Extra Work of *Introduction to Programming*: A very first inmersion in Neural Networks

Author:

Alfonso Mateos Vicente

Tutor:

Prof. Pascal Monasse

École des Ponts
ParisTech

Mathematics and Computer Engineering
École des Ponts ParisTech
France
December 15, 2023

Contents

1 Introduction

The exploration of neural networks is central to advancements in machine learning and deep learning. This report delves into the mathematical foundations and structural intricacies of neural networks, particularly emphasizing their application in image recognition and natural language processing tasks.

Neural networks are, at their essence, a series of mathematical transformations that draw inspiration from the processing mechanisms of biological neurons. The fundamental operation in a neural network involves each neuron computing a weighted sum of its inputs, followed by the application of a non-linear activation function. This can be mathematically represented as:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{1}$$

Here, x_i are the input values, w_i represent the weights assigned to these inputs, b is the bias, and f denotes the activation function. This equation encapsulates the core operation that enables neural networks to perform complex data transformations and learning.

The report aims to dissect and elucidate the implementation and mathematical rigor behind the neural network's architecture. We focus on the linear algebra and calculus underpinning the network layers, activation functions, and learning algorithms. Special attention is given to the practical translation of these mathematical principles into efficient computational models, specifically in the realm of C++ programming.

By examining the theoretical aspects alongside practical implementation, this report provides a comprehensive view of neural network functionality, from the initial input layer to the final output. The ultimate goal is to offer a clear understanding of how neural networks harness mathematical principles to learn from data and make predictions, thus serving as powerful tools in the field of artificial intelligence.

2 Implementation Insights

This section meticulously examines the project's implementation, emphasizing the mathematical underpinnings and their efficient transposition into C++ code.

2.1 Layer Implementations

In the development of our neural network model, a pivotal realization was that incorporating a variety of layer types significantly bolsters the network's efficacy and robustness. While the initial design was centered around the fully connected (Dense) layer, the introduction of dropout and batch normalization layers marked a significant enhancement in the network's architecture.

Each layer in a neural network can be conceptualized as a distinct mathematical entity characterized by a set of variables and functions. Commonly, these elements include:

- **Variables**:
  - **Inputs**: The data points or signals entering the layer.

- **Outputs**: The transformed data exiting the layer.
- **Weights**: Parameters adjusting the strength of connections between neurons.
- **Biases**: Parameters that shift the activation function curve, aiding in fitting.
- **Deltas**: Gradients of the loss function with respect to the layer's outputs.
- **Activation Function**: A non-linear function applied to the layer's output.

- **Functions**:
  - **Forward Pass**: Computes the output based on inputs, weights, and biases.
  - **Compute Deltas**: Calculates the error gradients for backpropagation.

In the following sections, we delve into the specific implementation and mathematical rationale behind each layer type.

2.1.1 Dense Layer

The Dense Layer, forming the backbone of the neural network, densely interconnects neurons from the preceding layer to those in the subsequent layer. Its primary function is to transform the input data linearly, followed by a non-linear activation. The output of a neuron in the Dense Layer is computed as:

$$y_j = f\left(\sum_{i=1}^{n} w_{ji}x_i + b_j\right) \tag{2}$$

Here, w_{ji} represents the weight connecting the i-th input neuron to the j-th output neuron, x_i is the input, b_j is the bias, and f is the activation function. This layer is crucial for learning high-level features from the input data.

2.1.2 Dropout Layer

The Dropout Layer is an ingenious solution to mitigate the issue of overfitting, a common challenge in neural network training. It randomly inactivates a subset of neurons during the training phase, which prevents the network from becoming overly reliant on any specific neuron. Mathematically, the output of the Dropout Layer is given by:

$$y_i = x_i \cdot D_i \tag{3}$$

where x_i is the input, y_i is the output, and D_i is a binary random variable that follows a Bernoulli distribution, determining whether the neuron is active or dropped.

2.1.3 Batch Normalization Layer

The Batch Normalization Layer addresses the issue of internal covariate shift by normalizing the layer inputs for each mini-batch. This process makes the training more stable and accelerates the convergence of the training process. The normalization is performed as follows:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{4}$$

Here, x_i is the input, μ_B and σ_B^2 are the mean and variance of the inputs calculated over the mini-batch, and ϵ is a small constant added for numerical stability. This layer significantly enhances the network's performance, especially in deeper architectures.

2.2 Neural Network Assembly

The architecture of our neural network is composed of a sequence of layers, each contributing uniquely to the network's functionality. The network is structured as a vector of these layers, with critical elements including the management of layer parameters and the gradients associated with them. Key functions implemented in this architecture include:

1. **addLayer**: Integrates a new layer into the network, expanding its depth and capabilities.
2. **trains**: Facilitates the training process of the network using provided data.
3. **predict**: Generates output predictions based on the learned parameters.
4. **updateWeightsAndBiases**: Modifies the network's weights and biases, based on the gradients computed during training.
5. **initializeGradientAccumulators**: Sets up structures for accumulating gradient information over batches.
6. **accumulateGradients**: Gathers gradient information during backpropagation.
7. **applyAccumulatedGradients**: Applies the accumulated gradients to update the network's parameters.

2.2.1 Training Methodology

The training of our network is governed by the principles of stochastic gradient descent (SGD), an optimization algorithm that updates the network's parameters iteratively. This process is mathematically represented as:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t) \tag{5}$$

where θ denotes the network parameters (weights and biases), α is the learning rate, and $\nabla L(\theta_t)$ is the gradient of the loss function with respect to the parameters. The training process involves initialization of parameters, iterating over epochs, processing mini-batches, and conducting forward and backward propagation to update the network parameters.

2.2.2 Activation Functions

Activation functions are crucial in introducing non-linearity to the network, allowing it to learn and model complex relationships. We utilize several activation functions, each with its unique characteristics:

- **Sigmoid Function**: The Sigmoid function is particularly effective in squashing the output between 0 and 1, making it ideal for binary classification tasks. It is mathematically described as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

- **ReLU Function**: The Rectified Linear Unit (ReLU) is widely used due to its computational efficiency and effectiveness in mitigating the vanishing gradient problem. It is defined as:

$$ReLU(x) = max(0, x) \tag{7}$$

ReLU promotes faster convergence in SGD by maintaining a linear behavior for positive inputs while nullifying negative inputs, thus simplifying the optimization landscape.

- **TanH Function**: The Hyperbolic Tangent (TanH) function is similar to the sigmoid function but maps the input values to a range between -1 and 1. It is particularly useful when the model needs to normalize the output, and is mathematically expressed as:

$$TanH(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{8}$$

TanH, like Sigmoid, is also S-shaped but provides the advantage of having outputs centered around zero, which often leads to improved convergence during training. This zero-centered nature makes it more suitable for layers that are not output layers, especially in deep networks.

3 Experimental Results

With the neural network already implemented, we wanted to prove if it was really working. For this aims we started with the usual logical problems, this is using the logical doors like AND, OR etc. to know if it was working. When we achieved good results with each door, we thought that maybe we could take a bigger dataset and give it to the neural network to see what happens. With this idea in mind we landed to the *Adult dataset* provided by *https://archive.ics.uci.edu/dataset/2/adult*. The idea of this dataset is to predict whether income exceeds 50.000$ per year based on census data. Also known as “Census Income” dataset.

3.1 Census Income Dataset

The *Adult* dataset, also known as the *Census Income* dataset, is a widely-used benchmark in machine learning for classification tasks. It is designed to predict whether a person’s income exceeds $50,000 per year based on census data.

3.1.1 Dataset Characteristics

- **Number of Instances:** 48,842
- **Number of Features:** 14
- **Feature Types:** Categorical, Integer
- **Associated Tasks:** Classification
- **Missing Values:** Yes

3.1.2 Feature Description

The dataset consists of the following features:

1. **age:** Continuous (integer).
2. **workclass:** Categorical (e.g., Private, State-gov, Federal-gov).
3. **fnlwgt:** Continuous (integer).
4. **education:** Categorical (e.g., Bachelors, HS-grad, 11th).
5. **education-num:** Continuous (integer).
6. **marital-status:** Categorical (e.g., Married-civ-spouse, Never-married).
7. **occupation:** Categorical (e.g., Tech-support, Craft-repair).
8. **relationship:** Categorical (e.g., Wife, Own-child, Husband).
9. **race:** Categorical (e.g., White, Black, Asian-Pac-Islander).
10. **sex:** Binary (Female, Male).
11. **capital-gain:** Continuous (integer).
12. **capital-loss:** Continuous (integer).
13. **hours-per-week:** Continuous (integer).
14. **native-country:** Categorical (e.g., United-States, India).

The target variable is **income**, which is binary (>$50K, <=$50K).

3.1.3 Additional Information

The dataset was extracted by Barry Becker from the 1994 Census database. The extraction criteria were: ((AAGE>16) && (AGI>100) && (AFNLWGT>1) && (HRSWK>0)). The prediction task is to determine whether a person makes over $50K a year.

3.2 Preprocessing the Dataset

Before introducing the dataset into our neural network, a crucial preprocessing step is required. Since the neural network cannot directly interpret strings and categorical data, these need to be transformed into a numerical format, specifically normalized to a range between 0 and 1.

3.2.1 Binary Items

Binary items represent the simplest case for preprocessing. We map the dataset's binary attributes to 0 for one category (e.g., 'False', 'No', 'Male') and to 1 for the other (e.g., 'True', 'Yes', 'Female').

3.2.2 Continuous Items

For continuous features, normalization is performed by dividing each value by the maximum value in that feature column. This approach scales the feature to a [0, 1] range, making it suitable for neural network processing.

3.2.3 Categorical Items

Categorical features require a more nuanced approach. We employ a technique of indexing followed by normalization. To illustrate, consider a feature with three categories: yellow, green, and red apples. We first assign each category an index (e.g., yellow = 0, green = 1, red = 2). Next, we normalize these indices by dividing by the total number of categories. In this example, a yellow apple would be represented as $0/3 = 0$, a green as $1/3 \approx 0.3333$, and a red as $2/3 \approx 0.6666$. This process effectively encodes categorical data into a numerical format that a neural network can interpret.

3.2.4 Training and Test Set

The final preprocessing step involves dividing the dataset into training and testing subsets. We allocate 80% of the data for training the model and reserve the remaining 20% for testing. This split helps in evaluating the model's performance and in checking for overfitting, despite the use of techniques like dropout layers in our neural network. Therefore the training set has 26047 rows and the test dataset has 6512 after processing the dataset.

3.3 Neural Network Architecture

For our experiment, we designed a neural network with the following specifications:

- **Training Mode:** Stochastic
- **Layer 1:** Dropout Layer with a dropout probability of 0.01 to prevent overfitting.
- **Layer 2:** Batch Normalization Layer to normalize inputs for 13 features, enhancing training stability.
- **Layer 3:** Fully Connected Layer transitioning from 13 to 8 neurons, utilizing TanH activation for non-linear transformation.
- **Layer 4:** Fully Connected Layer narrowing down from 8 to 1 neuron, with Sigmoid activation for binary classification.
- **Number of Epochs:** 100.
- **Learning Rate:** 0.05.

3.4 Results and Analysis

The neural network was trained on the dataset, and the progression of loss over epochs was meticulously recorded:

Epoch	Loss
1	0.147021
2	0.132685
3	0.128960
4	0.127195
5	0.123872
...	...
100	0.107419

A notable consistent decrease in loss was observed, indicating effective learning. The loss reduced from 0.147021 in the first epoch to 0.107419 by the final, 100th epoch.

Crucially, the loss on the test dataset was calculated to be 0.108675, closely aligning with the training loss. This proximity in values suggests that the model is not overfitting the training data and is generalizing well to unseen data. Such a result is indicative of a well-tuned model, demonstrating the efficacy of the chosen architecture and training parameters.