1. **Training, validation, and testing.**

   (a) (5 points) Briefly explain the purpose of the training, validation, and testing set. Comment on how many times each set should be used when doing k-fold cross validation.

   The training set is used to optimize parameters and is used as many times as desired. The validation set is used to optimize hyperparameters and is used once per iteration of hyperparameter optimization. There are k such sets in k-fold cross validation. The testing set is used to estimate the generalized accuracy of the model and is used once, at the very end.

   (b) (10 points) Consider a dataset where a signal evolves through time. The data is sampled at an extremely high rate, meaning that adjacent time points are highly correlated. Consider also that this dataset has few trials. Your colleague comes up with an idea to increase the number of effective trials: he decides to subsample the data with no overlapping time points (so, for instance time points $1, 3, 5, 7, 9...$ could represent one effective trial and time points $2, 4, 6, 8, ...$ another). He tells you he now has twice the number of effective trials.

      i. (3 points) Why might it be beneficial to subsample the data to increase the number of effective trials?

      Increasing the number of effective trials might be beneficial since having more data is usually helpful in training models, even if the data actually come from the same place. Subsampling is one technique to achieve this.

ii. (7 points) After subsampling the data, your colleague randomly split his trials (of which he now has double) into a training and validation set. He tells you that his validation accuracy is much higher than anything you have achieved. Suggest why this might be the case.

This method would produce deceptively high validation accuracy because the training and validation sets would be highly correlated.

(c) (5 points) Another colleague splits the data into 80% training, 10% validation, and 10% test. This colleague did not subsample the data. He has experimented with various architectures, each time training on the training set, validating on the validation set, and then testing on the testing set. Based on his testing accuracy, he would see which hyperparameters worked well, and make modifications to his model, reiterating the above process. Will the final test accuracy reported be an accurate proxy for how well his model will generalize to new data? Briefly explain.

No. By optimizing hyperparameters using the testing accuracy, he is tuning the model to that test set. To make the testing accuracy more general, he should use the validation accuracy for this and run the test set only once, at the very end.

2. **Backpropagation.** Consider a 3 layer neural network (NN), with $\mathbf{x} \in \mathbb{R}^n$ as input and $\mathbf{y} \in \mathbb{R}^m$ as the target value. The NN is constructed as the following:

$$\begin{aligned} \mathcal{L} &= \frac{1}{2}||h_3(h_2(h_1(\mathbf{x}))) - \mathbf{y}||^2 \\ h_1(\mathbf{x}) &= \text{PReLU}(\mathbf{W}_1\mathbf{x}) \\ h_2(\mathbf{x}) &= \text{ELU}(\mathbf{W}_2\mathbf{x}) \\ h_3(\mathbf{x}) &= \mathbf{W}_3\mathbf{x} \end{aligned}$$

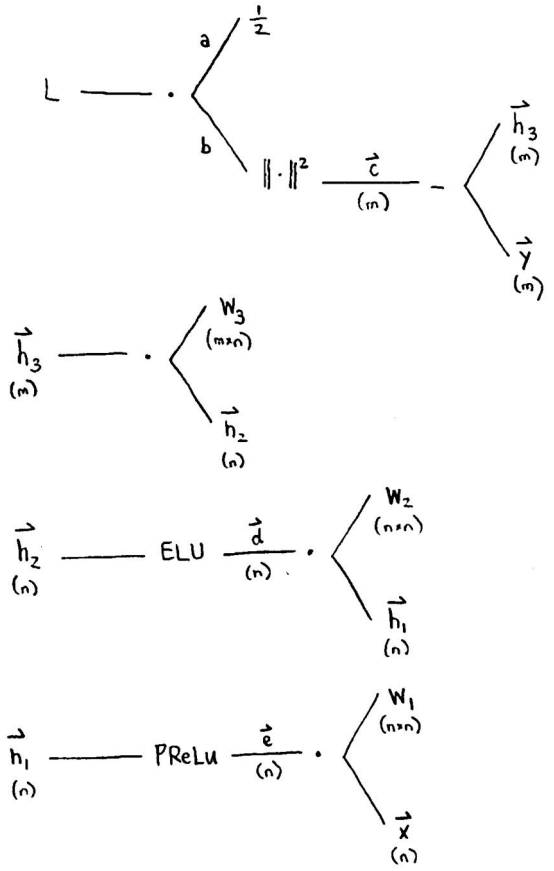
where

- $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n\times n}$ and $\mathbf{W}_3 \in \mathbb{R}^{m\times n}$
- For the PReLU unit, $f(x) = \max(\alpha x, x)$.
- For the ELU unit, $f(x) = \max(\alpha e^x - 1, x)$.

$\frac{\partial}{\partial x}\text{PReLu}(x) = \mathbb{I}(x\geq 0) + \mathbb{I}(x<0)\alpha$

$\frac{\partial}{\partial x}\text{ELU}(x) = \mathbb{I}(x\geq 0) + \mathbb{I}(x<0)\alpha e^x$

(a) (5 points) Draw the computational graph for this neural network.



(b) (15 points) Calculate $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}_2}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}_3}$ using backpropagation. You may define intermediate variables and write the gradients in terms of these intermediate variables.

Hint: For a ReLU function $f(x) = \max(0, x)$, we compute the gradient for $x < 0$ and $x > 0$ separately. For instance, $\frac{\partial f(x)}{\partial x} = \mathbb{1}\{x > 0\} \cdot 1 + \mathbb{1}\{x < 0\} \cdot 0$. Compute the gradient for PReLU and ELU with the same method. (Additional space provided on next page.)

@b $\quad L = ab \quad \frac{\partial L}{\partial b} = a = \frac{1}{2}$

@$\vec{c}$ (m) $\quad b = \|\vec{c}\|^2 \quad \frac{\partial L}{\partial \vec{c}} = \frac{\partial b}{\partial \vec{c}} \frac{\partial L}{\partial b} = 2\vec{c} \cdot \frac{1}{2} = \vec{c}$

@$\vec{h}_3$ (m) $\quad \vec{c} = \vec{h}_3 - \vec{y} \quad \frac{\partial L}{\partial \vec{h}_3} = \frac{\partial L}{\partial \vec{c}} = \vec{c}$

@W_3 $(m \times n)$ $\quad \vec{h}_3 = W_3 \vec{h}_2 \quad \frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial \vec{h}_3} \vec{h}_2^T = \vec{c}\,\vec{h}_2^T$

@$\vec{h}_2$ (n) $\quad \vec{h}_3 = W_3 \vec{h}_2 \quad \frac{\partial L}{\partial \vec{h}_2} = W_3^T \frac{\partial L}{\partial \vec{h}_3} = W_3^T \vec{c}$

@$\vec{d}$ (n) $\quad \vec{h}_2 = \mathrm{ELU}(\vec{d}) \quad \frac{\partial L}{\partial \vec{d}} = \frac{\partial \vec{h}_2}{\partial \vec{d}} \frac{\partial L}{\partial \vec{h}_2} = f'(\vec{d}) \odot W_3^T \vec{c}$

@W_2 $(n \times n)$ $\quad \vec{d} = W_2 \vec{h}_1 \quad \frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \vec{d}} \vec{h}_1^T = \left[f'(\vec{d}) \odot W_3^T \vec{c} \right] \vec{h}_1^T$

@$\vec{h}_1$ (n) $\quad \vec{d} = W_2 \vec{h}_1 \quad \frac{\partial L}{\partial \vec{h}_1} = W_2^T \frac{\partial L}{\partial \vec{d}} = W_2^T \left[f'(\vec{d}) \odot W_3^T \vec{c} \right]$

@$\vec{e}$ (n) $\quad \vec{h}_1 = \mathrm{PReLU}(\vec{e}) \quad \frac{\partial L}{\partial \vec{e}} = \frac{\partial \vec{h}_1}{\partial \vec{e}} \frac{\partial L}{\partial \vec{h}_1} = g'(\vec{e}) \odot W_2^T \left[f'(\vec{d}) \odot W_3^T \vec{c} \right]$

@W_1 $(n \times n)$ $\quad \vec{e} = W_1 \vec{x} \quad \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \vec{e}} \vec{x}^T = \left(\vec{g}'(\vec{e}) \odot W_2^T \left[f'(\vec{d}) \odot W_3^T \vec{c} \right] \right) \vec{x}^T$

$$\begin{cases} \frac{\partial L}{\partial W_1} = \left(\vec{g}'(\vec{e}) \odot W_2^T \left[f'(\vec{d}) \odot W_3^T \vec{c} \right] \right) \vec{x}^T \\ \frac{\partial L}{\partial W_2} = \left[f'(\vec{d}) \odot W_3^T \vec{c} \right] \vec{h}_1^T \\ \frac{\partial L}{\partial W_3} = \vec{c}\,\vec{h}_2^T \end{cases} \quad \text{where} \quad \begin{cases} \vec{h}_1 = h_1(\vec{x}),\ \vec{h}_2 = h_2(h_1(\vec{x})),\ \vec{h}_3 = h_3(h_2(h_1(\vec{x}))) \\ \vec{c} = \vec{h}_3 - \vec{y} \\ \vec{d} = W_2 \vec{h}_1 \\ \vec{e} = W_1 \vec{x} \\ f'(x) = \frac{\partial}{\partial x} \mathrm{PReLU}(x) = \mathbb{I}(x \geq 0) + \mathbb{I}(x < 0)\alpha \\ g'(x) = \frac{\partial}{\partial x} \mathrm{ELU}(x) = \mathbb{I}(x \geq 0) + \mathbb{I}(x < 0)\alpha e^x \end{cases}$$

(Additional space for question 2b.)

3. **Regularization.**

(a) (5 points) You are tasked with training a feedforward neural network. Your boss asks you to shrink the effective size of the model by using regularization. How can you complete the task?

I could apply L_1 regularization (i.e., add a term like $\sum_{\vec{w}} \|\vec{w}\|_1$ to the cost function), observe the weights that go to zero, and discard the corresponding features.

(b) (5 points) Why can $L2$ regularization be referred to as "weight decay"?

L_2 regularization can be called "weight decay" because it penalizes large weights and encourages smaller (decayed) weights.

(c) (5 points) How does batch normalization help a neural network to be more robust to initialization?

By normalizing the mean and variance of activations to 0 and 1, respectively, which helps keep diminishing and exploding activations under control.

(d) (5 points) How does dropout act as a regularizer?

By randomly zeroing activations, which is similar to bagging and increases generalizability by encouraging the preservation of only important features

4. **Optimization.**

(a) (12 points) Suppose, oddly, that an optimizer takes several gradient steps and arrives back at a same parameter setting, $\mathbf{W}$, that is *exactly* the same as in a prior step. This setting of $\mathbf{W}$ does not correspond to a local optima of the loss function. Imagine you are optimizing this neural network using a batch algorithm, i.e., using the entire training set to calculate a gradient. You are also using a fixed learning rate.

i. (4 points) Consider that the optimizer was a naive gradient descent optimizer (with no momentum or adaptive gradients). Is the gradient step that you will take the second time you are at this parameter setting, $\mathbf{W}$, (circle one)

- larger (in magnitude)
- smaller (in magnitude)
- ◉ the exact same, or
- can't be determined (i.e. not enough information)

than the gradient step the first time the optimizer was at the parameter setting, $\mathbf{W}$? Justify your answer.

In naive gradient descent, there is no state other than the current parameters, and there is no randomness from batching, so the calculated gradient must be the same as before.

ii. (4 points) Consider that the optimizer was Adagrad. Is the gradient step that you will take the second time you are at this parameter setting, **W**, (circle one)

- larger (in magnitude)
- (•) smaller (in magnitude)
- the exact same, or
- can't be determined (i.e. not enough information)

than the gradient step the first time the optimizer was at the parameter setting, **W**? Justify your answer.

Adagrad's square gradient accumulator will have grown since before, so by dividing by it, we will get a smaller gradient step.

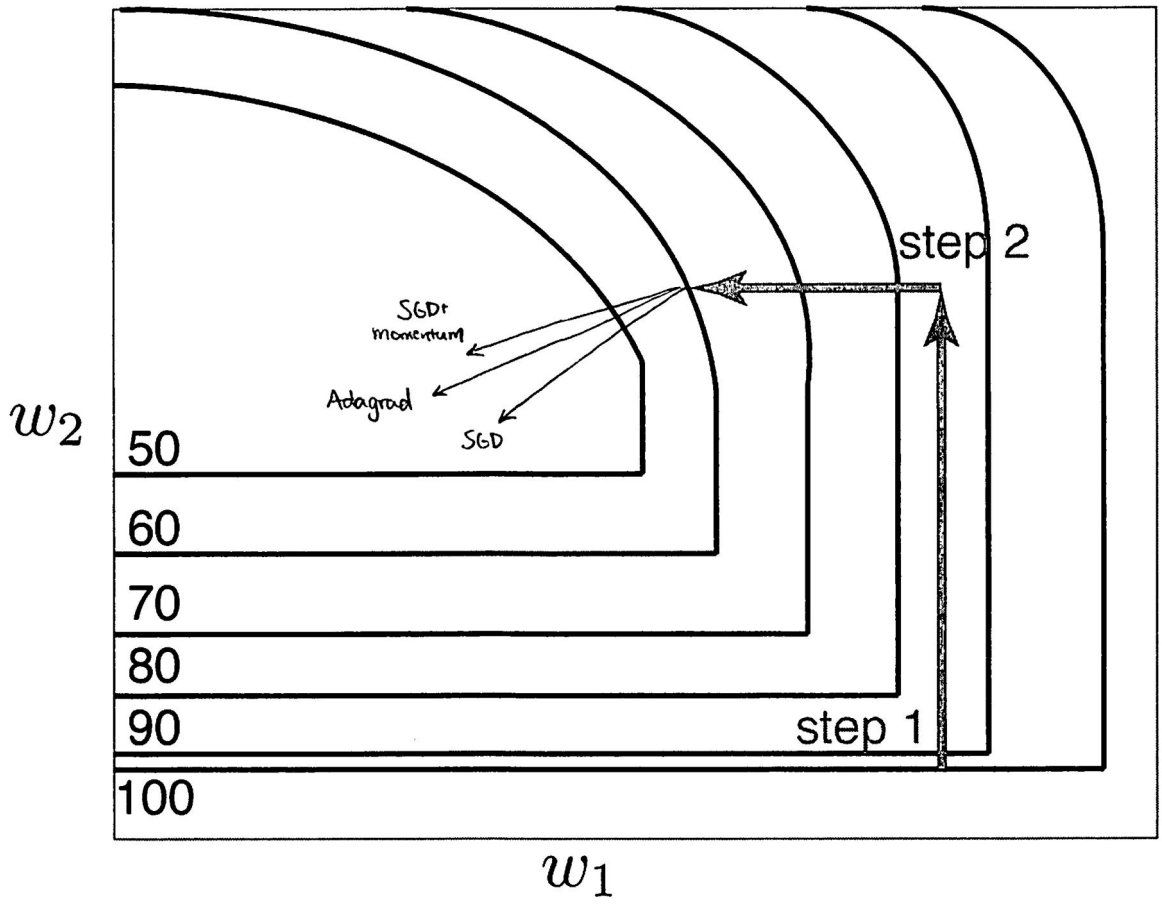
iii. (4 points) Consider that the optimizer was RMSprop. Is the gradient step that you will take the second time you are at this parameter setting, **W**, (circle one)

- larger (in magnitude)
- smaller (in magnitude)
- the exact same, or
- (•) can't be determined (i.e. not enough information)

than the gradient step the first time the optimizer was at the parameter setting, **W**? Justify your answer.

It depends on the size of the gradients encountered before each arrival at W, as well as β.

(b) (8 points) The following figure is a contour plot where the contour lines denote the value of the loss as a function of two weight variables (corresponding to the x and y-axis). Imagine we have taken two gradient steps given by the gray arrows. Sketch on the same plot the weight update steps taken by SGD, SGD+momentum, and Adagrad after step 2. Do not perform any calculations; the sketch should be arrived at through intuition. Give at most a two sentence explanation for each arrow you've drawn.



SGD: step based solely on local gradient

SGD+ momentum: a mix of SGD and the previous step

Adagrad: SGD divided by sum of previous square gradients — amplifies horizontal component, reduces vertical component

5. **Convolutional neural networks.** Consider a convolutional layer C followed by a max pooling layer P. The input to layer C is $120 \times 120 \times 50$. Layer C has 20 filters, each of which is of size 4×4. The convolution padding is 1 and the stride is 2. Layer P performs max pooling over each of the layer C's output feature maps, over a 3×3 receptive field, and stride 1. Given $x_1, x_2, ..., x_n$ all scalars, we assume:

   - A scalar multiplication $x_i \cdot x_j$ accounts for one FLOP;
   - A scalar addition $x_i + x_j$ accounts for one FLOP;
   - A max operation $\max(x_1, x_2, ..., x_n)$ accounts for $n - 1$ FLOPs.

   You do not need to calculate the products you write out (e.g., answers maybe left in terms like "$(x \cdot y \cdot w) \cdot z$").

   (a) (5 points) What is the total number of trainable parameters? Please account for the bias term.

   $20 \times (4 \times 4 \times 50 + 1)$
   filters dim bias

   (b) (4 points) What is the size of layer C's output feature map?

   $W = 1 + (120 + 2 - 4)/2 = 60$
   padding, rec. field, stride

   $20 \times 60 \times 60$

(c) (4 points) What is the size of layer P's output feature map?

$W = 1 + (60 - \underset{\text{rec. field}}{3}) / \underset{\text{stride}}{1} = 58$

$20 \times 58 \times 58$

(d) (7 points) How many FLOPs are there in layers C and P during one forward pass? Please include the bias term when calculating the FLOPs.

per output of C:

$(4 \times 4 \times 50) + (4 \times 4 \times 50 - 1) + 1 = 2(4 \times 4 \times 50)$

mult sum bias

total for C:

$(20 \times 60 \times 60) \times 2(4 \times 4 \times 50)$

dim

per output of P:

$3 \times 3 - 1 = 8$

rec. field

total for P:

$(20 \times 58 \times 58) \times 8$

dim

total:

$(20 \times 60 \times 60) \times 2(4 \times 4 \times 50) + (20 \times 58 \times 58) \times 8$