Stat 6910, Section 002
Deep Learning: Theory and Applications
Spring 2019

Homework 3

Jared Hansen

A-number: A01439768

e-mail: jrdhansen@gmail.com

Due: 5:00 PM, Friday 02/15/2019

Homework III

STAT 6910/7810-002 - Spring semester 2019

Due: Friday, February 15, 2019 - 5:00 PM

Please put all relevant files & solutions into a single folder titled `<lastname and initials>_assignment3` and then zip that folder into a single zip file titled `<lastname and initials>_assignment3.zip`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment3.zip`. Include a single PDF titled `<lastname and initals>_assignment3.pdf` and any Python scripts specified. Any requested plots should be sufficiently labeled for full points.

Unless otherwise stated, programming assignments should use built-in functions in Python, Tensorflow, and PyTorch. In general, you may use the `scipy` stack; however, exercises are designed to emphasize the nuances of machine learning and deep learning algorithms - if a function exists that trivially solves an entire problem, please consult with the TA before using it.

Problem 1

1. Provide a geometric interpretation of gradient descent in the one- dimensional case. (Adapted from the Nielsen book, chapter 1)

- First, to simplify some aspects of the explanation let's make the following assumptions:
  - The function f that we're working with is strictly convex (something like $f(x) = x^2$).
  - We have some pre-defined stopping criterion, i.e. a maximum number of iterations, a threshold for which the value of the gradient is "small enough", $\nabla f = 0$, etc.
  - We have some pre-defined learning rate, η, that is fixed for the duration of the algorithm.
- Now that we have made those assumptions, the first thing we can do is calculate the general gradient, ∇f, of the function. For the function $f(x) = x^2 \rightarrow \nabla f = 2x$.
- Next, we initialize some random draw from the input space of our function f. Let's call this draw x_0.
- The object of the optimization is to find the input x^* that gives us the global minimum, $f(x^*)$, of the function f. In other words we want to find x^* such that $\nabla f(x^*) = 0$, or if we have some other stopping rule, x^* such that $\nabla f(x^*) \approx 0$.
- Unless we're incredibly lucky, it's likely that $f(x_0) \neq 0$. But, by using the gradient, we can "take steps" from x_0 toward x^*, moving "down" the surface of the cost function toward $f(x^*)$.
- In higher-dimensional settings, "taking a step" means adjusting some/all of the inputs by some $\big[\Delta\text{input}\big]$ vector, but in the one-dimensional case this simply means adjusting x either to the left or the right since x is the only input. Assuming our input space is $x \in \mathbb{R}$ we can only adjust it to be smaller (to the left) or larger (to the right) to move us down the surface of the function.
- Since we can only move left or right, we can use the following decision rule to make our first step:
  - If $\nabla f(x_0) > 0 \rightarrow$ take a step to the left (make $x_1 < x_0$ by step amount)
  - If $\nabla f(x_0) < 0 \rightarrow$ take a step to the right (make $x_1 > x_0$ by step amount)
  - Define the step amount Δx to be: $\Delta x := -\eta\big(\nabla f(x_i)\big)$.
    So if we have x_0 and want to get x_1 this would simply be $x_1 = x_0 - \eta\big(\nabla f(x_0)\big)$
- Consider the function $f(x) = x^2$. Think of drawing lines tangent to the graph of f for values of $x > 0$: they all have positive slope, right? And the further right (larger x we get) the more positive the slope of those lines. In other words, the derivative (gradient) of f for $x > 0$ is positive, and ∇f becomes larger as x increases.
- So if we want to get closer to a derivative (∇f) of 0 at some x, we need to move our x away from values that gives large, positive derivatives. That means moving our input x_1 to the left of x_0 ("take a step to the left") in order to achieve a gradient closer to 0 when $\nabla f(x_0) > 0$.
- Very similar logic holds for taking a step to the right when $\nabla f(x_0) < 0$: we need to move down the function surface away from x values that produce highly negative gradeints to some x that gives a gradient closer to 0. Therefore we adjust such that $x_1 > x_0$, taking a step to the right.
- We continue taking steps toward the bottom (minimum) of the function by using the decision rule at each iteration. Depending on how we choose the learning rate, η, of our gradient descent algorithm, we might not ever actually reach the true minimum, $f(x^*)$ (think Zeno's Paradox).
- To prevent this interminable calculation, we have defined some stopping rule before starting. (This is mentioned above.) Let's say we chose our stopping rule to be: "assume the minimum of the function f is approximately achieved by x^i when $|\Delta x| \leq 0.005$."
- Geometrically speaking, this is saying we're in a flat enough part of the function (close enough to the true minimum) that we've reached a very close approximation of the true minimum (which is the only truly "flat" part of the function).
  Mathematically speaking, we're saying that $\Big[\nabla f(x^i) \approx 0\Big] \implies \Big[x^i \approx x^* \text{ and } f(x^i) \approx f(x^*)\Big]$

2. An extreme version of gradient descent is to use a mini-batch size of just 1. This procedure is known as online or incremental learning. In online learning, a neural network learns from just one training input at a time (just as human beings do). Name one advantage and one disadvantage of online learning compared to stochastic gradient descent with a mini-batch size of, say, 20. (Adapted from the Nielsen book, chapter 1)

   - One disadvantage of incremental learning is that the weights and biases, call them $\boldsymbol{w}$, in our network will jump around much more than they would with a mini-batch size of 20. This is a result of the fact that $\boldsymbol{w}$ gets updated after each training example passes through the network instead of after 20 examples.
     As a result, my guess is that it might be harder to converge to some reasonable minimum of our objective function. The $\boldsymbol{w}$ might just jump around indefinitely, never "settling down" to values that give good performance.
     Take the MNIST data set for example. Since $\boldsymbol{w}$ gets updated after each training image passes through the network, the network will tailor itself to better classify the most recently trained-on image. So if we pass a 4 through the network, if the next image is a 3 the network will adjust $\boldsymbol{w}$ dramatically relative to passing 20 images through before adjusting $\boldsymbol{w}$.
   - An advantage of incremental learning is that it would be more computationally efficient and simple than using a mini-batch size of 20. We'd only have to compute the gradient of the cost function relative to a single training example instead of 20 of them. From the standpoint of those who have built the computational architecture behind neural networks, I'm guessing that this wasn't easy. Dealing with computation in terms of tensors rather than single matrices and vectors would require much more complex coding (at least from my novice perspective).
     At the very least, a definite advantage of incremental learning is that it's more simple to teach to those new to deep learning. It makes notation and intuition more simple and easy to grasp.

3. Create a network that classifies the MNIST data set using only 2 layers: the input layer (784 neurons) and the output layer (10 neurons). Train the network using stochastic gradient descent on the training data. What accuracy do you achieve on the test data? You can adapt the code from the Nielson book, but make sure you understand each step to build up the network. Alternatively, you can use Tensorflow, or Pytorch. Please save your code as `prob1.py` and state which library/framework you used. Report the learning rate(s) and mini-batch size(s) you used. (Adapted from the Nielsen book, chapter 1)

NOTES:

- I used Pytorch to build the neural network described above.
- Not specified in the prompt, so I decided to use the following:
  - Activation function: sigmoid
  - Loss function: cross entropy
- I started with this tutorial and modified it a fair bit for this problem
  `https://towardsdatascience.com/a-simple-starter-guide-to-build-a-neural-network-3c2cf07b8d7c`
  Also, I made extensive comments to show I understood (at least most of) what the code is doing.
- I wrote code that runs only on CPU (roughly the first half of the code, see code comments) as well as code that makes use of GPU capabilities (roughly the second half of the code, see code comments). To be fair, I didn't entirely understand some of the GPU-related code I wrote.

  Also, if running my code, either delete or comment out the GPU section in order for the script to run on a machine that doesn't have GPU capability.
- Below are the results of the networks I built. The best test accuracy I achieved was with a mini-batch size of 20 and a learning rate of 1.0, correctly classifying 91.51% of the test images.
  From what little tuning I did, we can see that (ceteris paribus) a small mini-batch size gives a higher test accuracy, and that (ceteris paribus) a larger value for the learning rate also gives higher accuracy.

Test Accuracy of Simple Network on MNIST Data		
Mini-Batch Size	**Learning Rate**	**Test Accuracy**
100	0.001	77.24%
100	0.1	89.10%
100	1.0	90.99%
20	0.001	83.99%
20	0.1	90.55%
20	1.0	91.51%

Problem 2

1. Alternate presentation of the equations of backpropagation (Nielsen book, chapter 2)
   Show that $\delta^L = \nabla_a C \odot \sigma'(z^L)$ can be written as $\delta^L = \sum'(z^L)\nabla_a C$, where $\Sigma'(z^L)$ is a square matrix whose diagonal entries are the values $\sigma'(z_j^L)$ and whose off-diagonal entries are zero.

Work and answer for DL_hw3, prob 2.1.
(Would've LaTeX-ed, but it would have taken an inordinate amount of time)

- The approach for this problem is to show that LHS (left-hand side) of $\delta^L = [\nabla_{a^L} C] \odot [\sigma'(z^L)]$ can be written as (is equivalent to) RHS (right-hand side) of $\delta^L = [\Sigma'(z^L)][\nabla_{a^L} C]$ where $\Sigma'(z^L) = \text{diag}(\sigma'(z_j^L))$.

- By definition, $\nabla_{a^L} C$ = gradient of the cost function w.r.t. the activations of the output layer L. $= \begin{bmatrix} \frac{\partial C}{\partial a_1^L} \\ \frac{\partial C}{\partial a_2^L} \\ \vdots \\ \frac{\partial C}{\partial a_j^L} \end{bmatrix}$

- Since $\sigma(z^L) = a^L$ (where z^L and a^L are vectors of neuron inputs and neuron outputs for the L^{th} layer) we know that $\sigma'(z^L) = \frac{\partial a^L}{\partial z^L} = \begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} \\ \frac{\partial a_2^L}{\partial z_2^L} \\ \vdots \\ \frac{\partial a_j^L}{\partial z_j^L} \end{bmatrix}$

- Now let's define $\Sigma'(z^L)$. Per the prompt, $\Sigma'(z^L) = \text{diag}\left(\sigma'(z_j^L)\right)$, so

$$\Sigma'(z^L) = \begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} & 0 & \cdots & 0 \\ 0 & \frac{\partial a_2^L}{\partial z_2^L} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \frac{\partial a_j^L}{\partial z_j^L} \end{bmatrix}$$

- Substituting these vectors and matrix in to the original expression we see:

$$LHS = \underbrace{\begin{bmatrix} \frac{\partial C}{\partial a_1^L} \\ \vdots \\ \frac{\partial C}{\partial a_j^L} \end{bmatrix}}_{\nabla_{a^L} C} \odot \underbrace{\begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} \\ \vdots \\ \frac{\partial a_j^L}{\partial z_j^L} \end{bmatrix}}_{\sigma'(z^L)} = \underbrace{\begin{bmatrix} \frac{\partial C}{\partial z_1^L} \\ \vdots \\ \frac{\partial C}{\partial z_j^L} \end{bmatrix}}_{\delta^L} \longrightarrow LHS = \delta^L$$

$$RHS = \underbrace{\begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \frac{\partial a_j^L}{\partial z_j^L} \end{bmatrix}}_{\Sigma'(z^L)} \underbrace{\begin{bmatrix} \frac{\partial C}{\partial a_1^L} \\ \vdots \\ \frac{\partial C}{\partial a_j^L} \end{bmatrix}}_{\nabla_{a^L} C} = \begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} \cdot \frac{\partial C}{\partial a_1^L} + 0 + \ldots + 0 \\ \vdots \\ 0 + \ldots + 0 + \frac{\partial a_j^L}{\partial z_j^L} \cdot \frac{\partial C}{\partial a_j^L} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial C}{\partial z_1^L} \\ \vdots \\ \frac{\partial C}{\partial z_j^L} \end{bmatrix}}_{\delta^L} \longrightarrow RHS = \delta^L$$

- Since we've shown $LHS = \delta^L$ and $RHS = \delta^L$ we have shown that $\delta^L = [\nabla_{a^L} C] \odot [\sigma'(z^L)]$ can be written as $\delta^L = [\Sigma'(z^L)][\nabla_{a^L} C]$ as desired.

2. Show that $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$ can be rewritten as $\delta^l = \Sigma'(z^l)(w^{l+1})^T \delta^{l+1}$.

Problem 2.2

Show that $\delta^l = ((w^{l+1})^T (\delta^{l+1})) \odot \sigma'(z^l)$ can be rewritten $\delta^l = [\Sigma'(z^l)][w^{l+1}]^T[\delta^{l+1}]$.

- Again our approach will be to show that LHS = RHS $\Rightarrow$ LHS can be rewritten as RHS, where LHS = the first δ^l expression and RHS is the second.
- Let the l^{th} layer of the network have k neurons and the $(l+1)^{th}$ layer have j neurons (consistent with course notes).
- First, Let's establish notation for the (w^{l+1}) term in these expressions.
- The weights for/coming into the $(l+1)^{th}$ layer is the matrix w^{l+1} where $w^l_{j,k}$ is the weight connecting the k^{th} neuron in the $(l-1)^{th}$ layer with the j^{th} neuron in the l^{th} layer. $w^{l+1} =$

$$\begin{bmatrix} (w^{l+1}_{1,1}) & (w^{l+1}_{1,2}) & \cdots & (w^{l+1}_{1,k}) \\ (w^{l+1}_{2,1}) & (w^{l+1}_{2,2}) & \cdots & (w^{l+1}_{2,k}) \\ \vdots & \vdots & \ddots & \vdots \\ (w^{l+1}_{j,1}) & (w^{l+1}_{j,2}) & \cdots & (w^{l+1}_{j,k}) \end{bmatrix}$$

(w^{l+1} is jxk in dimension) $\longrightarrow$

- LHS =

$$\underbrace{\begin{bmatrix} (w^{l+1}_{1,1}) & (w^{l+1}_{2,1}) & \cdots & (w^{l+1}_{j,1}) \\ (w^{l+1}_{1,2}) & (w^{l+1}_{2,2}) & \cdots & (w^{l+1}_{j,2}) \\ \vdots & \vdots & \ddots & \vdots \\ (w^{l+1}_{1,k}) & (w^{l+1}_{2,k}) & \cdots & (w^{l+1}_{j,k}) \end{bmatrix}}_{(w^{l+1})^T} \underbrace{\begin{bmatrix} \delta^{l+1}_1 \\ \delta^{l+1}_2 \\ \vdots \\ \delta^{l+1}_j \end{bmatrix}}_{\delta^{l+1}} \odot \underbrace{\begin{bmatrix} \partial a^l_1/\partial z^l_1 \\ \partial a^l_2/\partial z^l_2 \\ \vdots \\ \partial a^l_k/\partial z^l_k \end{bmatrix}}_{\sigma'(z^l)} = \underbrace{\begin{bmatrix} \Sigma_j (w^{l+1}_{j,1})(\delta^{l+1}_j)(\partial a^l_1/\partial z^l_1) \\ \Sigma_j (w^{l+1}_{j,2})(\delta^{l+1}_j)(\partial a^l_2/\partial z^l_2) \\ \vdots \\ \Sigma_j (w^{l+1}_{j,k})(\delta^{l+1}_j)(\partial a^l_k/\partial z^l_k) \end{bmatrix}}_{[(w^{l+1})^T][\delta^{l+1}][\sigma'(z^l)]}$$

dim: kxj | dim: jx1 | dim: kx1 | dim: kx1

- Now let's write the expanded form of RHS and cross our fingers that it is identical to the expanded form of LHS:

$$RHS = \begin{bmatrix} \left(\frac{\partial a_1^l}{\partial z_1^l}\right) & 0 & \cdots & 0 \\ 0 & \left(\frac{\partial a_2^l}{\partial z_2^l}\right) & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \left(\frac{\partial a_k^l}{\partial z_k^l}\right) \end{bmatrix} \begin{bmatrix} \sum_j \left(w_{j,1}^{l+1}\right)\left(\delta_j^{l+1}\right) \\ \sum_j \left(w_{j,2}^{l+1}\right)\left(\delta_j^{l+1}\right) \\ \vdots \\ \sum_j \left(w_{j,k}^{l+1}\right)\left(\delta_j^{l+1}\right) \end{bmatrix} = \begin{bmatrix} \sum_j \left(w_{j,1}^{l+1}\right)\left(\delta_j^{l+1}\right)\left(\partial a_1^l / \partial z_1^l\right) \\ \sum_j \left(w_{j,2}^{l+1}\right)\left(\delta_j^{l+1}\right)\left(\partial a_2^l / \partial z_2^l\right) \\ \\ \sum_j \left(w_{j,k}^{l+1}\right)\left(\delta_j^{l+1}\right)\left(\partial a_k^l / \partial z_k^l\right) \end{bmatrix}$$

$\Sigma'(z^l)$ — dim: $k \times k$

$[(w^{l+1})^T(\delta^{l+1})]$ — dim: $(k \times j)(j \times 1) = (k \times 1)$

$[\Sigma'(z^l)][(w^{l+1})^T][\delta^{l+1}]$ — dim: $k \times 1$

- We can clearly see $LHS = RHS = \begin{bmatrix} \sum_j \left(w_{j,1}^{l+1}\right)\left(\delta_j^{l+1}\right)\left(\sigma'(z_1^l)\right) \\ \sum_j \left(w_{j,2}^{l+1}\right)\left(\delta_j^{l+1}\right)\left(\sigma'(z_2^l)\right) \\ \\ \sum_j \left(w_{j,k}^{l+1}\right)\left(\delta_j^{l+1}\right)\left(\sigma'(z_k^l)\right) \end{bmatrix}$

This implies that $\left[LHS = \delta^l = [(w^{l+1})^T(\delta^{l+1})] \odot [\sigma'(z^l)]\right]$ can be rewritten as $\left[RHS = \delta^l = [\Sigma'(z^l)][(w^{l+1})^T][\delta^{l+1}]\right]$, showing the desired result of the prompt.

3. By combining the results from problems 2.1 and 2.2, show that $\delta^l = \Sigma'(z^l)(w^{l+1})^T \dots \Sigma'(z^{L-1})(w^L)^T \Sigma'(z^L) \nabla_a C$.

- For notational ease, in this problem let $\big[term^i\big] := \big[\Sigma'(z^i)(w^{i+1})^T\big]$

- From the 2.2 prompt we're given that $\delta^l = \big[\Sigma'(z^l)(w^{l+1})^T\big]\big[\delta^{l+1}\big]$ and can rewrite using our new $\big[term^i\big]$ expression as
  $\delta^l = \big[term^l\big]\big[\delta^{l+1}\big]$ and can use this fact to make the following deductions:

  $\delta^{l+1} = \big[term^{l+1}\big]\big[\delta^{l+2}\big]$
  $\delta^{l+2} = \big[term^{l+2}\big]\big[\delta^{l+3}\big]$
  $\vdots$
  $\delta^{L-2} = \big[term^{L-2}\big]\big[\delta^{L-1}\big]$
  $\delta^{L-1} = \big[term^{L-1}\big]\big[\delta^{L}\big]$
  $\delta^{L} = \big[\Sigma'(z^L)\big]\big[\nabla_{a^L} C\big]$ $\leftarrow$ give in problem 2.1 prompt.

- Now we can recursively substitute terms into the first δ^l expression to find δ^l only in terms of $\big[term^l\big]\big[term^{l+1}\big] \dots \big[term^{L-1}\big]\big[\delta^L\big]$.

- We have that $\delta^l = \big[term^l\big]\big[\delta^{l+1}\big]$. We also know that $\delta^{l+1} = \big[term^{l+1}\big]\big[\delta^{l+2}\big]$ and can substitute this into the δ^l expression giving:
  $\delta^l = \big[term^l\big]\big[term^{l+1}\big]\big[\delta^{l+2}\big]$.

- This pattern of iterative substitution continues, giving:
  $\delta^l = \big[term^l\big]\big[term^{l+1}\big]\big[term^{l+2}\big] \dots$

- Now let's define what happens at the terminus of this pattern (refer to initial list of relationship patterns for definitions):
  $\delta^{L-2} = \big[term^{L-2}\big]\big[\delta^{L-1}\big] = \big[term^{L-2}\big]\big[term^{L-1}\big]\big[\delta^{L}\big] = \big[term^{L-2}\big]\big[term^{L-1}\big]\big[\Sigma'(z^L)\big]\big[\nabla_{a^L} C\big]$

- Putting all of this together we have:
  $\delta^l = \big[term^l\big]\big[term^{l+1}\big]\big[term^{l+2}\big] \dots \big[term^{L-2}\big]\big[term^{L-1}\big]\big[\Sigma'(z^L)\big]\big[\nabla_{a^L} C\big]$

  Substituting back in the definition made at the top $\big[term^i\big] := \big[\Sigma'(z^i)(w^{i+1})^T\big]$ we have:

  $$\boxed{\delta^l = \big[\Sigma'(z^l)(w^{l+1})^T\big]\big[\Sigma'(z^{l+1})(w^{l+2})^T\big] \dots \big[\Sigma'(z^{L-2})(w^{L-1})^T\big]\big[\Sigma'(z^{L-1})(w^L)^T\big]\big[\Sigma'(z^L)\big]\big[\nabla_{a^L} C\big]}$$

  which is what the prompt wanted us to show.

4. Backpropagation with linear neurons (Nielsen book, chapter 2)
Suppose we replace the usual non-linear σ function (*sigmoid*) with $\sigma(z) = z$ throughout the network. Rewrite the backpropagation algorithm for this case.

- Before writing out the new steps of the algorithm, let's define a couple of things:
  (a) Typically $\boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$, but we are redefining layer outputs such that $\boxed{\boldsymbol{a}^l = \boldsymbol{z}^l}$
  (b) This being the case, let's redefine what $\sigma'(\boldsymbol{z}^l)$.
  When $\boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$ we know that $\left[\sigma'(z_j^l) = \dfrac{\partial a_j^l}{\partial z_j^l}\right]$ so if $\sigma(\boldsymbol{z}^l) = \boldsymbol{z}^l \implies \left[\sigma'(z_j^l) = \dfrac{\partial z_j^l}{\partial z_j^l}\right]$
  $\implies \boxed{\left[\sigma'(\boldsymbol{z}^l) = \mathbf{1}\right]}$
- Now that we have established the changes that come from defining $\sigma(\boldsymbol{z}^l) = \boldsymbol{z}^l$, we can rewrite the backpropogation algorithm (given on slide 39 of the Backpropogation course notes).
- **Step 1 = input $\boldsymbol{x}$:** set the activation $\boldsymbol{a}^1$ for the input layer (just the input values of some training data point since we're dealing with only one $\boldsymbol{x}$ at a time).
- **Step 2 = Feedforward:** for each $l = 2, 3, \ldots, L-1, L$ compute:
  (a) $\boldsymbol{z}^l = (\boldsymbol{w}^l)(\boldsymbol{a}^{l-1}) + \boldsymbol{b}^l$ which, per our new definition of activations, is $\boldsymbol{z}^l = (\boldsymbol{w}^l)(\boldsymbol{z}^{l-1}) + \boldsymbol{b}^l$, and
  (b) $\boldsymbol{a}^l = \boldsymbol{z}^l$
- **Step 3 = Output error $\boldsymbol{\delta}^l$**: compute $\boldsymbol{\delta}^l = \left[\left(\nabla_{\boldsymbol{z}^l} C\right) \odot \left(\sigma'(\boldsymbol{z}^l)\right)\right] = \left[\left(\nabla_{\boldsymbol{z}^l} C\right) \odot \left(\mathbf{1}\right)\right] = \left[\nabla_{\boldsymbol{z}^l} C\right]$ recalling our defintion from the first of the prompt that $\left[\sigma'(\boldsymbol{z}^l) = \mathbf{1}\right]$
- **Step 4 = Backpropogate the error:** for each $l = L-1, L-2, \ldots, 3, 2$ compute
  $\boldsymbol{\delta}^l = \left[\left((\boldsymbol{w}^{l+1})^T(\boldsymbol{\delta}^{l+1})\right) \odot \left(\sigma'(\boldsymbol{z}^l)\right)\right] = \left[\left((\boldsymbol{w}^{l+1})^T(\boldsymbol{\delta}^{l+1})\right) \odot \left(\mathbf{1}\right)\right] \rightarrow \boldsymbol{\delta}^l = \left[(\boldsymbol{w}^{l+1})^T(\boldsymbol{\delta}^{l+1})\right]$
- **Step 5 = Output:** the cost function gradient is $\left[\dfrac{\partial C}{\partial w_{jk}^l} = \left(\boldsymbol{z}^{l-1}\right)\left(\delta_j^l\right)\right]$ and $\left[\dfrac{\partial C}{\partial b_j^l} = \delta_j^l\right]$

Problem 3

1. Let $f : \mathbb{R}^d \to \mathbb{R}$. Show that if f is strictly convex, then f has at most one global minimizer.

   This will be a proof by contradiction.

   Suppose $f(x^*_1) = g(x^*_2) = m$ where m is a global minimum (meaning there are two global minima), and $x^*_1 \neq x^*_2$.

   Let $x = \left[tx^*_1 + (1-t)x^*_2 \right]$ be a point between x^*_1 and x^*_2.

   Using the definition of convexity,

$$\begin{aligned} \left[f(x) = f(tx^*_1 + (1-t)x^*_2) \right] &< tx^*_1 + (1-t)x^*_2 \\ &< tm + (1-t)m \\ &< tm + m - tm \\ f(x) &< m \end{aligned}$$

   Since we found a value, $f(x)$, such that $f(x) < m$, we found that $\exists$ a value less than the global minimum. However, by definition, there <u>*cannot be*</u> something less than the global minimum. By this contradiction we can conclude that f is strictly convex when f has at most one global minimizer, proving the desired result.

2. Use the Hessian to give a simple proof that the sum of two convex functions is convex. You may assume that the two functions are twice continuously differentiable.
Hint: You may use the fact that the sum of two PSD matrices is also PSD.

Let $f(x)$ and $g(x)$ both be convex functions. Also, assume $f(x)$ and $g(x)$ are twice continuously differentiable. By definition, since $f(x)$ is twice continuously differentiable we know that at some local min, call this ${x^*}_1$, $\nabla^2 f({x^*}_1)$ is a positive semi-definite matrix. The same holds true for $g(x)$ at some local minima ${x^*}_2$: $\nabla^2 g({x^*}_2)$ is a positive semi-definite matrix.

Additionally, we know that:

- $result_1 = \left[\boldsymbol{z}^T\left[\nabla^2 f({x^*}_1)\right]\boldsymbol{z} \geq 0, \forall \boldsymbol{z} \in \mathbb{R}^d\right]$
- $result_2 = \left[\boldsymbol{z}^T\left[\nabla^2 g({x^*}_2)\right]\boldsymbol{z} \geq 0, \forall \boldsymbol{z} \in \mathbb{R}^d\right]$

We know that $[result_1 + result_2] \geq 0$ since both operands $result_1$ and $result_2$ are ≥ 0.

We know that $\nabla^2 f({x^*}_1)$ and $\nabla^2 g({x^*}_2)$ are both symmetric matrices $\Rightarrow \Big[[\nabla^2 f({x^*}_1)] + [\nabla^2 g({x^*}_2)]\Big]$ is also a symmetric matrix.

For shorter notation, let:

- $\boldsymbol{A} = [\nabla^2 f({x^*}_1)]$
- $\boldsymbol{B} = [\nabla^2 g({x^*}_2)]$

From properties of matrix-vector multiplication, we know that $\boldsymbol{z}^T(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{z} = \big[\boldsymbol{z}^T\boldsymbol{A}\boldsymbol{z}\big] + \big[\boldsymbol{z}^T\boldsymbol{B}\boldsymbol{z}\big]$. Since we know that $\boldsymbol{z}^T\boldsymbol{A}\boldsymbol{z} \geq 0$ and $\boldsymbol{z}^T\boldsymbol{B}\boldsymbol{z} \geq 0 \implies \boldsymbol{z}^T(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{z} \geq 0 \implies$ the matrix $\boldsymbol{A}+\boldsymbol{B}$ is positive semi-definite.
The matrix $(\boldsymbol{A}+\boldsymbol{B})$ is the Hessian for the function $\big(f(x)+g(x)\big)$ since $\nabla\big(f(x)+g(x)\big) = \nabla f(x)+\nabla g(x)$ and $\nabla^2\big(f(x)+g(x)\big) = \nabla^2 f(x)+\nabla^2 g(x) = (\boldsymbol{A}+\boldsymbol{B})$ which is positive semi-definite $\implies \Big(f(x)+g(x)\Big)$ is a convex function since it has a positive semi-definite Hessian.

Thus, we have used the Hessian to give a simple proof that the sum of two convex functions is convex.

3. Consider the function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c$ where A is a symmetric $d \times d$ matrix. Derive the Hessian of f. Under what conditions on A is f convex? Strictly convex?

- $\nabla^2 f(\boldsymbol{x}) = \dfrac{\partial}{\partial \boldsymbol{x}}\Big(\nabla f(\boldsymbol{x})\Big) = \dfrac{\partial}{\partial \boldsymbol{x}}\Big(\dfrac{\partial}{\partial \boldsymbol{x}}\big(f(\boldsymbol{x})\big)\Big)$ since all we have is $\boldsymbol{x}$ and no other independent variables.
- $\nabla f(\boldsymbol{x}) = \dfrac{\partial}{\partial \boldsymbol{x}} = \dfrac{1}{2}\dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{x}^T A\boldsymbol{x}\Big) + \dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{b}^T\boldsymbol{x}\Big) + \dfrac{\partial}{\partial \boldsymbol{x}}\Big(c\Big) \rightarrow \boxed{\nabla f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}}$
  (see items below for explanation)

- First let's look at this: $\dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{b}^T\boldsymbol{x}\Big) = \boldsymbol{b}^T$. Let's show this, where $\boldsymbol{b}^T$ is $1 \times d$ and $\boldsymbol{x}$ is $d \times 1$.
  $\boldsymbol{b}^T\boldsymbol{x} = \sum_{i=1}^{d} b_i x_i \rightarrow \dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{b}^T\boldsymbol{x}\Big) = \dfrac{\partial}{\partial \boldsymbol{x}}\Big(\sum_{i=1}^{d} b_i x_i\Big) = \sum_{i=1}^{n} b_i\Big(\dfrac{\partial}{\partial \boldsymbol{x}}(x_i)\Big) = \boldsymbol{b}^T$
- Second let's look at this: $\dfrac{1}{2}\dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{x}^T A\boldsymbol{x}\Big) = \dfrac{1}{2}\dfrac{2}{1}A\boldsymbol{x} = A\boldsymbol{x}$. Let's show this, where $\boldsymbol{x}$ is $d \times 1$ and A is $d \times d$.
  Let $\alpha = \boldsymbol{x}^T A\boldsymbol{x} \rightarrow$ by definition $\alpha = \sum_{j=1}^{d}\sum_{i=1}^{d} A_{ij}x_i x_j$ Differentiating with respect to the k^{th} element of $\boldsymbol{x}$ we have $\dfrac{\partial \alpha}{\partial x_k} = \Big[\sum_{j=1}^{d} A_{kj}x_j + \sum_{i=1}^{d} A_{ik}x_i\Big]\ \forall k \in \{1, 2, \ldots, d\}$
  Consequently $\dfrac{\partial \alpha}{\partial \boldsymbol{x}} = \Big[\boldsymbol{x}^T A^T + \boldsymbol{x}^T A\Big] = \boldsymbol{x}^T\big(A^T + A\big)$. But since A^T is symmetric $\implies A^T = A$.
  Therefore $\dfrac{\partial \alpha}{\partial \boldsymbol{x}} = \boldsymbol{x}^T(A^T + A) = \boldsymbol{x}^T(A + A) = \boldsymbol{x}^T 2A \implies \dfrac{1}{2}\dfrac{\partial \alpha}{\partial \boldsymbol{x}} = \dfrac{1}{2}\dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{x}^T A\boldsymbol{x}\Big) = \boldsymbol{x}^T A$
- It is obvious that $\dfrac{\partial}{\partial \boldsymbol{x}}(c) = 0$ since c is a scalar term with no dependence on $\boldsymbol{x}$.
- Thus, $\nabla f(\boldsymbol{x}) = \boldsymbol{x}^T A + \boldsymbol{b}^T$. However, the gradient needs to be a column vector to be consistent with notation from the course. To this end, we can slightly adjust such that $\boxed{\nabla f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}}$ to get a gradient of dimension $d \times 1$ in column vector form.

- Now we'll find ∇^2 by taking $\dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{x}^T A + \boldsymbol{b}^T\Big) = \dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{x}^T A\Big) + \dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{b}^T\Big)$
- Since $\boldsymbol{b}^T$ has no dependence on $\boldsymbol{x} \implies \dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{b}^T\Big) = 0$
- Now let's examine $\dfrac{\partial}{\partial \boldsymbol{x}}\Big(\boldsymbol{x}^T A\Big) = A$. Let's show this, where $\boldsymbol{x}^T$ is $1 \times d$ and A is $d \times d$.

$$\boldsymbol{x}^T A = \begin{bmatrix} x_1 & x_2 & \ldots & x_d \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,d} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,d} \\ \vdots & \vdots & \ddots & vdots \\ a_{d,1} & a_{d,2} & \ldots & a_{d,d} \end{bmatrix} =$$

$$\begin{bmatrix} (x_1)(a_{1,1}) + (x_2)(a_{2,1}) + \ldots + (x_d)(a_{d,1}) & \ldots & (x_1)(a_{1,d}) + (x_2)(a_{2,d}) + \ldots + (x_d)(a_{d,d}) \end{bmatrix}$$

Let's look at the partial derivative wrt just one of the components of $\boldsymbol{x}$, x_1:

$$\frac{\partial}{\partial x_1}\Big(\boldsymbol{x}^T A\Big) = \Big[\Big(a_{1,1} + 0 + \ldots + 0\Big) \quad \Big(0 + a_{1,2} + 0 + \ldots + 0\Big) \quad \ldots \quad \Big(0 + \ldots + 0 + a_{1,d}\Big)\Big]$$

which is simply the first row of A.

Thus for all x_i together, $\dfrac{\partial}{\partial \boldsymbol{x}} = \begin{bmatrix} \text{row 1 of A} \\ \text{row 2 of A} \\ \vdots \\ \text{row } d \text{ of A} \end{bmatrix} = A \implies$ the Hessian of f is the matrix A.

$\boxed{\nabla^2 f(\boldsymbol{x}) = A}$

- From the course notes it is given that:
  (a) f is convex $\iff \nabla^2 f(\boldsymbol{x})$ is PSD $\forall \boldsymbol{x} \in \mathbb{R}^d$
  (b) f is strictly convex $\impliedby \nabla^2 f(\boldsymbol{x})$ is PD $\forall \boldsymbol{x} \in \mathbb{R}^d$

- Therefore, since $\nabla^2 f(\boldsymbol{x}) = A$ and by (a) and (b) above we know that:

$\boxed{f \text{ is convex if and only if } A \text{ is PSD } \forall \boldsymbol{x} \in \mathbb{R}^d}$

$\boxed{\text{If } A \text{ is PD } \forall \boldsymbol{x} \in \mathbb{R}^d \text{ then } f \text{ is strictly convex}}$

4. Using the definition of convexity, prove that the function $f(x) = x^3$ is not convex.

- Defintion of convexity: we say that f is convex if $f(t\boldsymbol{x} + (1-t)\boldsymbol{y}) \leq tf(\boldsymbol{x}) + (1-t)f(\boldsymbol{y})$ $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and $t \in [0,1]$
- To prove a negative statement we simply need to provide a counterexample.
  Here "prove x^3 is not a convex function" simply means find some x, y, and t such that the inequality does not hold. The need to provide only one counterexample is thanks to $\forall$, the universal quantifier, since it mandates that the inequality hold for all cases of $\boldsymbol{x}$, $\boldsymbol{y}$, and t in order for the function to be convex.
- In this case, since $f(x) = x^3 : \mathbb{R} \to \mathbb{R}$ we simply need to find some $x \in \mathbb{R}$, some $y \in \mathbb{R}$, and some $t \in [0,1]$ such that the inequality $f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$ does not hold.
- Let $x = 0$ and $y = -1$ and substitute these values into the inequality giving:
  $\Big[\big((t)(0) + (1-t)(-1)\big)^3 \leq (t)(0^3) + (1-t)(-1)^3\Big] \to \Big[\big(-1+t\big)^3 \leq (-1+t)\Big]$
  $\to \Big[t^3 - 3t^2 + 3t - 1 \leq -1 + t\Big] \to \Big[t(t^2 - 3t + 2) \leq 0\Big] \to \Big[t^2 - 3t + 2 \leq 0\Big]$
- So for $x = 0$ and $y = -1$ the inequality for convexity to hold reduces to $\Big[t^2 - 3t + 2 \leq 0\Big]$
  Per the definition of convexity, this must hold for all $t \in [0,1]$.
  Let $t = \dfrac{1}{2}$, and evaluate in the inequality giving: $\Big[\left(\dfrac{1}{2}\right)^2 - 3\left(\dfrac{1}{2}\right) + 2 \leq 0\Big] \to \Big[\dfrac{1}{4} + \dfrac{6}{4} + \dfrac{8}{4} \leq 0\Big]$
  $\to \Big[\dfrac{3}{4} \leq 0\Big]$.
  Obviously since $\Big[\dfrac{3}{4} \not\leq 0\Big] \implies f(x) = x^3$ is not a convex function.
- To reiterate: since we found a set of function inputs ($x = 0$ and $y = -1$) and a value t ($t = \frac{1}{2}$) for which the inequality doesn't hold for the function $f(x) = x^3$, we have proven that $f(x) = x^3$ is not a convex function.

5. Using the fact that $f(x) = x^3$ is twice continuously differentiable, prove that f is not convex.

- In the notes we are given the following property: $\big[f \text{ is convex } \iff \nabla^2 f(\boldsymbol{x}) \text{ is PSD } \forall \boldsymbol{x} \in \mathbb{R}^d\big]$.
- Also, we know that a matrix A is PSD if $\boldsymbol{x}^T A \boldsymbol{x} \geq 0 \ \forall x \in \mathbb{R}^d$.
  Modifying this slightly for a function $g : \mathbb{R} \to \mathbb{R}$ of one variable , we have that a function $g(x)$ is PSD if $g(x) \geq 0 \ \forall x \in \mathbb{R}$ where $g(x)$ is the Hessian of some other function $f : \mathbb{R} \to \mathbb{R}$.
- To prove the desired result, we simply need to show that the Hessian of f is not PSD for some $\boldsymbol{x} \in \mathbb{R}$ (thanks again to the universal quantifier $\forall$ we only need to give one counterexample).
- Since the inputs of the function $f(x) = x^3$ are $x \in \mathbb{R} \implies \nabla f$ and $\nabla^2 f$ will be a "1×1 vector and matrix."
  In other words, the gradient and the Hessian for f will both just be functions of one variable, x, rather than the typical form of the gradient being a vector and the Hessian being a matrix.
- $\Big[\nabla f = \dfrac{\partial}{\partial x}\big(x^3\big) = 3x^2\Big] \to \Big[\nabla^2 f = \nabla\big(\nabla f\big) = \dfrac{\partial}{\partial x}\big(3x^2\big) = 6x\Big]$. From the definition of the Hessian in the second bullet, let $\nabla^2 f = g(x) = 6x$.
- If we let $\Big[\big(x = -1\big) \in \mathbb{R}\Big] \implies \Big[\nabla^2 f = g(-1) = 6(-1) = -6\Big]$.
  Per the definition in the second bulleted item we can see that $g(x) = 6x$ is not a PSD Hessian since $\big[g(-1) = -6\big] \not\geq 0$, and $g(x)$ must be $\geq 0 \ \forall x$ in order for $g(x)$ to be PSD.
  Since $\Big[g(x) = \nabla^2 f \text{ is not PSD}\Big] \implies f$ is not convex. Therefore we have shown that f is not convex by providing a counterexample, and have proven the desired result.

6. A function f is concave if $-f$ is convex. Prove that for all $x > 0$, the function $f(x) = \ln(x)$ is concave.

- Let's prove that $\Big[-f(x) = -ln(x) \text{ is convex } \forall x > 0 \Big] \implies \Big[f(x) = ln(x) \text{ is concave } \forall x > 0 \Big]$
- Since $-ln(x)$ is a twice continuously differentiable function, let's use the property $\Big[f \text{ is convex} \iff \nabla^2 f \text{ is PSD } \forall x \in \mathbb{R}^d \Big]$ to show that $-ln(x)$ is convex.
- Let $\Big[g(x) = -ln(x) \Big] \rightarrow \Big[\nabla g = (-1)(x^{-1}) \Big] \rightarrow \Big[\nabla^2 g = (-1)(-1)(x^{-2}) = \frac{1}{x^2} \Big]$
- Recall from the previous problem that a Hessian ∇^2 is PSD if every input x outputs a value ≥ 0. In the case of a function $f : \mathbb{R} \rightarrow \mathbb{R}$ gives a gradient and Hessian that are just functions rather than a vector and a matrix. Therefore rather than PSD being a matrix A such that $\boldsymbol{x}^T A \boldsymbol{x} \geq 0$ $\forall \boldsymbol{x} \in \mathbb{R}^d$ we define a Hessian to be some function h such that $h(x)$ is the second derivative of f and such that $h(x) \geq 0 \ \forall x \in \mathbb{R}$.
- We can see that $\Big[x^2 > 0, \forall x \in \mathbb{R} > 0 \Big] \implies \Big[\frac{1}{x^2} > 0 \Big] \implies \Big[\nabla^2 g \text{ is PSD } \forall x \in \mathbb{R} > 0 \Big]$
  $\implies \Big[\Big(g(x) = -ln(x) = -f(x) \Big) \text{ is convex } \forall x \in \mathbb{R} > 0 \Big]$
  $\implies \Big[f(x) = ln(x) \text{ is concave } \forall x \in \mathbb{R} > 0 \Big]$ proving the desired result.

7. Let $f(x) = ax + b$ where $a, b \in \mathbb{R}$. f is called an *affine function* (a linear function plus an offset).

   (a) Prove that f is both convex and concave.

   (b) Does f have a global minimum or a global maximum?

   (c) Are there any other functions that are twice continuously differentiable and both convex and concave that do not have the form of an affine function? Explain your reasoning.

(a)
- To show that f is convex, let's use the fact that it is twice continuously differentiable and show that it has a PSD Hessian.
- $\Big[f(x) = ax + b\Big] \rightarrow \Big[\nabla f = a\Big] \rightarrow \Big[\nabla^2 f = 0\Big]$. Since $\Big[\big[\nabla^2 f = 0\big] \geq 0 \; \forall x \in \mathbb{R}\Big] \implies \Big[\nabla^2 f \text{ is PSD}\Big] \implies \Big[\text{f is convex}\Big]$
- To show that f is concave, let's show that $-f$ is convex in a manner nearly identical to how we showed f to be convex. Define $g(x) = -f(x) = -ax - b$.
- $\Big[g(x) = -ax - b\Big] \rightarrow \Big[\nabla g = -a\Big] \rightarrow \Big[\nabla^2 g = 0\Big]$. Since $\Big[\big[\nabla^2 g = 0\big] \geq 0 \; \forall x \in \mathbb{R}\Big] \implies \Big[\nabla^2 g \text{ is PSD}\Big] \implies \Big[\big(g = -f\big) \text{ is convex}\Big] \implies \Big[f \text{ is concave}\Big]$
- We have shown that f is both convex and concave.

(b)
- KIND OF. In the case of $f(x) = 0x + b$, b is both the global maximum and global minimum since all function outputs have a value of b.
- But, for $f(x) = ax + b$ when $a \neq 0$ the function f does not have a global max or a a global min. We might want to say that [global max $= +\infty$] and [global min $= -\infty$], but these are not true extrema because we can always make $+\infty$ a little larger and $-\infty$ a little smaller, creating new extrema. Doing this iteratively, we'd always be able to increase $+\infty$ and decrease $-\infty$, thus having no definable global maximum or global minimum.

(c)
- $\boxed{\textit{No.}}$ There are not any other functions that are twice continuously differentiable and both convex and concave that do not have the form of an affine function.
- From part (a) we can deduce that some arbitrary function $h(x)$ must have $\nabla^2 h = 0$ in order to be both convex and concave.
- Integrating $\nabla^2 h = 0$ we get $\nabla = a$ where a is some constant $\in \mathbb{R}$.
  Integrating $\nabla h = a$ we get back to $h(x) = ax + b$ where b is also some constant $\in \mathbb{R}$.
- Thus only affine functions are twice continuously differentiable and are both convex and concave since they are the only function with a second derivative (Hessian) $= 0$.
  Stated similarly, polynomial functions of degree 1 are the only twice continuously differentiable functions whose second derivative (Hessian) $= 0$ (since polynomial function of degree 1 are synonymous with affine functions).