Homework 5
601.482/682 Deep Learning
Spring 2022

March 9, 2022

Due Wednesday, March 16 by 11:59pm
Please submit a latex generated PDF
to Gradescope (Entry Code WYPD4D)

1. Computing Network Sizes

   In this question, you will analyze the size of a state-of-the-art architecture 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 1 presents the architecture of GoogLeNet, which is built up of 9 stacked "inception" modules displayed in Figure 3.

   (a) Consider the layer "inception (3a)" from Table 1 (Figure 2) in the GoogLeNet paper. Notice how "3×3 reduce" and "5×5 reduce" are used between layers - from §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 with dimensionality reduction.
   Answer:
   num of parameter:
   (1x1x192+1)x64+(1x1x192+1)x96+(3x3x96+1)x128+(1x1x192+1)x16+(5x5x16+1)x32
   +(1x1x192+1)x32 = 163696
   = 159k

   (b) Now, consider that the reduction portion of "3×3 reduce" and "5×5 reduce" were omitted (i.e. no 1×1 filters were used before the 3×3 and 5×5 convolutions). Compute the number of free parameters for this naïve "inception (3a)" layer without dimensionality reduction. How does this compare to the original layer?
   Answer:
   num of parameter:
   (1x1x192+1)x64+(3x3x192+1)x128+(5x5x192+1)x32+(1x1x192+1)x32 = 393472
   = 384k
   The parameter size is much larger than the one with '3x3 reduce' and '5x5 reduce'

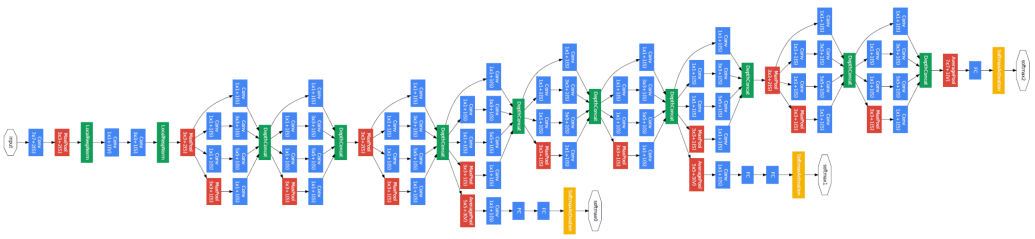
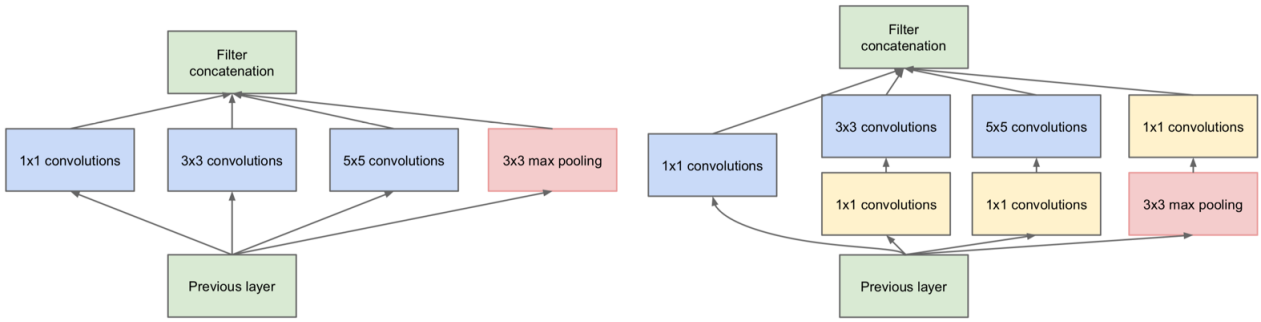


Figure 1: GoogleNet Architecture

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 2: Table 1: GoogLeNet incarnation of the Inception architecture. In the params column, $K = 1024$.



(a) Inception module, naïve version

(b) Inception module with dimensionality reduction

Figure 3: Inception Module

2. Receptive Fields

(a) Consider the network in Figure 4, 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. *Note:* The cases where the convolution kernels are on the edge of the feature map can be ignored in this question.
Answer:
1x1 pixel of output in the final layer comes from 3x3 5x5 pixels of 2nd layer depending on whether the final pixel is in the corner of the output, and which comes from 9x9 pixels of 1st layer. Attached a following graph to show the process.

(b) Now consider adding a n-by-n max-pooling layer following layer 2. What will be the receptive field after the max-pooling layer? *Note:* The cases where the pooling kernel is on the edge of the feature map can be ignored in this question.
Answer:
1x1 pixel of output in the final layer comes from nxn pixels of pooling layer, which comes from [2(n-1)+5]x[2(n-1)+5] of pooling layer, which comes from [2(n-1)+5]-1+5x[2(n-1)+5]-1+5 equals to [2n+7]x[2n+7] pixels of 1st layer.

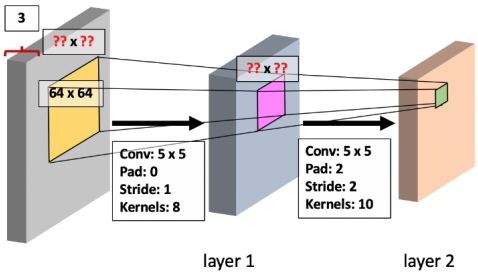


Figure 4: Convolutional Architecture

3. Gradient Descent Optimization

(a) Consider Figure 5, which depicts a loss 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 step t. Please sketch an estimate of the path of updates that would be taken by vanilla SGD until "convergence".
Answer:
the green line in the attached plot shows the sketchy convergent path of polyline.

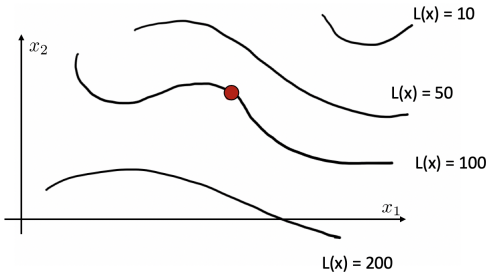


Figure 5: Contour lines of an arbitrary cost function with current estimate $\mathbf{x}_t$.

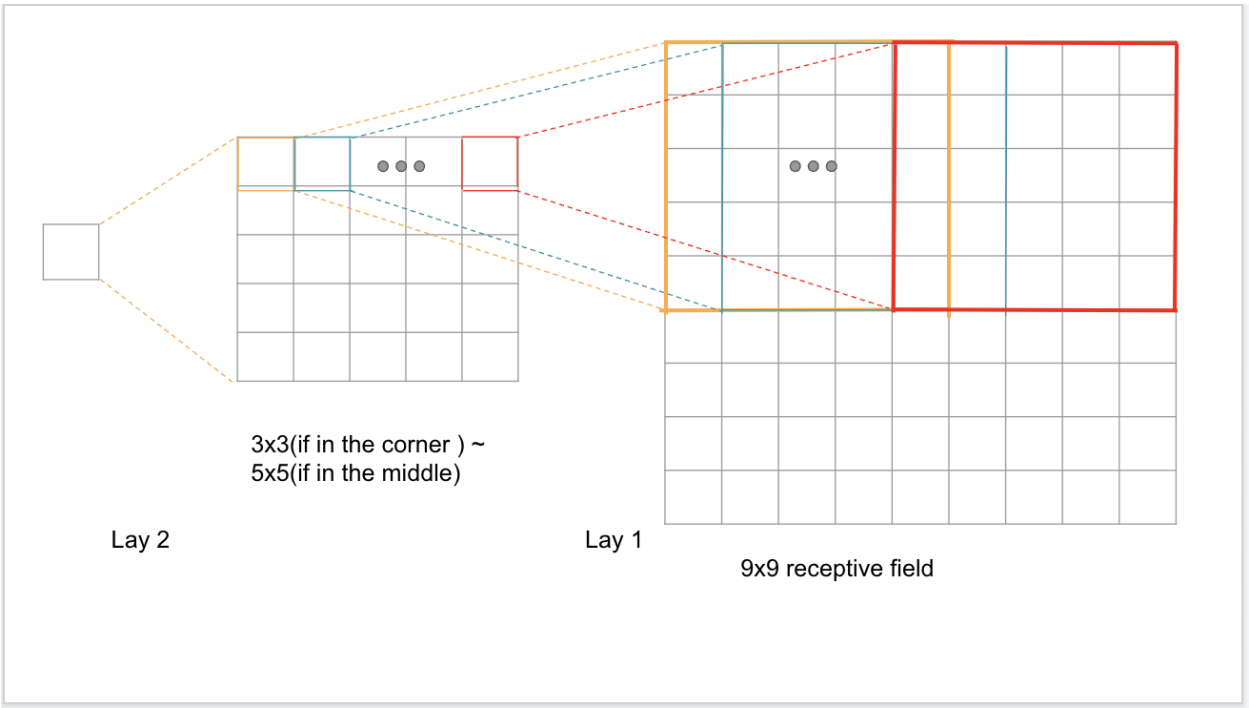
(b) It is worth mentioning that the contour lines shown in Fig. 5 will change during optimization since the loss is evaluated over a single batch rather than the whole dataset. As discussed in class, this observation suggests that for unfortunate updates we might get stuck in saddle points, where the vanilla SGD gradient is 0. One way to combat this problem is to use first and/or second order momentum. Please briefly answer the following questions:

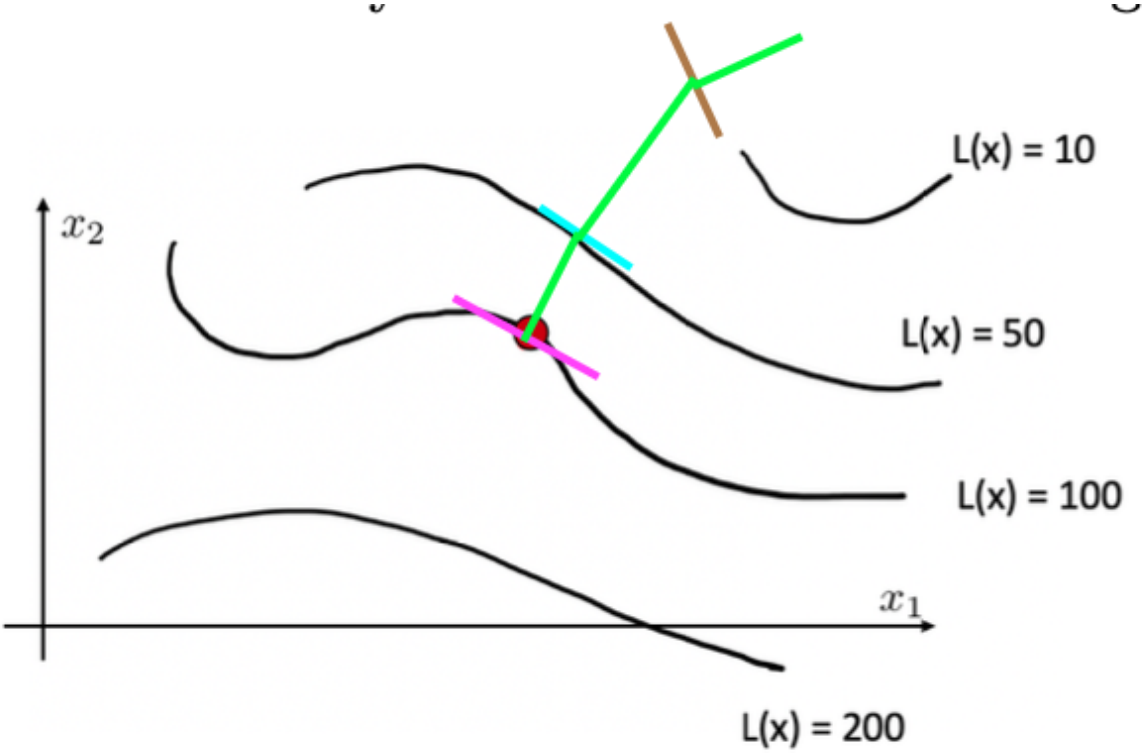
i. Why is the first order momentum helpful handling the for saddle points problem?

Answer:
Rather than updating the loss function with the direction of negative gradient, by adding first order momentum, it replaces the negative gradient with velocity term, which represents a running mean of previous gradients. The velocity term can carry the point through the saddle point.

ii. Give one example of other difficulties of optimization that the first order momentum can help address and explain why.
Answer:
It can also solve the zig-zag problem. Without the first order momentum, the stochastic gradient descent only considers the current direction of gradient descent. By adding first order momentum, it uses the running mean of the gradients, makes the path smoother. It can efficiently solve the problem of zig-zag path.

iii. How would using first and second order momentum change the update path sketched in (a) (sketch the changed update direction and explain why you think the updates will change in that way)?
Answer:
Since the first and second order momentums are methods of smoothing our outcome, I think the original convergent path of polyline will become a more smoother curve without obvious turning points.


3x3(if in the corner) ~
5x5(if in the middle)
Lay 2
Lay 1
9x9 receptive field


L(x) = 10
x_2
L(x) = 50
L(x) = 100
x_1
L(x) = 200