TTIC 31020: Intro to ML **Winter 2024**

Homework 8

Due: 6 p.m., March 1st (Friday). No late submissions after 12 p.m., March 4th (Monday)

Note *You may discuss these problems in groups. However, you must write up your own solutions and mention the names of the people in your group. Also, please do mention any books, papers or other sources you refer to. We recommend that you typeset your solutions in LaTeX. Please submit your solutions as a PDF document on Canvas.*

Challenge and Optional Questions *Challenge questions are marked with* challenge *and Optional questions are marked with* optional . *You can get extra credit for solving* challenge *questions. You are not required to turn in* optional *questions, but we encourage you to solve them for your understanding. Course staff will help with* optional *questions but will prioritize queries on non-*optional *questions.*

---

1. **Back Propagation.** In this question, our goal is to consider neural networks which have either sigmoid or softmax activation units and calculate the gradients with respect to model parameters.

   Consider a neural network architecture, which is represented by a directed acyclic graph $G(V, E)$. Each node has an output value associated with it, which are defined recursively as follows.

   The nodes with no incoming edges are the input nodes $v_1, \ldots v_d$. Their values are given by the input $x \in \mathbb{R}^d$. In particular, we have $o[v_i] = x[i]$ for $i \in [d]$. For any other node $v \in V$, having computed output values of its parent nodes $\{o[u] : (u, v) \in E\}$, the output value of v is computed using either one of the two parametric functions below.

   - **Sigmoid activation.** We have parameters $w(u, v) \in \mathbb{R}$ for each edge (u, v)[1]. Then

   $$o[v] = \text{sigmoid}\left(\sum_{u:(u,v)\in E} w(u, v)o[u]\right), \text{ and } \ \text{sigmoid}(z) = \frac{1}{1 + e^{-z}},$$

     where sigmoid : $\mathbb{R} \to \mathbb{R}$ is a continuous differentiable function.
   - **Softmax activation.** Let $k := |\{u : (u, v) \in E\}|$, i.e. k is the number of parents of v. We have k^2 parameters $\{w(i, u, v) : i \in [k], (u, v) \in E\}$[2]. The output of v is then computed by

   $$o[v] = \text{softmax}\left(\sum_{u:(u,v)\in E} w(1, u, v)o[u], \ \ldots, \sum_{u:(u,v)\in E} w(k, u, v)o[u]\right).$$

     Here softmax : $\mathbb{R}^k \to \mathbb{R}$ is defined as follows.

   $$\text{softmax}(z_1, \ldots, z_k) := \frac{\sum_{i=1}^k z_i e^{z_i}}{\sum_{i=1}^k e^{z_i}}.$$

   Finally, there is an output node v_{out} with no outgoing edges whose output value is the final output $\hat{y} = o[v_{\text{out}}]$.

   (a) Calculate $\frac{\partial \hat{y}}{\partial w}$ for any parameter w in the system. In other words: for any node v, whose output is calculated using sigmoid activation, calculate $\frac{\partial \hat{y}}{\partial w(u,v)}$. Similarly, for any v whose value is calculated using softmax activation, calculate $\frac{\partial \hat{y}}{\partial w(i,u,v)}$. You may define stimuli of different nodes recursively and provide your answer in terms of them.

[1] You may think of it as a vector of parameters; one associated with each parent node u.
[2] You may think of it as a matrix of parameters in $\mathbb{R}^{k \times k}$.

(b) Now consider the special case of the layered graph with depth two. In particular,

$$\hat{y}(x) = \text{softmax}\left(W^{(2)}\text{sigmoid}\left(W^{(1)}x\right)\right), \text{ where } W^{(1)} \in \mathbb{R}^{k\times d}, W^{(2)} \in \mathbb{R}^{k\times k}.$$

Here sigmoid is applied to a vector entrywise. Moreover, consider the squared loss $\ell^{\text{sq}}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$. For an example $(x, y) \in (\mathbb{R}^d \times \mathbb{R})$, compute

$$\nabla_{W^{(2)}}\ell^{\text{sq}}(\hat{y}(x), y) \text{ and } \nabla_{W^{(1)}}\ell^{\text{sq}}(\hat{y}(x), y).$$

Simplify your answer so that the final expression is only in terms matrix/vector multiplications, without using summation notation (over indices).

(c) challenge We now consider another vector valued activation function softargmax : $\mathbb{R}^k \to \mathbb{R}^k$.[3] For any $z \in \mathbb{R}^k$ and $i \in [k]$,

$$(\text{softargmax}(z))_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}.$$

Define

$$\hat{y}(x) = w^\top\text{softargmax}\left(W^{(2)}\text{sigmoid}\left(W^{(1)}x\right)\right),$$

where $W^{(1)} \in \mathbb{R}^{k\times d}, W^{(2)} \in \mathbb{R}^{k\times k}$, and $w \in \mathbb{R}^k$. Now the logistic loss for $\hat{y} \in \mathbb{R}$ and $y \in \{\pm 1\}$ is given by $\ell^{\text{logistic}}(\hat{y}, y) = \log(1 + e^{-y\hat{y}})$. For any $(x, y) \in \mathbb{R}^d \times \{\pm 1\}$, compute

$$\nabla_w\ell^{\text{logistic}}(\hat{y}(x), y), \quad \nabla_{W^{(2)}}\ell^{\text{logistic}}(\hat{y}(x), y) \text{ and } \nabla_{W^{(1)}}\ell^{\text{logistic}}(\hat{y}(x), y).$$

Express your final answers in terms of matrix-vector multiplications.

2. **Expressive Power of Neural Networks.** challenge Let $\mathcal{X} = \{0, 1\}^d$ and the hypothesis class of all parity functions in $\{0, 1\}^{\mathcal{X}}$. Formally,

$$\text{PARITIES}_d = \left\{h_I(x) = \bigoplus_{i\in I} x_i : I \subseteq [d]\right\}.$$

We additionally define a hypothesis class of two layer ReLU networks with at most $2d$ hidden units.

$$\mathcal{H}_{\text{ReLU}} := \left\{h_\theta(x) = \text{sign}\left(\sum_{i=1}^{2d} a_i\text{ReLU}(\langle w_i, x\rangle + b_i)\right) : \theta = (a_i \in \mathbb{R}, b_i \in \mathbb{R}, w_i \in \mathbb{R}^d)_{i\in[d]}\right\}.$$

(a) Show that $\text{PARITIES}_d \subseteq \mathcal{H}_{\text{ReLU}}$. In other words, for each parity function $h_I(x)$, there exists a setting of the weights of the ReLU network that realizes it.

(b) Recall from the lecture that VCdim of fully connected feed forward ReLU networks of a certain depth is $O(|E| \cdot \text{depth} \cdot \log |E|)$, where $|E|$ is the total number of edges in the architecture graph, or alternatively, the number of parameters. Relying on this fact, propose a neural network based learning rule (not necessarily with efficient runtime) that can learn PARITIES_d but using only poly(d) samples. What is the best sample complexity guarantee of your learning rule? Contrast this with $\text{VCdim}(\text{PARITIES}_d)$, which is the optimal sample complexity using any learning rule.

[3] You may often see this activation function being called just softmax.