CS 181 2024 Midterm 1 Topic List

The best way to prepare for the midterm is to review homeworks, section notes, lecture recaps, lecture concept checks, and the midterm practice questions.

The midterm will be conceptual and analytical, testing ideas and understanding. You are not expected to memorize formulas such as PDFs, or to memorize matrix cookbook rules, but you should be familiar with methods of probability theory (e.g. Bayes Rule) and the various models we've studied so far in the course.

Here is a brief list of topics that you could expect to be asked about. This list emphasizes the main focus areas and is not fully inclusive:

- Linear regression: least squares loss, can differentiate least squares loss and solve for weights analytically, be able to work with and interpret alternate (simple) loss functions when given, understand parametric vs. non-parametric regression.
- Basis functions [general idea, not specific versions].
- Generative model of linear regression, noise, maximum likelihood estimation.
- Linear classification. Perceptron algorithm, hinge loss. Logistic regression, understand (but don't memorize) derivative. Shape of decision boundaries. Different loss functions (e.g. hinge vs 0/1 vs logistic).
- Generative classification via class-conditional distributions (e.g. Gaussian or categorical/Naive Bayes), use of Bayes Rule for prediction, use of MLE. Understanding of Naive Bayes and Logistic regression as a pair of models for the same task, where NB is generative and LR is discriminative [LR does not model $p(\mathbf{x}, y)$].
- Bias-variance trade-off (not full derivation, but understand the role of each term, the intuition of "bias" and "variance", and connection to over-fitting).
- Use of (cross-)validation for model selection and to avoid over-fitting, role and mathematical form of major types of regularization, particularly when used with linear regression problems.
- Bayesian methods: terminology, MAP, posterior predictive, use of conjugate distributions (Beta-Bernoulli, Normal-Normal, don't need to memorize forms of PDFs), Bayesian linear regression.
- Neural nets: basic notation for weights in layers and use of sigmoid and ReLU activation functions [you don't need to memorize the functions but you need to understand that they are applied element-wise to vectors to create non-linearities]. Use of neural nets for both classification and regression. Idea of forward-prop and then back-prop.
- Support Vector Machines (SVM): Hard max-margin formulation, soft margin formulation, dual formulation, "kernel trick".

CS181 2024 Midterm 1 Practice Questions

IMPORTANT: This practice midterm includes many examples of mathematical derivation questions. The actual midterm will include a small number of these kinds of questions, combined with less technical, conceptual questions to test your understanding.

1. **Linear Regression**

Consider a one-dimensional regression problem with training data $\{x_i, y_i\}$. We seek to fit a linear model with no bias term:

$$\hat{y} = wx$$

a. Assume a squared loss $\frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ and solve for the optimal value of w^*.

b. Suppose that we have a generative model of the form $\hat{y} = wx + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and w is known. Given a new x, what is the expression for the probability of $\hat{y}$?
Note: The univariate Gaussian PDF is:

$$\mathcal{N}(a|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{-(a-\mu)^2}{2\sigma^2}\right)$$

c. Now assume that w is random and that we have a prior on w with known variance s_0^2:

$$w \sim \mathcal{N}(0, s_0^2)$$

Write down the form of the *posterior* distribution over w. Take logs and drop terms that don't depend on the data $\mathcal{D}$ and prior parameters, but you do not need to simplify further (i.e. you do not need to complete the square to make it look like a Normal).

a. This question is mostly just math. Take the derivative with respect to w, set it equal to 0, and solve for w:

$$-\sum_{i=1}^{N}(y_i - wx_i)x_i = 0$$

$$-\sum_{i=1}^{N} y_i x_i + w\sum_{i=1}^{N} x_i^2 = 0$$

$$w^* = \frac{\sum_{i=1}^{N} y_i x_i}{\sum_{i=1}^{N} x_i^2}$$

b. This is a definitions question. Use the form of the univariate Gaussian:

$$p(y|x) = \mathcal{N}(y|wx, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{-(y-wx)^2}{2\sigma^2}\right)$$

c. Here we combine everything above.
Prior:

$$p(w) = \mathcal{N}(w|0, s_0^2)$$

Likelihood:

$$p(\mathcal{D}|w) = p(\mathbf{y}|\mathbf{x}, w) = \prod_{i=1}^{N} \mathcal{N}(y_i|wx_i, \sigma^2)$$

Posterior:

$$p(w|\mathcal{D}) \propto p(w)p(\mathcal{D}|w) = \mathcal{N}(w|0, s_0^2) \prod_{i=1}^{N} \mathcal{N}(y_i|wx_i, \sigma^2)$$

Taking logs, we get

$$\ln p(w|\mathcal{D}) = \text{const} + \frac{-w^2}{2s_0^2} + \sum_{i=1}^{N} \frac{-(y_i - wx_i)^2}{2\sigma^2}.$$

It is worth noting the similarity to the ridge regression loss function.

2. **Regularization**

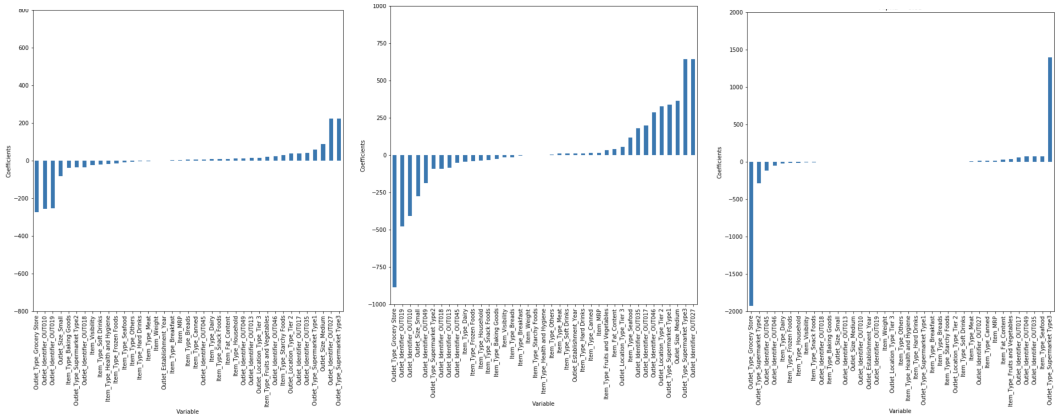
Suppose we predict sales according to features of a sold item and its sales location. Consider using a linear regression model $y = \mathbf{w}^T\mathbf{x}$. We try three different losses:

(a) No regularization: $L(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{x}_n)^2$

(b) Lasso regression: $L(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{x}_n)^2 + \frac{\lambda}{2}||\mathbf{w}||_1$

(c) Ridge regression: $L(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - \mathbf{w}^T x_n)^2 + \frac{\lambda}{2}||\mathbf{w}||_2^2$

We train our model with each loss, which gives us different final coefficients. The coefficients for each model are shown in the plots below (in random order):



Now answer the following questions:

a. Which plot of coefficients corresponds to which loss function? Why?

b. How can we expect the plots to change as we increase λ?

a. The first plot corresponds to ridge regression since it has slightly smaller coeficcients than the second plot, but the coefficients are not zeroed out. The second plot corresponds to no regularization, as its coefficients haven't zeroed out and they are the largest of the three. The third plot corresponds to Lasso regression because many of the coefficients have been driven to 0, and lasso regression leads to sparse solutions.

b. As we increase λ, larger coefficients get more penalized. Thus, we can predict that coefficients will be smaller overall in the ridge regression case, and that the coefficients will be even more sparse in the lasso regression case. As for the no regularization case, we should expect no difference, since λ doesn't affect it.

Note: Plots are from https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/.

3. **Basis Functions**

Basis functions $\phi(x)$ are often important in both regression and classification tasks. For 1-dimensional data x:

$$h(x; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(x)$$

Without them, linear and logistic regression can only fit linear functions to the data. The following question asks you to determine if a class of basis function can linearly separate the data $\mathcal{D} = \{(x, y)\} = \{(-\pi, 1), (0, -1), (\pi, 1)\}$. If so, find a setting of $\boldsymbol{w}$ that correctly classifies the data-points (assuming a logistic regression setup).

a. $\boldsymbol{\phi}(x) = [1, x]^T$

b. $\boldsymbol{\phi}(x) = [1, x, x^2]^T$

c. $\boldsymbol{\phi}(x) = [1, x, x^4]^T$

d. $\boldsymbol{\phi}(x) = [1, \cos x]^T$

a. No, this choice of basis cannot perfectly separate the data.

b. Yes. Set $\mathbf{w} = [-1, 0, 1]^T$.

c. Yes. Set $\mathbf{w} = [-1, 0, 1]^T$.

d. Yes. Set $\mathbf{w} = [0, -1]^T$.

4. **Probabilistic Linear Regression**

In class, we derived the optimal w^* to maximize the likelihood of training data given Normally distributed noise. In this problem, you will explore an alternative distribution on the noise of labels y.

Assume 1-dimensional data x, and that

$$\epsilon \sim Lap(0, 1)$$
$$y|x, \epsilon = wx + \epsilon$$

where ϵ is a Laplace random variable. The probability density function for a $Lap(\mu, b)$ random variable is given by

$$p(x) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right)$$

You can also take as given that when you linearly transform any Laplace random variable by a constant, the distribution of the new transformed variable is still Laplace with a linearly transformed mean. For example, if some random variable $a \sim Lap(0, b)$, then for any constant c, $a + c \sim Lap(0 + c, b)$.

a. What is the distribution of random variable y given x?

b. Given data $\{(x_i, y_i)\}_{i=1}^N$, write down an expression for the likelihood of observing the data in terms of unknown parameter w.

c. Write down an expression for the negative log likelihood of the data.

d. Recall from Section 2.6.2 of the CS 181 textbook that for probabilistic regression with Normally distributed noise, maximizing the likelihood function is equivalent to minimizing the squared error. What kind of loss function $\mathcal{L}(y, \hat{y})$ corresponds to minimizing your expression from part (c) for Laplacian noise?

e. Given that $\frac{d}{da}|a| = \text{sign}(a)$, where $\text{sign}(a) = 1$ when $a \geq 0$, $\text{sign}(a) = -1$ when $a < 0$, take the gradient of the negative log likelihood with respect to w. You can leave your expression in terms of the sign() operator.

   Does this model class seem more or less sensitive to outliers than probabilistic regression with Normally distributed noise? Why?

   Note: You won't be expected to solve for the optimal w^* in an expression with *sign*() operators on the actual midterm.

a. $y|x \sim Lap(wx, 1)$

b.

$$\begin{aligned} p(\mathcal{D}|w) &= p(\mathbf{y}|\mathbf{X}, w) \\ &= \prod_{i=1}^{N} p(y_i|x_i, w) \\ &= \prod_{i=1}^{N} \frac{1}{2} \exp(-\frac{|y_i - wx_i|}{1}) \\ &= \frac{1}{2^N} \exp(-\sum_{i=1}^{N} |y_i - wx_i|) \end{aligned}$$

c. Taking the log, we get

$$-\left(-N\log(2) - \sum_{i=1}^{N} |y_i - wx_i|\right) = N\log(2) + \sum_{i=1}^{N} |y_i - wx_i|.$$

d. The expression from (c) is equivalent to minimizing the L1 loss of $(y_i - \hat{y}_i)$.

e.

$$\begin{aligned} \frac{d}{dw}\left(N\log(2) + \sum_{i=1}^{N} |y_i - wx_i|\right) &= 0 \\ -\sum_{i=1}^{N} \text{sign}(y_i - wx_i)x_i &= 0 \end{aligned}$$

This model class (L1 loss) is less sensitive to outliers than normally distributed noise (L2 loss). L2 loss magnifies large differences $(y - \hat{y})^2$ much more than L1 loss; as such outliers in L2 contribute much more to the overall loss over a given dataset.

5. **Bayesian Linear Regression**

Consider the following setup. Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N, \mathbf{x}_i \in \mathbb{R}^D, y_i \in \mathbb{R}$. Consider the model:

$$y_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2)$$

The likelihood will then be:

$$P(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I}) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}\right)$$

Apply a conjugate Gaussian prior, specifically one where each weight is i.i.d.:

$$P(\mathbf{w}) = \mathcal{N}(0, \sigma_0^2\mathbf{I}) = \prod_{j=1}^{D} \frac{1}{\sigma_0\sqrt{2\pi}} \exp\left(-\frac{w_j^2}{2\sigma_0^2}\right)$$

a. Find the MAP estimate for the weights as a simplified argmax or argmin expression in non-matrix form. Do NOT derive the full posterior and do NOT solve the argmin/argmax equation. Just set up the right equation, which can include a sum over data points. (Hint: recall $\mathbf{w}_{MAP} = \arg\max_{\mathbf{w}} P(\mathbf{w}|\mathcal{D})$)

b. What does the expression that you derived in part (a) remind you of?

c. What happens to the posterior with a wider (larger σ_0^2) or narrower (smaller σ_0^2) prior? In particular, how it will affect both the mean and the variance of the posterior? You may want to make a connection based on the results in part (b).

d. The prior used here is Gaussian, which has a PDF of the form:

$$P(\mathbf{w}) = \prod_{j=1}^{D} \frac{1}{\sigma_0\sqrt{2\pi}} \exp\left(-\frac{w_j^2}{2\sigma_0^2}\right) \propto \prod_j \exp(-w_j^2)$$

Another popular prior uses a modification of the Laplace distribution, which can be loosely thought of as a symmetric exponential distribution. The PDF of this distribution is:

$$P(\mathbf{w}) = \prod_{j=1}^{D} \frac{\lambda}{2\sigma} \exp\left(\frac{-\lambda|w_j|}{\sigma}\right) \propto \prod_j \exp(-|w_j|)$$

How do you expect the result in part (a) to be different with a Laplacian prior instead of a Gaussian prior? How do you expect the connection in part (b) to change? Answer this conceptually without doing any math.

a. The MAP is expressed as an arg max of the posterior, which immediately suggest Bayes'

Rule:

$$
\begin{aligned}
\mathbf{w}_{MAP} &= \arg\max_{\mathbf{w}} \; P(\mathbf{w}|\mathcal{D}) \\
&= \arg\max_{\mathbf{w}} \; P(\mathcal{D}|\mathbf{w})P(\mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \; \log P(\mathcal{D}|\mathbf{w}) + \log P(\mathbf{w}) \\
&= \arg\max_{\mathbf{w}} \; \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}\right) + \log \prod_{j=1}^{D} \frac{1}{\sigma_0\sqrt{2\pi}} \exp\left(-\frac{w_j^2}{2\sigma_0^2}\right) \\
&= \arg\max_{\mathbf{w}} \; \log \prod_{i=1}^{N} \exp\left(-\frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}\right) + \log \prod_{j=1}^{D} \exp\left(-\frac{w_j^2}{2\sigma_0^2}\right) \\
&= \arg\max_{\mathbf{w}} \; \sum_{i=1}^{N} -\frac{(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2} + \sum_{j=1}^{D} -\frac{w_j^2}{2\sigma_0^2} \\
&= \arg\min_{\mathbf{w}} \; \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \frac{1}{2\sigma_0^2} \sum_{j=1}^{D} w_j^2 \\
&= \arg\min_{\mathbf{w}} \; \sum_{i=1}^{N} (y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \frac{\sigma^2}{\sigma_0^2} \sum_{j=1}^{D} w_j^2
\end{aligned}
$$

The normalizing constants can be dropped since they do not affect the argmax. The last step, in particular, involves multiplying the entire expression by σ^2, which does not change the maximization. Furthermore, maximizing the expression is the same as minimizing its negation to yield the above result.

b. The expression resembles a squared loss function. The second term is simply a sum of squared weights, which is the penalization term in ridge regression. In fact, the entire expression is exactly the same as ridge regression! To make the connection even more concrete, note that:

$$
\mathbf{w}_{MAP} = \arg\min_{\mathbf{w}} \; \sum_{i=1}^{N} (y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda \sum_{j=1}^{D} w_j^2
$$

where $\lambda = \dfrac{\sigma^2}{\sigma_0^2}$, which is now in the exact same form as ridge regression, with a penalization weight λ that can be directly expressed as a ratio of the likelihood variance and prior variance. In other words, adding a Bayesian prior is the same as ridge regularization!

c. A prior with higher variance suggests more uncertainty, so intuitively, the posterior will be wider as well and have higher variance due to the greater initial uncertainty. Consequently, a narrower prior will result in a narrower posterior.

However, it is not just the variance of the posterior that is affected, but also the mean (which is also the MAP), which actually has a strong connection to regularization. With a wider prior, due to the greater initial uncertainty, the posterior will rely more heavily on the data through the likelihood. With a narrower prior suggesting greater initial

confidence, the posterior will be more restricted by the prior (i.e. pulled closer to the prior mean of 0), which is the exact effect of regularization and goes to show yet again how a Bayesian prior has a regularizing effect.

d. The Laplace distribution PDF uses the absolute value of the weights, rather than the square of the weights. Since the rest of functional form takes the same shape as the Gaussian, the result can be expected to be the same as was derived in part (a) except with the sum of absolute weights instead of squared weights in the second term. In other words, the penalization will then be the Bayesian equivalent of Lasso regression.

6. **Multiclass Classification**

Suppose that we have a K-class classification scenario with training data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$, where the $\mathbf{y}_i$ are 1-hot column vectors. Let $\mathcal{C}_k$ represent a 1-hot vector with a 1 in the k^{th} index.

We model this problem using a neural network with d units in a single hidden layer, expressed as a vector $\boldsymbol{\phi}(\mathbf{x}; \mathbf{W}, \mathbf{w}_0) \in \mathbb{R}^d$, which we simplify as $\boldsymbol{\phi}$. Each entry of $\boldsymbol{\phi}$ applies an activation function to the weighted inputs.

We take a linear combination the values in the hidden layer and pass them to a softmax function to get a final set of K outputs.
For this, let $\mathbf{v}_\ell \in \mathbb{R}^d$ denote the weights corresponding to the ℓth output.

Putting this together, we have:

$$p_{\text{model}}(\mathbf{y} = \mathcal{C}_k | \mathbf{x}; \{\mathbf{v}_\ell\}_{\ell=1}^K, \mathbf{W}, \mathbf{w}_0) = \frac{\exp(\mathbf{v}_k^\top \boldsymbol{\phi})}{\sum_{\ell'=1}^K \exp(\mathbf{v}_{\ell'}^\top \boldsymbol{\phi})}$$

a. Suppose we add the same bias term v_0 to each vector of weights in the final layer, i.e. replace $\mathbf{v}_k^\top \boldsymbol{\phi}$ with $\mathbf{v}_k^\top \boldsymbol{\phi} + v_0$ for some scalar v_0, the same for all k. Does this increase the expressivity of our model? Why or why not?

b. Assuming the sigmoid activation function in the hidden layer, with $\boldsymbol{\phi}(\mathbf{x}; \mathbf{W}, \mathbf{w}_0) = \boldsymbol{\sigma}(\mathbf{W}\mathbf{x} + \mathbf{w}_0)$, write down and simplify the log likelihood of a particular observation $(\mathbf{x}_i, \mathbf{y}_i)$, including constants. You don't need to work with the sigmoid, just adopt this concrete form for $\boldsymbol{\phi}$.

c. How might you train the parameters in this neural network? What is the role of the loss function, sigmoid, and softmax functions? (Answer in words, don't use math.)

a. The same v_0 is added into the exp in the numerator and into each of the exps summed in the denominator. v_0 is constant with respect to choice of a particular class:

$$\begin{aligned} p_{\text{model}}(\mathbf{y} = C_k | \mathbf{x}; \{\mathbf{v}_\ell\}, \mathbf{W}, \mathbf{w}_0) &= \frac{1}{\text{const}} \exp(\mathbf{v}_k^\top \boldsymbol{\phi} + v_0) \\ &= \frac{1}{\text{const}} \exp(\mathbf{v}_k^\top \boldsymbol{\phi}) \exp(v_0) \\ &= \frac{1}{\text{const}'} \exp(\mathbf{v}_k^\top \boldsymbol{\phi}) \end{aligned}$$

So, this is effectively the same as the original formula.

b. Assume that y_i is in class k:

$$\begin{aligned}
\ln p_{\text{model}}(\mathbf{y} = C_k|\mathbf{x}; \{\mathbf{v}_\ell\}, \mathbf{W}, \mathbf{w}_0) &= \ln \frac{\exp(\mathbf{v}_k^\top \boldsymbol{\phi})}{\sum_{\ell=1}^K \exp(\mathbf{v}_\ell^\top \boldsymbol{\phi})} \\
&= \mathbf{v}_k^\top \boldsymbol{\phi} - \ln \sum_{\ell=1}^K \exp(\mathbf{v}_\ell^\top \boldsymbol{\phi}) \\
&= \mathbf{v}_k^\top \boldsymbol{\sigma}(\mathbf{W}\mathbf{x} + \mathbf{w}_0) - \ln \sum_{\ell=1}^K \exp(\mathbf{v}_\ell^\top \boldsymbol{\sigma}(\mathbf{W}\mathbf{x} + \mathbf{w}_0))
\end{aligned}$$

c. Training a neural network is done by forward propagation to find the predictions for the current parameters on the examples in the data, and then back-propagation to calculate derivates of loss with respect to parameters in an efficient way. We can then take a gradient step, for example on a mini-batch if doing SGD. The loss function here would be the negated log likelihood. This is differentiable for the choice of sigmoid and softmax activations.

7. **Probabilistic Generative Classification**

Suppose that we use a Naive Bayes classifier to classify binary feature vectors $\mathbf{x} \in \{0,1\}^D$ into two classes. The class conditional distributions will then be of the form

$$p(\mathbf{x} \,|\, y = C_k) = \prod_{j=1}^{D} \pi_{kj}^{x_j} \left(1 - \pi_{kj}\right)^{(1-x_j)}$$

where $x_j \in \{0,1\}$, and $\pi_{kj} = p(x_j = 1 \,|\, y = C_k)$. This is a Bernoulli Naive Bayes. Assume also that the class priors are $p(y = C_1) = p(y = C_2) = \frac{1}{2}$.

a. How is the quantity $\ln(p(y = C_1 \,|\, \mathbf{x})/p(y = C_2 \,|\, \mathbf{x}))$ used for classification of a new example $\mathbf{x}$?

b. If $D = 1$ (i.e., there is only one feature), use the equations above to write out $\ln \frac{p(y=C_1 \,|\, x)}{p(y=C_2 \,|\, x)}$ for a single binary feature x.

c. Now suppose we change our feature representation so that instead of using just a single feature, we use two redundant features. (i.e., two features that always have the same value). With this feature representation, instead of x we will use $\mathbf{x} = [x, x]^\top$. What is $\ln \frac{p(y=C_1 \,|\, \mathbf{x})}{p(y=C_2 \,|\, \mathbf{x})}$ in terms of the value for $\ln \frac{p(y=C_1 \,|\, x)}{p(y=C_2 \,|\, x)}$ you calculated in part (a.)?

d. In the sense of the performance of the classifier, do you view this as a bug or a useful property?

a. We will predict class C_1 if $p(y = C_1 \,|\, \mathbf{x}) \geq p(y = C_2 \,|\, \mathbf{x})$, and class C_2 otherwise. Equivalently, we will predict C_1 if $\ln(p(y = C_1 \,|\, \mathbf{x})/p(y = C_2 \,|\, \mathbf{x})) \geq 0$, and C_2 otherwise.

b. Because the class priors are the same and the denominators cancel, we have $p(y = C_1 \,|\, x)/p(y = C_2 \,|\, x) = p(y = C_1)p(x \,|\, y = C_1)/p(y = C_2)p(x \,|\, y = C_2) = p(x \,|\, y = C_1)/p(x \,|\, y = C_2)$, and we have:

$$\begin{aligned}
\ln \frac{p(y = C_1 \,|\, x)}{p(y = C_2 \,|\, x)} &= \ln \frac{\pi_{11}^x \left(1 - \pi_{11}\right)^{(1-x)}}{\pi_{21}^x \left(1 - \pi_{21}\right)^{(1-x)}} \\
&= x \ln \pi_{11} + (1 - x) \ln(1 - \pi_{11}) - x \ln \pi_{21} - (1 - x) \ln(1 - \pi_{21})
\end{aligned}$$

c. Because the two features are identical, we will have

$$\begin{aligned}
\ln \frac{p(y = C_1 \,|\, \mathbf{x})}{p(y = C_2 \,|\, \mathbf{x})} &= \ln \frac{\left(\pi_{11}^x \left(1 - \pi_{11}\right)^{(1-x)}\right)^2}{\left(\pi_{21}^x \left(1 - \pi_{21}\right)^{(1-x)}\right)^2} \\
&= \ln \left[\left(\frac{\pi_{11}^x \left(1 - \pi_{11}\right)^{(1-x)}}{\pi_{21}^x \left(1 - \pi_{21}\right)^{(1-x)}} \right)^2 \right] \\
&= 2 \ln \frac{p(y = C_1 \,|\, x)}{p(y = C_2 \,|\, x)}
\end{aligned}$$

(above, we use x to mean either redundant feature in $\mathbf{x}$ and dropped the subscripts)

d. This seems like a useful property! The classifier with the two identical features has

exactly the same behavior as the classifier with just a single feature. In particular,

$$\ln \frac{p(y = C_1 \,|\, \mathbf{x})}{p(y = C_2 \,|\, \mathbf{x})} \geq 0 \quad \Leftrightarrow \quad \ln \frac{p(y = C_1 \,|\, x)}{p(y = C_2 \,|\, x)} \geq 0$$

We are robust to adding redundant information in the feature space. Note: it does not matter that there is a new constant 2 in front of the expression. Only the sign is important for classification.

8. **Overfitting and Underfitting**

Harvard Insta-Ice Unit (HI2U) has built a robot that can deliver 24-hour shaved ice to student houses. To prevent collisions, they train three different approaches to classify camera images as containing nearby tourists or open space; if the robot identifies a tourist in its path, it is programmed to halt. The performances of the classifiers are:

	Training Accuracy	Testing Accuracy
Classifier A	75.3%	74.8%
Classifier B	80.3%	77.8%
Classifier C	90.2%	60.0%

where Classifier B has a more expressive model class than A, and classifier C has both a more expressive model class and more features than A. All the classifiers have closed-form solutions, so HI2U is pretty sure that the training procedure is not hindering performance.

a. If you had to choose either 'over' or 'under': might Classifier A be overfitting or underfitting? Explain your reasoning.

b. If you had to choose either 'over' or 'under': might Classifier C be overfitting or underfitting? Explain your reasoning.

c. If you had to guess yes or no: might more training examples significantly boost the test-time performance of Classifier A? Classifier C? Explain your reasoning.

Hint: throughout, try to relate your reasoning to model bias and model variance.

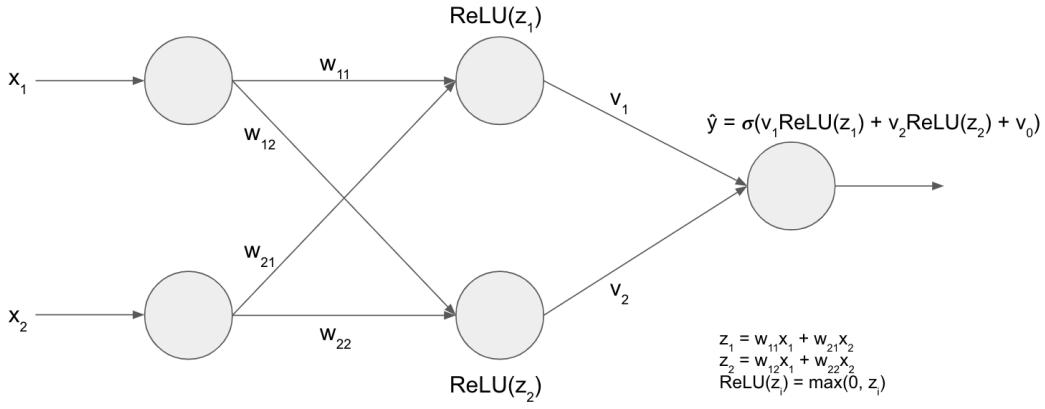
a. Likely it is underfitting, demonstrated by the results of Classifier B. This is likely a bias issue, as we have evidence that a richer model can improve on training accuracy.

b. Likely it is overfitting. 90% training accuracy indicates very little bias, but poor test accuracy shows variance issues.

c. It seems unlikely that more training will help classifier A. There is no indication of a variance issue (train/test accuracy are similar). However for classifier C, more training data would reduce the variance of the rich model.

9. **Neural Networks Part 1**

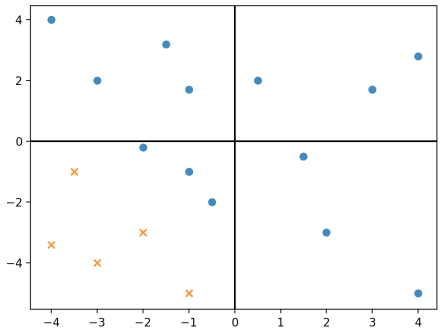
Consider the following 2-layer neural network, which takes in $\mathbf{x} \in \mathbb{R}^2$ and has two ReLU hidden units and a final sigmoid activation. There are no bias weights on the hidden units.



For a binary classification problem with true labels $y \in \{0, 1\}$, we will use the loss function $L = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$.

a. Suppose we update our network with stochastic gradient descent on a data point $\mathbf{x} = [x_1, x_2]^T$.
   i. Calculate the gradient of the loss with respect to v_1.
   ii. Calculate the gradient of the loss with respect to w_{11}.

b. Consider the classification of data points below. Is it possible that this classification was generated by the set of weights $w_{11}, w_{12}, w_{21}, w_{22} = \{1, 0, 0, 1\}$? Why or why not? What if additional hidden layers were applied to further transform the data (still keeping the specified set of weights fixed)?



c. i. Why is it a bad idea in general to have ReLU as the activation function of the output layer?
   ii. Suppose we want to classify our outputs into 5 categories. Why might it be a bad idea to use the label set $\{1, 2, 3, 4, 5\}$? What could we use instead?

a. i.

$$\begin{aligned}\frac{\partial L}{\partial v_1} &= -(y/\hat{y} + (y-1)/(1-\hat{y})) \cdot \hat{y}(1-\hat{y}) \cdot \text{ReLU}(z_1) \\ &= -(y(1-\hat{y}) + (y-1)\hat{y}) \cdot \text{ReLU}(z_1) \\ &= (\hat{y} - y) \cdot \text{ReLU}(z_1)\end{aligned}$$

ii.

$$\begin{aligned}\frac{\partial L}{\partial w_{11}} &= -(y/\hat{y} + (y-1)/(1-\hat{y})) \cdot \hat{y}(1-\hat{y}) \cdot v_1 \cdot \frac{\partial \text{ReLU}(z_1)}{\partial w_{11}} \\ &= -(y(1-\hat{y}) + (y-1)\hat{y}) \cdot v_1 \cdot x_1 \quad \text{if } z_1 > 0, \text{ 0 otherwise} \\ &= (\hat{y} - y) \cdot v_1 \cdot x_1 \quad \text{if } z_1 > 0, \text{ 0 otherwise}\end{aligned}$$

b. Regardless of whether there are additional hidden layers, this classification could not have been generated by the given weights. As described, all points in the bottom left quadrant would map to the origin, so it is not possible for points of differing predicted label to be in that quadrant.

c. i. If the values entering the ReLU layer are mostly negative, gradients will fail to backpropagate through the network.

ii. The numerical values carry unintended meaning; our model will assume that categories 1 and 2 are similar, whereas 1 and 5 are very distinct. We can fix the problem by using one-hot encoding.

10. **Neural Networks Part 2**

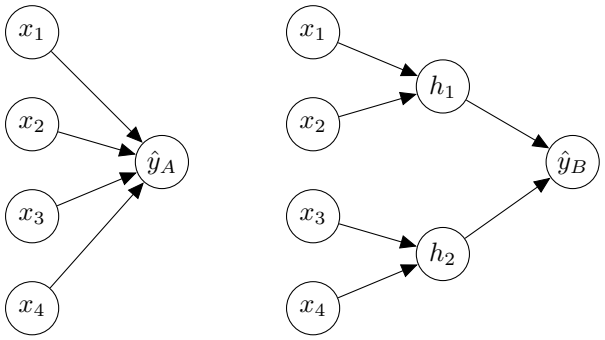
Consider the following non-linearity for use in a neural network: $f_{0/1}(z) = 1$ if $z \geq 0$ and $f_{0/1}(z) = 0$ otherwise. Let $\mathbf{x}$ be a binary feature vector of length 4: $\mathbf{x} \in \{0,1\}^4$. Define neural network A as follows:

$$\hat{y_A} \leftarrow f_{0/1}(\mathbf{w}^\top \mathbf{x} + w_0)$$

with weight vector $\mathbf{w} \in \mathbb{R}^4$ and bias scalar $w_0 \in \mathbb{R}$. Let $\mathbf{x}^L = [x_1, x_2]$ and $\mathbf{x}^R = [x_3, x_4]$. Define neural network B as follows:

$$\begin{aligned} h_1 &\leftarrow f_{0/1}(\mathbf{t}^\top \mathbf{x}^L + a) \\ h_2 &\leftarrow f_{0/1}(\mathbf{u}^\top \mathbf{x}^R + b) \\ \mathbf{h} &\leftarrow [h_1, h_2] \\ \hat{y_B} &\leftarrow f_{0/1}(\mathbf{v}^\top \mathbf{h} + c) \end{aligned}$$

with weight vectors $\mathbf{t}, \mathbf{u}, \mathbf{v} \in \mathbb{R}^2$ and bias scalars $a, b, c \in \mathbb{R}$. Basically, B can only look at the two halves of the input separately and has an extra layer to merge the transformations on the two halves of the input with another transformation:



a. i. Describe a logical formula on inputs that can be expressed by A but not by B and provide weights for $\mathbf{w}$ and w_0 that implement the formula in A (hint: think about things you may want to do with binary vectors, e.g. `AND`s, `OR`s)

ii. Provide an argument for why B cannot express this formula (not a rigorous proof, just a complete and convincing argument).

iii. How might you change the architecture of B to fix this issue? What downside might this have?

b. What is the concern about training the networks as currently defined? What changes can alleviate this concern?

c. State **two** ways in which a *validation set* can be used when training neural networks (one sentence for each).

a. i. One that works is to detect if three or more dimensions are 1:

$$\sum_{j=1}^{4} x_j \geq 3$$

Easy to see that this is solvable with network A:

$$\sum_{j=1}^{4} w_j x_j - 3 \geq 0$$

with $w_j = 1$ for all j.

ii. This can't work for network B though. Argument is that there are 5 cases with at least three 1's:

$$1111, 0111, , 1011, 1101, 1110$$

However, to detect any of the latter four patterns, either side of B's middle layer, h_1 and h_2, needs to be able to pick up the pattern 01 or 10. But that means that the both h_1 and h_2 will fire 1's for:

$$0101, 1010, 0110, 1001$$

iii. The easiest answer is to add more connections. Alternative answer is to add more basis functions. Either way, the downside is that you are adding more parameters to train to the model.

b. As defined, the networks use the 0/1 activation for its basis functions. Similarly to using 0/1 in final layers, this means that the gradients cannot pass back to the weight parameters and they cannot be learned. The easiest way to alleviate this is to switch to the sigmoid σ activation, which is a smooth approximation of 0/1.

c. i. Validation can be used to set the regularization parameters of the network.

ii. Validation can be used to structure the architecture of the network, by helping to select size and connection properties of layers.