Loss Functions

Classification:

- Log Loss
- Focal Loss: address data imbalance in classification problems.
-
- KL Divergence/ Relative Entropy (Multi)
- Exponential Loss
- Hinge Loss
- Cosine Normalization Loss
- Margin Loss
- CE loss: The cross-entropy loss is based on the idea of finding parameters θ that minimize the distance between the empirical distribution q(y) of the observed data y and a model distribution Pr(y|θ) (figure 5.12). The distance between two probability distributions q(z) and p(z) can be evaluated using the Kullback-Leibler (KL) divergence

- Sigmoid CE (Binary)
- Weighted CE (Binary)
- Softmax CE (Multi)
- Sparse CE (Multi)
- Negative Log Likelihood loss: Obtaining log-probabilities in a neural network is easily achieved by adding a LogSoftmax layer in the last layer of your network. You may use CrossEntropyLoss instead, if you prefer not to add an extra layer.

The sigmoid function is used for the two-class logistic regression, whereas the softmax function is used for the multiclass logistic regression (a.k.a. MaxEnt, multinomial logistic regression, softmax Regression, Maximum Entropy Classifier). One can observe that the softmax function is an extension of the sigmoid function to the multiclass case.

Summary				
Problem type	**Last layer output nodes**	**Hidden layer activation**	**Last layer activation**	**Loss function**
Binary classification	1	ReLU (first choice)	Sigmoid	Binary Cross Entropy
				Weighted Cross Entropy
			Tanh	Hinge Loss
Multi-class, single label classification	Number of classes		SoftMax	Categorical Cross Entropy
				Sparse Categorical Cross Entropy
				KullBack Leiber Divergence Loss
Multi-class, multi label classification	Number of classes		Sigmoid (one for each class)	Binary Cross Entropy

Fig 8: Summary of loss functions

Regression

- Mean Squared Loss: We see that the least squares loss function follows naturally from the assumptions that the prediction errors are (i) independent and (ii) drawn from a normal distribution with mean $\mu = f[x_i,\phi]$
- Mean Absolute Loss(Robust regression): This loss function follows from assuming a Laplace distribution over the outputs and estimates the median output for a given input rather than the mean. Barron (2019) presents a loss function that parameterizes the degree of robustness. When interpreted in a probabilistic context, it yields a family of univariate probability distributions that includes the normal and Cauchy distributions as special cases.
-
- Huber Loss/Smooth Mean Absolute Loss
- Log Cash Loss
- Quantile Loss

Segmentation

- Dist based
    - CE Loss
    - Weighted CE Loss
    - TopK Loss
    - Focal Loss
    - Distance Penalized CE Loss
- Region Based
    - Sensitivity-Specificity Loss
    - Dice Loss
    - IoU Loss
    - Tversky Loss
    - Generalized Dice Loss
    - Focal Tversky Loss
    - Penalty Loss
- Boundary Based
    - Boundary Loss
    - Hausdorff Distance

Optimization

This Gabor model maps scalar input x to scalar output y and consists of a sinusoidal component (creating an oscillatory function) multiplied by a negative exponential component (causing the amplitude to decrease as we move from the center).

SGD has several attractive features. First, although it adds noise to the trajectory, it still improves the fit to a subset of the data at each iteration. Hence, the updates tend to be sensible even if they are not optimal. Second, because it draws training examples

without replacement and iterates through the dataset, the training examples all still contribute equally. Third, it is less computationally expensive to compute the gradient from just a subset of the training data. Fourth, it can (in principle) escape local minima.
Fifth, it reduces the chances of getting stuck near saddle points; it is likely that at least some of the possible batches will have a significant gradient at any point on the loss function. Finally, there is some evidence that SGD finds parameters for neural networks that cause them to generalize well to new data in practice (see section 9.2).

Momentum A common modification to stochastic gradient descent is to add a momentum term. We update the parameters with a weighted combination of the gradient computed from the current batch and the direction moved in the previous step

Nesterov accelerated momentum Nesterov accelerated momentum (figure 6.8) computes the gradients at this predicted point rather than at the current point. One way to think about this is that the gradient term now corrects the path provided by momentum alone.

ADAM Gradient descent with a fixed step size has the following undesirable property: it makes large adjustments to parameters associated with large gradients (where perhaps we should be more cautious) and small adjustments to parameters associated with small gradients (where perhaps we should explore further). A straightforward approach is to normalize the gradients so that we move a fixed distance (governed by the learning rate) in each direction. To do this, we first measure the gradient mt+1 and the pointwise squared gradient vt+1. The result is that the algorithm moves a fixed distance α along each coordinate, where the direction is determined by whichever way is downhill. Adaptive moment estimation, or Adam, takes this idea and adds momentum to both the estimate of the gradient and the squared gradient. In practice, Adam also has the advantage of being less sensitive to the initial learning rate because it avoids situations like those in figures 6.9a–b, so it doesn't need complex learning rate schedules.

One potential problem with adaptive training algorithms is that the learning rates are based on accumulated statistics of the observed gradients. At the start of training, when there are few samples, these statistics may be very noisy. This can be remedied by learning rate warm-up.

SGD vs. Adam: There has been a lively discussion about the relative merits of SGD and Adam. Wilson et al. (2017) provided evidence that SGD with momentum can find lower minima than Adam, which generalizes better over a variety of deep learning tasks. However, this is strange since SGD is a special case of Adam (when $\epsilon = \gamma = 0$) once the modification term (equation 6.16) becomes one, which happens quickly. It is hence more likely that SGD outperforms Adam when we use Adam's default hyperparameters. Loshchilov & Hutter (2019) proposed AdamW, which substantially improves the performance of Adam in the presence of L2 regularization (see section 9.1). Choi et al. (2019) provide evidence that if we search for the best Adam hyperparameters, it performs just as well as SGD and converges faster. Keskar & Socher (2017) proposed a method called SWATS that starts using Adam (to make rapid initial progress) and then switches to SGD (to get better final generalization performance).

Label smoothing?