6.867: Exercises (Week 5)

October 11, 2017

Contents

Solution: Don't look at the solutions until you have tried your absolute hardest to solve the problems.

1 Silly friends

(a) Pat sees your neural network implementation with sigmoidal activation and says there's a much simpler way! Just leave out sigmoids, and let $g(a) = a$. The derivatives are a lot less hassle and it runs faster.

What might go wrong with Pat's approach?

(b) Chris comes in and says that your network is too complicated. The sigmoids are confusing and backpropagation would give basically the same answer if we used step functions instead.

What's wrong with Chris's approach?

(c) Sam sees your multi-layered layered network with linear activation function and says this is too complex! Just use a single-layered neural network without hidden units, they are equivalent anyway. Does Sam have a point?

(d) Jody sees that you are handling a multi-class problem with 4 classes by using 4 output values, where each target $y^{(i)}$ is actually a length-4 vector with three 0 values and a single 1 value, indicating which class the associated $x^{(i)}$ belongs to.

Jody suggests that you just encode the target $y^{(i)}$ values using integers 1, 2, 3, and 4.

What could go wrong with Jody's approach?

2 Yes vs No

In two-class classification with a standard sigmoid function, the negative log-likelihood error function is the cross entropy:

$$E(w) = -\sum_{n=1}^{N} (y^{(n)} \log h(x^{(n)}, w) + (1 - y^{(n)}) \log(1 - h(x^{(n)}, w)) \quad .$$

(a) Assuming input x and weight w are scalar, and that $N = 1$ (there is a single training example), what is $\partial E(w)/\partial w$?

(b) What would the neural network weight-update rule be?

3 Sigmoid

We decided to train a single sigmoid unit using the following error function:

$$E(w) = 1/2 \sum_i (y(x^i, w) - y^{i*})^2 + 1/2\beta \sum_j w_j^2$$

where $y(x^i, w) = s(x^i \cdot w)$ with $s(z) = 1/(1 + e^{-z})$ being our usual sigmoid function.

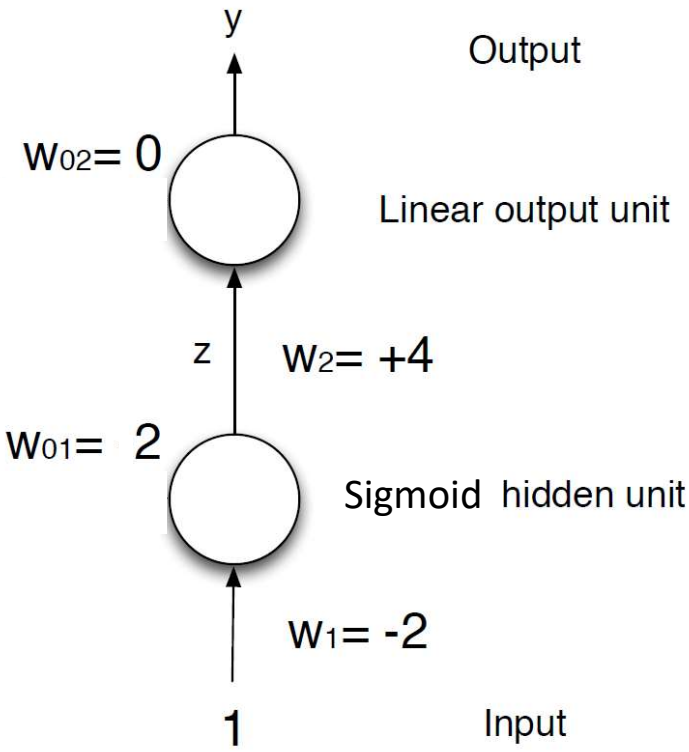
(a) Here are two graphs of the output of the sigmoid unit as a function of a single feature x. The unit has a weight for x and an offset. The two graphs are made using different values of the magnitude of the weight vector ($\|w\|^2 = \sum_j w_j^2$).

Which of the graphs is produced by the larger $\|w\|^2$? Explain.

(b) Why might penalizing large $\|w\|^2$, as we could do above by choosing a positive β, be desirable?

4 Backpropagation

Here you see a very small neural network: it has one input unit, one hidden unit (sigmoid), and one output unit (linear).



Here w_{01} and w_{02} refer to the offsets for hidden units 1 and 2 respectively. The output unit computes $f(z) = w_2 * z + w_{02}$, where z is the output of the previous layer.

Let's consider one training case. For that training case, the input value is 1 (as shown in the diagram), and the target output value $t = 1$. We're using the following loss function:

$$E = \frac{1}{2}(t - y)^2$$

Please supply numeric answers; the numbers in this question have been constructed in such a way that you don't need a calculator. Show your work in case of mis-calculation in earlier steps.

(a) What is the output of the hidden unit for this input?

(b) What is the output of the output unit for this input?

(c) What is the loss, for this training case?

(d) What is the derivative of the loss with respect to w_2, for this training case?

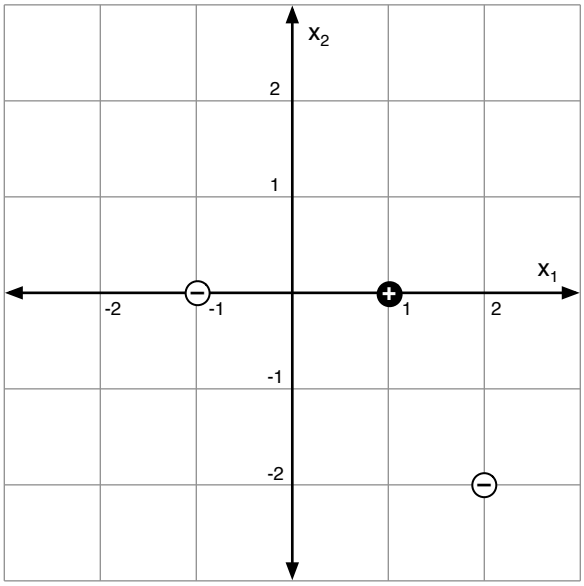
(e) What is the derivative of the loss with respect to w_1, for this training case?

(f) With sigmoidal activation, the derivative with respect to w_1 and w_2 are

$$\frac{\partial E}{\partial w_2} = -(t-y)z, \text{ and } \frac{\partial E}{\partial w_1} = -(t-y)\cdot w_2 \cdot z \cdot (1-z)\cdot x.$$

Assume that we now use the rectified linear unit (ReLU) as our activation (or a *ramp* function). This means that $z = max(0, w_1 x + w_{01})$. What is the derivative of the loss with respect to w_1 and w_2 *at differentiable points* with ReLU? Don't use numerical value for this question.

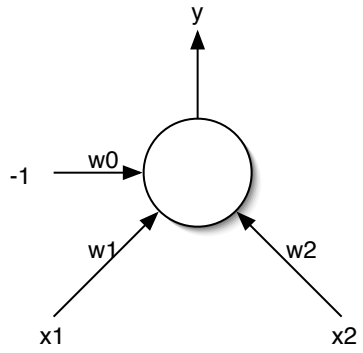
5 Neural Net



Data points are: Negative: (-1, 0) (2, -2) Positive: (1, 0)

Recall that for neural nets with sigmoidal output units, the negative class is represented by a desired output of 0 and the positive class by a desired output of 1. Hint: Some useful values of the sigmoid $s(z)$ are $s(-1) = 0.27$ and $s(1) = 0.73$.

Assume we have a single sigmoid unit:



Assume that the weights are $w_0 = 0, w_1 = 1, w_2 = 1$. What is the computed y value for each of the points on the diagram above?

(a) $x = (-1, 0)$

(b) $x = (2, -2)$

(c) $x = (1, 0)$

(d) What would be the change in w_2 as determined by backpropagation using a step size (η) of 1.0? Assume the squared loss function. Assume that the input is $x = (2, -2)$ and the initial weights are as specified above. Show the formula you are using as well as the numerical result.

6 Probable cause

You have a binary classification problem, but your training examples are only labeled with probabilities, so your data set consists of pairs $(x^{(i)}, p^{(i)})$, where $p^{(i)}$ is the probability that $x^{(i)}$ belongs to class 1.

You want to train a neural network **with a single unit** to predict these probabilities.

(a) What is a good choice for the activation function of your final output unit?

(b) You can think of the training label $p^{(i)}$ as specifying a true probability and the current output of your neural network as specifying an approximate probability $q^{(i)}$. You think a reasonable objective would be to minimize the KL divergence $KL(p \parallel q)$ between the distributions implicitly represented by the predicted and target outputs. So, the empirical risk would be

$$E = \sum_i -(p^{(i)} \log q^{(i)} + (1 - p^{(i)}) \log(1 - q^{(i)}))$$

If $q^{(i)} = f(w \cdot x^{(i)})$ where f is your activation function, what is the SGD (stochastic gradient descent) weight update rule when using the KL objective function above? For simplicity, assume that $x^{(i)}$ and w are both scalars and write f' for the derivative of f.