**Отчет лабы 4**

932003

Ву Нгок Зуй Хань

До За Бать

**1. Цель работы.**

- Искусственные нейронные сети (ИНС), обычно называемые просто нейронными сетями (НС), представляют собой вычислительные системы, вдохновленные биологическими нейронными сетями, составляющими мозг животных.

- Возможности ANN относятся к следующим широким категориям:

* Аппроксимация функций или регрессионный анализ, включая прогнозирование временных рядов, приближение пригодности и моделирование.
* Классификация, включая распознавание образов и последовательностей, обнаружение новизны и последовательное принятие решений.
* Обработка данных, включая фильтрацию, кластеризацию, слепое разделение источников и сжатие.
* Робототехника, включая направляющие манипуляторы и протезы.

**2. Постановку задачи.**

- Благодаря своей способности воспроизводить и моделировать нелинейные процессы искусственные нейронные сети нашли применение во многих дисциплинах. Области применения включают системную идентификацию и управление (управление транспортными средствами, прогнозирование траектории, управление процессами, управление природными ресурсами), квантовую химию, общие игры, распознавание образов (радарные системы, идентификацию лиц, классификацию сигналов, 3D-реконструкцию, распознавание объектов и многое другое). анализ данных датчиков, распознавание последовательности (распознавание жестов, речи, рукописного и печатного текста), медицинская диагностика, финансы (например, автоматизированные торговые системы), интеллектуальный анализ данных, визуализация, машинный перевод, фильтрация социальных сетей и фильтрация спама в электронной почте. ИНС использовались для диагностики нескольких типов рака и для различения высокоинвазивных линий раковых клеток от менее инвазивных линий, используя только информацию о форме клеток.

- ИНС использовались для ускорения анализа надежности инфраструктуры, подверженной стихийным бедствиям, и для прогнозирования осадки фундамента. ИНС также использовались для построения моделей черного ящика в науках о Земле: гидрологии, моделировании океана, прибрежной инженерии и геоморфологии. ИНС использовались в кибербезопасности с целью отличить законные действия от злонамеренных. Например, машинное обучение использовалось для классификации вредоносных программ для Android, для определения доменов, принадлежащих злоумышленникам, и для обнаружения URL-адресов, представляющих угрозу безопасности. Ведутся исследования систем ANN, предназначенных для тестирования на проникновение, обнаружения ботнетов, мошенничества с кредитными картами и сетевых вторжений.

- ИНС были предложены в качестве инструмента для решения уравнений в частных производных в физике и моделирования свойств открытых квантовых систем многих тел. В исследованиях мозга ИНС изучали краткосрочное поведение отдельных нейронов, динамику нейронных цепей, возникающую в результате взаимодействия между отдельными нейронами, и то, как поведение может возникать из абстрактных нейронных модулей, представляющих полные подсистемы. В исследованиях рассматривалась долгосрочная и краткосрочная пластичность нейронных систем и их связь с обучением и памятью от отдельного нейрона до системного уровня.

**3. Метод решения задачи.**

- ИНС основана на наборе связанных единиц или узлов, называемых искусственными нейронами, которые грубо моделируют нейроны в биологическом мозге. Каждое соединение, как и синапсы в биологическом мозге, может передавать сигнал другим нейронам. Искусственный нейрон получает сигналы, затем обрабатывает их и может передавать сигналы связанным с ним нейронам. «Сигнал» в соединении представляет собой действительное число, а выход каждого нейрона вычисляется с помощью некоторой нелинейной функции суммы его входов. Соединения называются ребрами. Нейроны и ребра обычно имеют вес, который корректируется по мере обучения. Вес увеличивает или уменьшает силу сигнала в соединении. Нейроны могут иметь такой порог, что сигнал отправляется только в том случае, если совокупный сигнал пересекает этот порог. Как правило, нейроны объединяются в слои. Различные уровни могут выполнять различные преобразования своих входных данных. Сигналы проходят от первого слоя (входного слоя) к последнему слою (выходному слою), возможно, после многократного прохождения слоев.

**4. Структурную схему алгоритма.**

Выход

Вход

**5. Листинг программы.**

#include <vector>

#include <iostream>

#include <cstdlib>

#include <cassert>

#include <cmath>

#include <fstream>

#include <sstream>

using namespace std;

class TrainingData

{

public:

TrainingData(const string filename);

bool isEof(void) { return m\_trainingDataFile.eof(); }

void getTopology(vector<unsigned> &topology);

unsigned getNextInputs(vector<double> &inputVals);

unsigned getTargetOutputs(vector<double> &targetOutputVals);

private:

ifstream m\_trainingDataFile;

};

void TrainingData::getTopology(vector<unsigned> &topology)

{

string line;

string label;

getline(m\_trainingDataFile, line);

stringstream ss(line);

ss >> label;

if (this->isEof() || label.compare("topology:") != 0) {

abort();

}

while (!ss.eof()) {

unsigned n;

ss >> n;

topology.push\_back(n);

}

return;

}

TrainingData::TrainingData(const string filename)

{

m\_trainingDataFile.open(filename.c\_str());

}

unsigned TrainingData::getNextInputs(vector<double> &inputVals)

{

inputVals.clear();

string line;

getline(m\_trainingDataFile, line);

stringstream ss(line);

string label;

ss>> label;

if (label.compare("in:") == 0) {

double oneValue;

while (ss >> oneValue) {

inputVals.push\_back(oneValue);

}

}

return inputVals.size();

}

unsigned TrainingData::getTargetOutputs(vector<double> &targetOutputVals)

{

targetOutputVals.clear();

string line;

getline(m\_trainingDataFile, line);

stringstream ss(line);

string label;

ss>> label;

if (label.compare("out:") == 0) {

double oneValue;

while (ss >> oneValue) {

targetOutputVals.push\_back(oneValue);

}

}

return targetOutputVals.size();

}

struct Connection

{

double weight;

double deltaWeight;

};

class Neuron;

typedef vector<Neuron> Layer;

class Neuron

{

public:

Neuron(unsigned numOutputs, unsigned myIndex);

void setOutputVal(double val) { m\_outputVal = val; }

double getOutputVal(void) const { return m\_outputVal; }

void feedForward(const Layer &prevLayer);

void calcOutputGradients(double targetVal);

void calcHiddenGradients(const Layer &nextLayer);

void updateInputWeights(Layer &prevLayer);

private:

static double eta;

static double alpha;

static double transferFunction(double x);

static double transferFunctionDerivative(double x);

static double randomWeight(void) { return rand() / double(RAND\_MAX); }

double sumDOW(const Layer &nextLayer) const;

double m\_outputVal;

vector<Connection> m\_outputWeights;

unsigned m\_myIndex;

double m\_gradient;

};

double Neuron::eta = 0.15;

double Neuron::alpha = 0.5;

void Neuron::updateInputWeights(Layer &prevLayer)

{

for (unsigned n = 0; n < prevLayer.size(); ++n) {

Neuron &neuron = prevLayer[n];

double oldDeltaWeight = neuron.m\_outputWeights[m\_myIndex].deltaWeight;

double newDeltaWeight =

eta

\* neuron.getOutputVal()

\* m\_gradient

+ alpha

\* oldDeltaWeight;

neuron.m\_outputWeights[m\_myIndex].deltaWeight = newDeltaWeight;

neuron.m\_outputWeights[m\_myIndex].weight += newDeltaWeight;

}

}

double Neuron::sumDOW(const Layer &nextLayer) const

{

double sum = 0.0;

for (unsigned n = 0; n < nextLayer.size() - 1; ++n) {

sum += m\_outputWeights[n].weight \* nextLayer[n].m\_gradient;

}

return sum;

}

void Neuron::calcHiddenGradients(const Layer &nextLayer)

{

double dow = sumDOW(nextLayer);

m\_gradient = dow \* Neuron::transferFunctionDerivative(m\_outputVal);

}

void Neuron::calcOutputGradients(double targetVal)

{

double delta = targetVal - m\_outputVal;

m\_gradient = delta \* Neuron::transferFunctionDerivative(m\_outputVal);

}

double Neuron::transferFunction(double x)

{

return tanh(x);

}

double Neuron::transferFunctionDerivative(double x)

{

return 1.0 - x \* x;

}

void Neuron::feedForward(const Layer &prevLayer)

{

double sum = 0.0;

for (unsigned n = 0; n < prevLayer.size(); ++n) {

sum += prevLayer[n].getOutputVal() \*

prevLayer[n].m\_outputWeights[m\_myIndex].weight;

}

m\_outputVal = Neuron::transferFunction(sum);

}

Neuron::Neuron(unsigned numOutputs, unsigned myIndex)

{

for (unsigned c = 0; c < numOutputs; ++c) {

m\_outputWeights.push\_back(Connection());

m\_outputWeights.back().weight = randomWeight();

}

m\_myIndex = myIndex;

}

class Net

{

public:

Net(const vector<unsigned> &topology);

void feedForward(const vector<double> &inputVals);

void backProp(const vector<double> &targetVals);

void getResults(vector<double> &resultVals) const;

double getRecentAverageError(void) const { return m\_recentAverageError; }

private:

vector<Layer> m\_layers;

double m\_error;

double m\_recentAverageError;

static double m\_recentAverageSmoothingFactor;

};

double Net::m\_recentAverageSmoothingFactor = 100.0;

void Net::getResults(vector<double> &resultVals) const

{

resultVals.clear();

for (unsigned n = 0; n < m\_layers.back().size() - 1; ++n) {

resultVals.push\_back(m\_layers.back()[n].getOutputVal());

}

}

void Net::backProp(const vector<double> &targetVals)

{

Layer &outputLayer = m\_layers.back();

m\_error = 0.0;

for (unsigned n = 0; n < outputLayer.size() - 1; ++n) {

double delta = targetVals[n] - outputLayer[n].getOutputVal();

m\_error += delta \* delta;

}

m\_error /= outputLayer.size() - 1;

m\_error = sqrt(m\_error);

m\_recentAverageError =

(m\_recentAverageError \* m\_recentAverageSmoothingFactor + m\_error)

/ (m\_recentAverageSmoothingFactor + 1.0);

for (unsigned n = 0; n < outputLayer.size() - 1; ++n) {

outputLayer[n].calcOutputGradients(targetVals[n]);

}

for (unsigned layerNum = m\_layers.size() - 2; layerNum > 0; --layerNum) {

Layer &hiddenLayer = m\_layers[layerNum];

Layer &nextLayer = m\_layers[layerNum + 1];

for (unsigned n = 0; n < hiddenLayer.size(); ++n) {

hiddenLayer[n].calcHiddenGradients(nextLayer);

}

}

for (unsigned layerNum = m\_layers.size() - 1; layerNum > 0; --layerNum) {

Layer &layer = m\_layers[layerNum];

Layer &prevLayer = m\_layers[layerNum - 1];

for (unsigned n = 0; n < layer.size() - 1; ++n) {

layer[n].updateInputWeights(prevLayer);

}

}

}

void Net::feedForward(const vector<double> &inputVals)

{

assert(inputVals.size() == m\_layers[0].size() - 1);

for (unsigned i = 0; i < inputVals.size(); ++i) {

m\_layers[0][i].setOutputVal(inputVals[i]);

}

for (unsigned layerNum = 1; layerNum < m\_layers.size(); ++layerNum) {

Layer &prevLayer = m\_layers[layerNum - 1];

for (unsigned n = 0; n < m\_layers[layerNum].size() - 1; ++n) {

m\_layers[layerNum][n].feedForward(prevLayer);

}

}

}

Net::Net(const vector<unsigned> &topology)

{

unsigned numLayers = topology.size();

for (unsigned layerNum = 0; layerNum < numLayers; ++layerNum) {

m\_layers.push\_back(Layer());

unsigned numOutputs = layerNum == topology.size() - 1 ? 0 : topology[layerNum + 1];

for (unsigned neuronNum = 0; neuronNum <= topology[layerNum]; ++neuronNum) {

m\_layers.back().push\_back(Neuron(numOutputs, neuronNum));

cout << "Made a Neuron!" << endl;

}

m\_layers.back().back().setOutputVal(1.0);

}

}

void showVectorVals(string label, vector<double> &v)

{

cout << label << " ";

for (unsigned i = 0; i < v.size(); ++i) {

cout << v[i] << " ";

}

cout << endl;

}

int main()

{

TrainingData trainData("/tmp/trainingData.txt");

vector<unsigned> topology;

trainData.getTopology(topology);

Net myNet(topology);

vector<double> inputVals, targetVals, resultVals;

int trainingPass = 0;

while (!trainData.isEof()) {

++trainingPass;

cout << endl << "Pass " << trainingPass;

if (trainData.getNextInputs(inputVals) != topology[0]) {

break;

}

showVectorVals(": Inputs:", inputVals);

myNet.feedForward(inputVals);

myNet.getResults(resultVals);

showVectorVals("Outputs:", resultVals);

trainData.getTargetOutputs(targetVals);

showVectorVals("Targets:", targetVals);

assert(targetVals.size() == topology.back());

myNet.backProp(targetVals);

cout << "Net recent average error: "

<< myNet.getRecentAverageError() << endl;

}

cout << endl << "Done" << endl;

}

**6. Результаты работы алгоритма.**

**7. Выводы.**

- Искусственная нейронная сеть (ИНС) — это парадигма обработки информации, основанная на том, как биологические нервные системы, такие как мозг, обрабатывают информацию. «Искусственная нейронная сеть» может использоваться для самых разных целей. Ключевым элементом этой парадигмы является новая структура системы обработки информации. Она состоит из большого количества взаимосвязанных элементов обработки (нейронов), работающих в унисон для решения конкретных задач. проблемы. ИНС, как и люди, учатся на примере. ИНС настраивается для конкретного приложения, такого как распознавание образов или классификация данных, посредством процесса обучения. Искусственные нейронные сети в одиночку можно рассматривать как высшую иронию человека, ищущего лучшего интеллекта и придумать его самостоятельно.Сегодня для обывателя на улице «ANN» может быть расплывчатой ​​технической аббревиатурой, еще одним термином, который отскочил от стен науки, но верите или нет, мы быстро туда движемся , где ИНС улучшает и помогает самому нашему существованию Не только нейробиология, но и психологи и инженеры также внесли свой вклад в развитие моделирования нейронных сетей.

- Чтобы научиться чему-либо, нам нужно научиться и понять это правильно, чтобы любые знания, которые мы получаем от этого, могли быть использованы в будущем. Для этого нам нужно изучить «Основы» чего-либо. Основы искусственной нейронной сети дают краткое представление об ИНС. Искусство нейронных сетей требует много тяжелой работы, поскольку данные загружаются в систему, производительность отслеживается, процессы настраиваются, добавляются соединения, изменяются правила и так далее, пока сеть не достигнет желаемых результатов. Эти желаемые результаты носят статистический характер. Сеть не всегда права. Именно по этой причине нейронные сети находят применение в приложениях, где люди также не могут всегда быть правы. Нейронные сети теперь могут выбирать акции, маркетинговые перспективы, одобрять кредиты, отказывать в кредитных картах, настраивать системы управления и проверять работу. Тем не менее, будущее сулит еще больше обещаний. Нейронным сетям нужно более быстрое оборудование. Они должны стать частью гибридных систем, которые также используют нечеткую логику и экспертные системы. Именно тогда эти системы смогут слышать речь, читать почерк и формулировать действия. Они смогут стать интеллектом роботов, которые никогда не устают и не отвлекаются. Именно тогда они станут авангардом в эпоху «интеллектуальных» машин.