### **Document Version and Revision History**

#### **Document Version: 1.0**

### **Date: 06.05.2024**

### **Note to Readers**

This document is a living document and may be subject to updates and revisions over time. Also not all elements of a project are presented in this document. Some parts of code responsible mainly for testing and graphical representation are not included in this document but will be attached to the main project report later on.

### **Net Class**

The Net class represents a neural network. It encapsulates the functionality for training the network, performing forward propagation to compute outputs, and backpropagation to adjust connection weights based on the error.

#### **Member Variables:**

* layers: A vector of [Layer](#_gn668rrpiygv) objects representing the layers of the neural network.
* error: The overall error of the network, computed during backpropagation (RMS of output neuron errors)
* recentAverageError: The recent average error of the network, used for monitoring the training progress.
* recentAverageSmoothingFactor: The smoothing factor for computing the recent average error.

#### **Member Functions:**

* getResults(std::vector<double> &resultVals) const: Retrieves the output values of the network's output layer.
* backProp(const std::vector<double> &targetVals): Performs backpropagation to update connection weights based on the target output values.
* feedForward(const std::vector<double> &inputVals): Performs forward propagation to compute output values based on input values.
* Net(const std::vector<unsigned> &topology): Constructor for the Net class, initializes the network with the specified topology.

#### **Constructor:**

* The Net class constructor initializes the network with the specified topology, creating layers and neurons according to the topology provided. One extra hidden neuron is also created in each layer.

### **Neuron Class**

The Neuron class represents a single neuron in a neural network. Each neuron receives input from other neurons, processes this input using a transfer function, and produces an output value.

#### **Member Variables:**

* outputVal: The output value produced by the neuron after processing its inputs.
* m\_gradient: The gradient of the neuron, which is used during backpropagation to adjust the weights of connections.
* outputWeights: A vector of [Connection](#_a16w635e614p) objects representing the connections between this neuron and neurons in the next layer. Each [Connection](#_a16w635e614p) object stores the weight of the connection and the change in weight (delta weight) during training.
* m\_myIndex: The index of this neuron within its layer.

#### **Member Functions:**

* updateInputWeights(Layer &prevLayer): Updates the weights of connections between this neuron and neurons in the previous layer based on the error gradient.
* sumDOW(const Layer &nextLayer) const: Calculates the sum of the products of output weights and gradients of neurons in the next layer, which is used during backpropagation.
* calcHiddenGradients(const Layer &nextLayer): Calculates the gradient of this neuron for hidden layers during backpropagation.
* calcOutputGradients(double targetVal): Calculates the gradient of this neuron for the output layer during backpropagation based on the target output value.
* transferFunction(double x): Computes the output of the neuron using a transfer function, often the sigmoid function.
* transferFunctionDerivative(double x): Computes the derivative of the transfer function, which is used during backpropagation.
* feedForward(const Layer &prevLayer): Computes the output value of the neuron by summing the weighted inputs from the previous layer and applying the transfer function.

#### **Constructor:**

* Neuron(unsigned numOutputs, unsigned myIndex): Initializes the neuron with a specified number of output connections and its index within the layer. It creates Connection objects with random weights for each output connection.

### **Connection Structure**

The Connection structure represents a connection between two neurons in adjacent layers. It stores the weight of the connection and the change in weight (delta weight) during training.

#### **Member Variables:**

* weight: The weight of the connection, which determines the strength of the connection between neurons.
* deltaWeight: The change in weight of the connection during training, which is used to update the weight during backpropagation.

#### **Member Functions:**

* Connection(): Constructor for the Connection structure, initializes the weight with a random value.

### **Layer Class**

The Layer class represents a layer of neurons in a neural network. Layers organize neurons into groups, with each neuron connected to every neuron in the previous and next layers.

#### **Member Variables:**

* m\_neurons: A vector that stores the neurons belonging to the layer.

#### **Member Functions:**

* push\_back(const Neuron& element): Adds a neuron to the end of the layer.
* operator[](int i): Provides access to the neuron at the specified index i in the layer. Throws an out\_of\_range exception if the index is invalid.
* back(): Returns a reference to the last neuron in the layer. Throws an out\_of\_range exception if the layer is empty.
* size(): Returns the number of neurons in the layer.

#### **Neuron Access:**

* The push\_back method allows users to add neurons to the layer.
* The operator[] method provides random access to individual neurons in the layer by index.
* The back method allows users to access the last neuron in the layer.

#### **Exception Handling:**

* The operator[] and back methods perform bounds checking and throw an out\_of\_range exception if the index is out of bounds or if the layer is empty.

Link to github: <https://github.com/Kokoszsz/neural-network>