Minor 2 solution

1. **[2.5 pts]** Answer `True` or `False`.
   Each correct answer scores 0.25 point, each incorrect answer scores −0.25 point and each missing answer scores 0 point.

   (a) The absolute error loss function is more robust to outliers than the squared error loss function T

   (b) (Full batch) Gradient Descent solution is guaranteed to converge to a local minimum always F (saddle pts)

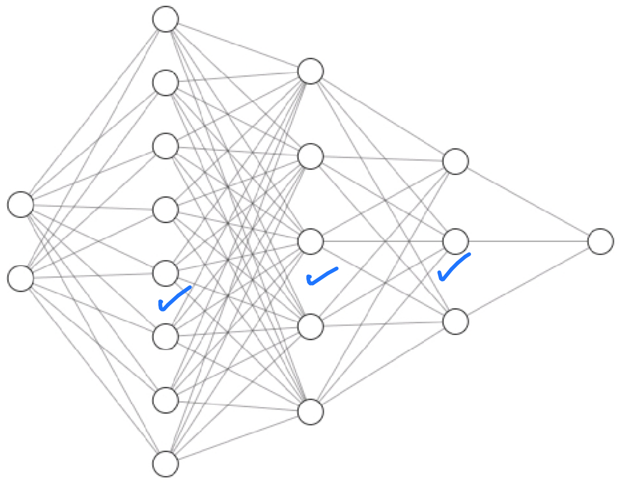
   (c) The statement: *It takes less time to navigate regions with gentle slopes* is true for momentum gradient descent but not for gradient descent T

   (d) The parameters of support vector regression have a closed form solution F

   (e) Neural Networks are parametric methods T

   (f) The weights in deep neural network always have a closed form solution, but it is to expensive to compute when the number of data points is large. F

   (g) The number of hidden layers of the following neural network is 4



F

   (h) Initialization of neural network weights can affect learning of the weights during the training process T

   (i) CNNs are less sensitive to spatial translation of objects within an image than fully connected NNs T

   (j) Weight sharing in CNN means a kernel is used throughout multiple locations of the whole input image T

2. [**2.5 pts**] Derive a relation between sigmoid activation and `tanh` activation function, where the respective activation functions are defined as:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$
$$\texttt{tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(z/2+z/2)}}$$

$$= \frac{1}{1+e^{-z/2}\cdot e^{-z/2}}$$

$$= \frac{e^{z/2}}{e^{z/2}+e^{-z/2}}$$

$$= \frac{e^{z/2} - e^{-z/2} + e^{-z/2}}{e^{z/2}+e^{-z/2}}$$

$$= \frac{e^{z/2} - e^{-z/2}}{e^{z/2}+e^{-z/2}} + \frac{e^{-z/2}}{e^{z/2}+e^{-z/2}}$$

$$= \tanh(z/2) + \frac{1}{1+e^{z}}$$

$$\sigma(z) = \tanh(z/2) + \sigma(-z)$$] Upto this ✓ 2 marks

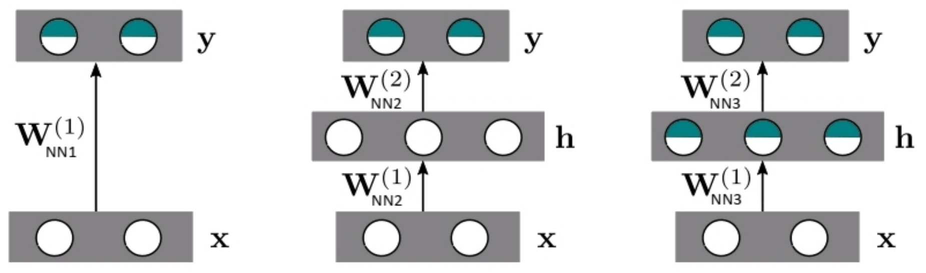
$$= \tanh\left(\frac{z}{2}\right) + \frac{e^{-z}}{e^{-z}+1}$$

$$= \tanh\left(\frac{z}{2}\right) + \frac{e^{-z}+1-1}{e^{-z}+1}$$

$$\sigma(z) = \tanh\left(\frac{z}{2}\right) + 1 - \sigma(z)$$

$$\boxed{\sigma(z) = \frac{1}{2}\tanh\left(\frac{z}{2}\right) + \frac{1}{2}}$$] 2.5

3. [**2 pts**] Consider the following three fully connected feedforward neural networks:



(a) NN1 (b) NN2 (c) NN3

White circular nodes have no activation function, whereas half-shaded-half-white circular nodes have a nonlinear activation function

- **NN1**: No hidden layer; directly maps the inputs to outputs
- **NN2**: Has a linear hidden layer (meaning $\mathbf{h} = \mathbf{W}^{(1)}_{\text{NN2}}\,\mathbf{x}$ at hidden layer)
- **NN3**: Has a non-linear hidden layer (meaning $\mathbf{h} = \sigma\left(\mathbf{W}^{(1)}_{\text{NN3}}\,\mathbf{x}\right)$ at hidden layer)

You may assume same nonlinear activation functions at the outputs and zero bias terms.

Choose the correct relation for the expressive power of the three neural networks. Explain in a few words.

(A) NN1 > NN2 > NN3
(B) NN3 > NN2 > NN1
[✓ (C) NN3 > NN2 = NN1]
(D) NN3 = NN2 > NN1

Neural networks are in essence composition of functions

$$\underline{y} = \underline{f}(\underline{h}(\underline{x})) \quad \text{in this case}$$

Show equivalence $\boxed{1.5 \text{ pts}}$

$\underline{\text{NN 1}}$: $\underline{y} = \sigma\left(\underline{\underline{W}}_{NN}^{(1)}\,\underline{x}\right)$

$\underline{\text{NN 2}}$: $\underline{y} = \sigma\left(\underline{\underline{W}}_{NN}^{(2)}\,\underline{h}\right) = \sigma\left(\underline{\underline{W}}_{NN}^{(2)}\,\underline{\underline{W}}_{NN}^{(1)}\,\underline{x}\right)$

$= \sigma\left(\underline{\underline{W}}^{eqv}\,\underline{x}\right)$

$\underline{\text{NN 3}}$: $\underline{y} = \sigma\left(W_{NN}^{(2)}\,h\right) = \sigma\left(\underline{\underline{W}}_{NN}^{(2)}\,\sigma\left(\underline{\underline{W}}_{NN}^{(1)}\,\underline{x}\right)\right)$ ✓ $\boxed{0.5 \text{ pts}}$

Flexibility-wise NN1 = NN2, and NN3 is more nonlinear

No marks without correct explanation

4. **[2 pts]** In gradient descent optimization, the learning rate is an important parameter. Briefly describe a problem encountered in optimization if we choose the learning rate too be (a) too high, and (b) too low for full batch gradient descent.

(a) Too high : Unstable (loss blows up)

parameter estimates diverge

parameter values oscillate a lot

Any of these

1 pt

(b) Too low : Parameter values change very slowly

1 pt

6. **[0.5 pt]** Suppose we have a convolution layer which takes as input an array $\underline{x} = [x_1\ x_2\ x_3]$ and convolves $\underline{x}$ with the 1D-kernel $\underline{w} = [2\quad -3]$.

Now think of a fully connected (FC) layer which computes the same functions. Assume that it only has a weight matrix and no bias, so it computes $\underline{y} = \underline{\underline{W}}\ \underline{x}$, where the output $\underline{y}$ is a vector of length 2. Write out the 2×3 weight matrix which makes this FC layer equivalent to the convolution layer above.

While learning the kernel, it does not matter whether you learn the kernel in its flipped form or the original form. However, if you are GIVEN a kernel, the convolution operation requires you to flip the kernel

$$\underline{w} * \underline{x} = [2\quad -3] * [x_1\quad x_2\quad x_3]$$

$$\begin{matrix} [\,x_1 & x_2 & x_3\,] \\ [\,-3 & 2\,] & \\ & [\,-3 & 2\,] \end{matrix} \equiv \begin{bmatrix} -3 & 2 & 0 \\ 0 & -3 & 2 \end{bmatrix}$$

✓

0.5

5. **[3 pts]** Consider a hidden layer with ReLU activation functions:

$$z_i = \sum_j w_{ij} x_j + b_i$$
$$h_i = \text{ReLU}(z_i)$$

where,

$$\text{ReLU}(a) = \begin{cases} a & a > 0 \\ 0 & a \leq 0 \end{cases}$$

(a) Draw the computation graph.

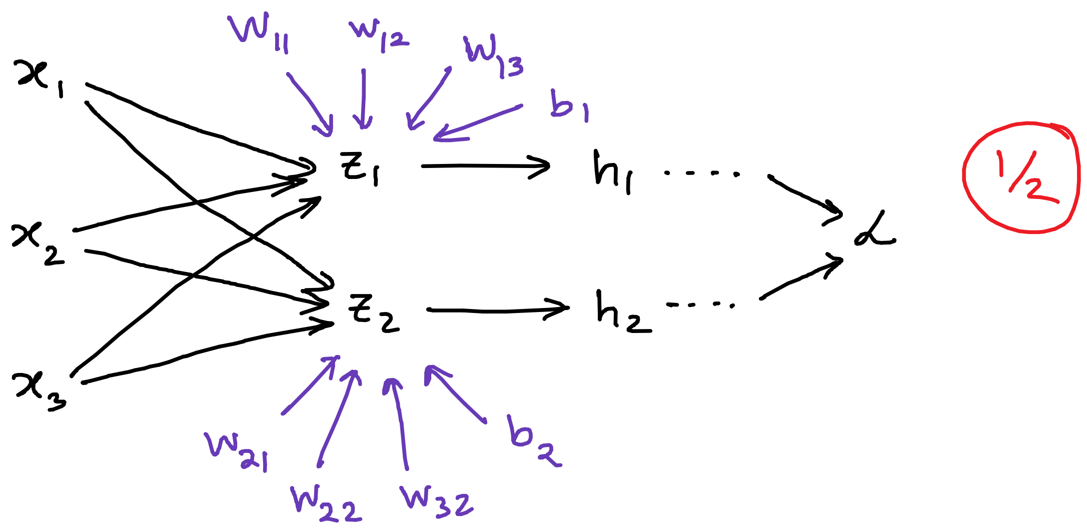
(b) Write out the backpropagation rules for computing the gradients of $\bar{z}_i$, $\bar{x}_j$, and $\bar{w}_{ij}$, in terms of $\bar{h}_i$
Note, the notation $\bar{v} = \frac{\partial \mathcal{L}}{\partial v}$, where $\mathcal{L}$ is some loss function.

(c) Based on your answer to part (b), for what values of x_j and z_i are we guaranteed that $\bar{w}_{ij} = 0$?

To draw the computation graph, one can either consider a vector format or use an example with i and j assigned to certain fixed value.

(a) Computation Graph

Considering an example with 3 inputs and 2 hidden nodes



(b) $\bar{h}_i = \frac{\partial \mathcal{L}}{\partial h_i}$

$$\bar{z}_i = \frac{\partial \mathcal{L}}{\partial z_i} = \frac{\partial \mathcal{L}}{\partial h_i} \times \frac{\partial h_i}{\partial z_i} = \bar{h}_i \frac{\partial h_i}{\partial z_i} = \bar{h}_i \frac{\partial \, \mathrm{ReLU}(z_i)}{\partial z_i}$$

1/2

$$\Rightarrow \bar{z}_i = \begin{cases} \bar{h}_i & z_i \geq 0 \\ 0 & z_i < 0 \end{cases}$$

$$\bar{x}_j = \frac{\partial \mathcal{L}}{\partial x_j} = \sum_i \frac{\partial \mathcal{L}}{\partial z_i} \times \frac{\partial z_i}{\partial x_j} = \sum_i \bar{z}_i w_{ij}$$ 1

$$\bar{w}_{ij} = \frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial z_i} \frac{\partial z_i}{\partial w_{ij}} = \bar{z}_i x_j$$ 1/2

(c) 1/2 If $x_j = 0 \longrightarrow \bar{w}_{ij} = 0$

1/2 If $z_i < 0 \longrightarrow \bar{z}_i = 0$

$\Big\}$ $\bar{w}_{ij} = 0$

7. **[2.5 pts]** Show that any kernel $\kappa(\underline{x}, \underline{x}')$ that is defined as an inner product between feature vectors $\underline{\phi}(\underline{x})$,

$$\kappa(\underline{x}, \underline{x}') = \underline{\phi}(\underline{x})^T \underline{\phi}(\underline{x})$$

is always positive semidefinite (PSD).

The Gram matrix $\underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}})$ obtained from a kernel defined via inner product will have the (ij)th element as

$$K_{ij} = \underline{\phi}(x^{(i)})^T \underline{\phi}(x^{(j)})$$

2.5 pts

For proving positive semi-definiteness

$$\underline{v}^T \underline{\underline{K}}(\underline{\underline{X}}, \underline{\underline{X}}) \underline{v} = \sum_i \sum_j v_i \underline{\phi}(x^{(i)})^T \underline{\phi}(x^{(j)}) v_j$$

$$= \Big(\sum_i v_i \underline{\phi}(x^{(i)}) \Big)^T \Big(\sum_j v_j \underline{\phi}(x^{(j)}) \Big) \geq 0$$

2

0.5

Hence $K(\underline{x}, \underline{x}')$ is always PSD if defined using inner product