Loss functions & training

Course of Machine Learning
Master Degree in Computer Science
University of Rome "Tor Vergata"
a.a. 2021-2022

Giorgio Gambosi

Loss function

- In general, the loss function $L : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbf{R}$ measures, for any two values y, t in target space, the *cost* of referring, for any subsequent action, to t instead of the better value y
- In supervised learning, it provides a measure of the quality of the prediction returned by the prediction function h

$$\mathcal{R}(\mathbf{x}, y) = L(h(\mathbf{x}), y)$$

- It is a fundamental component of the empirical risk, which is just the average value of the loss function applied to all predicted value - target value pairs in the training set $\mathcal{T}$

$$\overline{\mathcal{R}}_{\mathcal{T}}(h) = \frac{1}{|\mathcal{T}|} \sum_{(x,t) \in \mathcal{T}} L(h(x), t)$$

- That is, it provides a measure of the quality of the predictions performed by h, at least with respect to the available data (the training set)

Loss function & training

- During the training phase, the empirical risk is minimized wrt the prediction function h applied, and in particular to the set of parameters $\boldsymbol{\theta}$ which specifies the parametric function $h = h_\theta$
- This corresponds to minimizing the overall loss

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = \sum_{i=1}^{n} L_i(\boldsymbol{\theta})$$

that is the sum of the loss functions $L_i = L(\boldsymbol{\theta}; \mathbf{x}_i, y_i)$

Loss function minimization

How to deal with loss minimization?

- we would like to compute a *global minimum*
- methods based on calculus rely on setting all derivatives to 0, that is,

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = \mathbf{0}$$

that is

$$\frac{\partial}{\partial \theta_i} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = 0 \qquad \forall i$$

and solve the corresponding system of equations

Problems

- the system of equations has multiple solutions (local minima/maxima, saddle points)
- they can be hard (or impossible) to compute analytically

Loss function minimization

- A local minimum of $\overline{\mathcal{R}}_{\mathcal{T}}(\boldsymbol{\theta})$ can be computed numerically, by means of iterative methods such as *gradient descent*
- Initial assignment $\boldsymbol{\theta}^{(0)} = (\theta_0^{(0)}, \theta_1^{(0)}, \ldots, \theta_d^{(0)})$, with a corresponding error value

  $$\overline{\mathcal{R}}_{\mathcal{T}}(\boldsymbol{\theta}^{(0)})$$

- Iteratively, the current value $\boldsymbol{\theta}^{(i-1)}$ is modified in the direction of *steepest descent* of $\overline{\mathcal{R}}_{\mathcal{T}}(\boldsymbol{\theta})$, that is the one corresponding to the negative of the gradient evaluated at $\boldsymbol{\theta}^{(i-1)}$
- At step i, $\theta_j^{(i-1)}$ is updated as follows:

$$\theta_j^{(i)} := \theta_j^{(i-1)} - \eta \frac{\partial}{\partial \theta_j} \overline{\mathcal{R}}_{\mathcal{T}}(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta}^{(i-1)}} = \theta_j^{(i-1)} - \frac{\eta}{|\mathcal{T}|} \sum_{(\mathbf{x},t) \in \mathcal{T}} \frac{\partial}{\partial \theta_j} L(h_{\boldsymbol{\theta}}(\mathbf{x}), t) \Big|_{\boldsymbol{\theta}^{(i-1)}}$$

Gradient descent

- In matrix notation:

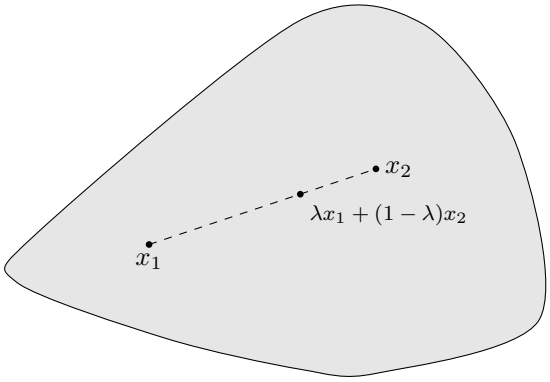
$$\boldsymbol{\theta}^{(i)} := \boldsymbol{\theta}^{(i-1)} - \eta \nabla_{\boldsymbol{\theta}} \overline{\mathcal{R}}_{\mathcal{T}}(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta}^{(i-1)}} = \theta_j^{(i-1)} - \frac{\eta}{|\mathcal{T}|} \sum_{(\mathbf{x},t) \in \mathcal{T}} \nabla_{\boldsymbol{\theta}} \overline{\mathcal{R}}_{\mathcal{T}}(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta}^{(i-1)}}$$

- clearly this approach makes it possible to find (approximate) a local minimum, depending from the initial values; some problems
  - we are looking for a global (not simply a local) minimum
  - how to deal with saddle points?
  - how fast does the method converge?
- More on this later

Convexity

A set of points $S \subset \mathbb{R}^d$ is *convex* iff for any $\mathbf{x}_1, \mathbf{x}_2 \in S$ and $\lambda \in (0, 1)$

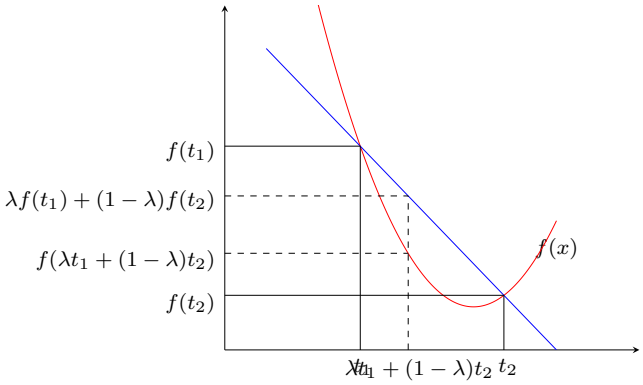
$$\lambda \mathbf{x}_1 + (1 - \lambda) \mathbf{x}_2 \in S$$



Convexity

A function $f(\mathbf{x})$ is convex iff the set of points lying above the function is convex, that is, forall $\mathbf{x}_1, \mathbf{x}_2$ and $\lambda \in (0, 1)$,

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$



Convexity

- Assuming $\mathcal{L}(\boldsymbol{\theta}; \mathcal{T})$ is convex is a relevant simplification: if $f(\mathbf{x})$ is a convex function, then any local minimum of f is also a global minimum
- Moreover, if f is a *strictly convex* function, there exists only one local minimum for f (and it is global), that is, solving

  $$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = \mathbf{0}$$

  provides the global minimum
- Definition: $f(\mathbf{x})$ is strictly convex iff forall $\mathbf{x}_1, \mathbf{x}_2$ and $\lambda \in (0, 1)$,

  $$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) < \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$

- A simple but relevant case: $f(\mathbf{x})$ is quadratic. This is the case for a number of simpler ML models. Unfortunately this is not true for more complex models such as neural networks

Convexity and empirical risk

- convex functions properties:

  - the sum of (strictly) convex functions is (strictly) convex
  - the product of a (strictly) convex function and a constant is (strictly) convex

- since

$$\overline{\mathcal{R}}_{\mathcal{T}}(h) = \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},t)\in\mathcal{T}} L(h(\mathbf{x}),t) \propto \sum_{(\mathbf{x},t)\in\mathcal{T}} L(\boldsymbol{\theta};\mathbf{x},t)$$

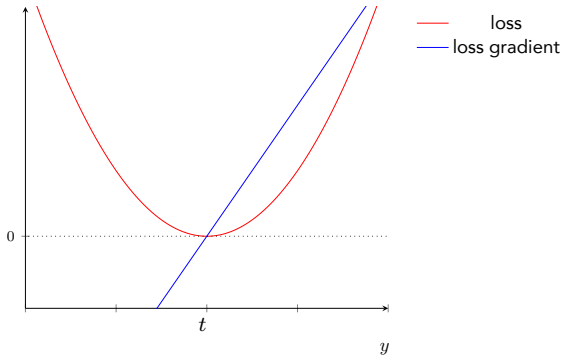
- if $L(\boldsymbol{\theta};\mathbf{x},t)$ is (strictly) convex then the overall cost is also (strictly) convex
- if $L(\boldsymbol{\theta};\mathbf{x},t)$ is convex then any local minimum of the empirical risk is also a global one
- if $L(\boldsymbol{\theta};\mathbf{x},t)$ is strictly convex then there exists only one minimum of the empirical risk

Some common loss functions

Let us first consider the case of *regression*.

- both y and $h(\mathbf{x})$ are real values
- loss is related to some type of point distance measure
- most common loss function for regression: *quadratic loss*

$$L(y,t) = (y-t)^2$$



Quadratic loss

- Applying quadratic loss results in the empirical risk

$$\overline{\mathcal{R}}_{\mathcal{T}}(h) = \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},t)\in\mathcal{T}} (h(\mathbf{x})-t)^2$$

- in the common case of linear regression, the prediction is performed by means of a linear function $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$: this results into an overall loss to be minimized

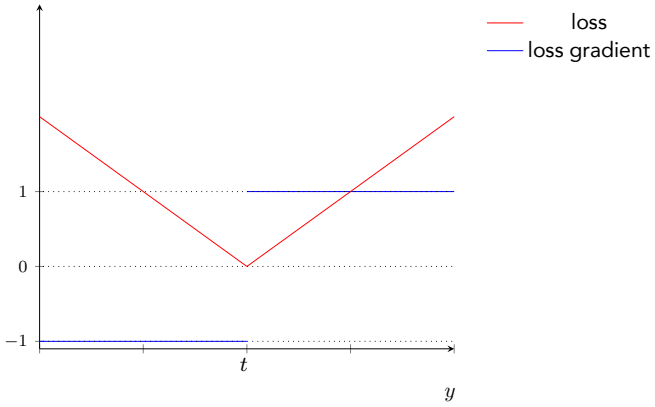
$$\mathcal{L}(\mathbf{w},b;\mathcal{T}) = \sum_{(\mathbf{x},t)\in\mathcal{T}} (\mathbf{w}^T\mathbf{x}+b-t)^2$$

- since the quadratic function is strictly convex, the overall loss has only one local minimum (which is global)
- the gradient is linear

$$\frac{\partial}{\partial w_i}\mathcal{L}(\mathbf{w},b;\mathcal{T}) = \sum_{(\mathbf{x},t)\in\mathcal{T}} (\mathbf{w}^T\mathbf{x}+b-t)w_i \qquad \frac{\partial}{\partial b}\mathcal{L}(\mathbf{w},b;\mathcal{T}) = \sum_{(\mathbf{x},t)\in\mathcal{T}} (\mathbf{w}^T\mathbf{x}+b-t)$$

Additional loss functions for regression: absolute loss

- Quadratic loss is easy to deal with mathematically, but not robust to outliers, i.e. pays too much attention to outliers.
- A different loss function: *absolute loss* $L(t, y) = |t - y|$

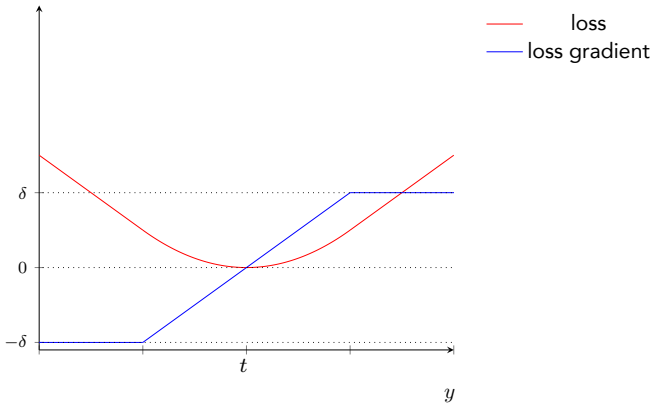


- The gradient is piecewise constant

Additional loss functions for regression: Huber loss

- Another different loss function: *Huber loss*

$$L(t, y) = \begin{cases} \frac{1}{2}(t - y)^2 & |t - y| \leq \delta \\ \delta(|t - y|) - \frac{\delta}{2} & |t - y| > \delta \end{cases}$$



Loss functions for classification

- Essentially, two approaches, depending on what we expect the prediction return:
  - prediction returns a specific class (prediction function)
  - prediction returns a probability distribution on the set of classes (prediction distribution)
- Different definition of error

- first case: coincidence of predicted and real classes
- second case: cumulative difference between predicted probability and $0/1$ for all classes

- We consider the binary case, with two classes identified by target values -1 and 1
- Assume a real value is returned as a prediction

0/1 loss

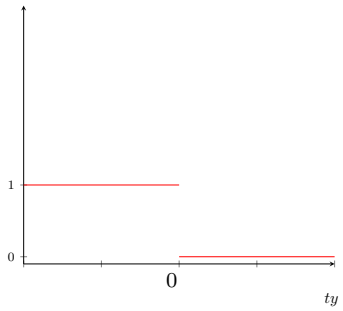
- The most "natural" loss function in classification

$$L(t, y) = \begin{cases} 1 & \text{sgn}(t) \neq y \\ 0 & \text{sgn}(t) = y \end{cases}$$

  where sgn(x) is 1 if $x > 0$ and -1 otherwise.

- This can be written as:

  $\mathbf{1}[ty > 0]$



0/1 loss

Problem:

- not convex
- not smooth (first derivative undefined in some points or not continue)
- gradient is 0 almost everywhere (undefined at 0): gradient descent cannot be applied
- if we assume a linear prediction function

$$\overline{\mathcal{R}}_{\mathcal{T}}(h) = \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},t)\in\mathcal{T}} \mathbf{1}[(\mathbf{w}^T\mathbf{x} + b)y > 0]$$

- the problem is finding the values $\mathbf{w}, b$ which minimize the overall number of errors: this is an NP-hard, hence a computationally intractable problem.

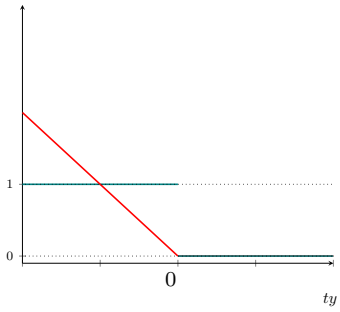
Convex surrogate loss functions

- Approximate *from above* 0/1 loss: real 0/1 error always less than function loss
- Convex: unique local minimum = global minimum

- Smooth: may use derivatives to find minimum
- Main difference: relevance given to erroneous predictions

Perceptron loss

- 0/1 loss assigns the same cost 1 to each error
- assume a prediction t is a real value: then, in the case of a misclassified element, the error can be measured as $-ty > 0$. That is, $L(t, y) = \max(0, -yt)$
- in the case of correctly classified element, the error is 0, while in the case of a wrong prediction, the error is equal to $|t|$
- Main difference: relevance given to erroneous predictions. The perceptron loss penalizes prediction which are largely wrong (for example a negative value $\simeq -1$ while correct class is 1)
- continuous, gradient continuous almost everywhere, convex (but not strictly convex), not surrogate

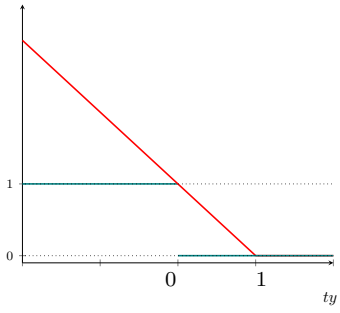


Hinge loss

- used in support vector machine training
- related to perceptron loss, but surrogate
- assume a prediction

$$L(t, y) = \max(0, 1 - yt)$$

- correct predictions can be penalized if "weak" (small value of t)
- continuous, gradient continuous almost everywhere, convex (but not strictly convex), surrogate



Hinge loss

Hinge loss $L_H(y,t) = \max(0, 1-yt)$ is not differentiable wrt to y at $ty = 1$. The same holds for perceptron loss at $ty = 0$.

For example,

$$\frac{\partial}{\partial y} L_H = \begin{cases} -t & ty < 1 \\ 0 & ty > 1 \\ \text{undefined} & ty = 1 \end{cases}$$

This is a problem if gradient descent should be applied. In this case a subgradient can be used.

Subgradient

Given a convex function (such as hinge loss) f at each differentiable point, the corresponding gradient $\nabla(x)$ provides a function which lower bounds f

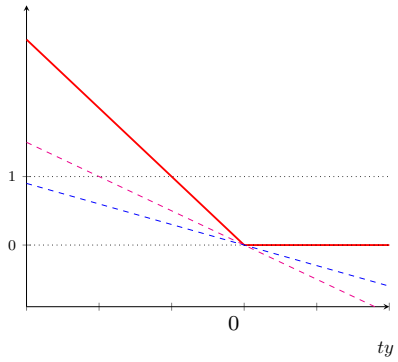
$$f(x') \geq f(x) + \nabla(x)(x - x')$$

If x is a singular point, where f is not differentiable and $\nabla(x)$ does not exist, a *subgradient* $\overline{\nabla}(x)$ is *any* function which lower bounds f

$$f(x') \geq f(x) + \overline{\nabla}(x)(x - x')$$

Subgradient and hinge loss

In the case of hinge loss, we may observe that any line whose slope in $[-t, 0]$ (if $t = 1$, in $[0, -t]$ if $t = -1$) is a subgradient



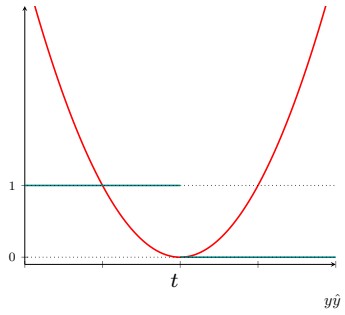
We may then choose the horizontal axis as the subgradient to use,

Square loss

- adapted to the classification case

$$L(t,y) = (1 - yt)^2$$

- continuous, gradient continuous, convex, not surrogate
- largely wrong predictions can be too penalized
- symmetric around 0: even largely wight predictions are penalized

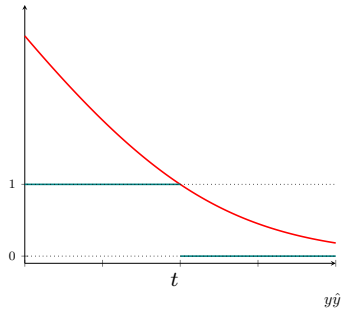


Log loss (cross entropy)

- used in logistic regression

$$L(t, y) = \frac{1}{\log 2} \log(1 + e^{-yt})$$

- a smoothed version of hinge loss
- continuous, gradient continuous, convex, surrogate
- largely wrong predictions can be too penalized
- symmetric around 0: even largely wrong predictions are penalized



What is the relation with cross entropy?

- given distributions p, q the cross entropy of q wrt p is defined as

$$-E_p[\log q(x)] = -\int p(x) \log q(x) dx$$

- the cross entropy is a measure of how much p and q are different
- it is related to the Kullback-Leibler divergence

$$KL(p||q) = -\int p(x) \log \frac{q(x)}{p(x)} dx = -\int p(x) \log q(x) dx + \int p(x) \log p(x) dx = -E_p[\log q(x)] - H(p)$$

where $H(p) = -E_p[\log p(x)]$ is the *entropy* of p

What is the relation with cross entropy?

- the entropy $H(p) = -E_p[\log p]$ denotes the expected number of bits per symbol x in a transmission channel where the distribution of symbols $p(x)$ is known
- the cross entropy $-E_p[\log q]$ denotes the additional (with respect to the minimum) expected number of bits per symbol x in a transmission channel where the distribution of symbols $q(x)$ is used, instead of $p(x)$
- the KL divergence $KL(p||q)$ denotes the total expected number of bits per symbol x in a transmission channel where the distribution of symbols $q(x)$ is used, instead of $p(x)$

What is the relation with cross entropy?

- consider now a classifier which predicts the probability that an element is in class C_1 and let
  - p be the probability that the element is in class C_1: in the training set this is either 0 or 1, that is equal to the target value t
  - $y(\mathbf{x})$ be the predicted probability of the element being in class C_1
- the cross entropy $\mathsf{CE}(\mathcal{T})$ between real and predicted probability distribution over the set of elements can be estimated as the average

$$\mathsf{CE}(\mathcal{T}) = -\frac{1}{|\mathcal{T}|}\sum_{(\mathbf{x},t)\in\mathcal{T}}\Big(t\log y(\mathbf{x})+(1-t)\log(1-y(\mathbf{x}))\Big) = -\frac{1}{|\mathcal{T}|}\Big(\sum_{(\mathbf{x},t)\in C_1}\log y(\mathbf{x})+\sum_{(\mathbf{x},t)\in C_0}\log(1-y(\mathbf{x}))\Big)$$

- assume now the classifier is a logistic regression, that is

$$y(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}+b) = \frac{1}{1+e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

then,

$$\mathsf{CE}(\mathcal{T}) = \frac{1}{|\mathcal{T}|}\Big(\sum_{(\mathbf{x},t)\in C_1}\log(1+e^{-(\mathbf{w}^T\mathbf{x}+b)}) + \sum_{(\mathbf{x},t)\in C_0}\log(1+e^{\mathbf{w}^T\mathbf{x}+b})\Big)$$

What is the relation with cross entropy?

- assuming now that the target encodes classes as $\bar{t}\in\{-1,1\}$ (that is class C_0 is denoted by $\bar{t}=-1$ and) we have

$$\mathsf{CE}(\mathcal{T}) = \frac{1}{|\mathcal{T}|}\sum_{(\mathbf{x},t)\in\mathcal{T}}\log(1+e^{-t(\mathbf{w}^T\mathbf{x}+b)})$$

that, apart from the constant $\log 2$ corresponds to the empirical risk if log loss is applied

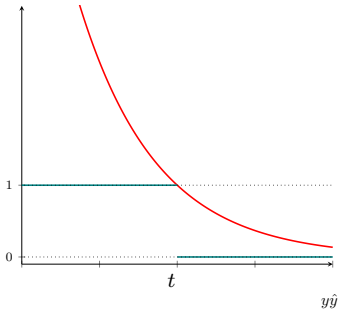
$$\overline{\mathcal{R}}_{\mathcal{T}}(h) = \frac{1}{|\mathcal{T}|\log 2}\sum_{(x,t)\in\mathcal{T}}\log(1+e^{-t(\mathbf{w}^T\mathbf{x}+b)})$$

Exponential loss

- used in boosting (Adaboost)

$$L(t,y) = e^{-yt}$$

- penalizes wrong predictions more than log loss: penalty grows more quickly as errors become larger
- continuous, gradient continuous, convex, surrogate



Convex surrogate loss functions

Let $y = \mathbf{w}^T\mathbf{x} + b$: some common loss functions

0/1 $L(t, y) = \mathbf{1}[yt \leq 0]$

Squared $L(t, y) = (t - y)^2$

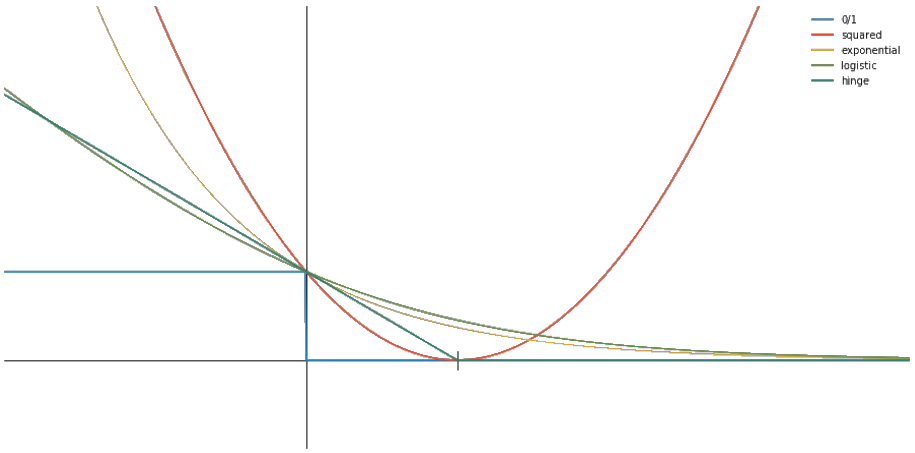
Logistic $L(t, y) = \frac{1}{\log 2} \log(1 + e^{-yt})$

Exponential $L(t, y) = e^{-yt}$

Hinge $L(t, y) = \max(0, 1 - yt)$

Convex surrogate loss functions

Plot of the defined loss functions: $y = \mathbf{w}^T\mathbf{x} + b$ on the horizontal axis, assuming $t = 1$



Computing h^*

- In most cases, $\Theta = \mathbf{R}^d$ for some $d > 0$: in this case, the minimization of $\overline{\mathcal{R}}_{\mathcal{T}}(h_\theta)$ is unconstrained and a (at least local) minimum could be computed setting all partial derivatives to 0

  $$\frac{\partial}{\partial \theta_i} \overline{\mathcal{R}}_{\mathcal{T}}(h_\theta) = 0$$

  that is, setting to zero the gradient of the empirical risk with respect to the vector of parameters θ

  $$\nabla_\theta \overline{\mathcal{R}}_{\mathcal{T}}(h_\theta) = \mathbf{0}$$

- The analytical solution of this set of equations is usually quite hard
- Numerical methods can be applied

Gradient descent

- Gradient descent performs minimization of a function $J(\theta)$ through iterative updates of the current value of θ (starting from an initial value $\theta^{(0)}$) in the opposite direction to the one specified by the current value of the gradient $J'(\theta) = \nabla_\theta J(\theta)$

  $$\theta^{(k+1)} = \theta^{(k)} - \eta J'(\theta) \mid_{\theta=\theta^{(k)}}$$

  that is, for each parameter θ_i

  $$\theta_i^{(k+1)} = \theta_i^{(k)} - \eta \frac{\partial J(\theta)}{\partial \theta_i}\Big|_{\theta=\theta^{(k)}}$$

- η is a tunable parameter, which controls the amount of update performed at each step

Gradient descent and ERM

In Machine learning, minimization of the Empirical Risk is performed, hence gradient descent takes the form, at the k-step, of a gradient evaluation

$$\begin{aligned} J'(\theta) \mid_{\theta=\theta^{(k)}} &= \frac{\partial}{\partial \theta_i} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},t)\in\mathcal{T}} L(h_\theta(\mathbf{x}), t)|_{\theta=\theta^{(k)}} \\ &= \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x},t)\in\mathcal{T}} \frac{\partial}{\partial \theta_i} L(h_\theta(\mathbf{x}), t)|_{\theta=\theta^{(k)}} \end{aligned}$$

followed by the update

$$\theta_i^{(k+1)} = \theta_i^{(k)} - \eta J'(\theta) \mid_{\theta=\theta^{(k)}}$$

This is called *batch gradient descent*: observe that, at each step, all items in the training set must be considered

Stochastic gradient descent

Batch gradient descent can be modified by performing the update, at each step, on the basis of the evaluation at a single item $\mathbf{x}_j, t_j$ of the training set.

For single parameters,

$$J'(\theta) \mid_{\theta=\theta^{(k)}} = \frac{\partial}{\partial \theta_i} L(h_\theta(\mathbf{x}_j), t_j)|_{\theta=\theta^{(k)}}$$

followed again by the update

$$\theta_i^{(k+1)} = \theta_i^{(k)} - \eta J'(\theta) \mid_{\theta=\theta^{(k)}}$$

Mini-batch gradient descent

An intermediate case is the one when a subset B_r of the items in the training is considered at each step for gradient evaluation

$$J'(\theta) \mid_{\theta=\theta^{(k)}} = \frac{1}{|B_r|} \sum_{(\mathbf{x},t)\in B_r} \frac{\partial}{\partial \theta_i} L(h_\theta(\mathbf{x}), t)|_{\theta=\theta^{(k)}}$$

This is called *mini-batch gradient descent*.

Observe that the size $|B_r|$ of mini-batches is itself a tunable parameter

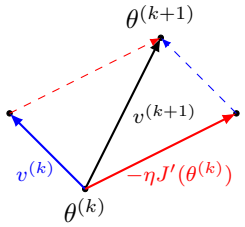
Momentum gradient descent

- Based on a physical interpretation of the optimization process: a body of mass $m = 1$ is moving on the surface of a cost function $J(\theta)$, with potential energy $U(\theta) = \eta J(\theta)$ and weight force (or acceleration, since $m = 1$) $F(\theta) = -\nabla U(\theta) = -\eta J'(\theta)$, at any point θ
- In gradient descent, the movement of the body is determined by the acceleration at that point, that is by the gradient $J'(\theta)$
- In momentum gradient descent, the velocity $v(\theta)$ of the body is considered: the movement of the body is determined by the velocity, that is,

$$\theta^{(k+1)} = \theta^{(k)} + v^{(k+1)}$$

with the velocity changing as determined by the acceleration

$$v^{(k+1)} = v^{(k)} - \eta J'(\theta) \mid_{\theta=\theta^{(k)}}$$



Momentum gradient descent

This results into

$$\begin{aligned} v^{(k+1)} &= -\eta J'(\theta) \mid_{\theta=\theta^{(k)}} + v^{(k)} = -\eta J'(\theta) \mid_{\theta=\theta^{(k)}} - \eta J'(\theta) \mid_{\theta=\theta^{(k-1)}} + v^{(k-1)} = \cdots \\ &= -\eta \sum_{i=0}^{k} J'(\theta) \mid_{\theta=\theta^{(i)}} + v^{(0)} \\ \theta^{(k+1)} &= \theta^{(k)} + v^{(k+1)} = \theta^{(k)} - \eta \sum_{i=0}^{k} J'(\theta) \mid_{\theta=\theta^{(i)}} + v^{(0)} \end{aligned}$$

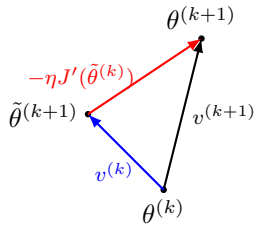
Note: an additional tuning parameter $\lambda \in [0, 1]$ can be introduced by applying the velocity update

$$v^{(k+1)} = \lambda v^{(k)} - \eta J'(\theta) \mid_{\theta=\theta^{(k)}}$$

Nesterov gradient descent

- The same approach of momentum gradient descent is applied, with the gradient estimation performed not at the current point $\theta^{(k)}$, but approximately at the next point $\theta^{(k+1)}$
- The approximation derives by considering $\tilde{\theta}^{(k)} = \theta^{(k)} + \gamma v^{(k)}$ instead of $\theta^{(k+1)}$
- The updates of v and θ are considered in advance with respect to momentum GD

$$\begin{aligned} \tilde{\theta}^{(k)} &= \theta^{(k)} + \gamma v^{(k)} \\ v^{(k+1)} &= \gamma v^{(k)} - \eta J'(\tilde{\theta}^{(k)}) \\ \theta^{(k+1)} &= \theta^{(k)} + v^{(k+1)} \end{aligned}$$



Dynamically updating the learning rate

Learning rate is a crucial parameter for SGD

- Too large: overshoots local minimum, loss increases
- Too small: makes very slow progress, can get stuck
- Good learning rate: makes steady progress toward local minimum

In practice: gradually decrease of the learning rate

- Step decay: periodically (every few epochs) decay η by a factor 2
- Exponential decay: $\eta^{(k)} = \eta^{(0)} e^{-\alpha k}$
- $1/t$ decay: $\eta^{(k)} = \dfrac{\eta^{(0)}}{1 + \alpha k}$

Extension: update η by monitoring the learning process

Adagrad

- In Adagrad, different learning rates are applied to the different parameters θ_i: larger gradients in the preceding steps results into smaller rates
- in particular, the learning rate of θ_j at step k is defined as

$$\eta_j^{(k)} = \frac{\eta}{\sqrt{G_{j,k} + \varepsilon}}$$

where η is a constant, $G_{j,k} = \sum_{i=0}^{k} g_{j,i}^2$ is the sum of the squared past gradients $g_{j,i} = \left.\frac{\partial J(\theta, X)}{\partial \theta_j}\right|_{\theta=\theta^{(i)}}$ and ε is a small *smoothing* constant, to deal with possible null denominators

- The update of θ_j at step $k+1$ is then

$$\theta_j^{(k+1)} = \theta_j^{(k)} - \frac{\eta}{\sqrt{G_{j,k} + \varepsilon}} g_{j,k}$$

- Learning rates decrease at each step, with the ones associated to parameters which had large gradients in the past decreasing more: that is, parameters which tended to large variation at each step are more "pushed" towards a stable value

Adadelta

- In Adadelta, past gradients are considered with a decreasing relevance of long past ones:
- A (decay) is applied my means of a coefficient $0 < \gamma < 1$

$$G_{j,k} = \gamma G_{j,k-1} + (1 - \gamma) g_{j,k}^2$$

Other adaptive methods have been introduced:

- RMSprop
- Adam

Second order methods

- Maxima (or minima) of $J(\theta)$ can be found by searching points where the gradient (all partial derivatives) zero
- Any iterative method to compute zeros of a function (such as Newton-Raphson) can then be applied on the gradient $J'(\theta)$
- Such methods rely on the gradient of the function considered, hence, the gradient of $J''(\theta) = \nabla J'(\theta)$ (that is, the Hessian) must be computed

$$J''_{ij}(\theta) = H_{ij}(J(\theta)) = \frac{\partial^2 J(\theta)}{\partial\theta_i \partial\theta_j}$$

- At each step, the following iteration is applied (in case Newton-Raphson method is used)

$$\theta^{(k+1)} = \theta^{(k)} - \left(J''(\theta)^{-1} J'(\theta)\right)\Big|_{\theta_{(k)}}$$