6.883: Online Methods in Machine Learning

Alexander Rakhlin

LECTURE 6

0.1 Logistic regression

Consider binary classification. When we talked about half-spaces, the prediction of any hypothesis w was given by

$$\mathrm{sign}(\langle w, x\rangle).$$

As $\langle w, x\rangle$ changes sign, the prediction suddenly jumps from + to −. A useful way to produce a more smooth decision is by first calculating

$$\frac{1}{1+\exp\{-\langle w, x\rangle\}} \tag{1}$$

and using this value (which is between 0 and 1) as the confidence in the choice of the class +1 vs −1. If $\langle w, x\rangle$ is close to zero, the value is around 1/2. As the inner product becomes large, the value approaches 0 or 1, indicating a more confident choice.

The confidence is often interpreted as probability assigned by the forecaster to the two events (class 1 and class −1). A common way to evaluate probability is via negative logarithmic loss of probability assigned by the forecaster to the realized outcome. That is, if $y = 1$, the prediction is more correct if (1) is close to 1, but should be penalized if it is close to 0. Hence, if $y = 1$, the hyperplane w will incur a cost of

$$-\log\left(\frac{1}{1+\exp\{-\langle w, x\rangle\}}\right) = \log\left(1+\exp\{-\langle w, x\rangle\}\right) \tag{2}$$

and if $y = -1$, the loss is

$$-\log\left(1-\frac{1}{1+\exp\{-\langle w, x\rangle\}}\right) = \log\left(1+\exp\{\langle w, x\rangle\}\right). \tag{3}$$

The two cases can be combined by simply writing the logistic loss function

$$\ell(w,(x,y)) = \log\left(1+\exp\{-y\langle w, x\rangle\}\right). \tag{4}$$

0.2 Multiclass logistic regression (softmax)

If $y \in \{1,\ldots,k\}$, the idea is again to produce k scores that measure the affinity of $x \in \mathbb{R}^d$ to each of the classes. Let us only concentrate on the linear case. For a given $k \times d$ matrix W, the scores are then given by $\langle W_j, x\rangle$, $j = 1,\ldots,k$. To interpret them as probabilities, we map the scores to normalized scores

$$\frac{\exp\{\langle W_j, x\rangle\}}{\sum_i \exp\{\langle W_i, x\rangle\}}, \quad j = 1,\ldots,k$$

and the loss function is the negative logarithm of the probability assigned to the correct class:

$$\boldsymbol{\ell}(W,(x,y)) = -\log\left(\frac{\exp\{\langle W_y, x\rangle\}}{\sum_i \exp\{\langle W_i, x\rangle\}}\right) = -\langle W_y, x\rangle + \log\left(\sum_i \exp\{\langle W_i, x\rangle\}\right)$$

Let $s_j = \langle W_j, x\rangle$ denote the score for class j given by W, and $p_j = \frac{\exp\{\langle W_j, x\rangle\}}{\sum_i \exp\{\langle W_i, x\rangle\}}$. Then

$$\frac{\partial \boldsymbol{\ell}}{\partial s_j} = -\mathbf{1}\{j = y\} + p_j \tag{5}$$

and

$$\frac{ds_j}{dW_j} = x^{\mathsf{T}}. \tag{6}$$

Hence,

$$\frac{\partial \boldsymbol{\ell}}{\partial W_j} = (-\mathbf{1}\{j = y\} + p_j)x^{\mathsf{T}} \tag{7}$$

and

$$\frac{\partial \boldsymbol{\ell}}{\partial W} = zx^{\mathsf{T}} \tag{8}$$

where $z = [p_1, \dots, p_y - 1, \dots, p_k]^{\mathsf{T}}$. We can now use this expression directly in the SGD update.

Homework: add b_j to the linear form $\langle W_j, x\rangle$. The vector $b \in \mathbb{R}^k$ is now an additional parameter. Find the subgradient with respect to b.

In the process of calculation softmax, one may quickly run into numerical problems. To mitigate these issues in practice, observe that we may subtract any constant from each $\langle W_j, x\rangle$ before exponentiating, without changing the distribution. For instance, we can subtract off $\max_j \langle W_j, x\rangle$, ensuring that the maximum value after exponentiating is no larger than 1.

0.3 A discussion of loss functions

At this point it might be worthwhile to draw the various loss functions for binary classification: indicator loss, hinge, logistic. We may think of them as functions of $\widehat{y}_t y_t$, where $\widehat{y}_t$ is the output of a classifier, and y_t is the label for the data point x_t: $\mathbf{1}\{\widehat{y}_t y_t < 0\}$, $\max\{0, 1 - \widehat{y}_t y_t\}$, etc.

Let us also introduce squared-hinge (which gives rise to the so-called L2-SVM):

$$\boldsymbol{\ell}(w,(x,y)) = \max\{0, 1 - y\langle w, x\rangle\}^2$$

This loss function is widely used in practice. Squared hinge is smooth, and hence convergence might be faster. It penalizes errors heavier than hinge or logistic (which may or may not be a desirable quality), and becomes zero once the margin of 1 is satisfied just like regular hinge (this is often viewed as a positive thing). On the other hand, logistic is closer to hinge in terms of its penalization of errors, yet it is "never happy" — even if prediction is correct. Especially in the case of multiclass categorization, softmax might keep being unhappy with one of the classes, and that will lead to global changes because updates are coupled through a probability distribution based on scores. Depending on a particular situation, one loss may be more preferable than another.

0.4 Nonlinearities

So far in this course we have studied *linear* classifiers: we compute $\langle w, x\rangle$, interpreted as a "score" of the classifier, and then apply a possibly nonlinear function that gives a prediction (sign for binary classification, probability distribution for multiclass logistic), and a nonlinear loss function (indicator, hinge, logarithm). Regardless of the nonlinear loss, the decision boundary is still linear in the space where x lives (or, k-linear for multiclass). This limits the applicability of the methods we described in some real-world problems, where the decision boundary (if we could only peek into this high-dimensional space) would be nonlinear.

There are two popular ways to make the function of x more wiggly. First is to replace $\langle w, x\rangle$ with $\langle w, \phi(x)\rangle$ for some feature map ϕ into a high (or infinite) dimensional space. We will discuss this "kernel methods" approach later. Notably it still leads to a *convex* optimization objective, but over the coefficients of the wiggly functions. Just because the decision boundary becomes non-convex does not mean we can't have a convex optimization problem over the coefficients of these functions!

The second approach is:

0.5 Neural networks

The second popular approach is to forego convexity and introduce nonlinearity in a way that does not look like $\langle w, \phi(x)\rangle$. One way to do this is to map x into Wx for some matrix $W \in \mathbb{R}^{d_1 \times d}$ and then apply a nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$ *coordinate-wise*:

$$x \mapsto \sigma(Wx).$$

There are many variants for the function σ; most popular are sigmoid $a \mapsto (1 + \exp(-a))^{-1}$ and positive part $a \mapsto \max\{0, a\}$ (also called ReLU). The resulting d_1-dimensional vector can now be used, for instance, with a linear classifier and hinge loss, resulting in the following optimization problem:

$$\min_{w,W} \frac{1}{n} \sum_{t=1}^{n} \max\{0, 1 - y_t \langle w, \sigma(Wx_t)\rangle\}. \tag{9}$$

This would be called a neural network with 1 hidden layer which is then combined in a linear fashion via w to yield prediction. One may also put a nonlinearity at the final output stage, as we will do later.

We may build a more complex model with more than 1 hidden layer. For instance, a two-layer network may look like

$$\min_{w,W^1,W^2} \frac{1}{n} \sum_{t=1}^{n} \max\{0, 1 - y_t \left\langle w, \sigma(W^2 \sigma(W^1 x_t))\right\rangle\}, \tag{10}$$

where $W^1 \in \mathbb{R}^{d_1 \times d}$, $W^2 \in \mathbb{R}^{d_2 \times d_1}$, $w \in \mathbb{R}^{d_2}$. And so forth. Note that combining several layers dramatically increases the number of linear pieces that make up the decision boundary.

We may use this architecture with multiclass softmax, which leads to

$$\min_{W,W^1,W^2} \frac{1}{n} \sum_{t=1}^{n} -\log\left(\frac{\exp\{\left\langle W_{y_t}, \sigma(W^2 \sigma(W^1 x_t))\right\rangle\}}{Z}\right) \tag{11}$$

And so forth.

The bottom line is that all these formulations are nonconvex. Yet, in practice, SGD still performs well. There is almost no theory as to why this happens. On the other hand, there are plenty of heuristic motivations for the use of multiple layers (depth). These motivations include hierarchical organization, reuse of information within the network, and so on. There is also some theoretical evidence that the representational power with an additional layer is significantly greater than adding these nodes to an existing layer.

Note: so far, we have not isolated a constant shift of the linear functions and instead included it as an additional coordinate. It is more common to write it explicitly and update it in SGD separately, without having to augment x's.

We now turn to the question of implementing SGD. To this end, we need to find a gradient or a subgradient of the loss with respect to each parameter. For this, all we need is to remember the chain rule from calculus since some of these parameters are buried within other functions. The calculation is termed "backpropagation" because the chain rule has a nice modular implementation as a backward pass following a forward calculation.

0.6 Backprop

Consider a network with r layers, and take square loss (for simplicity) as the loss function:

$$\frac{1}{n}\sum_{t=1}^{n}\frac{1}{2}\Big[y_t - \sigma(W^r\sigma(\ldots\sigma(W^1x)))\Big]^2. \tag{12}$$

Once again, the nonlinearity σ is applied coordinate-wise, and so the dimensions of the matrices $W^r,\ldots,W^1$ are $(1\times d_{r-1}),(d_{r-1}\times d_{r-2}),\ldots,(d_1\times d)$, and $x\in\mathbb{R}^d$. Let us introduce the notation:

$$O^j = \sigma(N^j), \quad N^j = W^j\sigma(\ldots\sigma(W^1x))$$

with O^r being the output of the network.

We have

$$\frac{dO_i^j}{dN_i^j} = O_i^j(1-O_i^j)$$

for the case of sigmoid, while for ReLU

$$\frac{dO_i^j}{dN_i^j} = \mathbf{1}\Big\{N_i^j > 0\Big\}.$$

Define a shorthand

$$\xi_i^j \triangleq \frac{\partial \boldsymbol{\ell}}{\partial O_i^j}\times\frac{\partial O_i^j}{\partial N_i^j}.$$

Then

$$\frac{\partial \boldsymbol{\ell}}{\partial W_{i,m}^j} = \frac{\partial \boldsymbol{\ell}}{\partial N_i^j}\times\frac{\partial N_i^j}{\partial W_{i,m}^j} = \frac{\partial \boldsymbol{\ell}}{\partial O_i^j}\times\frac{\partial O_i^j}{\partial N_i^j}\times\frac{\partial N_i^j}{\partial W_{i,m}^j} = \xi_i^j O_m^{j-1} \tag{13}$$

where we define $O_m^0 = x_m$. If $j\neq r$ (not the final layer),

$$\frac{\partial \boldsymbol{\ell}}{\partial O_i^j} = \sum_{s=1}^{d_{j+1}}\frac{\partial \boldsymbol{\ell}}{\partial N_s^{j+1}}\frac{\partial N_s^{j+1}}{\partial O_i^j} = \sum_{s=1}^{d_{j+1}}\frac{\partial \boldsymbol{\ell}}{\partial O_s^{j+1}}\frac{\partial O_s^{j+1}}{\partial N_s^{j+1}}\frac{\partial N_s^{j+1}}{\partial O_i^j} \tag{14}$$

$$= \sum_{s=1}^{d_{j+1}}\xi_s^{j+1}W_{s,i}^{j+1} \tag{15}$$

and thus

$$\xi_i^j = \frac{\partial \boldsymbol{\ell}}{\partial O_i^j} \times \frac{\partial O_i^j}{\partial N_i^j} = \frac{\partial O_i^j}{\partial N_i^j} \sum_{s=1}^{d_{j+1}} \xi_s^{j+1} W_{s,i}^{j+1}. \tag{16}$$

The key observation is that the quantities ξ can be calculated in a backward manner, starting with the output layer. Let us detail this for the sigmoid function. Then

$$\frac{\partial O_i^j}{\partial N_i^j} = O_i^j \cdot (1 - O_i^j)$$

If the layer is not final ($j \neq r$), then

$$\xi_i^j = O_i^j \cdot (1 - O_i^j) \sum_{s=1}^{d_{j+1}} \xi_s^{j+1} W_{s,i}^{j+1}. \tag{17}$$

On the other hand, if $j = r$, then (keeping in mind that the output final dimension is 1),

$$\xi^r = \frac{\partial \boldsymbol{\ell}}{\partial O^r} \times \frac{\partial O^r}{\partial N^r} = (O^r - y) O^r (1 - O^r). \tag{18}$$

Now that the recursive definition of ξ is complete, we recall (13) to define all the partial derivatives:

$$\frac{\partial \boldsymbol{\ell}}{\partial W_{i,m}^j} = \xi_i^j O_m^{j-1} \tag{19}$$

Note that O^j's are calculated during the forward pass and ξ^j's are calculated during the backward pass.

References