МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра математического моделирования и анализа данных

ДЕРКАЧ Максим Юрьевич

ОЦЕНИВАНИЕ НАДЕЖНОСТИ КРИПТОГРАФИЧЕСКИХ ПРЕОБРАЗОВАНИЙ НА ОСНОВЕ НЕЙРОННЫХ СЕТЕЙ

Магистерская диссертация

специальность 1-31 81 12 «Прикладной компьютерный анализ данных»

Научный руководитель:
Харин Юрий Семенович,
кандидат физ.-мат. наук,
профессор,
чл.-корр НАН Беларуси

Допущена к защите

«___» _______________ 2020 г.

Зав. кафедрой ММАД
Бодягин Игорь Александрович
кандидат физ.-мат. наук, доцент

Минск, 2020

Оглавление

Реферат

Магистерская диссертация, 55 с., 13 источников, 17 изображений.

КРИПТОГРАФИЯ, НЕЙРОННАЯ КРИПТОГРАФИЯ, НЕЙРОННЫЕ СЕТИ, ПРЕОБРАЗОВАНИЕ ФЕЙСТЕЛЯ.

Объект исследования – криптографические преобразования Фейстеля.

Цель работы — построение оценок надежности криптографических преобразований Фейстеля, используя искусственные нейронные сети. Также целью работы является построение искусственных нейронных сетей, аппроксимирующих преобразование Фейстеля.

Методы исследования — построение нейронных сетей, изучение результатов компьютерных экспериментов, работа с научными материалами.

Результаты — определены математические модели криптографических примитивов и преобразования Фейстеля. Были построены нейронные сети прямого распространения с однослойными и многослойными персептронами. Также в процессе работы был построен программный комплекс генерирующий модельные данные криптографического преобразования Фейстеля и криптографических примитивов.

В результате работы получено, что искусственная нейронная сеть, может аппроксимировать криптографические примитивы с высокой точностью. Нейронные сети с многослойными персептронами аппроксимируют, в том числе и преобразование Фейстеля.

Полученные результаты могут быть применены в криптоанализе криптосистем, основанных на сетях Фейстеля.

РЭФЕРАТ

Магістарская дысертацыя, 55 с., 13 крыніц, 17 выяў.

КРЫПТАГРАФІЯ, НЕЙРОНА КРЫПТАГРАФІЯ, НЕЙРОНОВАНЫЯ СЕТКІ, ПЕРАУТВАРЭННЕ ФЕЙСТЕЛЯ.

Аб’ект даследавання - крыптаграфічныя пераўтварэння Фейстеля.

Мэта работы — пабудова адзнак надзейнасці крыптаграфічных пераўтварэнняў Фейстеля, выкарыстоўваючы нейронавыя сеткі. Таксама мэтай работы з’яўляецца пабудова нейронных сетак, якi апраксімуюць пераўтварэнне Фейстеля.

Метады даследавання — пабудова нейронавых сетак, вывучэнне вынікаў камп’ютэрных эксперыментаў, праца з навуковымі матэрыяламі.

Вынікі — вызначаны матэматычныя мадэлі крыптаграфічных прымітываў і пераўтварэнні Фейстеля. Былі пабудаваны нейронавыя сеткі прамога распаўсюджвання з аднаслаёвымі і шматслаёвымі персептронамі. Таксама ў працэсе работы быў пабудаваны праграмны комплекс генеравальны мадэльныя дадзеныя крыптаграфічнага пераўтварэння Фейстеля і крыптаграфічных прымітываў.

У выніку працы атрымана, што штучная нейронных сетку, можа аппроксимировать крыптаграфічныя прымітывы з высокай дакладнасцю. Нейронавыя сеткі з шматпластовымі персептронамі апраксімуецца, у тым ліку і пераўтварэнне Фейстеля.

Атрыманыя вынікі могуць быць ужытыя ў криптоанализе криптосистем, заснаваных на сетках Фейстеля.

GENERAL DESCRIPTION OF WORK

Master's thesis, 55 pp., 13 sources, 17 images. CRYPTOGRAPHY, NEURAL CRYPTOGRAPHY, NEURAL NETWORKS, CONVERSION OF FAYSTEL.

The object of study is the cryptographic transformations of Feistel.

The purpose of the work — the construction of reliability estimates of cryptographic transformations of Feistel using artificial neural networks. Also, the aim of the work is to build artificial neural networks approximating the Feistel transform.

Research Methods — building neural networks, studying the results of computer experiments, working with scientific materials.

Results — mathematical models of cryptographic primitives and Feistel transforms were defined. Direct distribution neural networks with single-layer and multilayer perceptrons were built. Also in the process, a software package was generated that generated model data of the Feistel cryptographic transform and cryptographic primitives.

As a result of the work, it was found that an artificial neural network can approximate cryptographic primitives with high accuracy. Neural networks with multilayer perceptrons are able to approximate the Feistel transforms.

The results can be applied in cryptanalysis of cryptosystems based on Feistel networks.

Введение

Проблема защиты информации затрагивает практически все сферы деятельности человека. И среди способов защиты информации важнейшим считается криптографический [1].

Нейронная криптография — это раздел криптографии, посвященный анализу применения стохастических алгоритмов, особенно алгоритмов искусственных нейронных сетей, для использования в шифровании и криптоанализе [2].

Идея использовать нейронные сети в криптографии нова. Впервые она была озвучена Себастьяном Дорленсом в 1995 году, спустя 30 лет после определения основ нейронных сетей. В криптоанализе используется способность нейронных сетей исследовать пространство решений. Также имеется возможность создавать новые типы атак на существующие алгоритмы шифрования, основанные на том, что любая функция может быть представлена нейронной сетью. Взломав алгоритм, можно найти решение, по крайней мере, теоретически. При этом используются такие свойства нейронных сетей, как взаимное обучение, самообучение и стохастическое поведение, а также низкая чувствительность к шуму, неточностям (искажения данных, весовых коэффициентов, ошибки в программе). Также архитектура искусственных нейронных сетей позволяет эффективно проводить работы по распознаванию образов и классификации множества объектов по любым признакам. Кроме того, благодаря хорошо продуманному алгоритму обученные нейронные сети могут достигать чрезвычайно высоких уровней точности. Они позволяют решать проблемы криптографии с открытым ключом, распределения ключей, хэширования и генерации псевдослучайных чисел. [3].

В данной работе будет рассмотрены задачи, связанные с оценкой надежности криптографических преобразований. Одной из этих задач является задача оценки стойкости итерационного преобразования Фейстеля, которая имеет важное теоретическое и практическое значение. Задача находит приложения в области криптографии и защиты данных.

Цель исследования и постановка задач

Цель исследования – оценивание надежности криптографических преобразований сети Фейстеля, используя искусственные нейронные сети.

Задачи:

1. Провести аналитический анализ работ, выполненных ранее отечественными и зарубежными исследователями, в которых были проведены исследования по теме нейронные сети в криптографии.

2. Определить задачи, которые должны решать нейронные сети в ходе исследования. И основываясь на этих задачах, определить архитектуру и параметры нейронной сети.

3. Определить математические модели криптографических примитивов преобразования Фейстеля. Разработать генератор модельных данных. Построить нейронные сети, аппроксимирующие данные математические модели. Провести компьютерные эксперименты.

4. Определить математические модель криптографического преобразования Фейстеля. Разработать генератор модельных данных. Построить нейронные сети, аппроксимирующие данную математическую модель. Провести компьютерные эксперименты.

5. Оценить результаты полученные в ходе компьютерных экспериментов.

Глава 1
Искусственные нейронные сети и их применение в криптографии

1.1 Обзор литературы

В этом разделе будет выполнен аналитический обзор литературы по теме — анализ работ, выполненных ранее отечественными и зарубежными исследователями, описание имеющихся подходов к исследованию проблемы.

Впервые исследования по использованию нейронных сетей в криптографии были проведены в 1998 году в статье [4], в которой была представлена криптографическая система на основе нейронных сетей. Другое применение нейронных сетей в криптографии была опубликована Кинзелом и Кантером, которые представили способ использования нейронных сети при обмене секретными ключами по общедоступному каналу [5]. Значительный вклад в использование нейронных сетей в криптографии внес Ли в [6], который предложил использовать нейронные сети для оптимизации дифференциального и линейного криптоанализа. В 2009 году предложили использовать сигмоидальную функцию в качестве активационной функции в нейронной сети, используемой при криптоанализе шифров Фейстеля [7].

В 2012 была опубликовано работа [8], в которой нейронные сети успешно использовались для аппроксимации DES и Triple-DES. В этой работе была использована модель представленная на рисунке 1.1.1. Модель описывает атаку с открытым текстом. Эта атака основана на обучение нейронной сети, для последующего процесса расшифровки, не зная ключа используемого при шифровании.

В нейронную сеть подается зашифрованный текст в качестве входных данных, и открытый текст в качестве эталона. После достаточного количества тренировок, с достаточным количеством пар открытого текста и шифротекста, который зашифрован неизменняем ключом, предпологается, что нейронная сеть сможет расшифровать шифротекст, который не был представлен в обучающей выборке, при условии того, что он зашифрован на том же ключе, что и шифротекст из обучающей выборки.

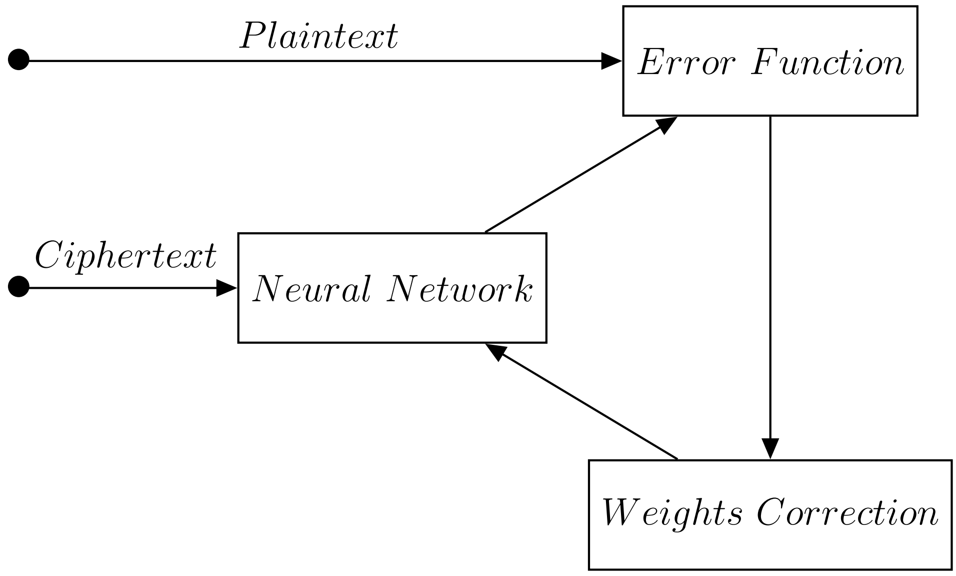

Рисунок 1.1.1 — Модель криптоанализа на основе нейронной сети

Исследования, которые использовали данную модель, показали высокий результат. Поэтому в нашем исследовании мы будем использовать схожую модель. Так как процесс шифрования и расшифрования в сети Фейстеля симметричен, то наша модель криптоанализа будет использовать модель симметричную предыдущей. Данная модель представлена на рисунке 1.1.2.

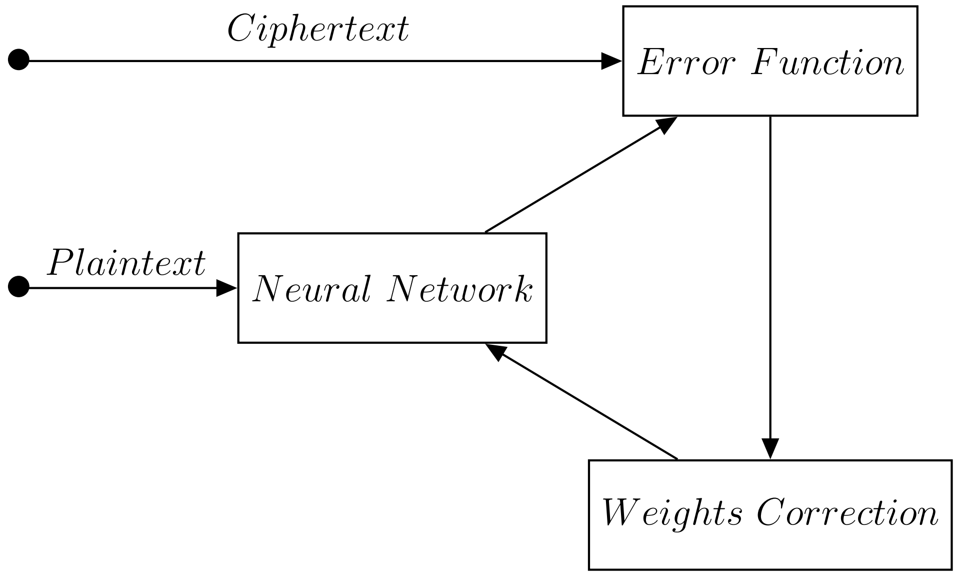

Рисунок 1.1.2 — Модель криптоанализа на основе нейронной сети

1.2 Искусственные нейронные сети: архитектура, параметры, программные средства

В данном разделе мы рассмотрим основные типы нейронных сетей, используемых в нейронной криптографии. А также в подразделе 1.2.3 проведем исследования для определения наилучшей модели искусственной нейронной сети и ее параметров для аппроксимации криптографических преобразований.

Для проведения исследований по определению наилучшей модели искусственной нейронной сети определим задачи, которые должна решить искомая нейронная сеть:

1. Ни одно криптографическое преобразование ни обходиться без таких бинарных операций как, циклический сдвиг вправо ($\ggg$), циклический сдвиг влево ($\lll$), операция побитового исключения ($\oplus$), операция сложения по модулю ($\boxplus$) и применения блока подстановок (S-блока, S-box).

   Следовательно, искомой нейронной сети необходимо уметь аппроксимировать данные операции.

2. Также в основе большинства криптографических преобразований лежат неизвестные (секретные) параметры, например ключ или S-блок. Поэтому необходимо подобрать нейронную сеть и её параметры такие, что сеть сможет на тренировочных данных научиться !имитировать эти неизвестные параметры.

3. И последняя, но не менее важная задача — это точность аппроксимации криптографического преобразования.

1.2.1 Архитектура нейронных сетей

Сперва опишем архитектуры нейронных сетей, которые используются в нейронной криптографии.

В нейронной криптографии широкое применение получили следующие архитектуры нейронных сетей:

1. Нейронные сети прямого распространения.

2. Рекуррентные нейронные сети.

Нейронные сети прямого распространения

В нейронных сетях прямого распространения (feedforward neural network) все связи направлены строго от входных нейронов к выходным. Примерами таких сетей являются перцептрон Розенблатта, многослойный перцептрон, сети Ворда. Особое место в нейронной криптографии занимает модель многослойного перцептрона.

Многослойный персептрон (MLP) — это способ объединения персептронов для построения классификатора для более сложных наборов данных.

Многослойный персептрон включает в себя следующие типы слоев:

1. Входной слой: в традиционной модели этот слой является только промежуточным между входными данными и остальной частью сети. Таким образом, выход из нейронов, принадлежащих к этому слою, – это просто сам входной вектор.

2. Скрытый слой: этот слой направлен на введение некоторой нелинейности в модель так что MLP сможет соответствовать нелинейному сепарабельному набору данных. Действительно, если данные, которые должны быть изучены линейно разделимы, нет необходимости в каких-либо скрытых слоях. В зависимости от нелинейности и сложности модели данных, число нейронов на скрытом слое или даже количество этих слоев можно увеличить. Однако, один скрытый слой достаточен для большого количества естественных проблем. Что касается количества нейронов на скрытых слоях, было продемонстрировано, что использование огромного количества нейронов может привести к переобучению – достижению хороших результатов на обучающей выборке и плохих – на других данных.

3. Выходной слой: это последний слой сети. Выходные данные узлов на этом слое непосредственно сопоставляются с классами, которые пользователь намеревается предсказать.

Рекуррентные нейронные сети

Рекуррентные нейронные сети — вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать серии событий во времени или последовательные пространственные цепочки. В отличие от многослойных персептронов, рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Поэтому сети RNN применимы в таких задачах, где нечто целостное разбито на части, например: распознавание рукописного текста или распознавание речи.

1.2.2 Параметры нейронных сетей

Можно выделить следующие параметры нейронной сети:

1. функция активации,
2. функция потерь,
3. количество нейронов на скрытых слоях.

Функция активации

Функция активации нейрона определяет выходное значение нейрона в зависимости от результата взвешенной суммы входов и порогового значения. Основная "функция"функции активации ввести нелинейность в выход нейрона. Если в нейронное сети будет отсутствовать функция активации, то выходной сигнал будет линейной функцией. Следовательно такая нейронная сеть не сможет изучать и моделировать (аппроксимировать) сложные данные или сложные математические модели.

Выделяют следующие функции активации:

1. сигмоидальная функция активации: $\frac{1}{1+e^{-x}}$,
2. ReLU (Rectified Linear Unit): $max(0, x)$,
3. Leaky ReLU: $max(0.01x, x)$,
4. тангенс (tanh): $tanh(x)$.

Функция потерь

Функция потерь используется для расчета ошибки между эталонными и предсказанными выходными векторами. Основная цель нейронной сети – минимизировать эту ошибку. Таким образом, функция потерь эффективно приближает обучение нейронной сети к этой цели. Функция потерь измеряет «насколько хороша» нейронная сеть в отношении данной обучающей выборки и ожидаемых ответов.

Наиболее популярные функции потерь:

1. квадратичная (среднеквадратичное отклонение):
   $MSE(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$,
2. бинарная кросс-энтропия:
   $J(w) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) = -\frac{1}{N}\sum_{n=1}^{n}[y_n \ log\hat{y_n} + (1 - y_n) \ log(1 - \hat{y_n})]$,

3. среднее абсолютное отклонение:
   $MAE(w) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$.

1.2.3 Оценивание нейронных сетей и ее параметров

Сперва, рассмотрим бинарные операции (элементы криптографических преобразований) и в подследствии определим нейронные сети аппроксимирующие их.

Рассмотрим n-мерное векторное пространство V_n над полем их двух элементов $\mathbb{F}_2$. Операция сложения в поле $\mathbb{F}_2$ обозначается символом $\oplus$. Пусть $a = (a_1, ..., a_n),\ b = (b_1, ..., b_n) \in V_n$, тогда вектор a будем отождествлять с числом $\bar{a} = 2^{n-1}a_1 + ... + 2a_{n-1} + a_n$. Тогда результат операции $a \boxplus b$ есть вектор $c \in V_n$ такой, что $\bar{c} = (\bar{a} + \bar{b})\ mod\ 2^n$ [9].

Лемма 1.1. *Операцию $\boxplus$ можно представить в следующем виде:*

$$a \boxplus b = (\bar{a} + \bar{b}) - z * 2^n,\ a, b \in V_n,\ z \in \{0, 1\}. \tag{1.1}$$

Доказательство:

$$\bar{a} + \bar{b} = 2^n c_1 + 2^{n-1}c_2 + ... + 2c_n + c_{n+1} = \bar{c}.$$

$$(\bar{a} + \bar{b})\ mod\ 2^n = 2^{n-1}c_2 + ... + 2c_n + c_{n+1} = \bar{c} - 2^n c_1 = \bar{a} + \bar{b} - 2^n c_1.$$

$$a \boxplus b = (\bar{a} + \bar{b}) - c_1 * 2^n.$$

Очевидно, что операция $\boxplus$ — нелинейная операция. И исходя из этого можно сделать предположение, что однослойная нейронная сеть не сможет аппроксимировать данную операцию исходя из своих свойств и нелинейности операции $\boxplus$. И, следовательно, из свойств многослойного перцептрона можно что сделать предположение, что нейронная сеть с такой архитектурой с легкостью предскажет (аппроксимирует) данную операцию.

Так как операция циклического сдвига и применение S-блока — это частный случай операции перестановки, то можно ввести следующие предположение.

Предположение 1.1. *Сложность аппроксимации операции циклического сдвига и применение S-блока сравнима со сложностью аппроксимации операции перестановки.*

Основываясь на следующие исследования [10] [11] и статью [12], будем считать, что так как операция побитового исключения ($\oplus$) - нелинейна, то однослойная нейронная сеть не способна аппроксимировать данную операцию. Но было доказано что, нейронные сети со скрытыми слоями при достаточном количестве обучающих данных могут аппроксимировать $\oplus$ с высокой долей точности.

Так же в данном исследовании [13] было показано, что нейронная сеть с многослойным персептроном способна успешно аппроксимировать операцию перестановки.

Основываясь на сделанных предположениях в дальнейших исследованиях мы будем использовать нейронную сеть прямого распространения с многослойным персептроном. А для проверки наших теоретических предположений, мы построим однослойную нейронную сеть и сравним две этих модели перцептрона. Если наши теоретически предположения верны, тогда нейронная сеть с многослойным персептроном должна показать точность значительно выше однослойной сети.

Глава 2
Аппроксимация криптографических примитивов преобразования Фейстеля

2.1 Математическое описание криптографических примитивов

В данной главе проведем исследования, в которых мы попытаемся аппроксимировать криптографические примитивы преобразования Фейстеля. В дальнейшем полученные результаты будут использованы в Главе 3.

Рассмотрим частный случай сети Фейстеля ГОСТ 28147-89. Опишем однотактовое преобразование шифрования ГОСТ 28147-89:

$Y = g(X, K) = g(X_1||X_2, K) \equiv (S[X_1 \boxplus K] \lll 11) \oplus X_2||X_1$, где
$X \in V_{64}$ - вектор входных данных,
$Y \in V_{64}$ - выходные данные,
$K \in V_{32}$ - ключ,
$\lll 11$ - циклический сдвиг влево на 11 бит,
S - стандартный S-блок из ГОСТ-28147.

Определим следующие математические модели, для которых в дальнейшем будут построены нейронные сети и проведены компьютерные эксперименты.

1. $Y_{g_0} = g_0(x) = g_0(x_1||x_2) \equiv x_1 \oplus x_2$, где

   $x \in V_8$ - вектор входных данных,

   $x_1, x_2 \in V_4$ - левая и правая часть входного вектора,

   $Y_{g_0} \in V_4$ - выходные данные модели g_0;

2. $Y_{g_1} = g_1(x) = g_1(x_1||x_2, k) \equiv S[x_1] \oplus x_2$, где

   $x \in V_8$ - вектор входных данных,

   $x_1, x_2 \in V4$ - левая и правая часть входного вектора,

   $Y_{g_1} \in V_4$ - выходные данные модели g_1,

   S - первый узел из стандартного S-блока из ГОСТ-28147

   $(S = \{13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7\})$;

3. $Y_{g_2} = g_2(x) = g_2(x_1||x_2, k) \equiv S[x_1 \boxplus k] \oplus x_2$, где

   $x \in V_8$ - вектор входных данных,

   $x_1, x_2 \in V4$ - левая и правая часть входного вектора,

   $k \in V_4$ - некоторый неизвестный постоянный в эксперименте ключ,

   $Y_{g_2} \in V_4$ - выходные данные модели g_2,

   S - первый узел из стандартного S-блока из ГОСТ-28147;

4. $Y_{g_4} = g_4(x) \equiv x \boxplus K$, где

   $x \in V_4$ - вектор входных данных,

   $k \in V_4$ - некоторый неизвестный постоянный в эксперименте ключ,

   $Y_{g_4} \in V_4$ - выходные данные модели g_4;

5. $Y_{g_8} = g_8(x) \equiv x \boxplus K$, где

   $x \in V_8$ - вектор входных данных,

   $k \in V_8$ - некоторый неизвестный постоянный в эксперименте ключ,

   $Y_{g_8} \in V_8$ - выходные данные модели g_8;

6. $Y_{g_{16}} = g_{16}(x) \equiv x \boxplus K$, где

   $x \in V_{16}$ - вектор входных данных,

   $k \in V_{16}$ - некоторый неизвестный постоянный в эксперименте ключ,

   $Y_{g_{16}} \in V_{16}$ - выходные данные модели g_{16};

7. $Y_{g_{32}} = g_{32}(x) \equiv x \boxplus K$, где

   $x \in V_{32}$ - вектор входных данных,

   $k \in V_{32}$ - некоторый неизвестный постоянный в эксперименте ключ,

   $Y_{g_{32}} \in V_{g_{32}}$ - выходные данные модели g_{32}.

Опишем определённые ранее модели как частный случай однотактового преобразования шифрования ГОСТ 28147-89:

1. Модель g_0 является однотактовым преобразованием шифрования ГОСТ 28147-89, при следующих условиях:

   - S-блок - прямая таблица подстановки (подстановка при которой исходное значение переходит в такое же значение).
   - $K = 0^{32}$.

- Отсутствует сдвиг влево на 11 бит.
- Рассматриваются только первые 4-бита X_1, X_2, Y.

2. Модель g_1 является однотактовым преобразованием шифрования ГОСТ 28147-89, при следующих условиях:

   - S-блок - стандартный S-блок из ГОСТ-28147.
   - $K = 0^{32}$.
   - Отсутствует сдвиг влево на 11 бит.
   - Рассматриваются только первые 4-бита X_1, X_2, Y.

3. Модель g_2 является однотактовым преобразованием шифрования ГОСТ 28147-89, при следующих условиях:

   - S-блок - стандартный S-блок из ГОСТ-28147.
   - K - некоторый неизвестный постоянный в эксперименте ключ.
   - Отсутствует сдвиг влево на 11 бит.
   - Рассматриваются только первые 4-бита X_1, X_2, Y.

4. Модель g_4 является однотактовым преобразованием шифрования ГОСТ 28147-89, при следующих условиях:

   - S-блок - прямая таблица подстановки (подстановка при которой исходное значение переходит в такое же значение).
   - K - некоторый неизвестный постоянный в эксперименте ключ.
   - Отсутствует сдвиг влево на 11 бит.
   - Левая часть входного вектора отсутствует, либо равна 0^{32}.
   - Рассматриваются только первые 4-бита X, Y, K.

5. Модель g_8 является однотактовым преобразованием шифрования ГОСТ 28147-89, при следующих условиях:

   - S-блок - прямая таблица подстановки (подстановка при которой исходное значение переходит в такое же значение).
   - K - некоторый неизвестный постоянный в эксперименте ключ.
   - Отсутствует сдвиг влево на 11 бит.
   - Левая часть входного вектора отсутствует, либо равна 0^{32}.
   - Рассматриваются только первые 8-бита X, Y, K.

6. Модель g_{16} является однотактовым преобразованием шифрования ГОСТ 28147-89, при следующих условиях:

- S-блок - прямая таблица подстановки (подстановка при которой исходное значение переходит в такое же значение).
- K - некоторый неизвестный постоянный в эксперименте ключ.
- Отсутствует сдвиг влево на 11 бит.
- Левая часть входного вектора отсутствует, либо равна 0^{32}.
- Рассматриваются только первые 16-бита X, Y, K.

7. Модель g_{32} является однотактовым преобразованием шифрования ГОСТ 28147-89, при следующих условиях:

   - S-блок - прямая таблица подстановки (подстановка при которой исходное значение переходит в такое же значение).
   - K - некоторый неизвестный постоянный в эксперименте ключ.
   - Отсутствует сдвиг влево на 11 бит.
   - Левая часть входного вектора отсутствует, либо равна 0^{32}.
   - Рассматриваются только первые 32-бита X, Y, K.

2.2 Аппроксимация криптографических примитивов с помощью нейронных сетей

2.2.1 Описание условий компьютерного эксперимента и используемых нейронных сетей

Основываясь на предположения из главы 2, чтобы аппроксимировать определённые ранее модели, необходимо построить искусственную нейронную сеть, с одним скрытым слоем.

Чтобы подтвердить наши теоретические предположения, построим следующие нейронные сети:

1. однослойную нейронную сеть,
2. многослойная нейронная сеть с одним скрытым слоем, с переменным количеством нейронов на скрытом слое.

Компьютерные эксперименты для определённых ранее моделей проведем по следующему плану:

1. Генерируем модельные данные с помощью разработанного генератора (обучающую и экзаменационную выборку).
2. Строим однослойную нейронной сети, оцениваем точность данной нейронной сети.
3. Если точность равна 100%, принимаем решение что однослойная нейронная сеть аппроксимирует данную модель.
4. В обратном случае строим график зависимости точности построенной нейронной сети с одним скрытым слоем от количества нейронов на скрытом слое.
5. Анализируем полученный график и находим нейронную сеть с минимальным количеством нейронов на скрытом слое, точность аппроксимации которой равна или близка 100%.
6. Для найденной нейронной сети строим график зависимости точность от количества итераций обучения.

Для оценки точность построенной нейронной сети использовалось расстояние Хэмминга:

$$w(y, \hat{y}) = \sum_{i=1}^{j} y_i \oplus \hat{y}_i,\ y_i \in V_j, V \in \{0, 1\}. \tag{2.1}$$

Для оценки точности проведенного численного эксперимента использовалось следующая функция:

$$\hat{f} = L - \frac{1}{T_e} \sum_{j=1}^{T_e} w(y^{(j)}, y^{\hat{(j)}}), \tag{2.2}$$

где L - количество бит выходного вектора оцениваемой модели,
T_e - размер тестовой (экзаменационной) выборки,
$y^{\hat{(j)}}$ - выходной вектор предсказанный нейронной сетью,
$y^{(j)}$ - эталонный выходной вектор.

В качестве функции потерь в процессе обучения нейронной сети использовалось среднеквадратичное отклонение (для многослойного персептрона) и кросс-энтропия (для однослойного персептрона).

В качестве функции активации использовалась сигмоидальная функция:

$$\phi(z) = \frac{1}{1 + e^{-z}}. \tag{2.3}$$

2.2.2 Описание данных эксперимента

Компьютерные эксперименты проводились на следующих данных:

1. Модель g_0 :
   - обучающая выборка $T_o = 128$ пар (x, y);
   - экзаменационная выборка $T_e = 24$ пар (x, y).
2. Модель g_1 :
   - обучающая выборка $T_o = 128$ пар (x, y);
   - экзаменационная выборка $T_e = 24$ пар (x, y).
3. Модель g_2 :
   - обучающая выборка $T_o = 128$ пар (x, y);
   - экзаменационная выборка $T_e = 24$ пар (x, y).
4. Модель g_4 :
   - обучающая выборка $T_o = 10$ пар (x, y);
   - экзаменационная выборка $T_e = 5$ пар (x, y).
5. Модель g_8:
   - обучающая выборка $T_o = 128$ пар (x, y);
   - экзаменационная выборка $T_e = 24$ пар (x, y).
6. Модель g_{16}:
   - обучающая выборка $T_o = 512$ пар (x, y);
   - экзаменационная выборка $T_e = 102$ пар (x, y).
7. Модель g_{32}:
   - обучающая выборка $T_o = 2048$ пар (x, y);
   - экзаменационная выборка $T_e = 409$ пар (x, y).

2.3 Численные результаты

2.3.1 Модель g_0

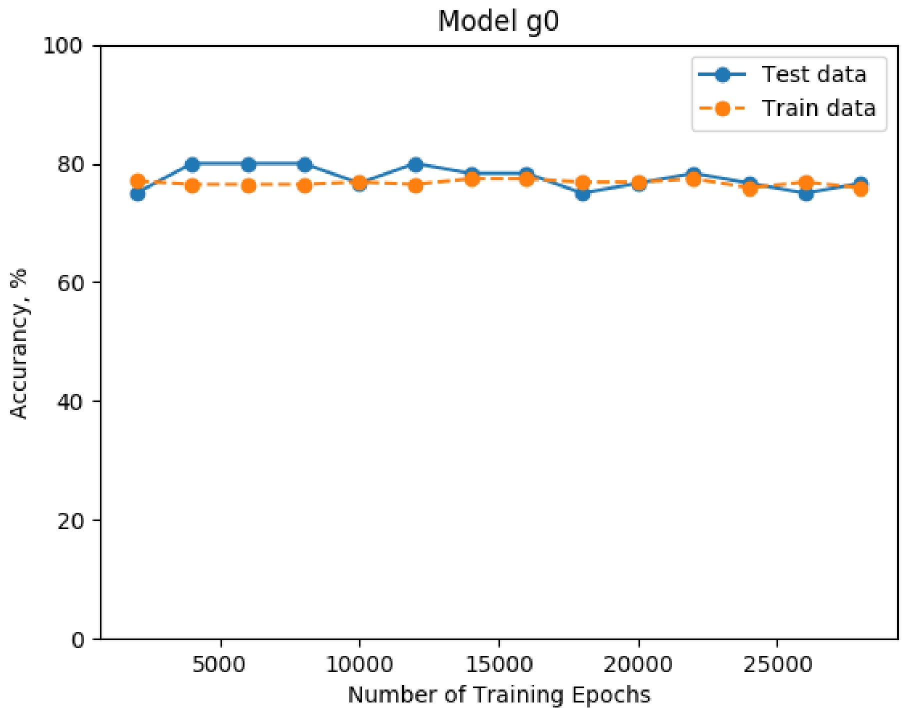

Рисунок 2.3.1 — График точности построенной однослойной нейронной сети модели g_0 от количества итераций обучения

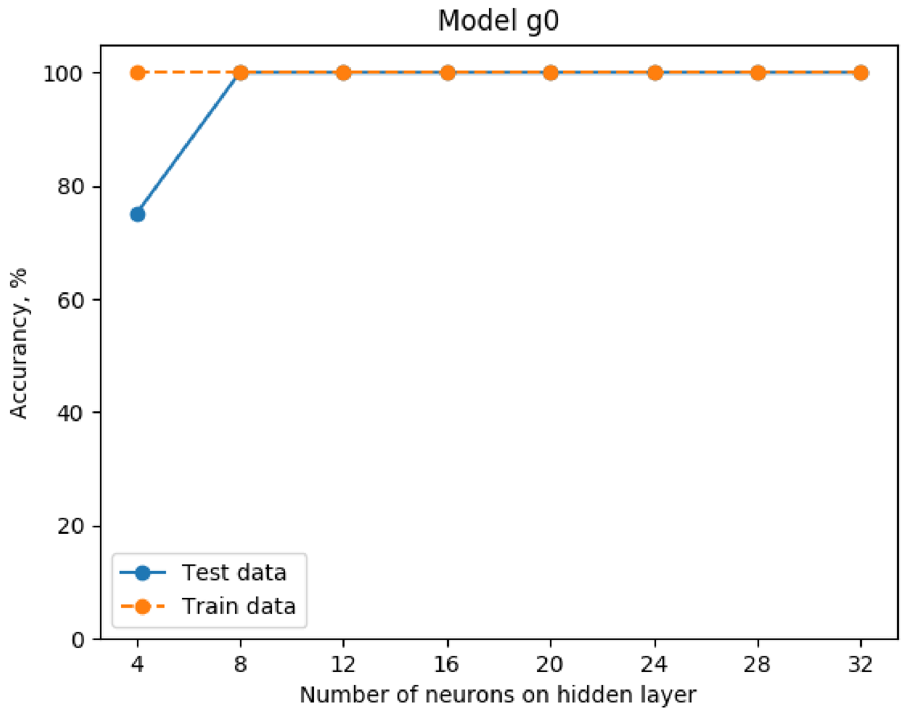

Рисунок 2.3.2 — График точности построенной нейронной сети с одним скрытым слоем модели g_0 от количества нейронов на скрытом слое

Из графика 2.3.2 видно, что нейронная сеть с одним скрытым слоем, с 8 нейронами на скрытом слое, показывает необходимую точность. Поэтому построим график зависимости точности данной нейронной сети от количества итераций обучения.

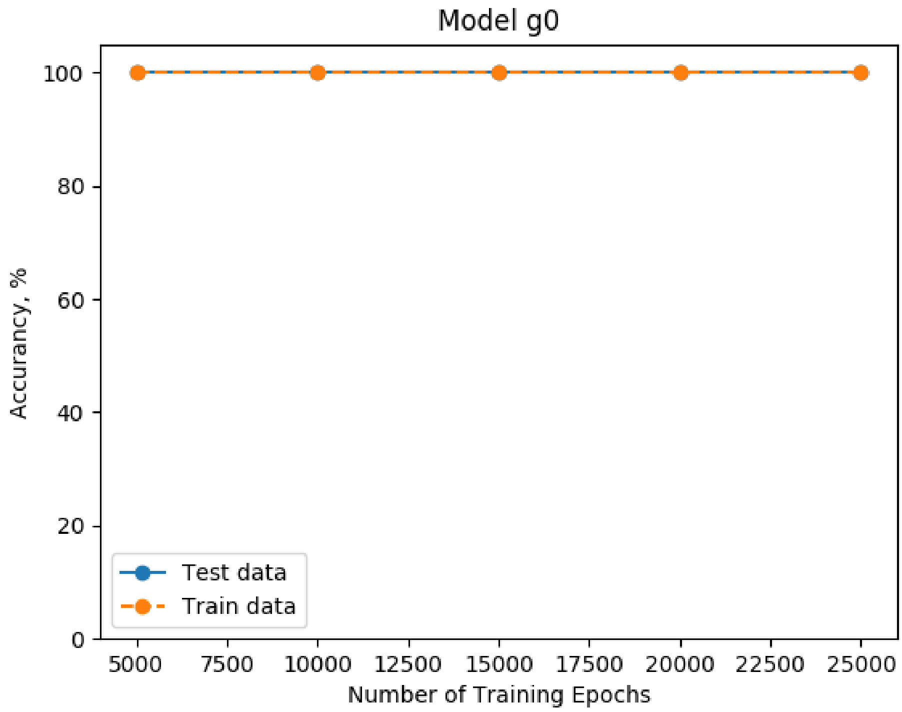

Рисунок 2.3.3 — График точности построенной нейронной сети с одним скрытым слоем модели g_0 от количества итераций обучения

2.3.2 Модель g_1

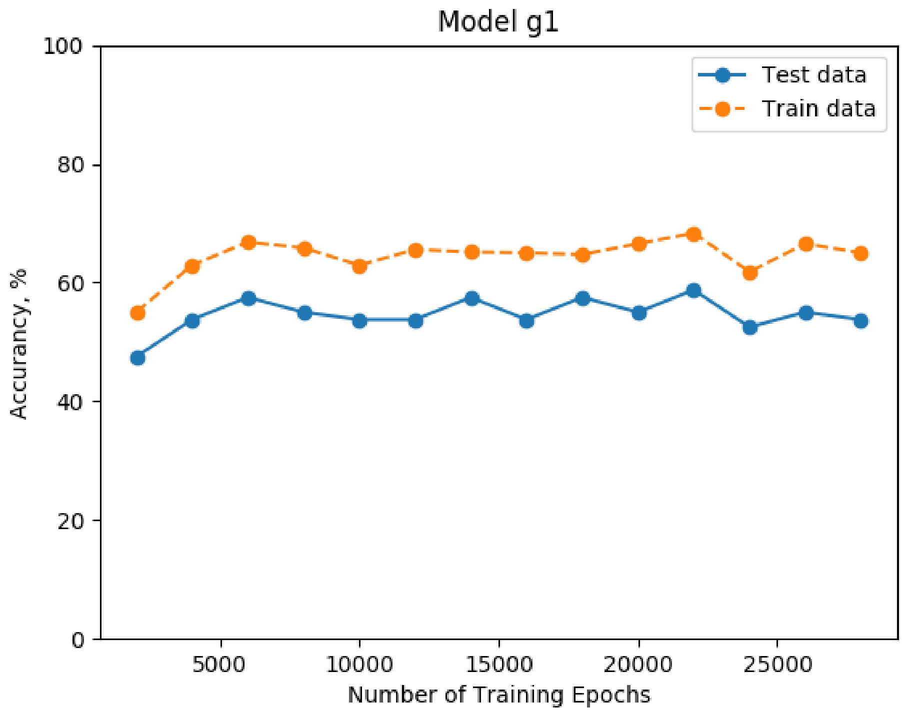

Рисунок 2.3.4 — График точности построенной однослойной нейронной сети модели g_1 от количества итераций обучения

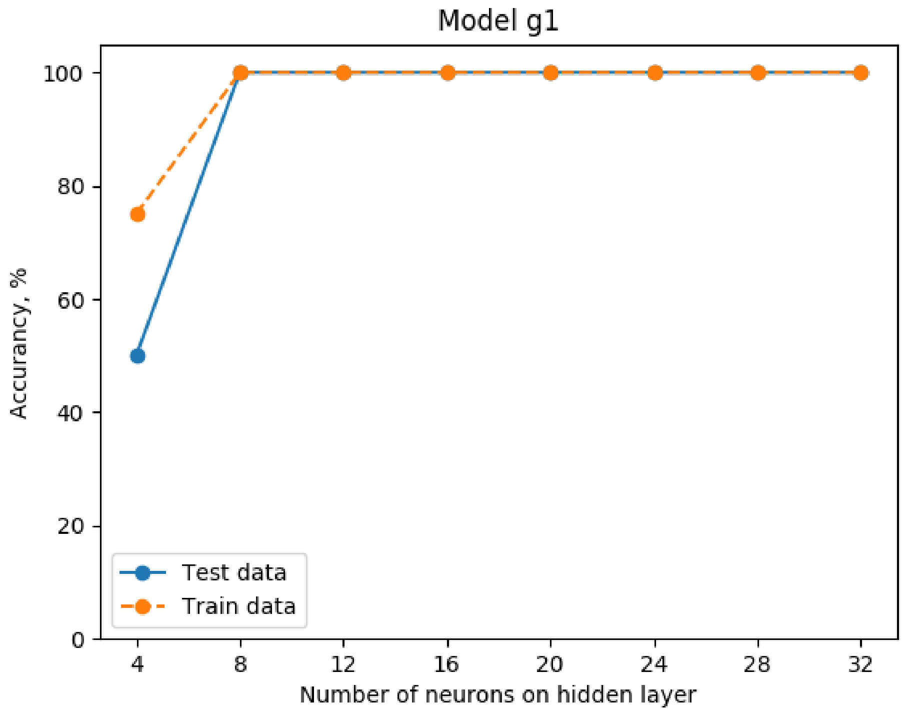

Рисунок 2.3.5 — График точности построенной нейронной сети с одним скрытым слоем модели g_1 от количества нейронов на скрытом слое

Из графика 2.3.5 видно, что нейронная сеть с одним скрытым слоем, с 8 нейронами на скрытом слое, показывает необходимую точность. Поэтому построим график зависимости точности данной нейронной сети от количества итераций обучения.

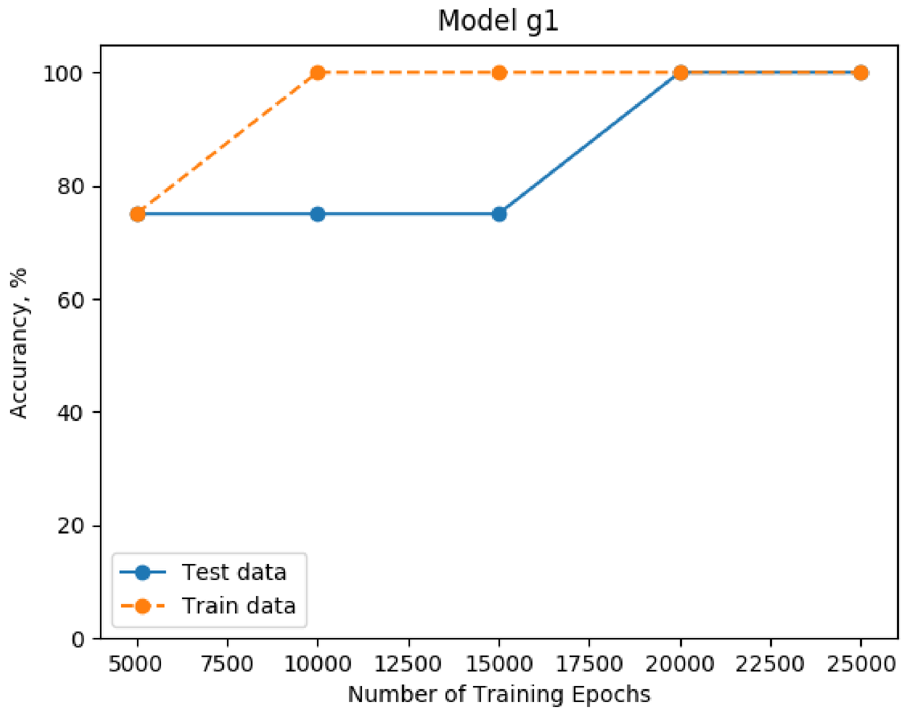

Рисунок 2.3.6 — График точности построенной нейронной сети с одним скрытым слоем модели g_1 от количества нейронов на скрытом слое

2.3.3 Модель g_2

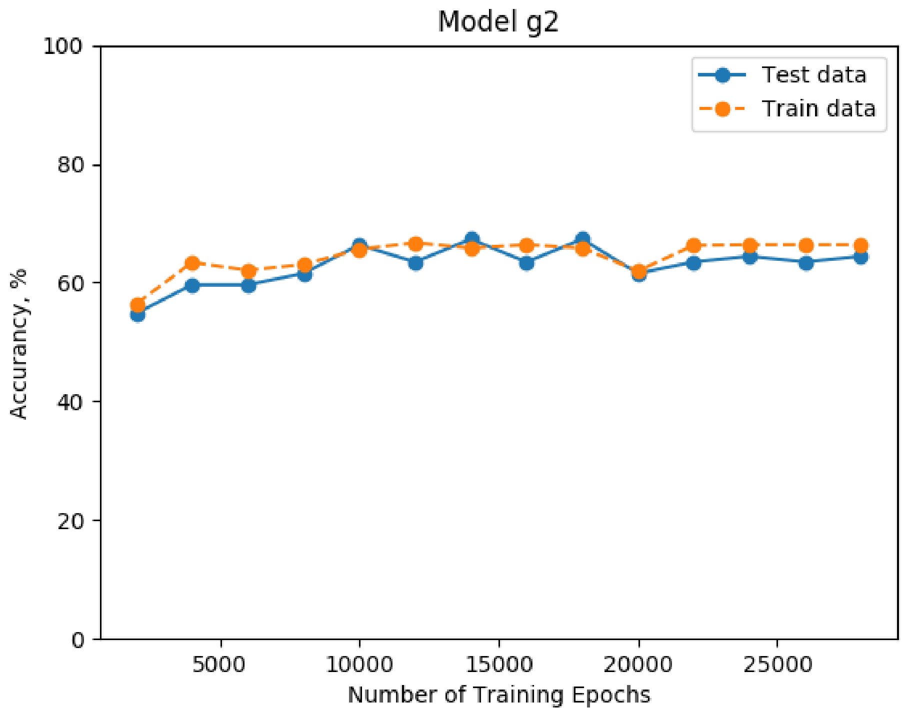

Рисунок 2.3.7 — График точности построенной однослойной нейронной сети модели g_2 от количества итераций обучения

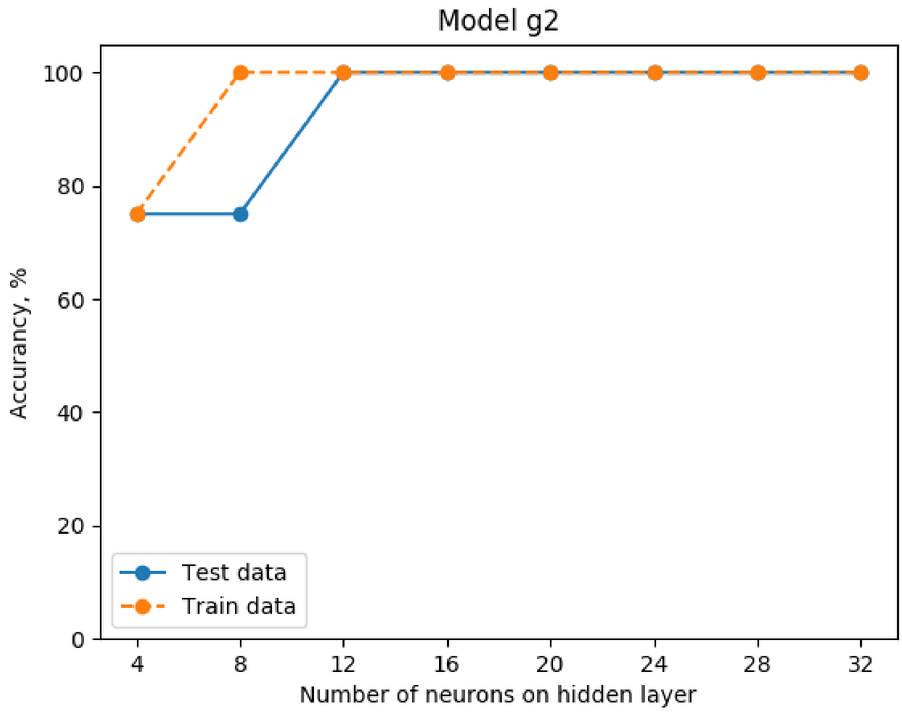

Рисунок 2.3.8 — График точности построенной нейронной сети с одним скрытым слоем модели g_2 от количества нейронов на скрытом слое

Из графика 2.3.8 видно, что нейронная сеть с одним скрытым слоем, с 8 нейронами на скрытом слое, показывает необходимую точность. Поэтому построим график зависимости точности данной нейронной сети от количества итераций обучения.

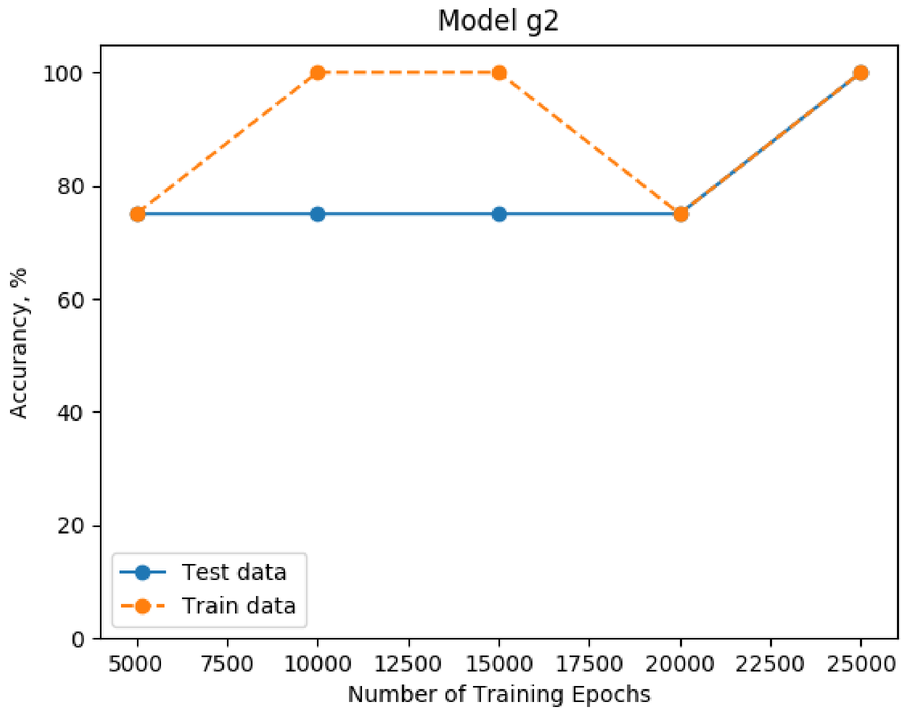

Рисунок 2.3.9 — График точности построенной нейронной сети с одним скрытым слоем модели g_0 от количества нейронов на скрытом слое

2.3.4 Модель g_4

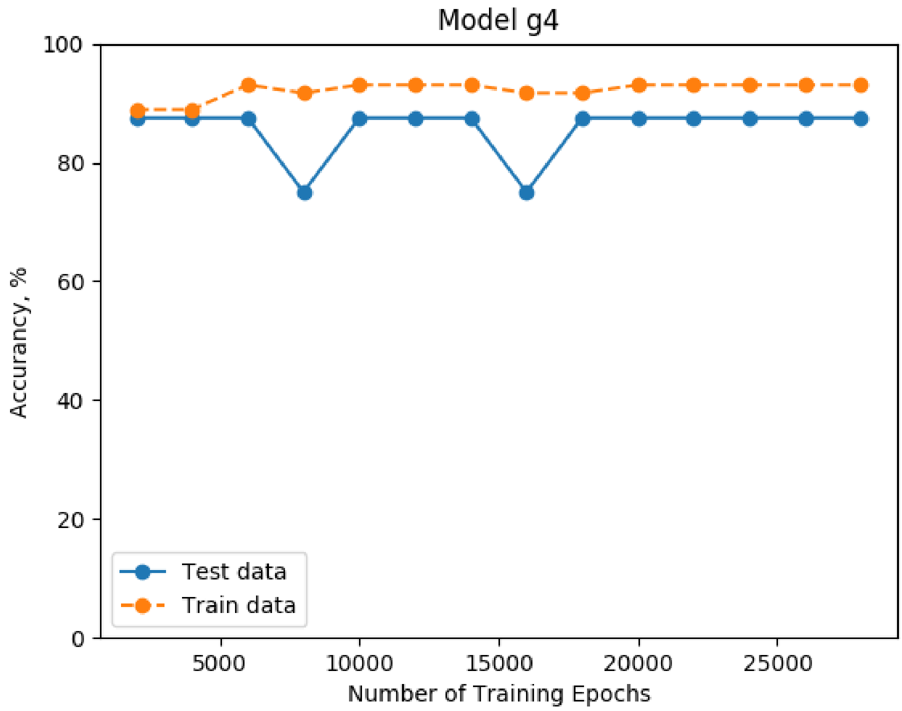

Рисунок 2.3.10 — График точности построенной однослойной нейронной сети модели g_4 от количества итераций обучения

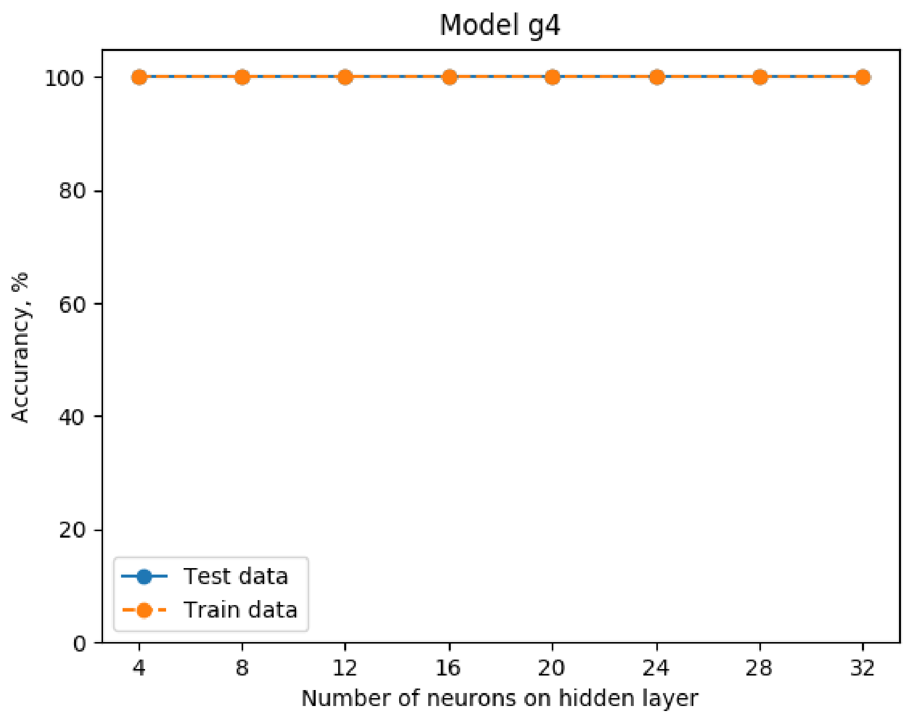

Рисунок 2.3.11 — График точности построенной нейронной сети с одним скрытым слоем модели g_4 от количества нейронов на скрытом слое

Из графика 2.3.11 видно, что нейронная сеть с одним скрытым слоем, с 4 нейронами на скрытом слое, показывает необходимую точность. Поэтому построим график зависимости точности данной нейронной сети от количества итераций обучения.я.

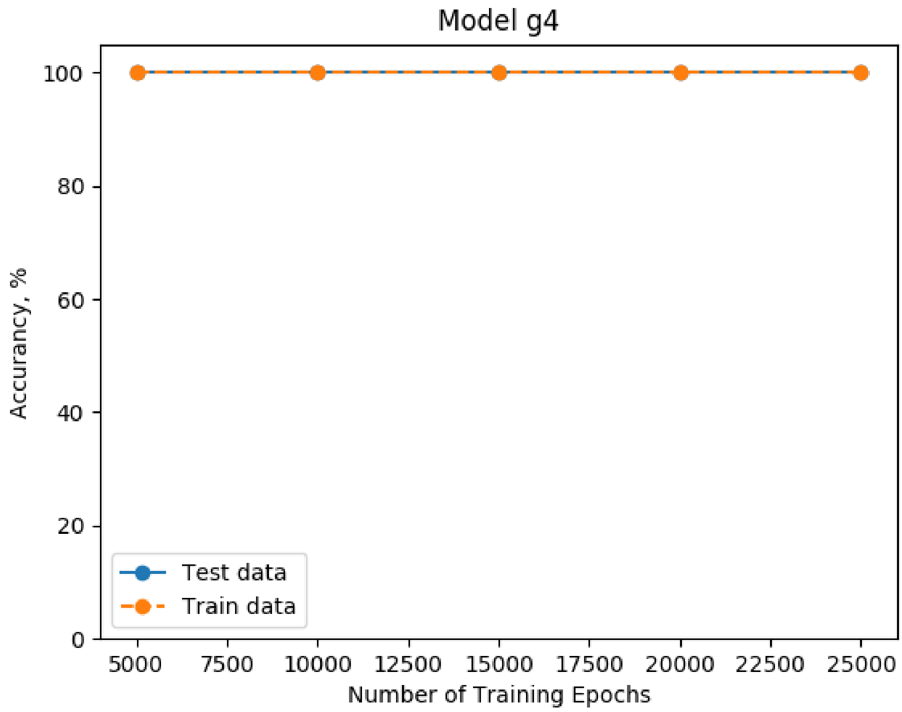

Рисунок 2.3.12 — График точности построенной нейронной сети с одним скрытым слоем модели g_4 от количества нейронов на скрытом слое

2.3.5 Модель g_8

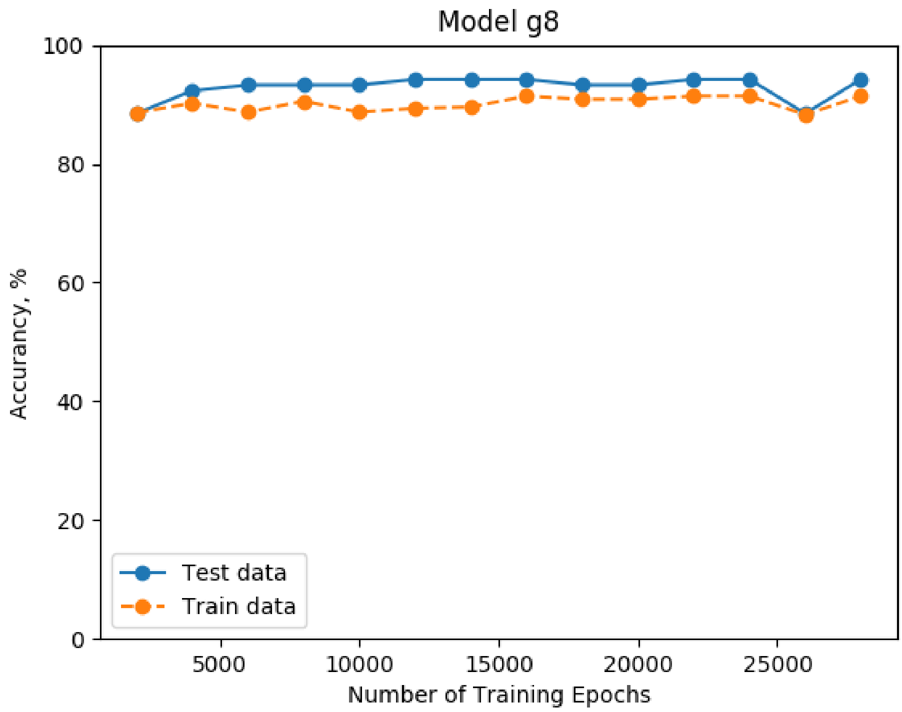

Рисунок 2.3.13 — График точности построенной однослойной нейронной сети модели g_8 от количества итераций обучения

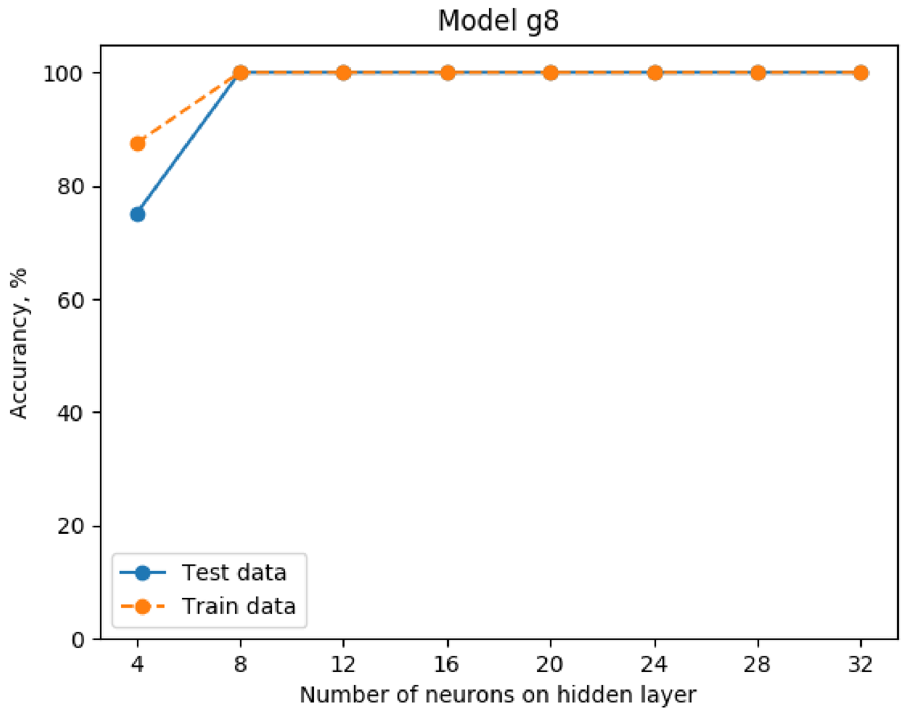

Рисунок 2.3.14 — График точности построенной нейронной сети с одним скрытым слоем модели g_8 от количества нейронов на скрытом слое

Из графика 2.3.14 видно, что нейронная сеть с одним скрытым слоем, с 8 нейронами на скрытом слое, показывает необходимую точность. Поэтому построим график зависимости точности данной нейронной сети от количества итераций обучения.

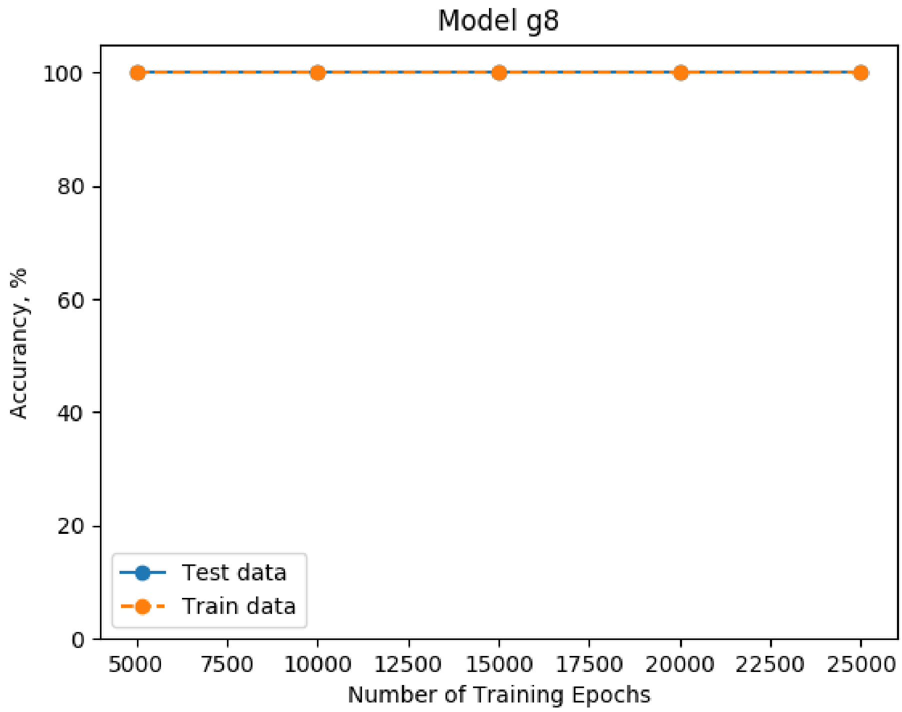

Рисунок 2.3.15 — График точности построенной нейронной сети с одним скрытым слоем модели g_8 от количества нейронов на скрытом слое

2.3.6 Модель g_{16}

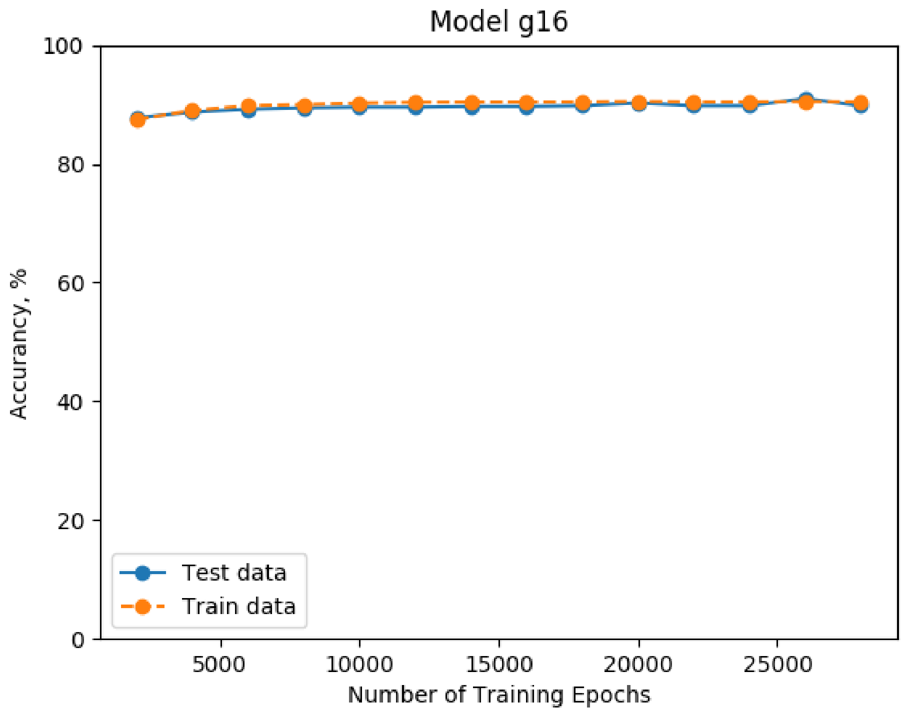

Рисунок 2.3.16 — График точности построенной однослойной нейронной сети модели g_{16} от количества итераций обучения

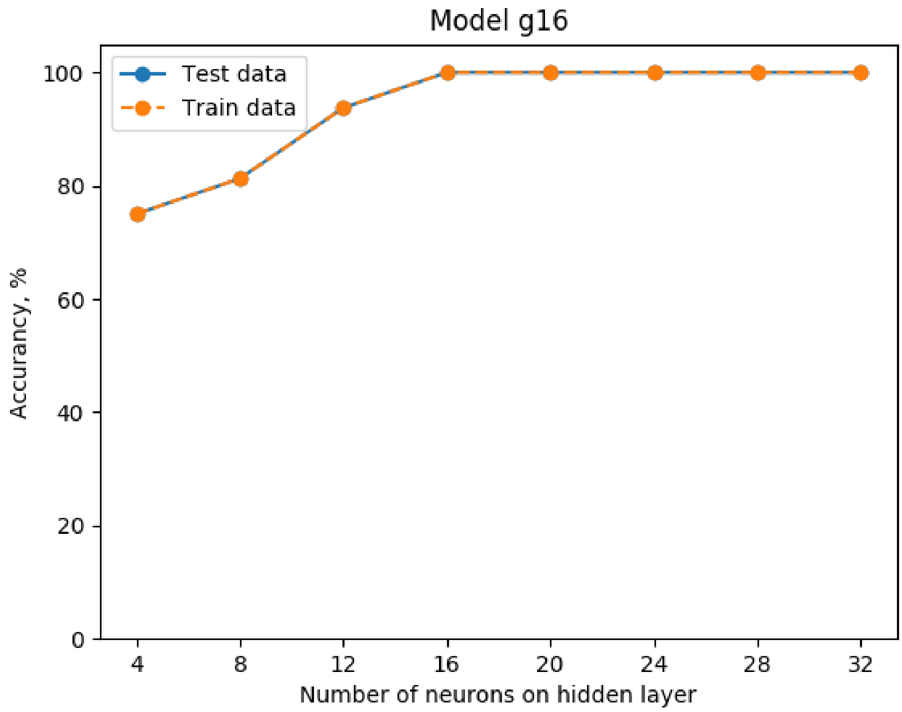

Рисунок 2.3.17 — График точности построенной нейронной сети с одним скрытым слоем модели g_{16} от количества нейронов на скрытом слое

Из графика 2.3.17 видно, что нейронная сеть с одним скрытым слоем, с 16 нейронами на скрытом слое, показывает необходимую точность. Поэтому построим график зависимости точности данной нейронной сети от количества итераций обучения.

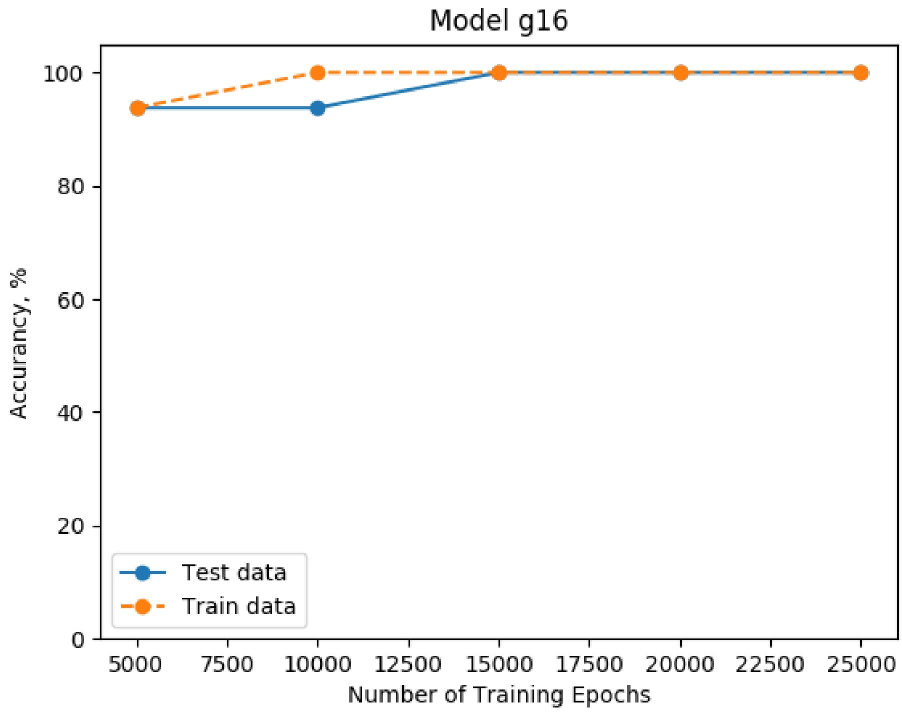

Рисунок 2.3.18 — График точности построенной нейронной сети с одним скрытым слоем модели g_{16} от количества нейронов на скрытом слое

2.3.7 Модель g_{32}

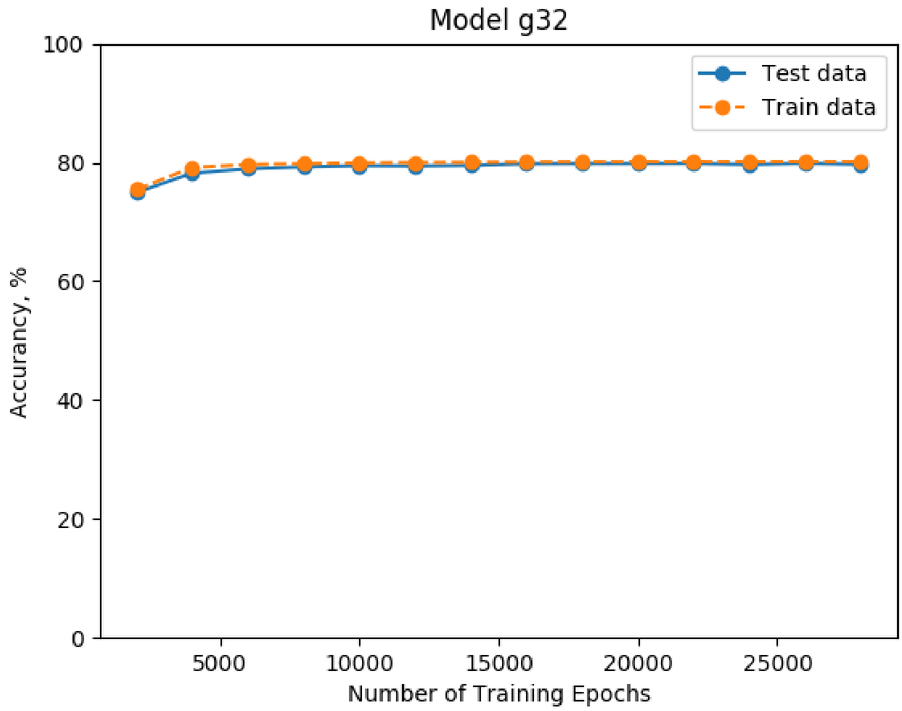

Рисунок 2.3.19 — График точности построенной однослойной нейронной сети модели g_{32} от количества итераций обучения

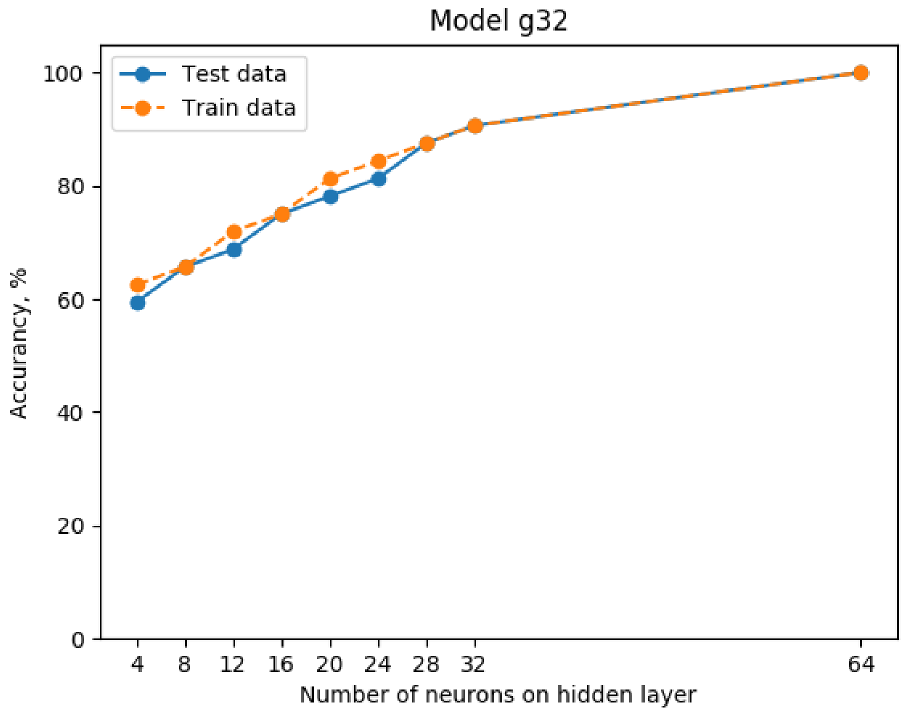

Рисунок 2.3.20 — График точности построенной нейронной сети с одним скрытым слоем модели g_{32} от количества нейронов на скрытом слое

Из графика 2.3.20 видно, что нейронная сеть с одним скрытым слоем, с 64 нейронами на скрытом слое, показывает необходимую точность. Поэтому построим график зависимости точности данной нейронной сети от количества итераций обучения.

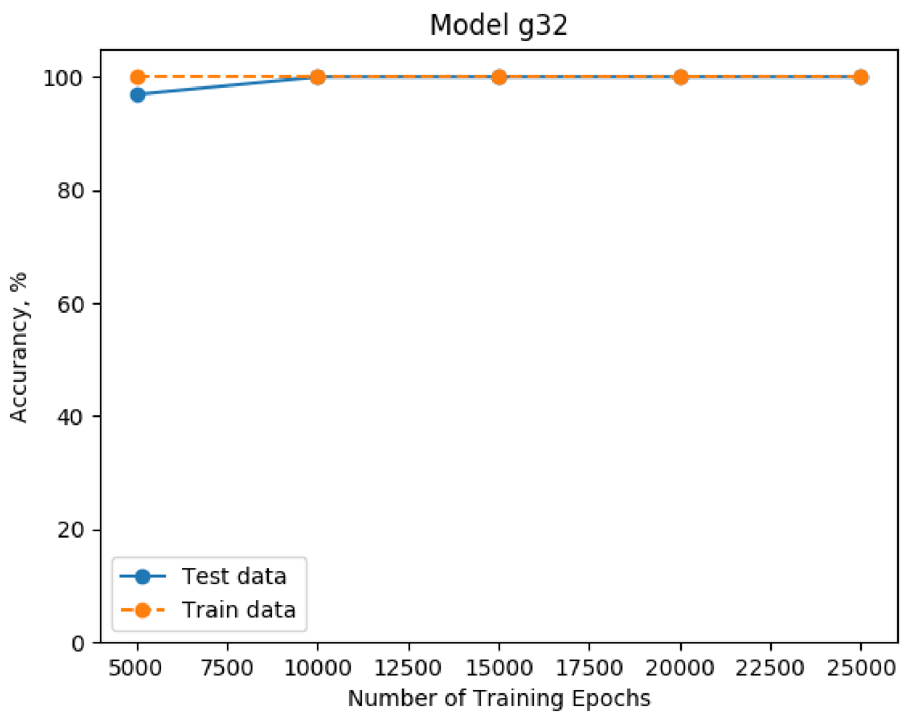

Рисунок 2.3.21 — График точности построенной нейронной сети с одним скрытым слоем модели g_{32} от количества нейронов на скрытом слое

2.3.8 Обобщение полученных результатов

Основываясь на проведенных экспериментах построим таблицу в которой сравним модели и построенный нейронные сети с минимальным количеством параметров. В таблице представим следующие параметры сравнения: размер обучающей выборки, размер тестовой выборки, параметры построенной нейронной сети, точность нейронной сети, потери при обучении, время затраченное на обучение.

Введем следующие обозначения:

- N - количество нейронов на скрытом слое,
- NP - число обучаемых параметров сети,
- T_o - размер обучающей выборки,
- T_e - размер тестовой (экзаменационной) выборки,
- $\hat{f}$ - точность построенной нейронной сети,
- l - потери при обучении,
- Ψ - время обучения нейронной сети в секундах,

- Σ - количество эпох обучения.

Модель	НС	N	NP	T_o	T_e	$\hat{f}$	l	Σ	Ψ
g_0	NN	0	32	128	24	3.2000	0.5927	4000	1.1266
g_0	MNN	8	96	128	24	4.0000	0.0631	5000	1.7937
g_1	NN	0	32	128	24	2.3500	0.6519	22000	7.2215
g_1	MNN	8	96	128	24	4.0000	0.1136	20000	7.1536
g_2	NN	0	32	128	24	2.6923	0.6338	14000	6.4174
g_2	MNN	8	96	128	24	4.0000	0.1547	25000	9.3551
g_4	NN	0	16	10	5	3.5000	0.6237	2000	3.6132
g_4	MNN	4	32	10	5	4.0000	0.0918	5000	4.7954
g_8	NN	0	64	128	24	7.5385	0.5015	12000	8.2843
g_8	MNN	8	128	128	24	8.0000	0.0603	5000	5.3191
g_{16}	NN	0	256	512	129	14.5577	0.5335	26000	16.2864
g_{16}	MNN	16	512	512	129	16.0000	0.0469	15000	10.5211
g_{32}	NN	0	1024	2048	409	25.5520	0.1447	26000	21.0100
g_{32}	MNN	64	4096	2048	409	32.0000	0.0211	10000	13.2120

Таблица 2.1 — Сравнение построенных нейронных сетей

Из представленных выше результатов (графики и таблица 2.1), можно сделать вывод, что точность нейронной сети с одним скрытым слоем значительно выше точности обычной нейронной сети прямого распространения. Следовательно, наши теоретические предположения были верны.

Глава 3
Оценка надежности криптографического преобразования Фейстеля с помощью его аппроксимации нейронной сетью

3.1 Описание используемой модели криптографического преобразования Фейстеля

В данной главе рассмотрим оценку надежности криптографического преобразования Фейстеля с помощью его аппроксимации нейронной сетью.

Для оценки стойкости итерационного проебразования определим согласно [1] модельное однотактовое криптографическое преобразования Фейстеля. Примем следующие обозначения:

- $V = 0, 1$;
- I - количество тактов;
- N - размерность преобразования Фейстеля;
- $X = (x_i) \in V^{2N}, X = (X_1 \| X_2) \in V^{2N}, X_i \in V^N, i = 1, 2$;
- $Y = (y_i) \in V^{2N}, Y = (Y_1 \| Y_2) \in V^{2N}, Y_i \in V^N, i = 1, 2$;
- $< X >, < Y > \in \{0, 1, ..., 2^{2N} - 1\}$ - числовое представление векторов X, Y;
- $K = (K1 \| ... \| K_8) = (k_1, ..., k_{8N})$ - ключ тактового преобразования;
- $K_i = (k_{(i-1)N+1}, k_{(i-1)N+2}, ..., k_{iN}) \in V^N, i = 1, ..., 8$ - i-ый подключ преобразования;
- $< K_i > \in \{0, 1, ..., 2^N - 1\}$ - числовое представление двоичного вектора K_i;
- $\lll L$ - циклический сдвиг влево на L бит;
- $f(\cdot) : V^N \times V^N \to V^N$ - функция криптографического преобразования Фейстеля;
- $g(\cdot) : V^N \to V^N$ - расписание ключей.

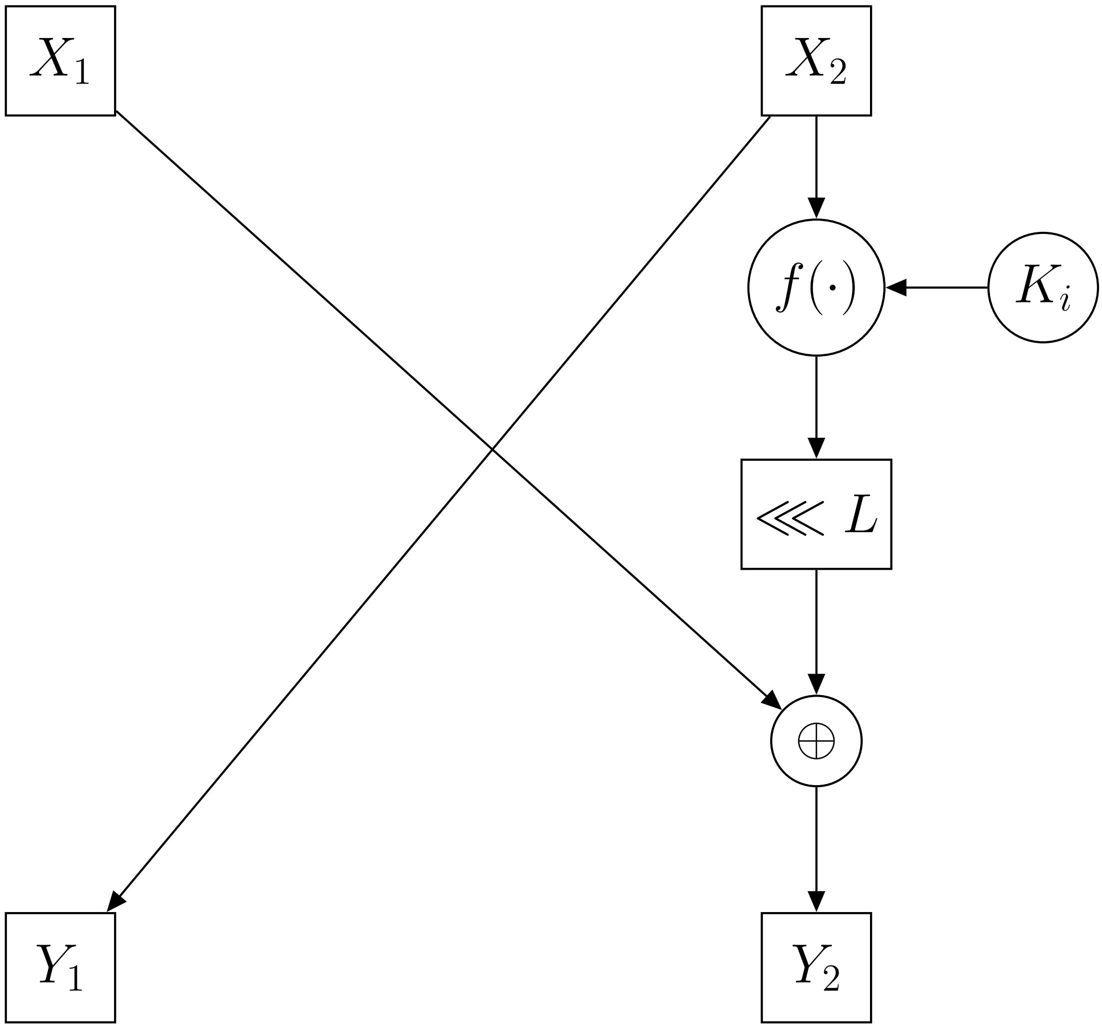

Рисунок 3.1.1 — Модельное 1-тактовое преобразование с i-ым подключом

В данном исследовании будем использовать использовать следующию предположения:

1. Размерность преобразования Фейстеля:

   $N = 8$.

2. Функция криптографического преобразования Фейстеля:

   $f(X_2; K) = S[X_2 \boxplus K]$, где S - два первых стандартных S-блока из ГОСТ-28147.

3. Расписание ключей:

   $g(K) = K_1, ..., K_8; K_1, ...K_8; ...; K_1, ...K_8$.

4. Параметры L, I - изменяемые параметры в следующих диапозонах:

   - $L \in \{0, 1, ..., 7\}$,
   - $I \in \{1, 2, ..., 8\}$.

Определим полученные математические модели:
f_{I-L} — преобразования Фейстеля,
где $I \in \{1, 2, ..., 8\}$ — количество тактов,
$L \in \{0, 1, ..., 7\}$ — количество битов при циклическом сдвиге влево.

3.2 Описание используемых нейронных сетей

3.2.1 Описание условий компьютерного эксперимента и используемых нейронных сетей

Для решения поставленных задач использовались следующие нейронные сети:

1. однослойная нейронная сеть,
2. многослойная нейронная сеть с одним скрытым слоем, с переменным количеством нейронов на скрытом слое,
3. многослойная нейронная сеть с двумя скрытым слоем, с переменным количеством нейронов на скрытом слое,
4. многослойная нейронная сеть с тремя скрытым слоем, с переменным количеством нейронов на скрытом слое.

Для оценки точность построенной нейронной сети использовалось расстояние Хэмминга:

$$w(y, \hat{y}) = \sum_{i=1}^{j} y_i \oplus \hat{y}_i,\ y_i \in V_j, V \in \{0, 1\}. \tag{3.1}$$

Для оценки точности проведенного компьютерного эксперимента использовалось следующая функция:

$$\hat{f} = L - \frac{1}{T_e} \sum_{j=1}^{T_e} w(y^{(j)}, \hat{y^{(j)}}), \tag{3.2}$$

где L - количество бит выходного вектора оцениваемой модели,
T_e - размер тестовой (экзаменационной) выборки,
$\hat{y^{(j)}}$ - выходной вектор предсказанный нейронной сетью,
$y^{(j)}$ - эталонный выходной вектор.

В качестве функции потерь в процессе обучения нейронной сети использовалось среднеквадратичное отклонение.

В качестве функции активации использовалась сигмоидальная функция:

$$\phi(z) = \frac{1}{1 + e^{-z}}. \tag{3.3}$$

3.2.2 Описание данных эксперимента

Компьютерные эксперименты проводились на следующих данных:

- обучающая выборка $T_o = 10240$ пар (x, y);
- экзаменационная выборка $T_e = 1024$ пар (x, y).

3.3 Численные результаты

3.3.1 Модели f_1

Рассмотрим модели преобразования Фестеля с одним тактом при изменяющемся параметре L. Построим однослойную нейронную сеть и нейронную сеть с одним скрытым слоем, с 32 нейронами на данном слое. Для данных моделей и нейронных сетей построим графики точности, и сравним полученные результаты.

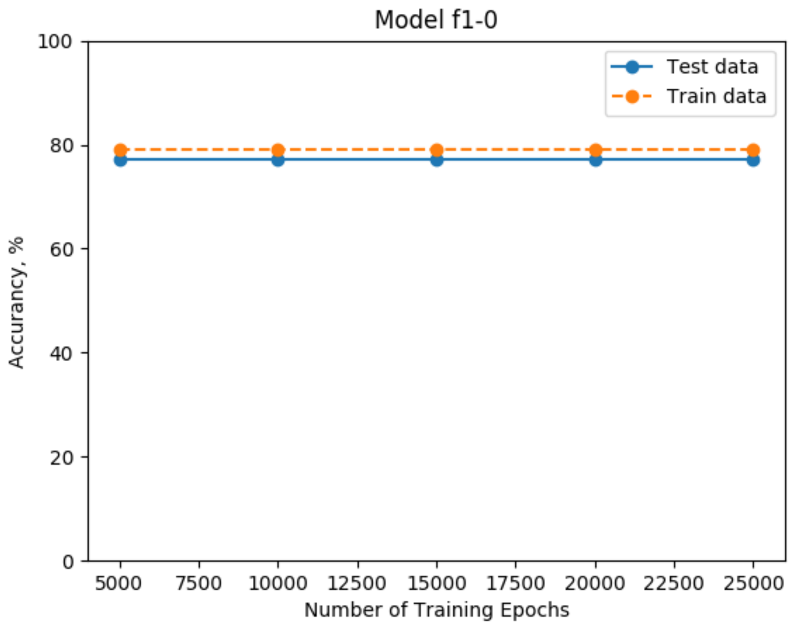

Рисунок 3.3.2 — График точности построенной однослойной нейронной сети модели f_{1-0} от количества итераций обучения

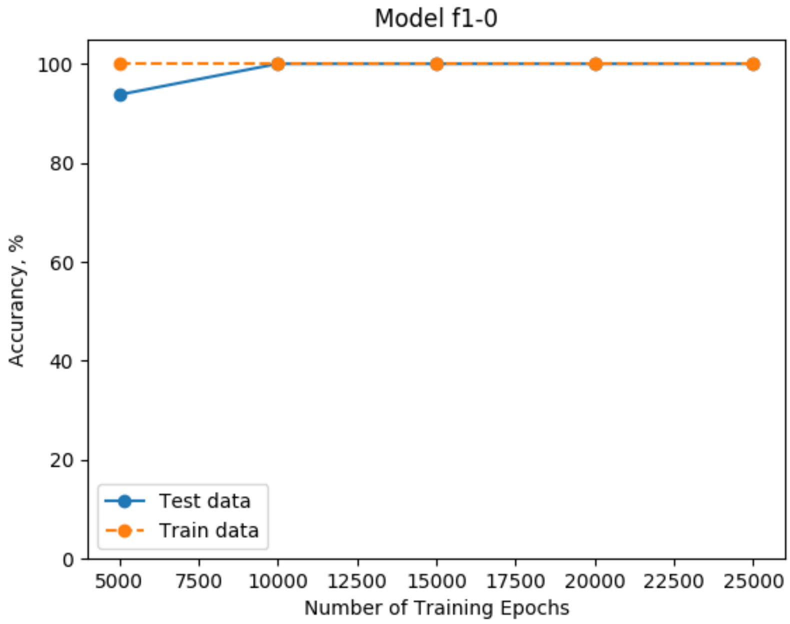

Рисунок 3.3.3 — График точности построенной нейронной сети с одним скрытым слоем модели f_{1-0} от количества итераций обучения

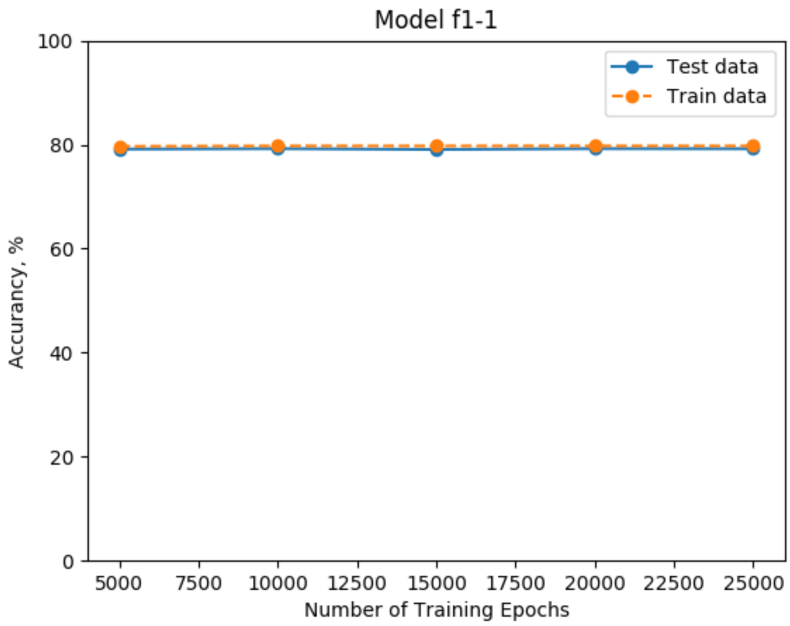

Рисунок 3.3.4 — График точности построенной однослойной нейронной сети модели f_{1-1} от количества итераций обучения

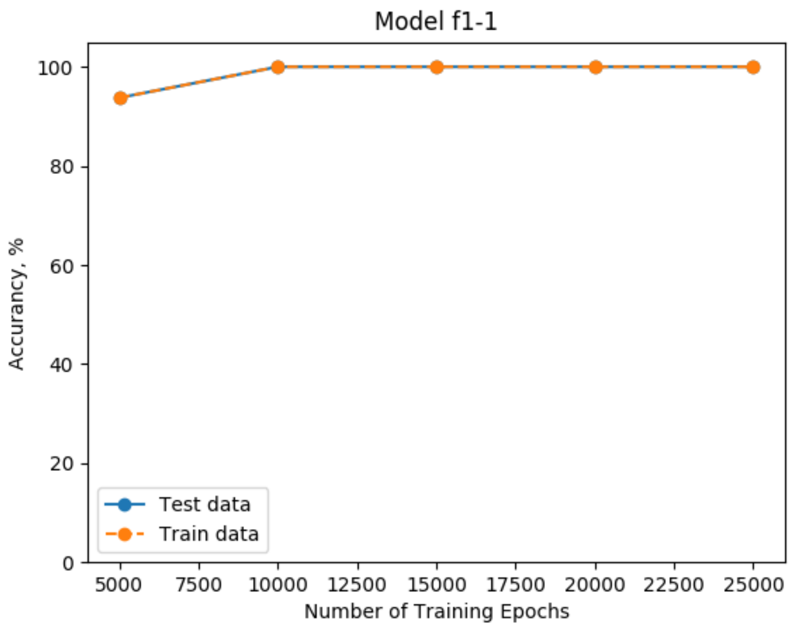

Рисунок 3.3.5 — График точности построенной нейронной сети с одним скрытым слоем модели f_{1-1} от количества итераций обучения

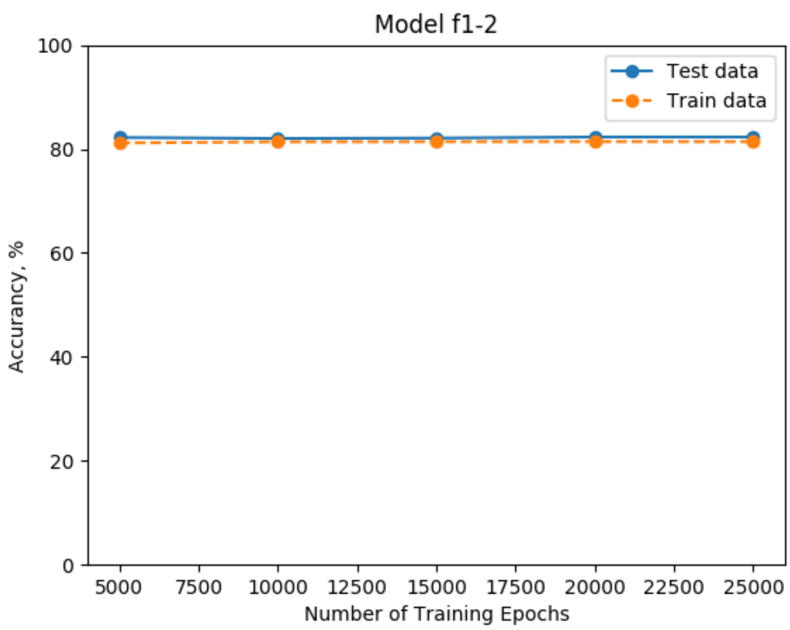

Рисунок 3.3.6 — График точности построенной однослойной нейронной сети модели f_{1-2} от количества итераций обучения

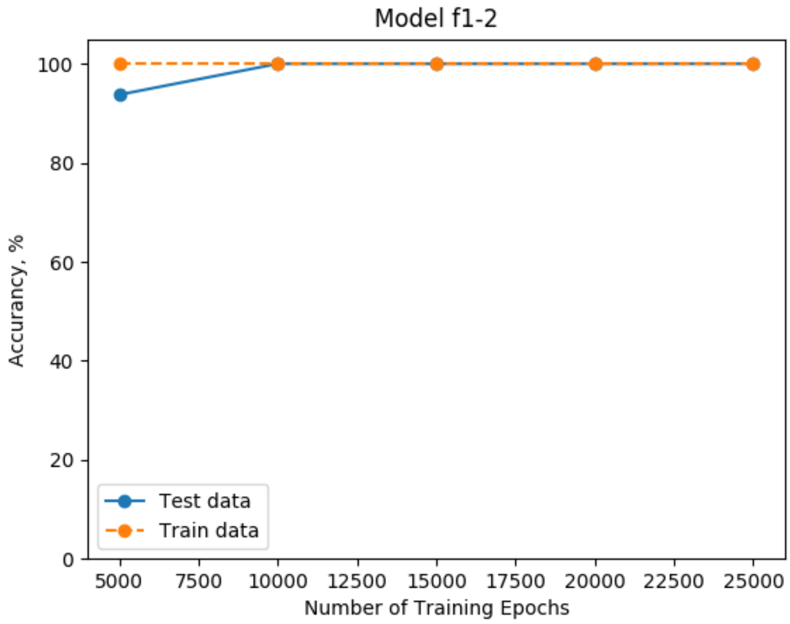

Рисунок 3.3.7 — График точности построенной нейронной сети с одним скрытым слоем модели f_{1-2} от количества итераций обучения

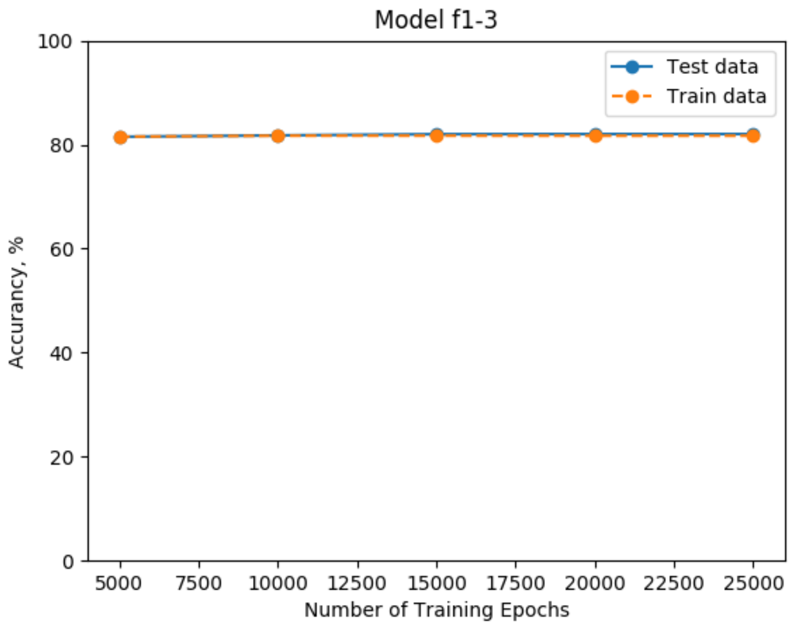

Рисунок 3.3.8 — График точности построенной однослойной нейронной сети модели f_{1-3} от количества итераций обучения

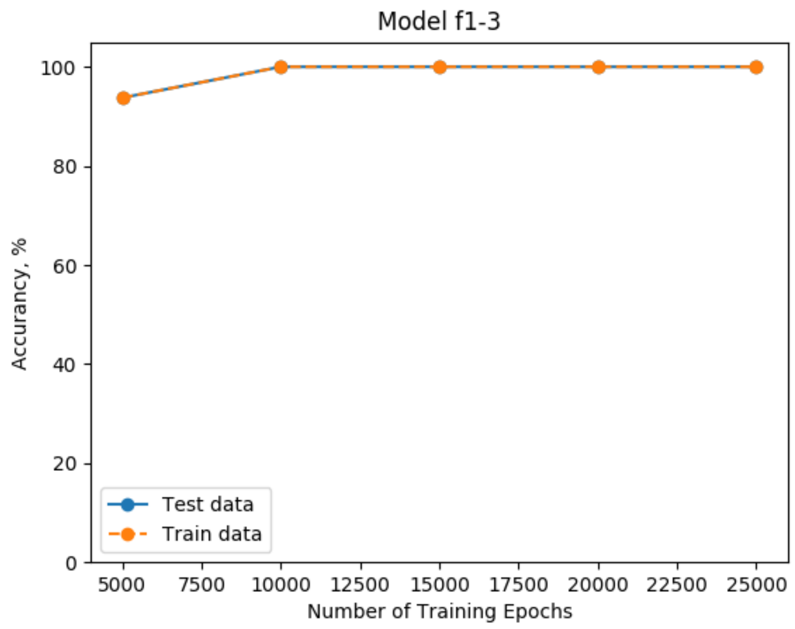

Рисунок 3.3.9 — График точности построенной нейронной сети с одним скрытым слоем модели f_{1-3} от количества итераций обучения

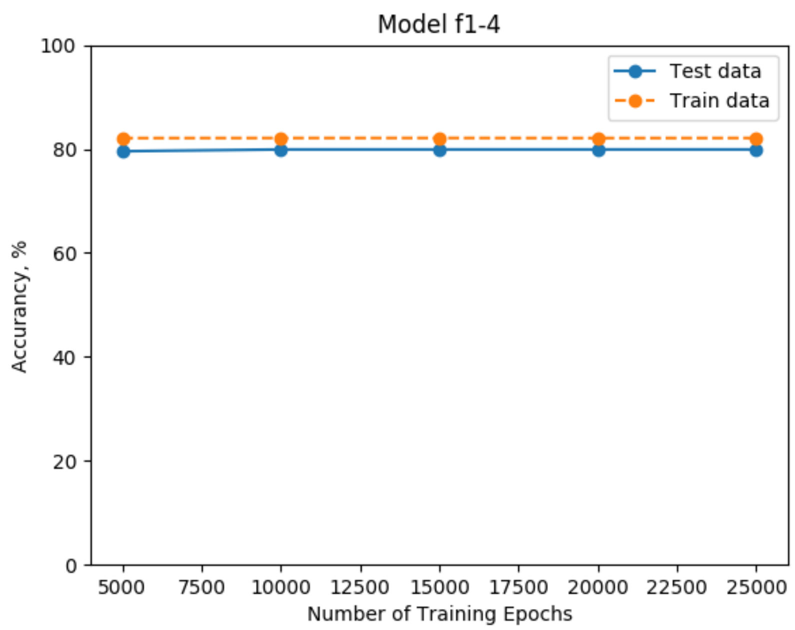

Рисунок 3.3.10 — График точности построенной однослойной нейронной сети модели f_{1-4} от количества итераций обучения

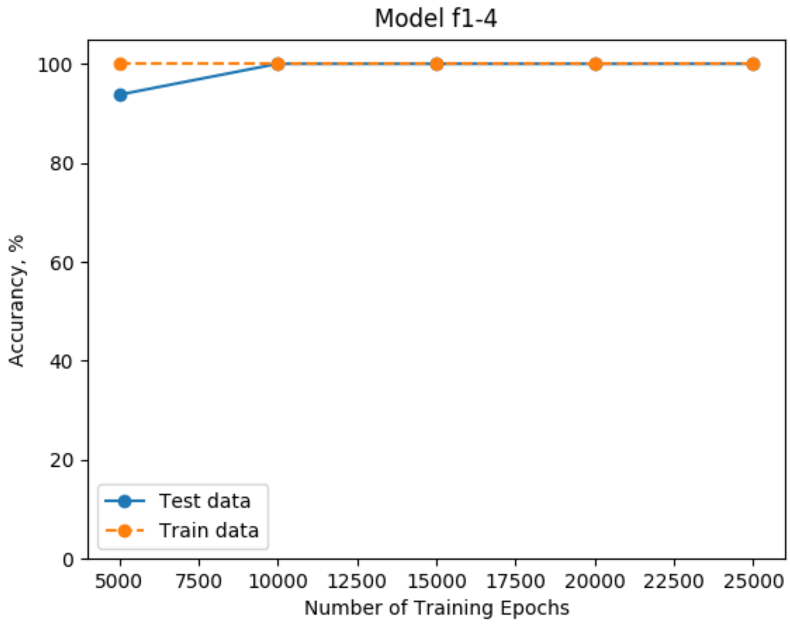

Рисунок 3.3.11 — График точности построенной нейронной сети с одним скрытым слоем модели f_{1-4} от количества итераций обучения

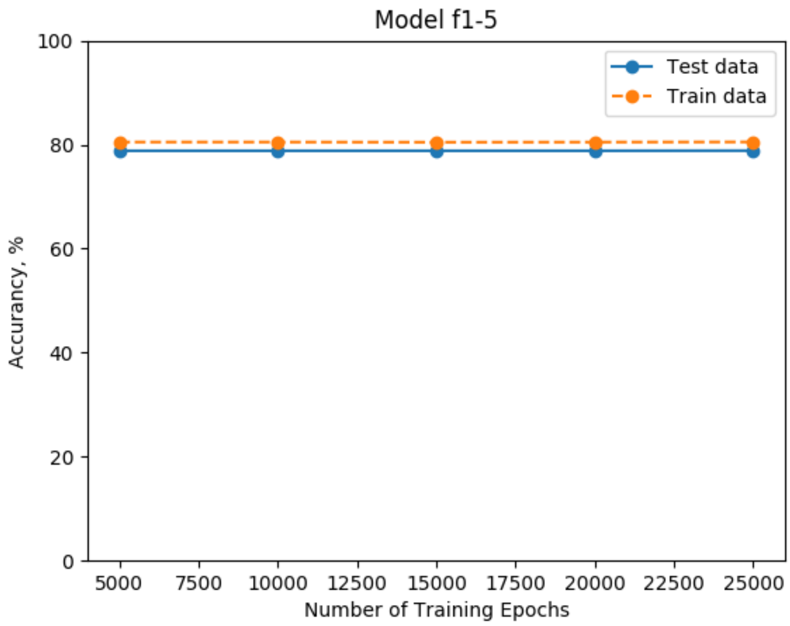

Рисунок 3.3.12 — График точности построенной однослойной нейронной сети модели f_{1-5} от количества итераций обучения

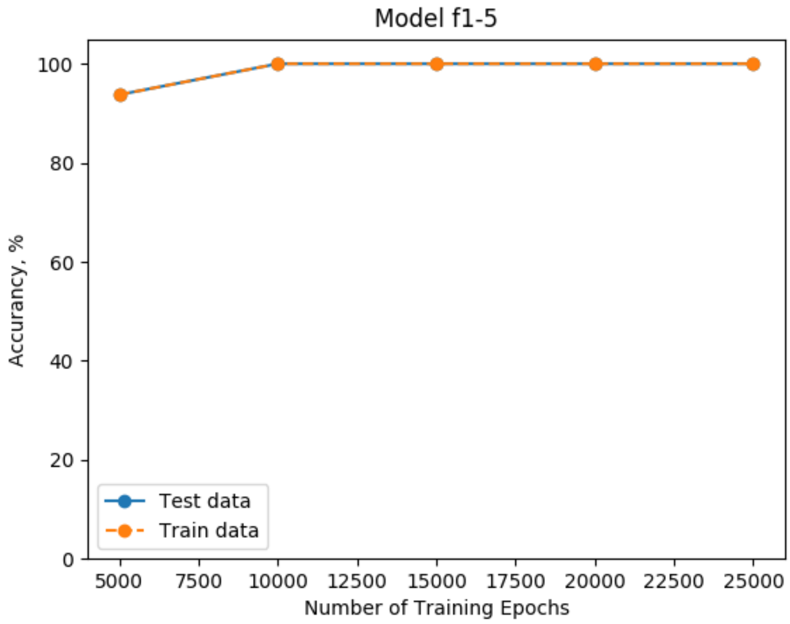

Рисунок 3.3.13 — График точности построенной нейронной сети с одним скрытым слоем модели f_{1-5} от количества итераций обучения

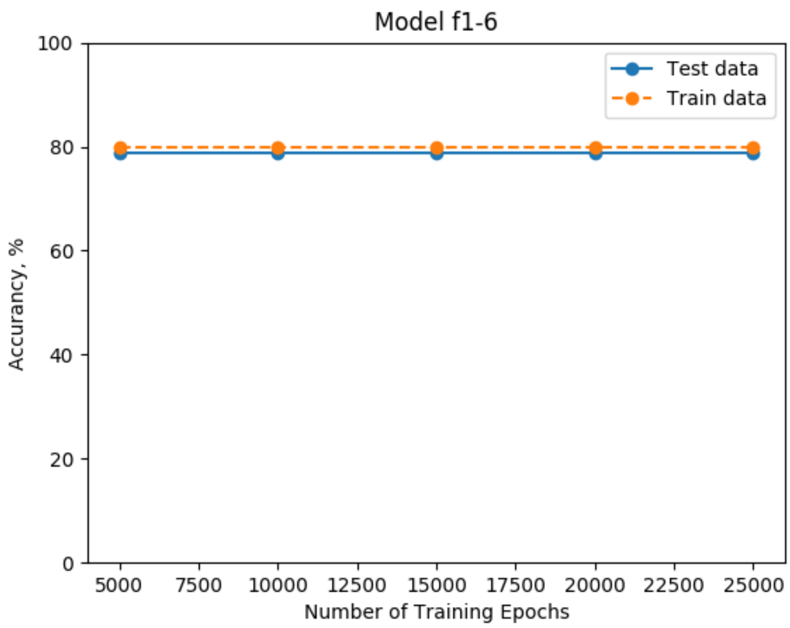

Рисунок 3.3.14 — График точности построенной однослойной нейронной сети модели f_{1-6} от количества итераций обучения

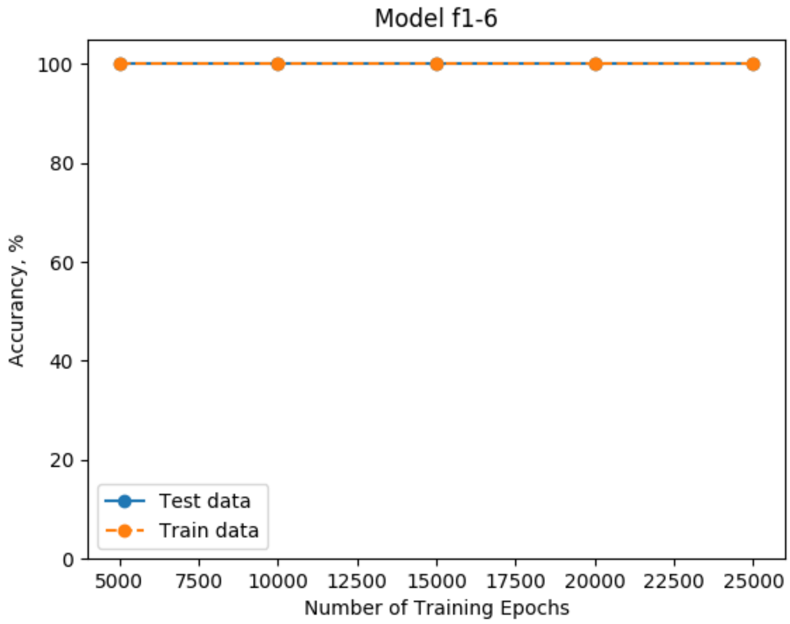

Рисунок 3.3.15 — График точности построенной нейронной сети с одним скрытым слоем модели f_{1-6} от количества итераций обучения

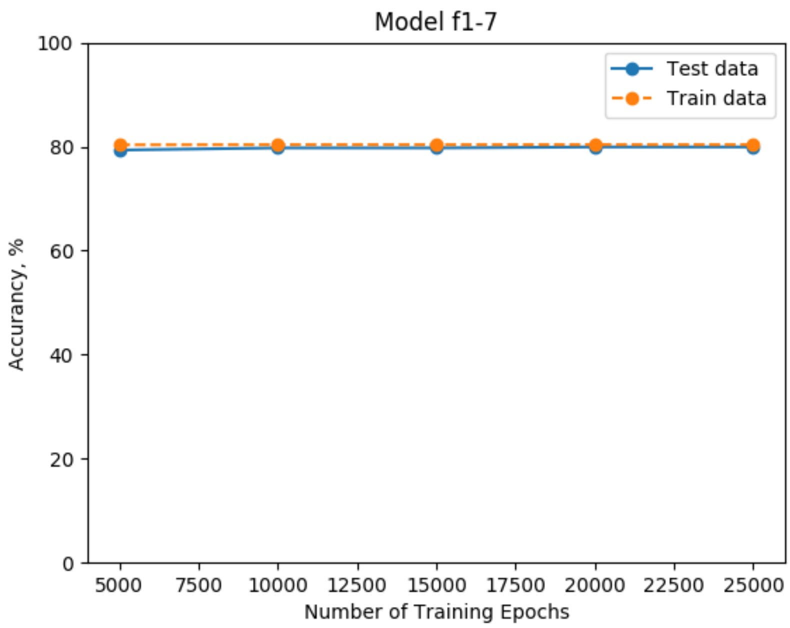

Рисунок 3.3.16 — График точности построенной однослойной нейронной сети модели f_{1-7} от количества итераций обучения

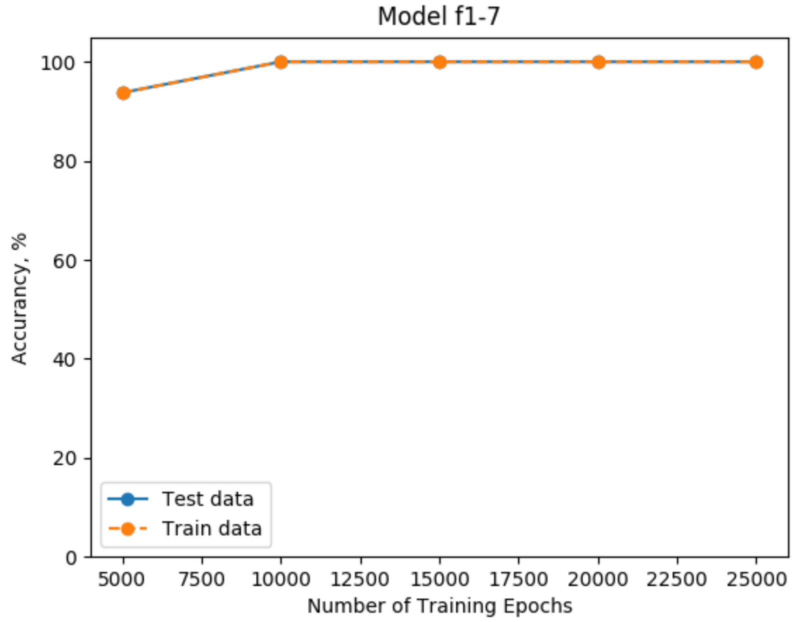

Рисунок 3.3.17 — График точности построенной нейронной сети с одним скрытым слоем модели f_{1-7} от количества итераций обучения

Модель	НС	N	NP	T_o	T_e	$\hat{f}$	l	Σ	Ψ
$f1-0$	NN	0	256	5120	512	12.3248	0.1264	5000	3.5308
$f1-0$	MNN	32	1024	5120	512	16.0000	0.0318	10000	11.2596
$f1-1$	NN	0	256	5120	512	12.6752	0.1258	10000	7.6633
$f1-1$	MNN	32	1024	5120	512	16.0000	0.0141	10000	11.7257
$f1-2$	NN	0	256	5120	512	13.1679	0.1258	20000	17.4358
$f1-2$	MNN	32	1024	5120	512	16.0000	0.0279	10000	13.2084
$f1-3$	NN	0	256	5120	512	13.1241	0.1230	20000	16.9254
$f1-3$	MNN	32	1024	5120	512	16.0000	0.0338	10000	13.3615
$f1-4$	NN	0	256	5120	512	12.7847	0.1287	10000	11.4368
$f1-4$	MNN	32	1024	5120	512	16.0000	0.0381	10000	14.1050
$f1-5$	NN	0	256	5120	512	12.6131	0.1168	25000	24.4405
$f1-5$	MNN	32	1024	5120	512	16.0000	0.0320	10000	14.7981
$f1-6$	NN	0	256	5120	512	12.5912	0.1312	5000	8.7702
$f1-6$	MNN	32	1024	5120	512	16.0000	0.0501	5000	10.5173
$f1-7$	NN	0	256	5120	512	12.7883	0.1165	20000	21.9568
$f1-7$	MNN	32	1024	5120	512	16.0000	0.0194	10000	16.5015

Таблица 3.1 — Сравнение модели f_1 при различных L

Из представленных выше графиков и таблицы 3.1, можно сделать вывод что параметр L не вносит существенных изменений в построение нейронной сети и точность аппроксимации. По этой причине, в дальнейшем исследовании (в рассмотрении следующих моделей) мы будем рассматривать только краевые случаи, а именно модели в которых $L \equiv 0$ и $L \equiv 8$.

3.3.2 Модели f_2

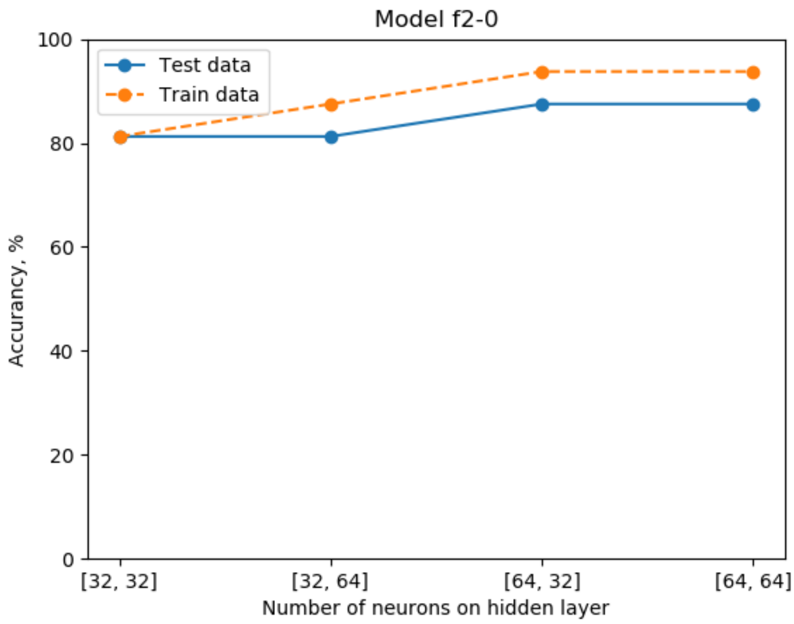

Рисунок 3.3.18 — График точности построенной многослойной нейронной сети модели f_{2-0} от архитектуры перцептрона

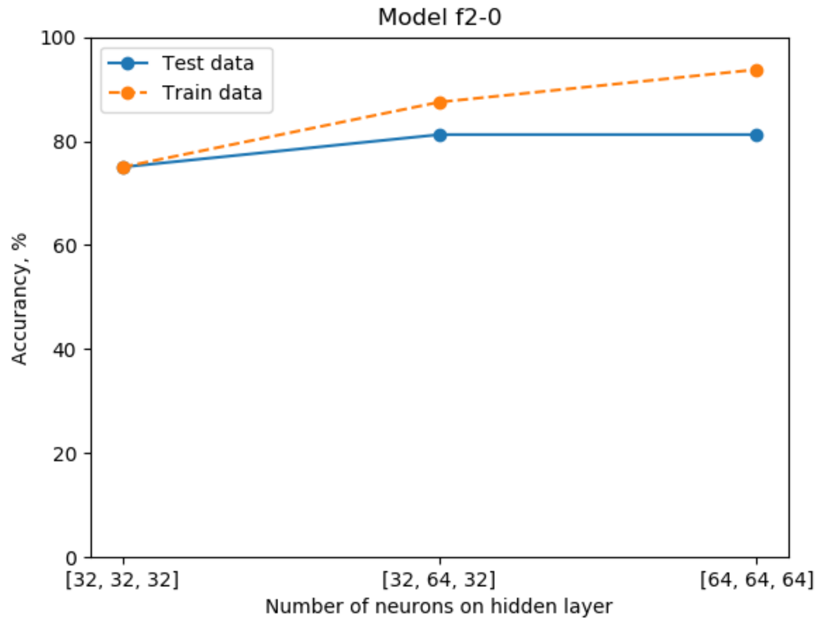

Рисунок 3.3.19 — График точности построенной многослойной нейронной сети модели f_{2-0} от архитектуры перцептрона

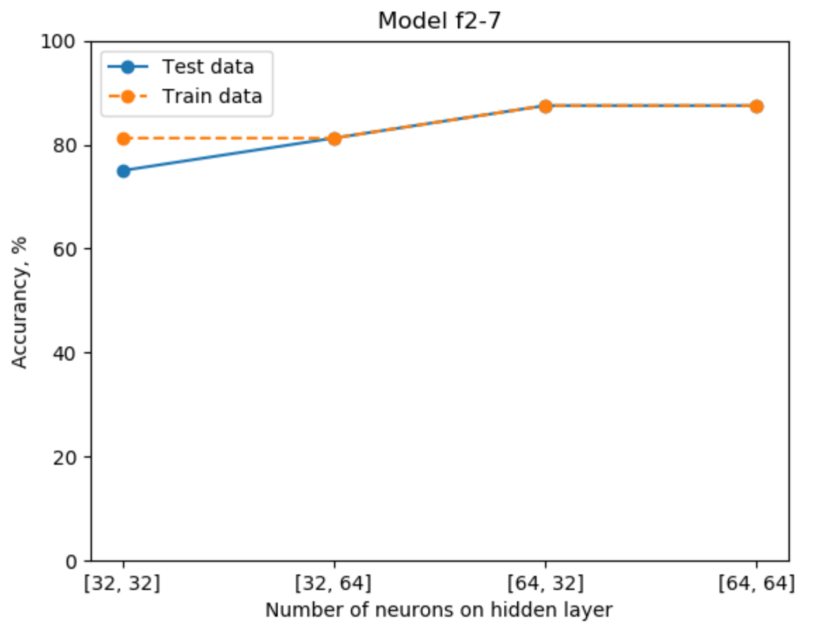

Рисунок 3.3.20 — График точности построенной многослойной нейронной сети модели f_{2-7} от архитектуры перцептрона

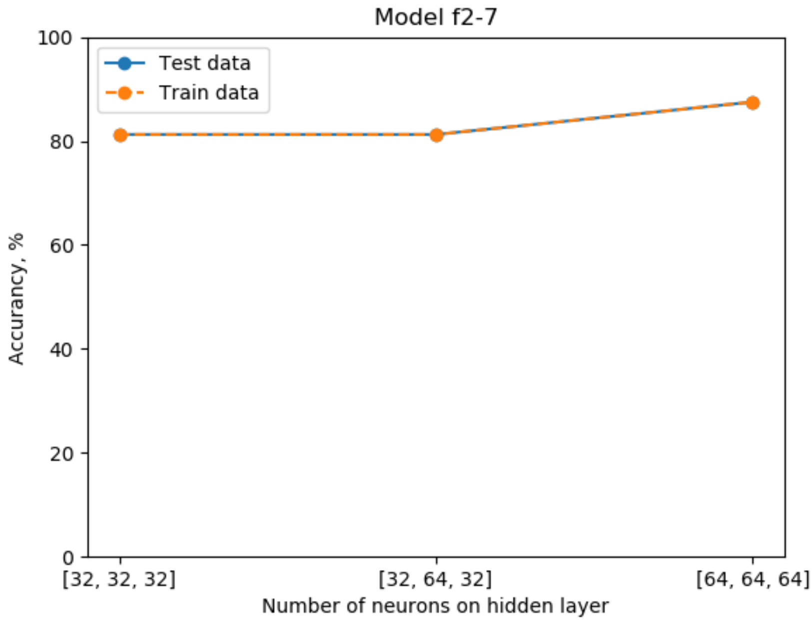

Рисунок 3.3.21 — График точности построенной многослойной нейронной сети модели f_{2-7} от архитектуры перцептрона

3.3.3 Модели f_3

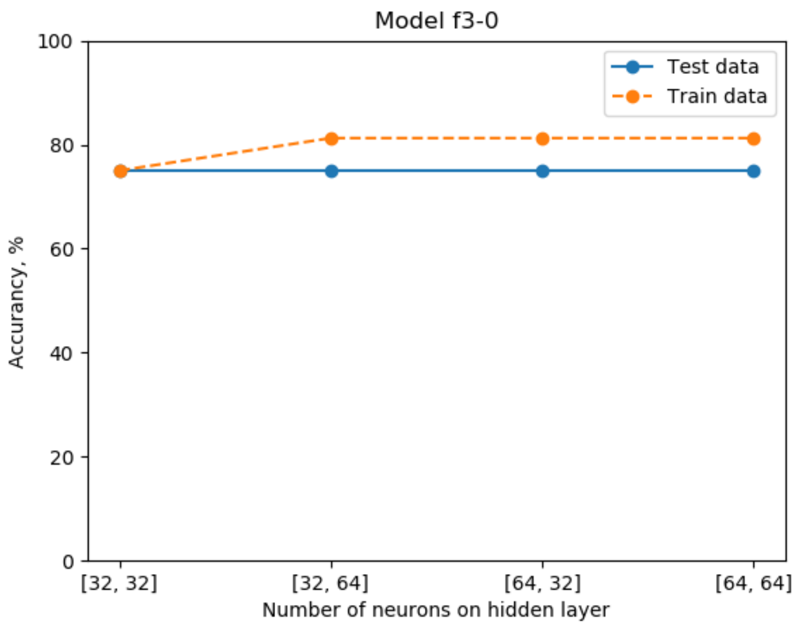

Рисунок 3.3.22 — График точности построенной многослойной нейронной сети модели f_{3-0} от количества итераций обучения

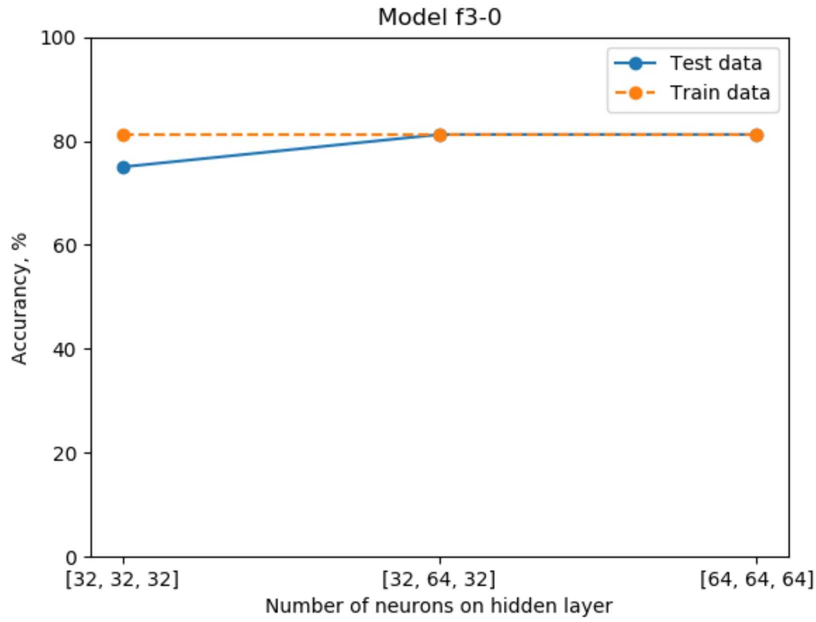

Рисунок 3.3.23 — График точности построенной многослойной нейронной сети модели f_{3-0} от архитектуры перцептрона

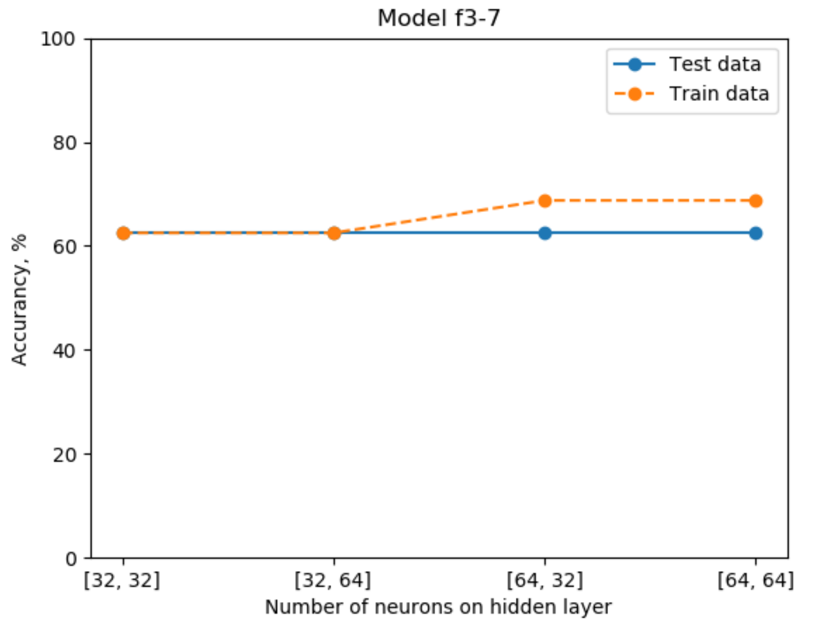

Рисунок 3.3.24 — График точности построенной многослойной нейронной сети модели f_{3-7} от архитектуры перцептрона

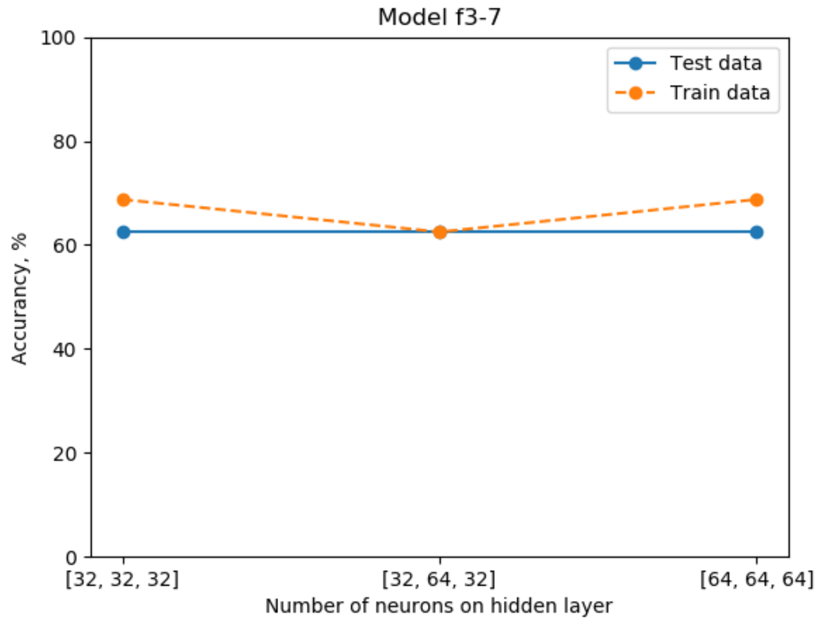

Рисунок 3.3.25 — График точности построенной многослойной нейронной сети модели f_{3-7} от архитектуры перцептрона

3.3.4 Модели f_4

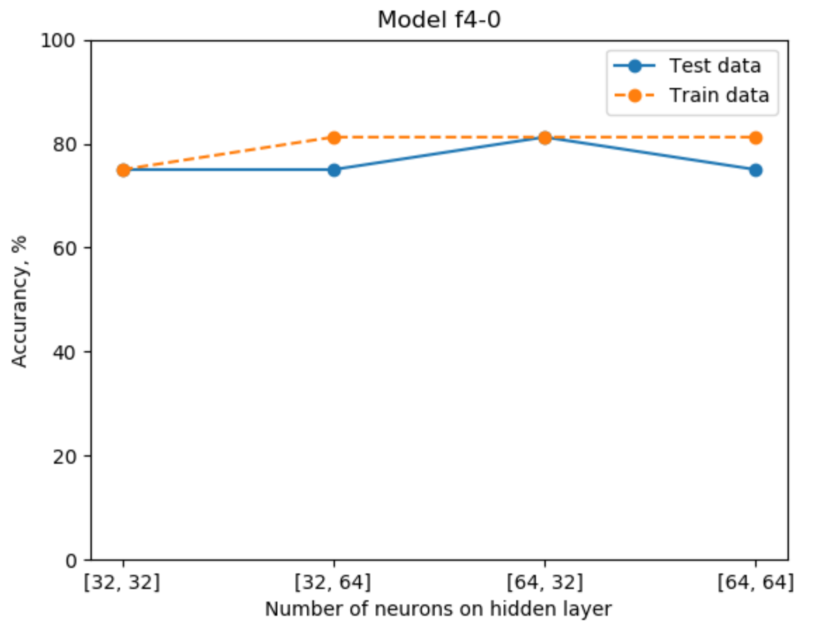

Рисунок 3.3.26 — График точности построенной многослойной нейронной сети модели f_{4-0} от архитектуры перцептрона

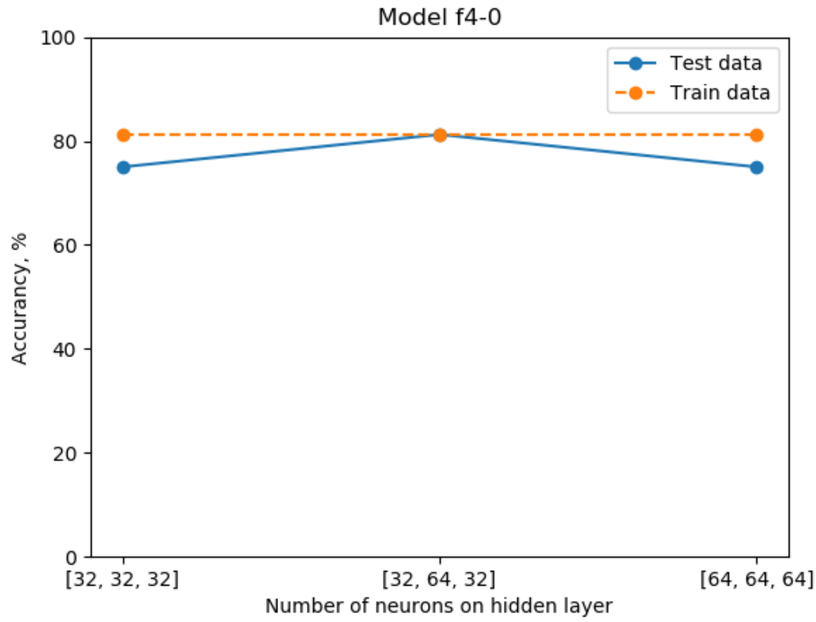

Рисунок 3.3.27 — График точности построенной многослойной нейронной сети модели f_{4-0} от архитектуры перцептрона

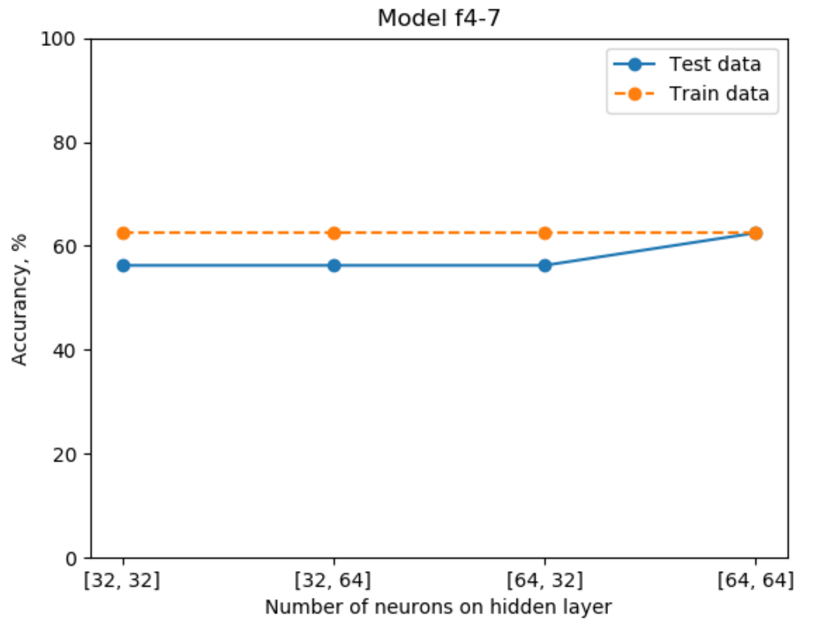

Рисунок 3.3.28 — График точности построенной многослойной нейронной сети модели f_{4-7} от архитектуры перцептрона

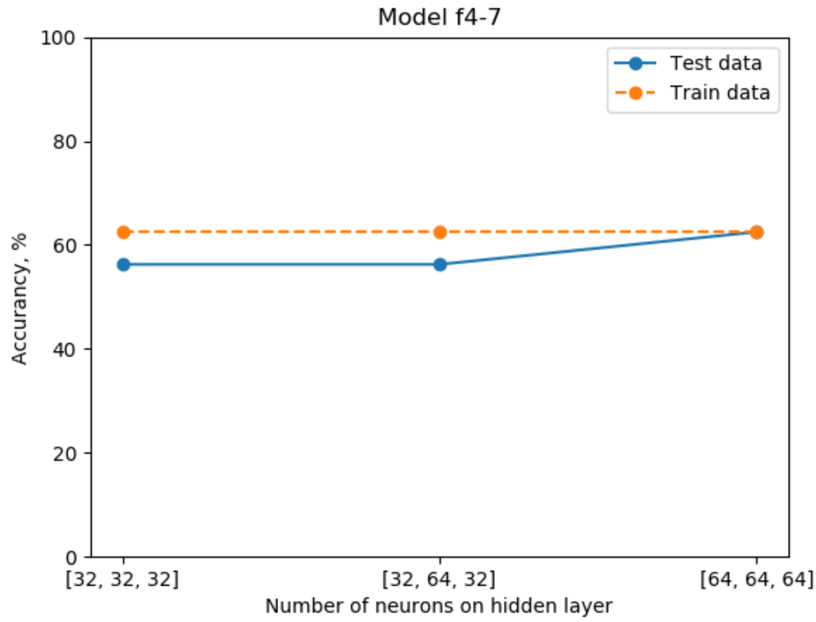

Рисунок 3.3.29 — График точности построенной многослойной нейронной сети модели f_{4-7} от архитектуры перцептрона

3.3.5 Модели f_5

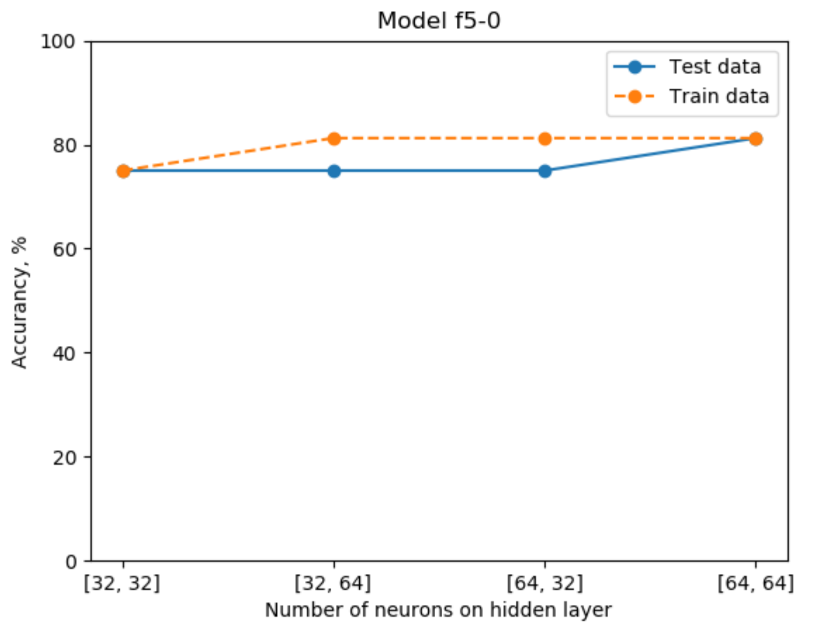

Рисунок 3.3.30 — График точности построенной многослойной нейронной сети модели f_{5-0} от архитектуры перцептрона

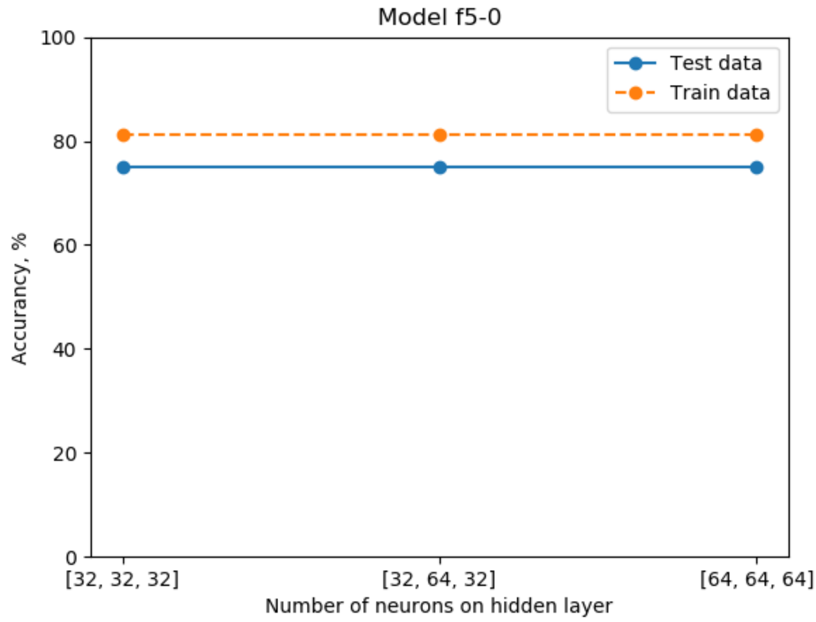

Рисунок 3.3.31 — График точности построенной многослойной нейронной сети модели f_{5-0} от архитектуры перцептрона

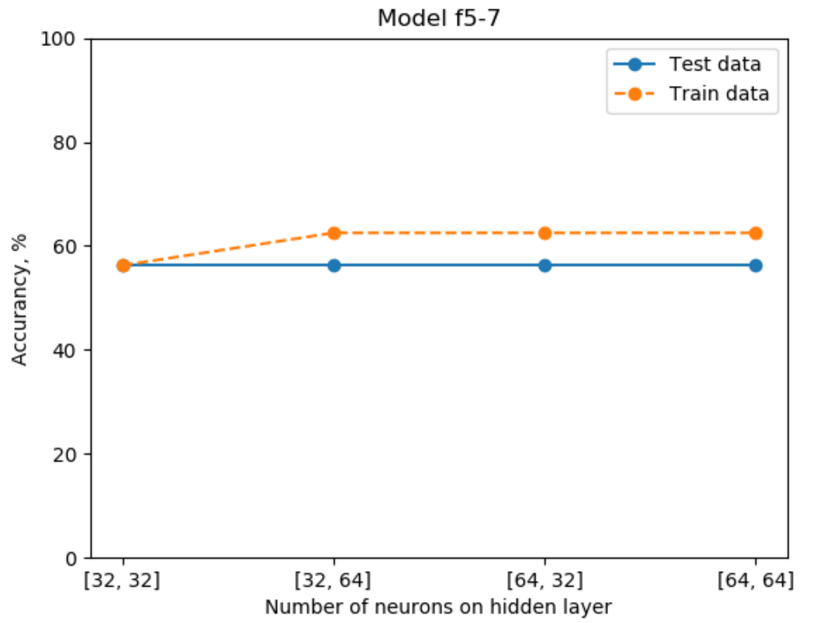

Рисунок 3.3.32 — График точности построенной многослойной нейронной сети модели f_{5-7} от архитектуры перцептрона

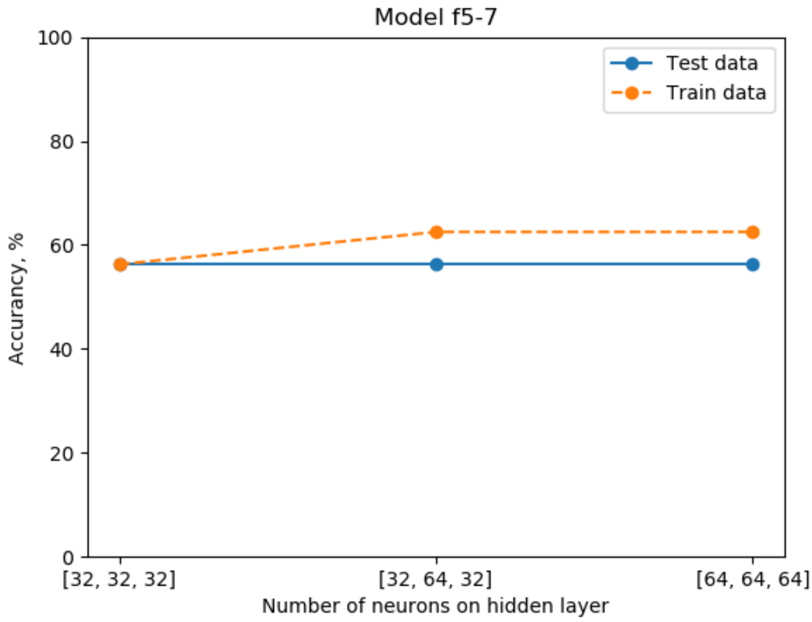

Рисунок 3.3.33 — График точности построенной многослойной нейронной сети модели f_{5-7} от архитектуры перцептрона

3.3.6 Модели f_6

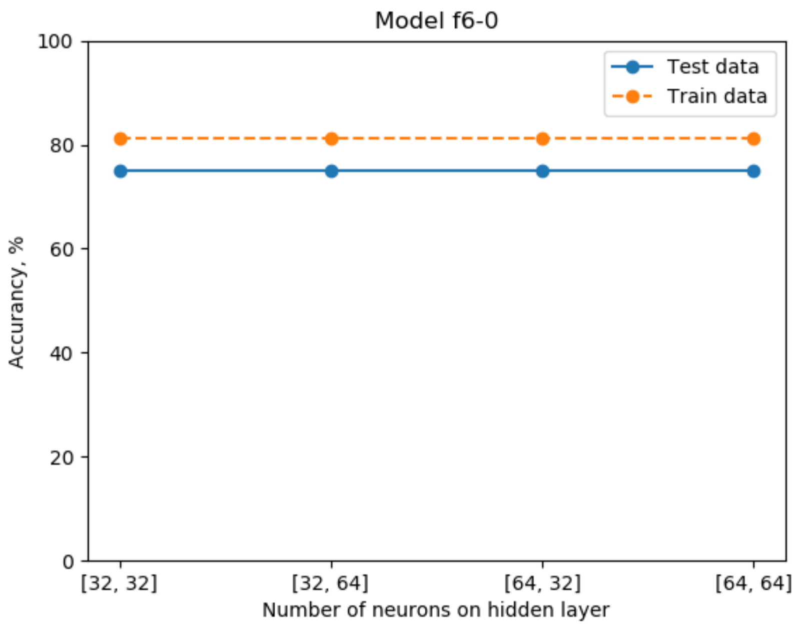

Рисунок 3.3.34 — График точности построенной многослойной нейронной сети модели f_{6-0} от архитектуры перцептрона

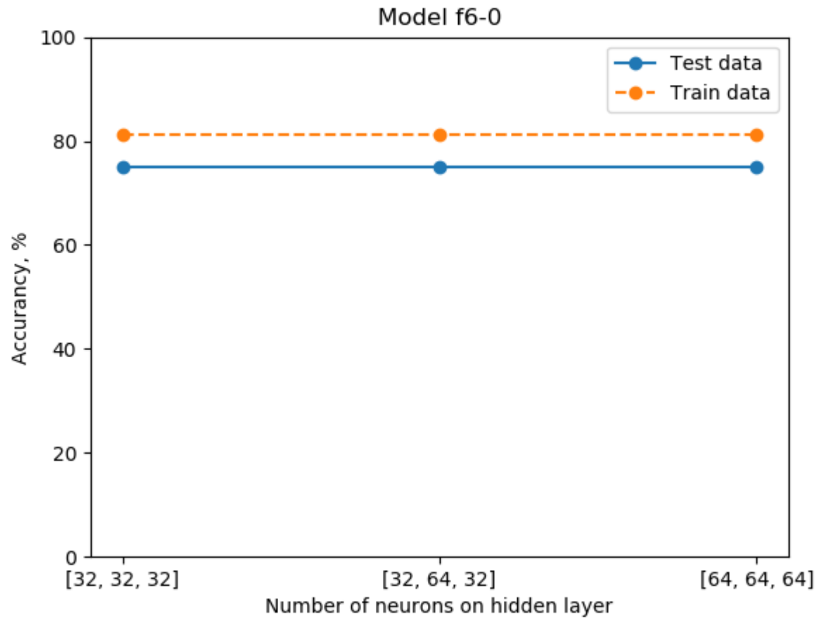

Рисунок 3.3.35 — График точности построенной многослойной нейронной сети модели f_{6-0} от архитектуры перцептрона

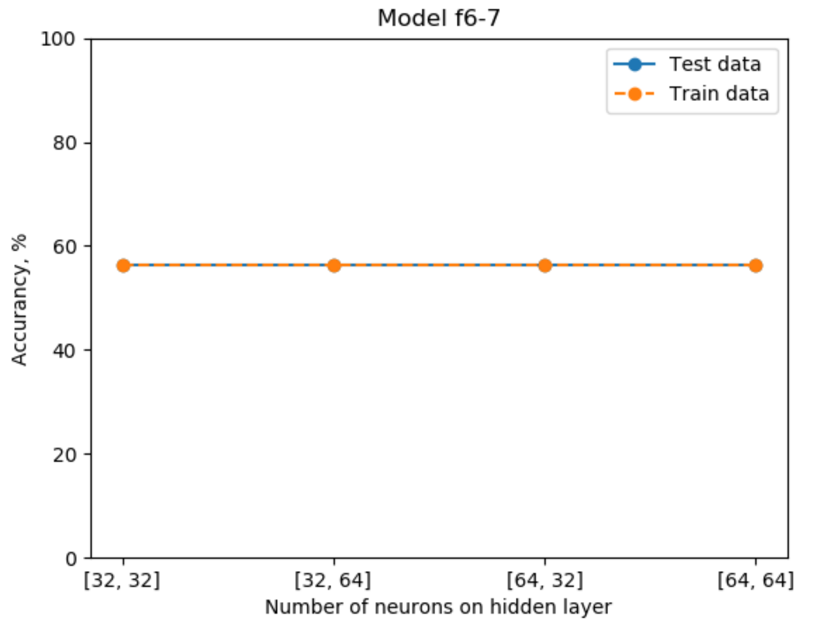

Рисунок 3.3.36 — График точности построенной многослойной нейронной сети модели f_{6-7} от архитектуры перцептрона

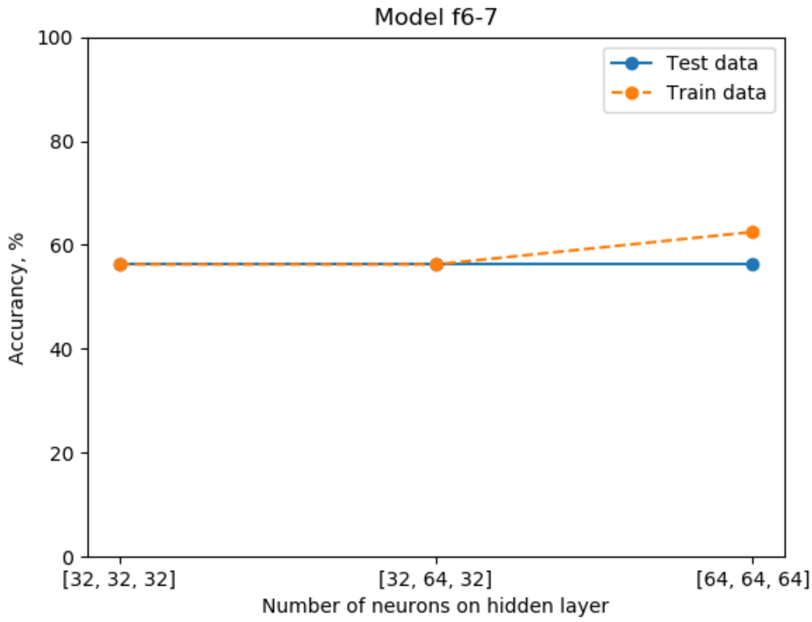

Рисунок 3.3.37 — График точности построенной многослойной нейронной сети модели f_{6-7} от архитектуры перцептрона

3.3.7 Модели f_7

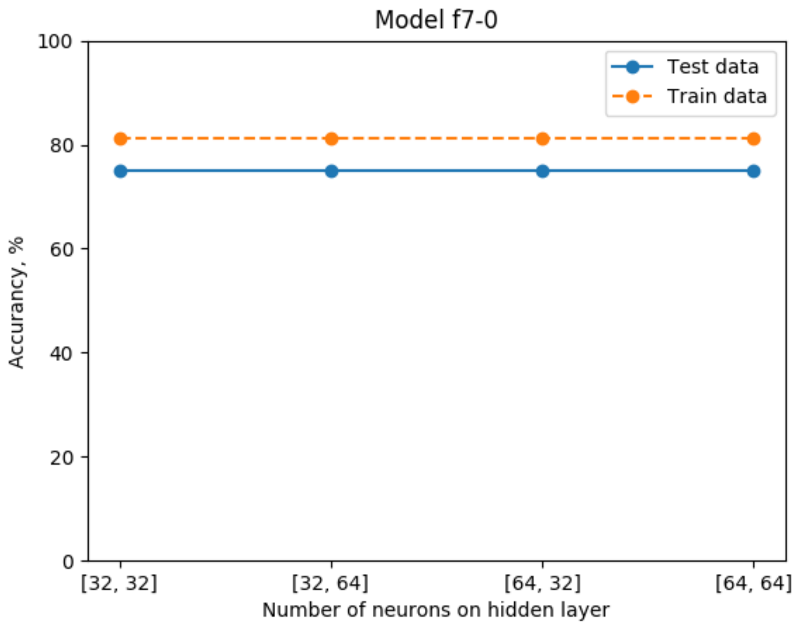

Рисунок 3.3.38 — График точности построенной многослойной нейронной сети модели f_{7-0} от архитектуры перцептрона

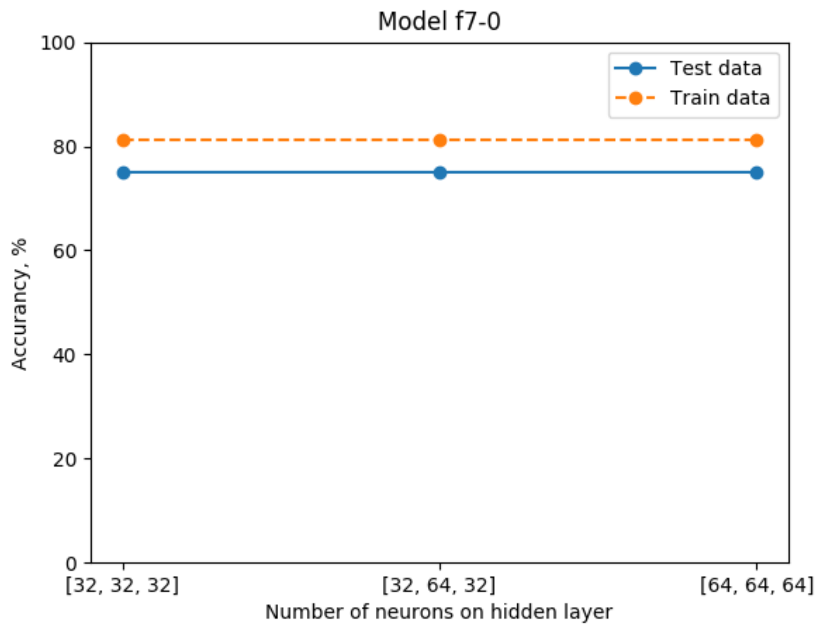

Рисунок 3.3.39 — График точности построенной многослойной нейронной сети модели f_{7-0} от архитектуры перцептрона

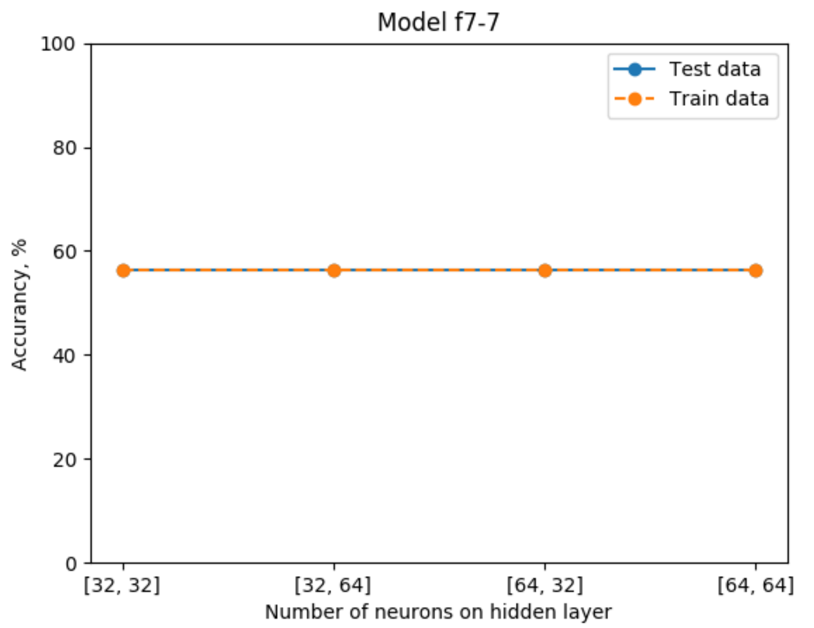

Рисунок 3.3.40 — График точности построенной многослойной нейронной сети модели f_{7-7} от архитектуры перцептрона

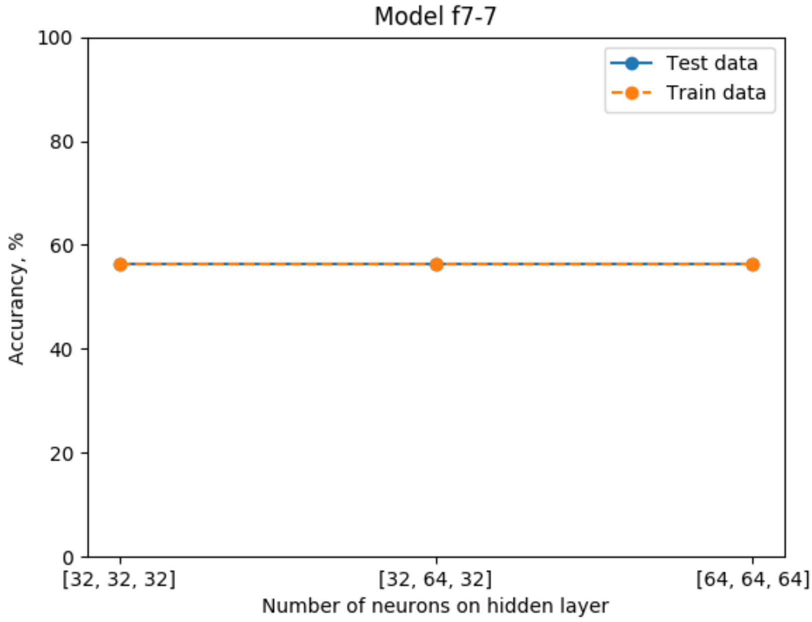

Рисунок 3.3.41 — График точности построенной многослойной нейронной сети модели f_{7-7} от архитектуры перцептрона

3.3.8 Модели f_8

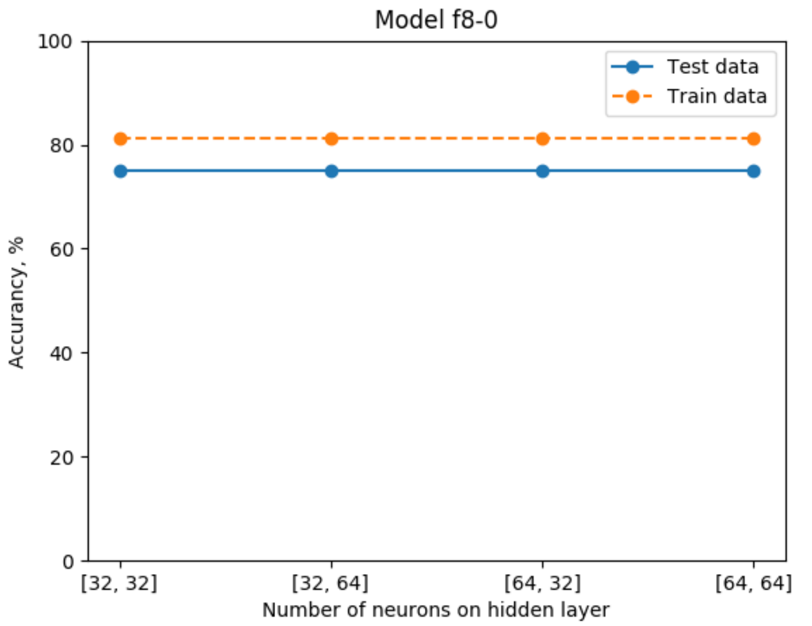

Рисунок 3.3.42 — График точности построенной многослойной нейронной сети модели f_{8-0} от архитектуры перцептрона

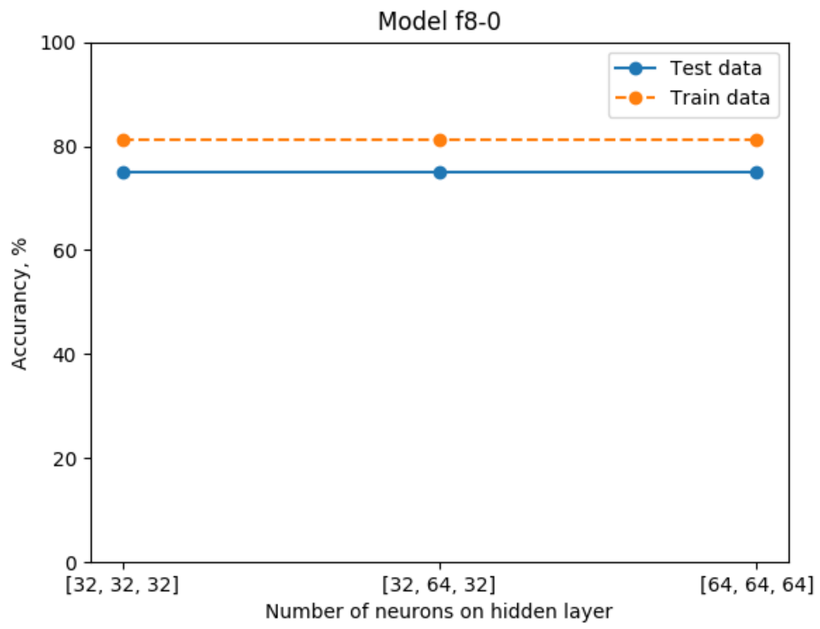

Рисунок 3.3.43 — График точности построенной многослойной нейронной сети модели f_{8-0} от архитектуры перцептрона

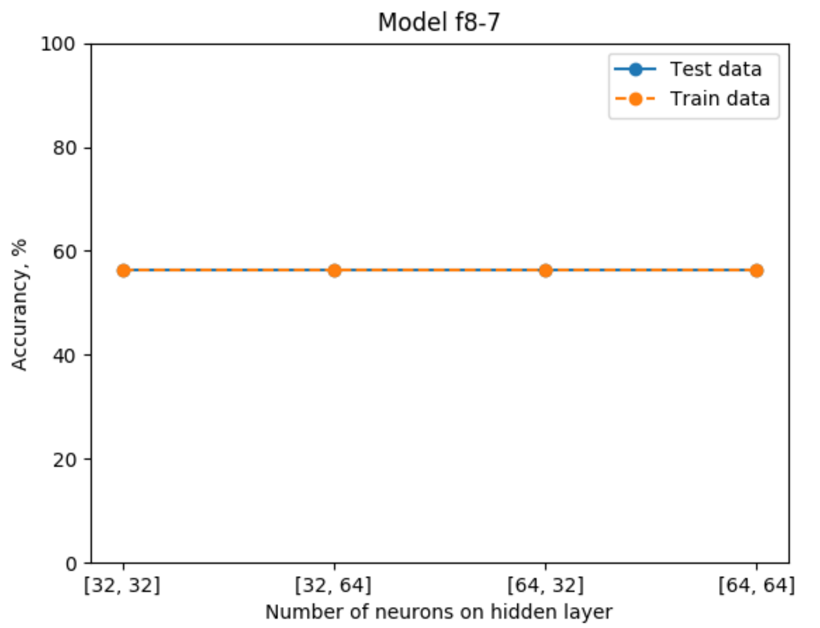

Рисунок 3.3.44 — График точности построенной многослойной нейронной сети модели f_{8-7} от архитектуры перцептрона

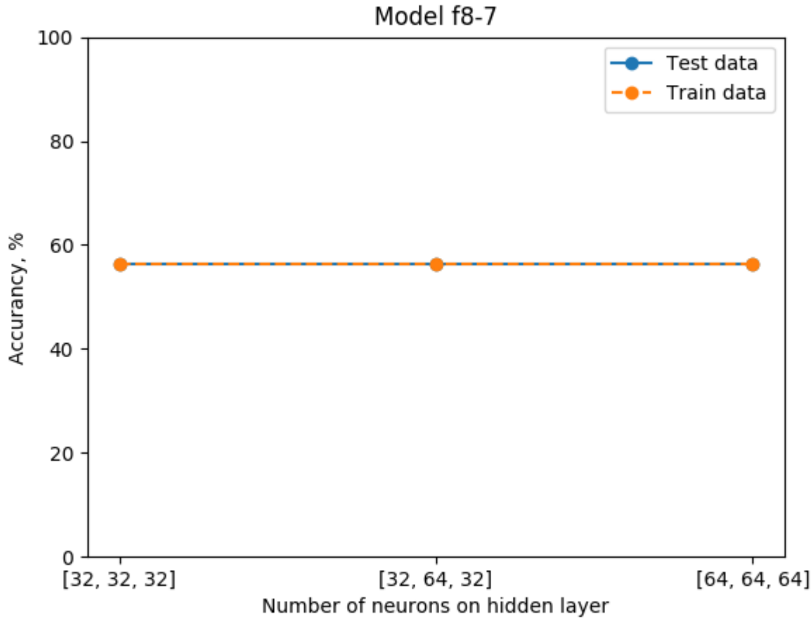

Рисунок 3.3.45 — График точности построенной многослойной нейронной сети модели f_{8-} от архитектуры перцептрона

3.3.9 Обобщение полученных результатов

Основываясь на проведенных экспериментах построим таблицу в которой сравним модели и построенный нейронные сети с минимальным количеством параметров. В таблице представим следующие параметры сравнения: размер обучающей выборки, размер тестовой выборки, параметры построенной нейронной сети, точность нейронной сети, потери при обучении, время затраченное на обучение.

Введем следующие обозначения:

- N - количество нейронов на скрытом слое,
- NP - число обучаемых параметров сети,
- T_o - размер обучающей выборки,
- T_e - размер тестовой (экзаменационной) выборки,
- $\hat{f}$ - точность построенной нейронной сети,
- l - потери при обучении,
- Ψ - время обучения нейронной сети в секундах,
- Σ - количество эпох обучения.

Модель	N	NP	T_o	T_e	$\hat{f}$	l	Σ	Ψ
$f1-0$	[64, 64]	6144	10240	1024	16.0000	0.0024	15000	45.9182
$f1-0$	[64, 64, 64]	10240	10240	1024	16.0000	0.0051	15000	70.5227
$f1-7$	[64, 64]	6144	10240	1024	16.0000	0.0053	15000	47.3304
$f1-7$	[64, 64, 64]	10240	10240	1024	16.0000	0.0191	15000	67.8144
$f2-0$	[64, 64]	6144	10240	1024	14.0000	0.1135	15000	287.8751
$f2-0$	[64, 64, 64]	10240	10240	1024	13.0000	0.1594	15000	415.4322
$f2-7$	[64, 64]	6144	10240	1024	14.0000	0.1007	15000	295.2384
$f2-7$	[64, 64, 64]	10240	10240	1024	14.0000	0.1126	15000	410.2425
$f3-0$	[64, 64]	6144	10240	1024	12.0000	0.1446	15000	294.4967
$f3-0$	[64, 64, 64]	10240	10240	1024	13.0000	0.1290	15000	416.2675
$f3-7$	[64, 64]	6144	10240	1024	10.0000	0.2218	15000	296.4944
$f3-7$	[64, 64, 64]	10240	10240	1024	10.0000	0.2152	15000	424.1930
$f4-0$	[64, 64]	6144	10240	1024	12.0000	0.1292	15000	306.2740
$f4-0$	[64, 64, 64]	10240	10240	1024	12.0000	0.1259	15000	434.2595
$f4-7$	[64, 64]	6144	10240	1024	10.0000	0.2403	15000	312.7606
$f4-7$	[64, 64, 64]	10240	10240	1024	10.0000	0.2371	15000	447.1662
$f5-0$	[64, 64]	6144	10240	1024	13.0000	0.1258	15000	319.0132
$f5-0$	[64, 64, 64]	10240	10240	1024	12.0000	0.1264	15000	457.8168
$f5-7$	[64, 64]	6144	10240	1024	9.0000	0.2458	15000	321.5454
$f5-7$	[64, 64, 64]	10240	10240	1024	9.0000	0.2462	15000	465.3593
$f6-0$	[64, 64]	6144	10240	1024	12.0000	0.1261	15000	321.7591
$f6-0$	[64, 64, 64]	10240	10240	1024	12.0000	0.1261	15000	460.5911
$f6-7$	[64, 64]	6144	10240	1024	9.0000	0.2491	15000	331.5827
$f6-7$	[64, 64, 64]	10240	10240	1024	9.0000	0.2499	15000	475.0901
$f7-0$	[64, 64]	6144	10240	1024	12.0000	0.1259	15000	273.3749
$f7-0$	[64, 64, 64]	10240	10240	1024	12.0000	0.1265	15000	421.1694
$f7-7$	[64, 64]	6144	10240	1024	9.0000	0.2524	15000	258.6057
$f7-7$	[32, 64, 32]	5120	10240	1024	9.0000	0.2515	15000	247.9778
$f7-7$	[64, 64, 64]	10240	10240	1024	9.0000	0.2527	15000	430.1847
$f8-0$	[64, 64]	6144	10240	1024	12.0000	0.1262	15000	226.1201
$f8-0$	[64, 64, 64]	10240	10240	1024	12.0000	0.1265	15000	332.4553
$f8-7$	[64, 64]	6144	10240	1024	9.0000	0.2523	15000	225.5158
$f8-7$	[64, 64, 64]	10240	10240	1024	9.0000	0.2544	15000	329.9783

Таблица 3.2 — Сравнение построенных нейронных сетей

Из представленных выше результатов (графики и таблица 3.2), можно сделать вывод, нейронные сети прямого распространения с многослойным перцептроном, позволяют аппроксимировать криптографические преобразования. Из этого можно сделать вывод, что такой метод криптоанализа можно использовать для оценки стойкости криптосистем.

Заключение

Таким образом, в рамках данной работы было выполнено следующее:

1. Проведен анализ литературы по теме нейронная криптография. Также проведен анализ и обзор искусственных нейронных сетей и их применение в криптографии.

2. Построены математические модели криптографических примитивов преобразования Фейстеля.

3. Построены математические модели криптографического преобразования Фейстеля.

4. Разработан программный комплекс для моделирования математических моделей.

5. Построены нейронные сети для аппроксимации математических моделей.

6. Проведены компьютерные эксперименты.

7. Проведен анализ полученных результатов.

Литература

1. Харин Ю.С., Берник В.И., Матвеев Г.В., Агиевич С.В. Математические и компьютерные основы криптологии. – 2003. – Минск.

2. Neural networks in cryptography [Электрон. ресурс]. – 2015. – $http://cryptowiki.net/index.php?title=Neural\_networks\_in\_cryptography$.

3. Pattanayak S., Ludwig S.A. Encryption based on Neural Cryptography. – 2017.

4. Clark M., Blank D. A Neural-Network Based Cryptographic System. – 1998

5. Kinzel F., Kanter I. Neural Cryptography. — 2002.

6. Li S. Analyses and New Designs of Digital Chaotic Ciphers. – 2003. – China.

7. Rao K., Krishna M., Babu D. Cryptanalysis of a Feistel Type Block Cipher by Feed Forward Neural Network Using Right Sigmoidal Signal. – 2009.

8. Mohammed M. Alani.Neuro-Cryptanalysis of DES and Triple-DES. – 2012.

9. Харин Ю.С., Агиевич С.В. Компьютерный практикум по математическим методам защиты информации. – 2001. – Минск.

10. Minjie K., Smaragdis P. Bitwise Neural Network. – 2016.

11. Lin X., Cong Z., Wei P. Towards Accurate Binary Convolutional NeuralNetwork. – 20017.

12. Implementing the XOR Gate using Backpropagation in Neural Networks [Электрон. ресурс]. — 2019. — $https://towardsdatascience.com/implementing-the-xor-gate-using-$

13. Sannai A., Takai Y., Cordonnier M. Universal approximations of permutation in-variant/equivariant functions by deep neural networks. – 2019.

14. Knudsen L.R. Block Ciphers – 1998.

15. Godhavari T., Alamelu N.R., Soundararajan R. Cryptography Using Neural Network. – 2005.

16. Kanter K.W., Kanter E. Secure exchange of information by synchronization of neural networks. – 2002.

17. Klimov A.B., Mityagin A., Shamir A.: Analysis of Neural Cryptography. – 2002.

18. Li L., Lin L., Hwang M. A remote password authentication scheme for multiserver ar- chitecture using neural networks. – 2001.

19. Dourlens, S. Neuro-Cryptography. – 1995. – France.

20. Li C., Li S., Zhang D., Chen G. Chosen-Plaintext Cryptanalysis of a Clipped-Neural- Network-Based Chaotic Cipher. – 2005.

21. Alani M.M. Neurocryptanalysis of DES. – 2012.

3.4 Исходный код реализации

Для построения нейронных сетей использовалась библиотека Tensorflow.

Программное обеспечение, используемое для реализации нейронных сетей: Python 3.7, Tensorflow (Tensorflow Core r1.15).

Программное обеспечение, используемое для реализации генератора модельных данных: C11, Make.

Процессор: Intel(R) Core(TM) i7-8750H 2.20GHz. Память: 16240MB.

Репозиторий с реализацией нейронных сетей и генератора модельных данных: https://github.com/MD-Levitan/neuro_crypt.

Выдержки из исходного кода:

```
/**
 * @brief Iterate from 0 to @size as input to @gen function
 *
 * @param ctx          Crypto context
 * @param params       Params for generator, can be NULL
 * @param size         size of output sequence <= UINT64_MAX
 * @param filename_x   filename with input
 * @param filename_y   filename with output
 * @param gen          generator of output sequence
 */
void iterate_generator(crypto_tfm *ctx, generator_params_t *params,
                       uint64_t size, const char *filename_x, const char *
                           filename_y,
                       generator_model gen);

/**
 * @brief Iterate from 0 to @size in two 32-bite blocks as input to @gen
     function
 *
 * @param ctx          Crypto context
 * @param params       Params for generator, can be NULL
 * @param size         size of output sequence <= UINT32_MAX
 * @param filename_x   filename with input
 * @param filename_y   filename with output
 * @param gen          generator of output sequence
 */
void iterate_parallel_generator(crypto_tfm *ctx, generator_params_t *params,
                                uint64_t size, const char *filename_x, const
                                    char *filename_y,
                                generator_model gen);

/**
 * @brief Iterate from 0 to @size in two <S>-bite blocks as input to @gen
     function.
 *        <S> from params. Support <S> = {4, 8, 16, 32}. Size of output is @size
     ^2.
 *
 * @param ctx          Crypto context
 * @param params       Params for generator, can be NULL
 * @param size         size of output sequence <= UINT32_MAX
 * @param filename_x   filename with input
 * @param filename_y   filename with output
 * @param gen          generator of output sequence
```

```
 */
void iterate_split_generator(crypto_tfm *ctx, generator_params_t *params,
                             uint64_t size, const char *filename_x, const char *filename_y,
                             generator_model gen);

/**
 * @brief Generate random @size values as input to @gen function
 *
 * @param ctx           Crypto context
 * @param params        Params for generator, can be NULL
 * @param size          size of output sequence <= UINT64_MAX
 * @param filename_x    filename with input
 * @param filename_y    filename with output
 * @param gen           generator of output sequence
 */
void random_generator(crypto_tfm *ctx, generator_params_t *params,
                      uint64_t size, const char *filename_x, const char *filename_y,
                      generator_model gen);

/**
 * @brief Iterate values from 0 to @size in random order as input to @gen function
 *
 * @param ctx           Crypto context
 * @param params        Params for generator, can be NULL
 * @param size          size of output sequence <= UINT64_MAX
 * @param filename_x    filename with input
 * @param filename_y    filename with output
 * @param gen           generator of output sequence
 */
void random_iterate_generator(crypto_tfm *ctx, generator_params_t *params,
                              uint64_t size, const char *filename_x, const char *filename_y,
                              generator_model gen);

/**
 * @brief Generator for 1-round of GOST
 *
 * @param ctx           Crypto context
 * @param out_file_x    File context to write input
 * @param out_file_y    File context to write output
 * @param in            Input value
 */
void round_generator(crypto_tfm *ctx, FILE *out_file_x, FILE *out_file_y, uint64_t in);

/**
 * @brief Generator for 2-round of GOST
 *
 * @param ctx           Crypto context
 * @param out_file_x    File context to write input
 * @param out_file_y    File context to write output
 * @param in            Input value
 */
void n_round_generator(crypto_tfm *ctx, FILE *out_file_x, FILE *out_file_y, uint64_t in);

/**
```

```
 * @brief Generator for G0 model. Input - 8 bits, output - 4 bits.
 *
 * @param ctx           Crypto context
 * @param out_file_x    File context to write input
 * @param out_file_y    File context to write output
 * @param in            Input value
 */
void primitive_g0_generator(crypto_tfm *ctx, FILE *out_file_x, FILE *out_file_y,
     uint64_t in);

/**
 * @brief Generator for G1 model. Input - 8 bits, output - 4 bits.
 *
 * @param ctx           Crypto context
 * @param out_file_x    File context to write input
 * @param out_file_y    File context to write output
 * @param in            Input value
 */
void primitive_g1_generator(crypto_tfm *ctx, FILE *out_file_x, FILE *out_file_y,
     uint64_t in);

/**
 * @brief Generator for G2 model. Input - 8 bits, output - 4 bits.
 *
 * @param ctx           Crypto context
 * @param out_file_x    File context to write input
 * @param out_file_y    File context to write output
 * @param in            Input value
 */
void primitive_g2_generator(crypto_tfm *ctx, FILE *out_file_x, FILE *out_file_y,
     uint64_t in);

/**
 * @brief Generator for G5 model. Input - 4 bits, output - 4 bits.
 *
 * @param ctx           Crypto context
 * @param out_file_x    File context to write input
 * @param out_file_y    File context to write output
 * @param in            Input value
 */
void primitive_g5_generator(crypto_tfm *ctx, FILE *out_file_x, FILE *out_file_y,
     uint64_t in);


/**
 * @brief Generator for G3 model. Input - 64 bits, output - 32 bits.
 *
 * @param ctx           Crypto context
 * @param out_file_x    File context to write input
 * @param out_file_y    File context to write output
 * @param in            Input value
 */
void primitive_g3_generator(crypto_tfm *ctx, FILE *out_file_x, FILE *out_file_y,
     uint64_t in);

/**
 * @brief Generator for G4 model.
 * Param in ctx shows (@input) number of bits in input.
 * Support next @input: 4 bits -> 4 bits
 *                      8 bits -> 8 bits
```

```
 *                      16 bits -> 16 bits
 *                      32 bits -> 32 bits
 *
 * @param ctx           Crypto context
 * @param out_file_x    File context to write input
 * @param out_file_y    File context to write output
 * @param in            Input value
 */
void primitive_g4_generator(crypto_tfm *ctx, FILE *out_file_x, FILE *out_file_y,
    uint64_t in);
```

generator.h

```

losses_default = {
    "sigmoid_cross_entropy": tf.compat.v1.losses.sigmoid_cross_entropy,
    "binary_cross_entropy": tf.keras.losses.binary_crossentropy,
    "square_difference": lambda y, x: tf.reduce_mean(tf.square(x - y)),
}


def init_one_layer_network(input_data: np.array, output_data: np.array, n_input:
    int, n_classes: int,
                           training_epochs: int = 100, display_step: int = 10,
                               optimizer=tf.compat.v1.train.AdamOptimizer,
                           loss_fun: str = "square_difference", verbose: bool =
                               True):
    """
    Create one-layer neural network.
    :param input_data: np.array with input values
    :param output_data: np.array with output values
    :param n_input: neuron's number on first layer
    :param n_classes: number of output classes
    :param training_epochs: number of training epochs
    :param display_step:
    :param optimizer:
    :param loss_fun:
    :param verbose:
    :return: Accuracy, Loss, Predict Array, Real Array
    """
    timer = time.time()
    x = tf.compat.v1.placeholder(tf.float32, [None, n_input])
    y = tf.compat.v1.placeholder(tf.float32, [None, n_classes])

    out = tf.layers.dense(x, n_classes, activation=tf.nn.sigmoid)

    hamming_distance = tf.math.count_nonzero(tf.round(out) - y, axis=-1)
    accuracy = tf.reduce_mean(hamming_distance)

    loss = losses_default[loss_fun](y, out)
    train_step = optimizer(learning_rate=0.002).minimize(loss)

    init = tf.global_variables_initializer()

    with tf.compat.v1.Session() as sess:
        sess.run(init)
        train_dataset = np.array(input_data[:18 * len(input_data) // 20])
        train_values = np.array(output_data[:18 * len(input_data) // 20])

        test_dataset = np.array(input_data[-len(input_data) // 20:])
```

```
        test_values = np.array(output_data[-len(input_data) // 20:])
        for i in range(training_epochs):
            if i % display_step == 0:
                feed = {x: train_dataset, y: train_values}
                result = sess.run([loss, accuracy], feed_dict=feed)
                print('Accuracy at step %s: %f - loss: %f' % (i, result[1], result[0]))
            else:
                feed = {x: train_dataset, y: train_values}
            sess.run(train_step, feed_dict=feed)

        predict_values = tf.round(out).eval(feed_dict={x: test_dataset})

        if verbose:
            print(list(map(lambda x: hex(utils.to_int(x)), predict_values)))
            print(list(map(lambda x: hex(utils.to_int(x)), test_values)))
            print(list(map(lambda x: hex(utils.to_int(x)), test_dataset)))

        accuracy = tf.reduce_mean(tf.cast(hamming_distance, "float"))

        acc = accuracy.eval(feed_dict={x: test_dataset, y: test_values})
        los = loss.eval(feed_dict={x: test_dataset, y: test_values})

        acc1 = accuracy.eval(feed_dict={x: train_dataset, y: train_values})
        los1 = loss.eval(feed_dict={x: train_dataset, y: train_values})
        timer = time.time() - timer

        if verbose:
            print("testing accuracy: {}".format(acc))
            print("testing Loss: {}".format(los))

            print("testing accuracy: {}".format(acc1))
            print("testing Loss: {}".format(los1))


            print("work time: {}s".format(timer))
            print("End Training with epoch {}\n\n\n".format(training_epochs))

        sess.close()

    return acc, los, acc1, los1, timer, predict_values, test_values


def multilayer_perceptron(x, hidden, num_classes, number_layers, activation_fun=tf.nn.sigmoid,
                          first_activation=False, drop_out=False):
    """
    Function to create layers.
    :param x: input data
    :param hidden: vector of hidden layer's sizes
    :param num_classes:  size of output layer
    :param num: number of layers(>=2)
    :param activation_fun: activation function
    :param first_activation: use or not activation function on first hidden layer
    :param drop_out: using of drop out layer
    :return: output layer
    """

    keep_prob = tf.compat.v1.placeholder(tf.float32)
```

```

    # Hidden layer with activation
    layer = tf.layers.dense(x, hidden[0],
                            activation=activation_fun if first_activation else
                               None,
                            kernel_initializer=tf.initializers.ones(),
                            bias_initializer=tf.initializers.ones())

    # Hidden layer with activation
    for i in range(1, number_layers):
        layer_new = tf.layers.dense(layer, hidden[i],
                                    activation=activation_fun)

        if drop_out:
            layer = tf.layers.dropout(layer, keep_prob)
        else:
            layer = layer_new

    # Output layer with linear activation
    out_layer = tf.layers.dense(layer, num_classes, activation=None)
    return out_layer


def recurrent_perceptron(x, hidden, num_classes, number_layers, activation_fun=
    tf.nn.sigmoid,
                         first_activation=False, drop_out=False):
    """
    Function to create layers.
    :param x: input data
    :param hidden: vector of hidden layer's sizes
    :param num_classes:  size of output layer
    :param num: number of layers(>=2)
    :param activation_fun: activation function
    :param first_activation: use or not activation function on first hidden
       layer
    :param drop_out: using of drop out layer
    :return: output layer
    """

    keep_prob = tf.compat.v1.placeholder(tf.float32)

    # Hidden layer with activation
    layer = tf.layers.dense(x, hidden[0],
                            activation=activation_fun if first_activation else
                               None,
                            kernel_initializer=tf.initializers.ones(),
                            bias_initializer=tf.initializers.ones())

    # Hidden layer with activation
    for i in range(1, number_layers):
        layer_new = tf.layers.dense(tf.concat([layer, x], 1), hidden[i],
                                    activation=activation_fun)

        if drop_out:
            layer = tf.layers.dropout(layer, keep_prob)
        else:
            layer = layer_new

    # Output layer with linear activation
    out_layer = tf.layers.dense(tf.concat([layer, x], 1), num_classes,
```

```
        activation=None)
    return out_layer


def init_multilayer_network(input_data: np.array, output_data: np.array, n_input: int, n_hidden: list,
                            n_classes: int, number_layers: list, learning_rate=0.001, training_epochs=100,
                            display_step=100, activation=tf.nn.sigmoid, optimizer=tf.compat.v1.train.AdamOptimizer,
                            loss_fun: str = "square_difference", verbose: bool = True):
    """
    Create multilayer neural network.
    :param input_data: np.array with input values
    :param output_data: np.array with output values
    :param n_input: neuron's number on first layer
    :param n_hidden: list with neuron's number on hidden layers
    :param n_classes: number of output classes
    :param number_layers: number of layers
    :param training_epochs: number of training epochs
    :param display_step:
    :param optimizer:
    :param loss_fun:
    :param verbose:
    :return: Accuracy, Loss, Predict Array, Real Array
    """
    timer = time.time()
    x = tf.compat.v1.placeholder(tf.float32, [None, n_input])
    y = tf.compat.v1.placeholder(tf.float32, [None, n_classes])

    prediction = multilayer_perceptron(x, n_hidden, n_classes, number_layers,
                                       activation_fun=activation, first_activation=True)

    hamming_distance = tf.math.count_nonzero(tf.round(prediction) - y, axis=-1)
    accuracy = tf.reduce_mean(hamming_distance)
    loss = losses_default[loss_fun](y, prediction)

    optimizer = optimizer(learning_rate=learning_rate).minimize(loss)

    init = tf.global_variables_initializer()

    with tf.compat.v1.Session() as sess:
        sess.run(init)

        train_dataset = np.array(input_data[:18 * len(input_data) // 20])
        train_values = np.array(output_data[:18 * len(input_data) // 20])

        test_dataset = np.array(input_data[-len(input_data) // 20:])
        test_values = np.array(output_data[-len(input_data) // 20:])

        for i in range(training_epochs):
            if i % display_step == 0:
                feed = {x: train_dataset, y: train_values}
                result = sess.run([loss, accuracy], feed_dict=feed)
                print('Accuracy at step %s: %f - loss: %f' % (i, result[1], result[0]))
            else:
                pass
```

```python
                sess.run(optimizer, feed_dict=feed)

            predict_values = sess.run(prediction, feed_dict={
                x: test_dataset,
            })

            predict_values1 = sess.run(prediction, feed_dict={
                x: train_dataset,
            })

            if verbose:
                print(list(map(lambda x: hex(utils.to_int(x)), predict_values)))
                print(list(map(lambda x: hex(utils.to_int(x)), test_values)))
                print(list(map(lambda x: hex(utils.to_int(x)), test_dataset)))

            acc = accuracy.eval(feed_dict={x: test_dataset, y: test_values})
            los = loss.eval(feed_dict={x: test_dataset, y: test_values})

            acc1 = accuracy.eval(feed_dict={x: train_dataset, y: train_values})
            los1 = loss.eval(feed_dict={x: train_dataset, y: train_values})
            timer = time.time() - timer

            if verbose:
                print("testing accuracy: {}".format(acc))
                print("testing Loss: {}".format(los))

                print("testing accuracy: {}".format(acc1))
                print("testing Loss: {}".format(los1))

                print("work time: {}s".format(timer))
                print("End Training with epoch {}\n\n\n".format(training_epochs))

            sess.close()

    return acc, los, acc1, los1, timer, \
           list(map(lambda x: utils.to_int(x), predict_values)),\
           list(map(lambda x: utils.to_int(x), test_values))


def init_recurrent_network(input_data: np.array, output_data: np.array, n_input: int, n_hidden: list,
                           n_classes: int, number_layers: list, learning_rate=0.001, training_epochs=100,
                           display_step=10, activation=tf.nn.sigmoid, optimizer=tf.compat.v1.train.AdamOptimizer,
                           loss_fun: str = "square_difference", verbose: bool = True):
    """
    Create recurrent neural network.
    :param input_data: np.array with input values
    :param output_data: np.array with output values
    :param n_input: neuron's number on first layer
    :param n_hidden: list with neuron's number on hidden layers
    :param n_classes: number of output classes
    :param number_layers: number of layers
    :param training_epochs: number of training epochs
    :param display_step:
    :param optimizer:
    :param loss_fun:
    :param verbose:
```

```python
    :return: Accuracy, Loss, Predict Array, Real Array
    """

    x = tf.compat.v1.placeholder(tf.float32, [None, n_input])
    y = tf.compat.v1.placeholder(tf.float32, [None, n_classes])

    prediction = recurrent_perceptron(x, n_hidden, n_classes, number_layers,
                                      activation_fun=activation, first_activation=True)

    hamming_distance = tf.math.count_nonzero(tf.round(prediction) - y, axis=-1)
    accuracy = tf.reduce_mean(hamming_distance)
    loss = losses_default[loss_fun](y, prediction)

    optimizer = optimizer(learning_rate=learning_rate).minimize(loss)

    init = tf.global_variables_initializer()

    with tf.compat.v1.Session() as sess:
        sess.run(init)

        train_dataset = np.array(input_data[:18 * len(input_data) // 20])
        train_values = np.array(output_data[:18 * len(input_data) // 20])

        test_dataset = np.array(input_data[-len(input_data) // 20:])
        test_values = np.array(output_data[-len(input_data) // 20:])

        for i in range(training_epochs):
            if i % display_step == 0:
                feed = {x: train_dataset, y: train_values}
                result = sess.run([loss, accuracy], feed_dict=feed)
                print('Accuracy at step %s: %f - loss: %f\n' % (i, result[1], result[0]))
            else:
                feed = {x: train_dataset, y: train_values}
            sess.run(optimizer, feed_dict=feed)

        predict_values = sess.run(prediction, feed_dict={
            x: test_dataset,
        })

        if verbose:
            print(list(map(lambda x: hex(utils.to_int(x)), predict_values)))
            print(list(map(lambda x: hex(utils.to_int(x)), test_values)))
            print(list(map(lambda x: hex(utils.to_int(x)), test_dataset)))

        accuracy = tf.reduce_mean(tf.cast(hamming_distance, "float"))
        acc = accuracy.eval(feed_dict={x: test_dataset, y: test_values})
        los = loss.eval(feed_dict={x: test_dataset, y: test_values})

        if verbose:
            print("testing accuracy: {}".format(acc))
            print("testing Loss: {}".format(los))

        sess.close()

    return acc, los, \
           list(map(lambda x: utils.to_int(x), predict_values)), \
           list(map(lambda x: utils.to_int(x), test_values))
```

networks.py