Due Date: October 17th, 2022 at 11:00 pm

Instructions

- *For all questions, show your work!*
- *Use LaTeX and the template we provide when writing your answers. You may reuse most of the notation shorthands, equations and/or tables. See the assignment policy on the course website for more details.*
- *Submit your answers electronically via Gradescope.*
- *TAs for this assignment are* ***Nanda Harishankar Krishna and Sarthak Mittal****.*

Question 1 (5-2-3-5)**.** (**Activation Functions**)

1. Consider the activation function $h(x) = x\sigma(\beta x)$ where $\sigma(\cdot)$ defines the sigmoid activation function $\sigma(z) = \frac{1}{1+\exp(-z)}$. Show that, for an appropriate choice of β, this activation function can approximate (a) the linear activation function, and (b) the ReLU activation function.
2. Consider the continuous piece-wise linear activation function $h(x)$ which has the following slopes at different parts of the input, in particular, $h'(x) = \begin{cases} a & x >= \alpha \\ 0 & \alpha < x < \beta \\ -b & x <= \beta \end{cases}$. Suppose you are only given access to arithmetic operations (addition, subtraction, multiplication and division) as well as the ReLU activation function $ReLU(x) = max(0, x)$. Can you construct $h(x)$ using these elementary operations and the ReLU activation ? Can you think of a popular activation function similar to $h(\cdot)$? *Hint: Try visualizing the activation function*
3. Recall the definition of the softmax function $\mathcal{S}(\boldsymbol{x})_i = e^{x_i} / \sum_j e^{x_j}$, where $\boldsymbol{x} \in \mathbb{R}^d$. Show that it is translation invariant, i.e. $\mathcal{S}(\boldsymbol{x} + \boldsymbol{c}) = \mathcal{S}(\boldsymbol{x})$ for any $\boldsymbol{c} \in \mathbb{R}^d$.
4. Given an i, it is often needed to compute $\log \mathcal{S}(\boldsymbol{x})_i = x_i - \log \sum_j e^{x_j}$. However, this can often lead to numerical unstabilities because (a) if all x_j are small, then it becomes close to $\log 0$ (underflow), and (b) if all x_j are large, then their exponents can lead to overflow. Can you use the translation invariance property of softmax to make it numerically stable ? *Hint: Think of* $\max_j e^{x_j}$.

Answer 1.

Question 2 (10)**.** (**Mixture Models meet Neural Networks**)

Consider modelling some data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, $\boldsymbol{x}_n \in \mathbb{R}^d$, $y_n \in \{0, 1\}$, using a mixture of logistic regression models, where we model each binary label y_n by first picking one of the K logistic regression models, based on the value of a latent variable $z_n \sim \text{Categorical}(\pi_1, ..., \pi_K)$ and then generating y_n *conditioned* on z_n as $y_n \sim \text{Bernoulli}[\sigma(\boldsymbol{w}_{z_n}^T \boldsymbol{x}_n)]$, where $\sigma(\cdot)$ is the sigmoid activation function.

Now consider the *marginal* probability of the label $y_n = 1$, given $\boldsymbol{x}_n$, i.e., $p(y_n = 1|\boldsymbol{x}_n)$, and show that this quantity can also be thought of as the output of a neural network. Clearly specify what is the input layer, the hidden layer(s), activations, the output layer, and the connection weights of this neural network.

Answer 2.

Question 3 (5-2-5-2-2-4)**.** (**Optimization Schemes**)

Optimization is one of the most important pillars for Deep Learning. With the increase in compute power as well as more complex optimization schemes like Momentum and Adam, we are able to optimize really large-scale and complex deep learning models. However, all the algorithms that we have seen yet only utilize first-order derivative information. Is something better possible ?

Let $f(\boldsymbol{w})$ be the function that you want to minimize. One simple way is to do gradient descent, that is, iteratively update $\boldsymbol{w}$ as $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \nabla f(\boldsymbol{w}^{(t)})$. Lets try to do better now.

1. Consider the second-order Taylor approximation of the function f at $\boldsymbol{w}^{(t)}$.

$$f(\boldsymbol{w}) \approx f(\boldsymbol{w}^{(t)}) + \langle \boldsymbol{w} - \boldsymbol{w}^{(t)}, \nabla f(\boldsymbol{w}^{(t)}) \rangle + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

   What is the solution of the following optimization problem ?

$$\arg\min_{\boldsymbol{w}} \left[f(\boldsymbol{w}^{(t)}) + \langle \boldsymbol{w} - \boldsymbol{w}^{(t)}, \nabla f(\boldsymbol{w}^{(t)}) \rangle + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^T \boldsymbol{H} (\boldsymbol{w} - \boldsymbol{w}^{(t)}) \right] \quad (1)$$

   where $\boldsymbol{H}$ is the Hessian matrix, that is, the matrix whose elements are $\frac{\partial^2}{\partial w_i w_j} f(\boldsymbol{w}^{(t)})$ Does this tell you something about the "learning rate" ?

2. Consider a Multi-Layer Perceptron Model (MLP) consisting of two hidden layers with 512 neurons each. The input consists of 32 features, and the output is 10-dimensional to represent the probability of each class in a 10-class classification problem. How many parameters does the model have (please include the bias terms as well) ? What is the size of the Hessian matrix $\boldsymbol{H}$?

3. Given the current scale of architectures (eg. GPT, Dall-E, etc.), it is easy to see that computing $\boldsymbol{H}$ is often intractable. However, often we do not need the full $\boldsymbol{H}$ matrix, and instead we just need some Hessian-vector products, that is, $\boldsymbol{H}\boldsymbol{v}$ for some vector $\boldsymbol{v}$. Let $g(\cdot)$ be the gradient function of f, and consider the Taylor series expansion

$$g(\boldsymbol{w} + \Delta\boldsymbol{w}) \approx g(\boldsymbol{w}) + \boldsymbol{H}\Delta\boldsymbol{w}$$

   Can we use this approximation to estimate the Hessian-vector product $\boldsymbol{H}\boldsymbol{v}$, given any vector $\boldsymbol{v}$. (*Hint: Consider* $\Delta\boldsymbol{w} = r\boldsymbol{v}$ *with small* r)

4. The actual Taylor series expansion of the gradient function is instead $g(\boldsymbol{w} + \Delta\boldsymbol{w}) = g(\boldsymbol{w}) + \boldsymbol{H}\Delta\boldsymbol{w} + O(||\Delta\boldsymbol{w}||^2)$. In this case, what is the error estimate when computing $\boldsymbol{H}\boldsymbol{v}$ as derived above, where $\Delta\boldsymbol{w} = r\boldsymbol{v}$. You can use the $O(\cdot)$ notation to describe what the error would be.

5. The problem with this approximate solution is that it is often prone to numerical errors. This is because we need r to be small to reduce the error, but when r is very small then $r\boldsymbol{v}$ loses precision when added to $\boldsymbol{w}$. However, maybe we can do better and get a more exact solution instead of an approximate one ?

   Consider the $\mathcal{R}-$operator, which is defined as

$$\mathcal{R}_v\{f(\boldsymbol{w})\} = \frac{\partial}{\partial r} f(\boldsymbol{w} + r\boldsymbol{v})\Big|_{r=0}$$

   Show that

$$\mathcal{R}_v\{cf(\boldsymbol{w})\} = c\mathcal{R}_v\{f(\boldsymbol{w})\} \qquad \text{(Linearity under Scalar Multiplication)}$$
$$\mathcal{R}_v\{f(\boldsymbol{w})g(\boldsymbol{w})\} = \mathcal{R}_v\{f(\boldsymbol{w})\}g(\boldsymbol{w}) + f(\boldsymbol{w})\mathcal{R}_v\{g(\boldsymbol{w})\} \qquad \text{(Product Rule)}$$

6. Finally, use the $\mathcal{R}-$operator to derive an equation to compute the Hessian-vector product $\boldsymbol{Hv}$ exactly.

   The algorithm that we derived is the popular *Newton Method*, which has been proven to converge faster than first-order methods under certain assumptions. Not only did we derive this optimization procedure, but we have also uncovered ways of tractably estimating the matrix-vector products needed.

Answer 3.

Question 4 (3-5-5-7). (**Convolution Basics**)

1. Consider a convolutional neural network with the following architecture

   Image → Conv-3x3 → ReLU → Conv-3x3 → MaxPool-2x2 → Conv-3x3 → ReLU → Output

   where Conv-3x3 refers to a Convolutional Layer with a 3x3 kernel size, and MaxPool-2x2 refers to the max-pooling operation over a 2x2 kernel window. The convolutions and pooling operations do not use any padding. Given an input image of resolution (32×32), find the resolution of the output when passing this image through the above defined network.
2. Consider another convolutional neural network with the architecture:

   Image → Conv-5x5 → ReLU → Conv-5x5 → ReLU → Global Avg. Pool → FC-128 → FC-1 (Output)

   where we reuse the Conv-axa notation to represent convolutional layers with axa kernel size, and FC-b refers to a Feedforward neural network with b neurons. Can this architecture handle variable sized images ? Are Multi-Layer Perceptron netowrks (without Convolutional or global pooling layers) able to handle variable sized inputs ? If so, why ; otherwise, why not ?

3. Convolution layers *intrinsically* compute a linear function (similar to feed-forward networks without non-linear activations!). Recall that given input features X, a linear function is any function that can be computed as $Y = AX$, for some matrix A.
   Consider a convolution layer with the kernel W and input X given by
$$W = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \quad X = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ x_{2,0} & x_{2,1} & x_{2,2} \end{bmatrix}$$
   Find the output map when convolving the input X with the kernel W, **using stride 2 and zero padding of size 1**. Can you re-write this as a linear function (that is, what is the matrix A for this transformation?). You can consider row-major expansion of features, which represents all elements of first row, then all elements of second row, and so on as a flattened vector, eg. $X = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{1,0} & \dots & x_{2,1} & x_{2,2} \end{bmatrix}$
4. Transposed convolutions can help upsample the features spatially. Consider a transposed convolution layer with kernel $W = \begin{bmatrix} w_{0,0} & w_{0,1} \\ w_{1,0} & w_{1,1} \end{bmatrix}$, used on the input $X = \begin{bmatrix} x_{0,0} & x_{0,1} \\ x_{1,0} & x_{1,1} \end{bmatrix}$, with **stride 2 and no padding**. Is the output still a linear transform of the input features? Can you show what the linear matrix should be, given row-major expansion for features as before?

Answer 4.

Question 5 (3-2-3-7)**.** (**Receptive Fields**)

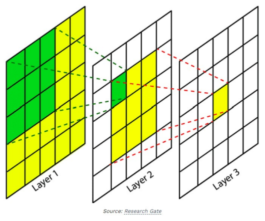

FIGURE 1 – Illustration of Receptive Field

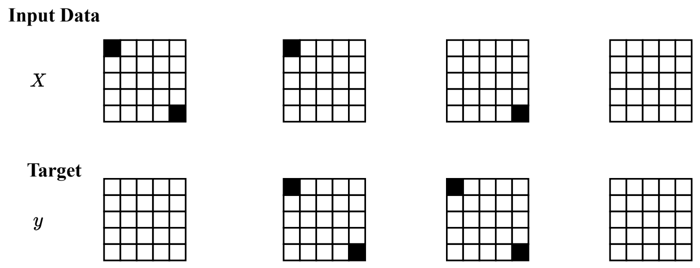

FIGURE 2 – Data for Question 5, where X are the input images and y are the ground-truth output images.

Receptive Field is defined as the size of the region in the input that produces a feature. For example, when using a Conv-3x3 layer, each feature is generated by a 3x3 grid in the input. Having two Conv-3x3 layers in succession leads to a 5x5 receptive field. An example of this is outlined in Figure 1. Note that there is a receptive field associated with each feature (eg. here it is for the yellow colored feature at Layer 3), where a feature is a cell in the output activation map.

Consider a network architecture consisting solely of **4 Convolution layers, each with kernel size 3, zero-padding of one pixel on all four sides, 16 intermediate channels, and one output channel.**

1. Find the resolution of the output map when the input to the model is of size 32×32. Is the output size the same as the input size ?
2. Are there cases when one might want to consider outputs that are of the same size as the input (in spatial resolutions) ? Please provide examples of such cases.
3. What is the receptive field of the final features obtained through this architecture, that is, what is the size of the input that corresponds to a particular pixel position in the output ?
4. Suppose you are given some data, as outlined in Figure 2. That is, you have 5×5 resolution images, where if the two diagonally opposite corner pixels in the input are the same, then the output map (of the same size), has the corresponding corners as white. If the two corners in the input are different, then the output map's corresponding corners are black. All the possible cases of the input are outlined in the Figure 2.

   Given the architecture described above, would it be possible to learn to do well on this data ? That is, can the model get 100% accuracy on this data ? Does anything change if you add a residual connection in between any two layers ?

Answer 5.