Convolutional Neural Network from Scratch

Le Chi Thanh - M23.ICT.011

June 16, 2024

Abstract

This report presents a comprehensive guide on building a Convolutional Neural Network (CNN) from scratch. It covers the fundamental aspects of network design, including the architecture, base layer, dense layers, activation layers, convolutinal layers and the implementation of various activation and loss functions. Additionally, it provides a step-by-step guide with detailed mathematical explanations and code implementations, serving as a personal resource for understanding and constructing CNNs from scratch.

1 Neural Network Design

This section provides a foundation for understanding and constructing the basic elements of a neural network, setting the stage for the addition of convolutional layers and more advanced functionalities in subsequent sections.

1.1 Architecture

1.2 Layer implementation

Given layer L1(X1,Y1) and layer L2(X2,Y2) stand next together in a network. We easily conclude output Y1 of layer L1 is input X2 of layer L2 (in Forward):

$$Y1 = X2$$

In Back propagation, loss derivative with respect to X2 is equal to loss derivative with respect to Y1:

$$\frac{\partial E}{\partial X2} = \frac{\partial E}{\partial Y1}$$

1.3 File description

- `models/network.py`: Defines the neural network process including forward and backpropagation.
- `layers/layer.py`: Defines the base `Layer` class which is the interface for all layers.
- `layers/dense.py`: Implements the `Dense` layer.
- `layers/convolutional.py`: Implements the `Convolutional` layer.
- `layers/pooling.py`: Implements the `MaxPooling` layer.
- `layers/reshape.py`: Implements reshaping layers.
- `activations/activation.py`: Base class for activation functions.
- `activations/activations.py`: Specific activation functions like `Sigmoid`.
- `helpers/algebra_helper.py`: Helper functions for algebraic operations.
- `helpers/convolutional_helper.py`: Helper functions for convolutional operations.
- `utils/losses.py`: Defines loss functions and their derivatives (including Binary Cross-Entropy, MSE, Categorial Cross-Entropy).

2 Experiment

2.1 MNIST Dataset

Binary classification for MNIST dataset using Sigmoid activation, we extract only image of handwritten 0 and 1.

```
network = [
    Convolutional(input_shape=(1, 28, 28), kernel_size=3, depth=5, mode='valid'),
    Sigmoid(log=False),
    Reshape(input_shape=(5, 26, 26), output_shape=(3380, 1)),
    Dense(5 * 26 * 26, 100),
    Sigmoid(log=False),
    Dense(100, 2),
    Sigmoid(log=False)
]
```

Binary classification for MNIST dataset using Softmax activation. Although Softmax for binary classification using Binary Cross Entropy is not a good comparsion, I still apply it here to have a glance in the state-of-art and to easily compare to Sigmoid binary classification.

```
network = [
    Convolutional(input_shape=(1, 28, 28), kernel_size=3, depth=5, mode='valid'),
    Sigmoid(log=False),
    Reshape(input_shape=(5, 26, 26), output_shape=(3380, 1)),
    Dense(5 * 26 * 26, 100),
    Sigmoid(log=False),
    Dense(100, 2),
    Softmax(log=False)
]
```

Categorial classification for MNIST dataset using Solfmax activation with Categorial Cross Entropy loss function. The traning model run prediction on 3 label handwritten 0,1,2 of MNIST.

```
    network = [
    Convolutional(input_shape=(1, 28, 28), kernel_size=3, depth=5, mode='valid'),
    Sigmoid(log=False),
    Reshape(input_shape=(5, 26, 26), output_shape=(3380, 1)),
    Dense(3380, 100),
    Sigmoid(log=False),
    Dense(100, 3),  # 3 classes: 0, 1, 2
    Softmax(log=False)
]
```

3 Comparison

The loss evaluation for both the softmax and sigmoid binary classification models shows the differences in their training progress:

Softmax Binary Classification:

- The error (loss) decreases rapidly and consistently over the epochs.
- By the final epoch, the error is extremely low, indicating that the model is making very few mistakes in its predictions.

Sigmoid Binary Classification:

- The error decreases much slower compared to the softmax model.
- The final error remains significantly higher, indicating more prediction errors compared to the softmax model.

Conclusion:

- The softmax binary classification model achieves an exceptionally low error, suggesting high accuracy in prediction.

In summary, the softmax binary classification model outperforms the sigmoid model significantly in terms of accuracy and loss reduction. The result of prediction is given in the attached notebook file that comes with this report.

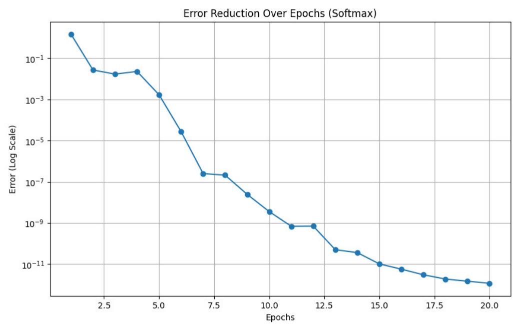

Figure 1: Softmax Binary Classification

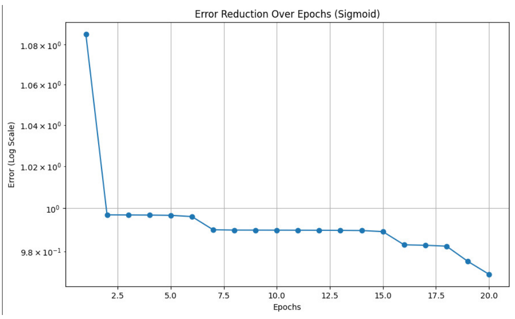

Figure 2: Sigmoid Binary Classification

Figure 3: Comparison of Loss Evaluation

3.1 Challenges

- **Mathematical implementation of each layer**: Providing the exact mathematical implementation of each layer is essential. It is necessary to derive the loss derivative with respect to all parameters (input, weight, bias) using only the output.
- **Handling data types**: Layers can transform a tensor to a normal matrix during the forward pass, and vice-versa during the backward pass. Writing condition checks and implementing methods to handle these data type conversions is required.
- **Numerical stability**: Checking for excessively large or small values, especially in functions like Sigmoid, Softmax, and the loss function, is crucial to maintain numerical stability.

3.2 Future work

- **Exploring new layer types**: Understanding the mathematical underpinnings opens up opportunities to create new types of layers. For example, exploring average mean pooling or applying new ideas to convolutional layers could be fruitful.
- **Innovative applications**: Leveraging mathematical concepts can lead to innovative applications. For instance, applying novel ideas to enhance image convolutions or introducing new types of pooling layers could improve network performance.