**ТИТУЛЬНЫЙ ЛИСТ**

**Содержание**

Введение …………………………………………………………………..…….3

1 Основные понятия в области рекуррентных нейросетей …………....……4

2 Алгоритм Левенберга-Марквардта для сети Элмана …………….....……..9

3 Исследование работы нейронной сети Элмана в Python …………………………….…14

Заключение …………………………………………………………..…..……..19

Литература …………………………………………………………...…..….….20

ПРИЛОЖЕНИЕ А

Листинг программной реализации рекуррентной нейросети в Python ...…21

**Введение**

Для обеспечения высокого уровня надёжности современных телекоммуникационных сетей применяются специализированные системы автоматического управления, осуществляющие мониторинг основных параметров системы и реагирующие заранее заданным образом на отклонения их от нормы. При этом сам процесс мониторирования может занимать может занимать довольно значительный промежуток времени, что обуславливает неизбежное запаздывание управляющих сигналов, существенно ограничивая эффективность системы управления. Существенное повышение эффективности управления может быть достигнуто с помощью систем с упреждающим воздействием, формируемым по результатам прогнозирования сетевого трафика.

Кроме этого, прогнозирование трафика используется в сетевых маршрутизаторах для энергосбережения, позволяя в моменты низкого трафика отключать дополнительные процессоры, существенно снижая энергопотребление оборудования, и своевременно включать их в моменты увеличения сетевых нагрузок. Особенно остро данная проблема стоит в автономных беспроводных сенсорных сетях. Другое, не менее важное приложение прогнозирования трафика связано с оптимизацией использования сетевых ресурсов, имеющих низкую пропускную способность. Особенно остро эта проблема стоит в мобильных сетях, отличающихся высокой неравномерностью загруженности. Приложения прогнозирования трафика вовсе не ограничиваются областью телекоммуникационных технологий, острая необходимость в нём наблюдается также в интернет-маркетинги и многих других областях.

Точность прогнозирования трафика напрямую связана с эффективностью решения обозначенных задач, в связи с чем, вопросам её повышения в последнее время уделяется всё больше внимания со стороны многих авторитетных исследователей. Прогностическая способность применяемых для этой цели линейных авторегрессионных моделей явно не удовлетворяет имеющимся требованиям, что заставляет искать новые пути решения, среди которых, нейросети являются одним из наиболее перспективных направлений.

**1 Основные понятия в области рекуррентных нейросетей**

Рекуррентные нейронные сети представляют собой сети, выходное состояние которых определяется не только входным воздействием, но также зависит от их внутреннего состояния, которое в сою очередь зависит от предшествующих входных воздействий. Таким образом, отклик рекуррентной сети на заданное постоянное входное воздействие изменяется во времени до тех пор, пока она не достигнет устойчивого внутреннего состояния.

По своей сути, рекуррентные нейронные сети представляют собой нелинейные системы с бесконечной импульсной характеристикой, и соотносятся с сетями прямого распространения точно так же, как линейные БИХ и КИХ фильтры. В силу указанной специфики, в отличие от сетей прямого распространения, динамика работы рекуррентной нейронной сети находится в тесной взаимосвязи с её параметрами и структурой, и поэтому – является важнейшей характеристикой, помимо статических соотношений входных и выходных воздействий. Следовательно, порядок поступления входных воздействий влияет на её работу также сильно, как и значения воздействий.

Благодаря своим особенностям, рекуррентные сети хорошо подходят для анализа и моделирования упорядоченных данных, в частности – различных временных последовательностей, изображений, текстов и т.п. Именно в этой области они получили наиболее широкое распространение. Однако это далеко не единственное, а главное – далеко не всегда наилучшее средство обозначенных задач, обладающее как достоинствами, так и недостатками.

В анализе временных последовательностей достаточно успешно могут использоваться и обычные нейронные сети прямого распространения. В этом случае, в качестве предикторов используются значения отсчётов с различными временными задержками, а в качестве прогнозируемой переменной – значение текущего отсчёта анализируемой последовательности. Обучающая выборка при этом формируется путём многократного последовательного смещения анализируемого временного интервала в будущее точно так же, как это делается при построении линейных авторегрессионных моделей.

Обучение сети прямого распространения на упорядоченных данных в целом практически ничем не отличается от обычных задач обучения и осуществляется с помощью стандартных алгоритмов. Простота использования и повышенная гибкость в сравнении с линейными моделями представляются серьёзными достоинствами сетей прямого распространения. Однако, при моделировании сложных долговременных закономерностей, в силу конечной импульсной характеристики и необходимости использования большого временного окна, они получаются очень громоздкими. Это негативно сказывается на обобщающей способности модели, и точности получаемых с её помощью прогнозов – одной из главных целей анализа временных рядов.

Рекуррентные сети в этом отношении имеют существенное преимущество, так как способны учитывать предысторию далёкого прошлого, находящегося за пределами используемого временного окна. Потенциально, они позволяют построить более компактную, а следовательно, и более качественную модель по сравнению с сетями прямого распространения. Точно так же, как БИХ фильтры, реализуя одну и ту же форму АЧХ, оказываются гораздо компактнее КИХ фильтров. Однако это преимущество в полной мере проявляется только при анализе вполне определённых данных. В противном случае, применение рекуррентной нейронной сети может быть неоправданным.

Наряду с указанными преимуществами, рекуррентные нейронные сети обладают рядом серьёзных недостатков. Главный недостаток, отличающий их от сетей прямого распространения, заключается в отсутствии гарантированной устойчивости работы и опасности самовозбуждения, приводящего к неограниченному возрастанию выходных сигналов и полной утрате работоспособности. Ситуация усугубляется тем, что в отличие от линейных рекурсивных фильтров, нелинейные многомерные рекурсивные системы в общем случае практически не подаются анализу. Известны лишь исключительные частные случаи, с доказанной устойчивостью, например – сеть Хопфилда.

Другой серьёзный недостаток рекуррентных нейронных сетей заключается в необходимости использования для их обучения весьма сложных и трудоёмких алгоритмов. Алгоритмы, применяемые для обучения сетей прямого распространения для этих целей совершенно, не подходят, так как не позволяют учесть влияние на состояние сети предыдущих входных воздействий.

Применительно к обучению рекуррентных нейронных четей, наиболее широкое распространение получил алгоритм обратного распространения ошибки с развёртыванием сети по времени, позволяющий вычислять адекватные градиенты весов. Идея развёртывания во времени заключается в том, что любой рекуррентный слой путём явного введения всех предыдущих входных воздействий, может быть представлен в виде эквивалентно сети прямого распространения, для обучения которой с успехом может быть использован обычный метод обратного распространения ошибки.

Развёрнутый во времени слой нейронной сети представляет собой множество копий исходного рекуррентного слоя, рекуррентные входы каждого из которых, вместо своих же выходов, связаны с выходами предыдущего вспомогательного слоя. При этом, все копии исходного рекуррентного слоя имеют одинаковые значения одноимённых весов, а число вспомогательных слоёв определяется количеством временных шагов, прошедших к моменту обучения. Таким образом, с увеличением количества пройденных временных шагов размеры развёрнутой сети постоянно увеличиваются, и она по своей сути превращается в глубокую нейросеть со всеми её нюансами.

Идея развёртывания во времени позволяет полностью исключить из рассмотрения внутренние состояния рекуррентной сети и базируется на предположении о постоянстве всех её параметров на протяжении анализируемого временного интервала. Недостатки оригинального алгоритма с развёртыванием во времени очевидны. С увеличением длительности анализируемых временных интервалов, требования к вычислительным ресурсам, а в особенности – к машинной памяти, резко возрастают. Кроме того, возникают известные проблемы глубокого обучения, в частности - проблема исчезающего градиента, исследованная Зеппом Хохрайтером ещё в 1991 г. Всё это сильно ограничивает длительность анализируемого временного интервала.

Учитывая вышеуказанные особенности, для обучения рекуррентных нейронных сетей на больших временных интервалах применяется метод частичного развёртывания во времени, позволяющий ограничить вычислительную сложность алгоритма. Но, вместе с этим, ограничивается предыстория, которую сеть способна учитывать при анализе долгосрочных закономерностей, т.е. теряется её основное принципиальное преимущество.

Как любая другая нейронная сеть, рекуррентная сеть предполагает возможность аналоговой реализации. Однако на практике, как правило, применяется только программная реализация, в которой они представляются в виде дискретных систем. В настоящее время известно сравнительно небольшое число рекуррентных нейросетевых структур, нашедших практическое применение. Базовая архитектура таких сетей с полносвязной схемой разработана ещё в 1980 г. Каждый нейрон такой сети имеет меняющийся во времени порог активации, а каждое соединение имеет меняющийся во времени вещественный вес. Узлы сети разделяются на входные выходные и скрытые.

Сеть имеет 2 режима обучения: обучение с учителем и обучение с подкреплением. В первом случае, на каждом временном шаге, на входы подаются обучающие примеры. При этом остальные узлы активируются в зависимости от уровня входных воздействий, а выходы устанавливаются в свои стационарные состояния, которые на новом временном шаге передаются на входы нейронов следующего уровня. В обучении с подкреплением вместо целевых сигналов, к которым адаптируется сеть, используются различные оценочные функции, по которым и проводится оценка качества работы сети.

Среди более современных рекуррентных нейросетевых структур можно выделить нейронные сети Элмана и Джордана, которые так же иногда называют простыми рекуррентными нейронными сетями. Структурная схема сети Элмана представлена на рисунке 1. Это трёхслойная рекуррентная нейронная сеть с одним скрытым слоем, который и является рекуррентным. Выходы скрытого слоя сети Элмана через элементы задержки z-1 определяют её внутренние состояния, связанные со входами этого же слоя.

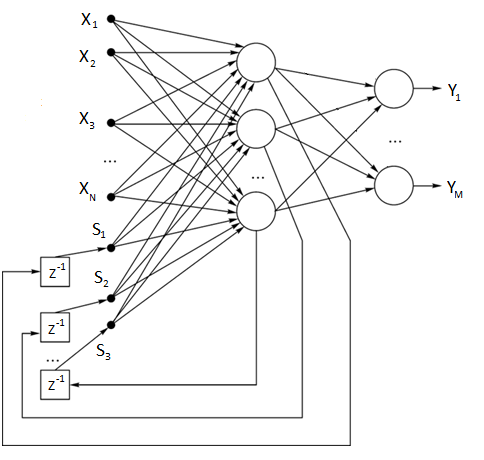


Рисунок 1 – Структурная схема рекуррентной сети Элмана

Сеть Джордана аналогична сети Элмана, лишь с тем отличием, что внутренние состояния (контекстные блоки) связаны в ней не с выходами скрытого слоя, а с внешними выходами нейросети. Таким образом, в сети Элмана количество контекстных блоков определяется числом внутренних нейронов, а в сети Джордана – количеством внешних выходов.

Среди других рекуррентных нейросетевых структур можно отметить нейронный компрессор исторических данных, предназначенный для выделения существенных особенностей анализируемого динамического процесса, а так же сеть долго-краткосрочной памяти (LSTM), способная запоминать свои состояния на длительные сроки, которая в настоящее время является пожалуй самой популярной рекуррентной нейросетевой структурой, а так же управляемые рекуррентные блоки (GRU) — имеющие меньше параметров, чем LSTM, и не имеющие выходного управления, но обеспечивающие на некоторых задачах сопоставимую с LSTM производительность.

**2 Алгоритм Левенберга-Марквардта для сети Элмана**

Выходное значение сети в момент времени t определяется уравнением

, (1)

где *b* - внешнее смещение, **w** – выходные веса, **h**t – выходной вектор скрытого слоя, значения которого в свою очередь определяются уравнением

, (2)

где **U** – матрица входных весов, **x** – входной вектор, **V** – матрица рекуррентных весов, **h**t-1 – вектор внутренний состояний, **s** – внутренние смещения.

В принятых обозначениях, ошибка прогноза сети определяется как

, (3)

где *y*\* - истинное значение прогнозируемой переменной.

Выходное смещение *b* и выходные веса **w** не влияют на охваченные обратной связью выходы скрытого слоя **h**t. Поэтому, частные производные ошибки по этим параметрам определяются достаточно просто

, . (4)

Для определения частных производных выходной ошибки по параметрам скрытого слоя предварительно необходимо вычислить производные его выходов. Применяя правило дифференцирования сложной функции



получаем следующие рекуррентные формулы

 (5)

где δi,j – дельта Кронекера, **v**k – обозначает k-ю строку матрицы **V**.

Объединяя элементы *h*k в векторы, получаем более наглядные формулы

 (6)

где **1**i вектор, состоящий из нулей и единицы в i-м положении.

На основании частных производныхвыходов скрытого слоя, частные производные выходной ошибки сети определяются по формулам

 (7)

Из (7) следует, что для вычисления частных производных выходной ошибки по параметру используется только частная производная выхода, непосредственно зависящего от этого параметра, а производные остальных выходов используются только в процедуре рекуррентного обновления частных производных, так как они могут повлиять на состояние рассматриваемого выхода только в последующие моменты времени, через обратные связи.

Для практической реализации алгоритма рекуррентного обновления частных производных выходов сети (6), все эти векторы удобно объединить в одной матрице, которая может быть представлена в следующем виде

, (8)

где n- число нейронов скрытого слоя сети, m – число её входов.

На первом шаге алгоритма рекуррентного обновления матрицы частных производных *d***H** вычисляется вспомогательный весовой вектор

, (9)

затем, используя этот вектор, вычисляется матричное произведение

, (10)

в котором строки **V** умножается на соответствующие элементы вектора **p**.

Далее, выполняется самая трудоёмкая операция – умножение матриц

. (11)

Завершающие операции рекуррентного обновления *d***H** заключаются в модификации диагоналей в *n*x*n* блоках матрицы *d***H**\*. Согласно (8) эта матрица содержит 1+*m*+*n* таких блоков. Для завершения обновления согласно (6), к диагональным элементам первого блока необходимо прибавить элементы ранее вычисленного весового вектора **p**. К элементам диагонали второго блока необходимо прибавить элементы вектора **p** умноженные на *x*1. К элементам диагонали третьего блока – элементы **p**, умноженные на *x*2, и т.д., до блока *m*+1. К элементам диагонали блока *m*+2 необходимо прибавить элементы весового вектора **p**, умноженные на *ht-1,1*, к элементам диагонали блока *m*+3 необходимо прибавить элементы **p**, умноженные на *ht-1,2*, и т.д., до конца.

Можно заметить, что именно диагональные элементы блоков *d***H** содержат элементы, необходимые для вычисления частных производных выходной ошибки сети, которые получаются из этих элементов путём умножения на соответствующие элементы **w**, взятые с отрицательным знаком.

Обозначив диагонали d**H** как **DH**1 … **DH**m+n+1, вектор частных производных выходной ошибки, без учёта его n+1 компонент (7), записывается как

. (12)

На основании векторов частных производных **d**, найденных для каждого момента времени, представленного в обучающей выборке, вектор градиента суммы квадратов ошибок сети определяется следующим образом

, (13)

где *L* – количество отсчётов времени в обучающей выборке, **e** – вектор ошибок, состоящий из выходных ошибок сети в моменты времени t1, t2, …tL,

В практических вычислениях непосредственное использование (13) может оказаться неудобным, так как для сохранения всех **d**t могут потребоваться очень большие объёмы памяти. Обычно, более предпочтительным является метод вычисления градиента с накоплением, по формуле

 (14)

Исходя из найденного градиента среднеквадратической ошибки можно реализовать градиентный алгоритм обучения сети, описываемый формулой

, (15)

где μ – шаг алгоритма, способ определения которого зависит от конкретной реализации алгоритма, **θ** – обобщённый вектор параметров сети, который в принятых обозначениях определяется следующим образом

, (16)

где **u**i и **v**i – обозначают соответствующие вектор-столбцы матриц **U** и **V**.

Градиентные методы имеют сравнительно медленную сходимость и, соответственно – обеспечивают сравнительно низкую скорость обучения, что является весьма серьёзным недостатком при работе в реальном масштабе времени. Значительного повышения скорости сходимости алгоритма, хотя и за счёт увеличения вычислительной сложности одного шага, можно достичь за счёт применения различных квазиньютоновских методов. Метод Ньютона в своём оригинальном виде для этих целей практически не используется, из-за неприемлемой вычислительной сложностью и известными проблемами с весьма ограниченной областью сходимости. По этому методу

, (17)

где **H** – матрица Гёссе, которая вычисляется следующим образом

 (18)

где **H**\* - матрицы Гёссе для отдельных ошибок *ei*, в вычислении которых и состоит основная сложность; **J** – матрица Якоби, определяемая как

. (19)

В широко распространённом алгоритме Левенберга-Марквардта предложено использовать следующую аппроксимацию матрицы Гёссе

 (20)

где α – регуляризующий параметр, оптимальное значение которого зависит от величины ошибок обучения, **E** – диагональная единичная матрица.

При малых ошибках обучения *ei* аппроксимация (20) работает достаточно хорошо и обеспечивает весьма быструю сходимость, хотя она и в этом случае, в отличие от классического метода Ньютона, остаётся линейной.

Таким образом, реализация алгоритма Левенберга-Марквардта в дополнение ко всему требует вычисления матрицы **J**T**J**, что так же, как и в случае с вычислением градиента предпочтительно делать методом накопления

. (21)

Кроме этого, на завершающем этапе, для определения приращений параметров сети **r**, необходимо решить систему линейных уравнений

. (22)

Только после этого, выполняется обновление параметров сети

. (23)

Следует отметить, что в случае сети Элмана, на фоне вычислений, связанных с рекуррентным обновлением частных производных, увеличение вычислительной сложности одного шага алгоритма, связанное с применением метода Левенберга-Марквардта представляется не существенным. Поэтому, его применение может быть предпочтительным не только с позиции уменьшения числа эпох обучения, но также и в аспекте общей вычислительной сложности алгоритма, за весь период обучения нейронной сети.

**3 Исследование работы сети Элмана в Python**

Описанный выше алгоритм обучения рекуррентной нейросети Элмана реализован на языке программирования Python. В программе использованы только самые базовые библиотеки, такие как NumPy (включающую разнообразные математические операции и функции линейной алгебры), Pandas (используемая для загрузки анализируемых данных), а также Matplotlib (используемая для графической визуализации результатов исследования).

Отсутствие привязки к готовым нейросетевым библиотекам, таким как Keras и т.п. позволяет всесторонне исследовать любые характеристики алгоритма, и предоставляет возможность его произвольной модификации. Полный листинг разработанной программы приведён в приложении А.

Загрузка исходных данных производится из файла MS Excel, расположенного в рабочей директории Python. Данный файл представляет собой отчёт по реальной статистике сетевого трафика, сгенерированный программой сетевого мониторинга DU Meter. В первом столбце сгенерированного отчёта расположены даты, во втором столбце – соответствующий объём входящего трафика, в третьем столбце – объём исходящего трафика и в четвёртом столбце – объём общего трафика (сумма значений 2 и 3 столбцов). Кроме этого, в пятом столбце отчёта, для достаточно продолжительных подключений, длящихся более минуты, представлены данные об их длительности.

В начальной части программы задаются параметры алгоритма – это число нейронов входного слоя m, число нейронов скрытого слоя n, а также число эпох обучения NEpoch и параметр сходимости алгоритма alpha. В соответствии с заданием, первые два параметра были установлены как m=24 и n=11, а остальные параметры произвольно менялись в ходе исследования.

Несмотря на небольшие размеры исследуемой нейросети, программа выполняется довольно медленно (отчасти из-за рудоёмкости алгоритма, а отчасти – из-за интерпретатора Python). Учитывая эту особенность, программа делает вывод на каждой итерации, что позволяет наблюдать за ходом обучения, как это сделано во многих аналогичных программах.

На каждой итерации в консоль Python выводится номер текущей эпохи обучения (начиная с 1), значение среднего квадрата ошибки подгонки модели под обучающие данные MSE, а также, полученное на данной эпохе обучение значение прогноза. Пример вывода программы приведён на рисунке 2.

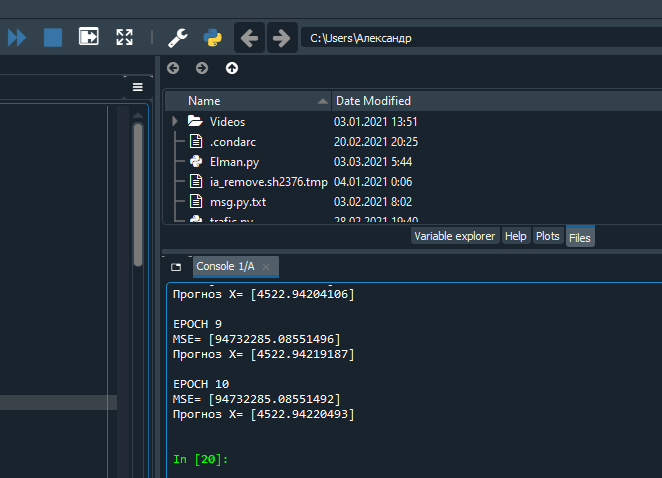


Рисунок 2 – Вывод программы в консоль Python

Для исследования сходимости алгоритмы проводилось обучение сети при разных параметрах α. Полученные кривые обучения представлены на рисунке 3. Из них видно, что для α≤103 алгоритм сходится уже за 1 шаг.

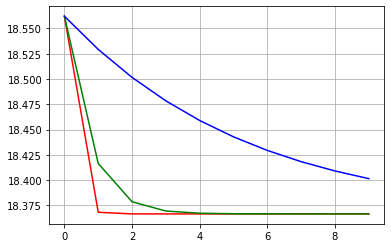


Рисунок 3 – Зависимость MSE от номера эпохи обучения

(для α=103, α=104 и α=105)

На рисунке 4 представлены наложенные друг на друга графики анализируемого входящего трафика и выходного значения нейросети на последней эпохе обучения, соответствующего оптимальным значениям параметров.



Рисунок 4 – Выходной сигнал обученной сети и данные трафика

Из этих графиков видно, что выходной сигнал представляет собой почти прямую линию, проходящую по среднему уровню анализируемого сетевого трафика. Следовательно, оптимальный прогноз представляет собой не зависящую от времени константу и сеть данной структуры практически не способна предсказать изменения трафика, в то время как именно это и является главной целью прогнозирования. Полученный результат свидетельствует о явной неадекватности выбранной нейросетевой структуры исследуемым данным сетевого трафика. Но, так как изменение структуры сети в задании не предусмотрено, исследования, связанные с подбором оптимального числа входных нейронов и числа нейронов скрытого слоя сети, не проводились.

Для исследования возможности прогнозирования данной нейросетевой структурой других сигналов, в загружаемом файле Excel отчёта, в свободном столбце рабочего листа, был создан модельный сигнал в виде синусоиды (пример создания представлен на рисунке 5), с периодом 100ч ~ 4 суток – явно не укладывающийся в апертуру исследуемой нейросети, шириной 24 ч.

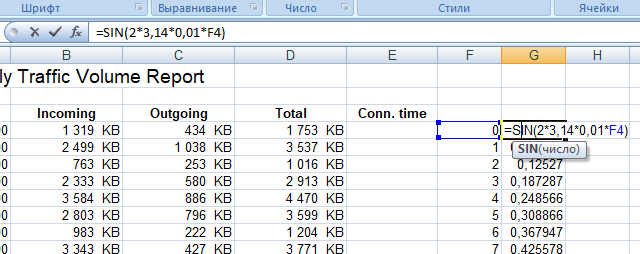


Рисунок 5 – Пример создания модельного сигнала в MS Excel

После этого был запущен процесс обучения с разными значениями параметра α. На рисунке 6 представлены графики выходного сигнала нейросети в сравнении с модельным сигналом. Эти графики полностью подтверждают работоспособность сети и наличие у неё прогностических свойств. В случае параметра сходимости α=103 качество подгонки оказывается сравнительно невысоким и отчётливо прослеживается тенденция сети сглаживать острые вершины, чем видимо и объясняется низкое качество прогнозирования реального трафика. При уменьшении α до 102 качество подгонки резко возрастает, а при дальнейшем уменьшении достигается почти точная подгонка.

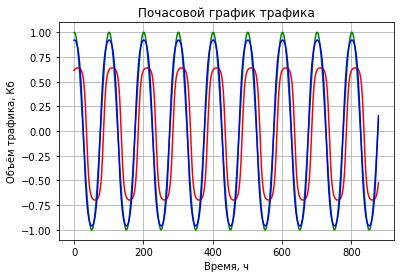


Рисунок 6 – Выходной сигнал обученной сети и модельный сигнал

Кроме этого, были исследованы так же и другие сигналы. Так на рисунке 7 представлены графики отклика обученной нейросети вместе с модельным сигналом, представляющим собой сумму 2-х синусоид, с периодами 24 и 16 часов, с прибавленным сигналом случайного шума, равномерно распределённого в диапазоне от 0 до 1. В данном случае появляется возможность определения оптимального значения α, обеспечивающего максимальную точность прогноза. На рисунке 7 представлена зависимость остаточной mse обученной сети, из которой следует, что оптимальное значение α=10.

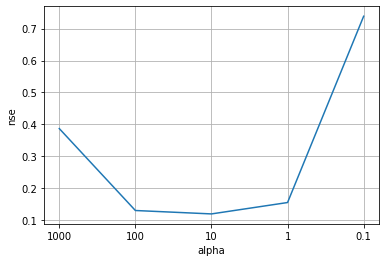


Рисунок 7 – Зависимость точности подгонки от параметра α

На рисунке 8 приведён отклик сети, соответствующий оптимальному α.

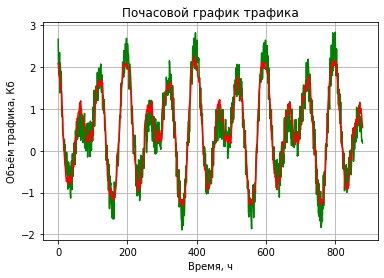


Рисунок 8 – Выходной сигнал обученной сети и модельный сигнал

**Заключение**

В результате выполнения данной работы, в полном соответствии с заданием, для прогнозирования сетевого трафика была разработана рекуррентная нейронная сеть типа Элмана, содержащая 24 входных и 11 скрытых нейронов, с полносвязной схемой соединения входного и скрытого слоёв, и содержащая только 1 нейрон в выходном слое. Для этой сети разработан алгоритм обучения, основанный на методе обратного распространения ошибки и квазиньютоновского метода оптимизации Левенберга-Марквардта.

Разработанный алгоритм реализован на языке Python, с использованием библиотеки математических вычислений NumPy. В качестве примера данных анализируемого сетевого трафика взят отчёт программы сетевого мониторинга DU Meter. Нейронная сеть продемонстрировала на них очень плохую прогностическую способность, однако на других искусственно сгенерированных модельных сигналах, состоящих из смеси синусоид и равномерно распределённого шума она показала очень хорошие результаты.

Из этого можно сделать вывод о неудачном выборе образца для тренировки нейросети, очень плохо поддающегося прогнозированию, либо – о неудачно выбранную структуру нейросети, неадекватной выбранному образцу.

В результате проведённых исследований было достоверно обнаружено наличие достаточно высокой чувствительности разработанной нейросети к медленным периодическим колебаниям, период которых значительно превышает размеры используемого временного окна шириной в 24 дискретных отсчёта. Это полностью подтверждает работоспособность рекуррентной части сети и правильную работу разработанного алгоритма её обучения.

В ходе проведённых исследований так же была продемонстрирована высокая эффективность разработанного алгоритма обучения, показывающего быструю монотонную сходимость на всех без исключения принимавших в исследовании обучающих примерах. Путём подбора оптимального значения удаётся добиться максимального качества подгонки модели под данные.

Всё это позволяет заключить об успешном выполнении работы.

**Литература**

1. Петров В.В. Структура телетрафика и алгоритм обеспечения качества обслуживания при влиянии эффекта самоподобия. Автореферат диссертации. Москва. -2005. - 20 с.
2. Rakkiyappan, R.; Chandrasekar, A.; Lakshmanan, S.; Park, Ju H. Exponential stability for markovian jumping stochastic BAM neural networks with mode-dependent probabilistic time-varying delays and impulse control (англ.) // Complexity : journal. — 2015. — 2 January (vol. 20, no. 3). — P. 39—65.
3. C.W. Omlin, C.L. Giles, «Constructing Deterministic Finite-State Automata in Recurrent Neural Networks» Journal of the ACM, 45(6), 937—972, 1996.
4. C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun, Y.C. Lee, «Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks», Neural Computation, 4(3), p. 393, 1992.
5. W. Maass, T. Natschläger, and H. Markram. A fresh look at real-time computation in generic recurrent neural circuits. Technical report, Institute for Theoretical Computer Science, TU Graz, 2002.
6. Seppo Linnainmaa (1970). The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master’s Thesis (in Finnish), Univ. Helsinki, 6-7.
7. Robinson, T. A real-time recurrent error propagation network word recognition system. — ICASSP. Icassp'92: 617–620. — 1992.

**ПРИЛОЖЕНИЕ А**

Листинг программной реализации рекуррентной нейросети в Python

# -\*- coding: utf-8 -\*-

"""

РЕКУРРЕНТНАЯ НЕЙРОСЕТЬ ЭЛМАНА

алгоритм Левенберга-Марквардта

"""

m=24 # число нейронов входного слоя

n=11 # число нейронов скрытого слоя

NEpoch=10 # число эпох обучения

alpha=1e3 # параметр сходимости

# Загрузка данных входного трафика из 1 столбца таблицы Excel

import pandas as pd

df1 = pd.read\_excel('trafic.xls', usecols = [1])

x=df1.iloc[2:,0].values

x=x[0:x.size-2]

import numpy as np # импорт математической библиотеки NumPy

err=np.zeros([x.size-m,1]) # инициализация векторов и матриц:

p=np.zeros([n,1])

pV=np.zeros([n,n])

h=np.zeros([n,1])

mse=np.zeros([NEpoch,1]) # вектор среднего квадрата ошибки

xp=np.zeros([NEpoch,1]) # вектор прогноза

d=np.zeros([1+(2+m+n)\*n,1])

w=np.zeros([1+(2+m+n)\*n,1])

ib=0 # начальные позиции весов:

iw=1

iis=1+n

iu=1+2\*n

iv=1+(2+m)\*n

for i in range(iv-1):

w[i]=0.1 # инициализация весов (кроме рекуррентных)

# основной цикл

for epoch in range(NEpoch):

dM=np.zeros([n,n\*(1+m+n)]) # обнуление матриц в начале новой эпохи

h0=np.zeros([n,1])

g=np.zeros([1+(2+m+n)\*n,1])

JJ=np.zeros([1+(2+m+n)\*n,1+(2+m+n)\*n])

for t in range(x.size-m+1):

for i in range(n): # вычисление выходного вектора

h[i]=w[iis+i]

for j in range(m):

h[i]+=x[j+t]\*w[iu+j\*n+i]

h[i]=np.tanh(h[i])

if t==x.size-m:

xp[epoch]=w[ib]

for i in range(n): xp[epoch]+=h[i]\*w[iw+i]

break

err[t]=x[t+m]-w[ib] # выходная ошибка

for i in range(n): err[t]-=h[i]\*w[iw+i]

for i in range(n): # обновление p и pV

p[i]=(1-h[i]\*\*2)

for j in range(n):

pV[i,j]=p[i]\*w[iv+i+j\*n]

dM=pV.dot(dM) # обновление dM

for j in range(n):

dM[j,j]+=p[j]

d[iis+j]=-w[iw+j]\*dM[j,j]

for i in range(m):

for j in range(n):

dM[j,j+(i+1)\*n]+=p[j]\*x[t+i]

d[iu+j+i\*n]=-w[iw+j]\*dM[j,j+(i+1)\*n]

for i in range(n):

for j in range(n):

dM[j,j+(i+1+m)\*n]+=p[j]\*h0[i]

d[iv+j+i\*n]=-w[iw+j]\*dM[j,j+(i+1+m)\*n]

d[0]=-1

for i in range(n):

d[iw+i]=-h[i]

h0[i]=h[i]

JJ+=d.dot(d.T) # вычисление JJ

g+=d\*err[t] # вычисление g

mse[epoch]+=err[t]\*\*2 # сумма квадратов ошибок

mse[epoch]/=(x.size-m)

print("EPOCH", epoch+1)

print("MSE=", mse[epoch])

print("Прогноз X=", xp[epoch],"\n")

for i in range(1+(2+m+n)\*n): JJ[i,i]+=alpha # регуляризация

r=np.linalg.solve(JJ, g) # решение СЛАУ

for i in range(1+(2+m+n)\*n): w[i]-=r[i] # пересчёт w

y=np.zeros(err.size)

time=np.zeros(err.size)

for i in range(err.size):

y[i]=x[i+m]-err[i]

time[i]=i

import matplotlib.pyplot as plt

plt.title("Почасовой график трафика") # заголовок

plt.xlabel("Время, ч") # ось абсцисс

plt.ylabel("Объём трафика, Кб") # ось ординат

plt.grid() # включение отображение сетки

plt.plot(time,x[24:x.size],'g',time,y,'r') # вывод графиков