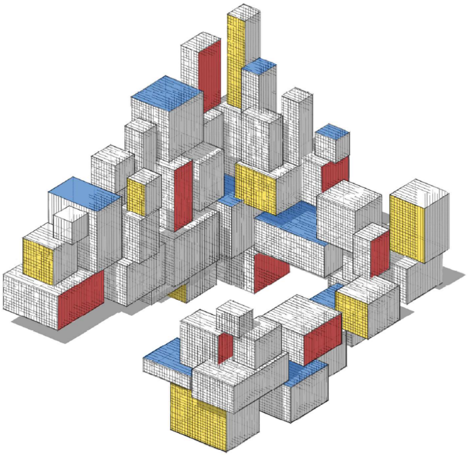
Back-Propagation



Purpose

In this lecture we discuss the training of neural networks through backpropagation.

Training Neural Networks

The training of neural networks is a major challenge that requires both ingenuity and much experimentation.

The algorithms for training neural networks with great depth are collectively referred to as deep learning methods.

One of the simplest and most effective methods for training is via steepest descent and its variations.

Steepest descent requires computation of the gradient with respect to all bias vectors and weight matrices.

Given the potentially large number of parameters (weight and bias terms) in a neural network, we need to find an efficient method to calculate this gradient.

Gradient Computations

Let $\boldsymbol{\theta} = \{\mathbf{W}_l, \boldsymbol{b}_l\}$ be a column vector of $\dim(\boldsymbol{\theta}) = \sum_{l=1}^{L}(p_{l-1}p_l + p_l)$ that collects all the weight parameters (numbering $\sum_{l=1}^{L} p_{l-1}p_l$) and bias parameters (numbering $\sum_{l=1}^{L} p_l$) of a multiple-layer network.

Consider the training loss:

$$\ell_\tau(\boldsymbol{g}(\cdot \,|\, \boldsymbol{\theta})) := \frac{1}{n}\sum_{i=1}^{n} \mathrm{Loss}(\boldsymbol{y}_i, \boldsymbol{g}(\boldsymbol{x}_i \,|\, \boldsymbol{\theta})).$$

Writing $C_i(\boldsymbol{\theta}) := \mathrm{Loss}(\boldsymbol{y}_i, \boldsymbol{g}(\boldsymbol{x}_i \,|\, \boldsymbol{\theta}))$ for short (using C for **<u>c</u>**ost), we have

$$\ell_\tau(\boldsymbol{g}(\cdot \,|\, \boldsymbol{\theta})) = \frac{1}{n}\sum_{i=1}^{n} C_i(\boldsymbol{\theta}), \tag{1}$$

so that obtaining the gradient of ℓ_τ requires computation of $\partial C_i/\partial\boldsymbol{\theta}$ for every i.

Gradient Computations

For activation functions of the form

$$\boldsymbol{S}_l(\boldsymbol{z}) = [S(z_{l,1}), \ldots, S(z_{l,p_l})]^\top, \quad l = 1, \ldots, L-1,$$

define $\mathbf{D}_l$ as the diagonal matrix with the vector of derivatives

$$\boldsymbol{S}'(\boldsymbol{z}) := [S'(z_{l,1}), \ldots, S'(z_{l,p_l})]^\top$$

down its main diagonal; that is,

$$\mathbf{D}_l := \mathrm{diag}(S'(z_{l,1}), \ldots, S'(z_{l,p_l})), \quad l = 1, \ldots, L-1.$$

The following theorem provides us with the formulas needed to compute the gradient of a typical $C_i(\boldsymbol{\theta})$.

Theorem: Gradient of Training Loss

For a given (input, output) pair $(\boldsymbol{x}, \boldsymbol{y})$, let $\boldsymbol{g}(\boldsymbol{x} \mid \boldsymbol{\theta})$ be the output of the feed-forward propagation, and let $C(\boldsymbol{\theta}) = \mathrm{Loss}(\boldsymbol{y}, \boldsymbol{g}(\boldsymbol{x} \mid \boldsymbol{\theta}))$ be an almost-everywhere differentiable loss function. Suppose $\{\boldsymbol{z}_l, \boldsymbol{a}_l\}_{l=1}^{L}$ are the vectors obtained during the feed-forward propagation ($\boldsymbol{a}_0 = \boldsymbol{x}, \boldsymbol{a}_L = \boldsymbol{g}(\boldsymbol{x} \mid \boldsymbol{\theta})$). Then, we have for $l = 1, \ldots, L$:

$$\frac{\partial C}{\partial \mathbf{W}_l} = \boldsymbol{\delta}_l \, \boldsymbol{a}_{l-1}^\top \quad \text{and} \quad \frac{\partial C}{\partial \boldsymbol{b}_l} = \boldsymbol{\delta}_l,$$

where $\boldsymbol{\delta}_l := \partial C / \partial \boldsymbol{z}_l$ is computed recursively for $l = L, \ldots, 2$:

$$\boldsymbol{\delta}_{l-1} = \mathbf{D}_{l-1} \mathbf{W}_l^\top \, \boldsymbol{\delta}_l \quad \text{with} \quad \boldsymbol{\delta}_L = \frac{\partial \boldsymbol{S}_L}{\partial \boldsymbol{z}_L} \frac{\partial C}{\partial \boldsymbol{g}}. \tag{2}$$

Proof

The scalar value C is obtained from the transitions

$$\underbrace{\boldsymbol{x}}_{\boldsymbol{a}_0} \to \underbrace{\mathbf{W}_1\, \boldsymbol{a}_0 + \boldsymbol{b}_1}_{\boldsymbol{z}_1} \to \underbrace{\boldsymbol{S}_1(\boldsymbol{z}_1)}_{\boldsymbol{a}_1} \to \underbrace{\mathbf{W}_2\, \boldsymbol{a}_1 + \boldsymbol{b}_2}_{\boldsymbol{z}_2} \to \underbrace{\boldsymbol{S}_2(\boldsymbol{z}_2)}_{\boldsymbol{a}_2} \to \cdots$$
$$\to \underbrace{\mathbf{W}_L\, \boldsymbol{a}_{L-1} + \boldsymbol{b}_L}_{\boldsymbol{z}_L} \to \underbrace{\boldsymbol{S}_L(\boldsymbol{z}_L)}_{\boldsymbol{a}_L} = \boldsymbol{g}(\boldsymbol{x}),$$

followed by the mapping $\boldsymbol{g}(\boldsymbol{x} \,|\, \boldsymbol{\theta}) \mapsto \mathrm{Loss}(\boldsymbol{y}, \boldsymbol{g}(\boldsymbol{x} \,|\, \boldsymbol{\theta}))$.

Using the chain rule of differentiation (see appendix), we have

$$\boldsymbol{\delta}_L = \frac{\partial C}{\partial \boldsymbol{z}_L} = \frac{\partial \boldsymbol{g}(\boldsymbol{x})}{\partial \boldsymbol{z}_L} \frac{\partial C}{\partial \boldsymbol{g}(\boldsymbol{x})} = \frac{\partial \boldsymbol{S}_L}{\partial \boldsymbol{z}_L} \frac{\partial C}{\partial \boldsymbol{g}}.$$

Proof (cont.)

Recall that the vector/vector derivative of a linear mapping $\boldsymbol{z} \mapsto \mathbf{W}\boldsymbol{z}$ is given by $\mathbf{W}^\top$.

It follows that, since $\boldsymbol{z}_l = \mathbf{W}_l\, \boldsymbol{a}_{l-1} + \boldsymbol{b}_l$ and $\boldsymbol{a}_l = \boldsymbol{S}(\boldsymbol{z}_l)$, the chain rule gives

$$\frac{\partial \boldsymbol{z}_l}{\partial \boldsymbol{z}_{l-1}} = \frac{\partial \boldsymbol{a}_{l-1}}{\partial \boldsymbol{z}_{l-1}} \frac{\partial \boldsymbol{z}_l}{\partial \boldsymbol{a}_{l-1}} = \mathbf{D}_{l-1}\mathbf{W}_l^\top.$$

Hence, the recursive formula (2):

$$\boldsymbol{\delta}_{l-1} = \frac{\partial C}{\partial \boldsymbol{z}_{l-1}} = \frac{\partial \boldsymbol{z}_l}{\partial \boldsymbol{z}_{l-1}} \frac{\partial C}{\partial \boldsymbol{z}_l} = \mathbf{D}_{l-1}\mathbf{W}_l^\top \boldsymbol{\delta}_l, \quad l = L, \ldots, 3, 2.$$

Proof (cont.)

Using the $\{\boldsymbol{\delta}_l\}$, we can now compute the derivatives with respect to the weight matrices and the biases.

In particular, applying the "scalar/matrix" differentiation rule to $\boldsymbol{z}_l = \mathbf{W}_l\, \boldsymbol{a}_{l-1} + \boldsymbol{b}_l$ gives:

$$\frac{\partial C}{\partial \mathbf{W}_l} = \frac{\partial C}{\partial \boldsymbol{z}_l}\frac{\partial \boldsymbol{z}_l}{\partial \mathbf{W}_l} = \boldsymbol{\delta}_l\, \boldsymbol{a}_{l-1}^\top, \quad l = 1, \ldots, L$$

and

$$\frac{\partial C}{\partial \boldsymbol{b}_l} = \frac{\partial \boldsymbol{z}_l}{\partial \boldsymbol{b}_l}\frac{\partial C}{\partial \boldsymbol{z}_l} = \boldsymbol{\delta}_l, \quad l = 1, \ldots, L.$$

This completes the proof. □

Back-Propagation

From the theorem we can see that for each pair $(\boldsymbol{x}, \boldsymbol{y})$ in the training set, we can compute the gradient $\partial C/\partial \boldsymbol{\theta}$ in a sequential manner, by computing $\boldsymbol{\delta}_L, \ldots, \boldsymbol{\delta}_1$. This procedure is called back-propagation.

Since back-propagation mostly involves simple matrix multiplication, it can be efficiently implemented using dedicated computing hardware such as graphical processor units (GPUs).

Note also that many "quadratic-time" matrix computations can be replaced with "linear-time" componentwise multiplication; e.g., multiplication of a vector with a diagonal matrix is equivalent to componentwise multiplication:

$$\underbrace{\mathbf{A}}_{\mathrm{diag}(\boldsymbol{a})}\boldsymbol{b} = \boldsymbol{a} \odot \boldsymbol{b}.$$

Consequently, we can write $\boldsymbol{\delta}_{l-1} = \mathbf{D}_{l-1}\mathbf{W}_l^\top \boldsymbol{\delta}_l$ as:
$\boldsymbol{\delta}_{l-1} = \boldsymbol{S}'(\boldsymbol{z}_{l-1}) \odot \mathbf{W}_l^\top \boldsymbol{\delta}_l,\ l = L, \ldots, 3, 2.$

Algorithm 1: Computing the Gradient of a Typical $C(\boldsymbol{\theta})$

Input: Training example $(\boldsymbol{x}, \boldsymbol{y})$, weight matrices and bias vectors $\{\mathbf{W}_l, \boldsymbol{b}_l\}_{l=1}^L =: \boldsymbol{\theta}$, activation functions $\{\boldsymbol{S}_l\}_{l=1}^L$.

Output: The derivatives with respect to all weight matrices and bias vectors.

```
1  $\boldsymbol{a}_0 \leftarrow \boldsymbol{x}$
2  for $l = 1, \ldots, L$ do                                  // feed-forward
3      $\boldsymbol{z}_l \leftarrow \mathbf{W}_l\, \boldsymbol{a}_{l-1} + \boldsymbol{b}_l$
4      $\boldsymbol{a}_l \leftarrow \boldsymbol{S}_l(\boldsymbol{z}_l)$
5  $\boldsymbol{\delta}_L \leftarrow \frac{\partial \boldsymbol{S}_L}{\partial \boldsymbol{z}_L} \frac{\partial C}{\partial \boldsymbol{g}}$
6  $\boldsymbol{z}_0 \leftarrow \mathbf{0}$ // arbitrary assignment needed to finish the loop
7  for $l = L, \ldots, 1$ do                               // back-propagation
8      $\frac{\partial C}{\partial \boldsymbol{b}_l} \leftarrow \boldsymbol{\delta}_l$
9      $\frac{\partial C}{\partial \mathbf{W}_l} \leftarrow \boldsymbol{\delta}_l\, \boldsymbol{a}_{l-1}^\top$
10     $\boldsymbol{\delta}_{l-1} \leftarrow \boldsymbol{S}'(\boldsymbol{z}_{l-1}) \odot \mathbf{W}_l^\top \boldsymbol{\delta}_l$
11 return $\frac{\partial C}{\partial \mathbf{W}_l}$ and $\frac{\partial C}{\partial \boldsymbol{b}_l}$, $l = 1, \ldots, L$ and the value $\boldsymbol{g}(\boldsymbol{x}) \leftarrow \boldsymbol{a}_L$ (if needed)
```

Differentiability of the Activation Function

Note that for the gradient of $C(\boldsymbol{\theta})$ to exist at every point, we need the activation functions to be differentiable everywhere.

This is the case, for example, for the logistic activation function.

It is not the case for the ReLU function, which is differentiable everywhere, except at $z = 0$.

However, in practice, the kink of the ReLU function at $z = 0$ is unlikely to trip the back-propagation algorithm, because rounding errors and the finite-precision computer arithmetic make it extremely unlikely that we will need to evaluate the ReLU at precisely $z = 0$.

This is the reason why in the back-propagation theorem we merely required that $C(\boldsymbol{\theta})$ is almost-everywhere differentiable.

Saturation

In spite of its kink at the origin, the ReLU has an important advantage over the logistic function: While the derivative of the logistic function decays exponentially fast to zero as we move away from the origin, a phenomenon referred to as saturation, the derivative of the ReLU function is always unity for positive z.

Thus, for large positive z, the derivative of the logistic function does not carry any useful information, but the derivative of the ReLU can help guide a gradient optimization algorithm.

In this respect, the lack of saturation of the ReLU function for $z > 0$ makes it a desirable activation function for training a network via back-propagation.

Gradient of Training Loss

To obtain the gradient $\partial \ell_\tau / \partial \boldsymbol{\theta}$ of the training loss, we simply need to loop Algorithm 1 over all the n training examples, and take averages, as follows:

Algorithm 2: Computing the Gradient of the Training Loss

Input: Training set $\tau = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, weight matrices and bias vectors $\{\mathbf{W}_l, \boldsymbol{b}_l\}_{l=1}^L =: \boldsymbol{\theta}$, activation functions $\{\boldsymbol{S}_l\}_{l=1}^L$.

Output: The gradient of the training loss.

```
1 for $i = 1, \ldots, n$ do    // loop over all training examples
2     Run Algorithm 1 with input $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ to compute $\left\{\frac{\partial C_i}{\partial \mathbf{W}_l}, \frac{\partial C_i}{\partial \boldsymbol{b}_l}\right\}_{l=1}^L$
3 return $\frac{\partial C}{\partial \mathbf{W}_l} = \frac{1}{n}\sum_{i=1}^n \frac{\partial C_i}{\partial \mathbf{W}_l}$ and $\frac{\partial C}{\partial \boldsymbol{b}_l} = \frac{1}{n}\sum_{i=1}^n \frac{\partial C_i}{\partial \boldsymbol{b}_l}$, $l = 1, \ldots, L$
```

Multi-Logit Classification

The BP part of Algorithm 1 requires a formula for $\boldsymbol{\delta}_L = \frac{\partial \boldsymbol{S}_L}{\partial \boldsymbol{z}_L} \frac{\partial C}{\partial \boldsymbol{g}}$.

We need to specify both a loss function and an $\boldsymbol{S}_L$ that defines the output layer: $\boldsymbol{g}(\boldsymbol{x} \mid \boldsymbol{\theta}) = \boldsymbol{a}_L = \boldsymbol{S}_L(\boldsymbol{z}_L)$.

In the multi-logit classification of inputs $\boldsymbol{x}$ into p_L categories labeled $0, 1, \ldots, p_L - 1$, the output layer is defined via the softmax function:

$$\boldsymbol{S}_L : \boldsymbol{z}_L \mapsto \frac{\exp(\boldsymbol{z}_L)}{\sum_{k=1}^{p_L} \exp(z_{L,k})}.$$

In other words, $\boldsymbol{g}(\boldsymbol{x} \mid \boldsymbol{\theta})$ is a probability vector such that its $(y+1)$-st component $g_{y+1}(\boldsymbol{x} \mid \boldsymbol{\theta}) = g(y \mid \boldsymbol{\theta}, \boldsymbol{x})$ is the estimate or prediction of the true conditional probability $f(y \mid \boldsymbol{x})$.

Multi-Logit Classification

Combining the softmax output with the cross-entropy loss, yields:

$$\begin{aligned}\text{Loss}(f(y\,|\,\boldsymbol{x}), g(y\,|\,\boldsymbol{\theta},\boldsymbol{x})) &= -\ln g(y\,|\,\boldsymbol{\theta},\boldsymbol{x}) \\ &= -\ln g_{y+1}(\boldsymbol{x}\,|\,\boldsymbol{\theta}) \\ &= -z_{y+1} + \ln \textstyle\sum_{k=1}^{p_L} \exp(z_k).\end{aligned}$$

Hence, we obtain the vector $\boldsymbol{\delta}_L$ with components ($k = 1, \ldots, p_L$)

$$\delta_{L,k} = \tfrac{\partial}{\partial z_k}\left(-z_{y+1} + \ln \textstyle\sum_{k=1}^{p_L} \exp(z_k)\right) = g_k(\boldsymbol{x}\,|\,\boldsymbol{\theta}) - \mathbb{I}\{y = k-1\}.$$

Note that we can remove a node from the final layer of the multi-logit network, because $g_1(\boldsymbol{x}\,|\,\theta)$ (which corresponds to the $y = 0$ class) can be eliminated, using the fact that $g_1(\boldsymbol{x}\,|\,\theta) = 1 - \sum_{k=2}^{p_L} g_k(\boldsymbol{x}\,|\,\theta)$.

Multi-Output Regression

In nonlinear multi-output regression the output function $\boldsymbol{S}_L$ is typically of the form

$$\boldsymbol{S}_l(\boldsymbol{z}) = [S(z_{l,1}), \ldots, S(z_{l,p_l})]^\top, \quad l = 1, \ldots, L-1,$$

so that $\partial \boldsymbol{S}_L / \partial \boldsymbol{z} = \mathrm{diag}(S_L'(z_1), \ldots, S_L'(z_{p_L}))$. Combining the output $\boldsymbol{g}(\boldsymbol{x} \,|\, \boldsymbol{\theta}) = \boldsymbol{S}_L(\boldsymbol{z}_L)$ with the squared-error loss yields:

$$\mathrm{Loss}(\boldsymbol{y}, \boldsymbol{g}(\boldsymbol{x} \,|\, \boldsymbol{\theta})) = \|\boldsymbol{y} - \boldsymbol{g}(\boldsymbol{x} \,|\, \boldsymbol{\theta})\|^2 = \sum_{j=1}^{p_L} (y_j - g_j(\boldsymbol{x} \,|\, \boldsymbol{\theta}))^2.$$

Hence, line 5 in Algorithm 1 simplifies to:

$$\boldsymbol{\delta}_L = \frac{\partial \boldsymbol{S}_L}{\partial \boldsymbol{z}} \frac{\partial C}{\partial \boldsymbol{g}} = \boldsymbol{S}_L'(\boldsymbol{z}_L) \odot 2(\boldsymbol{g}(\boldsymbol{x} \,|\, \boldsymbol{\theta}) - \boldsymbol{y}).$$