Homework 1 - Version 1.4

Deadline: Monday, Jan.27, at 11:59pm.

Submission: You must submit your solutions as a PDF file through MarkUs[1]. You can produce the file however you like (e.g. LaTeX, Microsoft Word, scanner), as long as it is readable.

See the syllabus on the course website[2] for detailed policies. You may ask questions about the assignment on Piazza[3]. *Note that 10% of the homework mark (worth 1 pt) may be removed for a lack of neatness.*

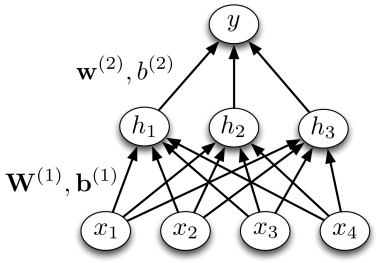
The teaching assistants for this assignment are Denny Wu and Jonathan Lorraine.
`mailto:csc413-2020-01-tas@cs.toronto.edu`

1 Hard-Coding Networks

The reading on multilayer perceptrons located at `https://csc413-2020.github.io/assets/readings/L02a.pdf` may be useful for this question.

1.1 Verify Sort [1pt]

In this problem, you need to find a set of weights and biases for a multilayer perceptron which determines if a list of length 4 is in sorted order. More specifically, you receive four inputs $x_1, \ldots, x_4$, where $x_i \in \mathbb{R}$, and the network must output 1 if $x_1 \leq x_2 \leq x_3 \leq x_4$, and 0 otherwise. You will use the following architecture:



All of the hidden units and the output unit use a hard threshold activation function:

$$\phi(z) = \mathbb{I}(z \geq 0) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Please give a set of weights and biases for the network which correctly implements this function (including cases where some of the inputs are equal). Your answer should include:

- A 3×4 weight matrix $\mathbf{W}^{(1)}$ for the hidden layer
- A 3-dimensional vector of biases $\mathbf{b}^{(1)}$ for the hidden layer
- A 3-dimensional weight vector $\mathbf{w}^{(2)}$ for the output layer

[1] `https://markus.teach.cs.toronto.edu/csc413-2020-01`
[2] `https://csc413-2020.github.io/assets/misc/syllabus.pdf`
[3] `https://piazza.com/class/k58ktbdnt0h1wx?cid=1`

- A scalar bias $b^{(2)}$ for the output layer

You do not need to show your work.

Answer

$$\mathbf{W}^{(1)} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}, \mathbf{b}^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{W}^{(2)} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \text{ and } \mathbf{b}^{(2)} = \tfrac{1}{2}.$$

1.2 Perform Sort [1pt]

Describe how to implement a sorting function $\hat{f} : \mathbb{R}^4 \to \mathbb{R}^4$ where $\hat{f}(x_1, x_2, x_3, x_4) = (\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4)$ where $(\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4)$ is (x_1, x_2, x_3, x_4) in sorted order. In other words, $\hat{x}_1 \leq \hat{x}_2 \leq \hat{x}_3 \leq \hat{x}_4$, and each $\hat{x}_i$ is a distinct x_j. Implement $\hat{f}$ using a feedforward or recurrent neural network with elementwise activations. Do not explicitly give the weight matrices or biases for the entire function. Instead, describe how to compose smaller, modular networks. You may combine information across nodes via summation as in 1.1, or with multiplication.

Hint: There are multiple solutions. You could brute-force the answer by using copies of *Verify Sort* on permutations of the input, or you could implement a more scalable sorting algorithm where each hidden layer i is the algorithms state at step i.

Answer

We implement $\texttt{permute}(p_i, x_1, x_2, x_3, x_4)$ where p_i is some permutation like $[i, j, k, l]$, by creating a matrix where the 1^{st} row has a 1 in the i^{th} column and 0 otherwise, the 2^{nd} row has a 1 in the j^{th} column and 0 otherwise, the 3^{rd} row has a 1 in the k^{th} column and 0 otherwise, and the 4^{th} row has a 1 in the l^{th} column and 0 otherwise.

Our modules are defined as follows:

$$\texttt{pass-if-sorted}(p_i, x_1, x_2, x_3, x_4) = \texttt{verify-sort}(\texttt{permute}(p_i, x_1, x_2, x_3, x_4)) \cdot \texttt{permute}(p_i, x_1, x_2, x_3, x_4) \tag{1}$$

We also store how many permutations are sorted, incase we have $x_i = x_j$ for $i \neq j$:

$$\texttt{how-many-sorted}(x_1, x_2, x_3, x_4) = \sum_i \texttt{verify-sort}(\texttt{permute}(p_i, x_1, x_2, x_3, x_4)) \tag{2}$$

Our final answer is the sum of the sorted permutations divided by the number of sorted permutations:

$$\texttt{sort}(x_1, x_2, x_3, x_4) = \frac{\sum_i \texttt{pass-if-sorted}(p_i, x_1, x_2, x_3, x_4)}{\texttt{how-many-sorted}(x_1, x_2, x_3, x_4)} \tag{3}$$

This is all shown together is Figure 1.2.

1.3 Universal Approximation Theorem [1pt]

We are going to build an intuition behind a simple Universal Approximation theorem, which shows that some class of function approximators can approximate a particular class of functions arbitrarily well.

In the reading we saw that neural networks can be universal approximators on binary functions, because with fixed input dimension there is a finite number of potential inputs and a network can

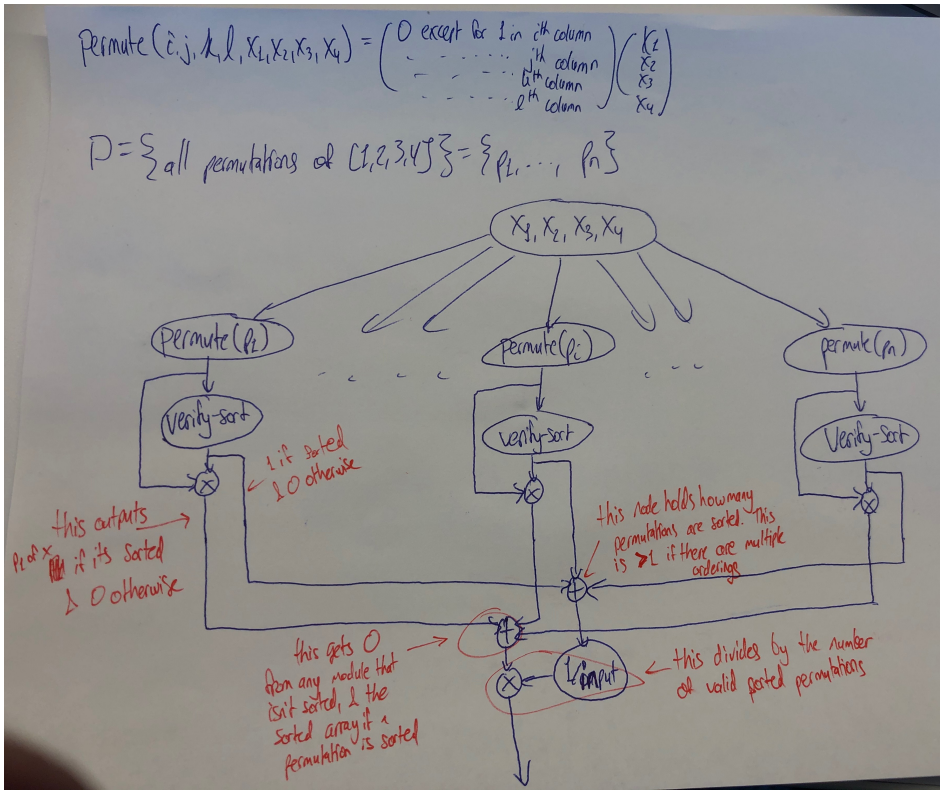


Figure 1: A visualization of the answer for Section 1.2

memorize a different output for each input. But, what can we do if we have an uncountably infinite set of potential inputs like $\mathbb{R}$? Here, our class of function approximators will be neural networks with a single hidden layer with a threshold function as the activation, and fixed choice of some concave f.

Suppose $f : I \to \mathbb{R}$, where $I = [a, b] \subset \mathbb{R}$ and $a \leq b$ is a closed interval. Also, let $\hat{f}_\tau : I \to \mathbb{R}$ be some function approximator from our network where τ is a description of our networks architecture and weights. Here, τ is a tuple of $(n, \mathbf{W}_0 \in \mathbb{R}^{n\times 1}, \mathbf{b}_0 \in \mathbb{R}^n, \mathbf{W}^1 \in \mathbb{R}^{1\times n}, \mathbf{b}^1 \in \mathbb{R})$, where n is the hidden layer size, $\mathbf{W}_0$ & $\mathbf{b}_0$ describe the input to hidden parameters, and $\mathbf{W}_1$ & $\mathbf{b}_1$ describe the hidden to output parameters. This is visualized in Figure 2.

The difference between our functions is defined as $\|f - \hat{f}_\tau\| = \int_I |f(x) - \hat{f}_\tau(x)|\, dx$. Our activation is an indicator function $a(y) = \mathbb{I}(y \geq 0)$, where $\mathbb{I}(s)$ is 1 when the boolean value s is true and 0 otherwise. The output is computed as $\hat{f}_\tau(x) = \mathbf{W}_1 a(\mathbf{W}_0 x + \mathbf{b}_0) + \mathbf{b}_1$. Here, applying a to a vector means $a(\mathbf{x}) = [a(\mathbf{x}_1), a(\mathbf{x}_2), \ldots, a(\mathbf{x}_n)]$.

We want to show that there exist a series of neural networks $\{\tau_i\}_{i=1}^N$ such that:

$$\forall \epsilon > 0, \exists M : \forall m > M, \|f - \hat{f}_{\tau_m}\| < \epsilon \tag{4}$$

1.3.1

Consider a bump function $g(h, a, b, x) = h \cdot \mathbb{I}(a \leq x \leq b)$ visualized in Figure 3. Given some (h, a, b) show $\exists \tau : \hat{f}_\tau(x) = g(h, a, b, x)$. You answer should be a specific choice of n, $\mathbf{W}_0$, $\mathbf{b}_0$, $\mathbf{W}_1$, and $\mathbf{b}_1$, which will be functions of the selected (h, a, b), where $h \in \mathbb{R}$, $a \in \mathbb{R}$, and $b \in \mathbb{R}$.

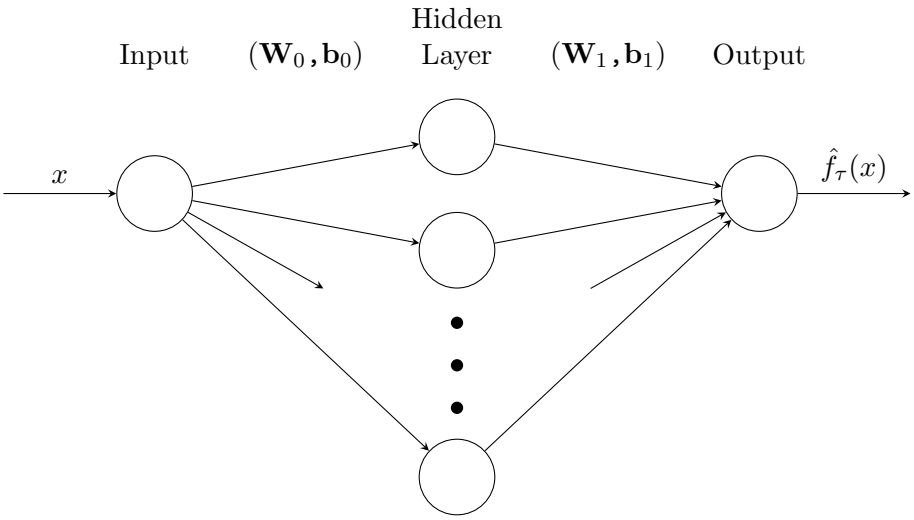


Figure 2: A neural network that has an input $x \in \mathbb{R}$ with a single hidden layer that has n units, and an output $\hat{f}_\tau(x) \in \mathbb{R}$. The network description is $\tau = (n, \mathbf{W}_0 \in \mathbb{R}^{n \times 1}, \mathbf{b}_0 \in \mathbb{R}^n, \mathbf{W}^1 \in \mathbb{R}^{1 \times n}, \mathbf{b}^1 \in \mathbb{R})$.

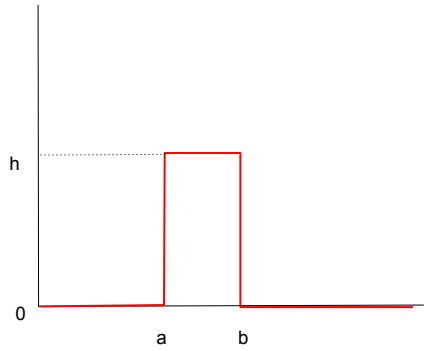


Figure 3: A bump function $g(h, a, b, x)$ is shown in red as a function of x, for some choice of (h, a, b).

Answer

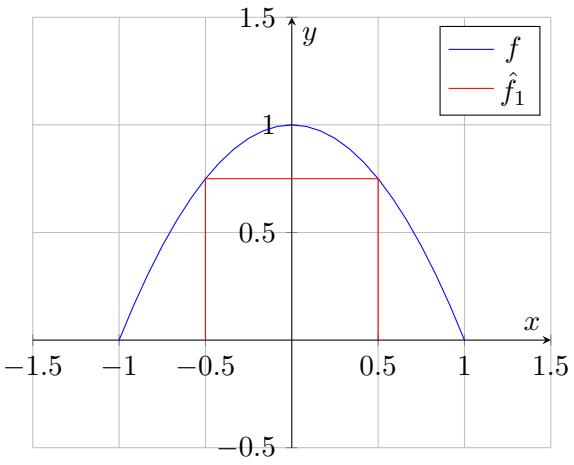
$n = 2$, $\mathbf{W}^{(0)} = \begin{bmatrix} 1 & -1 \end{bmatrix}$, $\mathbf{b}^{(0)} = \begin{bmatrix} -a \\ b \end{bmatrix}$, $\mathbf{W}^{(1)} = \begin{bmatrix} h \\ h \end{bmatrix}$ and $\mathbf{b}^{(1)} = -h$.

1.3.2

Given $f(x) = -x^2 + 1$ where $I = [-1, 1]$ and some initial function $\hat{f}_0(x) = 0$ which is identically 0, construct a new function $\hat{f}_1(x) = \hat{f}_0(x) + g(h_1, a_1, b_1, x)$ such that $\|f - \hat{f}_1\| < \|f - \hat{f}_0\|$, with the g defined in 1.3.1. Note that h_1, a_1, and b_1 are going to depend on our choice of f, $\hat{f}_0$ and I. Plot f & $\hat{f}_1$, write down h_1, a_1, and b_1, and justify why $\|f - \hat{f}_1\| < \|f - \hat{f}_0\|$.

Answer

We select a potential answer of $h_1 = -(\frac{1}{2})^2 + 1 = \frac{3}{4}$, $a_1 = -\frac{1}{2}$, and $b_1 = \frac{1}{2}$. Note how $\|f - \hat{f}_0\| - \|f - \hat{f}_1\| > 0$, because the difference is exactly the area of the red square in the graph which is equal to $\frac{3}{4}$.



1.3.3

Describe a procedure which starts with $\hat{f}_0(x) = 0$ and a fixed N then construct a series $\{\hat{f}_i\}_{i=0}^N$ where $\hat{f}_{i+1}(x) = \hat{f}_i(x) + g(h_{i+1}, a_{i+1}, b_{i+1}, x)$ which satisfies $\|f - \hat{f}_{i+1}\| < \|f - \hat{f}_i\|$. Use the definition of g from 1.3.1 and the choice of f from 1.3.2. Plot $f, \hat{f}_1, \hat{f}_2$, & $\hat{f}_3$, write down how to generate $h_{i+1}, a_{i+1}, b_{i+1}$, and justify why $\|f - \hat{f}_{i+1}\| < \|f - \hat{f}_i\|$.

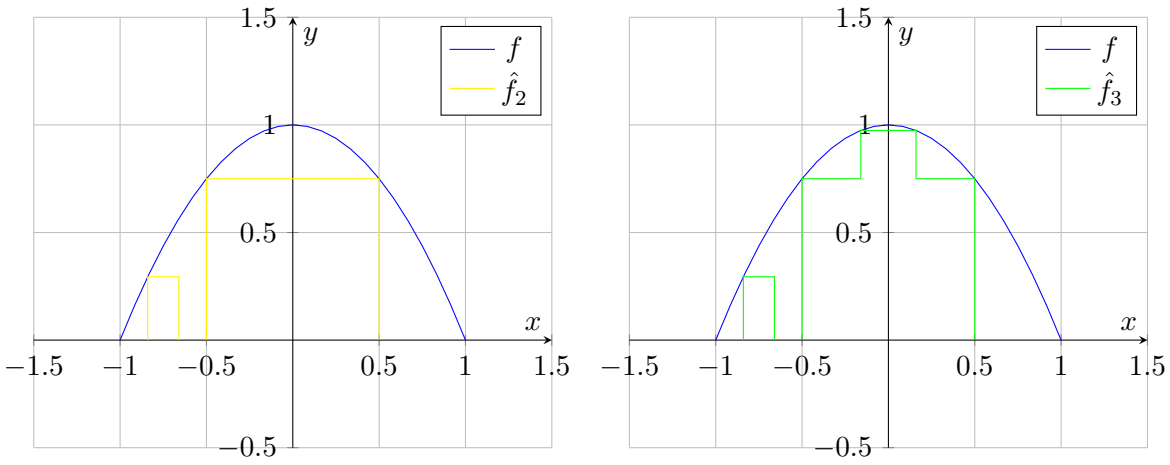
Answer

Create an queue of intervals A and initialize with 1 entry $[-1, 1]$. At each iteration we will select the first interval from our queue, approximate a section of the interval, then add sub-intervals to our array A. Given some interval $I \in A$, where $I = [z_1, z_2]$ we will select $a_{i+1} = z_1 + \frac{1}{4} * (z_2 - z_1)$, and $b_{i+1} = z_1 + \frac{3}{4} * (z_2 - z_1)$, then we add intervals $I_1 = [z_1, z_1 + \frac{1}{4} * (z_2 - z_1)], I_2 = [z_1 + \frac{1}{4} * (z_2 - z_1), z_1 + \frac{3}{4} * (z_2 - z_1)], I_3 = [z_1 + \frac{3}{4} * (z_2 - z_1), z_2]$ back into A. This is kind of like a Cantor set construction.

We select $h_i = \min(f(a_{i+1}) - \hat{f}_i(a_{i+1}), f(b_{i+1}) - \hat{f}_i(b_{i+1})$. This selection causes $\hat{f}_{i+1} \leq f$ over (a_{i+1}, b_{i+1}) Also, this selection $\hat{f}_{i+1} \geq \hat{f}_i$ over (a_{i+1}, b_{i+1}). Now we just need to show that either one of the inequalities is strict - i.e., $\hat{f}_{i+1} < f$ or $\hat{f}_{i+1} > \hat{f}_i$.

Well, at every step we match one of the endpoints of our interval, and our f is strictly concave. Thus, we are not exactly matching the f at the interior of our interval. Since our next selected points a_{i+1}, b_{i+1} are always on the interior and $a_{i+1} \neq b_{i+1}$, we always make an improvement of the minimum difference of our selected points.

Thus, $\|f - \hat{f}_i\| - \|f - \hat{f}_{i+1}\| > 0$.



1.3.4

Not for marks - do not submit. Describe how to recover τ_i from your $\hat{f}_i$. Generalize your series to work for $\{\hat{f}_i\}_{i=0}^{\infty}$, and show that $\lim_{m\to\infty} \|f - \hat{f}_{\tau_m}\| = c$. We need $c = 0$ for arbitrarily good approximations - does your solution for $\{\hat{f}_i\}_{i=0}^{\infty}$ force $c = 0$? Can you generalize your procedure to work for any concave f? What about an f which isn't unimodal, like a piecewise continuous f?

1.3.5 Answer

You could implement Riemann sums of order N then use the proof of the Riemann sum to generalize this to Riemann integrable functions. Implementing Riemann sums would also be easy to show satisfy properties for preceding parts, if you're careful about their heights. This construction wouldn't satisfy $\lim_{i\to\infty} \|\frac{\partial}{\partial x} f - \frac{\partial}{\partial x}\hat{f}_i\|$, which could be desirable for a stronger notion of approximation. How would you generalize our bump function and construction to converge here?

2 Backprop

The reading on backpropagation located at `https://csc413-2020.github.io/assets/readings/L02b.pdf` may be useful for this question.

2.1 Computational Graph [1pt]

Consider a neural network with N input units, N output units, and K hidden units. The activations are computed as follows:

$$\begin{aligned}
\mathbf{z} &= \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{h} &= \text{ReLU}(\mathbf{z}) \\
\mathbf{y} &= \mathbf{x} + \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}, \\
\mathbf{y}' &= \text{softmax}(\mathbf{y}),
\end{aligned}$$

where $\text{ReLU}(\mathbf{z}) = \max(\mathbf{z}, 0)$ denotes the ReLU activation function, applied elementwise, and $\text{softmax}(\mathbf{y}) = \frac{\exp(\mathbf{y})}{\sum_{i=1}^{M} \exp(\mathbf{y}_i)}$.

The cost will involve both $\mathbf{h}$ and $\mathbf{y}'$:

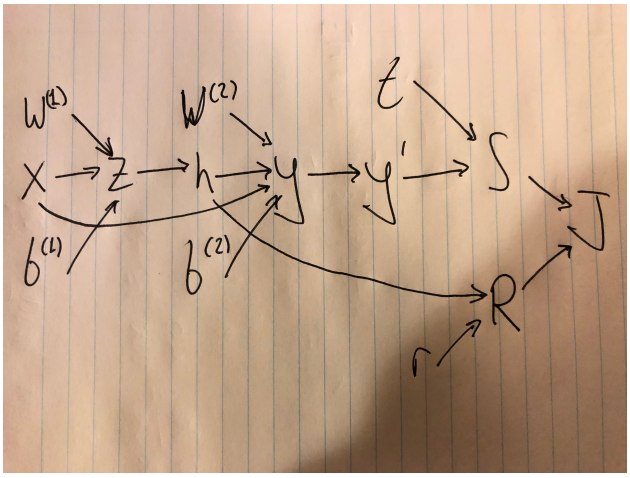
$$\mathcal{J} = \mathcal{R} - \mathcal{S}$$
$$\mathcal{R} = \mathbf{r}^\top \mathbf{h}$$
$$\mathcal{S} = \sum_{k=1}^{N} \mathbb{I}(t = k)\mathbf{y}'_k$$

for given vectors $\mathbf{r}$ and class label t with input x.

2.1.1

Draw the computation graph relating $\mathbf{x}$, t, $\mathbf{z}$, $\mathbf{h}$, $\mathbf{y}$, $\mathbf{y}'$, $\mathbf{r}$, $\mathcal{R}$, $\mathcal{S}$, and $\mathcal{J}$.

Answer



2.1.2

Derive the backprop equations for computing $\bar{\mathbf{x}} = \frac{\partial \mathcal{J}}{\partial \mathbf{x}}$. You may use softmax$'$ to denote the derivative of the softmax function (so you don't need to write it out explicitly).

Answer

$$
\begin{aligned}
\overline{\mathcal{J}} &= 1 \\
\overline{\mathcal{S}} &= \overline{\mathcal{J}} \frac{\partial \mathcal{J}}{\partial \mathcal{S}} \\
&= -\overline{\mathcal{J}} \mathbf{JL} \text{ :Won't grade on sign here} \\
\overline{\mathcal{R}} &= \overline{\mathcal{J}} \\
\overline{\mathbf{y}'} &= \overline{\mathcal{S}} \frac{\partial \mathcal{S}}{\partial \mathbf{y}'} \\
&= \overline{\mathcal{S}} \text{ one-hot}(t) \\
&= \overline{\mathcal{S}}[0, \ldots, 0, 1, 0, \ldots, 0]^T \\
&\text{where the 1 is at the } t^{th} \text{ index.} \\
\overline{\mathbf{y}} &= \overline{\mathbf{y}'} \frac{\partial \mathbf{y}'}{\partial \mathbf{y}} \\
&= \overline{\mathbf{y}'} \text{softplus}'(\mathbf{y}) \\
\overline{\mathbf{h}} &= \overline{\mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{h}} + \overline{\mathcal{R}} \frac{\partial \mathcal{R}}{\partial \mathbf{h}} \\
&= [\mathbf{W}^{(2)}]^\top \overline{\mathbf{y}} + \mathbf{r} \\
\overline{\mathbf{z}} &= \overline{\mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \\
&= \overline{\mathbf{h}} \circ \text{ReLU}'(\mathbf{z}) \\
&= \overline{\mathbf{h}_i} \circ \begin{cases} 0 & \mathbf{z}_i < 0 \\ 1 & \mathbf{z}_i > 0 \end{cases} \text{for each } i \in \{1, \ldots, K\} \\
\overline{\mathbf{x}} &= \overline{\mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{x}} + \overline{\mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \\
&= [\mathbf{W}^{(1)}]^\top \overline{\mathbf{z}} + \overline{\mathbf{y}}
\end{aligned}
$$

2.2 Vector-Jacobian Products (VJPs) [1pt]

Consider the function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ where $\mathbf{f}(\mathbf{x}) = \mathbf{v}\mathbf{v}^T\mathbf{x}$, and $\mathbf{v} \in \mathbb{R}^{n \times 1}$ and $\mathbf{x} \in \mathbb{R}^{n \times 1}$. Here, we will explore the relative costs of evaluating Jacobians and vector-Jacobian products. We denote the Jacobian of $\mathbf{f}$ with respect to $\mathbf{x}$ as $J \in \mathbb{R}^{n \times n}$.

2.2.1

Compute J as defined in 2.2 for $n = 3$ and $\mathbf{v}^T = [1, 2, 3]$ - i.e, write down the values in $J = \begin{pmatrix} j_{1,1} & j_{1,2} & \cdots & j_{1,n} \\ j_{2,1} & j_{2,2} & \cdots & j_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ j_{n,1} & j_{n,2} & \cdots & j_{n,n} \end{pmatrix}$.

Answer

$$J = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$$

2.2.2

What is the time and memory cost of evaluating the Jacobian of function $\mathbf{f}$ in terms of n?

Answer

The time and memory cost are $\mathcal{O}(n^2)$

2.2.3

Describe how to evaluate $J^T\mathbf{y}$ where $\mathbf{y} \in \mathbb{R}^n$ with a time and memory cost that is linear in n, where J is defined as in 2.2. Then, compute $\mathbf{z} = J^T\mathbf{y}$ where $\mathbf{v}^T = [1, 2, 3]$ and $\mathbf{y}^T = [1, 1, 1]$ - i.e., write down the entries in $\mathbf{z}^T = [z_1, \ldots, z_n]$.

Answer

Since $J = \mathbf{v}\mathbf{v}^T$, then $J\mathbf{y} = \mathbf{v}\mathbf{v}^T\mathbf{y}$. First, we compute $q = \mathbf{v}^T\mathbf{y}$ for a time and memory cost of $\mathcal{O}(n)$, then we compute $\mathbf{z} = \mathbf{v}q$ for a time and memory cost of $\mathcal{O}(n)$. This is a joint time and memory cost $\mathcal{O}(n + n) = \mathcal{O}(n)$

The answer is $q = 1 + 2 + 3 = 6$, and $\mathbf{z}^T = [6, 12, 18]$

3 Linear Regression

The reading on linear regression located at `https://csc413-2020.github.io/assets/readings/L01a.pdf` may be useful for this question.

Given n pairs of input data with d features and scalar label $(\mathbf{x}_i, t_i) \in \mathbb{R}^d \times \mathbb{R}$, we wish to find a linear model $f(\mathbf{x}) = \hat{\mathbf{w}}^\top\mathbf{x}$ with $\hat{\mathbf{w}} \in \mathbb{R}^d$ that minimizes the squared error of prediction on the training samples defined below. This is known as an empirical risk minimizer. For concise notation, denote the data matrix $X \in \mathbb{R}^{n\times d}$ and the corresponding label vector $\mathbf{t} \in \mathbb{R}^n$. The training objective is to minimize the following loss:

$$\min_{\hat{\mathbf{w}}} \frac{1}{n}\sum_{i=1}^{n}(\hat{\mathbf{w}}^\top\mathbf{x}_i - t_i)^2 = \min_{\hat{\mathbf{w}}} \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2.$$

We assume X is full rank: $X^\top X$ is invertible when $n > d$, and $XX^\top$ is invertible otherwise. Note that when $d > n$, the problem is *underdetermined*, i.e. there are less training samples than parameters to be learned. This is analogous to learning an *overparameterized* model, which is common when training of deep neural networks.

3.1 Deriving the Gradient [1pt]

Write down the gradient of the loss w.r.t. the learned parameter vector $\hat{\mathbf{w}}$.

Answer

$$\frac{dL}{d\hat{\mathbf{w}}} = \frac{2}{n} X^\top (X\hat{\mathbf{w}} - \mathbf{t}).$$

3.2 Underparameterized Model [1pt]

3.2.1

First consider the underparameterized $d < n$ case. Write down the solution obtained by gradient descent assuming training converges. Show your work. Is the solution unique?

Answer

$$\begin{aligned} &\frac{2}{n} X^\top (X\hat{\mathbf{w}} - \mathbf{t}) = 0; \\ \Leftrightarrow &X^\top X\hat{\mathbf{w}} = X^\top \mathbf{t}; \\ \Leftrightarrow &\hat{\mathbf{w}} = (X^\top X)^{-1} X^\top \mathbf{t}, \end{aligned}$$

where we used the invertibility of $X^\top X$ in the last step. This solution is unique.

3.2.2

Assume that ground truth labels are generated by a linear target: $t_i = \mathbf{w}^{*\top} \mathbf{x}_i$. Show that the solution in part 3.2.1 achieves perfect generalization when $d < n$, i.e. $\forall \mathbf{x} \in \mathbb{R}^d$, $(\mathbf{w}^{*\top}\mathbf{x} - \hat{\mathbf{w}}^\top \mathbf{x})^2 = 0$.

Answer

$$\hat{\mathbf{w}} = (X^\top X)^{-1} X\mathbf{t} = (X^\top X)^{-1} X^\top X\mathbf{w}^* = \mathbf{w}^*.$$

Thus gradient descent recovers the ground truth (target function).

3.3 Overparameterized Model: 2D Example [1pt]

3.3.1

Now consider the overparameterized $d > n$ case. We first illustrate that there exist multiple empirical risk minimizers. For simplicity we let $n = 1$ and $d = 2$. Choose $\mathbf{x}_1 = [2; 1]$ and $t_1 = 2$, i.e. the one data point and all possible $\hat{\mathbf{w}}$ lie on a 2D plane. Show that there exists infinitely many $\hat{\mathbf{w}}$ satisfying $\hat{\mathbf{w}}^\top \mathbf{x}_1 = y_1$ on a real line. Write down the equation of the line.

Answer

$$\hat{\mathbf{w}}^\top \mathbf{x}_1 = t_1 \;\Rightarrow\; \hat{w}_2 = -2\hat{w}_1 + 2.$$

Every $\hat{\mathbf{w}}$ on the line specified above achieves zero training error.

3.3.2

We know that multiple empirical risk minimizers exist in overparameterized linear regression and there is only one true solution. Thus, it seems unlikely that gradient descent will generalize if it returns an arbitrary minimizer. However, we will show that gradient descent tends to find certain solution with good properties. This phenomenon, know as *implicit regularization*, helps explain the success in using gradient-based methods to train overparameterized models, like deep neural networks.

First consider the 2-dimensional example in the previous part: starting from zero initialization i.e. $\hat{\mathbf{w}}(0) = 0$, what is the direction of the gradient? You should write down a unit-norm vector. Does the direction change along the trajectory? Based on this geometric intuition, which solution - along the line of solutions - does gradient descent find? Provide a pictorial sketch or a short description of your reasoning.

Answer

The direction of the gradient is always $\frac{1}{\sqrt{5}}[2; 1]$ independent of the position of $\hat{\mathbf{w}}$, which coincides with the direction of $\mathbf{x}_1$. To see this, simply note that the gradient is always in the form of $\mathbf{x}_1$ multiplied by a scalar $(\mathbf{x}_1\hat{\mathbf{w}} - t)$ which does not change the direction.

Therefore, starting from the origin, gradient descent finds the solution that is the intercept between $\hat{w}_2 = -2\hat{w}_1 + 2$ (line of solutions) and $\hat{w}_1 = 2\hat{w}_2$ (direction of gradient), which is $[\frac{4}{5}; \frac{2}{5}]$.

3.3.3

Give a geometric argument that among all the solutions on the line, the gradient descent solution from the previous part has the smallest Euclidean norm.

Answer

Note that the direction of the gradient is perpendicular to the line of solutions. Therefore, given the gradient descent solution $\hat{\mathbf{w}}$ and an arbitrary zero-loss solution on the line $\mathbf{w}_1$, by the Pythagorean Theorem, we know that $\|\mathbf{w}_1 - \hat{\mathbf{w}}\|_2^2 + \|\hat{\mathbf{w}}\|_2^2 = \|\mathbf{w}_1\|_2^2 \geq \|\hat{\mathbf{w}}\|_2^2$. This tells us that $\hat{\mathbf{w}}$ is the minimum Euclidean norm solution.

3.4 Overparameterized Model: General Case [1pt]

3.4.1

Now we generalize the previous geometric insight developed to general $d > n$. Show that gradient descent from zero initialization i.e. $\hat{\mathbf{w}}(0) = 0$ finds a unique minimizer if it converges. Write down the solution and show your work.

Answer

Since the gradient is always spanned by the rows of X, starting from zero initialization, every iterate and thus the final solution of gradient descent can also be written as a linear combination of rows of X. We can thus write $\hat{\mathbf{w}} = X^\top \mathbf{a}$ for some $\mathbf{a} \in \mathbb{R}^n$.

Plugging this into the stationary condition yields:

$$\begin{aligned} & X\hat{\mathbf{w}} - \mathbf{t} = XX^\top \mathbf{a} - \mathbf{t} = 0; \\ \Leftrightarrow & \mathbf{a} = (XX^\top)^{-1}\mathbf{t}; \\ \Leftrightarrow & \hat{\mathbf{w}} = X^\top (XX^\top)^{-1}\mathbf{t}, \end{aligned}$$

in which we utilized the invertibility of $XX^\top$ when $d > n$.

3.4.2

Given the gradient descent solution from the previous part $\hat{\mathbf{w}}$ and another zero-loss solution $\hat{\mathbf{w}}_1$, evaluate $(\hat{\mathbf{w}} - \hat{\mathbf{w}}_1)^\top \hat{\mathbf{w}}$. Use this quantity to show that among all the empirical risk minimizers for $d > n$, the gradient descent solution has the smallest Euclidean norm.

Answer

$$(\hat{\mathbf{w}} - \hat{\mathbf{w}}_1)^\top \hat{\mathbf{w}} = (\hat{\mathbf{w}} - \hat{\mathbf{w}}_1)^\top X^\top (XX^\top)^{-1}\mathbf{t}.$$

Note that since both $\hat{\mathbf{w}}$ and $\mathbf{w}_1$ are zero-loss solutions, we have $X\hat{\mathbf{w}} = \mathbf{t}$ and $X\mathbf{w}_1 = \mathbf{t}$. Therefore $X(\hat{\mathbf{w}} - \mathbf{w}_1) = 0$. This leads us to conclude that

$$(\hat{\mathbf{w}} - \hat{\mathbf{w}}_1)^\top \hat{\mathbf{w}} = 0.$$

Therefore

$$\|\mathbf{w}_1\|_2^2 = \|\mathbf{w}_1 - \hat{\mathbf{w}} + \hat{\mathbf{w}}\|_2^2 \overset{(i)}{=} \|\mathbf{w}_1 - \hat{\mathbf{w}}\|_2^2 + \|\hat{\mathbf{w}}\|_2^2 \geq \|\hat{\mathbf{w}}\|_2^2,$$

where (i) is due to the orthogonality between $\hat{\mathbf{w}}$ and $\mathbf{w}_1 - \hat{\mathbf{w}}$. We thus conclude that $\hat{\mathbf{w}}$ has the minimum Euclidean norm.

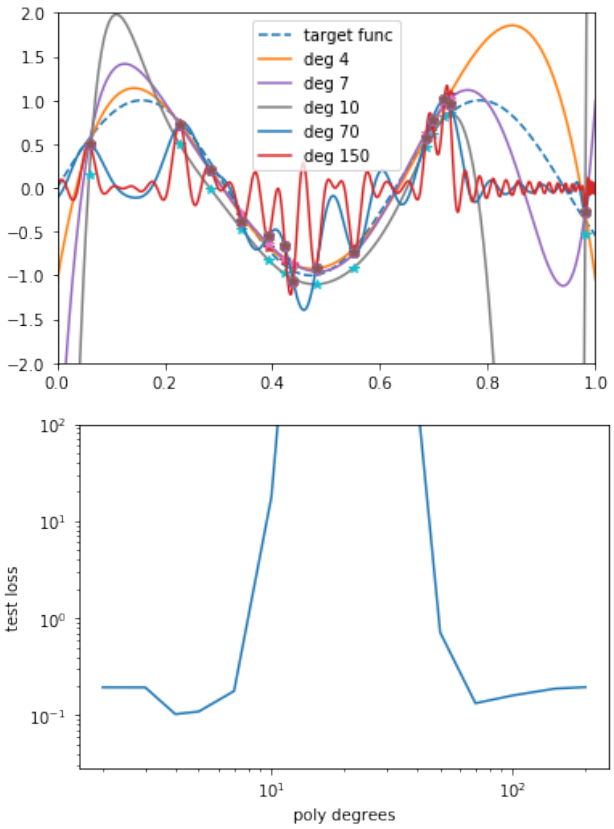
3.5 Benefit of Overparameterization

3.5.1

Visualize and compare underparameterized with overparameterized polynomial regression: `https://colab.research.google.com/drive/1NCSHp-gkh1kGhcySImbJo6AqeKoz-HcR`. Include your code snippets for the `fit_poly` function in the write-up. Does overparameterization (higher degree polynomial) always lead to overfitting, i.e. larger test error?

Answer

Overparameterization does not always lead to worse test error. Instead the test error peaks when degree of freedom of the model is roughly the same as the number of training samples. This surprising peak, known as *double descent* `https://openai.com/blog/deep-double-descent`, requires us to rethink the conventional view of bias-variance tradeoff in machine learning models.



3.5.2

Not for marks. What are some potential benefits of the minimum-norm solution?

Hint: readings on SVM might be helpful: `https://www.cs.toronto.edu/~urtasun/courses/CSC411_Fall16/15_svm.pdf`.

Answer

Potential explanations:

- the minimum-norm solution leads to the maximum Euclidean margin, and large margin models have good generalization properties (e.g. `https://arxiv.org/pdf/1009.3896.pdf`)
- the minimum-norm solution has smaller variance which is beneficial in the presence of label noise `https://arxiv.org/pdf/1903.08560.pdf`
- the norm of linear predictor relates to the complexity of the function `https://www.cs.cornell.edu/~sridharan/rad-paper.pdf`