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) What is the prediction for some new observation $x$?

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

   Note: The univariate Gaussian distribution is $\mathcal{N}(x; \mu, \sigma^2)$:

$$\frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{-(x-\mu)^2}{2\sigma^2})$$

   (d) Now assume that we have a known prior $s_0^2$ on $w \sim \mathcal{N}(0, s_0^2)$.

   Write down the form of the *posterior* in terms of the data and univariate Gaussian distribution. Take logs and drop terms that don't depend the data, but you do not need simplify further.

**Answers**

(a) This question is mostly just math. Take the derivative with respoect to $w$ and set to 0.

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

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

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

(b) Here we just plug in from above.

$$y = x\left(\frac{\sum_{i=1}^{n} y_i x_i}{\sum_{i=1}^{n} x_i^2}\right)$$

(c) This is a definitions question. Use the form of the univariate Gaussian.

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

(d) Here we combine everything above.

Prior:

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

Likelihood:

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

Posterior:

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

Take logs:

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

Worth noting similarity to ridge regression

2. **Multiclass Classification**

Suppose that we have a $c$-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.

We model this problem using a neural network with $d$ units in a single hidden layer, expressed as column vector $\boldsymbol{\phi}(\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1) \in \mathbb{R}^d$, which we write as $\boldsymbol{\phi}$. We take a linear combination of these values and pass them to a softmax function to get a final set of $c$ outputs:

$$p(\mathbf{y} = C_k \mid \mathbf{x}; \{\mathbf{w}_\ell\}, \mathbf{W}^1, \mathbf{w}_0^1) = \frac{\exp(\mathbf{w}_k^\mathsf{T} \boldsymbol{\phi})}{\sum_{\ell=1}^c \exp(\mathbf{w}_\ell^\mathsf{T} \boldsymbol{\phi})}$$

where $\mathbf{w} \in \mathbb{R}^d$ is a column vector of weights.

(a) Suppose we add the same, global bias to each vector of weights in the final layer, ie replace $\mathbf{w}_k^\mathsf{T} \boldsymbol{\phi}$ with $\mathbf{w}_k^\mathsf{T} \boldsymbol{\phi} + w_0$ for some scalar $w_0$. Does that increase the expressivity of our model? Why or why not?

(b) Write down and simplify the log likelihood of a particular observation $(\mathbf{x}_i, \mathbf{y}_i)$, including constants. Assume that we use a sigmoid activation function,

$$\boldsymbol{\phi}(\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1) = \boldsymbol{\sigma}(\mathbf{W}^1 \mathbf{x} + \mathbf{w}_0^1)$$

(c) In information theory, cross-entropy $\mathbb{E}_p[-\ln(q)]$ is the number of bits required to explain a true distribution $p$ (our desired outputs) if given some other distribution $q$. Write down the expression for the cross-entropy of true multi-class prediction (samples from our data) compared to our predictions. Describe its relationship to the log loss in part (b).

(d) When running SGD we need to compute the gradient of the loss for a single example with respect to each of the parameters. Use the above definitions of $\boldsymbol{\phi}$ to compute this gradient for the bias vector for the neural network layer.

$$\frac{\partial}{\partial \mathbf{w}_0^1} - \ln p(\mathbf{y} = C_k | \mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\})$$

Note:

- You may use $\sigma'$ for the derivative of the sigmoid function.
- You may use this intermediate term:

$$\frac{\partial}{\partial \mathbf{w}_k^\top \boldsymbol{\phi}} - \ln p(\mathbf{y} = C_k | \mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\}) = p(\mathbf{y} = C_k | \mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\}) - y_k$$

(a) With normalization becomes:

$$\begin{aligned} p(\mathbf{y} = C_k \mid \mathbf{x}; \{\mathbf{w}_\ell\}, \mathbf{W}^1, \mathbf{w}_0^1) &= \frac{1}{\text{const}} \exp(\mathbf{w}_k^\top \boldsymbol{\phi} + w_0) \\ &= \frac{1}{\text{const}} \exp(\mathbf{w}_k^\top \boldsymbol{\phi}) \exp(w_0) \\ &= \frac{1}{\text{const'}} \exp(\mathbf{w}_k^\top \boldsymbol{\phi}) \end{aligned}$$

So this is effectively the same as the original formula.

(b) Assume that $y_i$ is in class $k$,

$$\begin{aligned} \ln p(\mathbf{y} = C_k \mid \mathbf{x}; \{\mathbf{w}_\ell\}, \mathbf{W}^1, \mathbf{w}_0^1) &= \ln \frac{\exp(\mathbf{w}_k^\intercal \boldsymbol{\phi})}{\sum_{\ell=1}^{c} \exp(\mathbf{w}_\ell^\intercal \boldsymbol{\phi})} \\ &= \mathbf{w}_k^\intercal \boldsymbol{\phi} - \ln \sum_{\ell=1}^{c} \exp(\mathbf{w}_\ell^\intercal \boldsymbol{\phi}) \\ &= \mathbf{w}_k^\intercal \boldsymbol{\sigma}(\mathbf{W}^1 \mathbf{x}_i + \mathbf{w}_0^1) - \ln \sum_{\ell=1}^{c} \exp(\mathbf{w}_\ell^\intercal \boldsymbol{\sigma}(\mathbf{W}^1 \mathbf{x}_i + \mathbf{w}_0^1)) \end{aligned}$$

(c) This is a comprehension question. Here $p(\mathbf{y}|\mathbf{x})$ is our true data, and $q(\mathbf{y}|\mathbf{x})$ is our model i.e.:

$$p(\mathbf{y} = C_k \mid \mathbf{x}; \{\mathbf{w}_\ell\}, \mathbf{W}^1, \mathbf{w}_0^1)$$

The expectation here is computed empirically as the sum over the data, and the inner term is just applying our model.

$$\mathbb{E}_{\mathbf{y} \sim p(\mathbf{y}|\mathbf{x})}(-\ln q(\mathbf{y}|\mathbf{x})) = \sum_{i=1}^{n} -\ln q(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{n} -\ln p(\mathbf{y} = C_k \mid \mathbf{x}; \{\mathbf{w}_\ell\}, \mathbf{W}^1, \mathbf{w}_0^1)$$

You should recognize this last term as the loss.

(d) The key step here is to apply the chain rule.

$$\frac{\partial p(\mathbf{y} = C_k | \mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\})}{\partial \boldsymbol{\phi}} = \sum_k \frac{\partial \mathbf{w}_k^\top \boldsymbol{\phi}}{\partial \boldsymbol{\phi}} \frac{\partial p(\mathbf{y} = C_k | \mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\})}{\partial \mathbf{w}_k^\top \boldsymbol{\phi}}$$

$$\frac{\partial p(\mathbf{y} = C_k|\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\})}{\partial \boldsymbol{\phi}} = \sum_k \mathbf{w}_k (p(\mathbf{y} = C_k|\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\}) - y_k)$$

Then apply it once more (trivially.)

$$\begin{aligned}\frac{\partial p(\mathbf{y} = C_k|\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\})}{\partial w_0^1} &= \frac{\partial \boldsymbol{\phi}}{\partial w_0^1} \times \sum_k \mathbf{w}_k (p(\mathbf{y} = C_k|\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\}) - y_k) \\ &= \sum_k \mathbf{w}_k (p(\mathbf{y} = C_k|\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1, \{\mathbf{w}_\ell\}) - y_k)\end{aligned}$$

3. **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 | Test 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 inference is not hindering performance.

(a) If you had to choose one: might Classifer A be overfitting or underfitting? Explain your reasoning.

(b) If you had to choose one: might Classifer 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.

(a) Likely it is underfitting, demonstrated by the results of Classifier B. This is a 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.

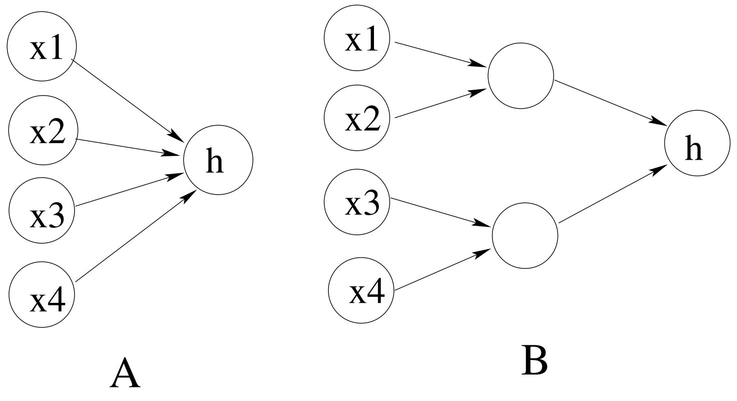
(c) It seems unlikely that more training will help 1. There is no indication of a variance issue (train/test accuracy similar.) However for classifier C, more training data would reduce the variance of the rich model.

4. **Neural Networks.**

(a) Consider neural networks with a basis functions that uses a **0/1 activation** $f_{0/1}$, that switches at 0 from 0 to 1, with:

$$f_{0/1}(h) = \begin{cases} 1 & \text{if } h \geq 0 \\ 0 & \text{otherwise,} \end{cases}$$

for weighted sum of input for each $\phi_j(\mathbf{x}) = f_{0/1}(\mathbf{w}_j^{1\top}\mathbf{x} + w_0^1)$ . The input layer to the networks $x_1, x_2, x_3$ and $x_4$, **all take on values 0 or 1.** i.e. $\mathbf{x} \in \{0,1\}^4$. Consider two networks, one which is linear in $\mathbf{x}$ and one that sparse connections in its first layer.



(i) Describe a logical formula on inputs that can be expressed by architecture (A) but not by (B), and provide weights that implement the formula in A.

(ii) Provide an argument for why B cannot express this formula (we don't expect a rigorous proof, but try to give a complete and convincing argument.)

(iii) How might you change the architecture of the second network to fix this issue? What downside might this have?

(b) What is the concern about training the networks as currently defined? What change would you make to the network to alleviate this concern?

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

(a) 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 $j$.

This can't work for network B though. Argument is that there are 4 cases:

$$1111, 0111, 1101, 0101$$

Since the first three must be on, so the middle layer must detect either 01. However, this means it will also fire for 0101.

(b) 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 the model, leading to the possibility of overfitting.

(c) As defined the network uses 0/1 activation for its basis functions. Similarly to using 0/1 for the final layers, this means that gradients cannot pass back to the $\mathbf{W}^1$ parameters and they cannot be learned (draw picture). The easiest way to alleviate this is to switch to the sigmoid $\sigma$ activation which is smooth approximation of 0/1.

(d)
   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.