# CS 7150: Deep Learning — Summer-Full 2020 — Paul Hand

Week 4 — Preparation Questions For Class
Due: Monday June 1, 2020 at 12:00 PM Eastern time via Gradescope

Name: Saurabh Vaidya
Collaborators: Sumeet Gajjar

You may consult any and all resources in answering the questions. Your goal is to have answers that are ready to be shared with the class (or on a hypothetical job interview) as written. **Make sure to tag each question when you submit to Gradescope.**

**Directions:** Read the articles 'ImageNet Classification with Deep Convolutional Neural Networks' (AlexNet) and 'Deep Residual Learning for Image Recognition' (ResNets) .

**Question 1.** *What is dropout? What evidence is there that it works? Why does it work?*

**Response:**Dropout is a technique of setting the output of neurons in a layer to 0 with a probability p.

As seen in the original paper *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, they carried out experiments on networks with and without dropout to compare their accuracies. They trained their models on variety of datasets such as MNIST, Cifar-10, Cifar-100, ImageNet and other speech and genetics datasets too to show its applicability to all problems and not just vision. As seen in their results, dropout networks performed better than their non-dropout counterparts on almost all datasets. Thus providing ample evidence which was model-invariant, dataset-invariant and problem-invariant, that dropout works.

Dropout works because it reduces the co-dependence of output neurons on input neurons in previous layer during forward and backwards pass. Since the number of 'active' neurons is changing during training, neurons are also capable to learning in conjunction with different random subset of neurons. This reduces co-adaptations- a phenomenon where stronger connections are formed between certain neurons while weaker connections with others thereby leading to weaker connections to be forgotten at the end of training. Overall, dropout makes network more robust.

Another reason dropout works is because it provides the same effect as ensemble of different models. At every training input, since some neurons become *'dead'*, the architecture of the model changes which becomes equivalent to training different models. At test time, we average out the output of each neuron by multiplying it by 0.5

**Question 2.** *What is weight decay? Why would one use it?*

**Response:**Weight decay is basically a form of regularization where we penalize large weights and make sure weights are evenly distributed across all features. In essence, each weight is multiplied by a scalar and that scalar is reduces the magnitude of weight before its updated.

Weight decay becomes equivalent in SGD optimization where the weight decay is actually $wd = 2\lambda$, $\lambda$ is regularization constant. Thus weight decay of 0.0005 actually means that

their regularization constant $\lambda = 0.00025$. It can be seen below the weight update rule with regularization is

$$\widehat{L} = L + \lambda \|w\|^2 \tag{1}$$

$$\tag{2}$$

Thus after taking derivative our update rule becomes

$$w_{i+1} = w_i - 2\lambda\epsilon w_i - \epsilon \frac{\partial \widehat{L}}{\partial w} \tag{3}$$

Where $\epsilon$ is learning rate.

This is the same formula used in the alexnet paper to update weights.

**The equivalence between weight decay and l2 regularization is however not true when using weight decay regularization in solvers such as Adam or Adadelta as shown by Loshchilov et al in their paper**

Weight decay is used to prevent network from overfitting. In Alexnet weight decay also helped improve the training error.

**Question 3.** *The AlexNet paper used a learning rate schedule where the learning rate was lowered when validation error stopped improving. Why is it reasonable to have a schedule where learning rate decreases? Why wait until validation error stops improving?*

**Response:**Learning rate multiplies magnitude of the gradient. It itensifies the size of step we will take in the direction of gradient during training. A high learning rate would mean we take larger strides through our loss function landscape. Having a rate schedule where the learning rate reduces in time, allows us to take smaller steps as we approach minima.

A justification for why we should wait until validation error stops is that if validation error is changing then we are still approaching flat region/minima. But when validation error stops improving then either we have reached a minima and thus we cannot improve further or we might be oscilatting back and forth around the minima. In the former case, we do not want to shoot over the minima by continuing to take large step. Thus we would lower our learning rate to prevent from skipping over the minima. The latter would mean we are constantly taking steps with same magnitude that are on either side of our minima, thereby jumping over the minima.Thus by reducing the magnitude of our learning rate we will take smaller steps in our loss function landscape. This way we can prevent skipping over the minima or even if go past the minima, it will be with a small step relative to previous iteration. In latter case, validation error would plateau again we would reduce learning rate further till we reach our minima/flat region of our loss function.

**Question 4.** *The authors in the AlexNet paper said that the kernels on one GPU were "largely color-agnostic," whereas the kernels on the other GPU were largely "color-specific." Further, they said this specialization occurs "during every run and is independent of any particular random weight initialization." Can you explain why this would be more likely to happen than having the color-agnostic and color-specific features interspersed on each GPU?*

**Response:**Group filters lead to more structured relationships learned between filters of adjacent layers. Filters in next layer are only dependent on filters in previous layer from same group. This reduces interdependency of filters in one layer to all filters in previous layers. This leads to high inter-layer correlation among filters in adjacent layers from same group and low inter-layer correlation among filters of other group. This is shown in a paper *Deep Roots: Improving CNN Efficiency with Hierarchical Filter Groups*.

Due to this strong inter-layer filter correlation within same group, backpropagation leads to weight updates in filters such that this strong inter-layer filter correlation property is maintained. So when n number of filters in layer i in group g has its gradients calculated, the gradients of filters of layer i-1 only depend on layer i of same group, layer i-2 depend on layer i-1 and so on but within same group.

In doing so what ends up happening is there is a distribution of responsibility across both groups for a layer. However, if there were no interactions between the two groups at all there would not be a singular common task whose responsibility is to distributed but would rather be two different tasks. Since alexnet has some positions where the two groups interact, it is at those positions where backpropagation distributes the error gradient between two groups. Thus we see that in AlexNet, two groups in initial layer take responsibility of being color-specific and color-agnostic.

**Question 5.** *Explain the different data augmentation strategies used in the AlexNet paper? What do these strategies accomplish?*

**Response:**The firs data augmentation startegy used is to translate and horizantally flip images. They extract 224x224 patches from original images and their horizontal reflections to create more training examples.

The second strategy is to alter the intensities of each pixel across RGB channels. They achieve this by computing PCA by eigenvalues, eigenvectors decomposition for each image. Then they add these eigenvectors with magintude of their eignevalue times some random number drawn from Gaussian distribution with 0 mean and 0.1 standard deviation.

Common goal of both strategies is to improve error rate. The firt strategy specifically makes it possible by making the model position and reflection invariant. The second strategy makes the model color invariant.

**Question 6.** *The ResNet paper reports 3.57% error on the ILSVRC. Some people would claim this performance is superhuman. Look up the rate of error achieved by humans. Why is the human error rate not 0%? (After all, wasn't it labelled by humans?) Do you think it is fair to say that this net can achieve superhuman performance at image classification?*

**Response:**The human error rate is not 0 because the data collection was not done a single human. ImageNet gives a search query for a particular class and humans are only presented with results of search engine to filter out noise and asked binary question of whether this image is class *'A'*. Search engine results are themselves tagged by other humans who are domain experts on those classes.

Thus a human like us, who does not have deep knowledge about say species of dogs, can barely distinguish between a border collie and australian shepherd dog. Thus humans are

bound to make errors on some prediction. In one study, its found that error rate of humans is 5.1%. Resnets can be considered to be superhuman, whereby they can outperform an average human being at classification. However it remains to be seen if humans are trained with the entire dataset, whether they can be 100% accurate or is it not possible.

**Question 7.** *Explain Figure 4 of the ResNet paper, including what the context is, what is being plotted, what is being observed, and what is being concluded. Make sure to explain why there are two sudden steep drops in error % in both plots.*

**Response:**Degradation problem is a phenomenon where deeper networks have their training accuracy first saturate and then start degrading. The authors want to prove that deeper residual networks tackle degradation problem versus plain deeper networks that show degradation problem. Two types of networks with two model in each type are evaluated on ImageNet dataset. One type of networks are residual networks while other have no residual connections(plain network).Both types have same number of layers, convolutions, feature maps and learning parameters. Only difference is residual blocks with shortcut connections.

Fig 4 shows the training and validation errors of 18-layer and 34-layer residual and plain networks.

The observations made are:

- For plain networks, the 34-layer network has higher validation and training error throughout the training. The training error is always more than validation error for both networks.
- For residual networks, the 34-layer network is instead better than the 18-layer one in both training and validation error throughout the training.
- the 34-layer residual network has lower training error and better accuracy than its plain counterpart.
- While 18-layer plain and residual networks have comparable accuracy, the residual network converges faster.
- In residual networks, the training error does gets lower than validation error towards the end of training, something not seen in plain networks at all.

When the network is not overly deep(18 layers in our case), SGD can still find better solutions thus the performance of residual and plain networks for 18-layer versions are similar.

The degradation problem is observed in plain network as the network gets deeper as seen in first point of observations. However in residual networks, the deeper network performs better than its shallower version suggesting residual networks seem to be handling the degradation problem. Thus increasing depth is giving increased performance.

As seen in both graphs in fig 4, there are two instances of sudden steep drop in error percentages. This is the result of learning rate being reduced just when the errors start to plateau. As we decrease learning rate, we tend to avoid constantly missing the minima and being stuck on either sides of it at each next iteration as also explained in answer to question 3.

**Question 8.** *Estimate the number of weight parameters in the three nets depicted in Figure 3 of the ResNet paper.*

**Response:**
For each layer, the number of weight parameters in that layer is
$w_n = (f_h * f_w * C_{i-1} + b) * C_i$

$w_n$ is number of weight parameters
$f_h$ is filter height
$f_w$ is filter width
$C_{i-1}$ is number of channels(features) in previous layer
$C_i$ is number of channels(features) in current layer.
$b$ is number of bias which is 1

So for example with filter of size 5x5 and number of channels in previous layers is 3 and the number of channels(features) in current layer is 128, the number of weight parameters in current layer would be 5*5*3*128 + 128 = 9728

For Fully connected network(MLP) the number of weight parameters in layer i is $w_i = (n_{i-1} + 1) * n_i$ where $n_{i-1}$ is number of neurons in previous layer plus 1 for bias and $n_i$ is number of neurons in current layer. when we flatten the convolutions before fully connected network, the total number of neurons is hxwxf where h,w is height and width of final convolution activation map and f is number of features maps in last conv layer.

For example if layer 1 has 100 neurons and layer 2 has 50 neurons, the total number of weight parameters in layer 2 would be (100+1)*50 = 5050.

- The VGG network(Fig 4 left) has 143,667,240 weight parameters
- The Plain 34 layer network(Fig 4 middle) has 21,616,232 weight parameters.
- The Residual 34 layer network(Fig 4 right) has same number of parameters as plain 34 layer network since the architecture is same and skipping connections do not add additional parameters of that architecture.