

Neural Networks Backpropagation General Gradient Descent

These notes are under construction

Now we consider regression of the following more general form.

$$\begin{aligned}w^* &= \operatorname{argmin}_w \mathcal{L}(w) \\ \mathcal{L}(w) &= \sum_{t=1}^N L(y_t, f_w(x_t)) + \lambda R(w)\end{aligned}\tag{1}$$

Here we are interested in the case where $f_w(x)$ is allowed to be *nonlinear* in the weight vector w . We consider the problem of finding a local optimum of (1) by gradient descent. Gradient descent iteratively updates w replacing w_t by w_{t+1} using the following update equation where η_t is a learning rate that typically declines with t .

$$\begin{aligned}w_{t+1} &= w_t - \eta_t \nabla \mathcal{L} \\ w_{j,t+1} &= w_{j,t} - \eta_t \frac{\partial \mathcal{L}}{\partial w_j}\end{aligned}\tag{2}$$

Note that here we are minimizing \mathcal{L} so we want to move in the opposite direction from the gradient. We add the gradient, rather than subtract, when we are maximizing (gradient ascent) rather than minimizing (gradient descent).

1 Neural Networks

A neural network is a particular kind of function $f_w(x)$ inspired by neurons in the brain. We assume a set of “neurons” or “units” f^1, f^2, \dots, f^k . For a given sensory input x (perhaps an image on the retina) each unit produces a response $f^j(x) \in (0, 1)$. Some units are connected directly to the sensory input while other units compute their response by combining the responses of earlier units. More specifically, each function $f^j(x)$ is computed as follows.

$$f^j(x) = \Phi_j(x) \text{ when } j \text{ is an input unit} \quad (3)$$

$$f^j(x) = s \left(\sum_{h \in INPUTS(j)} w_{h,j} f^h(x) \right) \text{ for } j \text{ an internal unit} \quad (4)$$

$$s(z) = \frac{1}{1 + e^{-z}} \quad (5)$$

We will assume that there is a “constant one” unit f^u with $f^u(x) = 1$ for all x . Here $s(z)$ is a (nonlinear) sigmoidal function. We can think of $s(z)$ as a soft threshold function — if $z \gg 1$ then $s(z) \approx 1$ and if $z \ll -1$ then $s(z) \approx 0$. This allows an arbitrary constant to be added in the linear combination inside the sigmoid. This allows each unit to compare a weighted sum of inputs to an arbitrary constant. So each unit takes a weighted sum of its inputs and produces an output if the input sum exceeds a threshold. This is similar to what actual neurons appear to do.

Independent of any relation to actual neurons, neural networks are an important model of computation. Neural networks can be viewed as a generalization of Boolean circuits. To compute disjunction it suffices to compare ten times the sum of the inputs to the threshold 5. To compute negation it suffices to compare negative ten times the input to the threshold five. Conjunction can be expressed directly or as a combination of negation and disjunction using deMorgan’s law. These “gates” will produce values very close to 0 and 1 even if the original inputs are not exactly 0 or 1.

Neural networks of a given number of units are much more expressive than a logic circuit of the same number of gates — making the same computation with floating point numbers takes many more gates. So it makes sense to consider neural networks as a model of computation in machine learning independent of any similarity to actual neurons.

2 Back Propagation

3 Optimization Methods: Repeated Line Searches

Again consider the optimization problem $w^* = \operatorname{argmin}_w f(w)$.

Consider the following update

$$w_j = \operatorname{argmin}_w f(w_1, \dots, w_{j-1}, w, w_{j+1}, \dots, w_d)$$

This one-dimensional optimization problem optimizes w_j while holding the other weights fixed. We can do this for each j and then, because the weights have

changed, we may need to repeat the process until all weights are simultaneously at optimal values.

4 Conjugate Gradient

Repeated line searches work well when the following condition is satisfied.

$$\frac{\partial^2 f}{\partial w_j \partial w_k} = 0 \text{ for } i \neq j$$

Intuitively this means that w_j remains optimal as we change w_k . In general this condition will not be satisfied. However, one can always find a coordinate system in which this is satisfied. The coordinates are the eigenvectors of the Hessian matrix H .

$$H_{j,k} = \frac{\partial^2 f}{\partial w_j \partial w_k}$$

These eigenvectors have the property that the change in the derivative of f in the direction of the eigenvector is along the eigenvector and hence does not change the derivative along other (orthogonal) eigenvectors. The conjugate gradient method does line searches along the “conjugate” directions given by the eigenvectors of the Hessian. Details can be found in Numerical Recipes <http://www.nr.com>.

5 Newton-Raphson

In the Newton-Raphson method we try to find the optimum by finding the point where the gradient is zero. We can approximate the gradient as follows. Let G be the gradient and H the Hessian.

$$G_j = \frac{\partial f}{\partial w_j}$$
$$H_{j,k} = \frac{\partial^2 f}{\partial w_j \partial w_k}$$

Both G and H are function of w so we can write, for example, $G_j(w)$ for the j th component of the gradient at point w . We want to find a point w where the gradient vector $G(w) = 0$. We will iteratively replace w_t by w_{t+1} . We will approximate $G(w_{t+1})$ as follows where G is $G(w_t)$ and H is $H(w_t)$.

$$G(w_{t+1}) \approx G + H(w_{t+1} - w_t)$$

Solving for $G(w_{t+1}) = 0$ gives the following.

$$w_{t+1} = w_t - H^{-1}G$$

This is the Newton-Raphson update rule.

6 Naive Gradient Descent

$$\begin{aligned} w_{j,t+1} &= w_{j,t} + \beta_t \sum_{i=1}^n \frac{\partial \ln P(y_i | x_i, w)}{\partial w_j} \\ &= w_{j,t} + \beta_t \sum_{i=1}^n [F_j(x_i, y_i) - \mathbb{E}_{y \sim P(y|x_i, w_t)} [F_j(x_i, y)]] \end{aligned}$$

7 On Line Gradient Descent

Assume that we have an infinite labeled data set $\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \dots$. We can always repeat a finite data set to generate an infinitely repeating infinite data set.

$$\begin{aligned} w_{j,t+1} &= w_{j,t} + \gamma_t \frac{\partial \ln P(y_t | x_t, w)}{\partial w_j} \\ &= w_{j,t} + \gamma_t [F_j(x_t, y_t) - \mathbb{E}_{y \sim P(y|x_t, w_t)} [F_j(x_t, y)]] \end{aligned}$$

8 Perception Algorithm

Again assume infinite labeled data set $\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \dots$

$$y^*(w, x) \equiv \operatorname{argmax}_y P(y|x, w)$$

$$w_{j,t+1} = w_{j,t} + \gamma_t [F_j(x_t, y_t) - F_j(x_t, y^*(x_t, w_t))]$$