CS231n- Lecture 3

February 15, 2017

1 Optimization

1.1 Multiclass SVM loss

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label, and using the shorthand for the scores vector: $s = f(x_i, W)$
The SVM Loss has the form

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

cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	10.9

Q: What if the sum was instead over all class?(i.e including $j = y_i$)
A: Score is just being inflated by one as $j = y_i$

Q: What if mean was used instead of sum?
A: Pointless as it is only a difference of $c_i * L_i$

Q: What if instead you use $L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)^2$
A: This changes the loss function.

Q: Min/Max values?
A: Min=0. Max= Infinite

Q: Usually at initialization W are small numbers so all s = 0. What is the loss?
A: Loss becomes $number\ of\ classes - 1$. This serves as a good sanity check.

Example numpy code for:

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

```
def L_i_vectorized(x, y, W):
        scores= W.dot(x)
        margins=np.maximum(0, scores - scores[y] +  1)
        margins[y] = 0
        loss_i = np.sum(margins)
        return loss_i
```

We now have

$f(x, W) = Wx$

$L = \frac{1}{N} \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$

Suppose we found a W such that L=0. Is this W unique?
No. If we double W. Loss is the same. Same goes if you multiply it with a number $n : n >= 1$
So there is a huge subspace of W for which this is optimal.
Weight Regularisation
$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} max(0, f(x_i; W)_j - f(x_i, W)_{y_i} + 1) + \lambda R(W)$
In Common use:
L2 Regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$
L1 Regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$
Elastic Net(L1+ L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$
Max norm regularization(will see later)
Dropout(Will see later)
L2 is most popular. λ indicates the regularisation strength.
$x = [1, 1, 1, 1]$
$w_1 = [1, 0, 0, 0]$
$w_2 = [0.25, 0.25, 0.25, 0.25]$
Although $w_1^T x = w_2^T x = 1$, we get w_2 and it tends to be better because we take into consideration all the xs when we use the L2 Norm is one way to consider and it generally tends to be better in general, another is we are taking weights which are lower all together

1.2 Softmax Classifier (Multinomial Logistic Regression)

scores = unnormalized log probabilities of the classes.

$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$ where $s = f(x_i; W)$
The exponential function is the softmax function

We want to maximize the log likelihood or(for a loss function we want to minimize the negative log likelihood of the correct class):
$L_i = -logP(Y = y_i|X = x_i)$

in Summarry: $L_i = -log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$

Suppose for a picture of a cat

cat	3.2		24.5	0.13 -> L_i = - log(0.13)=0.89
car	5.1	exp	->164.0	norm->0.87
frog	-1.7		0.18	0.00

Q:What is the min/max possible loss L_i ?
A: Min is correct class gets probability 1 => 0 . Max is correct class gets close to 0 probability implies infinity

Q: Usually at initialization W are small numbers, so all s =0. What is the loss?
A: $-log(1/N_{classes})$ Softmax vs SVM: Assume scores:
$[10, -2, 3]$
$[10, 9, 9]$
$[10, -100, -100]$
and $y_i = 0$
A: SVM doesn't care that much if you move it around since loss function will be same. but softmax still expresses preference for more negative values for incorrect labels.

1.3 Optimization

1.3.1 Random Selection(Aka the worst strategy every)

Basically choosing random weights and then selecting one that leads to lowest lost
For CIFAR 10 we get 15.5% accuracy for this piece of shit idea.

1.3.2 Follow the slope

In 1-Dimension, the derivative of a function: $\frac{df(x)}{dx} = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$
In multiple dimensions, the gradient is the vector of (partial derivatives).

Disadvantages:
Approximate
Very slow because it has to be done for each $W_{i,j}$
In practice we use Analytical gradient generally using Calculus. We might check after that using numerical gradient.

1.3.3 Mini-Batch Gradient Descent

We only sample a very few points of training data to compute gradient.
Faster but more prone to noise. We can however take many more steps
Common mini-batch sizes are 32/64/128 examples.
Learning rate in all these Descents are quite important hyperparameters. We generally want it to be just right. Too big or too small and you're screwed.
Parameter Update is generally done normally- Stochastic Gradient Descent Method. $weights+ = -step\_size * weights\_grad$
More advanced methods are Momentum(gives the idea of velocity and we accumulate this to go faster) Adagrad, RMSProp, Adaboost etc are much faster than SGDs
http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html for better visualizations of Optimizers.
Features extraction used to be very complex statistical models before (hand - engineering).
Now we eliminate this and have a single differentiable blob of combinations of layers to solve this using NN architecture.