Шпаргалка по нейронным сетям / Концепции

Cheatsheet (XeLaTeX)

Краткий справочник

April 2, 2025

Contents

1 VI. Введение в Нейронные Сети (NN)

Цель раздела

Понять базовые строительные блоки нейронных сетей, их терминологию и основной принцип обучения. Не бояться слов «перцептрон», «backpropagation», «оптимизатор». Это основа для понимания более сложных моделей.

1.1 VI.A Базовые Структуры: Перцептрон и MLP

Перцептрон: Простейший Нейрон

Перцептрон (Perceptron) — это самый базовый, *исторический* элемент нейросети, математическая модель нейрона.

- Он принимает несколько входов (чисел), умножает каждый на свой **вес** (weight).
- Суммирует взвешенные входы и добавляет **смещение** (bias).
- Пропускает результат через **функцию активации** (в классическом перцептроне — пороговую, «ступеньку»).

Аналогия: Представьте себе простого решателя, который взвешивает разные «за» и «против» (входы с весами), сравнивает с неким порогом (смещение) и выносит вердикт «да» или «нет» (выход активации).

$$y = f(\sum_i w_i x_i + b)$$

Где x_i - входы, w_i - веса, b - смещение, f - функция активации, y - выход.

Многослойный Перцептрон (MLP)

Многослойный Перцептрон (Multi-Layer Perceptron, MLP) — это уже полноценная нейронная сеть, состоящая из слоев перцептронов (или нейронов с другими, обычно нелинейными, функциями активации).

- **Входной слой** (Input Layer): Получает исходные данные (признаки). Нейронов столько, сколько признаков.
- **Скрытые слои** (Hidden Layers): Один или несколько слоев между входным и выходным. Здесь происходит основная «магия» обработки информации. Количество слоев и нейронов в них — это гиперпараметры.
- **Выходной слой** (Output Layer): Выдает конечный результат (предсказание класса, число и т.д.). Количество нейронов зависит от задачи (например, 1 для регрессии, N для N-классовой классификации).
- Нейроны каждого слоя соединены с нейронами следующего слоя (**полносвязная сеть**, Fully Connected).

Аналогия: Это как комитет экспертов. Первая группа (входной слой) просто получает факты. Затем они передают свои выводы следующей группе (скрытый слой), которая их анализирует глубже. Те, в свою очередь, передают дальше... Пока финальная группа (выходной слой) не вынесет итоговое решение.

1.2 VI.B Функции Активации: Придаем Нелинейность

Зачем нужны функции активации?

Ключевая роль функций активации — **внести нелинейность** в модель.

- Без нелинейных функций активации вся нейросеть (даже с многими слоями) была бы эквивалентна одному линейному преобразованию (просто большой линейной регрессии).
- Нелинейность позволяет сети изучать сложные, нелинейные зависимости в данных.

Аналогия: Если у вас есть только палки (линейные функции), как бы вы их ни складывали, вы все равно получите палку (другую линейную функцию). Чтобы построить что-то сложное, вам нужны «изгибы» и «углы» (нелинейные функции).

Популярные Функции Активации

- **ReLU (Rectified Linear Unit):**

$$f(x) = \max(0, x)$$

Плюсы: Вычислительно простая, помогает бороться с проблемой **затухания градиентов** (vanishing gradients) для положительных значений. Самая популярная в скрытых слоях. **Минусы:** «Умирающие ReLU» (нейроны, которые всегда выдают 0 и не активируются). **Аналогия:** Простой переключатель — пропускает сигнал, если он положительный, и блокирует, если отрицательный.

- **Sigmoid (Сигмоида):**

$$f(x) = \frac{1}{1 + e^{-x}}$$

Плюсы: Сжимает выходные значения в диапазон [0, 1], что удобно для интерпретации как вероятности (например, в выходном слое бинарной классификации). **Минусы:** Сильно страдает от затухания градиентов при значениях x, далеких от 0. Выход не центрирован около нуля. **Аналогия:** Регулятор, который плавно переводит любой входной сигнал в значение между 0 и 1.

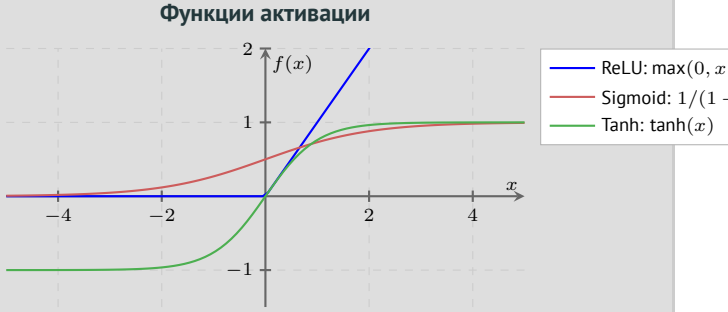
- **Tanh (Гиперболический тангенс):**

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Плюсы: Выход центрирован около нуля (диапазон [-1, 1]), что может ускорять сходимость. **Минусы:** Также страдает от затухания градиентов, хотя и меньше, чем Sigmoid. **Аналогия:** Похож на Sigmoid, но выдает значения от -1 до 1.

Другие важные: Leaky ReLU, ELU.

Графики функций активации (ReLU, Сигмоида, Tanh)



1.3 VI.C Backpropagation: Как Сеть Учится на Ошибках

Метод Обратного Распространения Ошибки (Backpropagation)

Backpropagation — это основной алгоритм для обучения нейронных сетей. Он позволяет эффективно вычислить, как нужно изменить каждый вес в сети, чтобы уменьшить общую ошибку предсказания.
Общая идея (2 этапа):

1. **Прямой проход (Forward Pass):** Данные проходят через сеть от входа к выходу. Сеть делает предсказание. Вычисляется **функция потерь** (Loss Function), которая показывает, насколько предсказание отличается от истинного значения.
2. **Обратный проход (Backward Pass):**
   - Ошибка «распространяется» обратно по сети, от выхода ко входу.
   - На каждом шаге вычисляется **градиент** функции потерь по весам данного слоя (используя **цепное правило / chain rule** из матанализа). Градиент показывает направление и «силу» необходимого изменения веса для уменьшения ошибки.
   - Веса обновляются с помощью **оптимизатора** (например, SGD, Adam) на основе вычисленных градиентов.

Аналогия: Представьте «разбор полетов» после командного задания. Сначала команда выполняет задание (прямой проход), затем оценивается результат и ошибка (функция потерь). Потом руководитель (алгоритм) идет «обратно» по цепочке выполнения, выясняя вклад каждого участника (веса) в общую ошибку (градиент), и дает указания, как каждому исправиться (обновление весов), чтобы в следующий раз результат был лучше.

1.4 VI.D Оптимизаторы: Навигаторы в Пространстве Ошибок

Алгоритмы Оптимизации

Оптимизатор — это алгоритм, который использует градиенты, посчитанные с помощью Backpropagation, чтобы обновить веса нейросети с целью минимизации функции потерь.
Популярные оптимизаторы:

- **SGD (Stochastic Gradient Descent)**: Стохастический Градиентный Спуск.
  - Базовый и простой.
  - Обновляет веса на основе градиента, вычисленного по небольшой случайной выборке (**батчу / batch**) данных.
  - **Аналогия:** Спуск с горы в тумане. Вы делаете небольшой шаг в направлении уклона прямо под вашими ногами. Быстро, но можно застрять в яме (локальный минимум) или «скакать» по склону из-за шумных оценок градиента.
  - Часто используется с **Momentum** (учет предыдущего шага, как бы «инерция») или **Nesterov Momentum**.
- **Adam (Adaptive Moment Estimation)**: Адаптивная Оценка Моментов.
  - Более продвинутый и часто используемый по умолчанию.
  - Адаптирует **скорость обучения (learning rate)** для каждого веса индивидуально. *Часто хорошо работает «из коробки» и является хорошей отправной точкой.*
  - Учитывает и «инерцию» (как Momentum), и масштабирование градиентов (как RMSprop).
  - **Аналогия:** Продвинутый альпинист с навигатором. Он не просто идет вниз, а учитывает свою прошлую скорость, крутизну склона в разных направлениях и подстраивает длину шага для каждого направления, чтобы спускаться эффективнее и не застревать.

Другие: Adagrad, RMSprop, Adadelta. Выбор оптимизатора и его параметров (особенно learning rate) — важные гиперпараметры.

1.5 VI.E Dropout и BatchNorm: Стабилизация и Регуляризация

Техники для Улучшения Обучения

Эти техники помогают сделать обучение более стабильным, быстрым и предотвратить переобучение.

- **Dropout (Прореживание)**:
  - **Что делает:** Во время *обучения* на каждом шаге случайным образом «выключает» (обнуляет выход) часть нейронов скрытого слоя с некоторой вероятностью p. На этапе *тестирования/предсказания* все нейроны используются, но их выходы масштабируются.
  - **Зачем нужен:** Это мощный метод **регуляризации** для борьбы с **переобучением (overfitting)**. Он заставляет сеть не полагаться слишком сильно на отдельные нейроны, а учить более робастные признаки.
  - **Аналогия:** Тренировка баскетбольной команды, где на каждой тренировке случайные игроки сидят на скамейке запасных. Это заставляет оставшихся игроков лучше взаимодействовать и развивать разные тактики, не надеясь только на одного «звездного» игрока.
- **Batch Normalization (BatchNorm, Пакетная Нормализация)**:
  - **Что делает:** Нормализует активации (*обычно до применения функции активации*, но есть варианты и после) внутри сети по каждому **батчу** данных во время обучения (приводит к нулевому среднему и единичной дисперсии), а затем сдвигает и масштабирует с помощью обучаемых параметров γ и β.
  - **Зачем нужен:**
    * Стабилизирует и ускоряет процесс обучения (борется с проблемой **internal covariate shift** — изменением распределения входов слоев во время обучения).
    * Позволяет использовать более высокие скорости обучения (learning rates).
    * Обладает легким регуляризующим эффектом.
  - **Аналогия:** Представьте конвейер сборки сложного прибора. BatchNorm — это как станции калибровки и стандартизации деталей на разных этапах конвейера (*перед* следующим этапом сборки). Это гарантирует, что на следующий этап детали поступают в ожидаемом виде, что делает весь процесс сборки более стабильным и быстрым.

1.6 VI.F Специализированные Архитектуры: CNN и RNN

За Пределами MLP: Сети для Конкретных Данных

MLP хороши для табличных данных, но для данных со специфической структурой существуют более эффективные архитектуры:

- **CNN (Convolutional Neural Networks): Сверточные Нейронные Сети**
  - **Для чего:** Идеальны для обработки данных с **пространственной структурой**, таких как **изображения** и видео.
  - **Ключевые идеи:** Используют **сверточные слои** (convolutional layers) для обнаружения локальных признаков (грани, текстуры) с помощью обучаемых фильтров и **пулинг слои** (pooling layers) для уменьшения размерности и инвариантности к небольшим сдвигам.
  - **Аналогия:** Поиск узоров на картинке с помощью маленькой «лупы» (фильтра), которая скользит по всему изображению и реагирует на определенные формы или текстуры.
- **RNN (Recurrent Neural Networks): Рекуррентные Нейронные Сети**
  - **Для чего:** Созданы для обработки **последовательных данных**, где важен порядок элементов, например, **текст**, **временные ряды**, речь.
  - **Ключевые идеи:** Имеют «память» — внутреннее состояние, которое обновляется на каждом шаге последовательности и влияет на обработку следующего элемента.
  - **Аналогия:** Чтение книги. Чтобы понять смысл текущего предложения, вам нужно помнить, о чем говорилось в предыдущих. RNN пытается делать то же самое с последовательностями данных.
  - **Важно:** Классические RNN имеют проблемы с долгими зависимостями (затухание/взрыв градиента). На практике почти всегда используют их продвинутые варианты: **LSTM (Long Short-Term Memory)** и **GRU (Gated Recurrent Unit)**, которые используют специальные *гейты* (gates) для контроля потока информации и памяти.