

Шпаргалка по нейронным сетям / Концепции Cheatsheet (XeLaTeX)

Краткий справочник

Содержание

1	VI. Введение в Нейронные Сети (NN)	1
1.1	VI.A Базовые Структуры: Нейроны и Слои	1
1.2	VI.B Функции Активации: Внесение Нелинейности	1
1.2.1	ReLU и его вариации	1
1.2.2	Sigmoid и Tanh	2
1.2.3	Softmax	2
1.2.4	Проблема Затухания/Взрыва Градиентов	2
1.3	VI.C Backpropagation: Как Сеть Учится	2
1.4	VI.D Оптимизаторы: Обновление Весов	3
1.5	VI.E Стабилизация и Регуляризация Обучения	3
1.5.1	Dropout	3
1.5.2	Batch Normalization	3
1.6	VI.F Специализированные Архитектуры: CNN и RNN	4
1.6.1	CNN (Convolutional Neural Networks) - Сети для "Зрения"	4
1.6.2	RNN (Recurrent Neural Networks) - Сети для Последовательностей	4

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, нормализация). Заложить основу для понимания сверточных и рекуррентных сетей.

1.1. VI.A Базовые Структуры: Нейроны и Слои

Искусственный Нейрон: Определение

Что это: Математическая модель, имитирующая базовую функцию биологического нейрона. Служит основным вычислительным элементом нейросети.

Искусственный Нейрон: Принцип Работы

Шаги вычисления:

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. Результат z (**пред-активация** или **логит**) пропускается через **функцию активации** $f(\cdot)$.
6. Выход нейрона $\rightarrow a = f(z)$ (**активация**).

Обучаемые параметры: Веса w_i и смещение b .

Многослойный Перцептрон (MLP): Архитектура

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

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

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

1.2.1. ReLU и его вариации

1.2. VI.B Функции Активации: Внесение Нелинейности

Зачем нужна Нелинейность?

Без нелинейных функций активации (f) в скрытых слоях вся нейросеть (даже глубокая) была бы математически эквивалентна одному линейному слою (т.е., простой линейной или логистической регрессии). Нелинейность позволяет сети изучать сложные, нелинейные зависимости в данных.

ReLU (Rectified Linear Unit)

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

Свойства: Вычислительно очень проста. Не "насыщается" для $x > 0$ (производная = 1), что помогает градиентам проходить через глубокие сети. **Недостаток:** "Умирующие ReLU" (Dying ReLU) — если вход нейрона стабильно ≤ 0 , градиент через него перестает проходить, и нейрон перестает обучаться. **Использование:** Стандартный выбор для скрытых слоев в большинстве современных архитектур.

Leaky ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \quad (\alpha - \text{малый коэфф., е.g., 0.01})$$

Свойства: Решает проблему "умирающих ReLU", позволяя небольшому градиенту (α) проходить при $x \leq 0$.

ELU (Exponential Linear Unit)

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

Свойства: Похожа на Leaky ReLU, но использует экспоненту. Может давать более гладкие градиенты и отрицательные выходы, что иногда полезно.

1.2.2. Sigmoid и Tanh

Sigmoid (Сигмоида)

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

Свойства: Сжимает выход в диапазон $[0, 1]$, удобен для интерпретации как вероятность. **Недостатки:** Сильно "насыщается" при больших $|x|$ (градиент близок к 0) \rightarrow проблема **затухания градиентов**. Выход не центрирован около нуля. **Использование:** Только выходной слой для **бинарной классификации**. Избегать в скрытых слоях.

Tanh (Гиперболический тангенс)

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

Свойства: Сжимает выход в диапазон $[-1, 1]$, выход центрирован около нуля (лучше Sigmoid для скрытых слоев). **Недостатки:** Также страдает от затухания градиентов, хотя и меньше, чем Sigmoid. **Использование:** Иногда в скрытых слоях, часто в ячейках RNN/LSTM/GRU.

1.2.3. Softmax

Softmax

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}} \quad (\text{для } i = 1..N \text{ нейронов})$$

Свойства: Преобразует вектор "сырых" оценок (логитов x_i) в вектор вероятностей (все $f(x_i) \geq 0$ и $\sum f(x_i) = 1$). **Использование:** Только в выходном слое для **многоклассовой классификации**.

Графики популярных функций активации



1.2.4. Проблема Затухания/Взрыва Градиентов

Определение Проблемы (Vanishing/Exploding Gradients)

Проблема: При обучении глубоких сетей (много слоев), во время обратного прохода (Backpropagation) градиенты, передаваемые от слоя к слою, могут либо экспоненциально уменьшаться (**затухание**, vanishing), становясь близкими к нулю, либо экспоненциально расти (**взрыв**, exploding). **Причины:** Повторное

умножение на веса и производные функций активации (особенно насыщающихся).

Последствия Проблемы Градиентов

- **Затухание:** Ранние слои сети почти не обучаются, так как до них не доходит "сигнал" ошибки. Обучение очень медленное или останавливается.
- **Взрыв:** Большие изменения весов приводят к нестабильности обучения, расходящимся значениям функции потерь (NaN).

Решения Проблемы Градиентов

- Использование ненасыщающихся активаций (ReLU и его варианты).
- Batch Normalization.
- Правильная инициализация весов (Xavier/Glorot, He).
- Residual Connections (проброс связей в обход слоев, как в ResNet).
- Обрезание градиентов (Gradient Clipping) - в основном для борьбы со взрывом.

1.3. VI.C Backpropagation: Как Сеть Учится

Backpropagation: Ключевой Алгоритм Обучения

Цель: Эффективно вычислить **градиенты** (частные производные) функции потерь J по **всем** обучаемым параметрам (W_l, b_l каждого слоя l). Градиент $\partial J / \partial w$ показывает, насколько сильно и в каком направлении изменится итоговая ошибка J , если немного изменить вес w .

Этап 1: Прямой Проход (Forward Pass)

Что происходит: Данные X проходят через сеть слой за слоем от входа к выходу. На каждом слое l вычисляются пред-активационные значения $z_l = W_l a_{l-1} + b_l$ и активации $a_l = f_l(z_l)$. Промежуточные z_l и a_l сохраняются. **Результат:** Итоговые предсказания сети $a_L = \hat{y}$. **Затем:** Вычисляется **функция потерь** $J(\hat{y}, y)$, измеряющая ошибку предсказания.

Этап 2: Обратный Проход (Backward Pass) - Идея

Задача: Распространить ошибку J обратно через сеть для вычисления градиентов $\partial J / \partial W_l$ и $\partial J / \partial b_l$. **Метод:** Применение **цепного правила (chain rule)** дифференцирования для вычисления производной сложной функции (функции потерь J , которая зависит от всех весов и смещений через цепочку вычислений).

Этап 2: Обратный Проход (Backward Pass) - Механика

Как работает (от слоя L к слою 1):

1. **Начало (выходной слой L):** Вычисляется $\partial J / \partial a_L$ (из производной функции потерь) и $\partial J / \partial z_L = (\partial J / \partial a_L) \odot f'_L(z_L)$.
2. **Вычисление градиентов для слоя l :** Зная $\partial J / \partial z_l$, вычисляем:
 - $\partial J / \partial W_l = (\partial J / \partial z_l) \cdot a_{l-1}^T$
 - $\partial J / \partial b_l = \sum (\partial J / \partial z_l)$ (суммирование по батчу/примерам)
3. **Передача ошибки на предыдущий слой ($l - 1$):** Вычисляем $\partial J / \partial a_{l-1} = W_l^T \cdot (\partial J / \partial z_l)$.
4. **Переход к z_{l-1} :** Вычисляем $\partial J / \partial z_{l-1} = (\partial J / \partial a_{l-1}) \odot f'_{l-1}(z_{l-1})$.
5. **Повторение:** Шаги 2-4 повторяются для всех слоев от $L - 1$ до 1.

(Примечание: \odot обозначает поэлементное умножение (Hadamard product), \cdot - матричное умножение). **Результат:** Градиенты $\partial J / \partial W_l$ и $\partial J / \partial b_l$ для всех слоев l .

1.4. VI.D Оптимизаторы: Обновление Весов

Роль Оптимизатора

Использует градиенты ∇J , полученные от Backpropagation, для вычисления и применения обновлений к весам \mathbf{w} и смещениям \mathbf{b} сети, с целью минимизировать функцию потерь J . Определяет как именно использовать градиент для шага.

Mini-batch Gradient Descent (MBGD)

Подход: Градиент ∇J_{batch} вычисляется (усредняется) по небольшой случайной подвыборке (**мини-батч**, mini-batch) данных. **Обновление:** $\mathbf{w} := \mathbf{w} - \alpha \cdot \nabla J_{batch}(\mathbf{w})$. **Преимущества:** Быстрее Batch GD, стабильнее SGD, позволяет использовать матричные операции. **Гиперпараметр:** Learning rate α (скорость обучения).

Momentum

Идея: Добавить "инерцию" к шагу, учитывая предыдущее направление движения v . **Формула (идея):** $v_t = \beta v_{t-1} + \alpha \nabla J(\mathbf{w})$; $\mathbf{w} := \mathbf{w} - v_t$. **Гиперпараметры:** α, β (коэфф. момента, обычно 0.9). **Польза:** Ускоряет сходимость в "оврагах" функции потерь, помогает преодолевать локальные минимумы и плато.

AdaGrad (Adaptive Gradient)

Идея: Адаптивный learning rate для каждого параметра отдельно. Уменьшает шаг для параметров с часто большими градиентами. **Механика:** Накапливает сумму квадратов всех прошлых градиентов G . Делит learning rate α на $\sqrt{G + \epsilon}$. **Польза:** Хорош для разреженных данных. **Недостаток:** Learning rate может слишком быстро уменьшиться до нуля.

RMSProp

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

Adam (Adaptive Moment Estimation)

Идея: Сочетает идеи Momentum (EMA градиентов m) и RMSProp (EMA квадратов градиентов v). **Механика:** Использует m и v для адаптивного шага. Включает коррекцию смещения m и v на начальных этапах. **Польза:** Часто самый эффективный оптимизатор по умолчанию. **Гиперпараметры:** α, β_1 (для m , 0.9), β_2 (для v , 0.999).

1.5. VI.E Стабилизация и Регуляризация Обучения

1.5.1. Dropout

Dropout: Определение и Цель

Что это: Метод регуляризации для борьбы с переобучением в нейросетях. **Идея:** Создание подобия ансамбля из множества "прореженных" подсетей.

Dropout: Механизм Работы

На обучении: Перед каждым прямым проходом для каждого нейрона в слое (кроме выходного) с вероятностью p (e.g., $p = 0.5$) его выход искусственно **обнуляется** для данного прохода. Разные нейроны обнуляются на разных проходах. **На предсказании/тесте:** Используются **все** нейроны, но их выходы умножаются на $(1 - p)$ (inverted dropout) или используется усреднение по сетям (менее практично). **Эффект:** Заставляет сеть учить более робастные и распределенные представления, не полагаясь на отдельные нейроны.

1.5.2. Batch Normalization

Batch Normalization (BatchNorm): Определение и Цель

Что это: Техника для стабилизации и ускорения обучения глубоких сетей. **Цель:** Борьба с проблемой **Internal Covariate Shift** (изменение распределения входов слоев во время обучения из-за изменения параметров предыдущих слоев).

BatchNorm: Работа на Обучении

Процесс для входа z слоя по батчу B :

1. **Считаем статистику батча:** $\mu_B = \text{mean}(z)$, $\sigma_B^2 = \text{variance}(z)$.
2. **Нормализуем:** $\hat{z} = (z - \mu_B) / \sqrt{\sigma_B^2 + \epsilon}$. (\hat{z} имеет среднее 0, дисперсию 1 по батчу).
3. **Масштабируем и сдвигаем:** $y_{BN} = \gamma \hat{z} + \beta$. (γ, β - обучаемые параметры).
4. **Обновляем скользящие средние:** Параллельно обновляются $\mu_{run}, \sigma_{run}^2$ (оценка среднего и дисперсии по всей датасету через EMA).

BatchNorm: Работа на Предсказании/Тесте

Процесс: Использует **сохраненные** скользящие статистики $\mu_{run}, \sigma_{run}^2$ и **обученные** параметры γ, β . **Формула:** $y_{BN} = \gamma \frac{z - \mu_{run}}{\sqrt{\sigma_{run}^2 + \epsilon}} + \beta$.

BatchNorm: Эффекты и Применение

Эффект:

- Стабилизирует и значительно ускоряет обучение.
- Позволяет использовать больший learning rate.
- Действует как слабая форма регуляризации.
- Снижает чувствительность к инициализации весов.

Применение: Обычно вставляется между линейным преобразованием ($Wx + b$) и нелинейной активацией (f).

1.6. VI.F Специализированные Архитектуры: CNN и RNN

Зачем нужны специализированные сети?

MLP (полносвязные сети) универсальны, но не учитывают структуру данных (пространственную, временную). Для таких данных CNN и RNN часто гораздо более эффективны и требуют меньше параметров, так как используют априорные знания о структуре через свои архитектурные особенности.

1.6.1. CNN (Convolutional Neural Networks) - Сети для "Зрения"

Ключевые Идеи CNN

- Локальность Связей:** Нейроны смотрят только на небольшую локальную область входа (**рецептивное поле**). Учитывает пространственную близость пикселей/элементов.
- Общие Веса (Parameter Sharing):** Один и тот же набор весов (**фильтр**) используется для обработки разных частей входа → инвариантность к сдвигу, резкое снижение числа параметров.
- Иерархия Признаков:** Слои учатся распознавать все более сложные паттерны, комбинируя выходы предыдущих слоев (границы → текстуры → части объектов → объекты).

Основное Применение: Изображения, видео, аудио (спектрограммы), иногда тексты (1D свёртки).

Сверточный Слой (Conv Layer)

Задача: Извлечение локальных признаков с помощью набора обучаемых **фильтров (ядер, kernels)**. **Принцип работы:** Каждый фильтр (маленькая матрица весов, e.g., 3x3) "скользит" по входу. В каждой позиции вычисляется скалярное произведение весов фильтра и соответствующего участка входа + смещение. Результаты для одного фильтра формируют **карту признаков (feature map)**. **Гиперпараметры:** Число фильтров (глубина выхода), размер фильтра (e.g., 3x3, 5x5), шаг (stride), заполнение (padding: 'same' для сохранения размера, 'valid' без заполнения). **После Conv:** Обычно следует функция активации (ReLU).

Пулинг Слой (Pooling Layer)

Задача: Уменьшение пространственного размера карт признаков (downsampling). **Цели:**

- Снижение вычислительной нагрузки и числа параметров в последующих слоях.

- Повышение робастности к малым сдвигам/искажениям входных данных (инвариантность).

Принцип работы: Применяет агрегирующую функцию к непересекающимся (обычно) окнам (e.g., 2x2).

- Max Pooling:** Выбирает максимальное значение в окне (сохраняет самые сильные активации признаков). Наиболее распространен.
- Average Pooling:** Вычисляет среднее значение в окне.

Гиперпараметры: Тип пулинга, размер окна (pool size), шаг (stride, часто равен размеру окна).

Типичная Архитектура CNN

Часто состоит из чередующихся блоков свертки и пулинга, за которыми следуют полносвязные слои для классификации/регрессии: Вход → [Conv → Activation → (BatchNorm) → Pool]*N → Flatten → [Dense → Activation → (Dropout/BatchNorm)]*M → Выходной Dense Слой (N, M - количество блоков/слоев)

1.6.2. RNN (Recurrent Neural Networks) - Сети для Последовательностей

Ключевые Идеи RNN

- Обработка последовательностей:** Предназначены для данных, где важен порядок элементов (текст, временные ряды).
- Рекуррентная связь ("Память"):** Выход сети на шаге t зависит не только от входа x_t , но и от информации с предыдущих шагов, хранящейся в **скрытом состоянии (hidden state)** h_t . Состояние h_{t-1} передается на шаг t .
- Общие Веса во Времени (Parameter Sharing):** Один и тот же набор весов используется для обработки каждого элемента последовательности x_t .

Основное Применение: Обработка естественного языка (NLP), анализ временных рядов, распознавание речи, генерация музыки.

Простая RNN Ячейка (Simple RNN / Elman RNN)

Формула: $h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$. Новое состояние h_t вычисляется на основе текущего входа x_t и предыдущего состояния h_{t-1} с использованием функции активации f (часто \tanh). **Выход (опционально):** $y_t = g(W_{hy}h_t + b_y)$. **Основная Проблема:** Затухание/взрыв градиентов при обучении на длинных последовательностях → трудности с запоминанием долгосрочных зависимостей.

LSTM Ячейка (Long Short-Term Memory)

Цель: Решить проблему градиентов RNN и улучшить долговременную память с помощью **гейтов** (управляющих механизмов на основе сигмоид). **Ключевые Компоненты:**

- Состояние Ячейки (C_t):** Основной канал для хранения информации ("конвейер памяти"). Может передавать информацию почти без изменений.
- Forget Gate (f_t):** Решает, какую информацию из C_{t-1} нужно "забыть".
- Input Gate (i_t):** Решает, какая новая информация из входа x_t и h_{t-1} будет сохранена в C_t . Состоит из двух частей: сигмоиды i_t и $\tanh \tilde{C}_t$ (кандидат на добавление).
- Output Gate (o_t):** Решает, какая часть состояния ячейки C_t будет выведена как скрытое состояние h_t .

Результат: Может эффективно хранить, читать и записывать информацию, управляя потоком данных через гейты.

GRU Ячейка (Gated Recurrent Unit)

Цель: Упрощенная версия LSTM с меньшим числом параметров, также эффективно борется с проблемой градиентов. **Ключевые Компоненты:**

- Update Gate (z_t):** Комбинирует Forget и Input гейты LSTM. Определяет, сколько информации из прошлого состояния h_{t-1} сохранить и сколько добавить из нового кандидата \tilde{h}_t .
- Reset Gate (r_t):** Определяет, насколько информация из прошлого состояния h_{t-1} будет использоваться для вычисления нового кандидата \tilde{h}_t .

Особенности: Нет отдельного состояния ячейки C_t . **Результат:** Сравнимая с LSTM производительность на многих задачах при меньшей сложности.

Использование RNN/LSTM/GRU

Обрабатывают последовательность входов x_1, \dots, x_T . Возможные сценарии использования выходов:

- Многие-к-одному (Many-to-one):** Используется только последний выход y_T или состояние h_T (e.g., классификация текста по всей последовательности).
- Один-ко-многим (One-to-many):** Один вход x_1 , генерация последовательности y_1, \dots, y_T (e.g., генерация описания по картинке).
- Многие-ко-многим (Many-to-many, синхронный):** Для каждого x_t генерируется y_t (e.g., разметка частей речи в предложении).
- Многие-ко-многим (Many-to-many, асинхронный):** Вся входная последовательность читается, затем генерируется выходная (seq-to-seq, e.g., машинный перевод).

Часто используется несколько слоев (Stacked RNN), где выход h_t одного слоя является входом x_t для следующего.