Afternoon section

1. **[1pt]** *Carla tells you, "Overfitting is bad, so you want to make sure your model is simple enough that the test error is no higher than the training error." Is she right or wrong? Justify your answer.*

   Note that there are good arguments for either side, so you will receive full credit as long as you justify your answer well.

   Solution: The intended answer is that Carla's statement is false, because in order to achieve the best generalization error, we need the model to be powerful enough to explain the data. Making it so simple that there is no training/test gap could make it too simplistic to explain the data. Rather, we want to tune the complexity by minimizing the error on a validation set.

   Another acceptable answer is that Carla is correct, if we are interested in interpreting the parameters that we fit. E.g., statisticians often want to interpret linear regression coefficients. In this case, we might want the model to be simple enough that it not reflect any idisyncrasies in the training data.

   Marking: Full marks for discussion of the tradeoff between underfitting and overfitting, or of choosing the complexity to minimize validation error.

   Mean: 0.45/1

2. **[1pt]** *Suppose you are training a neural net using stochastic gradient descent (SGD), and you compute the cost function on the entire training set after each update. TRUE or FALSE: if you ever see the training cost increase after an SGD update, that means your learning rate is too large. Justify your answer.*

   Solution: FALSE. SGD moves downhill on average, but individual updates can certainly increase the cost.

   Marking: Half a point for the answer, half a point for the correct explanation.

   Mean: 0.77/1

3. *Suppose you are given the following two-dimensional dataset for a binary classification task:*

x_1	x_2	t
0	*0*	*1*
1	*1*	*1*
1	*0*	*0*

You use a linear model with a hard threshold activation function, and a dummy dimension x_0 so that w_0 functions as a bias. You would like to find a weight vector in the strictly feasible region, i.e. none of the training examples should lie on the decision boundary.

(a) **[1pt]** *Each of the training examples gives a constraint on w_0, w_1, and w_2. Write down all three constraints.*

Solution:

$$\begin{aligned} w_0 &> 0 \\ w_0 + w_1 + w_2 &> 0 \\ w_0 + w_1 &< 0 \end{aligned}$$

(b) **[1pt]** *Find a set of weights which satisfies all the constraints. You do not need to show your work or justify your answer. Hint: pick w_0 first, then w_1, then w_2.*

Solution: Since the scale is arbitrary, let's start with $w_0 = 1$. The third equation tells us $1+w_1 < 0$, so let's try $w_1 = -2$. The second equation tells us $1-2+w_2 > 0$, so let's pick $w_2 = 2$. So one solution is:

$$w_0 = 1 \quad w_1 = -2 \quad w_2 = 2.$$

Mean: 1.79/2

4. **[2pts]** *In Homework 5, we analyzed dropout for a linear regression model; the predictions had the form*

$$y = \sum_j m_j w_j x_j,$$

where the m_j's were i.i.d. Bernoulli random variables. Let's modify the dropout algorithm so that the m_j's take on continuous values, and they are i.i.d. Gaussian random variables with mean 1 and variance σ^2.

Under this model, determine the variance of the predictions, Var$[y]$, *as a function of the x_j's and w_j's. Show your work. Hint: use the properties of variance.*

Solution:

$$\begin{aligned}
\mathrm{Var}(y) &= \mathrm{Var}\left(\sum_j m_j w_j x_j\right) \\
&= \sum_j \mathrm{Var}\left(m_j w_j x_j\right) && \text{by independence} \\
&= \sum_j w_j^2 x_j^2 \,\mathrm{Var}(m_j) && \text{by the scalar multiplication rule} \\
&= \sigma^2 \sum_j w_j^2 x_j^2
\end{aligned}$$

Mean: 1.42/2

5. [**2pts**] *Briefly explain the difference between invariant and equivariant feature detectors. Give an example of an equivariant operation.*

   Solution: A feature detector is invariant if it does not change (much) in response to a particular transformation of the input. It is equivariant if it transforms the same way as the input does. Convolution is an example of an equivariant operation which we've covered in this course; it is equivariant with respect to translation. There are lots more examples of equivariant operations, and any are acceptable as long as you explain why they're equivariant.

   Marking: 1 point for the explanation of invariance and 1 point for the explanation of equivariance. We're not counting the example of equivariance since we couldn't think of a reasonable way to mark it.

   Mean: 0.96/2

6. *Recall that in the domain of binary classification, the logistic activation function combined with cross-entropy loss does not suffer from saturated units. Now let's suppose we don't like making overly confident predictions, so we transform the predictions y to lie in the interval $[0.1, 0.9]$. In other words, we take*

$$y = 0.8\,\sigma(z) + 0.1,$$

   where σ denotes the logistic function. We still use cross-entropy loss.

   (a) [**1pt**] *For a positive training example, sketch the cross-entropy loss as a function of z. Your sketch doesn't need to be precise, but it should make clear the asymptotic behavior as $z \to \pm\infty$. (You should label any asymptote lines.)*

Solution: The sketch should show a roughly sigmoidal function with horizontal asymptotes at $\mathcal{L} = -\log 0.1$ as $z \to -\infty$ and $\mathcal{L} = -\log 0.9$ as $z \to \infty$.

(b) **[1pt]** *Based on your answer to Part (a), will gradient descent on this model suffer from saturation when the predictions are very wrong? Why or why not?*

Solution: Yes it will suffer from saturation, because the loss is flat for very negative z, which means $\partial \mathcal{L} / \partial z \approx 0$.

Mean: 1.02/2

7. *Suppose we somehow know that the weights for a two-dimensional regression problem should lie on the unit circle. We can parameterize the weights in terms of the angle θ, i.e. let $(w_1, w_2) = (\cos\theta, \sin\theta)$. The model and loss function are as follows:*

$$\begin{aligned} w_1 &= \cos\theta \\ w_2 &= \sin\theta \\ y &= w_1 x_1 + w_2 x_2 \\ \mathcal{L} &= \frac{1}{2}(y-t)^2 \end{aligned}$$

(a) **[1pt]** *Draw the computation graph relating θ, w_1, w_2, y, and $\mathcal{L}$.*

(b) **[2pts]** *Determine the backprop update rules which let you compute the derivative $\mathrm{d}\mathcal{L}/\mathrm{d}\theta$.*

Your equations should refer to previously computed values (e.g. your formula for $\overline{z}$ should be a function of $\overline{y}$). You do not need to show your work, but it may help you get partial credit. The first two steps have been filled in for you.

Solution:

$$\begin{aligned} \overline{\mathcal{L}} &= 1 \\ \overline{y} &= \overline{\mathcal{L}} \cdot (y - t) \\ \overline{w_1} &= \overline{y}\, x_1 \\ \overline{w_2} &= \overline{y}\, x_2 \\ \overline{\theta} &= -\overline{w_1} \sin\theta + \overline{w_2} \cos\theta \end{aligned}$$

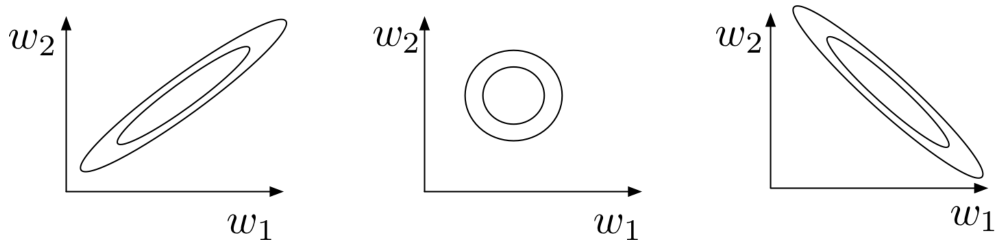
Marking: One point for $\overline{w_1}$ and $\overline{w_2}$, and the other point for $\overline{\theta}$.

Mean: 2.80/3

8. **[2pts]** *Suppose you are given a two-dimensional linear regression problem (with no bias parameter), using the following dataset:*

x_1	x_2	t
83.1	*-82.4*	*3.3*
83.2	*-82.8*	*1.5*
83.5	*-82.1*	*2.0*
$\vdots$	$\vdots$	$\vdots$

Circle the contour plot which best represents the cost function for this regression problem. Justify your answer.



Solution: Circle the first one. The reason is that any choice of weights such that $w_1 - w_2$ takes a given value will result in fairly similar predictions, so they will achieve fairly similar loss.

Marking: One point for the correct answer, and one point for a correct explanation.

Mean: 0.93/2

Night Section

1. *Consider a convolution layer. The input consists of 6 feature maps of size* 20×20*. The output consists of 8 feature maps, and the filters are of size* 5×5*. The convolution is done with a stride of 2 and zero padding, so the output feature maps are of size* 10×10*.*

   For both parts, you can leave your expression as a product of integers; you do not need to actually compute the product. You do not need to show your work, but doing so can help you receive partial credit.

(a) **[1pt]** *Determine the number of weights in this convolution layer.*

Solution: There's one filter for each pair of an input and output feature map, and the filters are each 5×5. Therefore, the number of weights is $6 \times 8 \times 5 \times 5 = 1200$.

(b) **[1pt]** *Now suppose we made this a fully connected layer, but where the number of input and output units are kept the same as in the network described above. Determine the number of weights in this layer.*

Solution: There are $20 \times 20 \times 6$ units in the input layer and $10 \times 10 \times 8$ units in the output layer, so the number of weights is $20 \times 20 \times 6 \times 10 \times 10 \times 8 = 1{,}920{,}000$.

Mean: 1.13/2

2. **[2pts]** *Recall that the learning rate is an example of a hyperparameter which must be tuned. Alice wants to tune the learning rate by doing a grid search over values and choosing the one which achieves the lowest training error. Bob tells her it's important to tune all hyperparameters on a separate validation set. Who is right? Justify your answer.*

bob generalization consideration

Note that there are good arguments for either side, so you will receive full credit as long as you justify your answer well.

Solution 1: Alice is right. It is OK to tune the learning rate on the training set, because the learning rate relates to optimization rather than generalization.

Solution 2: Bob is right. The learning rate can affect generalization in various ways. E.g., a lower learning rate can have a similar effect to early stopping, since the optimizer will have made less progress. Also, SGD has a regularization effect (see the Lecture 9 slides on stochastic regularization), and this effect might be stronger with a larger learning rate.

Marking: One point for pointing out that measuring generalization is the reason for using a separate validation set. Full credit for an answer like one of the above.

Mean: 0.74/2

3. **[2pts]** *Your job is to design a multilayer perceptron which receives three binary-valued (i.e. 0 or 1) inputs x_1, x_2, x_3, and outputs 1 if exactly two of the inputs are 1, and outputs 0 otherwise. All of the units use a hard threshold activation function:*

$$z = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Specify weights and biases which correctly implement this function. You do not need to explain your solution. Hint: one of the hidden units should activate if 2 or more inputs are on, and the other should activate if all of the inputs are on.

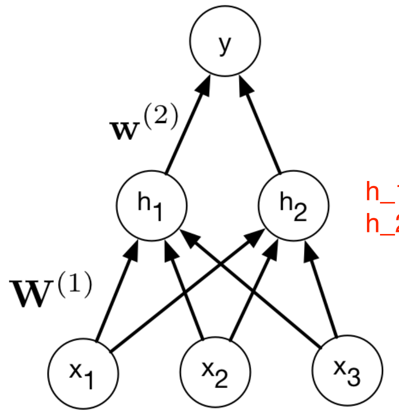
Solution:

$$\mathbf{W}^{(1)} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

$$\mathbf{b}^{(1)} = \begin{pmatrix} -1.5 \\ -2.5 \end{pmatrix}$$

$$\mathbf{w}^{(2)} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$b^{(2)} = -0.5$$



h_1 2 or 3 input =1
h_2 3 input = 1

Marking: One point for the first layer, and one point for the second layer.

Mean: 1.67/2

4. **[3pts]** *You want to train the following model using gradient descent. Here, the input x and target t are both scalar-valued.*

$$\begin{aligned} z &= w_0 + w_1 x + w_2 x^2 \\ y &= 1 + e^z \\ \mathcal{L} &= \frac{1}{2}(\log y - \log t)^2. \end{aligned}$$

Determine the backprop rules which will let you compute the loss derivative $\partial \mathcal{L}/\partial w_2$.

Your equations should refer to previously computed values (e.g. your formula for $\overline{z}$ should be a function of $\overline{y}$). You do not need to show your work, but it may help you get partial credit. The dummy step has been filled in for you.

Solution:

$$\begin{aligned} \overline{\mathcal{L}} &= 1 \\ \overline{y} &= \overline{\mathcal{L}}\,\frac{1}{y}\,(\log y - \log t) \\ \overline{z} &= \overline{y}\,e^z \\ \overline{w_2} &= \overline{z}\,x^2 \end{aligned}$$

Marking: One point for each of the three equations.

Mean: 2.70/3

5. [**2pts**] *Suppose we are training a linear regression model using gradient descent with momentum. The update rules are as follows:*

$$p_j \leftarrow \mu p_j - \frac{\alpha}{N} \sum_{i=1}^{N} x_j^{(i)} (y^{(i)} - t^{(i)})$$
$$w_j \leftarrow w_j + p_j$$

Now suppose that, as usual, the inputs are stored as an $N \times D$ matrix $\mathbf{X}$, where N is the number of data points and D is the input dimension. The targets and predictions are represented as N-dimensional vectors $\mathbf{t}$ and $\mathbf{y}$, respectively. The weights and momentum vector are represented as D-dimensional vectors $\mathbf{w}$ and $\mathbf{p}$, respectively. Write the vectorized form of these update rules, i.e. mathematical expressions which could be translated into NumPy without requiring for-loops.

You may assume $\mathbf{y}$ has already been computed. You do not need to show your work, though it may help you receive partial credit.

Solution:

$$\mathbf{p} \leftarrow \mu \mathbf{p} - \frac{\alpha}{N} \mathbf{X}^\top (\mathbf{y} - \mathbf{t})$$
$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{p}$$

Marking: 1.5 points for $\mathbf{p}$ and half a point for $\mathbf{w}$.

Mean: 1.83/2

6. [**1pt**] Write the formula for the logistic function $\sigma(z)$. You do not need to justify your answer or explain anything.

Solution:

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

Mean: 0.72/1

7. [**1pt**] *Recall that the perceptron algorithm cycles through the training examples, applying the following rule:*

```
$z^{(i)} \leftarrow \mathbf{w}^T \mathbf{x}^{(i)}$
If $z^{(i)} t^{(i)} \leq 0$,
    $\mathbf{w} \leftarrow \mathbf{w} + t^{(i)} \mathbf{x}^{(i)}$
```

(Recall that the targets take values in $\{-1, 1\}$.) Suppose we make the inequality strict in the conditional, i.e. we update the weights only if $z^{(i)} t^{(i)} < 0$. What would go wrong? You may assume the weights are initialized to 0. Hint: what happens on the first training example?

Solution: For the first training example, z evaluates to 0, so the strict version of the inequality will not be triggered, and then the weights won't be updated. Then the weights will still be zero for the second example, and so on, so the weights never actually get updated.

Mean: 0.89/1

8. **[2pts]** *Suppose the word representations $\mathbf{r}_1$ and $\mathbf{r}_2$ are both unit vectors. Show that the Euclidean distance $\|\mathbf{r}_1 - \mathbf{r}_2\|$ is a monotonically decreasing function of the dot product $\mathbf{r}_1^\top \mathbf{r}_2$. Hint: start by expanding out the formula for squared Euclidean distance.*

Let's expand out the formula for squared Euclidean distance:

$$\begin{aligned}\|\mathbf{r}_1 - \mathbf{r}_2\|^2 &= (\mathbf{r}_1 - \mathbf{r}_2)^\top (\mathbf{r}_1 - \mathbf{r}_2) \\ &= \mathbf{r}_1^\top \mathbf{r}_1 - 2\mathbf{r}_1^\top \mathbf{r}_2 + \mathbf{r}_2^\top \mathbf{r}_2 \\ &= 2 - 2\mathbf{r}_1^\top \mathbf{r}_2\end{aligned}$$

Hence,

$$\|\mathbf{r}_1 - \mathbf{r}_2\| = \sqrt{2 - 2\mathbf{r}_1^\top \mathbf{r}_2},$$

which is a monotonically decreasing function of $\mathbf{r}_1^\top \mathbf{r}_2$.

Mean: 1.28/2