CS 5600/6600: F20: Intelligent Systems

Gradient Descent and Backpropagation: Part 02

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

Stochastic Gradient Descent

Computing ANN's output with Matrices

Backpropagation
  Four Backprop Equations

Backpropagation Example

A Challenge for the Gradient Descent Algorithm

Let's take a look at the gradient descent rule $C(w, b)$ one more time.

$$C(w, b) = \frac{1}{n} \sum_x \frac{||y(x) - a||^2}{2} = \frac{1}{n} \sum_x C_x,$$

where

$$C_x = \frac{||y(x) - a||^2}{2}.$$

In practice, to compute ∇C we have to compute ∇C_x for each input x separately and then average them as $\nabla C = \frac{1}{n} \sum_x \nabla C_x$. It gets expensive when we have lots of training inputs.

Stochastic Gradient Descent

To speed up learning, we can use *stochastic gradient descent*. The idea is to estimate ∇C by computing ∇C_x for a small random sample of training inputs and computing their average.

Stochastic Gradient Descent

Let's make the ideas of stochastic gradient descent more precise. Let $X_1, X_2, ..., X_m$ be a small number of m training inputs. This sample is called *mini-batch*. We expect that if m is sufficiently large (but still small compared to the overall number of samples!), the average of ∇C_{X_j} is approximately equal to the average over all ∇C_x:

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}$$

or, more succinctly,

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}.$$

Modifying the Update Rules for Stochastic Gradient Descent

We need to modify the update rules for stochastic gradient descent as follows:

$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k},$$

$$b_j \rightarrow b_j' = b_j - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_j}.$$

Outline

Stochastic Gradient Descent

Computing ANN's output with Matrices

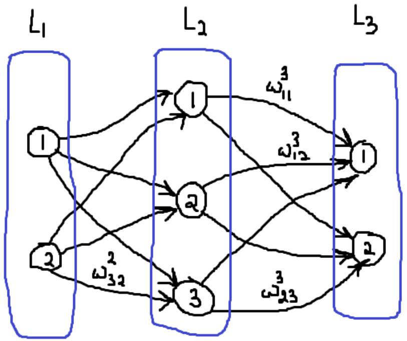
Backpropagation

Four Backprop Equations

Backpropagation Example

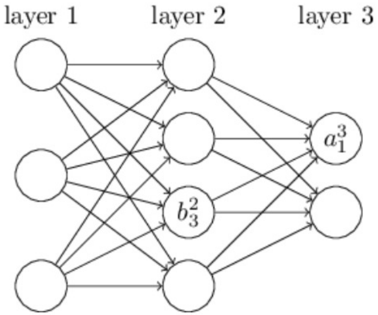
Connection Weights

w^l_{jk} is the weight from neuron k in layer $l-1$ to neuron j in layer l.



Neuron Activations and Biases

b^l_k is the bias of neuron k in layer l; a^l_k is the activation of neuron k in layer l.



Hadamard/Schur Product

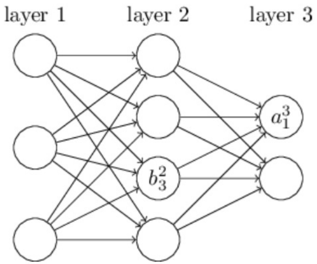
The Hadamard product, $v_1 \odot v_2$, is the elementwise product of two vectors.

$$\begin{bmatrix} 4 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 5 \\ 7 \end{bmatrix} = \begin{bmatrix} 20 \\ 14 \end{bmatrix}$$

Computing Neuron Activations

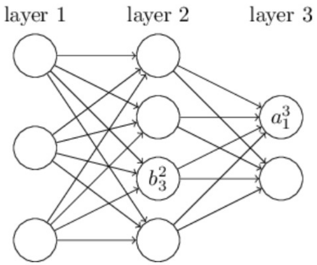
The activation of neuron j in layer l is related to the activations in layer $l-1$ as follows (the sum is over all neurons k in layer $l-1$):

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$



Example

$$a_1^3 = \sigma\left(\sum_{k=1}^{4} w_{1k}^3 a_k^2 + b_1^3\right) = \sigma\left(w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + w_{13}^3 a_3^2 + w_{14}^3 a_4^2 + b_1^3\right).$$



Activation Vector for Layer l

Let w^l be the weight matrix for layer l where the entry (j, k) is w^l_{jk}. Let b^l be the bias vector for layer l. Let a^l be the activation vector for layer l.

$$a^l = \sigma(w^l a^{l-1} + b^l).$$

The quantity $z^l = w^l a^{l-1} + b^l$ is called the *weighted input* to the neurons in layer l.

Weighted Input

$$z^l = \begin{bmatrix} z_1^l \\ z_2^l \\ \cdot \\ \cdot \\ \cdot \\ z_m^l \end{bmatrix} = \begin{bmatrix} \sum_k w_{1k}^l a_k^{l-1} + b_1^l \\ \sum_k w_{2k}^l a_k^{l-1} + b_2^l \\ \cdot \\ \cdot \\ \cdot \\ \sum_k w_{mk}^l a_k^{l-1} + b_m^l \end{bmatrix}.$$

Activation Vector for Layer l As Function of Weighted Input

$$a^l = \sigma(z^l)$$

Outline

Stochastic Gradient Descent

Computing ANN's output with Matrices

Backpropagation

Four Backprop Equations

Backpropagation Example

Learning Representation by Backpropagating Errors

The original backpropagation procedure was outlined in the seminal paper “Learning representations by backpropagating errors” by David Rumelhart, Geoffrey Hinton, and Ronald Williams (Nature 323, pp. 533-536, 9 October 1986).

Backpropagation is an algorithm for computing the gradient of a cost function in networks of neuron-like units.

Backpropagation can also be viewed as a learning procedure for networks of neuron-like units that repeatedly adjusts the weights of the connections in the network to minimize a measure of the difference between the actual output vector of the net and the target output vector.

Neuron Error

Backpropagation shows how changing the weights and biases in a network changes the cost function.

Backpropagation computes $\frac{\partial C}{\partial w_{jk}^l}$ and $\frac{\partial C}{\partial b_j^l}$.

To compute these two quantities we introduce a new quantity δ_j^l, which is called the error in neuron j in layer l.

Neuron Error

Consider neuron j in layer l. Suppose instead of outputing $\sigma(z_j^l)$ it computes $\sigma(z_j^l + \Delta z_j^l)$.

This change Δz_j^l propagates through layers in the network, which causes the overall cost to change by an amount $\frac{\partial C}{\partial z_j^l} \Delta z_j^l$.

The neuron's error is defined as

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$

δ^l is the vector of errors in layer l.

Equation 1: Error in Output Layer δ^L

The components of δ^L are given by

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$$

$\partial C / \partial a_j^L$ measures how fast the cost is changing as a function of the j-th output activation.

$\sigma'(z_j^L)$ measures how fast the activation function σ is changing at z_j^L.

Equation 1 in Matrix Form

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$$

We can write the above equation in matrix form:

$$\delta^L = \nabla_a C \odot \sigma'(z^L),$$

$\nabla_a C$ is a vector whose components are $\partial C / \partial a_j^L$. In the case of the quadratic cost the above equation becomes

$$\delta^L = (a^L - y) \odot \sigma'(z^L).$$

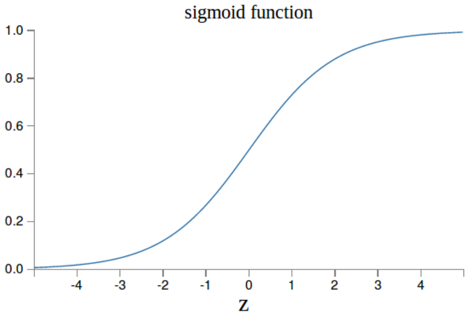
Equation 2: Error in Layer l as Function of Error in Layer $l+1$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

We can think of applying $(w^{l+1})^T$ to δ^{l+1} as moving the error backward to the network and then applying the Hadamard $\odot \sigma'(z^l)$ as moving the error back through the activation function in layer l.

Question on Equation 2

Consider the term $\sigma'(z_j^L)$. What happens to δ_j^l when $\sigma(z_j^L) \approx 0/1$?



Equation 3: Rate of Change of Cost with Respect to Bias

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

We can drop the subscripts and simply write

$$\frac{\partial C}{\partial b} = \delta.$$

Equation 4: Rate of Change of Cost with Respect to Weight

$$\frac{\partial C}{\partial w_{jk}^{l}} = a_k^{l-1}\delta_j^l,$$

where a_k^{l-1} is the input to the weight w_{jk}^l and δ_j^l is the error of the output of neuron j in layer l. We can drop the subscripts and simply write

$$\frac{\partial C}{\partial w} = a_{in}\delta_{out}.$$

Backprop Equations

EQ1) $$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

EQ2) $$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^L)$$

EQ3) $$\frac{\partial C}{\partial b_j^L} = \delta_j^l$$

EQ4) $$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Backprop Insight 1

A weight learns slowly if either the input neuron is low-activation or if the output neuron is either high- or low-activation (i.e., saturated).

Backprop Insight 2

Backprop equations can be used to design activation functions with specific learning properties.

Backpropagation Algorithm

1. **Input** x: Set the corresponding activation a^1 for the input layer.
2. **Feedforward**: For each $l \in (2, 3, ..., L)$, compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. **Output error** δ^L: Compute $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate error**: For each $l \in (L-1, L-2, ..., 2)$, compute $\delta^l = ((w^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)$.
5. **Output**: Compute the gradients of the cost function given by

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1} \delta^l_j \text{ and } \frac{\partial C}{\partial b^l_j} = \delta^l_j.$$

Combining Backpropagation with Gradient Descent

The backpropagation algorithm computes the gradient of the cost function for a single example C_x.

In practice, we combine backprop with a learning algorithm such as stochastic gradient descent where the gradient is computed for many training examples.

Combining Backprop with Gradient Descent

1. **Input**: a set of m training examples.

2. **For each training example** x, set the corresponding input activation $a^{x,1}$ and perform the following steps:

   - **Feedforward**: For each $l \in (2, 3, ..., L)$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
   - **Output error** $\delta^{x,L}$: Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
   - **Backpropagate error**: For each $l \in (L-1, L-2, ..., 2)$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.

3. **Gradient descent**: For each $l \in (L, L-1, ..., 2)$, update the weights and biases according to these rules:

$$w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T;$$

$$b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}.$$

Outline

Stochastic Gradient Descent

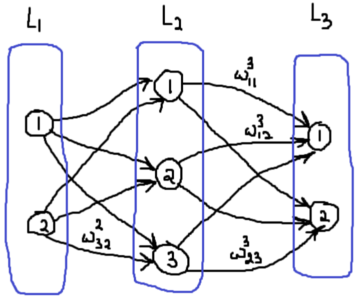
Computing ANN's output with Matrices

Backpropagation

- Four Backprop Equations

Backpropagation Example

Backpropagation Example: ANN



$$w_1 = \begin{bmatrix} -0.81 & -0.34 \\ -1.57 & -0.03 \\ 0.60 & -0.52 \end{bmatrix};\ w_2 = \begin{bmatrix} 0.51 & 0.65 & -0.74 \\ -1.42 & -0.89 & 2.02 \end{bmatrix}.$$

$$b_1 = \begin{bmatrix} 0.27 \\ 0.87 \\ 0.13 \end{bmatrix};\ b_2 = \begin{bmatrix} 0.33 \\ 0.65 \end{bmatrix}.$$

Backpropagation Example: Feedforward

$$z^l = w^l a^{l-1} + b^l; a^l = \sigma(z^l), l \in \{2, 3\}.$$

$$z^2 = w^2 a^1 + b^2; a^2 = \sigma(z^2)$$

$$z^3 = w^3 a^2 + b^3; a^3 = \sigma(z^3)$$

Backpropagation Example: Feedforward

$$z_2 = \begin{bmatrix} -0.81 & -0.34 \\ -1.57 & -0.03 \\ 0.60 & -0.52 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.27 \\ 0.87 \\ 0.13 \end{bmatrix} = \begin{bmatrix} -0.88 \\ -0.73 \\ 0.21 \end{bmatrix}$$

$$a_2 = \sigma(z^2) = \begin{bmatrix} 0.29 \\ 0.33 \\ 0.55 \end{bmatrix}$$

$$z_3 = \begin{bmatrix} 0.51 & 0.65 & -0.74 \\ -1.42 & -0.89 & 2.02 \end{bmatrix} \begin{bmatrix} 0.29 \\ 0.33 \\ 0.55 \end{bmatrix} + \begin{bmatrix} 0.33 \\ 0.65 \end{bmatrix} = \begin{bmatrix} 0.28 \\ 1.06 \end{bmatrix}$$

$$a_3 = \sigma(z^3) = \begin{bmatrix} 0.57 \\ 0.74 \end{bmatrix}$$

Backpropagation Example: Error

$$\delta^3 = (a^3 - y) \odot \sigma'(z^3)$$

$$= \left(\begin{bmatrix} 0.57 \\ 0.74 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) \odot \sigma'(z^3) = \begin{bmatrix} -0.1054 \\ -0.0492 \end{bmatrix};$$

$$\delta^2 = ((w^3)^T \delta^3) \odot \sigma'(z^2)$$

$$= \left(\begin{bmatrix} 0.51 & -1.42 \\ 0.65 & -0.89 \\ -0.74 & 2.02 \end{bmatrix} \times \begin{bmatrix} -0.1054 \\ -0.0492 \end{bmatrix}\right) \odot \sigma'(z^2) = \begin{bmatrix} 0.0033 \\ -0.0054 \\ -0.0053 \end{bmatrix}$$

Backpropagation Example: Output: Weight Gradients

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}\delta^l_j$$

$$\frac{\partial C}{\partial w^3_{jk}} = \begin{bmatrix} \frac{\partial C}{\partial w^3_{11}} = a^2_1\delta^3_1, & \frac{\partial C}{\partial w^3_{12}} = a^2_2\delta^3_1, & \frac{\partial C}{\partial w^3_{13}} = a^2_3\delta^3_1 \\ \frac{\partial C}{\partial w^3_{21}} = a^2_1\delta^3_2, & \frac{\partial C}{\partial w^3_{22}} = a^2_2\delta^3_2, & \frac{\partial C}{\partial w^3_{23}} = a^2_3\delta^3_2 \end{bmatrix}$$

$$= \begin{bmatrix} -0.031 & -0.034 & -0.0098 \\ -0.014 & -0.016 & -0.027 \end{bmatrix}$$

$$\frac{\partial C}{\partial w^2_{jk}} = \begin{bmatrix} \frac{\partial C}{\partial w^2_{11}} = a^1_1\delta^2_1, & \frac{\partial C}{\partial w^2_{12}} = a^1_2\delta^2_1 \\ \frac{\partial C}{\partial w^2_{21}} = a^1_1\delta^2_2, & \frac{\partial C}{\partial w^2_{22}} = a^1_2\delta^2_2 \\ \frac{\partial C}{\partial w^2_{21}} = a^1_1\delta^2_3, & \frac{\partial C}{\partial w^2_{22}} = a^1_2\delta^2_3 \end{bmatrix}$$

$$= \begin{bmatrix} 0.0033 & 0.0033 \\ -0.0054 & -0.0054 \\ -0.0053 & -0.0053 \end{bmatrix}$$

Backpropagation Example: Output: Bias Gradients

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial C}{\partial b_j^3} = \begin{bmatrix} \delta_1^3 \\ \delta_2^3 \end{bmatrix} = \delta^3$$

$$\frac{\partial C}{\partial b_j^2} = \begin{bmatrix} \delta_1^2 \\ \delta_2^2 \\ \delta_3^2 \end{bmatrix} = \delta^2$$