ANN Design guideline

Tommy Petersen
AIA
Copenhagen
Denmark
tp@aia.dk

Abstract

This article presents some notes on a design guideline for an artificial neural network (ANN). The ANN is a feedforward network implementing stochastic gradient descent by using the backpropagation algorithm.

1 Introduction

In [1], an introduction to ANN implementing stochastic gradient descent using the backpropagation algorithm is given. This article presents a design guideline to be used for implementing the ANN itself. The guideline is independent of programming language.

2 Overview

Figure 1 presents an overview of learning via gradient descent.

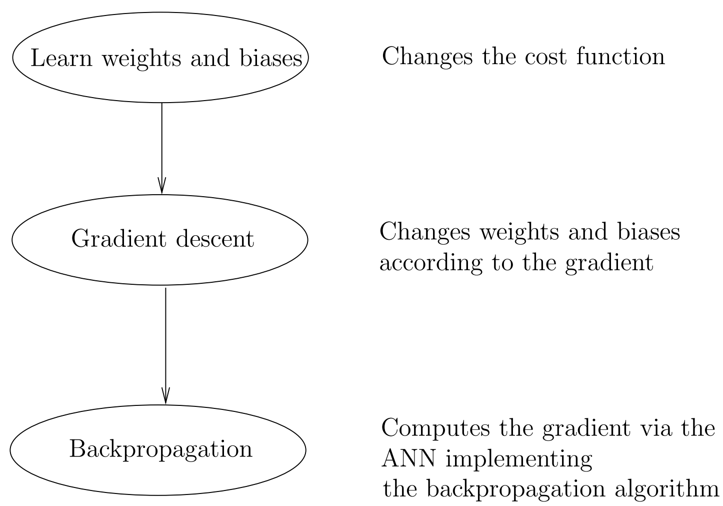

Figure 1: Overview of learning via gradient descent.

<u>SGD (Stochastic Gradient Descent)</u>

η: Learning rate, small positive.

Uses mini-batches to compute approximated gradient.

Uses epochs in which the training inputs are exhausted through sampling of mini-batches.

3 Backpropagation overview

Figure 2 presents an overview of the backpropagation algorithm.

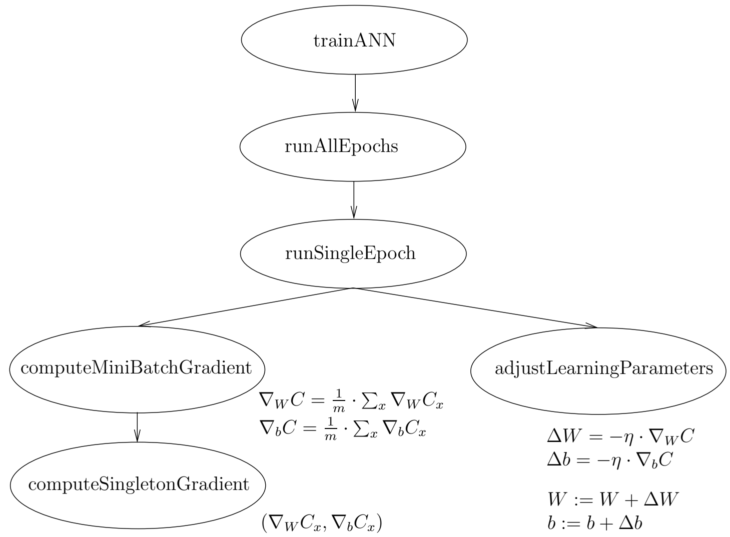

Figure 2: Overview of the backpropagation algorithm.

In `computeSingletonGradient`, the gradient $(\nabla_W C_x, \nabla_b C_x)$ is computed using the following four equations:

$$\delta^L = \nabla_{a^L} C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = (W^{l+1})^T \cdot \delta^{l+1} \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = \delta^l_j \cdot a^{l-1}_k \tag{BP4}$$

, where $\delta^l_j = \frac{\partial C}{\partial z^l_j}$, $z^l = W^l \cdot a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

4 trainANN

Trains the network by calling `runAllEpochs(miniBatchSize,` η`, numberOfEpochs)`.

<u>Datastructures</u>

Datastructure	**Description**
trainingData	The training data
miniBatchSize	The size of each mini-batch except a possible mini-batch remainder (input)
η	Learning rate (input)
numberOfEpochs	The number of epochs to run (input)

5 runAllEpochs

Runs all epochs as follows:

1. For each epoch
   (a) `runSingleEpoch(miniBatchSize,` η`)`

Datastructures

Datastructure	Description
miniBatchSize	The size of each mini-batch except a possible mini-batch remainder (input)
η	Learning rate (input)
numberOfEpochs	The number of epochs to run (input)

6 runSingleEpoch

Runs a single epoch as follows:

1. Permute the set of training examples.
2. Partition the training examples into mini-batches.
3. For each mini-batch
   (a) `computeMiniBatchGradient(mini-batch)`
   (b) `adjustLearningParameters(`η`, WGradientList, bGradientList)`

Datastructures

Datastructure	Description
trainingData	List of all training examples
miniBatchSize	The size of each mini-batch except a possible mini-batch remainder
η	Learning rate
WGradientList	List of gradient weight matrices
bGradientList	List of gradient bias vectors
miniBatch	List of training examles

7 computeMiniBatchGradient

Computes $(\nabla_W C, \nabla_b C)$ given mini-batch consisting of training examples (x, y).

$$[(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)] \qquad \text{(mini-batch)}$$

Gradients are computed by averaging sums of singleton gradients:

$$\nabla_W C = \frac{1}{m} \cdot \sum_x \nabla_W C_x$$
$$\nabla_b C = \frac{1}{m} \cdot \sum_x \nabla_b C_x$$

$\nabla_W C_x$ and $\nabla_b C_x$ are computed by calling `computeSingletonGradient(`$\mathtt{x_i}$`,`$\mathtt{y_i}$`)`

Datastructures

Datastructure	Description
miniBatch	List of training examples (x, y) (input)
m	Number of elements in miniBatch
WSingletonGradientAggrList	List of aggregated singleton gradient weight matrices (for local use)
bSingletonGradientAggrList	List of aggregated singleton gradient bias vectors (for local use)
WGradientList	List of gradient weight matrices (output)
bGradientList	List of gradient bias vectors (output)

8 adjustLearningParameters

Computes (Δ_W, Δ_b) and updates (W, b) by

$$W := W + \Delta_W$$
$$b := b + \Delta_b$$

, where $\Delta_W := -\eta \cdot \nabla_W C$ and $\Delta_b := -\eta \cdot \nabla_b C$.
Datastructures

Datastructure	Description
WList	List of weight matrices
bList	List of bias vectors
η	Learning rate
WGradientList	List of gradient weight matrices (input)
bGradientList	List of gradient bias vectors (input)

9 computeSingletonGradient

Computes $(\nabla_W C_x, \nabla_b C_x)$ given training example (x, y). In figure 3 an ANN is showed.

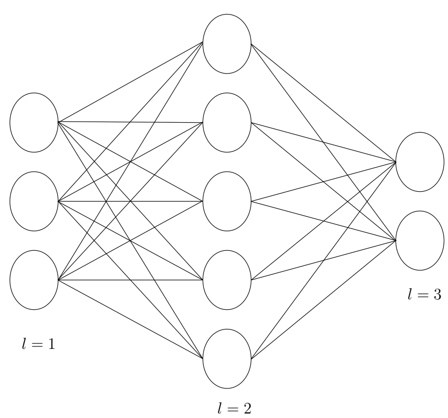

Figure 3: ANN.

For input to and output from the network, we use the following:

- a^1 is input to the network.
- a^L is output from the network.

Feedforward
Computes z^l and a^l for $l = 2, 3, ..., L$.

$$z^l := W^l \cdot a^{l-1} + b^l$$
$$a^l := \sigma(z^l) \tag{1}$$

The equation (1) can be used to compute $\sigma'(z^l)$ when back propagating.

Backpropagate
Computes $(\nabla_W C_x, \nabla_b C_x)$ by sending the error δ^l backwards. The computations are done using the following four equations:

$$\delta^L = \nabla_{a^L} C \odot \sigma'(z^L) \tag{BP1}$$
$$\delta^l = (W^{l+1})^T \cdot \delta^{l+1} \odot \sigma'(z^l) \tag{BP2}$$
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{BP3}$$
$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l \cdot a_k^{l-1} \tag{BP4}$$

, where $\delta_j^l = \frac{\partial C}{\partial z_j^l}$, $z^l = W^l \cdot a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

For BP2, $l = L-1, L-2, ..., 2$, and for BP3 and BP4, $l = L, L-1, ..., 2$.

For BP1, the whole matrix $\frac{\partial C}{\partial W^l}$ can possibly be computed by multiplication of vector and transposed vector, $\delta^l \cdot (a^{l-1})^T$.
So, given the quantities δ^l and a^{l-1}, we must compute

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

for all $j = 1, 2, ..., J$ and $k = 1, 2, ..., K$.
We have the following matrix dimensions:

$$\frac{\partial C}{\partial W^l} \in \mathbb{M}(J, K)$$
$$\delta^l \in \mathbb{M}(J, 1)$$
$$a^{l-1} \in \mathbb{M}(K, 1)$$

The claim is, that

$$\frac{\partial C}{\partial W^l} = \delta^l \cdot (a^{l-1})^T$$

<u>Proof</u>

$$\begin{aligned} \left(\frac{\partial C}{\partial W^l}\right)_{jk} &= \frac{\partial C}{\partial w_{jk}^l} \\ &= \delta_j^l \cdot a_k^{l-1} \\ &= \delta_j^l \cdot (a^{l-1})_k^T \\ &= (\delta^l \cdot (a^{l-1})^T)_{jk} \end{aligned}$$

For all $j = 1, 2, ..., J$ and $k = 1, 2, ..., K$.

This shows, that BP4 can be computed as $\delta^l \cdot (a^{l-1})^T$. $\square$

$\sigma(z^l)$**, activation**(z^l)

The sigmoid function $\sigma(z)$:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

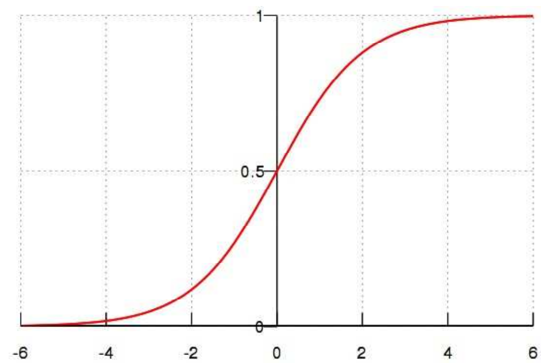

Figure 4: Sigmoid $\frac{1}{1+e^{-z}}$ (from wikipedia).

$\sigma'(z^l)$**, derivativeActivation**(z^l)

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

$\nabla_a^L C_x$**, gradientCostOutput**(a^L)

$$\begin{aligned} C_x(a^L) &= \frac{1}{2} \cdot \|a^L - y\|^2 && \text{(Quadratic cost)} \\ &= \frac{1}{2} \cdot \sum_{j=1}^{L} (a_j^L - y_j^L)^2 \end{aligned}$$

Computation of $\nabla_{a^L} C_x$:

$$\begin{aligned} \frac{\partial}{\partial a_i^L} C_x(a^L) &= \frac{\partial}{\partial a_i^L} \frac{1}{2} \cdot \sum_{j=1}^{L} (a_j^L - y_j^L)^2 \\ &= \frac{\partial}{\partial a_i^L} \frac{1}{2} \cdot (a_i^L - y_i^L)^2 \\ &= (a_i^L - y_i^L) \cdot \frac{\partial}{\partial a_i^L} (a_i^L - y_i^L) \\ &= a_i^L - y_i^L \end{aligned}$$

So

$$\nabla_{a^L} C_x = \begin{pmatrix} a_1^L - y_1^L \\ \vdots \\ a_J^L - y_J^L \end{pmatrix}$$

Datastructures

Datastructure	**Description**	**Size**	**Remark**
WList	List of weight matrices	L	WList[0] is not used
bList	List of bias vectors	L	bList[0] is not used
WSingletonGradientList	List of singleton gradient weight matrices	L	WSingletonGradientList[0] is not used
bSingletonGradientList	List of singleton gradient bias vectors	L	bSingletonGradientList[0] is not used
x	Input vector for the network	$K \times 1$	
deltaList	List of delta vectors	L	deltaList[0] is not used
zList	List of z-vectors	L	zList[0] is not used
aList	List of a-vectors	L	aList[0] is not used
y	Correct output vector from the network	$J \times 1$	

References

[1] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015, http://neuralnetworksanddeeplearning.com/.