6.867: Exercises (Week 6)

October 14, 2016

Contents

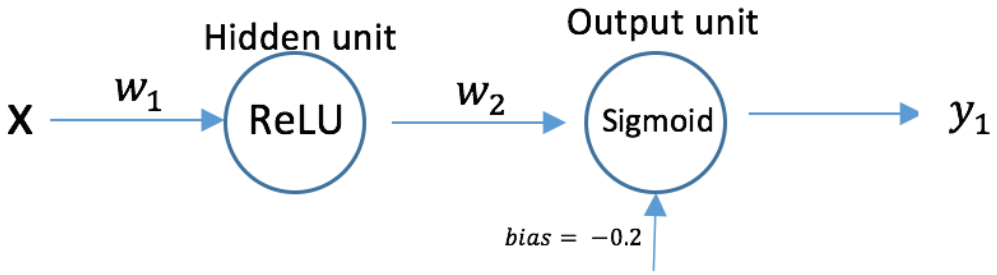
Solution: Don't look at the solutions until you have tried your absolute hardest to solve the problems.

1 ReLU Backpropagation**

1.1 Single output network

The rectified linear unit (ReLU) is a popular activation function for hidden layers. The activation function is a ramp function $f(z) = \max(0, z)$ where $z = wx$. This has the effect of simply thresholding its input at zero. Unlike the sigmoid, it does not saturate near 1 and is also simpler in gradient computations, resulting in faster convergence of SGD. Furthermore, ReLUs can allow networks to find sparse representations, due to their thresholding characteristic, whereas sigmoids will always generate non-zero values. However, ReLUs can have zero gradient when the activation is negative, blocking the backpropagation of gradients.

Here you use a very small neural network: it has one input unit, taking in a value x, one hidden unit (ReLU), and one output unit (sigmoid). We include a bias term of -0.2 on the sigmoid unit.



We use the following quantities in this problem:

$$z_1 = w_1 x$$
$$a_1 = \text{ReLU}(z_1)$$
$$z_2 = w_2 a_1 - 0.2$$
$$y = \sigma(z_2)$$

The weights are initially $w_1 = \frac{1}{10}$ and $w_2 = -1$.

Let's consider one training example. For that training case, the input value is $x = 2$ (as shown in the diagram), and the target output value $t = 1$. We're using the following loss function:

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

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?

Solution:

$$a = \text{ReLU}(z_1) = \max(0, w_1 x) = \max(0, \frac{1}{10} \times 2) = \frac{1}{5}$$

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

Solution:

$$y = \sigma(w_2 \max(0, w_1 x)) = \sigma(-1 \times \frac{1}{5} - 0.2) = \sigma(-.4) \approx \frac{2}{5}$$

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

Solution:

$$E = \frac{1}{2}(y - t)^2 = \frac{1}{2}(\frac{2}{5} - 1)^2 = 9/50$$

(d) Write out an abstract symbolic expression for derivative of the loss with respect to w_1 as repeated applications of the chain rule. For example, for the derivative of the loss with respect to w_2, we would write $\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial w_2}$.

Solution: $\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial a} \frac{\partial a}{\partial z_1} \frac{\partial z_1}{\partial w_1}$

(e) Write the expression for each partial derivative in the chain rule expansion from the previous part. For example, $\frac{\partial y}{\partial z_2} = y(1 - y)$.

Solution: $\frac{\partial E}{\partial y} = y - t$

$\frac{\partial y}{\partial z_2} = y(1 - y)$

$\frac{\partial z_2}{\partial a} = w_2$

$\frac{\partial a}{\partial z_1} = \begin{cases} 1 & \text{if } w_1 x > 0 \\ 0 & \text{if } w_1 x < 0. \end{cases} = I[w_1 x > 0]$ (indicator function)

$\frac{\partial z_1}{\partial w_1} = x$

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

Solution:

$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial y} \frac{\partial y}{\partial z_2} \frac{\partial z_2}{\partial a} \frac{\partial a}{\partial z_1} \frac{\partial z_1}{\partial w_1} =$

$(y - t) \times y(1 - y) \times w_2 \times I[w_1 x > 0] \times x =$

$(\frac{2}{5} - 1) \times \frac{2}{5}\frac{3}{5} \times -1 \times 1 \times 2 = 36/125 = .288$

(g) What would the update rule for w_1 be? With $\eta =$

Solution:

$$w_1 := w_1 - \eta \frac{\partial E}{\partial w_1} := w_1 - .288\eta$$

(h) If η is large enough, w_1 will update from its current value of 0.1 to a negative value. Assume our new value is $w_1 = -0.1$. What will be the output of the output unit for an input of $x = 2$?

Solution: The ReLU will output 0, since w_1 is negative, so only the bias term will remain in the sigmoid and the output will be $\sigma(-0.2)$

(i) What will happen when we try to update the weight, using this new example, for w_1 for any value of target? Why?

Solution: The ReLU gate is closed and gradients will not flow backwards through the ReLU unit. w_1 will not be updated with SGD. In fact, if the training examples are all positive, the input to the ReLU will be always be negative, effectively killing the ReLU.

(j) Is it a bad idea to have a ReLU activation at the output layer?

Solution: Yes, if the input to the ReLU is mostly negative, it will fail to backpropagate gradients through the entire network.

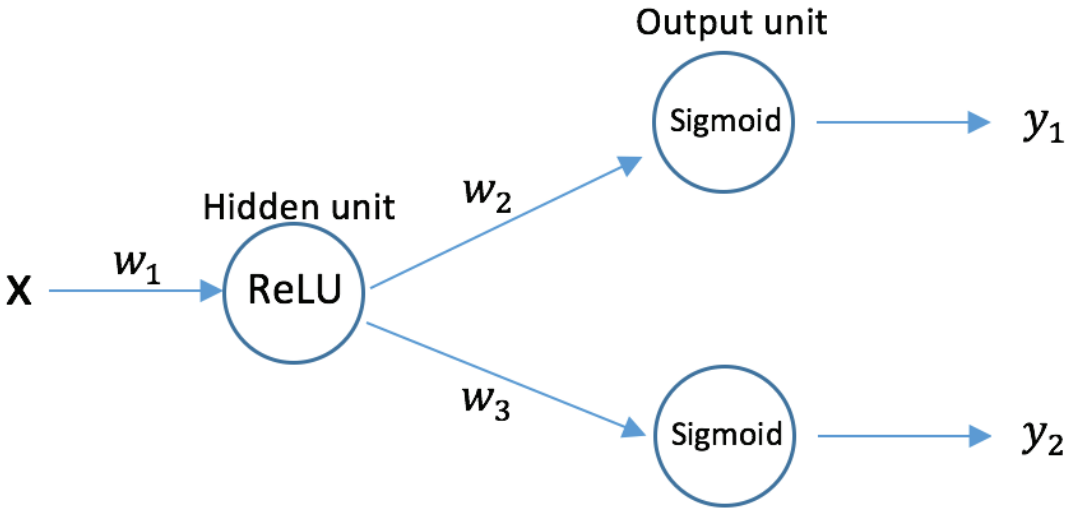
(k) Consider the following activation function:

$$f(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } otherwise. \end{cases}$$

for some small alpha, e.g. $\alpha = 0.01$, and $z = wx$. Does this address the problem of dying ReLUs?

Solution: ReLU units don't backpropagate any error for negative inputs. The leaky ReLU allows a small error to backpropagate even with negative input.

1.2 Multiple output network



(l)

$$a_1 = \text{ReLU}(0, w_1 x)$$
$$y_1 = \sigma(w_2 a_1)$$
$$y_2 = \sigma(w_3 a_1)$$

Write out an abstract symbolic expression for the derivative of the loss with respect to w_1 for the network above with two output units, as repeated applications of the chain rule.

Solution:

$$E_{total} = \frac{1}{2}(y_1 - t_1)^2 + \frac{1}{2}(y_2 - t_2)^2 = E_1 + E_2$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_1}{\partial w_1} + \frac{\partial E_2}{\partial w_1}$$

$$\frac{\partial E_1}{\partial w_1} = \frac{\partial E_1}{\partial y} \frac{\partial y}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$

Similarly, for E_2

Multi-output (multi-class) networks are used in many settings such as object recognition, where we are trying to classify an image as being one of K objects. Each of the K possible objects would correspond to an output unit in the network. For this purpose, the sigmoid activation and squared loss are replaced by softmax activation and cross-entropy loss. This is similar to the multi-class logistic regression we saw in the week 4 exercises.

The softmax is given by:

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad .$$

(m) When $K > 3$, why might sigmoid units be a bad idea?

> **Solution:** With sigmoid output units for multiple classes, we cannot guarantee that at most one output unit activates. The normalization in the softmax makes this happen

2 Regularization**

2.1 Weight Decay

We can add L_1 and L_2 penalties to our cost function to regularize our network and prevent overfitting, as we have done with other classification methods. See Neural Networks and Deep Learning by Michael Nielson for derivations and details.

(a) The loss for L_2 regularized case is:

$$L(w) = E(w) + \lambda |w|^2$$

where E(w) is the error. The update for the L_2 regularized case is:

$$w := (1 - \frac{2\eta\lambda}{n})w - \eta\frac{\partial E}{\partial w}$$

with regularization. How does this update affect our weights w as we increase λ?

> **Solution:** This update first rescales the weight by $(1 - \frac{n\lambda}{n})$, which is the sole difference from the non-regularized update. As we increase λ, this increases the rate at which the weights decrease.

(b) Similarly, the loss for L_1 regularized case is:

$$L(w) = E(w) + \lambda |w|$$

The update for this case is:

$$w := w - (\frac{\eta\lambda}{n})sgn(w) - \eta\frac{\partial E}{\partial w}$$

where $sgn(w)$ is +1 is w is positive and -1 if it is negative. How does this compare to the L_2 update?

> **Solution:** Rather than shrinking w by an amount proportional to w, as in the L_2 case, here w shrinks by a constant amount. This has the effect of shrinking weights with small magnitude w, L_1 regularization will shrink weights more than L_2, concentrating weights on a smaller number of connections.

2.2 Dropout

Dropout is a regularization technique to reduce overfitting to the training data. A randomly selected portion (eg. 20%) of neurons are ignored during each update cycle of training - these neurons 'drop out' of the network.

During training, without dropout, we compute the activations of a layer l as $a_l = f(z_l)$ Assume we are using a dropout probability of 20%, $a_l = f(z_l)m_l$, where m_l is a mask of the same length with 20% of the entries set to 0 and the others set to 1.

(a) How might this have a regularization affect?

> **Solution:** Weights in the network settle into their context within the network over the course of training. Each weight becomes tuned for specific features providing. Neighboring neurons become to rely on this specialization, which if taken too far can overfit the training data. These neurons are said to co-adapt.
>
> If neurons are randomly dropped out of the network during training, other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This results in multiple independent internal representations being learned by the network.

(b) During the feedforward pass without dropout, the inputs to layer $l+1$ are $z_{l+1} = W_{l+1}a_l$. With 20% dropout, how should we change this expression?

> **Solution:** $z_{l+1} = 0.8W_{l+1}a_l$

2.3 Augmenting with Noisy Training Data

A third way to introduce regularization is to augment the training data with slightly perturbed training examples. We perturb them by applying operations that we might see in the real-world.

(a) Why might this be a good way to improve performance?

> **Solution:** Performance can be improved by adding more examples to the training set. However, it is often difficult to obtain more new training data. This method allows us to expand our training set and incorporate interesting variation in the data.

(b) How might you do this on MNIST?

> **Solution:** Minor rotations or stretching of each input image would reflect variation we might expect in handwritten digits

3 Convolutional Network Architecture**

Consider the following 1D input and two convolutional filters:

(a) Convolve the first filter with the input.

Solution: 2 1 2

(b) Convolve the second filter with the input.

Solution: 2 -4 5

(c) What would the output of the convolutional layer look like with both filters?

Solution: A 3x2 matrix with columns 2 1 2 and 2 -4 5.

(d) In the previous part, the spatial size of the output decreased from that of the original input -ie. it had fewer neurons. In some settings, we preserve the size by applying zero padding. What would the output of the convolutional layer look like in this case?

Solution: A 5x2 matrix with columns -2 2 1 2 1 and -1 2 -4 5 -4

(e) Continuing to use zero padding of size 1, we now use a stride length of 2. What is the output of the convolutional layer now? Compare how this relates to the case with no stride.

Solution: A 3x2 matrix with columns -2 1 1 and -1 -4 -4

This contains alternating weights from the output of the previous part.

(f) If our filters were size 5, rather than size 3, with stride 1, how much padding would you apply to maintain output size?

Solution: Pad each side with 2 zeros. In general, set to (F-1)/2.

(g) What is a general expression for the output size (number of neurons) in a convolutional layer, in terms of the filter size (F), stride length (S), amount of padding (P), and the input size (W)?

Solution: $(W - F + 2P)/S + 1$

(h) Consider a 10x10 grayscale image input (no RGB channel). If you made a 1-layer fully connected network, with 1 hidden, how many weight parameters would be required in the hidden layer including bias terms (ignore the output layer)?

> **Solution:** 100 weights for the first layer + 1 bias terms = 101

(i) Consider a 10x10 grayscale image input (no RGB channel). If you made a 2-layer fully connected network, with 10 and 5 hidden units respectively, how many weight parameters would be required in these hidden layers including bias terms (ignore the output layer)?

> **Solution:** 1000 weights for the first layer + 10 bias terms and 50 weights + 5 bias terms for the second layer

(j) With the same 10x10 input image, you now use a convolutional layer with F = 2 (square filters), S = 2 (in both dimensions), and P = 0. How is the output volume of the first convolutional layer with 2 filters, including bias terms?

> **Solution:** (10 - 2 + 2*0)/2 + 1 = 5 in each dimension. The convolutional layer has output volume 5x5x2.

(k) How many parameters does the above convolutional layer have?

> **Solution:** The convolutional layer has output volume 5x5x2 = 50 neurons. Each one of these 50 neurons is connected to a 2x2 region of the input (its receptive field), giving it 4 weights and 1 bias term. As a result, the number of parameters is 50x5 = 250.

(l) One advantage of convolutional networks is that the number of free parameters can be controlled by parameter sharing for a filter across spatial dimensions. Each filter is replicated across the entire visual field (image). That is, if a filter is used at one spatial coordinate (x_1, y_1), the same weights are also used in a different position (x_2, y_2). Adopting the parameter sharing scheme, how many parameters does the convolutional layer now have?

> **Solution:** With parameter sharing, we effectively don't have to have different weights across the 5x5 dimension. So rather than 5x5x2x5 = 250 parameters, we have $2 * 5 = 10$ parameters.

4 Silly friends*

(a) Like Jody last week, Evelyn sees that you are handling a multi-class problem with 4 classes and suggests that you use as a target value $y^{(i)}$ a vector of two output values, each of which can be 0 or 1, to encode which one of four classes it belongs to.

What is good or bad about Evelyn's approach?

> **Solution:** This is a bad idea. 1) Labels are not equally treated, "00" is closer to "01" or "10" than to "11"; 2) It only works for number of labels of 2^i for some i due to it's encoding nature.

(b) Seeing you working on an implementation of a neural network with sigmoid units, where the inputs are in the range $[0, 1]$, Jean suggests that you initialize the weights randomly in the range $[0, 100]$.

Is Jean's idea good?

> **Solution:** No. The range is too large, causing the output variables to be extremely close to 0 or 1, and the gradient to be near zero. It's also generally better to initialize to both positive and negative values.

5 Activation energy

A bunch of AI students walked into a bar...and promptly started debating alternative activation functions for a neural network intended to do binary classification. The input data is in R^2; the weight vector is 3-dimensional.

The network in fact has just a single unit with two input dimensions.

For each of the suggested activation functions, say:

- Whether it makes sense to train it with the cross-entropy objective function:

$$y \log o + (1 - y) \log(1 - o)$$

  where y is the desired output and o is the actual (the output of the network).

- Whether it can represent the same class of separators in the two-dimensional input space as the sigmoidal activation function.

- If it can represent the same class, explain why.

- If it cannot, then draw a separator that can be represented by a sigmoidal activation on a linear combination of the inputs but not by this activation on a linear combination of the inputs **or** a separator that can be represented by this activation function on a linear combination of the inputs, but not by a sigmoidal activation function on a linear combination of the inputs and explain why.

(a)

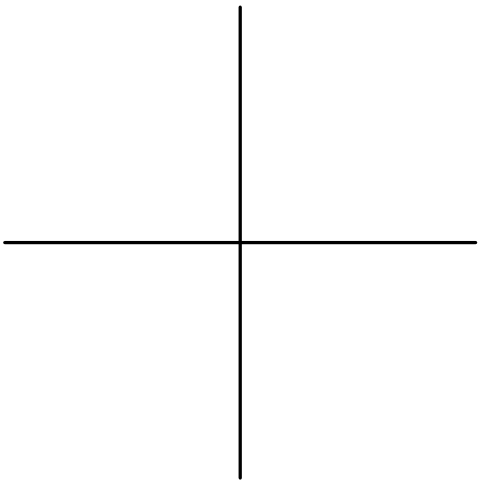
$$f(a) = \sin(a)$$

Sensible to use cross-entropy: ◯ Yes √ **No**

Same class of separators as sigmoid? ◯ No √ **No**

If yes, explain why. If no, draw separator as explained above.

Solution: It will have parallel stripes of separation through the space.



(b)

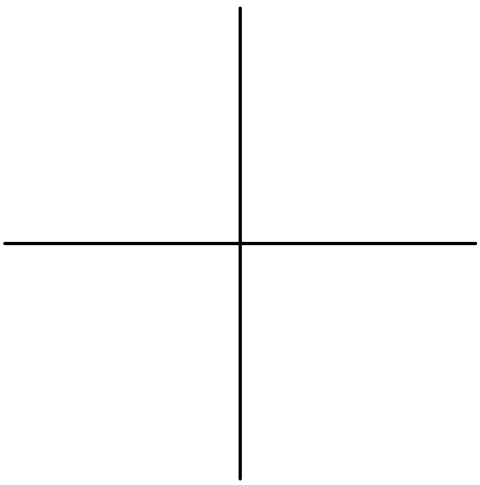
$$f(a) = \begin{cases} 0 & \text{if } a < 0 \\ 1 & \text{if } a > 1 \\ x & \text{otherwise} \end{cases}$$

Sensible to use cross-entropy: $\surd$ **Yes** ◯ No

Same class of separators as sigmoid? $\surd$ **Yes** ◯ No

If yes, explain why. If no, draw separator as explained above.

Solution: It will cross 0.5 along a single line through the original space.



(c)

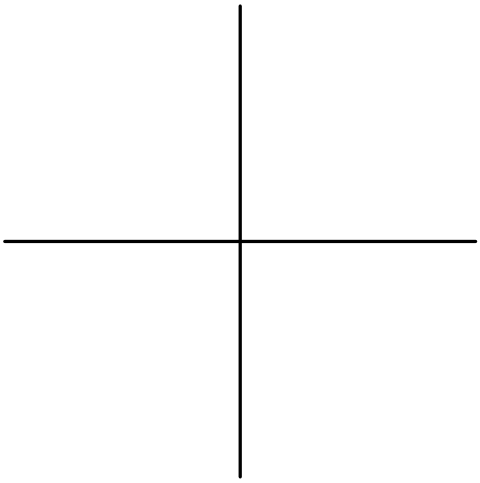
$$f(a) = e^a$$

Sensible to use cross-entropy: ◯ Yes √ **No**

Same class of separators as sigmoid? √ **Yes** ◯ No

If yes, explain why. If no, draw separator as explained above.

Solution: It will cross 0.5 along a single line through the original space.



(d)

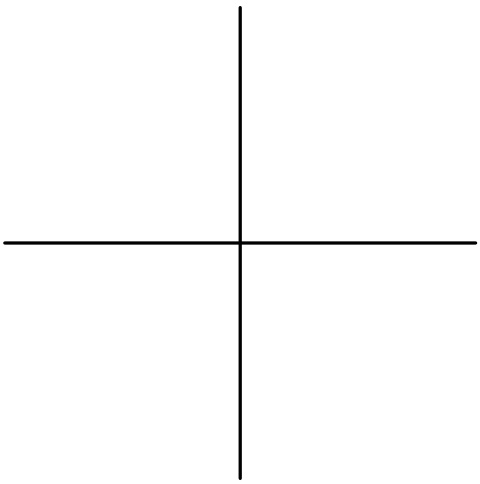
$$f(a) = a^2$$

Sensible to use cross-entropy: ◯ Yes √ **No**

Same class of separators as sigmoid? ◯ Yes √ **No**

If yes, explain why. If no, draw separator as explained above.

Solution: It may cross 0.5 in two lines.



(e)

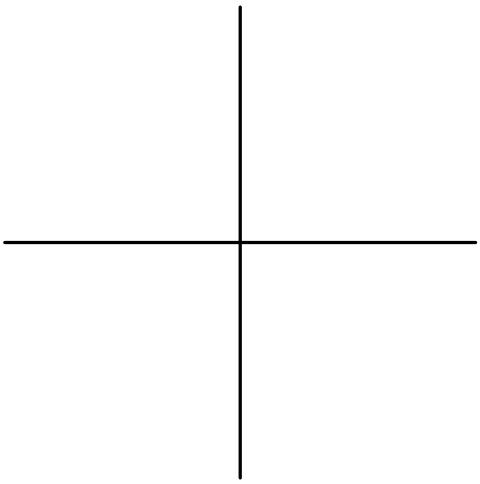
$$f(a) = a$$

Sensible to use cross-entropy: ◯ Yes √ **No**

Same class of separators as sigmoid? √ **Yes** ◯ No

If yes, explain why. If no, draw separator as explained above.

Solution: It will cross 0.5 along a single line through the original space.



6 Neural network short answer

1. If you train a neural-net classifier using stochastic gradient descent, and you use a too small learning rate, what can go wrong?

   Solution: The network learns too slowly, not reaching an optimum

2. How can you determine that you have used too small a learning rate?

   Solution: Compare to a larger learning rate to see if the objective decreases further

3. If you train a neural-net classifier using stochastic gradient descent, and you use a too large learning rate, what can go wrong?

   Solution: The weights and objective function diverge, so there is no learning at all

4. How can you determine that you have used too large a learning rate?

   Solution: SGD will oscillate dramatically

5. Is final error on the training set a good measure for finding out whether the learning rate was set too large?

   Solution: No

6. Suppose you train a neural-net classifier using stochastic gradient descent, and you measure training error and validation error after N iterations.

   Which of the two errors is a better estimate for the error on unseen future data and why?

   Solution: Validation error. It is based on unseen data during training and used to update hyperparameters.

7. In a convolutional layer of a neural network, what gives rise to a smaller output dimentionality: a small stride or a large stride?

   Solution: A large stride

8. In a convolutional layer of a neural network, can you use 1x1 convolution to learn a linear classifier on top of the inputs? If yes, what format should the input have?

   Solution: Yes. The input should have multiple channels.

9. For effective training of a neural network, the network should have at least 4 times as many weights as there are training samples? If yes, why? If no, why not?

   Solution: No, the number of training samples should be larger than the number of weights

10. Assume the following (momentum) weight update rule:

$$m_0 = 0 \tag{6.1}$$

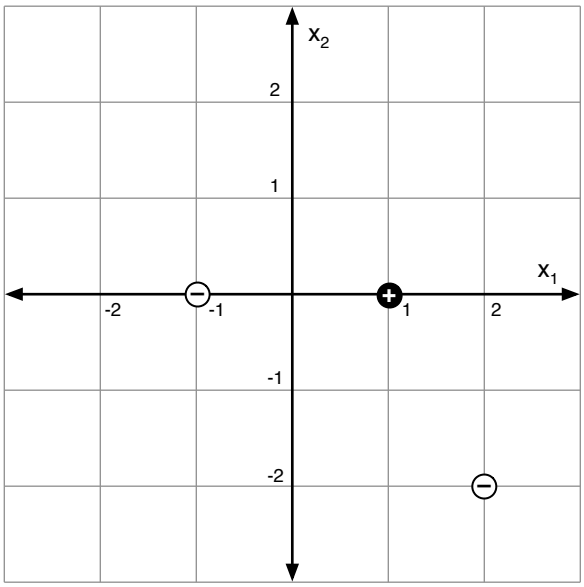
$$m_t = \beta m_{t-1} + (1-\beta)\frac{\partial E}{\partial w} \tag{6.2}$$

$$w_t = w_{t-1} + \lambda m_t \tag{6.3}$$

11. For what value of β do you obtain the standard gradient descent rule?

Solution: $/\mathtt{beta} = 0$

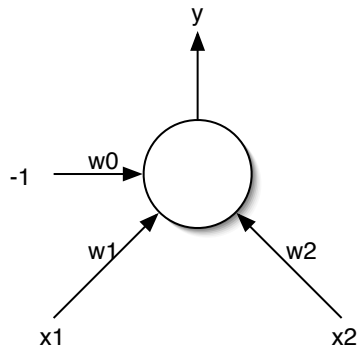
7 Neural Net*



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) $x = (-1, 0)$

Solution: $y = \mathbf{s}(\mathbf{0} \cdot -\mathbf{1} + \mathbf{1} \cdot -\mathbf{1} + \mathbf{1} \cdot \mathbf{0}) = \mathbf{s}(-\mathbf{1}) = \mathbf{0.27}$

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

Solution: $y = \mathbf{s}(\mathbf{0} \cdot -\mathbf{1} + \mathbf{1} \cdot -\mathbf{2} + \mathbf{1} \cdot \mathbf{2}) = \mathbf{s}(\mathbf{0}) = \mathbf{0.5}$

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

Solution: $y = \mathbf{s}(\mathbf{0} \cdot -\mathbf{1} + \mathbf{1} \cdot \mathbf{1} + \mathbf{1} \cdot \mathbf{0}) = \mathbf{s}(\mathbf{1}) = \mathbf{0.73}$

(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.

1. $\Delta w_2 =$

Solution: Solution:

$$\begin{aligned}
\Delta w_2 &= -\eta \frac{\partial E}{\partial w_2} \\
&= -\eta \frac{\partial E}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_2} \\
&= -\eta (y - y^i) y (1 - y) x_2 \\
&= (-1)(0.5 + 0)(0.5)(0.5)(-2) \\
&= 0.25
\end{aligned}$$

Derivations:

$$\begin{aligned} E &= \frac{1}{2}(y - y^i)^2 \\ y &= s(z) \\ z &= \sum_{i=0}^{2} w_i x_i \\ \frac{\partial E}{\partial y} &= y - y^i \\ \frac{\partial y}{\partial z} &= y(1-y) \\ \frac{\partial z}{\partial w_2} &= x_2 \end{aligned}$$

8 Lots of class

(Bishop 5.5) Show that the maximizing likelihood for a multi-class neural network model in which the network outputs have the interpretation $h_k(x, w) = p(y = k \mid x)$ is equivalent to the minimization of the cross-entropy error function

$$E(w) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_k^{(i)} \ln h_k(x^{(i)}, w) \quad .$$

Solution: For the given interpretation of $h_k(x, w)$, the conditional distribution of the target vector for a multi-class neural network is given by

$$p(y|w_1, \ldots, w_K) = \prod_{k=1}^{K} h_k(x, w)^{y_k}$$

Thus, for a dataset of N points, the total likelihood function will be

$$\prod_{n=1}^{N}\prod_{k=1}^{K} h_k(x^{(n)}, w)^{y_k^{(n)}}$$

Taking the negative logarithm will result in the cross-entropy error function.

9 Noisy targets

(Bishop 5.4) Consider a binary classification problem in which the target values are $y \in \{0, 1\}$ with a network output $h(x, w)$ that represents $p(y = 1 \mid x)$, and suppose that there is a probability ϵ

that the class label on a training data point has been incorrectly set.

(a) Assuming independent and identically distributed data, write down the error function corresponding to the negative log likelihood. Verify that the *cross entropy* error function is obtained when $\epsilon = 0$. Note that this error function makes the model robust to incorrectly labeled data, in contrast to the usual error function.

Solution: Let $t^{(n)} \in \{0, 1\}$ denote the dataset label and $y^{(n)} \in \{0, 1\}$ denote the true class label for $x^{(n)}$. From the rules of probability we have

$$p(t = 1|x) = \sum_{y=0}^{1} p(t = 1|y)p(y|x) = (1 - \varepsilon)h(x^{(n)}, w) + \varepsilon(1 - h(x^{(n)}, w)) \tag{9.1}$$

The conditional probability of the data label is then

$$p(t|x) = p(t = 1|x)^t(1 - p(t = 1|x))^{1-t} \tag{9.2}$$

Forming the likelihood and taking the negative logarithm we have the error function in the form

$$E(w) = -\sum_{n=1}^{N} \Big\{ t^{(n)} \log \Big[(1 - \varepsilon)h(x^{(n)}, w) + \varepsilon(1 - h(x^{(n)}, w)) \Big] + \tag{9.3}$$

$$(1 - t^{(n)}) \log \Big[1 - (1 - \varepsilon)h(x^{(n)}, w) - \varepsilon(1 - h(x^{(n)}, w)) \Big] \Big\} \tag{9.4}$$

(b) What is the form of the partial derivative with respect to a single weight in the output layer?

(c) How does the stochastic gradient update rule for $\epsilon = 0.1$ differ from the case when $\epsilon = 0$?

10 MLE

(Bishop 5.2) Show that maximizing the likelihood function under the conditional distribution

$$p(y \mid x, w) = \mathcal{N}(y \mid h(x, w), \beta^{-1}\mathbf{I})$$

for a multi-output neural network is equivalent to minimizing the sum-of-squares error function

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} \|h(x^{(n)}, w) - y^{(n)}\|^2 \ .$$

Solution: The likelihood function for an i.i.d. dataset $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ under the corresponding conditional distribution is given by

$$\prod_{n=1}^{N} \mathcal{N}(y^{(n)}|h(x^{(n)}, w), \beta^{-1}\mathbf{I}) \tag{10.1}$$

Taking the logarithm we have

$$\sum_{n=1}^{N} \log \mathcal{N}(y^{(n)}|h(x^{(n)}, w), \beta^{-1}\mathbf{I}) \tag{10.2}$$

$$= -\frac{1}{2}\sum_{n=1}^{N}(h(x^{(n)}, w) - y^{(n)})^{\top}(\beta\mathbf{I})(h(x^{(n)}, w) - y^{(n)}) + \textit{Const} \tag{10.3}$$

$$= -\frac{\beta}{2}\sum_{n=1}^{N}\|h(x^{(n)}, w) - y^{(n)}\|^2 + \textit{Const} \tag{10.4}$$

where the "Const" term is independent of w. Thus maximizing the likelihood is equivalent to maximizing the log-likelihood, and is subsequently equivalent to minimizing $E(w)$.

11 Not independent

(Bishop 5.3) Consider a regression problem involving multiple target variables in which it is assumed that the distribution of the targets, conditioned on the input vector x is a Gaussian of the form

$$p(y \mid x, w) = \mathcal{N}(y \mid h(x, y), \Sigma)$$

where $h(x, y)$ is the output of a neural network with input vector x and weight vector w and Σ is the covariance of the assumed Gaussian noise on the targets.

Given a set of independent observations $x^{(i)}, y^{(i)}$, write down the error function that must be minimized in order to find the maximum likelihood solution for w. First, assume that Σ is fixed and known.

Second, assume that Σ is also to be determined from the data and write down an expression for the maximum likelihood solution for Σ. Note that the optimizations for w and Σ are now coupled, in contrast to the case of independent target variables.

Solution: In this case, the likelihood function becomes

$$\prod_{n=1}^{N} \mathcal{N}(y^{(n)}|h(x^{(n)}, w), \Sigma)$$

And the corresponding log-likelihood function is

$$\sum_{n=1}^{N} \log \mathcal{N}(y^{(n)}|h(x^{(n)}, w), \Sigma)$$

$$= -\frac{1}{2}\sum_{n=1}^{N}(h(x^{(n)}, w) - y^{(n)})^{\top}\Sigma^{-1}(h(x^{(n)}, w) - y^{(n)}) - \frac{N}{2}(\log|\Sigma| + K\log(2\pi))$$

If we first treat Σ as fixed and known, we can drop terms that are independent of w, and by changing the sign we get the error function that we are going to minimize:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}(h(x^{(n)}, w) - y^{(n)})^\top \Sigma^{-1}(h(x^{(n)}, w) - y^{(n)})$$

If we consider maximizing log-likelihood w.r.t. Σ, the terms that need to be kept are

$$-\frac{1}{2}\sum_{n=1}^{N}(h(x^{(n)}, w) - y^{(n)})^\top \Sigma^{-1}(h(x^{(n)}, w) - y^{(n)}) - \frac{N}{2}\log|\Sigma|$$

Rewriting the first term we get

$$-\frac{1}{2}\mathrm{Tr}\left[\Sigma^{-1}\sum_{n=1}^{N}(h(x^{(n)}, w) - y^{(n)})(h(x^{(n)}, w) - y^{(n)})^\top\right] - \frac{N}{2}\log|\Sigma|$$

Taking the derivative w.r.t. Σ^{-1} to be 0 we have

$$\Sigma = \frac{1}{N}\sum_{n=1}^{N}(h(x^{(n)}, w) - y^{(n)})(h(x^{(n)}, w) - y^{(n)})^\top$$

which is dependent on w. A possible way to address this mutual dependency between w and Σ when doing optimization, is to adopt an iterative scheme, alternating between updates of w and Σ until some convergence criterion is reached.

12 Noisification

This problem is more difficult, but kind of cool.

(Bishop 5.27) Consider a dataset with input-target pairs $(x^{(i)}, t^{(i)})$. Define the expected error as

$$E = \frac{1}{2}\int\int (y(x) - t)^2 p(t|x)p(x)\,dx\,dt.$$

Consider training with transformed inputs $z^{(i)} = x^{(i)} + \xi$ where $\xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Define the expected error of transformed inputs as

$$\tilde{E} = \frac{1}{2}\int\int\int (y(x + \xi) - t)^2 p(z|x)p(x)p(\xi)\,dx\,dt\,d\xi.$$

By following an argument analogous to that of section 5.5.5 in Bishop, show that the expected error of transformed inputs can be written in the form of Tikhonov regularization as

$$\tilde{E} = E + \lambda\Omega,$$

where

$$\Omega = \frac{1}{2}\int \|\nabla h(x, w)\|^2 p(x)\,dx.$$

Solution: If we have the transformed data $z^{(i)} = x^{(i)} + \xi$, then

$$\frac{\partial z_i}{\xi_j} = \delta_{ij}$$

where $\delta_{ij} = 1$ if $i = j$ and 0 if otherwise. Since the first order derivative is constant, there are no higher order derivatives. We now make use of this result to obtain the derivatives of $h(x, w)$ w.r.t. ξ:

$$\frac{\partial h(x,w)}{\partial \xi_i} = \sum_j \frac{\partial h(x,w)}{\partial z_j}\frac{\partial z_j}{\partial \xi_i} = \frac{\partial h(x,w)}{\partial z_i} = b_i$$
$$\frac{\partial h(x,w)}{\partial \xi_i \partial \xi_j} = \frac{\partial b_i}{\partial \xi_j} = \sum_k \frac{\partial b_i}{\partial z_k}\frac{\partial z_k}{\partial \xi_j} = \frac{\partial b_i}{\partial z_j} = B_{ij}$$

Using these results, we can write the expansion of $\tilde{E}$ as follows:

$$\begin{aligned}\tilde{E} = &\frac{1}{2}\int\int\int (y - h(x,w))^2 p(y|x)p(x)p(\xi)d\xi dx dt \\ &+ \int\int\int (y - h(x,w))b^\top \xi p(\xi)p(t|x)p(x)d\xi dx dt \\ &+ \frac{1}{2}\int\int\int \xi^\top ((h(x,w) - y)B + bb^\top)\xi p(\xi)p(t|x)p(x)d\xi dx dt\end{aligned}$$

The second term will disappear since $\mathbb{E}[\xi] = 0$ and thus we can write $\tilde{E}$ in the form given in Bishop (5.131) with

$$\Omega = \frac{1}{2}\int\int\int \xi^\top ((h(x,w) - y)B + bb^\top)\xi p(\xi)p(t|x)p(x)d\xi dx dt$$

Again the first term within the parenthesis vanishes to leading order in ξ and we are left with

$$\begin{aligned}\Omega &\sim \frac{1}{2}\int\int \xi^\top (bb^\top)\xi p(\xi)p(x)d\xi dx \\ &= \frac{1}{2}\int\int Tr(\xi\xi^\top bb^\top)p(\xi)p(x)d\xi dx \\ &= \frac{1}{2}\int Tr[\mathbf{I}bb^\top]p(x)dx \\ &= \frac{1}{2}\int b^\top b p(x)dx = \frac{1}{2}\int \|\nabla h(x,w)\|^2 p(x)dx\end{aligned}$$

where we used the fact that $\mathbb{E}[\xi\xi^\top] = \mathbf{I}$.

13 Convolution

This problem is significantly difficult, and requires reading Bishop. Interesting method that is being used in

a lot of computer vision applications. Don't feel you have to understand this.

Consider a neural network, such as the convolutional network discussed in Section 5.5.6 in Bishop, in which multiple weights are constrained to have the same value. Discuss how the standard backpropagation algorithm must be modified in order to ensure that such constraints are satisfied when evaluating the derivatives of an error function with respect to the adjustable parameters in the network.

Solution: The modifications only affect derivatives w.r.t. weights in the convolutional layer. The units within a feature map (indexed by m) have different inputs, but share commen weight vector $w^{(m)}$. Thus, errors $\delta^{(m)}$ from all units within a feature map will contribute to the derivatives of the corresponding weight vector. In this situation, (5.50) in Bishop becomes

$$\frac{\partial E_n}{\partial w_i^{(m)}} = \sum_j \frac{\partial E_n}{\partial a_j^{(m)}} \frac{\partial a_j^{(m)}}{\partial w_i^{(m)}} = \sum_j \delta_j^{(m)} z_{ji}^{(m)}$$

Here $a_j^{(m)}$ denotes the activation of the j-th unit in the m-th feature map, whereas $w_i^{(m)}$ denotes the i-th element of the corresponding feature vector and, finally, $z_{ji}^{(m)}$ denotes the i-th input for the j-th unit in the m-th feature map; the latter may be an actual input or the output of a preceding layer.

Note that $\delta_j^{(m)} = \partial E_n / \partial a_j^{(m)}$ will typically be computed recursively from the δ's of the units in the following layer, using (5.55) in Bishop. If there are layer(s) preceding the convolutional layer, the standard backward propagation equations will apply; the weights in the convolutional layer can be treated as if they were independent parameters, for the purpose of computing the δ's for the preceding layer's units.