Lecture 14: Perceptron Learning, Backpropagation

Key Questions

- To train a perceptron, for each ________________ we first ________________; if ________________, then we ____________________.
- How do we know whether a weight contributed to an incorrect answer from a perceptron? In other words, how do we know how much each weight contributes to the overall activation for a given example?
- When can we stop updating weights for our perceptron?
- What special trick is necessary to train multi-layer perceptrons and other "deep" neural networks?
- What is a fundamental shortcoming of single perceptrons which is overcome by stacking them in multiple layers?
- Given the following labeled examples, can you train a neurode to approximate this function?

x_0	x_1	$f(x_0, x_1)$
5.0	3.69027	9.76109
5.0	3.31015	8.24062
4.0	0.55642	-1.77431
8.0	2.12101	0.48403

Notes

- The perceptron update rule:
  - $w_i = w_i + \Delta w_i$
  - $\Delta w_i = \lambda * (\text{actual} - \text{predicted}) * x_i$
  - (Sometimes actual is called y and predicted is called $\hat{y}$)
- Backpropagation uses the chain rule for derivatives to "blame" error on inputs from the bottom up to the top of the network.
  - This is part of why this "layer cake" model is popular
  - This is also why differentiable loss and activation functions are important: we are taking partial derivatives of loss with respect to weights and activations of input nodes, so all three operations (loss, activation, dot product) had better be differentiable.
- Backpropagation and gradient descent aren't guaranteed to find totally optimal parameter values
  - But in many practical problems this doesn't matter
  - See `https://en.wikipedia.org/wiki/Backpropagation` for more information

Your Questions