Neural Networks from the Ground Up

Mehul Motani

September 2020

1 The neural network optimization problem

Consider a training dataset with features $\mathbf{x}$ and labels y. You can think of $\mathbf{x}$ as a matrix where the rows are samples/instances and the columns are features. So if we have M features, then the i_{th} sample is represented as $\mathbf{x}_i = \{x_{i1}, x_{i2} \cdots, x_{iM}\}$ and has label y_i. If we have N samples, the label vector is $y = \{y_1, y_2, \cdots, y_N\}^\intercal$. Think of the neural network as a black box which takes in $\mathbf{x}_i$ and outputs an approximation $\hat{y}_i$ (see Fig. 1).

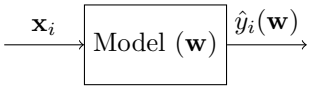

Figure 1: Neural network as a black box

The error between the true label y_i and the approximation $\hat{y}_i$ is defined as $E_i = L(y_i, \hat{y}_i)$ for some loss function L. For a regression problem, one can use the squared error loss: $L(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^2$. For a classification problem, one can use the cross-entropy loss (which measures the distance between distributions). For example, if we have a binary classification problem, then the cross-entropy loss is $L(y_i, \hat{y}_i) = y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$.

Suppose that the neural network model is characterized by K parameters $\mathbf{w} = \{w_1, w_2, \cdots, w_K\}$. Then the output $\hat{y}_i = \hat{y}_i(\mathbf{w})$ and the error is $E_i(\mathbf{w}) = L(y_i, \hat{y}_i(\mathbf{w}))$. The total error (over all N samples) is $E(\mathbf{w}) = \sum_{i=1}^{N} E_i(\mathbf{w})$. We are interested in minimizing the total error, which means finding the $\mathbf{w}$ which minimizes $E(\mathbf{w})$. The optimal $\mathbf{w}$ is given by

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w}) = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} E_i(\mathbf{w}) = \arg\min_{\mathbf{w}} \sum_{i=1}^{N} L(y_i, \hat{y}_i(\mathbf{w})). \tag{1}$$

The optimization problem in equation (1) is the core problem to be solved in training a neural network. What we mean by training a neural network is to find the optimal values for $\mathbf{w}$. The optimization in (1) can be solved in many ways.

One of the ways to solve the problem posed in (1) is to use an algorithm called gradient descent. Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. Gradient descent is also known as steepest descent. If, instead, one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent. (Description taken from [1]).

Gradient descent requires computing gradients. In vector calculus, the gradient is a multi-variable generalization of the derivative [2]. The algorithm called backpropagation is an efficient way to compute gradients using repeated application of the chain rule for derivatives [3]. We provide a review of gradient descent in Section 3 and some mathematical background for calculus in Section 5. In the next section, we see how a multilayer perceptron can be viewed as the compostion of functions.

2 Neural networks as function composition

Consider the two hidden layer multilayer perceptron shown in Fig. 2.

- Suppose there are D_0 nodes in the input layer (layer 0), D_1 nodes in hidden layer 1, D_2 nodes in hidden layer 2, and D_3 nodes in the output layer ($D_3 = 1$ in Fig. 2).
- The inputs $\mathbf{x} \in \mathbb{R}^{D_0}$ and the output $y \in \mathbb{R}$.
- The weights between hidden layer i and layer $i - 1$ are represented by the matrix $\mathbf{A}_i \in \mathbb{R}^{D_{i-1}} \times \mathbb{R}^{D_i}$.

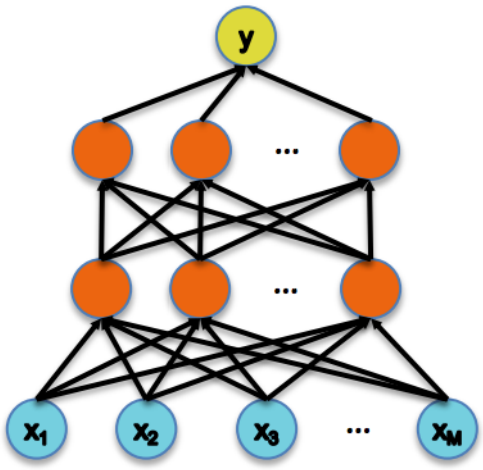

Figure 2: MLP with two hidden layers

- Denote the nonlinear activation function by $\sigma(\mathbf{a})$ and assume it is applied elementwise, i.e., $\sigma(\mathbf{a}) = [\sigma(a_1), \sigma(a_2), \cdots, \sigma(a_K)]^\intercal$. As an example, σ could be the sigmoid function, i.e., $\sigma(a) = \frac{1}{1+\exp^{-a}}$

Then the output of the neural network can be written as $\hat{y} = \sigma(\mathbf{A}_3^\intercal \sigma(\mathbf{A}_2^\intercal \sigma(\mathbf{A}_1^\intercal \mathbf{x})))$.

3 Gradient Descent

The following is taken directly from [1]. Gradient descent is based on the observation that if the multi-variable function $F(\mathbf{x})$ is defined and differentiable in a neighborhood of a point $\mathbf{a}$, then $F(\mathbf{x})$ decreases fastest if one goes from $\mathbf{a}$ in the direction of the negative gradient of F at $\mathbf{a}$, $-\nabla F(\mathbf{a})$. It follows that, if $\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$ for $\gamma \in \mathbb{R}_+$ small enough, then $F(\mathbf{a_n}) \geq F(\mathbf{a_{n+1}})$. In other words, the term $\gamma \nabla F(\mathbf{a})$ is subtracted from $\mathbf{a}$ because we want to move against the gradient, toward the minimum.

With this observation in mind, one starts with a guess $\mathbf{x}_0$ for a local minimum of F, and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ such that $\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n),\ n \geq 0$.

We have a monotonic sequence $F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots$, so hopefully the sequence $(\mathbf{x}_n)$ converges to the desired local minimum.

With certain assumptions on the function F and particular choices of γ, convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case gradient descent can converge to the global solution.

This process is illustrated in Fig. 3. Here F is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of F is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient descent leads us to the bottom of the bowl, that is, to the point where the value of the function F is minimal.

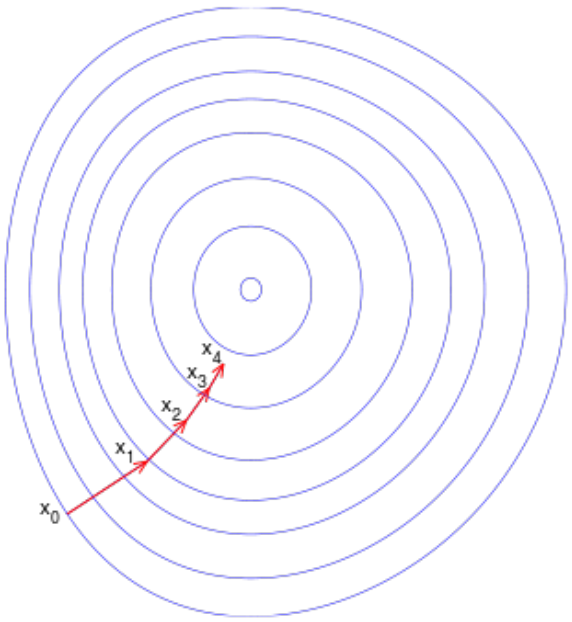

Figure 3: Gradient Descent [1]

4 Training a Neural Network via Backpropagation

The backpropagation algorithm [4] is used to train a neural network through a method called the chain rule for partial derivatives. In other words, backpropagation is a technique to efficiently compute gradients (or vector of partial derivatives) in a systematic manner. Specifically, after each forward pass through the network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases).

5 Calculus Review

We review some basic ideas from calculus. We start with derivatives [5].

Differentiation is the action of computing a derivative. The derivative of a function $y = f(x)$ of a variable x is a measure of the rate at which the value y of the function changes with respect to the change of the variable x. It is called the derivative of f with respect to x. If x and y are real numbers, and if the graph of f is plotted against x, the derivative is the slope of this graph at each point. In other words, the derivative of y with respect to x at a is, geometrically, the slope of the tangent line to the graph of f at $(a, f(a))$.

The slope of the approximately tangent line is

$$m = \frac{\Delta y}{\Delta x} = \frac{\Delta f(a)}{\Delta a} = \frac{f(a+h) - f(a)}{(a+h) - (a)} = \frac{f(a+h) - f(a)}{h}.$$

Passing from an approximation to an exact answer is done using a limit. This limit is defined to be the derivative of the function f at a:

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}.$$

When the limit exists, f is said to be differentiable at a. Here $f'(a)$ is one of several common notations for the derivative. From this definition it is obvious that a differentiable function f is increasing if and only if its derivative is positive, and is decreasing iff its derivative is negative. This fact is used extensively when analyzing function behavior, e.g. when finding local extrema.

Proposition 1. *Chain Rule for Derivatives*
If g is a function that is differentiable at a point c and f is a function that is differentiable at $g(c)$, then the composite function $(f \circ g)(x) = f(g(x))$ is differentiable at c, and the derivative is $(f \circ g)'(c) = f'(g(c)) \cdot g'(c)$. If $u = g(x)$ and $y = f(u) = f(g(x))$, then this can also be written as: $\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$.

Proof. Let $u = g(x)$ and $y = f(u) = f(g(x))$. Choosing an infinitesimal $\Delta x \neq 0$, we compute the corresponding $\Delta u = g(x + \Delta x) - g(x)$ and the corresponding $\Delta y = f(u + \Delta u) - f(u)$. Then $\frac{\Delta y}{\Delta x} = \frac{\Delta y}{\Delta u} \frac{\Delta u}{\Delta x}$. Taking limits we get $\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$. □

Remark 1. *We can extend the chain rule to an arbitrary number of compositions. Given n functions $f_1, \ldots, f_n$ with the composite function $y = f_1 \circ (f_2 \circ \cdots (f_{n-1} \circ f_n))(x) = f_1(f_2(\cdots f_n(x))$. If each function f_i is differentiable at its immediate input, then the composite function is also differentiable by the repeated application of Chain Rule: $\frac{dy}{dx} = \frac{df_1}{dx} = \frac{df_1}{df_2} \frac{df_2}{df_3} \cdots \frac{df_n}{dx}$.*

Remark 2. *We can extend the chain rule for single variables to multiple variables [6]. Let $z = h(u, v)$, $u = f(x)$, and $v = g(x)$. Then $\frac{dz}{dx} = \frac{dh}{du} \frac{du}{dx} + \frac{dh}{dv} \frac{dv}{dx}$.*

Proposition 2. *Product Rule for Derivatives*
Let $f(x)$ and $g(x)$ be two differentiable functions of x, and let $y = f \cdot g$. Then $\frac{dy}{dx} = g \cdot \frac{df}{dx} + f \cdot \frac{dg}{dx}$.

Proof. This is a special case of the chain rule for multiple variables. Let $z = h(u, v) = u \cdot v$, $u = f(x)$, and $v = g(x)$. Then $\frac{dz}{dx} = \frac{dh}{du} \frac{du}{dx} + \frac{dh}{dv} \frac{dv}{dx} = v \cdot \frac{du}{dx} + u \cdot \frac{dv}{dx} = g \cdot \frac{df}{dx} + f \cdot \frac{dg}{dx}$. □

References

[1] Wikipedia. Gradient descent. `https://en.wikipedia.org/wiki/Gradient_descent`. [Online; accessed 19-Oct-2019].

[2] Wikipedia. Gradient. `https://en.wikipedia.org/wiki/Gradient`. [Online; accessed 19-Oct-2019].

[3] WikiPedia. Chain rule. `https://en.wikipedia.org/wiki/Chain_rule`. [Online; accessed 19-Oct-2019].

[4] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[5] WikiPedia. Deriatives. `https://en.wikipedia.org/wiki/Derivative`. [Online; accessed 19-Oct-2019].

[6] Paul Dawkins. Chain rule. `http://tutorial.math.lamar.edu/Classes/CalcIII/ChainRule.aspx`. [Online; accessed 19-Oct-2019].