Author: Benjamin Smidt
Created: September 9, 2022
Last Updated: September 9, 2022

Assignment 2: Multiclass SVM

Note to the reader. This is my work for assignment two of Michigan's course EECS 498: Deep Learning for Computer Vision. This document is thoroughly researched but may not be perfect. If there's a typo or a correction needs to be made, feel free to email me at benjamin.smidt@utexas.edu so I can fix it. Thank you! I hope you find this document helpful.

1 Mathematics

1.1 Loss

Resources

- CS 231N: SVM's and Softmax
- StatQuest with Josh Starmer

Let's begin by defining some variables. X is an NxD matrix containing N examples and D dimension for each example. W is a DxC matrix containing weights for a given example x_i in X where C indicates the number of classes we want to be able to classify x_i into. From here we can look at the loss function for Multiclass Support Vector Machines.

$$L_i = \sum_{j \neq y_i}^{C} max(0, s_j - s_{y_i} + \Delta) \tag{1}$$

$$s_j = f(x_i, W)_j = (Wx_i)_j \tag{2}$$

s_j is the "score" for class j of a single example x_i computed by taking the dot product of weight vector w_j (a column of W) and a single example x_i. s_{y_i} is the s_j value that is the correct classification for example x_i. L_i is the loss for a single example x_i and Δ is a fixed margin for which SVM "wants" the correct class score s_{y_i} to be greater than all the other class scores (s_j where $j \neq y_i$).

This loss is called *hinge loss* because the loss is 0 as long as $(s_{y_i} - s_j) \geq \Delta$. SVM doesn't care how large $s_{y_i} - s_j$ is, only that is at least as large as Δ. It could be infinite or equal to Δ, and it will treat those scores the same since they both have a loss of 0. You may also see the squared hinge loss used, which is of the form $max(0, -)^2$ instead of $max(0, -)$. Finally, another (possibly helpful) way to write the loss function is:

$$L_i = \sum_{j \neq y_i}^{C} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \tag{3}$$

1.2 Gradient

Resources

- CS 231N: Optimization

With the loss function understood, let's turn our attention to how we can compute the gradient for SVM's loss function. The loss function for one example is given as:

$$L_i = \sum_{j \neq y_i}^{C} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \tag{4}$$

There's a few important details to note before we begin deriving. The first is the $max()$ function. Although slightly scary at first, all we need to do is split this function into two cases, when $w_j^T x_i - w_{y_i}^T x_i + \Delta > 0$ and when $w_j^T x_i - w_{y_i}^T x_i + \Delta < 0$.

$$L_{i,j} = \begin{cases} 0 \text{ if } w_{y_i}^T x_i - w_j^T x_i > \Delta \\ w_j^T x_i - w_{y_i}^T x_i + \Delta \text{ if } w_{y_i}^T x_i - w_j^T x_i < \Delta \end{cases} \tag{5}$$

Of course, when $L_{i,j} = 0$ the gradient is 0. We'll later use the indicator function to put our gradient in a single, succinct form when $L_{i,j} = 0$. For the second case $w_{y_i}^T x_i - w_j^T x_i < \Delta$ we have two different cases: $j = y_i$ and $j \neq y_i$. These derivatives are pretty easy so let's just go ahead and compute both real quick. We'll start with the derivative with respect to w_{y_i}. Remember, these derivatives only apply when $w_{y_i}^T x_i - w_j^T x_i < \Delta$.

$$\frac{\partial L_i}{\partial w_{y_i}} = \frac{\partial L_i}{\partial w_{y_i}} \sum_{j \neq y_i}^{C} w_j^T x_i - w_{y_i}^T x_i + \Delta \tag{6}$$

$$\frac{\partial L_i}{\partial w_{y_i}} = \sum_{j \neq y_i}^{C} -x_i \tag{7}$$

$$\frac{\partial L_i}{\partial w_j} = \frac{\partial L_i}{\partial w_j} \sum_{j \neq y_i}^{C} w_j^T x_i - w_{y_i}^T x_i + \Delta \tag{8}$$

$$\frac{\partial L_i}{\partial w_j} = x_i \tag{9}$$

We have all our derivatives but these only apply when $w_{y_i}^T x_i - w_j^T x_i < \Delta$. To write them succinctly we use indicator functions

$$\nabla_{w_{y_i}} L_i = -\sum_{j \neq y_i}^{C} \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i \tag{10}$$

$$\nabla_{w_j} L_i = \sum_{j \neq y_i}^{C} \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i \tag{11}$$

If you're unfamiliar with indicator functions it's quite alright. They're very simple. If whatever is inside the function is true ($w_{y_i}^T x_i - w_j^T x_i < \Delta$ for us) then the function returns a 1. We can then multiply that by $-x_i$ or x_i to get our gradients for both cases. If whatever is in the function isn't true, ($w_{y_i}^T x_i - w_j^T x_i > \Delta$ for us), then the function returns a 0 which is the gradient we need when our $max()$ function returns a 0 to us. As you can see this is a concise (and elegant!) method to write down our gradient.

1.3 Regularization

Support Vector Machines have an important bug that is crucial to understand. Recall the loss function is defined as follows:

$$L_i = \sum_{j \neq y_i}^{C} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \tag{12}$$

The geometric interpretation of this loss function is that the SVM algorithm tries to find a single hyperplane (generalization of a dividing line for any number of dimensions) for each class that divides the correct classifications from every other classification for all examples in the dataset. If you haven't encountered this interpretation before, I highly suggest you watch this video. Many times this is impossible so the SVM algorithm opts for the hyperplane that simply minimizes the magnitude of misclassifications (which is the loss L_i).

It turns out that the weight vector w_j, which can be interpreted as the hyperplane dividing the correct classifications from all other classifications, is unique as long the data isn't linearly separable. Said another way, the hyperplane dividing correct and incorrect classifications is unique as long as it is impossible to achieve 0 loss. As you may be able to guess, this is not the case when the dataset is linearly separable (0 loss can be achieved).

If 0 loss can be achieved, then we can scale w_j by any constant λ and still achieve 0 loss on the dataset. This is because scaling w_j, the hyperplane, doesn't change the hyperplane at all. Think of a line (a one dimensional hyperplane separating data in two dimensions). If we scale each dimension describing the line (in the case both the "rise" and the "run") by a constant, the slope doesn't change ($\frac{3}{4}x = \frac{6}{8}x$).

What does change for the SVM algorithm is the magnitude of the scores computed using those scaled weights. And since SVM's use hinge loss, if 0 loss can be achieved, then the SVM algorithm can (and will) arbitrarily scale all scores s_j (since they are less than zero no matter their magnitude) so that they are as far from the margin Δ as possible. Again, the SVM algorithm won't scale w_j when the dataset isn't linearly separable because it would also scale the incorrect values to be more incorrect while not offsetting that effect with "less loss" for correct values since hinge loss prevents the loss from being less than 0.

To fix this issue we introduce regularization, a term we tack onto the end of the loss function that makes the loss function favor small weights. This term is known as the *regularization loss* while our original loss function terms are known as the *data loss*. Here's an example of L2 regularization which uses the L2 norm.

$$L = 1/N \sum_{i}^{N} \sum_{j \neq y_i}^{C} max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) + \lambda \sum_{k}^{D} \sum_{l}^{C} W_{k,l}^2 \tag{13}$$

Regularization only makes sense in the context of a dataset, not a single example, so we use L instead of L_i. The first term, our data loss term, is simply indicates the average of all the losses L_i, so nothing new (just different notation). The second term, our regularization loss, is just the sum every squared value in the weight matrix multiplied by some constant λ. λ is there for us to be able to tell the SVM algorithm how important it is for it to keep the values in the weight matrix small.

Again, regularization just keeps the SVM algorithm from arbitrarily scaling linearly seperable datasets. It does have other nice properties and there are different types of regularization. For more details on this, see the References section.

2 Programming

Assignment 2: Linear Classifiers

2.1 SVM Loss/Gradient Naive

This function is pretty straightforward. Just follow the math and you should be able to code it yourself if not follow along at least.

2.2 SVM Loss/Gradient Vectorized

Vectorizing code can certainly be difficult but the loss isn't too difficult for this problem. I matrix multiplied W (DxC) and X (NxD) to get the *scores* matrix (CxD). I then used vector y (Nx1), the range function, and integer indexing to grab the scores of the correct class and stored them in $correct-class-score$. Using broadcasting it's easy to compute the loss for every score, add delta, and store it in $loss-mat$. Use integer indexing to set all correct scores values to 0 (since not included in the loss) and set any value less than 0 equal to 0 ($max()$ function). Finally, just average all the losses in the matrix and add L2 regularization.

The gradient isn't difficult either in this case. We use boolean tensor indexing to set all positive values in the loss matrix equal to 1. We then sum over each training example and multiply by -1 to get the constant to multiply by x_i for each example and insert that value into the position corresponding to the correct score for every example in the loss matrix. Finally, we can get the sum of all the gradients by matrix multiplying $loss - mat$ and X using broadcasting, get the average by dividing by N, and add our regularization.

2.3 Sample Batch

This is a pretty basic function that generates a random sample. We just use torch.randint to get indices to sample from, grab those indices, and return the sample batch.

2.4 Train Linear Classifier

This function is fairly straightforward, we're just bringing all our hard work together to peform gradient descent on whichever linear classifer we're using (SVM for this part of the assignment).

2.5 Predict Linear Classifier

Since we have our weights, we just compute the scores with XW and return the indices with the largest score for each example

2.6 SVM Get Search Parameters

This function just returns different parameter combinations in seach of the best one.

2.7 Test One Parameter Set

As you can guess, this function tests a single instance of a parameters. We train the model using the data from our data-dict (data dictionary). This class just reuses all our hardwork. We predict, compute training accuracy, validation accuracy, and return.

3 References

1. CS 231N: SVM's and Softmax

2. StatQuest with Josh Starmer