Understanding PyTorch (1 point)

Read the first four tutorials on this page ("*What is PyTorch?*", "*Autograd: Automatic Differentiation*", "*Neural Networks*", and "*Training a Classifier*") and then answer the following questions.

1. (0.25 points) What is a tensor? How is it different than a matrix?

   The basic idea, though, is that a matrix is just a 2-D grid of numbers.

   A tensor is often thought of as a generalized matrix. That is, it could be a 1-D matrix (a vector is actually such a tensor), a 3-D matrix (something like a cube of numbers), even a 0-D matrix (a single number), or a higher dimensional structure that is harder to visualize. The dimension of the tensor is called its rank.

2. (0.5 points) What is automatic differentiation? How does it relate to gradient descent and back-propagation? Why is it important for PyTorch?

   The autograd package provides automatic differentiation for all operations on Tensors. It is a define-by-run framework, which means that your backprop is defined by how your code is run, and that every single iteration can be different. torch.Tensor is the central class of the package. If you set its attribute .requires_grad as True, it starts to track all operations on it. When you finish your computation you can call .backward() and have all the gradients computed automatically. The gradient for this tensor will be accumulated into .grad attribute.

   To stop a tensor from tracking history, you can call .detach() to detach it from the computation history, and to prevent future computation from being tracked.

   To prevent tracking history (and using memory), you can also wrap the code block in with torch.no_grad():. This can be particularly helpful when evaluating a model because the model may have trainable parameters with requires_grad=True, but for which we don’t need the gradients.

There's one more class which is very important for autograd implementation - a Function.

Tensor and Function are interconnected and build up an acyclic graph, that encodes a complete history of computation. Each tensor has a .grad_fn attribute that references a Function that has created the Tensor (except for Tensors created by the user - their grad_fn is None).

If you want to compute the derivatives, you can call .backward() on a Tensor. If Tensor is a scalar (i.e. it holds a one element data), you don't need to specify any arguments to backward(), however if it has more elements, you need to specify a gradient argument that is a tensor of matching shape.

3. (0.25 points) Why does PyTorch have its own tensor class (`torch.Tensor`) when it is extremely similar to numpy's `np.ndarray`?

   Tensors are similar to NumPy's ndarrays, with the addition being that Tensors can also be used on a GPU to accelerate computing.

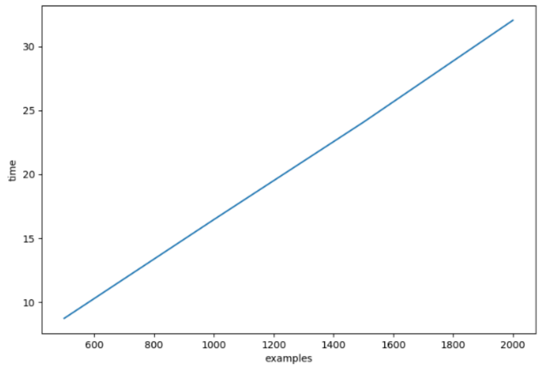
Training on MNIST (2 points)

Let us train neural networks to classify handwritten digits from the MNIST dataset and analyze the accuracy and training time of these neural networks.

Build the model architecture: Create a neural network with two fully connected (aka, dense) hidden layers of size 128, and 64, respectively. Your network should have a total of four layers: an input layer that takes in examples, two hidden layers, and an output layer that outputs a predicted class (10 possible classes, one for each digit class in MNIST). Your hidden layers should have a ReLU activation function. Your last (output) layer should be a linear layer with one node per class (in this case 10), and the predicted label is the node that has the max value. *Hint: In PyTorch the fully connected layers are called `torch.nn.Linear()`.
Use these training parameters: When you train a model, train for 100 epochs with batch size of 10 and use cross entropy loss. In PyTorch's `CrossEntropyLoss` class, the softmax operation is built in. Use the SGD optimizer with a learning rate of 0.01.
Making training sets: Create one training set of each of these sizes {500, 1000, 1500, 2000} from MNIST. Note that you should be selecting examples in such a way that you minimize bias, i.e., make sure all ten digits are equally represented in each of your

training sets. To do this, you can use `load_mnist_data` function in `load_data.py` where you can adjust the number of examples per digit and the amount of training / testing data.
*Hint: To read your MNIST dataset for training, you may want to use a PyTorch `DataLoader`, but this is not required. If you do want to use it with your numpy NMIST dataset, you should use a custom PyTorch `DataLoader` class. We included the class definition for you in the HW (`MyDataset` in `my_dataset.py`) You can see more details about using custom dataset in this blog or github repo)
Train one model per training set: Train a new model for each MNIST training set you created and test it on the MNIST testing subset. Use the same architecture for every model. For each model you train, record the loss function value every epoch. Record the time required to train for 100 epochs. From python's built in `time` module, use `time.time()`.

4. (0.5 points) Given the data from your 4 trained models, create a graph that shows the amount of training time along the y-axis and number of training examples along the x-axis.

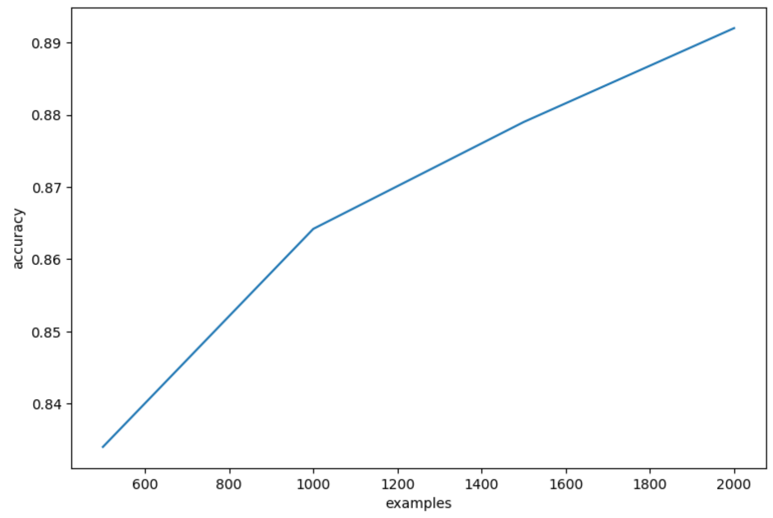


5. (0.5 points) What happens to your training time as the number of training examples increases? Roughly how many hours would you expect it to take to train on the full MNIST training set using the same architecture on the same hardware you used to create the graph in question 4?

   More examples, spend more training time.

   Total 60000 training examples. I expect spending about 15 minutes to train, that is about ¼ hour.

6. (0.5 points) Create a graph that shows classification accuracy on your testing set on the y-axis and number of training examples on the x-axis.



7. (0.5 points) What happens to the accuracy as the number of training examples increases?

   With more training examples, accuracy will be higher.

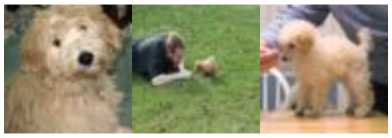
Exploring DogSet (1 point)

DogSet is a subset from a popular machine learning dataset called ImageNet (more info here and here) which is used for image classification. The DogSet dataset is available here. (Note: you need to be signed into your `@u.northwestern.edu`google account to view this link). As it name implies, the entire dataset is comprised of images of dogs and labels indicating what dog breed is in the image. The metadata, which correlates any particular image with its label and partition, is provided in a file called `dogs.csv`. We have provided a general data loader for you (in `data/dogs.py`), but you may need to adopt it to your needs when using PyTorch. **Note: You may need to use the dataset class we provided in MNIST questions if you want to use a PyTorch `DataLoader`**
Validation sets: Thus far, you have only used "train" and "test" sets. But it is common to use a third partition called a "validation" set. The validation set is used during training to determine how well a model generalizes to unseen data. The model does *not* train on examples in the validation set, but periodically predicts values in the validation set while training on the training set. Diminishing performance on the validation set is used as an early stopping criterion for the training stage. Only after training has stopped is the testing set used. Here's what this looks like in the context of neural networks: for each epoch a model trains on every example in the training partition, when the epoch is finished the model makes predictions for all of the examples in the validation set and a loss is computed. If the difference between the calculated loss for this iteration and the

previous is below some *epsilon* for *N* number of epochs in a row, then training stops and we move onto the testing phase.

8. (0.5 points) In Dogset, how many are in the `train` partition, the `valid` partition and the `test` partition? What is the color palette of the images (greyscale, black & white, RBG)? How many dog breeds are there?
   Training data 7665
   Testing data 555
   Validation data 2000
   RBG
   10 dog breeds

9. (0.5 points) Select one type of breed. Look through variants of images of this dog breed. Show 3 different images of the same breed that you think are particularly challenging for a classifier to get correct. Explain why you think these three images might be challenging for a classifier.

   The dog occupies different size of that image. One only with little background noise. Another has more. And the other has most background noise of the picture.

Training A Model on DogSet (2 points)

Build the model architecture: Create a neural network with two fully connected (aka, dense) hidden layers of size 128, and 64, respectively. Note that you should be flattening your *NxNxC* image to 1D for your input layer (where *N* is the height/width, and *C* is the number of color channels). Your network should have a total of four layers: an input layer that takes in examples, two hidden layers, and an output layer that outputs a predicted class (one node for each dog class in DogSet). Your hidden layers should have a ReLU activation function. Your last (output) layer should be a linear layer with one node per class, and the predicted label is the node that has the max value.

Use these training parameters: Use a batch size of 32 and using cross entropy loss. Use the SGD optimizer with a learning rate of 0.001.

When to stop training: Stop training after 100 epochs or when your validation loss doesn't decrease by more than 1e-4 for 3 epochs in a row., whichever happens first.

Training, Testing, Validation sets: You should use training examples from `train` partition of DogSet. Validation should come from the `valid` partition and testing examples should come from the `test` partition.

10. (0.5 points) How many connections (weights) does this network have?

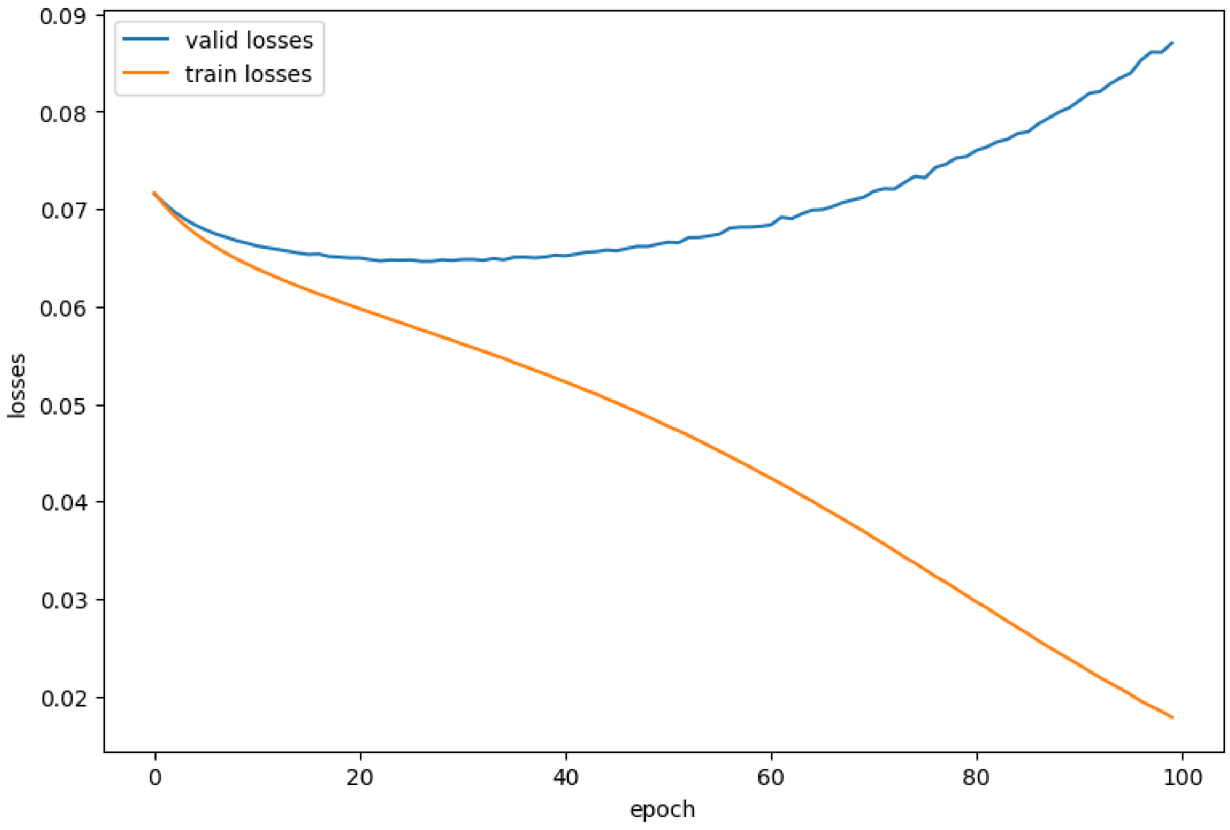
    64*64*3 *128 + 128* 64 + 64 * 10

11. (1.0 point) Train a model on DogSet. After every epoch, record three things: the loss of your model on the training set, the loss of your model on the validation set, and the accuracy of your model on the validation set.

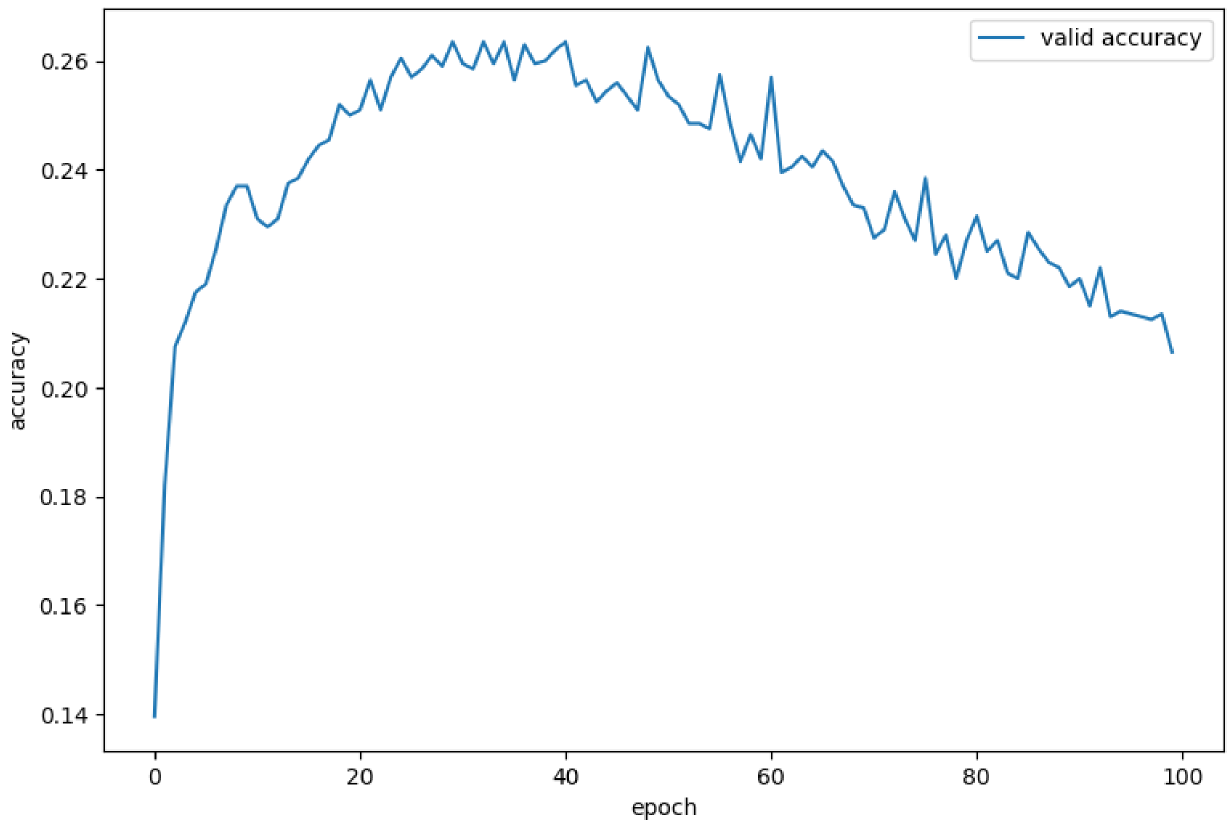
    - Report the number of epochs your model trained, before terminating.

      100th , from 0 to 99

    - Make a graph that has both training and validation loss on the y-axis and epoch on the x-axis.



    - Make a graph that has the validation accuracy on the y-axis and epoch on the x-axis.



- Report the accuracy of your model on the testing set.

  0.21981981981981982

12. (0.5 points) Describe the interaction between training loss, validation loss and validation accuracy. When do you think your network stopped learning something meaningful to the problem? Why do you think that? Back up your answer by referring to your graphs.

    In the beginning, training loss, and validation loss decrease and validation accuracy increase.

    After overfit, training loss keep increase, validation loss start to decrease, and validation accuracy decrease later.

    When validation loss stop to decrease and start to increase, in this case that means overfitting.

    Network should stop when validation loss start to increase.

Convolutional Layers (2 points)

Convolutional layers are layers that sweep over and subsample their input in order to represent complex structures in the input layers. For more information about how they work, see this blog post. Don't forget to read the PyTorch documentation about Convolutional Layers (linked above).

13. (0.5 points) Convolutional layers produce outputs that are of different size than their input by representing more than one input pixel with each node. If a 2D convolutional layer has `3` channels, batch size `16`, input size `(32, 32)`, padding `(4, 8)`, dilation `(1, 1)`, kernel size `(8, 4)`, and stride `(2, 2)`, what is the output size of the layer?

    [16, 3, 17, 23]
    Batch size, channel size, height, and weight
    (32 + 2 * 4 – 7 – 1)/2 +1 = 17
    (32 + 2 * 8 – 3 - 1)/2 + 1 = 23

14. (0.5 point) Combining convolutional layers with fully connected layers can provide a boon in scenarios involving learning from images. Using a similar architecture to the one used in question 10, replace each of your first two hidden layers with a convolutional layers, and add a fully connected layer to output predictions as before. When you call the PyTorch convolutional layer function, leave all of the arguments to their default settings except for kernel size and stride. Determine reasonable values of kernel size and stride for each layer and report what you chose. Tell us how many connections (weights) this network has.

    First convolutional layer

    Kernel size = 5, stride = 2, output channel = 6, weight = 6 * 3 * 5 * 5

    Second convolutional layer

    Kernel size = 5, stride = 2, output channel = 10, weight = 10 * 6 *5 * 5

    Full connected layer

    Weights = 10 * 13 * 13 * 10

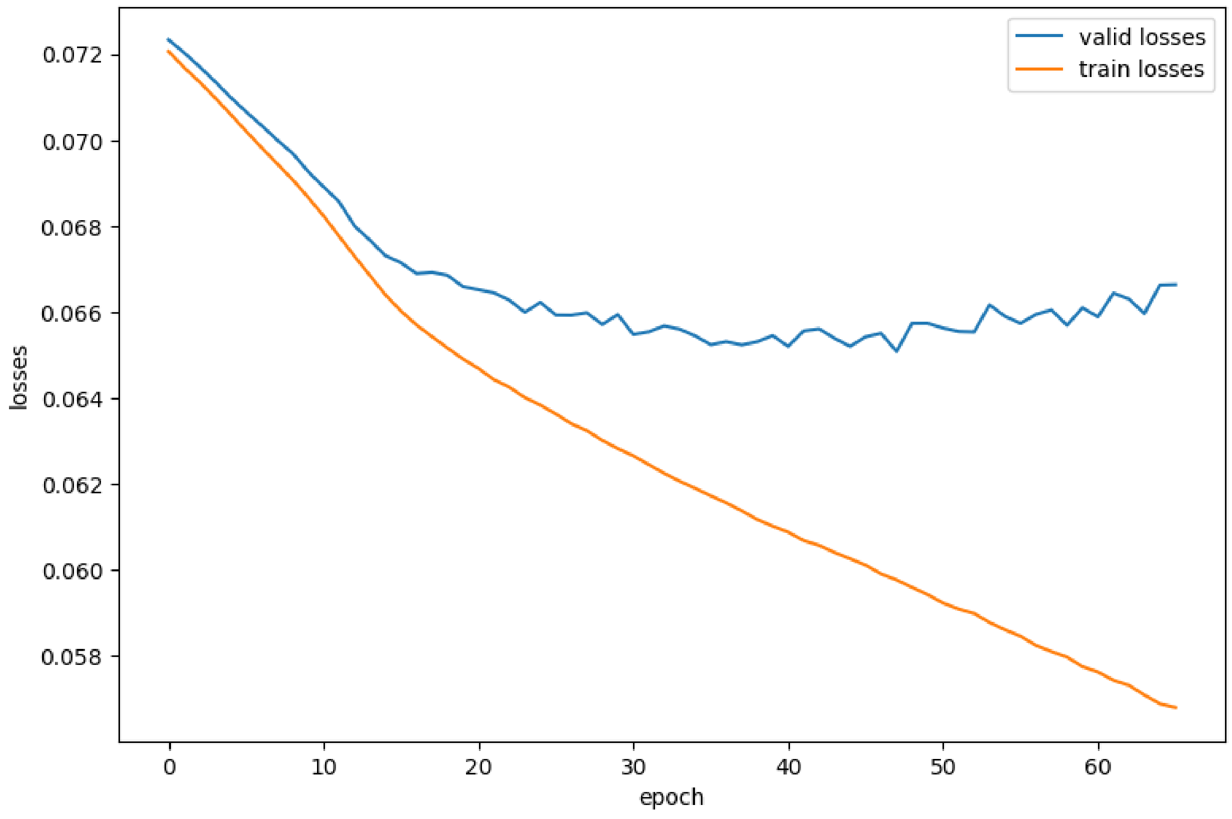
    Total weight = 6 * 3 * 5 * 5 + 10 * 6 *5 * 5 + 10 * 13 * 13 * 10

15. (1 point) Train your convolutional model on DogSet. After every epoch, record three things: the loss of your model on the training set, the loss of your model on the validation set, and the accuracy of your model on the validation set.

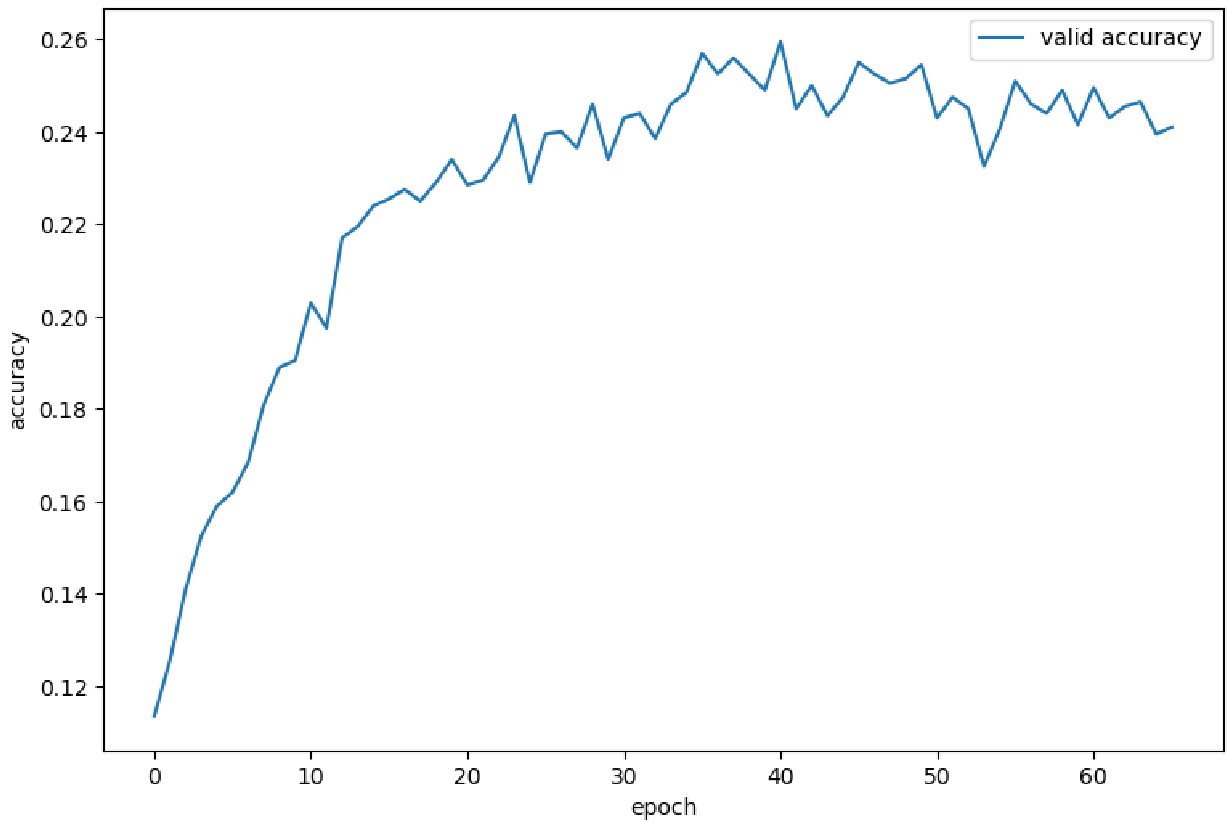
   - Report the number of epochs your model trained, before terminating.

     65, from 0 to 65

   - Make a graph that has both training and validation loss on the y-axis and epoch on the x-axis.



   - Make a graph that has the validation accuracy on the y-axis and epoch on the x-axis.



- Report the accuracy of your model on the testing set.

  0.26126126126126126

Thinking about deep models (2.0 points)

16. (0.5 points) For any binary function of binary inputs, is it possible to construct some deep network built using only perceptron activation functions that can calculate this function correctly? If so, how would you do it? If not, why not?

    No, the binary function can be non-linear function. If so, we need to use other activation functions such as sigmoid, tanh, or relu.

17. (0.5 points) Is it possible to learn any arbitrary binary function from data using a network build only using linear activation functions? If so, how would you do it? If not, why not?

Yes, the combination result of a lot of linear activation functions is still linear activation function. First, we can write the equation. And then we can combine all weights that relate to the same variable and combine them together for the variable.

18. (1 point) An adversarial example is an example that is designed to cause your machine learning model to fail. Gradient descent ML methods (like deep networks) update their weights by descending the gradient on the loss function L(X,Y,W) with respect to W. Here, X is a training example, Y is the true label and W are the weights. Explain how you could create an adversarial example by using the gradient with respect to X instead of W.

    We can use the gradient ascent with respect to X.

    We can pick a data point and keep adding its gradient to it until it shows up in the adversarial part.