Loss functions

We have discussed the regularization loss part of the objective, which can be seen as penalizing some measure of complexity of the model. The second part of an objective is the *data loss*, which in a supervised learning problem measures the compatibility between a prediction (e.g. the class scores in classification) and the ground truth label. The data loss takes the form of an average over the data losses for every individual example. That is, $L = \frac{1}{N}\sum_i L_i$ where N is the number of training data. Lets abbreviate $f = f(x_i; W)$ to be the activations of the output layer in a Neural Network. There are several types of problems you might want to solve in practice:

Classification is the case that we have so far discussed at length. Here, we assume a dataset of examples and a single correct label (out of a fixed set) for each example. One of two most commonly seen cost functions in this setting is the SVM (e.g. the Weston Watkins formulation):

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

As we briefly alluded to, some people report better performance with the squared hinge loss (i.e. instead using $\max(0, f_j - f_{y_i} + 1)^2$). The second common choice is the Softmax classifier that uses the cross-entropy loss:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

Problem: Large number of classes. When the set of labels is very large (e.g. words in English dictionary, or ImageNet which contains 22,000 categories), computing the full softmax probabilities becomes expensive. For certain applications, approximate versions are popular. For instance, it may be helpful to use *Hierarchical Softmax* in natural language processing tasks (see one explanation here (pdf)). The hierarchical softmax decomposes words as labels in a tree. Each label is then represented as a path along the tree, and a Softmax classifier is trained at every node of the tree to disambiguate between the left and right branch. The structure of the tree strongly impacts the performance and is generally problem-dependent.

Attribute classification. Both losses above assume that there is a single correct answer y_i. But what if y_i is a binary vector where every example may or may not have a certain attribute, and where the attributes are not exclusive? For example, images on Instagram can be thought of as labeled with a certain subset of hashtags from a large set of all hashtags, and an image may contain multiple. A sensible approach in this case is to build a binary classifier for every single attribute independently. For example, a binary classifier for each category independently would take the form:

Back to Top

$$L_i = \sum_j \max(0, 1 - y_{ij} f_j)$$

where the sum is over all categories j, and y_{ij} is either +1 or -1 depending on whether the i-th example is labeled with the j-th attribute, and the score vector f_j will be positive when the class is predicted to be present and negative otherwise. Notice that loss is accumulated if a positive example has score less than +1, or when a negative example has score greater than -1.

An alternative to this loss would be to train a logistic regression classifier for every attribute independently. A binary logistic regression classifier has only two classes (0,1), and calculates the probability of class 1 as:

$$P(y = 1 \mid x; w, b) = \frac{1}{1 + e^{-(w^T x + b)}} = \sigma(w^T x + b)$$

Since the probabilities of class 1 and 0 sum to one, the probability for class 0 is $P(y = 0 \mid x; w, b) = 1 - P(y = 1 \mid x; w, b)$. Hence, an example is classified as a positive example (y = 1) if $\sigma(w^T x + b) > 0.5$, or equivalently if the score $w^T x + b > 0$. The loss function then maximizes this probability. You can convince yourself that this simplifies to minimizing the negative log-likelihood:

$$L_i = -\sum_j y_{ij} \log(\sigma(f_j)) + (1 - y_{ij}) \log(1 - \sigma(f_j))$$

where the labels y_{ij} are assumed to be either 1 (positive) or 0 (negative), and $\sigma(\cdot)$ is the sigmoid function. The expression above can look scary but the gradient on f is in fact extremely simple and intuitive: $\partial L_i / \partial f_j = \sigma(f_j) - y_{ij}$ (as you can double check yourself by taking the derivatives).

Regression is the task of predicting real-valued quantities, such as the price of houses or the length of something in an image. For this task, it is common to compute the loss between the predicted quantity and the true answer and then measure the L2 squared norm, or L1 norm of the difference. The L2 norm squared would compute the loss for a single example of the form:

$$L_i = \|f - y_i\|_2^2$$

The reason the L2 norm is squared in the objective is that the gradient becomes much simpler, without changing the optimal parameters since squaring is a monotonic operation. The L1 norm would be formulated by summing the absolute value along each dimension:

$$L_i = \|f - y_i\|_1 = \sum_j \mid f_j - (y_i)_j \mid$$

Back to Top

where the sum $\sum_j$ is a sum over all dimensions of the desired prediction, if there is more than one quantity being predicted. Looking at only the j-th dimension of the i-th example and denoting the difference between the true and the predicted value by δ_{ij}, the gradient for this dimension (i.e. $\partial L_i / \partial f_j$) is easily derived to be either δ_{ij} with the L2 norm, or $sign(\delta_{ij})$. That is, the gradient on the score will either be directly proportional to the difference in the error, or it will be fixed and only inherit the sign of the difference.

Word of caution: It is important to note that the L2 loss is much harder to optimize than a more stable loss such as Softmax. Intuitively, it requires a very fragile and specific property from the network to output exactly one correct value for each input (and its augmentations). Notice that this is not the case with Softmax, where the precise value of each score is less important: It only matters that their magnitudes are appropriate. Additionally, the L2 loss is less robust because outliers can introduce huge gradients. When faced with a regression problem, first consider if it is absolutely inadequate to quantize the output into bins. For example, if you are predicting star rating for a product, it might work much better to use 5 independent classifiers for ratings of 1-5 stars instead of a regression loss. Classification has the additional benefit that it can give you a distribution over the regression outputs, not just a single output with no indication of its confidence. If you're certain that classification is not appropriate, use the L2 but be careful: For example, the L2 is more fragile and applying dropout in the network (especially in the layer right before the L2 loss) is not a great idea.

> *When faced with a regression task, first consider if it is absolutely necessary. Instead, have a strong preference to discretizing your outputs to bins and perform classification over them whenever possible.*

Structured prediction. The structured loss refers to a case where the labels can be arbitrary structures such as graphs, trees, or other complex objects. Usually it is also assumed that the space of structures is very large and not easily enumerable. The basic idea behind the structured SVM loss is to demand a margin between the correct structure y_i and the highest-scoring incorrect structure. It is not common to solve this problem as a simple unconstrained optimization problem with gradient descent. Instead, special solvers are usually devised so that the specific simplifying assumptions of the structure space can be taken advantage of. We mention the problem briefly but consider the specifics to be outside of the scope of the class.

Summary

In summary:

- The recommended preprocessing is to center the data to have mean of zero, and normalize its scale to [-1, 1] along each feature