Homework 5
601.482/682 Deep Learning
Fall 2020

October 16, 2020

Due Fri. Oct 16 11:59pm.
Please submit a latex generated PDF
to Gradescope with entry code M63JEZ

1. Computing Network Sizes

   (a) Given the multi-layer perceptron below in Figure 1, compute the number of free parameters for this model (Note that biases are explicitly disregarded in this case).

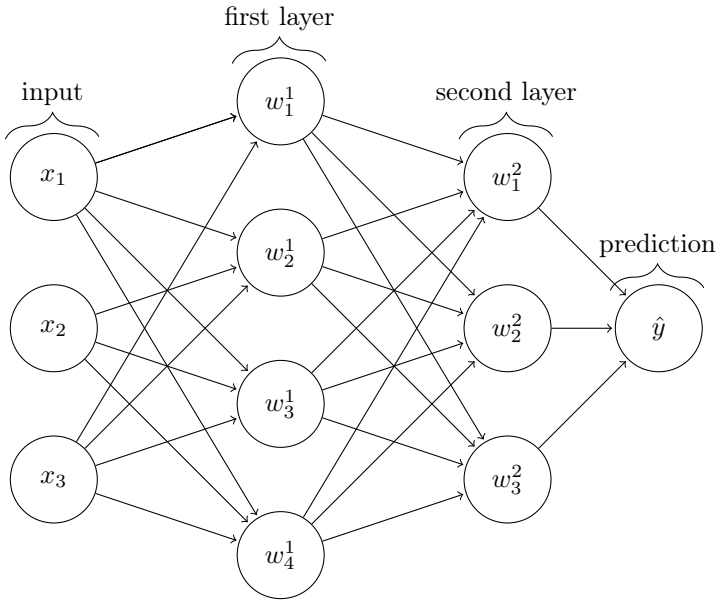

Figure 1: Multi-layer Perceptron

Solution: Since there are no biases, we just need to count the number of connections. There are 11 connections linking the first and second layer, 12 linking the second and third layer, and 3 linking the third layer with the output. So there are a total of 26 parameters.

In this part, you will analyze the size of a state-of-the-art architecture not discussed in class. In the ImageNet ILSVRC 2014 contest, two fairly deep networks performed very well: The VGG network and the GoogLeNet. While VGG performed best in the localization task and ranked second in classification, GoogLeNet won the classification task achieving a top-5 error rate of 6.67%. Figure **??** presents the architecture of GoogLeNet, which is built up of 9 stacked "inception" modules displayed in Figure **??**.

   (b) Consider the layer "inception (3a)" from Table 1(Figure **??**) in the GoogLeNet paper. Notice how "3x3 reduce" and "5x5 reduce" are used between layers - from section 5, this "stands for the number of 1x1 filters in the reduction layer used before the 3×3 and

5×5 convolutions". Compute the number of free parameters for the "inception (3a)" layer.

Solution: The formula for number of parameters is $p = (d^2 * c + 1) * f$, where d is the length of the convolution kernel, c is the number of channels in the input image, and f is the number of output features. We will go through the listed convolutions from left to right:

$$(1^2 * 192 + 1) * 64 = 12352$$
$$(1^2 * 192 + 1) * 96 = 18528 \qquad (3^2 * 96 + 1) * 128 = 110720$$
$$(1^2 * 192 + 1) * 16 = 3088 \qquad (5^2 * 16 + 1) * 32 = 12832$$
$$(1^2 * 192 + 1) * 32 = 6176$$

The total number of parameters is the sum of all the convolution parameters, which is 163696.

(c) Now, consider that the reduction portion of "3x3 reduce" and "5x5 reduce" were omitted (i.e. no 1x1 filters were used before the 3x3 and 5x5 convolutions). Compute the number of free parameters for this updated "inception (3a)" layer. How does this compare to the original layer?

Solution: We can redo our calculations using the same formula:

$$(1^2 * 192 + 1) * 64 = 12352$$
$$(3^2 * 192 + 1) * 128 = 221312$$
$$(5^2 * 192 + 1) * 32 = 153632$$
$$(1^2 * 192 + 1) * 32 = 6176$$

The total number of parameters is the sum of all the convolution parameters, which is 393472. This is about 2.5 times the number of parameters that were used when 1×1 dimensionalality reduction filters were deployed.

2. Receptive Fields

(a) Consider the network in Figure 5, which is constructed by 2 5×5 convolution kernels. What is the receptive field of one pixel in layer 2? Please draw a 2D graph (excluding the channel dimension) to illustrate your calculation.

Solution: One pixel of layer 2 corresponds to a 5×5 patch of pixels in layer 1. Since the convolution between the input and layer 1 has stride 1, there will be overlap between the receptive fields of each pixel in that 5×5 patch. We can consider the edges of that patch to find the full receptive field. Each of the edge pixels will be the center of a 5x5 convolution, so their receptive field will extend 2 pixels outside of the 5×5 patch. This will happen on both sides. So the receptive field of one pixel in layer 2 is $(2 * 2 + 5)^2 = 9 \times 9$

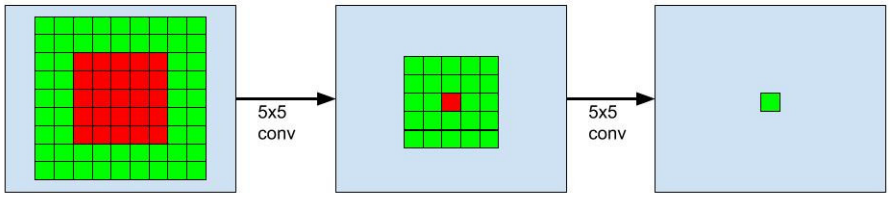

Figure 2: Receptive field illustration. The second layer has 1 pixel in green. The first layer has 5x5 corresponding pixels. The center is in red. The input has 9x9 corresponding pixels. The overlay of the 5x5 in layer 1 is shown in red

(b) Now consider that we add a n-by-n max-pooling layer following layer 2. What will be the receptive field after the max-pooling layer? (The cases where the pooling kernel is on the edge of the feature map can be ignored in this question)
Solution: One pixel of layer 3 corresponds to nxn pixels of layer 2. Since the convolution between layer 1 and 2 has stride of 2, there will be a gap between the centers of convolutions on layer 1 that corresponds to consecutive pixels in layer 2. Since we are concerned with a path of length n pixels in layer 2, there will $n-1$ gaps between each conv center in layer 1. Now we can apply the same analysis as the last part. The receptive field in layer 1 is $(n+(n-1)+2*2)^2 = (2n+3)\times(2n+3)$. Finally, we translate that back into the input, just as we did in the last part. The receptive field is $((2n+3)+2*2)^2 = (2n+7)\times(2n+7)$

3. Gradient Descent Optimization

(a) Consider Figure 6, which depicts a cost function $L(x) : \mathbb{R}^2 \to \mathbb{R}$. The red dot represents the current estimate of $\mathbf{x}_t = [x_1, x_2]$ at time t. Please sketch the next direction of update (one step) that would be taken by vanilla SGD.

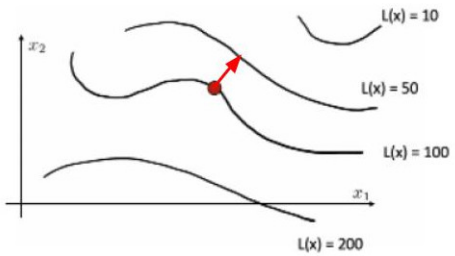

Figure 3: SGD step is supposed to be perpendicular to the contour.

(b) It is worth mentioning that the contour lines shown in Fig. 6 will change during optimization since the loss is evaluated over a single batch rather than the whole dataset. As discussed in class, this observation suggests that for unfortunate updates we might get stuck in saddle points, where the vanilla SGD gradient is 0. One way to combat this problem is to use first and/or second order momentum. Please briefly explain why these moments are helpful and how they would change the update direction sketched in (a).
Solution: Vanilla SGD just uses the current local gradient as the direction of the next step. First and second order momentum factor in information from previous gradient updates to inform the direction of the next step.
This solves a couple problems with vanilla SGD. First, it smoothens out the path of descent. Consecutive steps using local gradients can have large perpendicular components that eventually cancel out, but which slow down convergence. using momentum will help remove those perpendicular components and make convergence more fast and robust. Second, the descent might get stuck in a saddle point or local minimum where there is no local gradient. Having momentum is essentially having a memory of the previous gradients, which you can use to get out of the saddle point.
For the diagram above, the exact direction depends on the direction of approach, since previous gradients are used. However, the gradient would probably be more vertical, since the contour at 200 is more smooth and doesn't have the outward bump of the level set at 100 that the current point is on.