Обучение больших гибридных нейронных сетей для прогнозирования временных рядов

Д.Ю. Нагорных

Аннотация. В статье описывается гибридный подход при создании нейронных сетей, используемых для прогнозирования временных рядов. Данный подход основан на объединении в рамках одной сети самообучающегося слоя Кохонена и многослойного персептрона. Приводится результат практических экспериментов обучения такой гибридной сети с помощью алгоритма «справедливого» обучения.

Ключевые слова: нейронные сети, гибридные нейронные сети, самоорганизующиеся карты, прогнозирование временных рядов, аппроксимация функций.

Введение

Задача прогнозирования временных рядов часто возникает на практике и на сегодняшний день является достаточно актуальной в различных областях человеческой деятельности. На данный момент существует множество подходов к ее решению, такие, например, как прогнозирование по тренду или адаптивное прогнозирование [3]. Оба этих метода, по большому счету, являются частными случаями нейронной сети с линейной функцией активации. Нейронные сети в более общем случае прекрасно зарекомендовали себя в качестве инструмента для прогнозирования временных рядов [1].

Процесс обучения нейронных сетей принято делить на обучение с учителем и обучение без учителя. В процессе работы по прогнозированию временного ряда использовалась нейронная сеть специфической структуры – гибридная сеть, состоящая из одного самообучающегося слоя Кохонена, а также трех персептронных слоев, обучаемых уже с учителем. Использование нейронной сети такой структуры позволило добиться неплохих экспериментальных результатов и на практике испробовать метод, который может применяться в более широком смысле для аппроксимации произвольных функций. Похожий подход описывался в [2] в качестве возможного направления исследований в сфере улучшения работы традиционных нейронных сетей обратного распространения ошибки. По сути, метод, описанный ниже, претендует на то, чтобы стать конструктивным алгоритмом построения нейронной сети, аппроксимирующей произвольную функцию, представленную в виде пар «вход-значение», с любой наперед заданной точностью.

Задача

Изначально перед автором стояла задача прогнозирования временного ряда котировок фьючерса на индекс РТС с минимальной ошибкой.

Решалась задача прогнозирования по значениям как самого ряда, то есть поиск автокорреляции, так и по значениям других временных рядов, которые, как предполагалось, могли ока-

зывает влияние на прогнозируемый ряд (например, котировок других биржевых инструментов). Другими словами, без ограничения общности можно считать, что входными значениями для нейронной сети являются значения некоторого временного ряда, а на выходе сети мы ожидаем увидеть прогноз будущего значения прогнозируемого временного ряда, возможно, отличного от подаваемого на вход. Глубина погружения временного ряда определялась из эмпиричесих предположений.

Предобработка входов

Как известно, нейронная сеть имеет на входе некоторое, заранее определенное, количество входов, а значений временного ряда в окне погружения может быть существенно больше. Более того, значительная их часть может быть шумом или понижать суммарную энтропию входов, то есть быть неинформативными за счет, например, линейной зависимости. Поэтому необходимо выделить, во-первых, заранее определенное количество значений для входа, а во-вторых, выбрать из них максимально информативные. Эту несложную задачу можно решить с помощью алгоритма прореживания (Рис. 1), поочередно удаляя точки с наименьшей площадью треугольника, образованного с двумя соседними точками. Повторяя эту процедуру необходимое число раз, мы можем получить требуемое число входов для нейронной сети. Эти входы будут максимально отличными друг от друга, а разность между ними будет иметь высокую информативность, что существенно ускоряет процесс обучения персептронного слоя гибридной сети, а также повышает качество обучения.

Подзадача прореживания входов в процессе работы решалась и с помощью дискретного преобразования Фурье с последующей фильтрацией частот с наименьшей мощностью [4]. На исходных данных этот алгоритм показал неудовлетворительные результаты и в целом не оправдал себя и этому есть некоторые объяснения. Во-первых, восстановленный ряд теряет экстремальные значения пиков, что ведет к снижению информативности входов. Во-вторых, за счет специфики самого метода, приводит к тому, что весьма близкие друг к другу

Рис. 1. Прореживание исходного ряда

По вертикальной оси – значения временного ряда, по горизонтальной – время

графики могут порождать абсолютно разные наборы частот. Это в свою очередь ведет к тому, что близкие примеры порождают абсолютно разные входные значения для нейронной сети, что недопустимо при их обучении. В конечном итоге, вместо дискретного преобразования Фурье было решено использовать алгоритм прореживания, описанный выше.

После преобразования алгоритмом прореживания из исходного временного ряда, представленного в лаговом пространстве произвольным числом значений, извлекается заранее определенное количество точек, скажем n+1. Далее, чтобы дополнительно повысить энтропию, мы берем не сами значения ряда, а разности между ними. Теперь каждый такой набор из n+1 значений ряда можно представить как точку в n-мерном пространстве. Для этого каждое из n разностей мы будем нормировать на единицу путем деления на dif_{max} - максимальную разность в примере, усредненную по всем примерам обучающей выборки. Так как dif_{max} – это средняя величина, и некоторые значения разности все-таки могут ее превышать, окончательный входной вектор целесообразно получать с помощью покомпонентного взятия функции гиперболического тангенса.

Пусть $(p_1, p_2, \dots p_{n+1})$ – n+1 компонента исходного ряда после процедуры прореживания.

Входные данные для нейронной сети представляются как

$$\left(\tanh\left(\frac{p_2-p_1}{dif_{max}}\right), \tanh\left(\frac{p_3-p_2}{dif_{max}}\right), \dots, \tanh\left(\frac{p_{n+1}-p_n}{dif_{max}}\right)\right),$$

где $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$.

Это дополнительно повышает энтропию, и что важно – гарантирует нормировку на единицу. Таким образом, мы можем добиться, чтобы каждый пример из выборки представлял собой точку в n-мерном пространстве для заранее определенного n, причем каждая компонента данного вектора будет нормирована на единицу и будет иметь высокую информативность.

Для дополнительного повышения информативности можно использовать и процедуру «выбеливания» входов, описанную в [1, 5]. В решаемой задаче данная процедура виделась излишней, так как каждый вход имеет одну и ту же природу, а соответственно, все значения будут представлять собой некоторый шар в n-мерном пространстве.

Для многих задач такого преобразования входной информации было бы более чем достаточно, и нейронная сеть показала бы хорошие результаты при прогнозировании, но в решаемой задаче данной процедуры оказалось недостаточно. Это было вызвано слабовыраженной зависимостью между входными и выходными данным, а также довольно большим объемом обучающей выборки – порядка 30 миллионов примеров.

Описание алгоритма обучения гибридной сети

Прежде чем обучать нейронную сеть обратного распространения, было решено разобрать все примеры на «кучки по похожести», то есть провести кластеризацию входного множества примеров.

Действительно, если просто учить персептронную сеть, то будет большая разность входных значений между абсолютно различными входными примерами при обучении сети, что в свою очередь не позволит тонко настроить сеть на нахождение даже небольших различий во входных примерах, которые подчас бывают весьма значимыми, так как зачастую у двух практически одинаковых примеров на входе прогнозируемые значения разительно отличаются.

С помощью обучения слоя Кохонена гибридной сети мы можем разбить n-мерное пространство входов на кластеры. Обычная процедура при самообучении слоя Кохонена выбирает ближайший кластер, а затем корректирует вес соответствующего ему нейрона на определенную величину, сдвигая тем самым центр кластера в сторону предъявленного обучающего примера. В этом случае каждый раз приходится сталкиваться с определенными проблемами.

Во-первых, необходимо определиться с первоначальным заданием центров кластеров. На практике зачастую применяют случайное распределение центров в пространстве, но примеры во входном пространстве распределены совсем неравномерно и какие-то нейроны будут обучаться чаще других. Более того, возможны и другие проблемные ситуации, когда, например, один из нейронов постоянно обучается, центр соответствующего кластера постоянно смещается и опять оказывается самым ближним к вновь предъявляемому обучающему примеру. Таким образом, этот кластер будет постоянно двигаться, в то время как остальные, имея случайные значения своих начальных центров, будут неподвижно стоять на месте. Непосредственно эта проблема может быть решена резким снижением скорости обучения.

В связи с этим напрашивается возможность вводить штраф за слишком большое количество примеров, на которых обучался нейрон. С одной стороны, это позволяет добиться желаемого – все нейроны учатся на приблизительно одинаковом количестве примеров и равномерно распределяются в пространстве входных значений. Но с другой стороны не позволяет учитывать то свойство, что сами по себе примеры в пространстве могут распределяться совершенно неравномерно и образовывать определенные «сгустки». На практике удалось найти компромисс, который позволяет с помощью введения коэффициента «справедливости» создать распределение по кластерам с одной стороны – достаточно равномерным, а с другой – учесть тот факт, что сами по себе примеры распределены в пространстве неравномерно.

Пусть дан некоторый входной вектор из n-мерного входного пространства. По обычной Евклидовой метрике находим расстояния до синтетических центров всех k кластеров, а затем домножаем его на коэффициент $c_i = (1 + \mu(\delta_i - \delta_{min})/\delta_{min})$, где $\mu\epsilon(0,1)$ – коэффициент, определяющий желательное отношение коли-

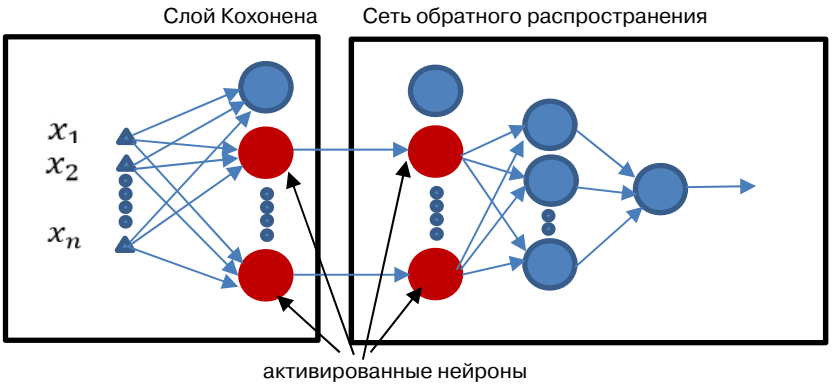

Рис. 2. Структура гибридной сети

чества элементов между кластерами, имеющими наименьшую и наибольшую мощность, δ_i – количество элементов, попавших в i-ый кластер, δ_{min} – количество элементов, попавших в кластер с наименьшей мощностью. Так, при $\mu = 0{,}2$, штраф, налагаемый на расстояние, будет существенным только в том случае, если $(\delta_i - \delta_{min})/\delta_{min}$)~5, то есть мощность данного кластера будет превосходить количество элементов в наименее мощном кластере приблизительно в 5 раз, а при $\mu = 1$ число элементов в кластерах будет приближаться к равномерному.

В итоге, алгоритм обучения слоя Кохонена в нашем случае выглядит следующим образом:

1) из значений исходного временного ряда извлекаются n компонент и нормируются на единицу, где n – количество входов обучаемой сети;

2) определяется самый близкий к данному входному примеру кластер с учетом коэффициента «справедливости»;

3) центр данного кластера сдвигается в сторону предъявленного примера на небольшое расстояние $\Delta \boldsymbol{w} = \alpha * (\boldsymbol{w} - \boldsymbol{x})$, где $\boldsymbol{w}$ – текущий синтетический центр кластера, $\boldsymbol{x}$ – текущий вектор входных значений, $\alpha = 0.01$ – скорость обучения.

При значениях, близких к единице, центр будет сильно сдвигаться в сторону предъявленного обучающего примера. Данный коэффициент желательно выбирать из соображений, чтобы кластер после обучения на эпохе «сохранил» память о самом первом примере. То есть желательно брать $\alpha < \frac{1}{\delta_{max}}$, где δ_{max} – число элементов попавших в самый объемный кластер.

Теперь опишем процедуру обучения персептронной подсети обратного распространения. Во входном слое персептронной подсети находится ровно такое же количество нейронов, что и в самообучающемся слое Кохонена. Входной пример исходного временного ряда преобразуется по описанной выше процедуре в n-мерный вектор, нормированный на единицу. Затем определяется некоторое небольшое число m самых близких к нему кластеров. И только соответствующие данным кластерам нейроны персептронного слоя получают данный пример на вход и обучаются на нем. По сути для этого выбирается небольшая подсеть и обучается на данном примере. Стоит сказать несколько слов о том, каким образом выбирается число m «компетентных» нейронов. С одной стороны, оно должно быть много меньше суммарного числа нейронов во всей сети, а с другой – быть достаточным, чтобы обеспечить приемлемую степень гладкости поверхности, которая и будет аппроксимировать исходную функцию. В итоге было решено остановиться на m~n, чтобы число одновременно активизируемых нейронов было сравнимо с размерностью входного пространства и отличалось не более чем на порядок. Конечно, хотелось бы иметь более точную и обоснованную оценку данного параметра, но она вполне удовлетворила практические потребности при решении исходной задачи.

Таким образом, за аппроксимацию искомой функции в некоторой окрестности данного входного примера будут отвечать только ближайшие нейроны, которые, так сказать, являются компетентными в данной окрестности пространства и «хорошо» разбираются в «по-

добных» примерах. При вычислении меры близости слоя Кохонена алгоритм пытается выяснить, какие из нейронов будут отвечать за данный пример. Остальные нейроны как бы говорят «мы ничего о таких примерах сказать не можем», и соответственно не активируются и не обучаются.

Описание алгоритма в общем случае

Если речь идет о временных рядах, то для выделения входных значений можно выбирать различные алгоритмы для предобработки данных либо подавать на вход сети необработанные значения самого ряда. Главное, как следует из специфики самих нейронных сетей, число входов всегда должно быть одинаково. Пусть мы имеем **n** входных значений нейронной сети.

Обучение разделяется на два этапа - обучение самоорганизующегося слоя и обучение сети обратного распространения.

Обучение самоорганизующегося слоя

1) Создаем самообучающийся слой, имеющий n входов и содержащий p≥n нейронов.

2) Определим скорость обучения $\alpha\epsilon(0,1)$.

3) Определим $\mu\epsilon(0,1)$ – коэффициент, задающий максимальное желательное соотношение между числом примеров в кластере.

4) Поочередно подаем входные значения из обучающей выборки.

5) Определяем ближайший центр с учетом коэффициента справедливости -

Находим порядковый номер **i** такой, что $L(w_i, x)$ минимально

$$L(w_i, x) = \\ = (1 + \mu(\delta_i - \delta_{min})/\delta_{min})\sqrt{\sum_{j=1}^{n}(w_i^j - x^j)^2}, \quad (1)$$

где δ_i – количество примеров в кластере i, а δ_{min} – число примеров в наименьшем кластере.

6) Сдвигаем в сторону приведенного примера i-ый кластер $\Delta \boldsymbol{w_i} = \alpha * (\boldsymbol{w_i} - \boldsymbol{x})$

Обучение персептронной сети обратного распространения

1) Определяем число одновременно активируемых нейронов k≥n

2) Задаем трехслойную полносвязную персептронную сеть p-m-1, где p – число нейронов во входном слое, равное числу узлов в самоорганизующемся слое, причем N>>p, где N – объем обучающей выборки, и p>>k, а m~n – число нейронов в скрытом слое.

3) Поочередно подаем на вход обучающие примеры.

4) Определяем k ближайших к нему кластеров по метрике L (1) , имеющих среди всех узлов самообучающегося слоя номера $(j_1, \dots, j_k)$

5) Значения обучающего примера подаются на вход нейронам входного слоя персептронной сети, имеющим те же порядковые номера $(j_1, \dots, j_k)$.

6) Далее персептронная подсеть обучается классическим алгоритмом обратного распространения ошибки.

В случае, если после обучения сети в течение нескольких эпох среднеквадратическая ошибка неудовлетворительна, необходимо увеличивать p и повторять всю процедуру обучения заново. В предельном случае, число нейронов входного слоя в сети вырастет до размера обучающей выборки, и за каждый входной пример будет отвечать только один активируемый нейрон. Очевидно, что ошибка после обучения будет равняться нулю. Главное, чтобы сеть при этом не потеряла способности к обобщению. Но это в первую очередь зависит от качества и репрезентативности обучающей выборки. На самом деле такого роста не потребуется, так как достаточно такого p, чтобы в окрестности одновременно активируемых k нейронов искомая функция вела себя равномерно.

Необходимость каждый раз полностью заново переобучать всю сеть кажется проблемой. Но на практике сеть обучается очень быстро в связи с небольшим количеством необходимых вычислений, и это не представляет особого труда.

При росте числа p, кластеры можно успешно делить на мета-кластеры, чтобы проводить меньшее число подсчетов метрики. Тогда вместо p подсчетов метрики L, можно получать $r\sqrt[r]{p}$ сравнений, где r – уровень иерархии первоначальных кластеров. Для мета-кластеров синтетический центр принимается как среднее между центрами входящих в них кластеров.

После чего обучающий пример сравнивается сначала с каждым синтетическим центром мета-кластера, а затем только с кластерами, нахо-

дящимися в ближайшем к обучающему примеру мета-кластеру. Данный способ позволяет избежать ненужных подсчетов метрики с кластерами, сильно непохожими на входной пример. Или, другими словами, лежащими от него далеко во входном пространстве. Так, при r=2 и p=16384, можно получить 256 подсчетов метрики L вместо 16384.

С помощью данного подхода удается добиться существенного преимущества в нескольких аспектах. Во-первых, мы имеем возможность строить нейронную сеть большого размера, причем добавление новых нейронов в сеть будет гарантированно увеличивать качество ее предсказания. Действительно, при росте p каждые k кластеров отвечают за меньшую область входного пространства, что позволяет на меньшей области лучше аппроксимировать исходную функцию. В пределе, с ростом числа нейронов, можно добиться, чтобы функция в данной области вела себя, например, равномерно. Такой случай не составит никакого труда даже для небольшой подсети, активизируемой в данной области. На практике успешно обучалась нейронная сеть, имеющая во входном слое $2^{14} = 16384$ нейрона. По сути, данный метод представляет собой одновременное обучение большого количества маленьких нейронный сетей, которые гладко склеиваются друг с другом на границах «ответственности» нейронов самообучающегося слоя.

В дополнение, сеть получает возможность аппроксимировать функции, чья поверхность напоминает «лунную», что особенно важно, так как заранее поведение искомой функции неизвестно и оно может иметь произвольных характер. В-третьих, такой подход существенно сокращает объем необходимых вычислений, что существенным образом ускоряет процесс обучения сети. Действительно, если обучать все десятки тысяч нейронов в многомерном входном пространстве, на миллионах входных примеров, то это может растянуться на годы, с учетом того, что для обучения сети обычно необходимо порядка тысячи эпох. На практике же встречаются задачи и с большими объемами обучающей выборки и набора входных параметров. В-четвертых, с помощью данного подхода можно конструктивно построить сеть, аппроксимирующую любую непрерывную гладкую функцию за конечное число шагов с любой наперед заданной точностью.

Пример работы алгоритма

Рассмотрим график простой функции $f(x, y) = \sin(10x) * \sin(10y)$, которая представляет собой поверхность с большим количеством локальных минимумов и максимумов и попробуем обучить классический трехслойный персептрон алгоритмом обратного распространения ошибки (Рис. 3). Зададим для сети следующую конфигурацию: 1024 нейронов в первом слое, 20 во втором и один нейрон в выходном слое. Случайным образом зададим весовые коэффициенты, возьмем скорость обучения равной $\lambda = 0{,}1$ и параметр инерции $\eta = 0{,}1$

Предъявляем сети 800000 примеров значений данной функции, выбирая значения x и y случайным образом из диапазона (-1,1), а затем останавливаем обучение. Время обучения составило больше двух суток на двухядерном процессоре Core i5-3210 M с тактовой частотой 2.5 GHz. Среднеквадратическая ошибка сети составляет ~0,4 после обучения. Другими словами можно констатировать, что сеть не обучилась ничему и среднеквадратическая ошибка непомерно велика.

Теперь предлагаем ту же задачу гибридной сети. Многослойный персептрон имеет ту же самую топологию, только теперь мы добавим к этой конструкции слой Кохонена, состоящий из 1024 самообучающихся нейронов. До обучения веса нейронов слоя Кохонена задаются случайными небольшими величинами. В нашем случае – от -0,2 до 0,2 (Рис. 4).

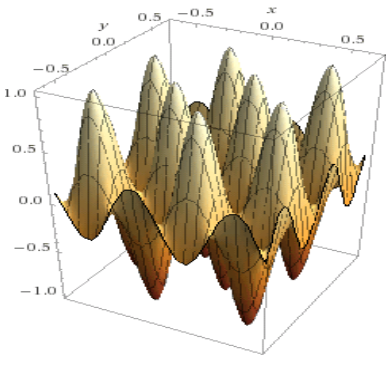

Рис. 3. sin(10x)*sin (10y)

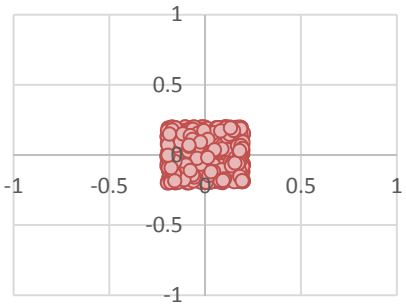

Рис. 4. Первоначальное распределение весов слоя Кохонена

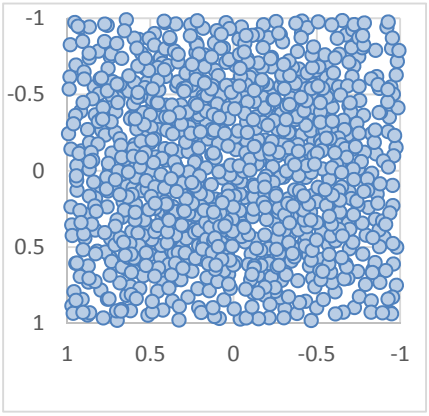

Рис. 5. Веса самообучающегося слоя

Сначала обучаем слой Кохонена на 10 000 случайных примеров. Положим $\mu = 0{,}5$. Время, затраченное на тренировку самообучающегося слоя - 39 секунд. Видно, что центры кластеров сместились в положительную область и равномерно заполнили все пространство (Рис. 5). Собственно, это нам и было необходимо.

Считаем, что одновременно в принятии решений в конкретном обучающем примере участвует 10 «экспертных» нейронов. Теперь обучаем многослойный персептрон. Предлагаем сети те же 800 000 примеров. После остановки обучения среднеквадратическая ошибка составляет около 0.05, что говорит о прекрасном качестве обучения. Теперь попробуем с помощью обученной нейронной сети построить поверхность, которую она аппроксимирует. Для этого построим 10 000 точек, и отобразим поверхность. Видно, что это и есть наша искомая функция.

Время обучения составило 1 час 13 минут. Это время можно было существенно сократить, так как ошибка упала довольно быстро - через 10 минут, а потом практически не изменялась (Рис. 6). Время на обучение сократилось в 50 раз, что связано с необходимостью на каждой итерации обучать только 10 нейронов вместо 1024. Причем для поиска необходимых десяти был введено еще одно дополнительное улучшение. Для того, чтобы определить ближайшие десять, необходимо посчитать Евклидову метрику от текущего примера до 1024 центров кластеров, чтобы выбрать ближайшие. Для ускорения 1024 кластера разбивались на 32 мета-кластера. Пример сначала сравнивался с синтетическим центром мета-кластера, а уже затем только с нейронами из слоя Кохонена из самого ближайшего мета-кластера. Что дает 64 подсчета метрики на каждой итерации обучения вместо 1024. При увеличении числа нейронов во входном слое можно строить иерархическую структуру мета-кластеров глубиной n, чтобы получать сравнений $n\sqrt[n]{m}$, где m – количество нейронов во входном слое. Так, при 16384

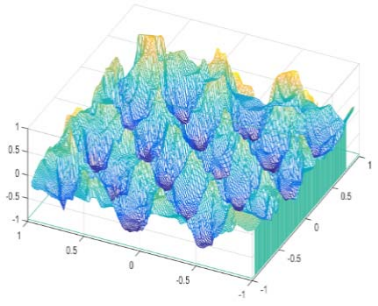

Рис. 6. Поверхность в процессе обучения через 10 мин.

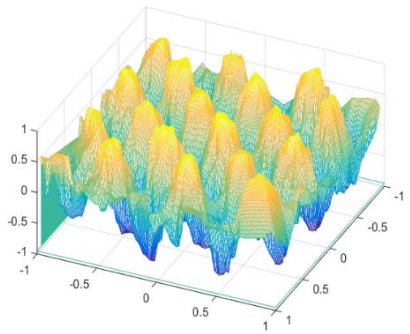

Рис. 7. Поверхность, реализуемая нейросетью, 25 мин.

нейронов и двухуровневой иерархии кластеров достаточно сделать 256 сравнений на каждом обучающем примере вместо 16384.

Заключение

Главным результатом работы стал созданный в процессе решения первоначальной задачи алгоритм обучения гибридной нейронной сети. В приведенном примере алгоритм продемонстрировал кратное улучшение среднеквадратической ошибки, что демонстрирует его успешную применимость даже на функциях, имеющих ярко выраженную неоднородную поверхность. Данный алгоритм также позволяет гарантированно улучшать точность предсказания за счет увеличения размера сети, что, в сущности, является научной новизной, а также на порядки ускорить работу алгоритмов обучения. Высокая эффективность предложенного гибридного подхода открывает широкие возможности для его применения при решении практических задач.

Литература

1. Ежов А., Шумский С. Нейрокомпьютинг и его применения в экономике и бизнесе. – Москва, 1998.
2. Осовский С. Нейронные сети для обработки информации. – М.: Финансы и статистика, 2002. – С. 252-256.
3. Афанасьев В.Н. Анализ временных рядов и прогнозирование. – М.: Финансы и статистика, 2010. – С. 224-241.
4. Dinh Nghia Do, Osowski S. Shape recognition using FFT preprocessing and neural network. Compel, 1998. – Vol. 17, No 5/6. -С. 658-666
5. Мицель А., Ефремова Е. Методы предобработки входных данных для системы прогнозирования финансовых временных рядов. Журнал "Доклады Томского государственного университета систем управления и радиоэлектроники" № 3 (11), 2005.

Нагорных Дмитрий Юрьевич Аспирант Института систем информатики им. А.П. Ершова СО РАН, г. Новосибирск, ст. преподаватель кафедры Общей Информатики в Новосибирском национальном исследовательском государственном университете (НГУ). Окончил Новосибирский государственный университет в 2007 году. Количество печатных работ: 1. Область научных интересов: искусственный интеллект, нейронные сети, большие данные, вычислительные системы. E-mail: nagor@academ.org

Teaching big hybrid neural networks for time series prediction

D.Y. Nagornykh

Abstract. The article describes hybrid approach in neural networks, used in prediction of time series, as well as the specific aspects of teaching hybrid neural networks consisting of Self-Organizing Maps (SOM) and Multilayer Perceptron (MLP). Also, paper contains the results, gained during the process of building and teaching the large hybrid neural network and the new algorithm of equitable teaching for self-organizing layer.

Keywords: neural nets, hybrid neural nets, self-organizing maps, time series prediction, function approximation.

References

1. Ejov A., Shumskiy S. Neuro-computing and it's adoption in economics and business. – Moscow, 1998.
2. Osovskiy S. Neural nets for data processing. – Moscow: Finance and statistics, 2002. – p. 252-256
3. Afanasiev V.N. Time series analysis and prediction. – Moscow: Finance and statistics, 2010. – p. 224-241.
4. Dinh Nghia Do, Osowski S. Shape recognition using FFT preprocessing and neural network. Compel, 1998. –Vol. 17, No 5/6. -C. 658-666
5. Micel A., Efremova E. Data preprocessing methods for financial time series prediction systems. "Proceedings of Tomsk State University of Control Systems and Radioelectronics" № 3 (11), 2005.

Dmitry Yurievich Nagornykh is a post-graduate student in A.P. Ershov Institute of Informatics Systems, Siberian Branch of the Russian Academy of Sciences, Senior Teacher in Novosibirsk State University, General Informatics Department. Master of science in Informational Technologies and Computational Systems. Graduated Novosibirsk State University in 2007. Number of papers: 1. Interest in scientific field: artificial intelligence, neural nets, big data, computational systems. E-mail: nagor@academ.org