CS 601.471/671 NLP: Self-supervised Models

Homework 3: Building Your Neural Network!

For homework deadline, check the calendar on the course website*

Name: _____

Collaborators, if any: _____

Sources used for your homework, if any: _____

This assignment is testing your ability to understand the fundamental properties of neural networks and their training.

Homework goals: After completing this homework, you should be comfortable with:

- building your neural networks, particularly in PyTorch,
- explaining and deriving Backpropagation,
- debugging your neural network in case it faces any failures.

How to hand in your written work: Via Gradescope as before.

1 Concepts, intuitions and big picture

1. Which of these are reasons for the recent wave of neural networks taking off? (check the options that apply.)
   ☐ We have access to a lot more computational power.
   ☐ Neural Networks are a brand new field.
   ☐ We have access to a lot more data.
   ☐ There has been significant improvements in benchmarks in speech recognition and NLP.
   Answer: *TBD*

2. What does a neuron compute?
   ☐ A neuron computes an activation function followed by a linear function ($z = Wx + b$)
   ☐ A neuron computes a linear function ($z = Wx + b$) followed by an activation function.
   ☐ A neuron computes a function g that scales the input x linearly ($Wx + b$).
   ☐ A neuron computes the mean of all features before applying the output to an activation function.
   Answer: *TBD*

3. What is cached ("memoized") in the implementation of forward propagation and backward propagation?
   ☐ Variables computed during forward propagation are cached and passed on to the corresponding backward propagation step to compute derivatives.
   ☐ Caching is used to keep track of the hyperparameters that we are searching over, to speed up computation.
   ☐ Caching is used to pass variables computed during backward propagation to the corresponding forward propagation step. It contains useful values for forward propagation to compute activations.
   Answer: *TBD*

4. Which of the following statements is true?
   ☐ The deeper layers of a neural network are typically computing more complex features of the input than the earlier layers.
   ☐ The earlier layers of a neural network are typically computing more complex features of the input than the deeper layers.
   Answer: *TBD*

*https://self-supervised.cs.jhu.edu/sp2023/

5. During forward propagation, in the forward function for a layer l you need to know what is the activation function in a layer (Sigmoid, tanh, ReLU, etc.). During Backpropagation, the corresponding backward function also needs to know the activation function for layer l, since the gradient depends on it.
☐ True
☐ False
Answer: *TBD*

2 Revisiting Jacobians

Recall that Jacobians are generalizations of multi-variate derivatives. A Jacobian can be between any two variables involved in a computational graph. The shape of a Jacobian is an important thing to note. A Jacobian can be a vector, a matrix, or a tensor of arbitrary ranks. For example, if $\mathbf{f}$ is a scalar and $\mathbf{w}$ is a $d \times 1$ vector, the Jacobian $\mathbf{J_w f}$ is a $d \times 1$ vector. If $\mathbf{y}$ is a $n \times 1$ vector and $\mathbf{w}$ is a $d \times 1$ vector, the Jacobian $\mathbf{J_w y}$ is a $d \times n$ matrix. As you can see, the shape of the Jacobian is generally determined as (shape of the input)$\times$(shape of the output). However, this gets funky when inputs and outputs have more than one dimension. For example, if we computed $\hat{\mathbf{X}} = \mathbf{ZW}$ as in matrix factorization, the Jacobian $\mathbf{J_Z \hat{X}}$ is a tensor of shape $(n \times k) \times (n \times d)$ where $\mathbf{Z} \in \mathbb{R}^{n \times k}$ and $\hat{\mathbf{X}} \in \mathbb{R}^{n \times d}$.

Here we will get more familiar with Jacobians, which are useful for understanding Backpropagation and implementing automatic differentiation. Suppose we have:

- X, an $n \times d$ matrix, $x_i \in \mathbb{R}^{n \times 1}$ are the columns of X
- Y, a $n \times k$ matrix
- W, a $k \times d$ matrix and w, a $d \times 1$ vector

For the following items, compute (1) the shape of each Jacobian, and (2) an expression for each Jacobian:

1. $f(w) = c$ (constant)
Answer: *TBD*
2. $f(w) = \|w\|^2$ (squared L2-norm)
Answer: *TBD*
3. $f(w) = w^\top x_i$ (vector dot product)
Answer: *TBD*
4. $f(w) = Xw$ (matrix-vector product)
Answer: *TBD*
5. $f(w) = w$ (vector identity function)
Answer: *TBD*
6. $f(w) = w^2$ (element-wise power)
Answer: *TBD*
7. $f(W) = W$ (matrix identity function)
Answer: *TBD*

3 Activations Per Layer, Keeps Linearity Away!

Based on the content we saw at the class lectures, answer the following:

1. Why are activation functions used in neural networks?
Answer: *TBD*
2. Write down the formula for three common action functions (sigmoid, ReLU, Tanh) and their derivatives (assume scalar input/output). Plot these activation functions and their derivatives on $(-\infty, +\infty)$.
Answer: *TBD*
3. What is the "vanishing gradient" problem? (respond in no more than 3 sentences) Which activation functions are subject to this issue and why? (respond in no more than 3 sentences).
Answer: *TBD*

4. Remember the Softmax function $\sigma(\mathbf{z})$ and how it extends sigmoid to multiple dimensions? Let's compute the derivative of Softmax for each dimension. Prove that:

$$\frac{d\sigma(\mathbf{z})_i}{dz_j} = \sigma(\mathbf{z})_i(\delta_{ij} - \sigma(\mathbf{z})_j)$$

where δ_{ij} is the Kronecker delta function.*
Answer: *TBD*

5. Use the above point to prove that the Jacobian of the Softmax function is the following:

$$\mathbf{J}[\sigma(\mathbf{z})] = \text{diag}(\sigma(\mathbf{z}) - \sigma(\mathbf{z})\sigma(\mathbf{z})^\top)$$

where diag(.) turns a vector into a diagonal matrix. Also, note that $\mathbf{J}[\sigma((z))] \in \mathbb{R}^{K\times K}$.
Answer: *TBD*

4 Simulating XOR

1. Can a single-layer network simulate (represent) an XOR function on $\mathbf{x} = [x_1, x_2]$?

$$y = \text{XOR}(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{x} = (0,1) \text{ or } \mathbf{x} = (1,0) \\ 0, & \text{if } \mathbf{x} = (1,1) \text{ or } \mathbf{x} = (0,0). \end{cases}$$

Explain your reasoning using the following single-layer network definition: $\hat{y} = \text{ReLU}(W \cdot \mathbf{x} + b)$
Answer: *TBD*

2. Repeat (1) with a two-layer network:

$$\mathbf{h} = \text{ReLU}(W_1 \cdot \mathbf{x} + \mathbf{b}_1)$$
$$\hat{y} = W_2 \cdot \mathbf{h} + b_2$$

Note that this is a multi-layer perceptron of three layers: an input layer $\mathbf{x} \in \mathbb{R}^2$, a hidden layer $\mathbf{h}$ with ReLU activation functions that are applied component-wise, and a linear output layer, resulting in scalar prediction $\hat{y}$. Provide a set of weights W_1 and W_2 and biases $\mathbf{b}_1$ and b_2 such that this model can accurately model the XOR problem.
Answer: *TBD*

3. Consider the same network as above (with ReLU activations for the hidden layer), with an arbitrary differentiable loss function $\ell : \{0,1\}\ \{0,1\} \to \mathbb{R}$ which takes as input $\hat{y}$ and y, our prediction and ground truth labels, respectively. Suppose all weights and biases are initialized to zero. Show that a model trained using standard gradient descent will not learn the XOR function given this initialization.
Answer: *TBD*

4. **Extra Credit:** Now let's consider a more general case than the previous question: we have the same network with an arbitrary hidden layer activation function:

$$\mathbf{h} = f(W_1 \cdot \mathbf{x} + \mathbf{b}_1)$$
$$\hat{y} = W_2 \cdot \mathbf{h} + b_2$$

Show that if the initial weights are any uniform constant, then gradient descent will not learn the XOR function from this initialization.
Answer: *TBD*

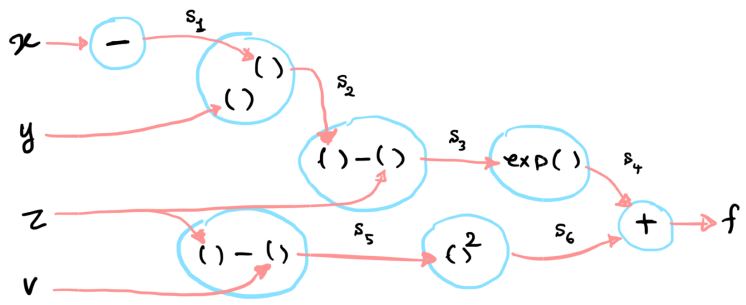
*https://en.wikipedia.org/wiki/Kronecker_delta

5 Computation Graph

Consider the following function:

$$f(x, y, z, v) = \exp(y^{-x} - z) + (z - v)^2$$

where $\log(.)$ i s the natural logarithm. Consider the following drawing that shows the computation graph for f. Notice that all the intermediate calculations are named (s_1, s_2, ...). Now answer the



A computation graph, so elegantly designed
With nodes and edges, so easily combined
It starts with inputs, a simple array
And ends with outputs, in a computationally fair way

Each node performs, an operation with care
And passes its results, to those waiting to share
The edges connect, each node with its peers
And flow of information, they smoothly steer

It's used to calculate, complex models so grand
And trains neural networks, with ease at hand
Backpropagation, it enables with grace
Making deep learning, a beautiful race
–ChatGPT Feb 3 2023

following questions:

1. Evaluate the computation graph from step 1 with input $(x, y, z, v) = (0, 1, 2, 3)$. Fill in all intermediate variables $\forall i : s_i$ and f in the drawing.
   Answer: *TBD*

2. Apply Backpropagation to the computation graph (step 1) to compute all the local gradients $\frac{\partial s}{\partial \tilde{s}}$ (for neighboring variables s and $\tilde{s}$) making use of the values obtained from forward propagation (step 1) evaluated at $(0, 1, 2, 3)$.
   Answer: *TBD*

3. Use the local gradients to compute the global gradients $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial v}]$. Hint: Use the chain rule.
   Answer: *TBD*

4. Compute $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial v}]$ with symbolic differentiation (i.e., using the chain rule).
   Answer: *TBD*

5. How many computations are repeated in step 4 compared to step 3, compared to step 2? Use this observation to explain why Backpropagation can be much more efficient than independently computing gradients for each variable.
   Answer: *TBD*

6 Programming

See the course website for the link to Google Colab: Colab Link