,  6],
                [ 8, 10, 12],
                [14, 16, 18]])

c = b[1:,1:]
print(c)
OUTPUT: tensor([[10, 12],
                [16, 18]])
```

10. What is broadcasting?

Broadcasting is a capability of PyTorch that allows for operations between tensors of different ranks. For example, in case of a subtraction operation between tensors of two different ranks, it expands the tensor of the smaller rank to have the same rank as the the one with the larger rank without allocating any extra memory. It also does the actual operation in C (or CUDA as applicable).

11. Are metrics generally calculated using the training set, or the validation set? Why?

Metrics are generally calculated on a validation set. As the validation set is unseen data for the model, evaluating the metrics on the validation set is better in order to determine if there is any overfitting and how well the model might generalize if given similar data.

12. What is SGD?

SGD, or stochastic gradient descent, is an optimization algorithm. Specifically, SGD is an algorithm that will update the parameters of a model in order to minimize a given loss function that was evaluated on the predictions and target. The key idea behind SGD (and many optimization algorithms, for that matter) is that the gradient of the loss function provides an indication of how that loss function changes in the parameter space, which we can use to determine how best to update the parameters in order to minimize the loss function. This is what SGD does.

13. Why does SGD use mini batches?

- We can not calculate our gradient on the whole dataset due to compute limitations and time constraints.
- We can not calculate our gradient on each data point because the gradient will be unstable and imprecise because it would not use much information. The model's parameters would be updated but only to improve the performance of the model on that single data point.

Hence, a compromise is used: the average loss for a few data items at a time is calculated. This is called a mini-batch.

14. What are the 7 steps in SGD for machine learning?

- Initialize the parameters
- Calculate the predictions
- Calculate the loss
- Calculate the gradients
- Step the parameters
- Repeat the process (from step 2)
- Stop the training process

15. How do we initialize the weights in a model?

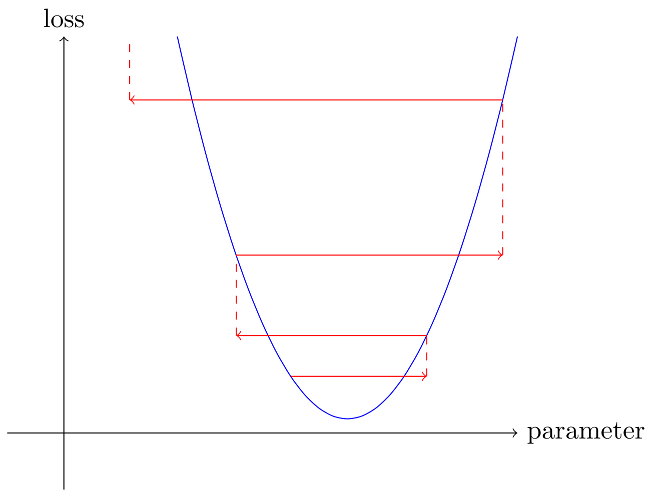
We use random weights.

16. What is “loss”?

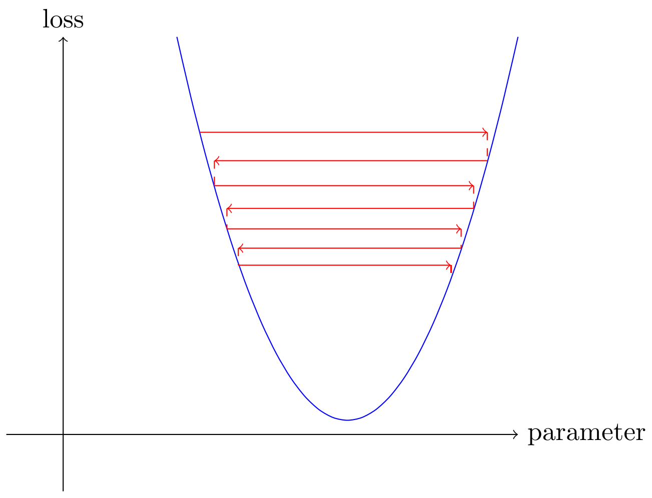
Loss is a measurement of the performance of the model, where lower loss corresponds to more accurate predictions and vice-versa.

17. Why can’t we always use a high learning rate?

- It can result in the loss getting worse



- It may also bounce around, taking too long to converge to a solution



18. What is a “gradient”?

Gradient is the slope, or the rate of change of the loss with respect to the parameter. It tells us how much we have to adjust the parameters to decrease the loss, and hence, improve the model.

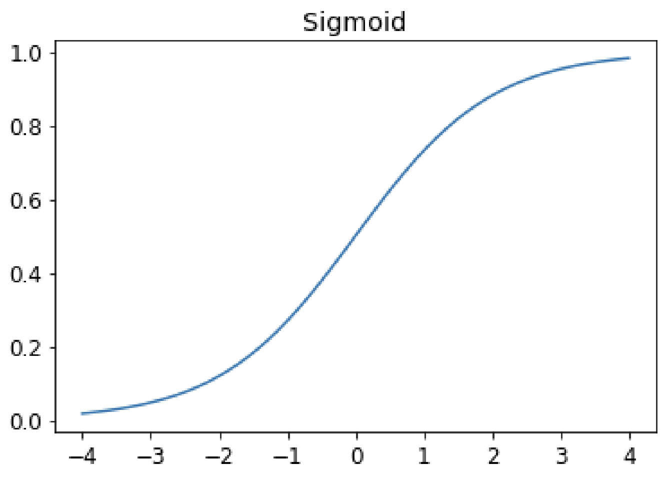
19. Do you need to know how to calculate gradients yourself?

Manual calculation of the gradients are not required, as deep learning libraries will automatically calculate the gradients for you. This feature is known as automatic differentiation. In PyTorch, if requires_grad=True, the gradients can be returned by calling the backward method: a.backward()

20. Why can't we use accuracy as a loss function?

A loss function needs to change as the weights are being adjusted. Accuracy only changes if the predictions of the model change. So if there are slight changes to the model that, say, improves confidence in a prediction, but does not change the prediction, the accuracy will still not change. Therefore, the gradients will be zero everywhere except when the actual predictions change. The model therefore cannot learn from the gradients equal to zero, and the model's weights will not update and will not train. A good loss function gives a slightly better loss when the model gives slightly better predictions.

21. Draw the sigmoid function. What is special about its shape?



Sigmoid function is a smooth function (has gradients everywhere) that squishes all values between zero and one, which is a requirement for the loss defined for the binary classification problem.

22. What is the difference between loss and metric?

The key difference is that metrics drive human understanding and losses drive automated learning. In order for loss to be useful for training, it needs to have a meaningful derivative. Many metrics, like accuracy are not like that. Metrics instead are the numbers that humans care about, that reflect the performance of the model.

23. What is the function to calculate new weights using a learning rate?

Optimizer step function

24. What does the DataLoader class do?

`DataLoader` takes any collection and converts it into an iterator of mini-batches.

25. Write pseudo-code showing the basic steps taken each epoch for SGD.

```
for xb ,yb in dl:
  preds = model(xb)
  loss = loss_function(preds, yb)
  loss.backward()
  parameters -= learning_rate * parameters.grad
  parameters.grad = None
```

26. Create a function which, if passed two arguments [1,2,3,4] and 'abcd' , returns [(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd')] . What is special about that output data structure?

```
def return_zip(l1, l2):
  return list(zip(l1,l2))
```

This data structure is useful for machine learning models when you need lists of tuples where each tuple would contain input data and a label.

27. What does view do in PyTorch?

It changes the shape of a tensor.

28. What are the “bias” parameters in a neural network? Why do we need them?

Without the bias parameter, if the input is zero, the output would be zero. We might want to output some value even if the input is zero.

29. What does the @ operator do in python?

Matrix multiplication

30. What does the backward method do?

Calculates and adds the currents gradients to the parameters' existing gradient.

31. Why do we have to zero the gradients?

`.backward` calculates and adds to the existing value in the paramters' `.grad` value. Hence, PyTorch will add the gradients of a variable to any previously stored gradients. If the training loop function is called multiple times, without zeroing the gradients, the gradient of current loss would be added to the previously stored gradient value.

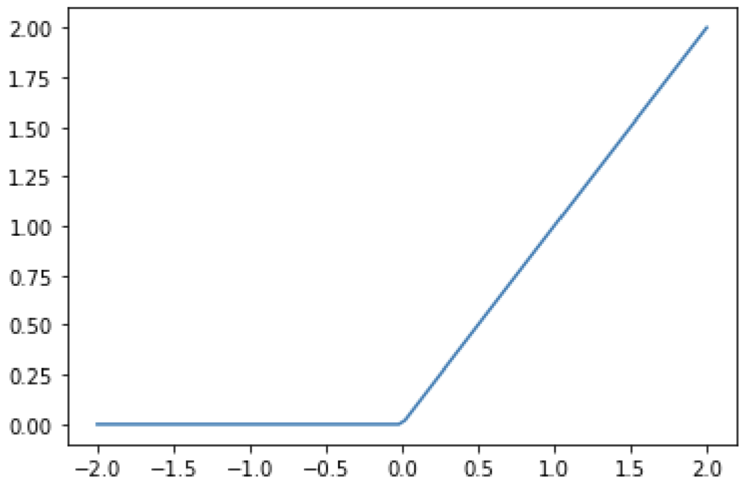
32. What information do we have to pass to Learner ?

- the `DataLoaders`
- the model/architecture
- optimization function
- loss function
- metrics to print

33. Show python or pseudo-code for the basic steps of a training loop.

```
def train_epoch(model, lr, params):
  for xb, yb in dl:
    preds = model(xb)
    loss = loss_function(preds, targets)
    loss.backward()
    for p in params:
      p.data -= lr * p.grad
      p.grad = None
```

34. What is “ReLU”? Draw a plot of it for values from -2 to +2 .



ReLU (or rectified linear unit) is a function that compares a value with 0, and takes the higher value. It translates to `result.max(0.0)` .

35. What is an “activation function”?

The activation function is another function that is part of the neural network, which has the purpose of providing non-linearity to the model. Without a non-linearity, the nueral network is just a series of linear functions, which can be composed into a singular linear function, y = mx + b. By introducing a non-linearity between every linear function, this is no longer true.

It can be mathematically proven that such a model can solve any computable problem to an arbitrarily high accuracy, if the model is large enough with the correct weights. This is known as the universal approximation theorem.

36. What’s the difference between F.relu and nn.ReLU ?

F.relu is a Python function for the relu activation function. On the other hand, nn.ReLU is a PyTorch module class that can be called similarly to a function.

37. The universal approximation theorem shows that any function can be approximated as closely as needed using just one nonlinearity. So why do we normally use more?

With a deeper model (that is, one with more layers) we do not need to use as many parameters; it turns out that we can use smaller matrices with more layers, and get better results than we would get with larger matrices, and few layers.