УДК 004.032.26:519.6

ПРИМЕНЕНИЕ ЧИСЛЕННЫХ МЕТОДОВ В НЕЙРОННЫХ СЕТЯХ

Ляшко М.С., Гурина Т.Д., студенты

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Воронкова Е.В. – старший преподаватель

Аннотация. В работе рассматривается применение численных методов оптимизации для обучения искусственных нейронных сетей. Приводится математическая формализация модели нейрона и процесса прямого распространения сигнала. Подробно анализируются методы градиентного спуска, в том числе стохастические версии и адаптивные оптимизаторы (Momentum, Adam). Особое внимание уделяется функциям активации, их классификации и влиянию на сходимость численных алгоритмов. Установлена взаимосвязь между свойствами функций активации и эффективностью оптимизации.

Ключевые слова: нейронная сеть, численные методы, градиентный спуск, оптимизация, функция активации, функция потерь, метод импульса, Adam.

Введение. Современное развитие искусственного интеллекта и нейронных сетей напрямую зависит от того, как развивается вычислительная математика. Искусственные нейронные сети – это математические модели, которые созданы по образцу биологических нейронов, и по своей сути они являются вычислительными конструкциями. Их невозможно реализовать без численных методов. В отличие от классических моделей, где можно получить аналитическое решение, здесь всё строится на приближенных вычислениях – от задания начальных параметров до окончательной оптимизации.

Особая значимость этой темы вызвана усложнением архитектур нейросетей. Сейчас модели могут содержать миллиарды параметров, и обучить их эффективно можно только с помощью специализированных численных алгоритмов. Именно эти алгоритмы обеспечивают сходимость, устойчивость и вычислительную эффективность – те вещи, с которыми мы, так или иначе, сталкиваемся при изучении математики и программирования.

Численные методы на основе нейронных сетей относятся к классу методов с «непрямым» представлением исследуемого объекта. В отличие от классических подходов, объект здесь описывается не аналитически, а косвенно — через систему связанных узлов. Важной особенностью является то, что сами узлы, их входные параметры и выходные значения, как правило, не имеют физической интерпретации.

Предполагается, что модель функционирует по аналогии с человеческим мозгом. Теоретическим фундаментом методов искусственных нейронных сетей (ИНС) выступают стохастические модели, базирующиеся на численно-экспериментальном подходе.

Область знаний, занимающаяся разработкой и применением нейросетей, носит название Deep Learning (глубокое обучение). Данная сфера носит ярко выраженный междисциплинарный характер, интегрируя методы статистики, теории вероятностей, линейной алгебры, а также требуя компетенций в областях машинного обучения и науки о данных (Data Science).

Искусственная нейронная сеть представляет собой математическую модель (а также её программную или аппаратно-программную реализацию), построенную по принципу организации и функционирования биологических нейронных сетей. Само понятие возникло в ходе исследований процессов, протекающих в мозге, и попыток их моделирования. Исторически первой такой моделью принято считать нейронную сеть, предложенную У. Маккалоком и У. Питтсом [1].

Математическая модель нейрона и сети. Базовым элементом любой нейронной сети является искусственный нейрон — упрощенная математическая модель биологического прототипа. Он получает несколько входных сигналов, каждый из которых умножается на свой весовой коэффициент (вес). Веса — это и есть память сети; чем вес больше, тем значимее соответствующий вход. Далее все взвешенные сигналы суммируются, и к этой сумме прибавляется специальное число, называемое смещением (bias). Смещение позволяет сдвигать результат суммирования, делая нейрон более гибким. Наконец, полученная сумма проходит через нелинейное преобразование — функцию активации. Сигнал на выходе нейрона описывается следующим выражением:

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{1}$$

где x_i – входные сигналы; w_i – весовые коэффициенты; b – смещение; φ – функция активации; y – выходной сигнал.

Нейроны объединяются в слои, а слои — в сеть. Самой распространенной архитектурой является многослойный перцептрон, где сигнал распространяется строго в одном направлении — от входа к

выходу. Согласно универсальной аппроксимационной теореме, такая сеть с одним скрытым слоем способна аппроксимировать любую непрерывную функцию при достаточно большом количестве нейронов и подходящей функции активации. Это теоретическое обоснование объясняет широкую применимость многослойных сетей.

Процесс вычисления выходного сигнала такой сети (прямое распространение) можно представить как последовательное применение формул (2) – (4). Сначала входной вектор x подается на первый слой. Для каждого последующего слоя l мы умножаем матрицу весов $W^{(l)}$на выход предыдущего слоя $a^{(l-1)}$, прибавляем смещение $b^{(l)}$ и получаем вектор взвешенных сумм $z^{(l)}$. Затем применяем функцию активации $\varphi^{(l)}$ к каждому элементу этого вектора, получая выход текущего слоя $a^{(l)}$:

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}, \tag{2}$$

$$a^{(l)} = \varphi^{(l)}z^{(l)}2, \tag{3}$$

Повторяя эту процедуру для всех слоев, на последнем слое L мы получаем итоговый прогноз сети $\hat{y}$:

$$\hat{y} = a^{(L)}. \tag{4}$$

Обучение как оптимизация. Чтобы сеть давала правильные ответы, ее нужно обучить. Для этого нам нужен критерий, который покажет, насколько хорошо или плохо сеть работает в данный момент. Таким критерием является функция потерь $J(\theta)$. Она вычисляет среднюю ошибку между ответами сети $\hat{y}$ и правильными ответами y на обучающих примерах. Для задач регрессии часто используется среднеквадратичная ошибка:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{n}(y - \hat{y}_i)^2, \tag{5}$$

где θ – совокупность всех параметров сети (веса и смещения); N – размер обучающей выборки. Для задач классификации более предпочтительна функция кросс-энтропии, которая лучше отражает вероятностную природу выходов.

Таким образом, задача обучения превращается в классическую задачу оптимизации: найти такой набор параметров θ^*, при котором функция потерь $J(\theta)$ принимает минимальное значение. Представить себе эту функцию в пространстве с тысячами или миллионами измерений невозможно, но мы можем изучать ее свойства локально — в той точке, где находимся в данный момент. Важно отметить, что функция потерь нейронной сети, как правило, невыпуклая и содержит множество локальных минимумов и седловых точек. Однако на практике многие из этих минимумов дают схожее качество, и задача сводится к нахождению достаточно хорошего решения.

Численные методы оптимизации. Для нахождения минимума функции $J(\theta)$ мы используем итеративные численные методы. Самый фундаментальный из них — градиентный спуск. Его идея проста и основана на математическом анализе: если мы вычислим градиент функции в текущей точке (вектор частных производных), то он укажет направление наискорейшего роста функции. Следовательно, чтобы уменьшить значение функции, нужно сделать шаг в противоположном направлении. Правило обновления параметров имеет вид:

$$\theta_{t+1} = \theta_t - \eta\nabla J(\theta_t), \tag{6}$$

где η – скорость обучения (гиперпараметр, определяющий величину шага); $\nabla J(\theta_t)$ – градиент функции потерь.

Выбор скорости обучения — это искусство. Если шаг слишком маленький, мы будем идти к минимуму очень долго. Если слишком большой — можем "перепрыгнуть" минимум и начать расходиться, уходя в бесконечность.

Разновидности градиентного спуска. На практике градиент $J(\theta)$ вычисляется не по всей выборке сразу (это слишком дорого), а по её небольшим случайным порциям — мини-пакетам. Это мини-пакетный градиентный спуск. Он вносит некоторый шум в процесс, но этот шум даже полезен: он помогает алгоритму "выскакивать" из неглубоких локальных минимумов и седловых точек, где обычный градиентный спуск мог бы застрять [2]. Размер мини-пакета является гиперпараметром: маленькие пакеты дают более шумную оценку градиента, что может улучшить обобщающую способность, но замедляет сходимость; большие пакеты дают более точную оценку, но требуют больше памяти и вычислений.

Кроме мини-пакетного, существует стохастический градиентный спуск (SGD), где градиент вычисляется по одному случайному примеру, и пакетный градиентный спуск, где используется вся

выборка. В современных библиотеках глубокого обучения по умолчанию применяется мини-пакетный вариант [3].

Метод импульса. Однако даже мини-пакетный метод не идеален. Представьте себе овраг: функция быстро убывает в одном направлении (крутой склон) и медленно в другом (дно оврага). Градиентный спуск будет "болтаться" от стенки к стенке, очень медленно продвигаясь вдоль дна. Для решения этой проблемы были разработаны более продвинутые численные методы.

Одним из таких методов является метод импульса (Momentum). Его можно представить как тяжелый шар, который катится по поверхности функции потерь. Шар накапливает инерцию: если он долго движется в одном направлении, его скорость возрастает, а резкие колебания (например, удары о стенки оврага) сглаживаются. Математически это выражается введением дополнительной переменной v — вектора скорости:

$$v_{t+1} = \gamma v_t + \eta \nabla J(\theta_t), \tag{7}$$

$$\theta_{t+1} = \theta_t - v_{t+1}, \tag{8}$$

где γ – коэффициент затухания импульса (обычно принимается равным 0.9). Он определяет, как быстро "забываются" старые градиенты.

Методы адаптивной скорости обучения. Дальнейшим развитием идеи адаптивности стали методы, которые подбирают индивидуальную скорость обучения для каждого параметра. Один из первых таких методов — Adagrad, который накапливает сумму квадратов градиентов и уменьшает шаг для параметров с большими градиентами. Однако Adagrad имеет проблему: со временем шаг становится слишком маленьким из-за монотонного роста знаменателя. На смену ему пришел RMSProp, использующий экспоненциально затухающее среднее квадратов градиентов, что позволяет адаптироваться к нестационарным ландшафтам.

Метод Adam (Adaptive Moment Estimation) [3] объединяет идеи импульса и RMSProp. Он адаптирует скорость обучения для каждого параметра индивидуально, используя две скользящие средние: среднее значение градиентов (первый момент m), которое показывает общее направление, и среднее значение квадратов градиентов (второй момент v), которое показывает разброс или "активность" градиента. Обновление параметров в методе Adam производится по следующему алгоритму:

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla J(\theta_t), \tag{9}$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)(\nabla J(\theta_t))^2, \tag{10}$$

$$\theta_{t+1} = \theta_t - \eta \frac{m_{t+1}}{\sqrt{v_{t+1}} + \varepsilon}, \tag{11}$$

где β_1, β_2 – коэффициенты затухания моментов (обычно 0.9 и 0.999); ε – малая константа для предотвращения деления на ноль. Adam стал одним из стандартов индустрии, так как он устойчив к выбору гиперпараметров и хорошо работает на огромном разнообразии задач [4].

Методы второго порядка. Наряду с методами первого порядка (использующими только градиент), существуют методы второго порядка, которые учитывают кривизну функции потерь через матрицу Гессе. Классический метод Ньютона обновляет параметры по формуле: $\theta_{t+1} = \theta_t - \eta[\nabla^2 J(\theta_t)]^{-1}\nabla J(\theta_t)$. Однако вычисление и обращение гессиана для миллионов параметров практически невозможно из-за колоссальной вычислительной сложности $(O(n^3))$. Поэтому в глубоком обучении методы второго порядка используются редко, хотя существуют их аппроксимации, например, метод L-BFGS, который иногда применяется для задач небольшого размера. [3]

Роль функций активации. Классические функции активации. Почему же все эти методы работают? Они опираются на градиент, а градиент вычисляется с помощью цепного правила и проходит через каждый слой сети, включая функции активации. Производная функции активации — это "коэффициент пропускания" для градиента на данном нейроне. Именно здесь кроется ключ к пониманию многих проблем обучения.

Функция активации вносит в модель нелинейность. Без неё многослойная сеть оставалась бы просто комбинацией линейных преобразований и не могла бы обучаться сложным зависимостям [5]. Исторически сложилось несколько классов функций активации, каждый из которых имеет свои особенности.

Ступенчатая функция (пороговая) (рисунок 1). Это исторически первая функция активации, использовавшаяся в модели Маккалока–Питтса [1]. Она определяется как:

$$\varphi(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases} \quad (12)$$

Её главное преимущество — простота и бинарность выхода, что удобно для логических схем. Однако она недифференцируема (производная равна нулю всюду, кроме точки разрыва), что делает её непригодной для обучения градиентными методами. Кроме того, она даёт только жёсткое решение «всё или ничего», что ограничивает выразительные способности сети.

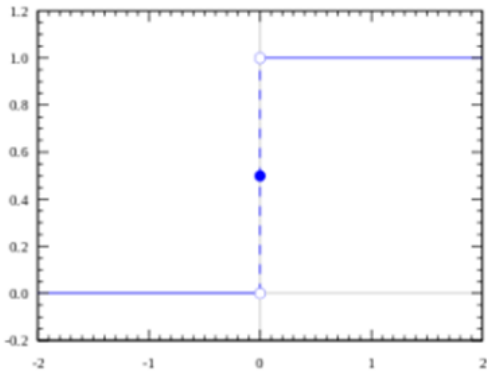

Рисунок 1 – Ступенчатая функция

Ступенчатая функция совершенно не подходит для сложных задач и при использовании нейросетей с большим количеством нейронов в одном слое, т.к. может активировать несколько нейронов сразу. Как следствие этому, сеть будет работать некорректно из-за конфликта данных.

Линейная функция (рисунок 2). В простейшем случае нейрон может иметь линейную активацию:

$$\varphi(z) = z. \quad (13)$$

Производная такой функции постоянна и равна единице, что обеспечивает стабильное распространение градиента. Однако ключевой недостаток — композиция линейных преобразований остаётся линейной. В отличие от ступенчатой функции линейная функция позволяет получать спектр значений, а не только один ответ из двух.

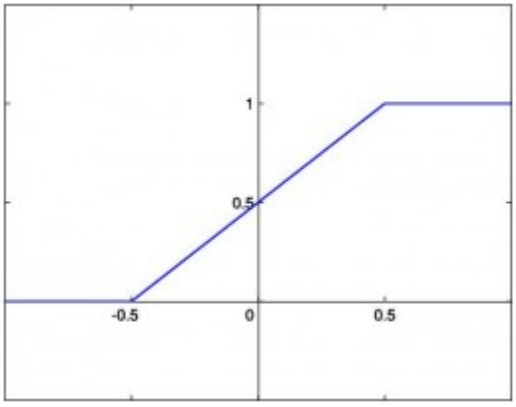

Рисунок 2 – Линейная функция

Многослойная сеть с линейными активациями эквивалентна однослойной и не может аппроксимировать нелинейные зависимости. Поэтому линейные функции иногда используют только в выходном слое для задач регрессии, где требуется предсказание действительных чисел.

Сигмоидальная функция (логистическая) (рисунок 3). Это классическая гладкая функция, которая сжимает любой вход в интервал от 0 до 1:

$$\varphi(\mathrm{z}) = \frac{1}{1+e^{-z}}. \quad (14)$$

Она хороша тем, что её выход можно интерпретировать как вероятность. Но у неё есть фатальный недостаток: при больших положительных или отрицательных значениях z функция становится очень

пологой, и её производная стремится к нулю ($\varphi'(\mathrm{z}) = \varphi(\mathrm{z})(1 - \varphi(\mathrm{z}))$). Это означает, что градиент, проходя через такой "насыщенный" нейрон, становится исчезающе малым. В глубоких сетях, где таких слоёв много, градиент, дойдя до первых слоёв, практически полностью затухает. Возникает проблема "затухающего" градиента и сеть перестаёт обучаться.

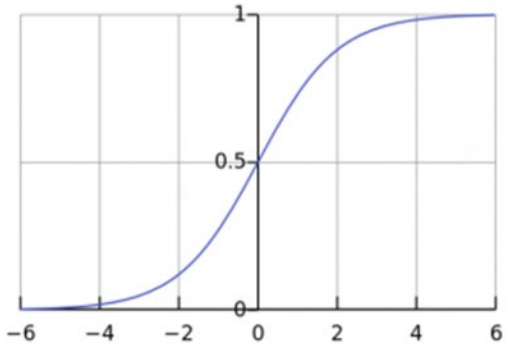

Рисунок 3 – Сигмоидальная функция

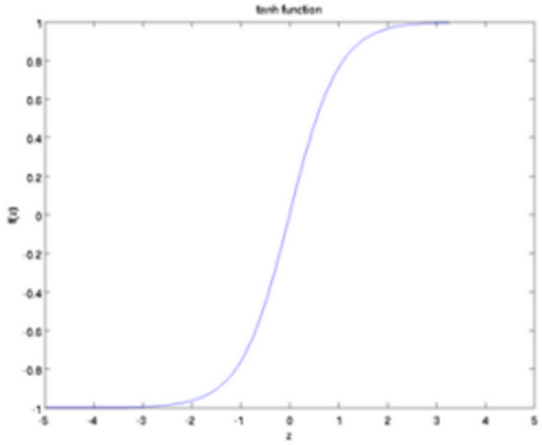

Рисунок 4 – Гиперболический тангенс

Функция гиперболического тангенса (tgh) (рисунок 4). Является сдвинутой и масштабированной версией сигмоиды, её выход лежит в диапазоне от -1 до 1:

$$\varphi(z) = tgh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \quad (15)$$

Она часто работает лучше сигмоиды, потому что её выход центрирован относительно нуля. Это облегчает оптимизацию, так как градиенты следующих слоев получают сигналы разных знаков. Однако проблема насыщения и затухания градиента для $tanh$ также актуальна, хоть и в чуть меньшей степени [6].

ReLU и его модификации. Прорывом стало появление функции ReLU (Rectified Linear Unit). Это удивительно простая функция:

$$\varphi(z) = max(0, z). \quad (16)$$

Она оставляет положительные числа без изменения, а отрицательные превращает в ноль. Её производная равна 1 для положительных аргументов и 0 для отрицательных. Благодаря тому, что для положительных входов производная всегда равна 1, градиент проходит через активный нейрон без затухания. Это позволило обучать очень глубокие сети, которые раньше было невозможно обучить. Кроме того, ReLU вычислительно эффективна, так как требует лишь операции сравнения.

Однако у ReLU есть своя проблема — "умирающий ReLU". Если в процессе обучения веса нейрона сложатся так, что на его вход постоянно приходят отрицательные числа, то его выход всегда будет равен нулю, а градиент — нулю. Нейрон "умирает" и больше никогда не обучается. Для борьбы с этим были придуманы модификации, например, Leaky ReLU, который для отрицательных аргументов даёт маленький отрицательный наклон (например, 0,01 z), позволяя градиенту хоть как-то проходить и, возможно, "оживить" нейрон:

$$\varphi(z) = max(az, z) \text{ с малым } \alpha > 0. \quad (17)$$

Другие варианты: PReLU, где α обучается в процессе оптимизации; ELU (Exponential Linear Unit), которая имеет плавный изгиб для отрицательных значений, что улучшает свойства градиента и ускоряет сходимость; SELU (Scaled ELU), которая обеспечивает самопормализацию сети — активация автоматически приводит выходы слоя к нулевому среднему и единичной дисперсии. Выбор конкретной функции активации зависит от задачи и архитектуры, но ReLU и его варианты остаются наиболее популярными для скрытых слоев.

Проблема взрывающегося градиента. Помимо затухания, существует и обратная проблема — взрывающийся градиент, когда градиенты становятся слишком большими и дестабилизируют обучение. Это может происходить при неудачной инициализации весов или слишком большой глубине сети. Для борьбы с взрывающимся градиентом применяют клипинг градиентов (ограничение его нормы) или

специальные методы инициализации, такие как инициализация Ксавье (для сигмоиды и tgh) или инициализация Хе (для ReLU).

Заключение. В ходе работы мы рассмотрели, что обучение нейронных сетей — это, по сути, прикладная задача численной оптимизации. Мы проследили эволюцию методов: от простого градиентного спуска до адаптивных алгоритмов типа Adam, которые учитывают историю градиентов и адаптируют шаг для каждого параметра. Мы также увидели, что эффективность этих методов неразрывно связана с выбором функций активации. Функции активации определяют "ландшафт" функции потерь и то, насколько свободно градиенты могут распространяться по сети. Понимание этой взаимосвязи необходимо любому специалисту, который хочет осознанно проектировать архитектуры и выбирать алгоритмы обучения, добиваясь наилучших результатов. Дальнейшее развитие численных методов, вероятно, будет связано с созданием ещё более адаптивных алгоритмов, учитывающих структуру задачи, а также с гибридными подходами, использующими информацию второго порядка.

Список использованных источников:

1. *McCulloch, W.S. A logical calculus of the ideas immanent in nervous activity / W.S. McCulloch, W. Pitts // The bulletin of mathematical biophysics.– 1943.– Vol. 5, № 4.– P. 115–133.*

2. Хайкин, С. Нейронные сети: полный курс / С. Хайкин. – 2-е изд. – М.: Вильямс, 2006. – 1104 с.

3. Горбань, А.Н. Обучение нейронных сетей / А.Н. Горбань. – М.: ПараГраф, 1990. – 160 с.

4. Численные методы оптимизации в задачах машинного обучения [Электронный ресурс] / Habr.com. – Режим доступа: https://habr.com/ru/articles/813221/. – Дата доступа: 15.03.2026.

5. Алексейчук, А.С. Введение в нейронные сети: модели, методы и программные средства / А.С. Алексейчук. – М.: МАИ, 2023. – 105 с.

6. Романов, П.С. Системы искусственного интеллекта. Моделирование нейронных сетей в системе MATLAB / П.С. Романов, И.П. Романова. – 4-е изд. – СПб. : Лань, 2024. – 140 с.

UDC 004.032.26:519.6

APPLICATION OF NUMERICAL METHODS IN NEURAL NETWORKS

Lyashko M.S., Gurina T.D., students

Belarusian State University of Informatics and Radioelectronics
Minsk, Republic of Belarus

Voronkova E.V. – Senior Lecturer

Annotation. The paper considers the application of numerical optimization methods for training artificial neural networks. The mathematical formalization of the neuron model and the process of direct signal propagation is given. Methods of gradient descent, including stochastic versions and adaptive optimizers (Momentum, Adam), are analyzed in detail. Special attention is paid to activation functions, their classification, and the effect on the convergence of numerical algorithms. The relationship between the properties of activation functions and optimization efficiency has been established.

Keywords: neural network, numerical methods, gradient descent, optimization, activation function, loss function, momentum method, Adam.