CS 229: Assignment 2

Santiago de Buen

October 21, 2020

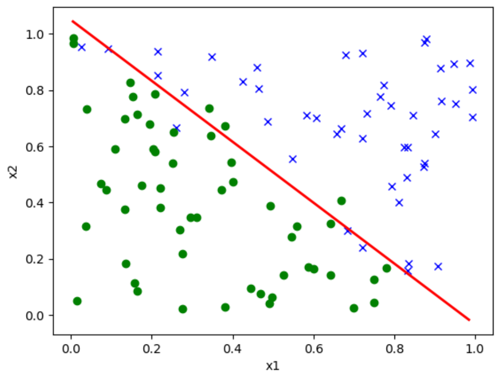
1 Logistic Regression: Training Stability

(a) **Difference in training logistic regression on datasets A and B**

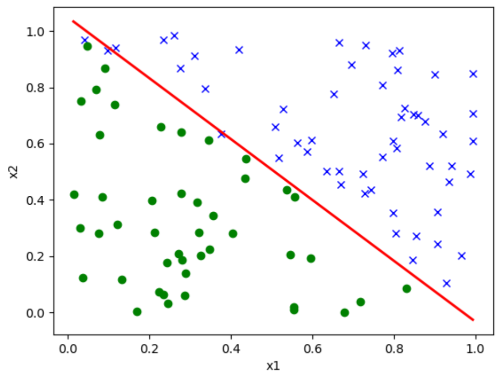
The logistic regression trained on dataset A quickly converges, whereas the regression run on dataset B does not converge, or takes many, many iterations to converge (did not converge after running over a million iterations of gradient ascent).

(b) **Why does this happen?**

The plots below will help us understand the behavior of the logistic regression logarithm on both data sets. Although the algorithm does not converge on Dataset B, we use the decision plot characterized by the parameters after 250,000 iterations.



(a) Dataset A



(b) Dataset B

Figure 1: Decision Boundary Produced by Logistic Regression

Observe that in Dataset B, the examples are almost perfectly separable. All examples above the boundary are classified as positive, and all examples below the boundary are classified as negative. The quasi-perfect separability is causing the lack of convergence.

To corroborate this hypothesis, I turn to a mathematical explanation. If the data is completely separable by a decision boundary κ, then we would like the algorithm to predict:

$$\text{Classifier}\begin{cases} h_\theta(x) \to 1 \text{ if } x > \kappa \\ h_\theta(x) \to 0 \text{ if } x < \kappa \end{cases}$$

Define $x' = x - \kappa$, which we are able to do because it is simply a linear displacement of our original data. Now, $x' > 0$ for positive examples and $x' < 0$ for negative examples.

Positive examples: As $h_\theta(x') \to 1$, $log(h_\theta(x')) \to 0$, which occurs as $\theta \to \infty$ because $x' > 0$

Negative examples: As $h_\theta(x') \to 0$, $log(1 - h_\theta(x')) \to 0$, which occurs as $\theta \to \infty$ because $x' < 0$

Recall the log-likelihood function:

$$\ell(\theta) = \sum_{i=1}^{n} y^{(i)} log(h_\theta(x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))$$

Because the log-likelihood of both positive and negative examples is increasing in θ, the Likelihood function $L(\theta)$ is strictly increasing in θ. We can always find a θ that will make the probabilities more extreme and the likelihood function larger. The likelihood function does not have a maximum, and therefore our algorithm will never truly converge.

Implementing the algorithm on Dataset B, we can see how θ continues to grow with the number of iterations t.

$$t = 10,000 \qquad \theta = \begin{bmatrix} -52.74 & 52.92 & 52.69 \end{bmatrix}$$

$$t = 100,000 \qquad \theta = \begin{bmatrix} -121.21 & 121.48 & 121.24 \end{bmatrix}$$

$$t = 250,000 \qquad \theta = \begin{bmatrix} -165.75 & 166.38 & 165.71 \end{bmatrix}$$

Dataset A is not separable. Let us first discuss the case with positive examples, where for $y^{(i)} = 1$, we have both $x' > 0$ and $x' < 0$. When our predictions are correct, $h_\theta(x^{(i)} \to 1$ and $log(h_\theta(x^{(i)}) \to 0$. When the predictions are incorrect however, which happens because the data is not separable, we have $h_\theta(x^{(i)}) \to 0$ and $log(h_\theta(x^{(i)}) \to -\infty$. The negative case is analogous, where the log-likelihood function goes to 0 when our predictions are correct but to $-\infty$ when they are incorrect. Therefore, the likelihood function will overall go to $-\infty$ if our data is not separable and $\theta \to \infty$.

In conclusion, there must exist a maximum in the function when the data is not separable that makes the function converge.

(c) **Modifications to the algorithm**

(i) **Different constant learning rate**

Would not lead to convergence. The problem in the gradient ascent rule is that $\nabla_\theta(\theta)$ is strictly positive, and no constant learning rate would lead to convergence.

(ii) **Decreasing learning rate over time**

Would lead to *false* convergence. In the update rule, $\theta := \theta + \alpha \nabla_\theta(\theta)$, θ will converge when $\alpha \to 0$ or $\nabla_\theta(\theta) \to 0$. The gradient will never be zero, but the update rule for α means that the term will decrease quickly. By implementing this learning rate, the algorithm converges after 120,000 iterations. At that point, $\alpha = 5.8e^{-17}$, which shows that the convergence is coming from α being extremely low, and not because the function has reached a maximum. Therefore, the algorithm has not truly converged.

(iii) **Linear scaling of the input features**

Would not lead to convergence. Any linear scaling will not change that the data is separable. If the data is separable, we still have the problem that the likelihood is strictly increasing in the parameters. Therefore, the algorithm will not converge.

(iv) **Adding a regularization term**

Would lead to convergence. Adding a regularization term to the optimization function would cause the model to avoid choosing parameters θ that are always increasing. Therefore, the model would choose a small θ such that the decision boundary accurately classifies the examples.

(v) **Adding zero-mean Gaussian noise**

Would lead to convergence. Adding Gaussian noise will cause the data to no longer be separable, which would lead to the case of dataset A, eliminating the convergence problem.

(d) **Are support vector machines vulnerable to datasets like B?**

SVM are not vulnerable to datasets like B, because the SVM chooses the most separable decision boundary, maximizing the distance between the datapoints and the boundary. Although there are many decision boundaries that can separate the data in a dataset like B, the support vector machine chooses a single decision boundary where the separation is maximized.

2 Spam classification

(a) **Implementing spam message processing in code**

The vocabulary size after the pre-processing step is 1,722 words.

(b) **Implementing multinomial event model**

The accuracy of the trained model on the test set is 97.85%

(c) **Most indicative tokens to classifying spam**

The five most indicative tokens are: "claim", "won", "prize", "tone", "urgent!"

(d) **Implementing SVM with radial basis function kernel**

The optimal radius for the SVM model is 0.1, achieving an accuracy of 96.95% on the test set.

(e) **Implementing logistic regression on high-quality features**

The best learning rate is 0.0001, achieving an accuracy of 99.64% on the test set.

3 Kernelizing the Perceptron

(a) **Applying the kernel trick to a high-dimensional feature map**

(i) **How to represent high-dimensional vector $\theta^{(i)}$**

In this version of the perceptron algorithm, we are using a version of stochastic gradient descent where we only make one pass through the entire training set. Lets look at the first iterations of θ.

$$\theta^{(1)} = \theta^{(0)} + \alpha\left[y^{(1)} - g({\theta^{(0)}}^T\phi(x^{(1)}))\right]\phi(x^{(1)})$$

Substituting $\theta^{(0)} = 0$:

$$\theta^{(1)} = \alpha(y^{(1)} - 1)\phi(x^{(1)}) = \beta_1\phi(x^{(1)})$$

Now for $\theta^{(2)}$:

$$\theta^{(2)} = \theta^{(1)} + \alpha\left[y^{(2)} - g({\theta^{(1)}}^T\phi(x^{(2)}))\right]\phi(x^{(2)})$$

$$\theta^{(2)} = \beta_1\phi(x^{(1)}) + \alpha\left[y^{(2)} - g(\beta_1\phi(x^{(1)})^T\phi(x^{(2)}))\right]\phi(x^{(2)})$$

$$\theta^{(2)} = \beta_1\phi(x^{(1)}) + \alpha\left[y^{(2)} - g(\beta_1\langle\phi(x^{(1)}), \phi(x^{(2)})\rangle)\right]\phi(x^{(2)})$$

$$\theta^{(2)} = \beta_1\phi(x^{(1)}) + \alpha\left[y^{(2)} - g(\beta_1 K(x^{(1)}, x^{(2)}))\right]\phi(x^{(2)}) = \beta_1\phi(x^{(1)}) + \beta_2\phi(x^{(2)})$$

where K is the kernel function. Generalizing,

$$\theta^{(i)} = \sum_{j=0}^{i}\beta_j\phi(x^{(j)})$$

where

$$\beta_0 = 0$$

(ii) **Making a prediction on new input** $x^{(i+1)}$
Before making a new prediction, let us explore the form of β_j. To do so, let us start with the first few iterations of β

$$\beta_1 = \alpha(y^{(1)} - 1)$$

$$\beta_2 = \alpha\left[y^{(2)} - g\left(\beta_1\phi(x^{(1)})^T\phi(x^{(2)})\right)\right]$$

$$\beta_3 = \alpha\left[y^{(3)} - g\left((\beta_1\phi(x^{(1)}) + \beta_2\phi(x^{(2)}))^T\phi(x^{(3)})\right)\right]$$

$$\beta_3 = \alpha\left[y^{(3)} - g\left(\beta_1\phi(x^{(1)})^T\phi(x^{(3)}) + \beta_2\phi(x^{(2)})^T\phi(x^{(3)})\right)\right]$$

$$\beta_3 = \alpha\left[y^{(3)} - g\left(\beta_1 K(x^{(1)}, x^{(3)}) + \beta_2 K(x^{(2)}, x^{(3)})\right)\right]$$

Generalizing,

$$\beta_j = \alpha\left[y^{(j)} - g\left(\sum_{k=1}^{j-1}\beta_k K(x^{(k)}, x^{(j)})\right)\right]$$

If we want to make a prediction on a new input $x^{(i+1)}$, we want to find $g(\theta^{(i)^T}\phi(x^{(i+1)}))$. Taking the argument of this function:

$$\theta^{(i)^T}\phi(x^{(i+1)}) = \left[\sum_{j=1}^{i}\beta_j\phi(x^{(j)})^T\right]\phi(x^{(i+1)})$$

$$= \sum_{j=1}^{i}\beta_j\phi(x^{(j)})^T\phi(x^{(i+1)})$$

$$= \sum_{j=1}^{i}\beta_j\langle\phi(x^{(j)}), \phi(x^{(i+1)})\rangle$$

$$= \sum_{j=1}^{i}\beta_j K(x^{(j)}, x^{(i+1)})$$

Therefore, the prediction for a new observation can be efficiently made by computing

$$h_{\theta^{(i)}}(x^{(i+1)}) = g\left(\sum_{j=1}^{i} \beta_j K(x^{(j)}, x^{(i+1)})\right)$$

where

$$\beta_j = \alpha\left[y^{(j)} - g\left(\sum_{k=1}^{j-1} \beta_k K(x^{(k)}, x^{(j)})\right)\right] \tag{1}$$

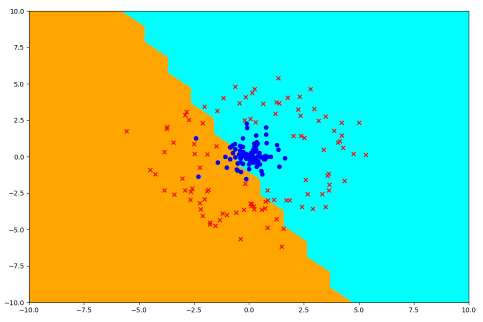
The neat property of the kernel trick is that we never have to compute $\phi(x)$, we only need to compute the kernel in order to make new predictions.

(iii) **Modifying the update rule to perform an update to θ on a new training example**
From the discussion above, we can express the update rule for a new training example as

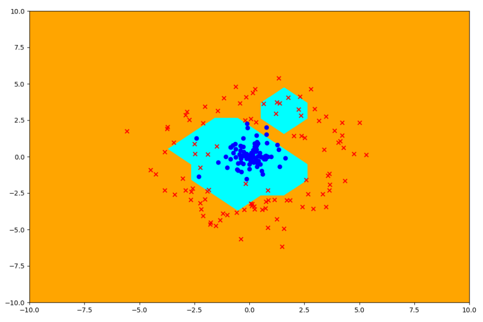
$$\theta^{(i+1)} := \sum_{j=0}^{i+1} \beta_j \phi(x^{(j)})$$

where β_j is defined by (1), which incorporates the training label $y^{(i+1)}$. We never compute θ explicitly, as we can use the kernel method when producing predictions.

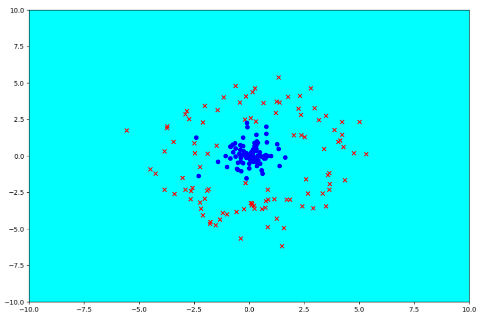
(b) **Implementation of kernel**



(a) Dot-product kernel



(b) Radial basis function kernel



(c) Non-PSD kernel

Figure 2: Results of kernelized perceptron

(c) **Discussing performance of each kernel function**

The **dot-product kernel** is defined as $K(x, z) = x^T z$. In our coding example, each input is a two-dimensional vector. Therefore, we can represent the kernel function as

$$K(x, z) = x_1 z_1 + x_2 z_2$$

which is a simple linear combination of the kernel's inputs. This produces a relatively linear decision boundary, which is shown in the first figure. The test data is clearly not generated by a linear function, which is why this method does not work well.

The **radial basis function kernel** does much better at classifying the data. The function allows for a heavy degree of non-linearity on the original inputs by taking a radial approach. The figure shows how a circular decision boundary classifies the data. The kernel function for our example can be expressed as

$$K(x, z) = exp\left(\frac{-||x - z||_2^2}{2\sigma^2}\right) = exp\left(-\frac{1}{2}((x_1 - z_1)^2 + (x_2 - z_2)^2)\right)$$

Finally, the **non-PSD kernel** does not work at all because it is not a positive semidefinite matrix. If the kernel function is not PSD, it cannot be a valid function. As we can see in the third figure, there is no decision boundary, as every data point is predicted to be in the blue region.

4 Bayesian Interpretation of Regularization

(a) **Deriving maximum a posteriori distribution of the parameters**

The *maximum a posteriori estimation* (MAP) is estimated as

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta|x, y)$$

From Bayes theorem, we have

$$p(\theta|x, y) = \frac{p(x, y|\theta)p(\theta)}{p(x, y)} \tag{2}$$

From chain rule,

$$p(x, y|\theta) = p(y|\theta, x)p(x|\theta)$$

From the law of total probability,

$$p(x, y) = p(y|x)p(x)$$

Substituting in (2):

$$p(\theta|x, y) = \frac{p(y|\theta, x) \cdot p(x|\theta) \cdot p(\theta)}{p(y|x) \cdot p(x)} \tag{3}$$

Using Bayes theorem again,

$$p(x|\theta) = \frac{p(\theta|x) \cdot p(x)}{p(\theta)}$$

Substituting in (3):

$$p(\theta|x, y) = \frac{p(y|\theta, x) \cdot p(\theta)}{p(y|x) \cdot p(x)} \left(\frac{p(\theta|x) \cdot p(x)}{p(\theta)} \right)$$

$$= \frac{p(y|\theta, x) \cdot p(\theta|x)}{p(y|x)}$$

Using assumption that $p(\theta) = p(\theta|x)$:

$$p(\theta|x, y) = \frac{p(y|\theta, x) \cdot p(\theta)}{p(y|x)} \tag{4}$$

Because we are maximizing over θ, we can disregard the term in the denominator

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(y|x, \theta) \cdot p(\theta)$$

(b) **Regularization and MAP estimation**

We want to show that

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta|x,y) \tag{5}$$

is equivalent to the following expression if the prior is distributed as $\theta \sim \mathcal{N}(0, \eta^2 I)$

$$\theta_{\text{MAP}} = \arg\min_{\theta} -\log p(y|x,\theta) + \lambda||\theta||_2^2$$

We can start by using the result from (a) to rewrite (5):

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(y|x,\theta) \cdot p(\theta) \tag{6}$$

We know $p(\theta)$ is distributed as a multivariate normal:

$$p(\theta) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(\theta-\mu)^T\Sigma^{-1}(\theta-\mu)\right)$$

Replacing with $\theta \sim \mathcal{N}(0, \eta^2 I)$:

$$p(\theta) = \frac{1}{(2\pi)^{\frac{n}{2}}|\eta^2 I|^{\frac{1}{2}}} exp\left(-\frac{1}{2}\theta^T(\eta^2 I)^{-1}\theta\right)$$

$$p(\theta) = \frac{1}{(2\pi)^{\frac{n}{2}}|\eta^2 I|^{\frac{1}{2}}} exp\left(-\frac{1}{2\eta^2}\theta^T\theta\right)$$

Going back to (6), we can apply logarithms given that it is a monotonic transformation

$$\begin{aligned}
\theta_{\text{MAP}} &= \arg\max_{\theta} \log p(y|x,\theta) + \log p(\theta) \\
&= \arg\max_{\theta} \log p(y|x,\theta) + \log\left[\frac{1}{(2\pi)^{\frac{n}{2}}|\eta^2 I|^{\frac{1}{2}}} exp\left(-\frac{1}{2\eta^2}\theta^T\theta\right)\right] \\
&= \arg\max_{\theta} \log p(y|x,\theta) + \log\left(\frac{1}{(2\pi)^{\frac{n}{2}}|\eta^2 I|^{\frac{1}{2}}}\right) - \left(\frac{1}{2\eta^2}\theta^T\theta\right) \\
&= \arg\max_{\theta} \log p(y|x,\theta) + \log\left(\frac{1}{(2\pi)^{\frac{n}{2}}|\eta^2 I|^{\frac{1}{2}}}\right) - \left(\frac{1}{2\eta^2}||\theta||_2^2\right)
\end{aligned}$$

The second term is constant and does not depend on θ, so we can ignore it. Therefore, we can write θ_{MAP} as:

$$\theta_{\text{MAP}} = \arg\max_{\theta} \log p(y|x,\theta) - \left(\frac{1}{2\eta^2}||\theta||_2^2\right)$$

Which is equivalent to

$$\theta_{\text{MAP}} = \arg\min_{\theta} -\log p(y|x,\theta) + \lambda||\theta||_2^2 \tag{7}$$

where $\lambda = \frac{1}{2\eta^2}$.

(c) **Linear Regression Model with Gaussian prior**

Our objective is to find a closed form expression for θ_{MAP}. Because we are still assuming a Gaussian prior over the distribution of θ, we can use the result from (7). Focusing on $p(y|x,\theta)$ first:

$$p(y^{(i)}|x^{(i)},\theta) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

$$p(y|x,\theta) = \prod_{i=1}^{n}\frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

Taking the logarithm of the probability,

$$\log p(y|x,\theta) = \sum_{i=1}^{n} log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \sum_{i=1}^{n}\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)$$

Substituting into (7),

$$\theta_{\text{MAP}} = \arg\min_{\theta} -\sum_{i=1}^{n} log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) + \sum_{i=1}^{n}\left(\frac{-(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) + \lambda||\theta||_2^2$$

We can ignore the first term as it does not depend on θ, and rewriting using vector notation:

$$\theta_{\text{MAP}} = \arg\min_{\theta} \frac{1}{2\sigma^2}(\vec{y} - \theta^T X)^T(\vec{y} - \theta^T X) + \lambda\theta^T\theta$$

Taking the derivative and setting equal to zero to find the minimum:

$$\frac{1}{2\sigma^2}(2X^T X\theta - 2X^T\vec{y}) + 2\lambda\theta = 0$$

$$\frac{1}{\sigma^2}(X^T X\theta - X^T\vec{y}) + 2\lambda\theta = 0$$

$$X^T X\theta - X^T\vec{y} + 2\sigma^2\lambda\theta = 0$$

$$(X^T X + 2\sigma^2\lambda I)\theta = X^T\vec{y}$$

$$\theta = (X^T X + 2\sigma^2\lambda I)^{-1}X^T\vec{y}$$

Substituting $\lambda = \frac{1}{2\eta^2}$

$$\theta_{\text{MAP}} = \left(X^T X + \frac{\sigma^2}{\eta^2}I\right)^{-1} X^T\vec{y}$$

which is the closed form solution for θ_{MAP}.

(d) **Coding Problem: Double Descent**

The coding problem produces the following figure.

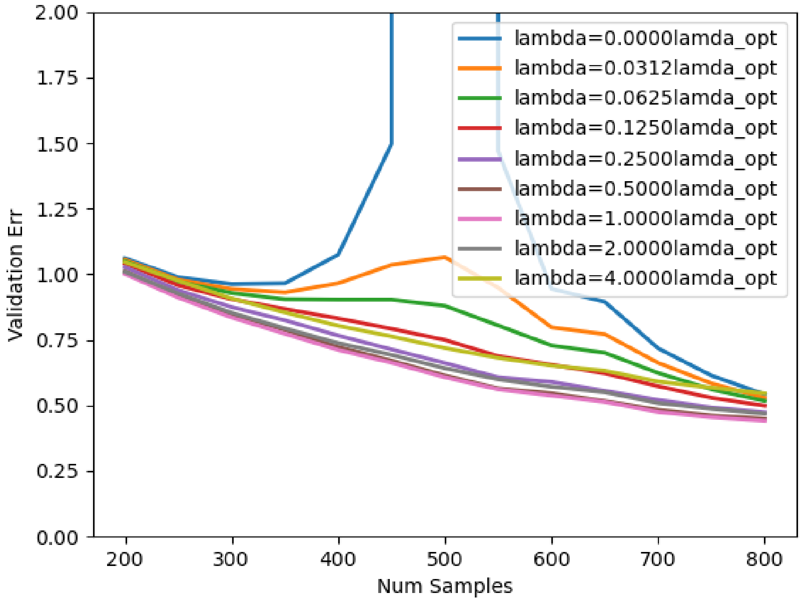


Figure 3: Double descent and regularization strength

When $\lambda = \lambda_{opt}$, we do not have a double descent problem. With regularization strength that is not close to the optimal, we can see how the validation loss does not monotonically decrease as we increase the number of training samples.

(e) **Linear Regression Model with Laplace prior**

We know the maximum a posteriori estimation fulfills

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta|x, y)$$

As in (b), we can express the probability as

$$p(\theta|x, y) = p(y|x, \theta) \cdot p(\theta|x)$$

We can apply a logarithmic transformation

$$\theta_{\text{MAP}} = \arg\max_{\theta} \log(p(y|x, \theta)) + \log(p(\theta|x)) \tag{8}$$

Using the Laplace distribution, we can write $p(\theta|x)$ as:

$$p(\theta|x) = \frac{1}{2b}\exp\left(-\frac{|\theta|}{b}\right)$$

Taking the logarithm,

$$\log p(\theta|x) = log\left(\frac{1}{2b}\right) - \frac{|\theta|}{b}$$

From (c), we know that

$$\log p(y|x,\theta) = n \cdot log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{1}{2\sigma^2}(\vec{y} - \theta^T X)^T(\vec{y} - \theta^T X)$$

Substituting in (8):

$$\theta_{\text{MAP}} = \arg\max_\theta n \cdot log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{1}{2\sigma^2}(\vec{y} - \theta^T X)^T(\vec{y} - \theta^T X) + log\left(\frac{1}{2b}\right) - \frac{|\theta|}{b}$$

Keeping only terms that depend on θ:

$$\theta_{\text{MAP}} = \arg\max_\theta -\frac{1}{2\sigma^2}(\vec{y} - \theta^T X)^T(\vec{y} - \theta^T X) - \frac{|\theta|}{b}$$

$$\theta_{\text{MAP}} = \arg\min_\theta \frac{1}{2\sigma^2}||\vec{y} - \theta^T X||_2^2 + \frac{|\theta|}{b}$$

$$\theta_{\text{MAP}} = \arg\min_\theta ||\vec{y} - \theta^T X||_2^2 + \frac{2\sigma^2}{b}||\theta||_1$$

$$\theta_{\text{MAP}} = \arg\min_\theta ||\vec{y} - \theta^T X||_2^2 + \gamma||\theta||_1$$

where $\gamma = \frac{2\sigma^2}{b}$. This is equivalent to the solution of linear regression with L_1 regularization.

5 A Simple Neural Network

(a) **Gradient Descent Update Using Sigmoid Activation Function**

We will use the sigmoid activation function.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Our simple two-layer neural network can be described by the following steps (note intercepts are included within the weight matrices)

$$\begin{aligned} z &= W^{[1]}x \\ h &= \frac{1}{1+e^{-z}} \\ o &= W^{[2]}h \\ J &= (o-y)^2 \\ L &= \frac{1}{n}\sum_{i=1}^{n} J^{(i)} \end{aligned}$$

Using backpropagation, we can find the partial derivatives with respect to the model's parameters using the chain rule.

$$\frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial J}\cdot\frac{\partial J}{\partial o}\cdot\frac{\partial o}{\partial W^{[2]}}$$

$$\frac{\partial L}{\partial W^{[1]}} = \frac{\partial L}{\partial J}\cdot\frac{\partial J}{\partial o}\cdot\frac{\partial o}{\partial h}\cdot\frac{\partial h}{\partial z}\cdot\frac{\partial z}{\partial W^{[1]}}$$

Finding these derivatives, we have

$$\frac{\partial L}{\partial J} = \frac{1}{n}\cdot n = 1$$

$$\frac{\partial J}{\partial o} = 2(o-y)$$

$$\frac{\partial o}{\partial W^{[2]}} = h = \frac{1}{1+e^{-W^{[1]}x}}$$

$$\frac{\partial o}{\partial h} = W^{[2]}$$

$$\frac{\partial h}{\partial z} = h(1-h) = \frac{1}{1+e^{-W^{[1]}x}}\left(1-\frac{1}{1+e^{-W^{[1]}x}}\right)$$

$$\frac{\partial z}{\partial W^{[1]}} = x$$

Putting it all together,

$$\frac{\partial L}{\partial W^{[2]}} = 2(o-y)\left(\frac{1}{1+e^{-W^{[1]}x}}\right) \tag{9}$$

$$\frac{\partial L}{\partial W^{[1]}} = 2(o-y)W^{[2]}x\left(\frac{e^{-W^{[1]}x}}{(1+e^{-W^{[1]}x})^2}\right) \tag{10}$$

Finally, we can express the gradient descent rule for each set of weights as

$$W^{[1]} := W^{[1]} - \alpha\frac{\partial L}{\partial W^{[1]}}$$
$$W^{[2]} := W^{[2]} - \alpha\frac{\partial L}{\partial W^{[2]}}$$

Note that we have vectorized the notation, but it is easy to express the update for $W_{i,j}$ in terms of $x^{(i)}$, $o^{(i)}$, $y^{(i)}$. For example, the prompt explicitly asks for the update rule of $W^{[1]}_{1,2}$:

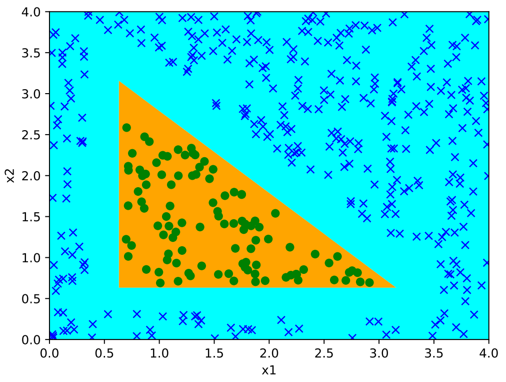
$$W^{[1]}_{1,2} := W^{[1]}_{1,2} - \alpha\frac{\partial L}{\partial W^{[1]}_{1,2}}$$

where

$$\frac{\partial L}{\partial W^{[1]}_{1,2}} = 2(o^{(i)}-y^{(i)})W^{[2]}_{2}x^{(i)}\left(\frac{e^{-W^{[1]}_{1,2}x^{(i)}}}{(1+e^{-W^{[1]}_{1,2}x^{(i)}})^2}\right)$$

(b) **Using step activation function**

It is possible to have a set of weights that allow the neural network to classify this dataset with 100% accuracy. Figure (4) shows the plot with perfect accuracy.



(a) Neural Network Decision Boundary

Figure 4: Prediction using step function activation function

(c) **Using linear activation function**

If the network's activation function is linear, we can collapse the network into a single linear function over the original input. In this example, our network before taking the step activation function of o, would look like:

$$h = W^{[1]}x$$
$$o = W^{[2]}(W^{[1]}x) = \tilde{W}x$$

where $\tilde{W} = W^{[2]}W^{[1]}$. Therefore, we would be constrained to a linear decision boundary over the dataset. Looking at the data plot, it is clear that we would be unable to classify with 100% accuracy using a linear decision boundary.

6 Batch Normalization

(a) **Calculating gradient of $\mathcal{L}$ with respect to β, γ, and $\hat{x}^{(i)}$**

The batch normalization model is described by the following equations

$$\mathcal{B} = (x^{(1)}, ..., x^{(m)})$$

$$\mu^{(\mathcal{B})} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \tag{11}$$

$$v_j^{(\mathcal{B})} = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j^{(\mathcal{B})})^2 \tag{12}$$

$$\hat{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j^{(\mathcal{B})}}{\sqrt{v_j^{(\mathcal{B})}}} \tag{13}$$

$$y^{(i)} = \gamma \odot \hat{x}^{(i)} + \beta \tag{14}$$

$$\mathcal{L} = \mathcal{L}(y^{(i)}, ..., y^{(m)})$$

We want to calculate the gradients of $\mathcal{L}$ with respect to β, γ and $\hat{x}^{(i)}$. Note that β and γ are common across training examples, which is why the partial derivatives sum over the examples in the batch.

$$\frac{\partial \mathcal{L}}{\partial \beta} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial y^{(i)}}$$

$$\frac{\partial \mathcal{L}}{\partial \gamma} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial y^{(i)}} \cdot \hat{x}^{(i)}$$

$$\frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} = \frac{\partial \mathcal{L}}{\partial y^{(i)}} \cdot \frac{\partial y^{(i)}}{\partial \hat{x}^{(i)}} = \frac{\partial \mathcal{L}}{\partial y^{(i)}} \cdot \gamma$$

(b) **Calculating gradient of $\mathcal{L}$ with respect to $v^{(\mathcal{B})}$**

Like before, note that the variance in each dimension is common across examples in the batch, which is why the partial derivatives sum over the examples of the batch.

$$\frac{\partial \mathcal{L}}{\partial v^{(\mathcal{B})}} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial y^{(i)}} \cdot \frac{\partial y^{(i)}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial v^{(\mathcal{B})}} = \sum_{i=1}^{m} \left(\frac{\partial \mathcal{L}}{\partial y^{(i)}} \cdot \gamma \right) \left(-\frac{1}{2} \cdot \frac{(x^{(i)} - \mu^{(\mathcal{B})})}{(v^{(\mathcal{B})})^{\frac{3}{2}}} \right)$$

(c) **Calculating gradient of $\mathcal{L}$ with respect to $\mu^{(\mathcal{B})}$**

The gradient with respect to $\mu^{(\mathcal{B})}$ depends both on $\mu^{(\mathcal{B})}$ directly and $v^{(\mathcal{B})}$ indirectly. The derivative must take into account both parts.

$$\frac{\partial \mathcal{L}}{\partial \mu^{(\mathcal{B})}} = \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial y^{(i)}} \cdot \frac{\partial y^{(i)}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial v^{(\mathcal{B})}} \cdot \frac{\partial v^{(\mathcal{B})}}{\partial \mu^{(\mathcal{B})}} + \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial y^{(i)}} \cdot \frac{\partial y^{(i)}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial \mu^{(\mathcal{B})}}$$

$$= -\frac{2}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial v^{(\mathcal{B})}} (x^{(i)} - \mu^{(\mathcal{B})}) + \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \left(\frac{1}{(v^{(\mathcal{B})})^{\frac{1}{2}}} \right) (-1)$$

$$= -2 \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial v^{(\mathcal{B})}} \left[\frac{1}{m} \sum_{i=1}^{m} x^{(i)} - \frac{1}{m} \sum_{i=1}^{m} \mu^{(\mathcal{B})} \right] - \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \left(\frac{1}{(v^{(\mathcal{B})})^{\frac{1}{2}}} \right)$$

$$= -2 \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial v^{(\mathcal{B})}} \left[\mu^{(\mathcal{B})} - \mu^{(\mathcal{B})} \frac{m}{m} \right] - \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \left(\frac{1}{(v^{(\mathcal{B})})^{\frac{1}{2}}} \right)$$

$$= \frac{-1}{\sqrt{v^{(\mathcal{B})}}} \odot \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}}$$

(d) **Calculating gradient of $\mathcal{L}$ with respect to $x^{(i)}$**

The gradient with respect to $x^{(i)}$ has four components: (1) through $v^{(\mathcal{B})}$ and then $\mu^{(\mathcal{B})}$, (2) through $v^{(\mathcal{B})}$ directly, (3) through $\mu^{(\mathcal{B})}$ directly, and (4) through $\hat{x}^{(i)}$ directly.

$$\frac{\partial \mathcal{L}}{\partial x^{(i)}} = \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial v^{(\mathcal{B})}} \cdot \frac{\partial v^{(\mathcal{B})}}{\partial \mu^{(\mathcal{B})}} \cdot \frac{\partial \mu^{(\mathcal{B})}}{\partial x^{(i)}} + \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial v^{(\mathcal{B})}} \cdot \frac{\partial v^{(\mathcal{B})}}{\partial x^{(i)}} + \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial \mu^{(\mathcal{B})}} \cdot \frac{\partial \mu^{(\mathcal{B})}}{\partial x^{(i)}} + \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial x^{(i)}} \quad (15)$$

From (c), we know that the first term is 0:

$$\frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial v^{(\mathcal{B})}} \cdot \frac{\partial v^{(\mathcal{B})}}{\partial \mu^{(\mathcal{B})}} = 0$$

As for the other terms:

$$\frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial v^{(\mathcal{B})}} \cdot \frac{\partial v^{(\mathcal{B})}}{\partial x^{(i)}} = \frac{2(x^{(i)} - \mu^{(\mathcal{B})})}{m} \odot \frac{\partial \mathcal{L}}{\partial v^{(\mathcal{B})}}$$

$$\frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial \mu^{(\mathcal{B})}} \cdot \frac{\partial \mu^{(\mathcal{B})}}{\partial x^{(i)}} = \frac{1}{m} \cdot \frac{\partial \mathcal{L}}{\partial \mu^{(\mathcal{B})}}$$

$$\frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} \cdot \frac{\partial \hat{x}^{(i)}}{\partial x^{(i)}} = \frac{1}{\sqrt{v^{(\mathcal{B})}}} \odot \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}}$$

Putting it all together,

$$\frac{\partial \mathcal{L}}{\partial x^{(i)}} = \frac{1}{\sqrt{v^{(\mathcal{B})}}} \odot \frac{\partial \mathcal{L}}{\partial \hat{x}^{(i)}} + \frac{2(x^{(i)} - \mu^{(\mathcal{B})})}{m} \odot \frac{\partial \mathcal{L}}{\partial v^{(\mathcal{B})}} + \frac{1}{m} \cdot \frac{\partial \mathcal{L}}{\partial \mu^{(\mathcal{B})}} \tag{16}$$

7 Favorite Book

Brave New World by Aldous Huxley.