CSCE 580: Artificial Intelligence
Coding Project 3
Due: 04/01/2021 at 11:59pm

When debugging, use this code to set a breakpoint. It will be your best friend.

```
import pdb
pdb.set_trace()
```

Installation

We will be using the same `conda` environment as in Homework 0.

The entire GitHub repository should be downloaded again as changes were made to other files. You can download it with the green "Code" button and click "Download ZIP".

1 Backpropagation

For this exercise, you will implement forward propagation and backward propagation for a linear layer and a pointwise non-linear layer for a neural network using numpy. These two functions that you will implement will build your understanding of how neural networks work. The two layers you implement in this exercise can be composed in a variety of ways to build powerful function approximators.

There are N inputs of dimension D. The input matrix is $\mathbf{X} \in \mathbb{R}^{N \times D}$.
Where $\mathbf{x}_n$ is the n^{th} input and x_{nd} is the n^{th} input at index d.

For each layer, the gradient of some error function E will be calculated with respect to the parameters of the layer as well as with respect to the inputs to the layer. The simplicity of the backpropagation algorithm is in the fact that you do not need to know what E is. You only have to execute the chain rule when given the backpropagated gradients δ.

To check your implementations run:

```
python run_check_gradients.py --layer relu --num_inputs 5 --input_dim 10 --hidden_dim 100
python run_check_gradients.py --layer linear --num_inputs 5 --input_dim 10 --hidden_dim 100
```

The goal is to have the squared error between your implementation of the output computation step and the gradient computation be zero. **Vary the arguments to `run_check_gradients` (especially to check edge cases) to make sure your implementation is correct.**

1.1 Gradient of a Rectified Linear Unit (ReLU) Layer (10 pts)

Implement `relu_forward` and `relu_backward` in `assignments_code/assigment3.py`.

Given an input $\mathbf{X} \in \mathbb{R}^{N \times D}$, `relu_forward` returns the output $\mathbf{O} \in \mathbb{R}^{N \times D}$.
where $o_{nd} = \max(x_{nd}, 0)$.

Given a backpropagated gradient, $\delta \in \mathbb{R}^{N \times D}$, `relu_backward` backpropagates the gradient to the:

- inputs: $\frac{\partial E}{\partial x_{nd}} = \delta_{nd}(x_{nd} > 0)$.

1.2 Gradient of Linear Layer (40 pts)

Implement `linear_forward` and `linear_backward` in `assignments_code/assigment3.py`.
This linear layer takes as input a D dimensional vector and outputs a K dimensional vector.

There is a weight matrix $\mathbf{W} \in \mathbb{R}^{K \times D}$ and a bias vector $\mathbf{b} \in \mathbb{R}^K$.
`linear_forward` computes $\mathbf{O} \in \mathbb{R}^{N \times K} = \mathbf{X}\mathbf{W}^T + \mathbf{b}$
$o_{nk} = \sum_{i=0}^{D} x_{ni} w_{ki} + b_k$

Given a backpropagated gradient, $\delta \in \mathbb{R}^{N \times K}$, `linear_backward` backpropagates the gradient to the:

- weights: $\nabla_{\mathbf{W}} E = \delta^T \mathbf{X}$, $\frac{\partial E}{\partial w_{kd}} = \sum_{n=0}^{N} \delta_{nk} x_{nd}$
- biases: $\frac{\partial E}{\partial b_k} = \sum_{n=0}^{N} \delta_{nk}$
- inputs: $\nabla_{\mathbf{X}} E = \delta \mathbf{W}$, $\frac{\partial E}{\partial x_{nd}} = \sum_{k=0}^{K} \delta_{nk} w_{kd}$

2 PyTorch (50 pts)

For this exercise, you will implement regression using a neural network. You will be given states from the 8-puzzle and the shortest distance between the given state and the goal state (also known as the cost-to-go). You will design a neural network architecture (`get_nnet_model`) and an optimization procedure (`train_nnet`) using PyTorch for learning to map 8-puzzle states to their cost-to-go.

To test your code, run `python run_nn_puzzle8.py`. Your neural network should train in under a minute on a laptop CPU and should achieve a total MSE (mean-squared error) of 0.2 or less.

What to Turn In

Turn in your implementation of `proj_code/proj3.py` to Blackboard.
You can add helper functions to `proj_code/proj3.py`, if need be.