RDFIA - Homework 2

Johan Pardo - Shubhamkumar Patel

November 25, 2021

1 Deep Learning Practical Work 2-a and 2-b

1.1 Supervised dataset

Question 1: What are the train, val and test sets used for ?

- **Train**: The train sets is used to train our neural network
- **Val** : The val set is used to have a result of our model and hyper-parameter tuning. Its role is purely to have information about our model, like a checkpoint. This can be useful in the case of early stopping and therefore avoiding overfitting.
- **Test** : The Test set is used to have a true, unbiased result on the performances of our model.

Question 2: What is the influence of the number of examples N ?
The number of examples N influences how much the model can learn from the data set, and therefore the number of features that we may be able to detect. It is better to have more samples for generalization and to avoid over-fitting.

1.2 Network architecture (forward)

Question 3: Why is it important to add activation functions between linear transformations?
The activation functions between linear transformations makes the whole mathematical function non-linear. If we didn't have this functions we would have a simple matrix multiplication which would not work as well for this kind of problems. Furthermore, depending on the activation function, it can set the range of output values, to either have a probability results like the $Softmax$.

Question 4: What are the sizes nx, nh, ny in the figure 1? In practice, how are these sizes chosen?
By reading the figure 1 and understanding the sizes of the input, hidden and output layer we get the following values. $n_x = 2$, $n_h = 4$ and $n_y = 2$.
In practice, we can get this values from the hyperparameter that we chose nx will be equal to the size of the input. ny will be equal to the size of the output. And nh depends on the complexity of the task.

Question 5: What do the vectors $\hat{y}$ and y represent? What is the difference between these two quantities ?

- $\tilde{y}$: It is the output vector of an Affine transformation using a matrix of weights W_y of size $n_y \times n_h$ and a bias vector b_y of size n_y.
- $\hat{y}$: It is the output vector of the transformation applied using an Activation function called SoftMax.
- y : It is the output vector that we want to achieve (Ground truth).

Because $\hat{y}$ is the prediction of the model and y is the ground truth, the difference between $\hat{y}$ and y corresponds to the error of our model.

Question 6: Why use a $SoftMax$ function as the output activation function?

The $SoftMax$ function of the output activation function is used for 2 reasons. The first one is to normalize the output, because the $\tilde{y}$ result can be hard to understand, and the second is to have a probabilistic result.

Question 7: Write the mathematical equations allowing to perform the forward pass of the neural network, i.e., allowing to successively produce $\tilde{h}$, h, $\tilde{y}$ and $\hat{y}$ starting at x

$$\begin{aligned}
\tilde{h} &= W_h x + b_h \\
h &= tanh(\tilde{h}) \\
\tilde{y} &= W_y h + b_y \\
\hat{y} &= Softmax(\tilde{y})
\end{aligned}$$

1.3 Loss Function

Question 8: During training, we try to minimize the loss function. For cross entropy and squared error, how must the $\hat{y}_i$ vary to decrease the global loss function L ?

In order to decrease the global loss function L we need $\hat{y}_i$ to get closer to y.

Question 9: How are these functions better suited to classification or regression tasks?
The cross correlation is better suited for classification because it can handle labels and the Mean Squared Error (MSE) is better suited for regression because it focuses on real numbers and it is a strictly convex function.

1.4 Optimization algorithm

Question 10: What seem to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic and online stochastic versions? Which one seems the most reasonable to use in the general case?

- The advantage of the classic methos with the use of full training set is that we get the gradient from all images so we can converge towards the global minima, but we need to have enough memory to store and use the full image set.

- The online stochastic is the opposite with a good memory usage, but the gradient might not go in the right direction.

- Mini-batch stochastic will quickly reach local minima but will struggle finding the exact minima because of overshooting.

So the best method is the stochastic gradient descent by mini-batch that has a good compromise between both methods.

Question 11: What is the influence of the learning rate η on learning?

If the learning rate is higher we will converge faster towards a local minimum but in the end we might have some convergence issues and in the contrary we will converge slower if the learning rate is lower.

Question 12: Compare the complexity (depending on the number of layers in the network) of calculating the gradients of the loss with respect to the parameters, using the naive approach and the back propagation algorithm

The naive approach of calculating the gradients of the loss with respect to the parameters is much less complex than the back propagation algorithm with a complexity of $O(N)$. The back-propagation algorithm's complexity is proportional to the depth(i.e nb of layer) of the network.

Question 13: What criteria must the network architecture meet to allow such an optimization procedure ?

The criteria is that all layers need to have an activation function that is derivable.

Question 14: The function SoftMax and the loss of cross-entropy are often used together, and their gradient is very simple. Show that the loss can be simplified by: $l(y, \widehat{y}) = -\sum_{i=0} y_i \log \widehat{y}_i$

$$
\begin{aligned}
l(y, \widehat{y}) &= -\sum_{i=0} y_i \log \widehat{y}_i \\
l(y, SoftMax(\tilde{y})) &= -\sum_{i=0} y_i \log \frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}} \\
l(y, SoftMax(\tilde{y})) &= -\sum_{i=0} y_i (\log e^{\tilde{y}_i} - \log \frac{1}{\sum_j e^{\tilde{y}_j}}) \\
l(y, SoftMax(\tilde{y})) &= -\sum_{i=0} (y_i \log e^{\tilde{y}_i} - y_i \log \frac{1}{\sum_j e^{\tilde{y}_j}}) \\
l(y, SoftMax(\tilde{y})) &= -\sum_{i=0} (y_i \tilde{y} + y_i \log \sum_j e^{\tilde{y}_j}) \\
l(y, SoftMax(\tilde{y})) &= -\sum_{i=0} y_i \tilde{y} + \log \sum_j e^{\tilde{y}_j}
\end{aligned}
$$

Question 15: Write the gradient of the loss (cross-entropy) relative to the intermediate output $\tilde{y}$ We know the loss function of the cross-entropy is $l(y, \tilde{y}) = -\sum_i y_i \tilde{y_i} + \log(\sum_i e^{\tilde{y_i}})$

$$
\begin{aligned}
l(y, \tilde{y}) &= -\sum_i y_i \tilde{y_i} + \ \log \sum_i \exp^{\tilde{y_i}} \\
\frac{\partial l}{\partial \tilde{y_i}} &= -y_i + \frac{e^{\tilde{y_i}}}{\sum_k e^{\tilde{y_k}}}
\end{aligned}
$$

Question 16: Using the *backpropagation*, **write the gradient of the** *loss* **with respect to the weights of the output layer** $\nabla_{w_y} l$**. Note that writing this gradient uses** $\nabla_{w_{\tilde{y}}} l$**. Do the same for** $\nabla_{b_y} l$

$$
\begin{aligned}
\frac{\partial l(y, \tilde{y})}{\partial W_{y,i,j}} &= \sum_k (\frac{\partial l}{\partial \tilde{y_k}} \frac{\partial \tilde{y_k}}{\partial W_{y,i,j}}) = \frac{\partial l}{\partial \tilde{y_i}} h_j \\
\frac{\partial l(y, \tilde{y})}{\partial b_{y,j}} &= \frac{\partial l}{\partial \tilde{y_j}}
\end{aligned}
$$

Question 17: Compute other gradients: $\nabla_{\tilde{y}} l$**,**$\nabla_{W_h} l$**,**$\nabla_{bh} l$**,**

$$
\begin{aligned}
\frac{\partial l}{\partial \tilde{h}_i} &= (1 - h_i^2) \sum \delta_{y,j} W_{y,ji} \\
\frac{\partial l}{\partial W_{h,ij}} &= (1 - h_i^2)(\sum \delta_{y,j} W_{y,ji}) x_j \\
\frac{\partial l}{\partial b_{h,i}} &= \delta_i^h
\end{aligned}
$$

2 Convolutional network Work 2-c and 2-d

2.1 Introduction to convolutional networks

Question 1: Considering a single convolution filter of padding p, stride s, and kernel size k, for an input of size, $x * y * z$ what will be the output size?
How much of the weight is there to learn ?,
How much weight would it have taken to learn if a fully-connected layer were to produce an output of the same size ?
- We can get the output size using the following formula : $(\lfloor \frac{x-k+2p}{s} \rfloor)$ x $(\lfloor \frac{y-k+2p}{s} \rfloor) * z$

- We will have (k * k * z + 1) weights to learn.

- If we had a fully connected layer we would have $(x*y*z+1)*(\lfloor \frac{x-k+2p}{s} \rfloor+1)$ x $(\lfloor \frac{y-k+2p}{s} \rfloor+1)*z$ weights to learn.

Question 2: What are the advantages of convolution over fully-connected layers ? What is its main limit ?
The advantage of the convolution over the fully connected layer is the limitation of weights and therefore the limitation of computation power as well.

Question 3: Why do we use spatial pooling ?
The use of spatial pooling is to factorize the image and so limits computation and number of parameters, but we might lose some information.

Question 4: Suppose we try to compute the output of a classical convolutional network (for example the one in Figure 2) for an input image larger than the initially planned size (224 $\times$224 in the example). Can we (without modifying the image) use all or part of the layers of the network on this image ?
We need to add pooling steps as a first layer so that we can have invariance over the input image. Also we can reuse the classical CNN as the size of the filters won't depend on the input image, however for the fully connected layers we won't be able to do anything because the input will not be of the same size as before.

Question 5: Show that we can analyze fully-connected layers as particular convolutions.
A Fully-connected layer of size n can be represented by n convolutions applied using a kernel of size the input of the FC Layer, with no padding and a stride of our choice.

Question 6: Suppose that we therefore replace fully-connected by their equivalent in convolutions, answer again the question 4. If we can calculate the output, what is its shape and interest ?
If we can replace FC by their equivalent in convolutions we would have in variance over the input image dimensions. The shape of the ouput will still depend on the input image. The interest of such output will be that similarly to FC, we could learn non-linear combinations of the features we get from the input image.

Question 7: We call the receptive field of a neuron the set of pixels of the image on which the output of this neuron depends. What are the sizes of the receptive fields of the neurons of the first and second convolutional layers ? Can you imagine what happens to the deeper layers ? How to interpret it ?
The size of the receptive fields of the neurons of the first and second convolution layers are $(k_1)^2$ and $(k_1 + k_2/2)^2$ with k_1 and k_2 the sizes of kernel at depth 1 and 2 respectively. So we can imagine that the receptive fields on the deepest layers can be high or even equal to the input layer. We can interpret it by even if the convolution is local at the end, we still have a global context for the output.

2.2 Training from scratch of the model

Question 8: For convolutions, we want to keep the same spatial dimensions at the output as the input. What padding and stride values are needed ?

If we want to keep the same spatial dimensions at the output as the input, we need to have a padding of size 2 and a stride of size 1.

Question 9: For max poolings, we want to reduce the spatial dimensions by a factor of 2. What padding and stride values are needed ? If we want to reduce the spatial dimensions by a factor of 2 we need a kernel of size 2, padding of size 0 and a stride of size 2.

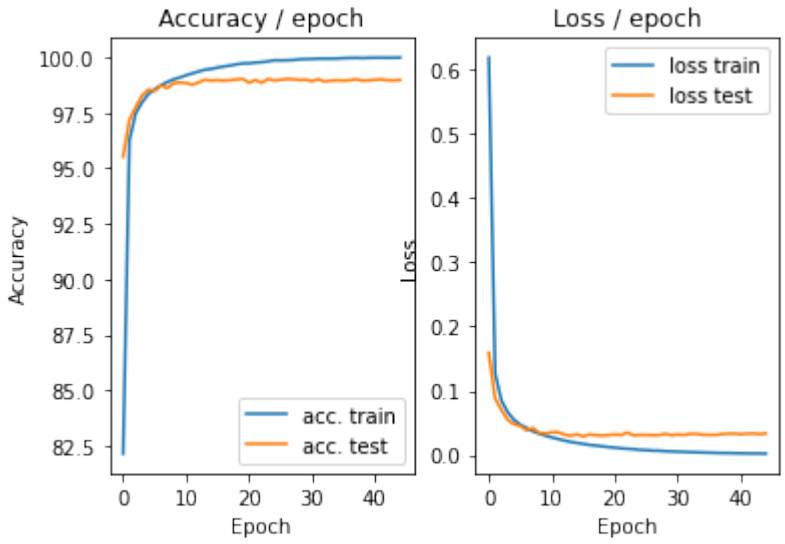
Question 10: For each layer, indicate the output size and the number of weights to learn. Comment on this repartition.

- conv1: shape is 32, 32, 32 and 2432 weights
- pool1 shape is 32, 16, 16 with 0 weights
- conv1: shape is 64, 16, 16 with 51264 weights
- pool2: shape is 64, 8, 8 with 0 weights
- conv3: shape is 64, 8, 8 with 102464 weights
- pool3: shape is 64, 4, 4 with 0 weights
- fc4: shape is 1000 with 1025000 weights
- fc5: shape is 10 with 10010 weights

Question 11: What is the total number of weights to learn ? Compare that to the number of examples.
The number of weights is 1191170 which is way bigger than the number of examples in the MNIST dataset.

Question 12: Compare the number of parameters to learn with that of the BoW and SVM approach.
The number of parameters varies greatly between each method from 1 million for our model to only 1000 for the SVM approach with BoW in between with around 100,000.

Question 13: Read and test the code provided. You can start the training with this command : main (batch_size, lr, epochs, cuda = True)



We can see that during the first epochs the accuracy and loss in the train and test dataset converge pretty quickly but then stabilize at around 98% and 0.05 respectively afterwards.

Question 14: In the provided code, what is the major difference between the way to calculate loss and accuracy in train and in test (other than the difference in data) ?
The major difference in the way to calculate loss accuracy in train and in test is the lack of back propagation in previous layers.

Question 15: Modify the code to use the CIFAR-10 dataset and implement the architecture requested above. (the class is datasets.CIFAR10). Be careful to make enough epochs so that the model has finished converging.

```
class ConvNet(nn.Module):
    """
    This class defines the structure of the neural network
    """

    def __init__(self):
        super(ConvNet, self).__init__()
        # We first define the convolution and pooling layers as a features
    extractor
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, (5, 5), stride=1, padding=2),
```

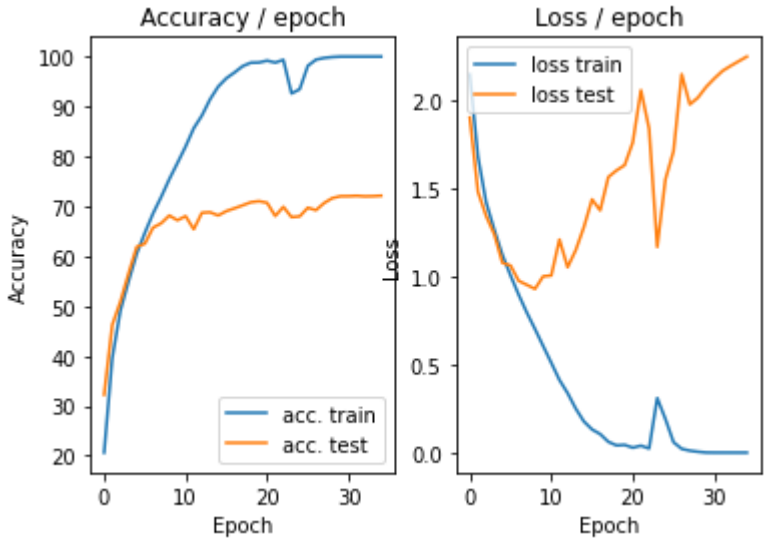
```
            nn.ReLU(),
            nn.MaxPool2d((2,2), stride=2, padding=0),
            nn.Conv2d(32, 64, (5, 5), stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d((2, 2), stride=2, padding=0),
            nn.Conv2d(64, 64, (5, 5), stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d((2, 2), stride=2, padding=0, ceil_mode = True),

        )
        # We then define fully connected layers as a classifier
        self.classifier = nn.Sequential(
            nn.Linear(1024, 1000),
            nn.ReLU(),
            nn.Linear(1000, 10)
            # Reminder: The softmax is included in the loss, do not put it
    here
        )

def get_dataset(batch_size, cuda=False):
    """
    This function loads the dataset and performs transformations on each
    image (listed in `transform = ...`).
    """
    train_dataset = datasets.CIFAR10(PATH, train=True, download=True,
        transform=transforms.Compose([
            transforms.ToTensor()
        ]))
    val_dataset = datasets.CIFAR10(PATH, train=False, download=True,
        transform=transforms.Compose([
            transforms.ToTensor()
        ]))

    train_loader = torch.utils.data.DataLoader(train_dataset,
                        batch_size=batch_size, shuffle=True, pin_memory=cuda,
     num_workers=2)
    val_loader = torch.utils.data.DataLoader(val_dataset,
                        batch_size=batch_size, shuffle=False, pin_memory=cuda
    , num_workers=2)

    return train_loader, val_loader
```

Listing 1: Python example

Question 16: What are the effects of the learning rate and of the batch-size ?
The effects of the learning rate is the speed at which we want to converge. And the effect of the batch-size is how close to the true gradient we want to stay by limiting the error by taking an average of all loses in the batch.

Question 17: What is the error at the start of the first epoch, in train and test ? How can you interpret this ?
The error at the start of the first epoch is around 10% It is due to the random initialization of the parameters that we do during the first epoch.

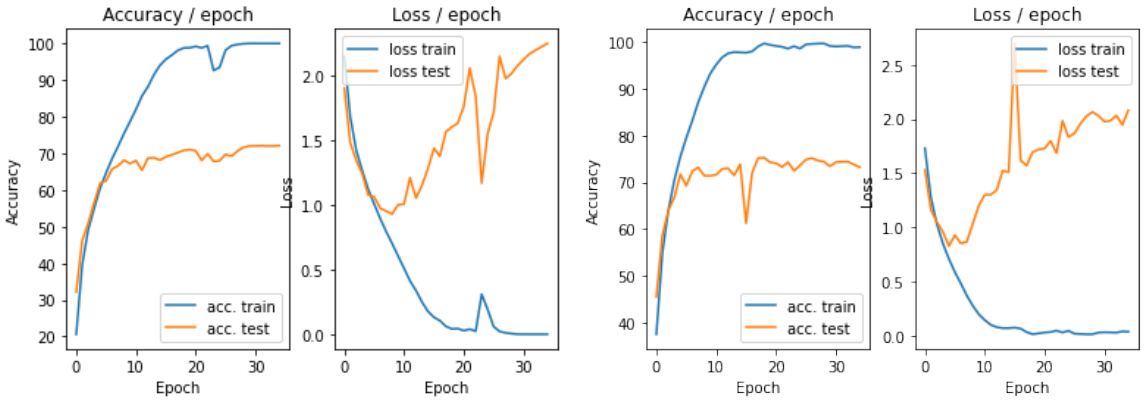
Question 18: Interpret the results. What's wrong ? What is this phenomenon ?



We can see that the accuracy in the test dataset doesn't follow the accuracy in the train dataset. This phenomenon is called overfitting.

2.3 Results improvements

Question 19: Describe your experimental results.



We can see that by using a normalization in the pre-processing, the train loss is much smoother and it drop quicker (10 epochs) than in the previous results (Left) and the test loss doesn't get as high as before. This shows that introducing the standardization we are able to influence the loss as well as the accuracy as we are able to reach optimal or good results straight from the beginning.

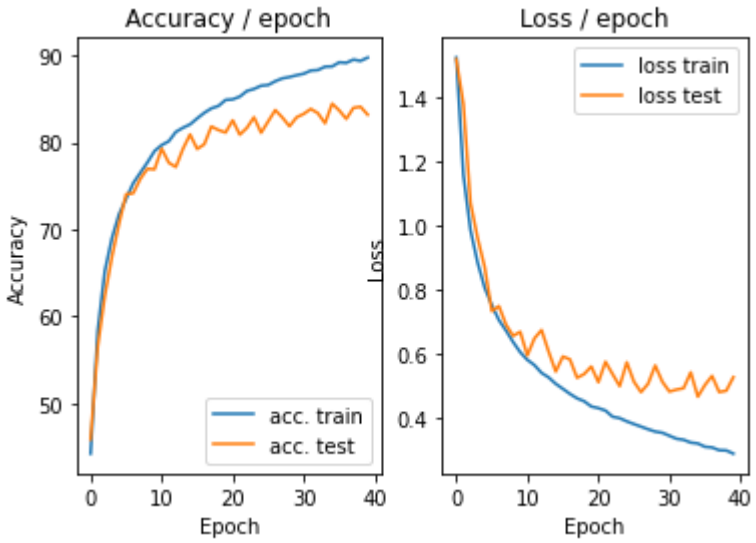
Question 20: Why only calculate the average image on the training examples and normalize the validation examples with the same image ?

We should only calculate the average image on the training examples, as we don't want the

validation examples to avoid having results that are biased by the test data. And we should normalize the validation examples with the same image because we are supposed this to test our model during validation to test our model more accurately.

Question 21: There are other normalization schemes that can be more efficient like ZCA normalization. Try other methods, explain the differences and compare them to the one requested.
The ZCA normalizing is greatly use in computer vision because it keeps in mind the detection of edges, by using neighbours values, during it's normalization function which are key features in images

Question 22: Describe your experimental results and compare them to previous results.



We can see that the data augmentation can minimize the over-fitting problem that we had before.

Question 23: Does this horizontal symmetry approach seems usable on all types of images ? In what cases can it be or not be ?
The horizontal symmetry approach doesn't seem usable on all types of approach, for example when we want to determine the direction in which a plane is going during the night the position relative to each other of the green and red light let us know in which direction it is going. But in other cases where relative position to each other is not so important, like labelization of cats and dogs, this can be useful.

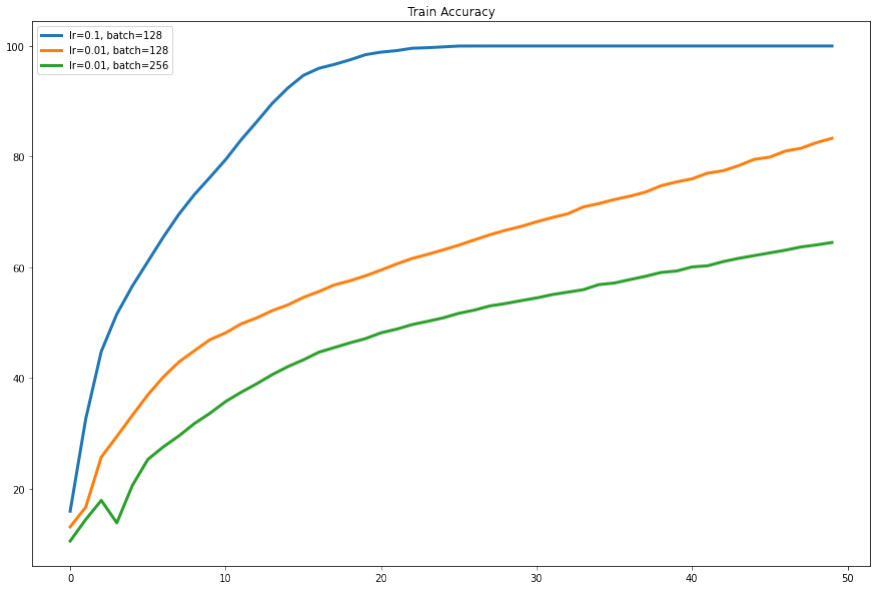
Question 24: What limits do you see in this type of data increase by transformation of the dataset ?
This increase of data transformation doesn't have the same "power" than a new labeled image. Arguably it won't generalize as efficiently as introducing new unseen data would.

Question 25: Other data augmentation methods are possible. Find out which ones and test some.
The other augmentation methods possible are:

- compose
- crop
- gray-scale
- pad
- flip
- zoom
- rotation
- ..., others *Data augmentations*

In general the data augmentation improves accuracy and helps generalizing our model but some of them are better like the rotation.

Question 26: Describe your experimental results and compare them to previous results, including learning stability.

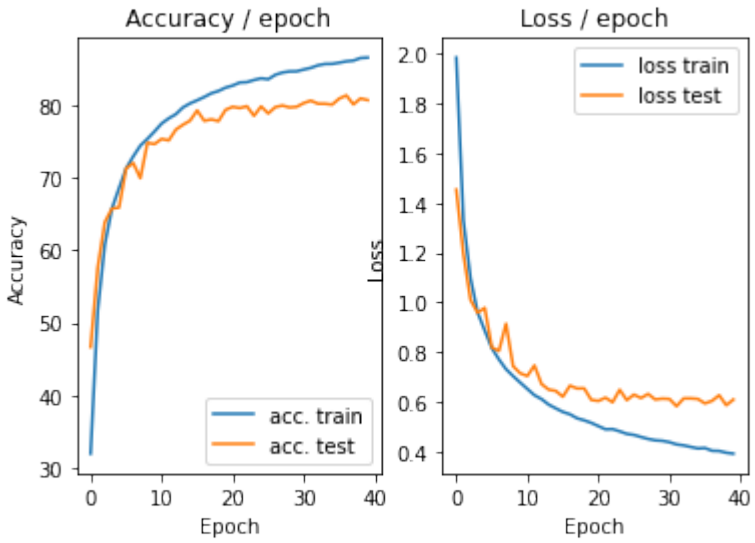


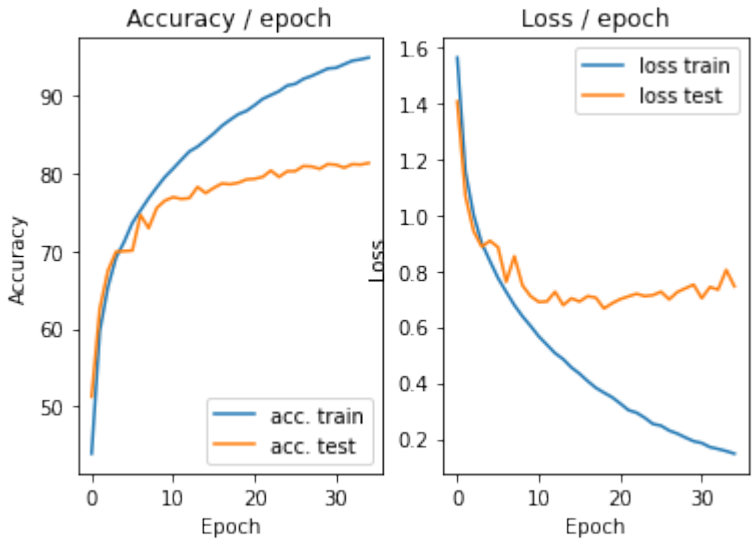
By decreasing the batch size we are able to see fewer data (so features) during training and this results in better accuracy. For the learning rate, we get bad accuracy if the values are too high(≥ 1). If they are too low we get better results but not the best (≤ 0.001). The best result we get are in between too big and too small lr values ($= 0.1$). The change in momentum also allows us to optimize the behaviour of the convergence, hence improving the overall learning.

Question 27: Why do these two methods improve learning ?
The two methods improve learning because for once we can adapt the learning rate at each epoch so that we can converge faster from the beginning and than reduce the lr value to converge smoothly towards the minimum value during the gradient descent.

Question 28: Many other variants of SGD exist and many learning rate planning strategies exist. Which ones ? Test some of them.

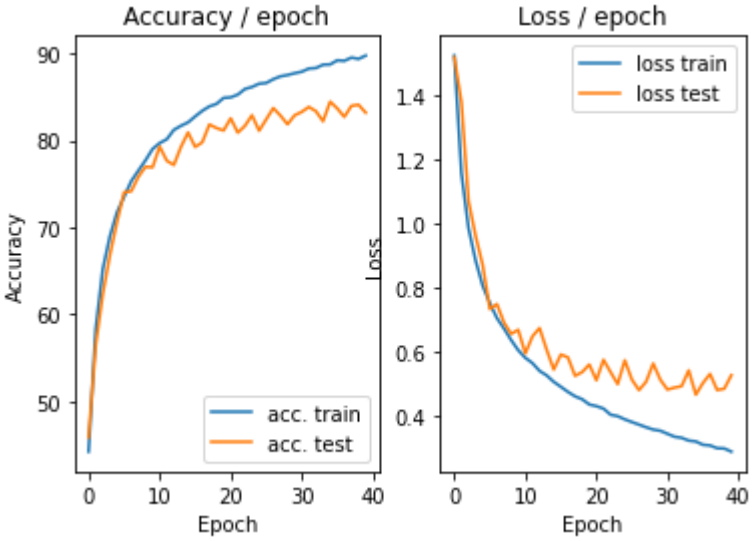
- ADAM
- ASGD
- SGD with momentum
- LBFGS
- ..., other optimizer

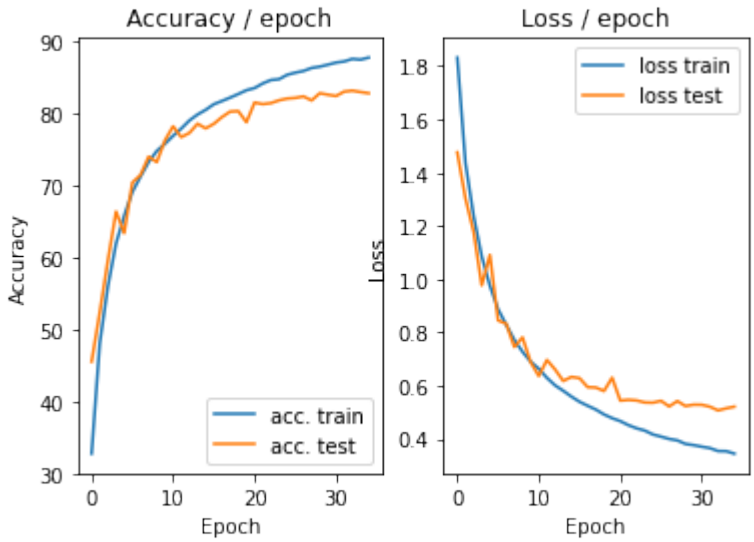




We can see that the Adam optimizer accuracy (First Figure) is quite good from the beginning for both train and test, compared to SGD with momentum and Exponential lrscheduler (Second Figure) which takes more than 30 epoch to get to the same results and yet the loss is higher than Adam.

Question 29: Describe your experimental results and compare them to previous results.





After adding the dropout (Second Figure) we can see that compared to the previous results (First Figure) we have significantly reduced over-fitting. The loss stays more consistent between train and test and there are less fluctuations while we reach a plateau for the loss as well as for the accuracy.

Question 30: What is regularization in general ?
Regularization is the act to balancing or penalizing the complexity of the model in order to avoid over-fitting.

Question 31: Research and "discuss" possible interpretations of the effect of dropout on the behavior of a network using it ?
The dropout try to add randomness in the neural network by removing activation during its creation and by doing so can limit that multiple node in the same layer learn the same things.

Question 32: What is the influence of the hyperparameter of this layer ?
The hyperparameter for the dropout layer is the probability of a neuron to be deactivated during training. If the probability is too high, we risk having under-learning issues, whereas if it is too low, we may have over-learning issues.

Question 33: What is the difference in behavior of the dropout layer between training and test ?
According to this blog dropout is a technique that prevents over-fitting in artificial neural networks by randomly dropping units during training. As a result, the trained model works as an ensemble model consisting of multiple neural networks. At test time, the prediction of those assembled networks is averaged in every layer to get the final model prediction.

Question 34. Describe your experimental results and compare them to previous results.