Шпаргалка по нейронным сетям / Концепции

Cheatsheet (XeLaTeX)

Краткий справочник

April 6, 2025

Contents

1 VI. Введение в Нейронные Сети (NN)

Цель раздела

Понять базовые компоненты нейронных сетей (нейроны, слои, функции активации), основной механизм обучения (Backpropagation) и методы его улучшения (оптимизаторы, регуляризация). Заложить основу для понимания сверточных и рекуррентных сетей.

1.1 VI.A Базовые Структуры: Нейроны и Слои

Искусственный Нейрон: Вычислительный Элемент

Что это: Математическая модель, имитирующая работу биологического нейрона. **Как работает:**

1. Принимает входы (x_i).
2. Умножает каждый вход на его **вес** (w_i) $\rightarrow w_i x_i$.
3. Суммирует взвешенные входы $\rightarrow z_{sum} = \sum_i w_i x_i$.
4. Добавляет **смещение** (b) $\rightarrow z = z_{sum} + b$.
5. Пропускает результат z через **функцию активации** $f(\cdot) \rightarrow y = f(z)$ (выход нейрона).

Обучаемые параметры: Веса w_i и смещение b.

Многослойный Перцептрон (MLP): Архитектура

Что это: Классическая нейросеть из нескольких слоев нейронов. **Слои:**

- **Входной (Input):** Принимает признаки X. Не содержит вычислительных нейронов.
- **Скрытые (Hidden):** Один или более. Здесь происходит основная обработка, извлечение паттернов.
- **Выходной (Output):** Формирует результат. Структура зависит от задачи (1 нейрон/линейная для регрессии, 1 нейрон/сигмоида для бинарной клас., N нейронов/Softmax для многоклассовой).

Связи: Обычно **полносвязные** (Dense) — каждый нейрон слоя связан с каждым нейроном следующего.

1.2 VI.B Функции Активации: Нелинейность и Свойства

Зачем нужна Нелинейность?

Без нелинейных функций активации в скрытых слоях вся сеть была бы эквивалентна простой линейной модели. Нелинейность позволяет изучать сложные зависимости.

ReLU (Rectified Linear Unit)

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

Свойства: Вычислительно проста. Не насыщается для $x > 0$ (помогает с затуханием градиента). **Недостаток:** "Умирающие ReLU" (нейрон перестает активироваться и обучаться, если x всегда ≤ 0). **Использование:** Стандартный выбор для скрытых слоев.

Leaky ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \quad (\alpha \approx 0.01 - 0.2)$$

Свойства: Решает проблему "умирающих ReLU", давая малый ненулевой градиент при $x \leq 0$.

ELU (Exponential Linear Unit)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \quad (\alpha > 0)$$

Свойства: Похожа на Leaky ReLU, но использует экспоненту. Может давать лучшие результаты, чем ReLU/Leaky ReLU. Выход для $x < 0$ отрицательный.

Sigmoid (Сигмоида)

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

Свойства: Выход [0, 1], удобен для вероятностей. **Недостатки:** Затухание градиентов. Выход не центрирован около нуля. **Использование:** Выходной слой бинарной классификации. Редко в скрытых слоях современных сетей.

Tanh (Гиперболический тангенс)

$$f(x) = \tanh(x)$$

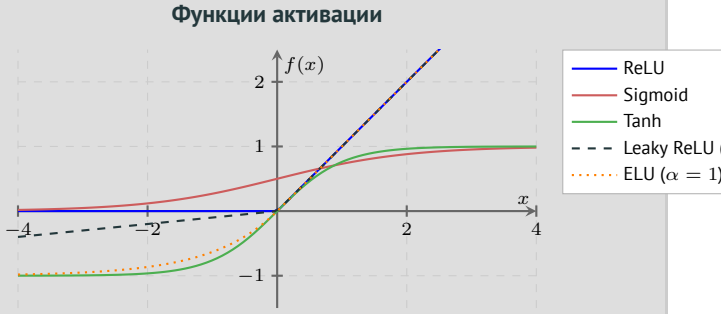
Свойства: Выход [-1, 1], центрирован около нуля (лучше Sigmoid для скрытых слоев). **Недостатки:** Затухание градиентов (хотя меньше, чем у Sigmoid). **Использование:** Иногда в скрытых слоях, часто в RNN/LSTM.

Softmax

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Свойства: Преобразует вектор логитов в распределение вероятностей (сумма=1). **Использование:** **Только** в выходном слое для **многоклассовой классификации**.

Графики популярных функций активации



Проблема Затухания/Взрыва Градиентов

Проблема: При обучении глубоких сетей градиенты могут стать исчезающе малыми (**затухание**) или аномально большими (**взрыв**). **Последствия:** Замедление или остановка обучения (затухание), нестабильность (взрыв). **Решения:** Выбор активаций (ReLU и др.), правильная инициализация весов, Batch Normalization, обрезание градиентов (для взрыва).

1.3 VI.C Backpropagation: Как Сеть Учится

Backpropagation: Ключевой Алгоритм Обучения

Цель: Эффективно вычислить **градиенты** функции потерь J по всем обучаемым параметрам (w, b). Градиент $\partial J / \partial w$ показывает, как сильно изменение веса w повлияет на итоговую ошибку J.

Этап 1: Прямой Проход (Forward Pass)

Что происходит: Данные X проходят через сеть слой за слоем от входа к выходу. На каждом слое вычисляются взвешенные суммы (z) и активации (a). Получаем итоговые предсказания $\hat{y}$. **Результат:** Предсказания $\hat{y}$ и значения активаций a на всех слоях (они понадобятся для обратного прохода). **Затем:** Вычисляется **функция потерь** $J(\hat{y}, y)$, измеряющая ошибку предсказания.

Этап 2: Обратный Проход (Backward Pass)

Что происходит: "Ошибка" J распространяется обратно от выхода ко входу. На каждом слое вычисляются градиенты по параметрам этого слоя и по его входам (активациям предыдущего слоя). **Шаги (идем от слоя L к слою 1):**

1. **Слой L (Выходной):**

- Вычисляем $\partial J/\partial a_L$ (как ошибка зависит от выхода сети).
- Вычисляем $\partial J/\partial z_L = (\partial J/\partial a_L) \odot f'_L(z_L)$. (*Пояснение: Насколько ошибка зависит от пред-активационного значения z_L? Зависит от того, как она зависит от a_L и как a_L меняется с z_L (это f'_L). $\odot$ - поэлементное умножение.*).
- Вычисляем $\partial J/\partial W_L = (\partial J/\partial z_L) \cdot a_{L-1}^T$ и $\partial J/\partial b_L = \sum(\partial J/\partial z_L)$. (*Пояснение: Зная, как z_L влияет на ошибку, и зная, как W_L, b_L влияют на z_L (через вход a_{L-1}), находим градиенты для параметров.*).

2. **Слой l (Скрытый):**
   - Вычисляем $\partial J/\partial a_l = W_{l+1}^T \cdot (\partial J/\partial z_{l+1})$. (*Пояснение: Ошибка "приходит" из следующего слоя $l+1$. Насколько она зависит от выхода a_l этого слоя? Зависит от того, как ошибка зависит от z_{l+1} и как z_{l+1} зависит от a_l (через веса W_{l+1})*).
   - Вычисляем $\partial J/\partial z_l = (\partial J/\partial a_l) \odot f'_l(z_l)$. (*Аналогично выходному слою*).
   - Вычисляем $\partial J/\partial W_l = (\partial J/\partial z_l) \cdot a_{l-1}^T$ и $\partial J/\partial b_l = \sum(\partial J/\partial z_l)$. (*Аналогично выходному слою*).
3. **Повторение:** Шаги для скрытого слоя повторяются до слоя 1.

Результат: Градиенты $\partial J/\partial W_l$ и $\partial J/\partial b_l$ для всех слоев l. **Механизм:** Эффективное применение **цепного правила (chain rule)** дифференцирования.

1.4 VI.D Оптимизаторы: Обновление Весов

Роль Оптимизатора

Использует градиенты, полученные от Backpropagation, для вычисления и применения обновлений к весам **w** и смещениям **b**, чтобы минимизировать функцию потерь J.

SGD (Stochastic Gradient Descent)

Идея: Простой шаг в направлении анти-градиента, вычисленного по батчу. **Формула:** $\mathbf{w} := \mathbf{w} - \alpha \cdot \nabla J(\mathbf{w})$. **Параметр:** Learning rate α.

Momentum

Идея: Добавить "инерцию" к SGD. Учитывает предыдущий шаг обновления v. **Формула:** $v_t = \beta v_{t-1} + \alpha \nabla J(\mathbf{w})$; $\mathbf{w} := \mathbf{w} - v_t$. **Параметры:** α, β (момент, обычно 0.9). **Польза:** Ускоряет сходимость, помогает преодолевать плато.

AdaGrad (Adaptive Gradient)

Идея: Адаптивный learning rate для каждого параметра. Уменьшает шаг для часто обновляемых параметров. **Формула:** Накапливает квадрат градиента G; $\Delta w_i = \frac{\alpha}{\sqrt{G_{ii}+\epsilon}} \nabla J_i(w)$. **Польза:** Хорош для разреженных данных. **Недостаток:** Learning rate может слишком быстро затухнуть.

RMSProp

Идея: Исправить проблему AdaGrad с затуханием шага. Использует скользящее среднее квадратов градиентов $E[g^2]$. **Формула:** $E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma)(\nabla J)^2$; $\Delta w = \frac{\alpha}{\sqrt{E[g^2]_t+\epsilon}} \nabla J(w)$. **Параметры:** α, γ (коэфф. затухания, 0.9).

Adam (Adaptive Moment Estimation)

Идея: Сочетает Momentum (скользящее среднее градиентов m) и RMSProp (скользящее среднее квадратов градиентов v). **Формула:** Использует m и v для вычисления адаптивного шага. Включает коррекцию смещения. **Польза:** Часто эффективен по умолчанию, хорошо работает на широком круге задач. **Параметры:** α, β_1 (0.9), β_2 (0.999).

1.5 VI.E Стабилизация и Регуляризация Обучения

Dropout (Прореживание)

Что это: Метод **регуляризации** для борьбы с переобучением. **Как работает (на обучении):** Случайным образом обнуляет выходы части нейронов слоя с вероятностью p. **Как работает (на предсказании):** Использует все нейроны, но масштабирует их выходы на $(1-p)$. **Эффект:** Заставляет сеть учить более робастные и распределенные представления.

Batch Normalization (BatchNorm)

Что это: Техника для **стабилизации и ускорения** обучения. **Как работает (на обучении):**

1. Нормализует входы z слоя по батчу (среднее 0, дисперсия 1).
2. Масштабирует и сдвигает результат с помощью обучаемых γ и β.
3. Обновляет скользящие средние $\mu_{run}, \sigma^2_{run}$.

Как работает (на предсказании): Использует $\mu_{run}, \sigma^2_{run}$ и обученные γ, β. **Эффект:** Борется с internal covariate shift, позволяет использовать больший learning rate, имеет легкий регуляризующий эффект. Обычно вставляется *до* функции активации.

1.6 VI.F Специализированные Архитектуры

Зачем нужны специализированные сети?

MLP универсальны, но для данных с внутренней структурой (пространственной или временной) CNN и RNN часто более эффективны.

CNN (Convolutional Neural Networks)

Применение: Изображения, видео, данные с сетчатой структурой. **Ключевые Идеи:**

- **Сверточный слой:** Применяет **фильтры (ядра)** для обнаружения локальных паттернов (грани, текстуры). Использует *локальные связи* и *разделяемые веса*. Выход - **карты признаков**.
- **Пулинг слой:** Уменьшает пространственный размер карт признаков (Max Pooling, Average Pooling), обеспечивая инвариантность к малым сдвигам.

Архитектура: Чередование [Conv -> Activation -> Pooling]. Затем полносвязные слои для классификации/регрессии.

RNN (Recurrent Neural Networks)

Применение: Последовательные данные (текст, временные ряды, речь). **Ключевая Идея: Рекуррентная связь** позволяет сети иметь "память" (**скрытое состояние** h_t), передаваемую от шага к шагу. **Простая RNN:** $h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$. Веса W разделяемые по времени. **Проблема:** Затухание/взрыв градиентов на длинных последовательностях (BPTT).

LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit)

Что это: Продвинутые RNN-ячейки для решения проблемы градиентов. **Как работают:** Используют **гейты** (механизмы управления информацией с Sigmoid/-Tanh), чтобы контролировать, что запоминать, что забывать, и что передавать дальше. **LSTM:** Имеет 3 гейта (input, forget, output) и состояние ячейки (cell state). **GRU:** Упрощенная версия с 2 гейтами (reset, update). **Использование:** Стандарт де-факто для задач с последовательностями вместо простых RNN.