Building Artificial Neural Networks Using Keras

Omid Saberi
Dept. of Industrial Automation
University West
Trollhättan, Sweden
https://orcid.org/0009-0005-3596-9182

***Abstract*—This report demonstrates the solutions for three tasks that involve tensor manipulations, constructing basic neural network building blocks, and training neural networks for classification. Libraries such as NumPy and Keras are utilized to achieve the objectives. Tensor operations, model creation, and evaluation steps are discussed in detail.**

***Index Terms*—Tensor Operations, Neural Networks, NumPy, Keras, Classification, MNIST, IMDb**

I. Introduction

Artificial Neural Networks (ANNs) are computational models inspired by the human brain, consisting of layers of interconnected nodes or "neurons" designed to process and analyze data. They are widely used in machine learning applications, including image recognition, natural language processing, and predictive modeling, due to their ability to capture complex patterns in data.

Keras, a high-level deep learning library, simplifies the construction and training of ANNs by providing an intuitive interface to TensorFlow. It enables rapid prototyping and deployment of neural networks for various tasks, including classification, regression, and clustering.

Classification tasks, a core application of ANNs, involve assigning input data to predefined categories. This report focuses on implementing and evaluating ANNs for classification tasks using the MNIST and IMDb datasets. The work is divided into three tasks: tensor manipulations using NumPy, constructing a basic neural network with Keras, and building and training classification models. Each task demonstrates foundational concepts critical to working with ANN.

This report covers the essential steps for manipulating tensors, constructing neural network components, and solving classification problems. The report is divided into three tasks: Tensor Manipulations (using NumPy), Neural Network Foundations (using Keras), and Classification Tasks on MNIST and IMDb datasets.

II. Tensor Manipulations Using NumPy

A. Creating Two Sample Tensors

Two random integer tensors are created using `numpy.random.randint` to manipulate the tensor in subsequent steps. The tensors are of a random rank yet compatible with the operations. Moreover, another tensor was created to be used with the unary operations.

B. Element-wise Addition of Tensors

The element-wise addition is performed using the + operator between the two tensors, resulting in a new tensor with summed corresponding elements.

C. Element-wise Multiplication of Tensors

The element-wise multiplication is achieved using the `*` operator. The resulting tensor contains products of the corresponding elements of the two tensors.

D. Reshaping a Tensor

The `numpy.reshape()` function is used to change the shape of one of the tensors to a 3x4 matrix while retaining the original data. Also notable is that the total number of elements must be the same in the desired shape as in the original matrix.

E. Slicing and Indexing Tensors

Specific rows and columns are accessed using NumPy slicing. For example, the first two rows are retrieved with `tensor[0:2,:]`.

F. Combining Tensors

Tensors are combined along an axis using `numpy.concatenate()` to form a larger tensor.

G. Splitting Tensors

The `numpy.split()` function divides a tensor into smaller tensors along a specified axis.

H. Basic Mathematical Operations on Tensors

NumPy provides functions such as `numpy.add()` and `numpy.multiply()` to perform basic mathematical operations on tensors. These operations can be applied both element-wise and with scalars. For example, `numpy.add()` can add a scalar to all tensor elements or can be used to compute element-wise addition between two tensors of the same shape. Similarly, `numpy.multiply()` enables element-wise multiplication, which is essential for various tensor manipulations in machine learning workflows.

III. Constructing a Basic Neural Network Using Keras

A. Defining a Simple Neural Network Model

A sequential model is defined using `keras.models.Sequential()`, comprising one input layer, one hidden layer, and one output layer [1].

B. Compiling the Model

The model is compiled with the `adam` optimizer, `categorical_crossentropy` loss, and `accuracy` as a metric using `model.compile()` [1].

C. Printing Model Summary

The model structure, including the number of parameters, is displayed using `model.summary()` [1].

IV. Training and Evaluating Neural Networks on MNIST and IMDb Datasets

A. Loading and Preprocessing the Dataset

The MNIST and IMDb datasets are loaded using Keras' `keras.datasets`. Data are normalized to the range [0, 1] for MNIST and tokenized for IMDb.

B. Defining a Neural Network for Classification

A feedforward network is designed for MNIST with input, hidden, and output layers. For IMDb, an embedding layer is added to process textual data [2].

C. Compiling the Model

Both models are compiled using `adam` optimizer and respective loss functions: `sparse_categorical_crossentropy` for MNIST and `binary_crossentropy` for IMDb.

D. Training the Model

The models are trained using `model.fit()` with specified epochs and batch sizes. Validation splits are used to monitor performance.

E. Evaluating the Model and Plotting Performance

Evaluation is performed using `model.evaluate()` on the test set. Accuracy and loss values are plotted over epochs using `matplotlib`.

F. Plotting Training and Validation Loss Over Epochs

The loss values during training and validation are extracted and plotted using `matplotlib` [3].

G. Plotting Training and Validation Accuracy Over Epochs

Accuracy values are plotted similarly to visualize improvement over epochs.

H. Displaying Test Loss and Accuracy

Final test loss and accuracy are displayed to summarize model performance.

V. Conclusion

This lab report demonstrates essential skills in tensor manipulations, constructing basic neural networks, and training models for classification tasks. The results show the successful implementation of operations and models using NumPy and Keras.

References

[1] F. Chollet, "Mathematical building blocks of neural networks," GitHub repository, GitHub, Nov. 18, 2024. [Online]. Available: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/chapter02_mathematical-building-blocks.ipynb

[2] Keras, "Bidirectional LSTM on IMDB," Keras Documentation, Nov. 18, 2024. [Online]. Available: https://keras.io/examples/nlp/bidirectional_lstm_imdb/

[3] O. Saberi, "IAI600Lab7.ipynb," GitHub repository, GitHub, Nov. 6, 2024. [Online]. Available: https://github.com/omisa69/IAI600Lab7

[4] O. Saberi, "DAU500Lab1.ipynb," GitHub repository, GitHub, Nov. 18, 2024. [Online]. Available: https://github.com/omisa69/DAU500Lab1