**Count-based methods**

Определение:

- Count-based methods - это методы представления слов и документов, основанные на подсчете частоты слов в корпусе текстов.

- Они используют статистическую информацию о совместной встречаемости слов для создания векторных представлений.

1. **Bag-of-Words (BoW)**:

- BoW - это простой метод представления документов в виде векторов фиксированной длины.

- Каждый элемент вектора соответствует уникальному слову в словаре и представляет частоту этого слова в документе.

- BoW игнорирует порядок слов и грамматическую структуру, рассматривая документ как неупорядоченный набор слов.

*формула:*

Bag-of-Words (BoW):

- Представление документа d как вектора BoW(d) = [tf(w1, d), tf(w2, d), ..., tf(wn, d)]

- где tf(wi, d) - частота слова wi в документе d, а n - размер словаря.

2. **Term Frequency-Inverse Document Frequency (TF-IDF)**:

- TF-IDF - это метод взвешивания слов в документах на основе их важности.

- TF (Term Frequency) измеряет частоту слова в документе, а IDF (Inverse Document Frequency) измеряет редкость слова во всем корпусе.

- Слова с высоким TF-IDF считаются более важными и информативными для документа.

- TF-IDF помогает выделить ключевые слова и уменьшить влияние часто встречающихся слов.

*формула:*

Term Frequency-Inverse Document Frequency (TF-IDF):

- TF(w, d) = частота слова w в документе d / общее количество слов в документе d

- IDF(w) = log(общее количество документов / количество документов, содержащих слово w)

- TF-IDF(w, d) = TF(w, d) \* IDF(w)

**3. Co-occurrence Matrix:**

- Co-occurrence Matrix - это матрица, которая фиксирует совместную встречаемость слов в заданном окне контекста.

- Каждая ячейка матрицы представляет количество раз, когда два слова встречаются вместе в заданном окне.

- Co-occurrence Matrix может быть использована для получения векторных представлений слов на основе их контекстной информации.

*формула:*

Co-occurrence Matrix:

- Матрица M размера n x n, где n - размер словаря

- M[i, j] = количество раз, когда слово wi и слово wj встречаются вместе в заданном окне контекста

**4. Pointwise Mutual Information (PMI):**

- PMI - это мера ассоциации между двумя словами, основанная на их совместной встречаемости и индивидуальных вероятностях.

- PMI вычисляется как логарифм отношения совместной вероятности двух слов к произведению их индивидуальных вероятностей.

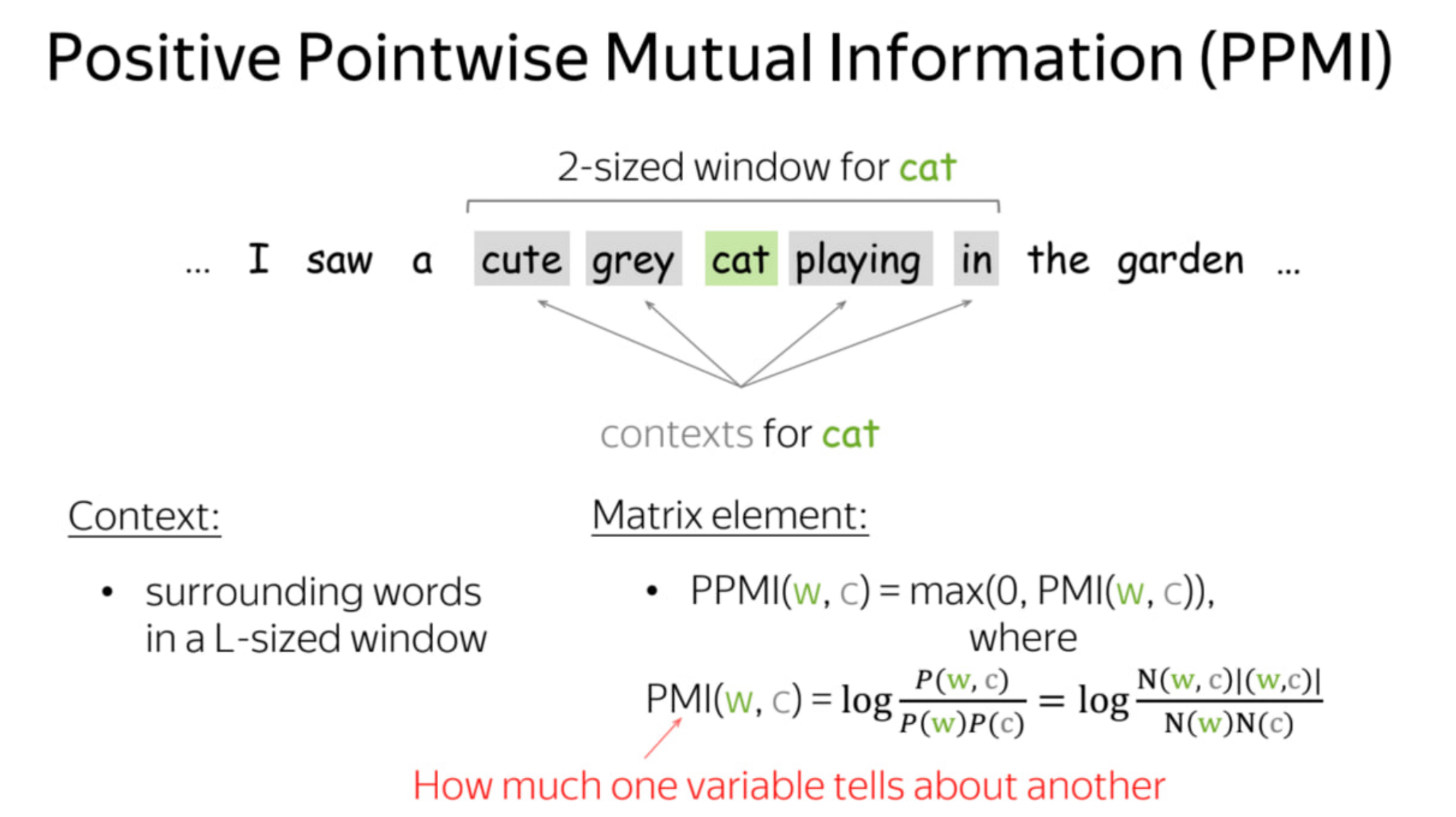
- Высокое значение PMI указывает на сильную ассоциацию между словами, что может быть полезно для задач, таких как извлечение ключевых слов или кластеризация слов.

*формула:*

Pointwise Mutual Information (PMI):

- PMI(w1, w2) = log(P(w1, w2) / (P(w1) \* P(w2)))

- где P(w1, w2) - совместная вероятность слов w1 и w2, а P(w1) и P(w2) - индивидуальные вероятности слов w1 и w2 соответственно.



Преимущества и недостатки:

- Преимущества count-based methods: простота реализации, интерпретируемость, возможность работы с большими корпусами текстов.

- Недостатки: игнорирование семантики и контекста слов, проблема разреженности векторов, ограниченная способность захватывать сложные языковые структуры.

**Word2Vec:**

- Разработан командой Google в 2013 году.

- Два основных подхода: Continuous Bag-of-Words (CBOW) и Skip-gram.

- CBOW предсказывает целевое слово по контексту, Skip-gram предсказывает контекст по целевому слову.

- Использует неглубокую нейронную сеть для обучения эмбедингов.

- Функция потерь: обычно используется Negative Sampling или Hierarchical Softmax.

*формула:*

Word2Vec (Skip-gram):

- Функция потерь: L = -log(σ(u\_o^T v\_c)) - Σ\_k log(σ(-u\_k^T v\_c))

- где:

- u\_o - вектор выходного слова

- v\_c - вектор входного (контекстного) слова

- u\_k - вектора отрицательных примеров (слова, не связанные с контекстом)

- σ - сигмоидная функция: σ(x) = 1 / (1 + exp(-x))

Объяснение:

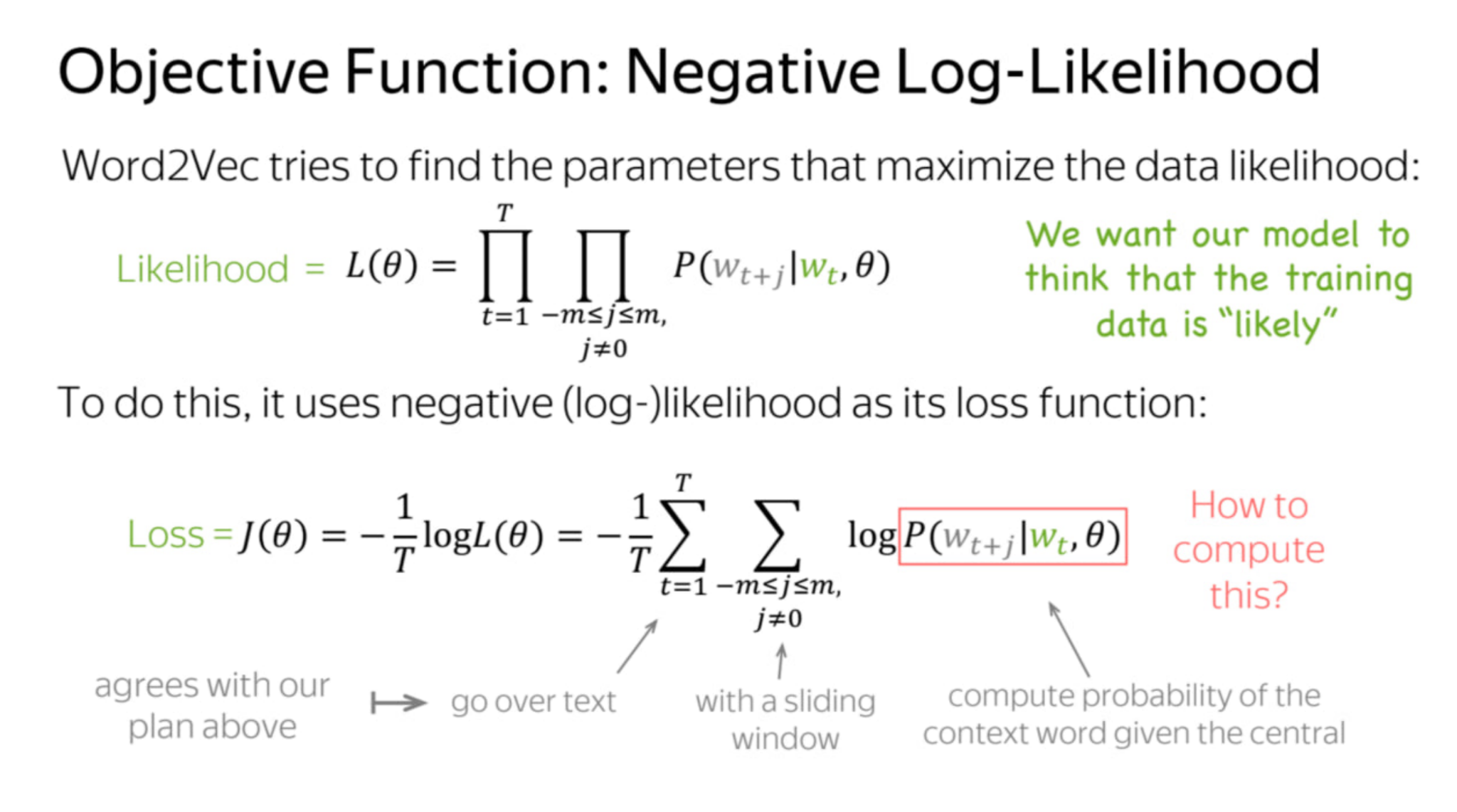
- Цель модели Skip-gram - **максимизировать вероятность предсказания выходного слова по заданному входному (контекстному) слову**.

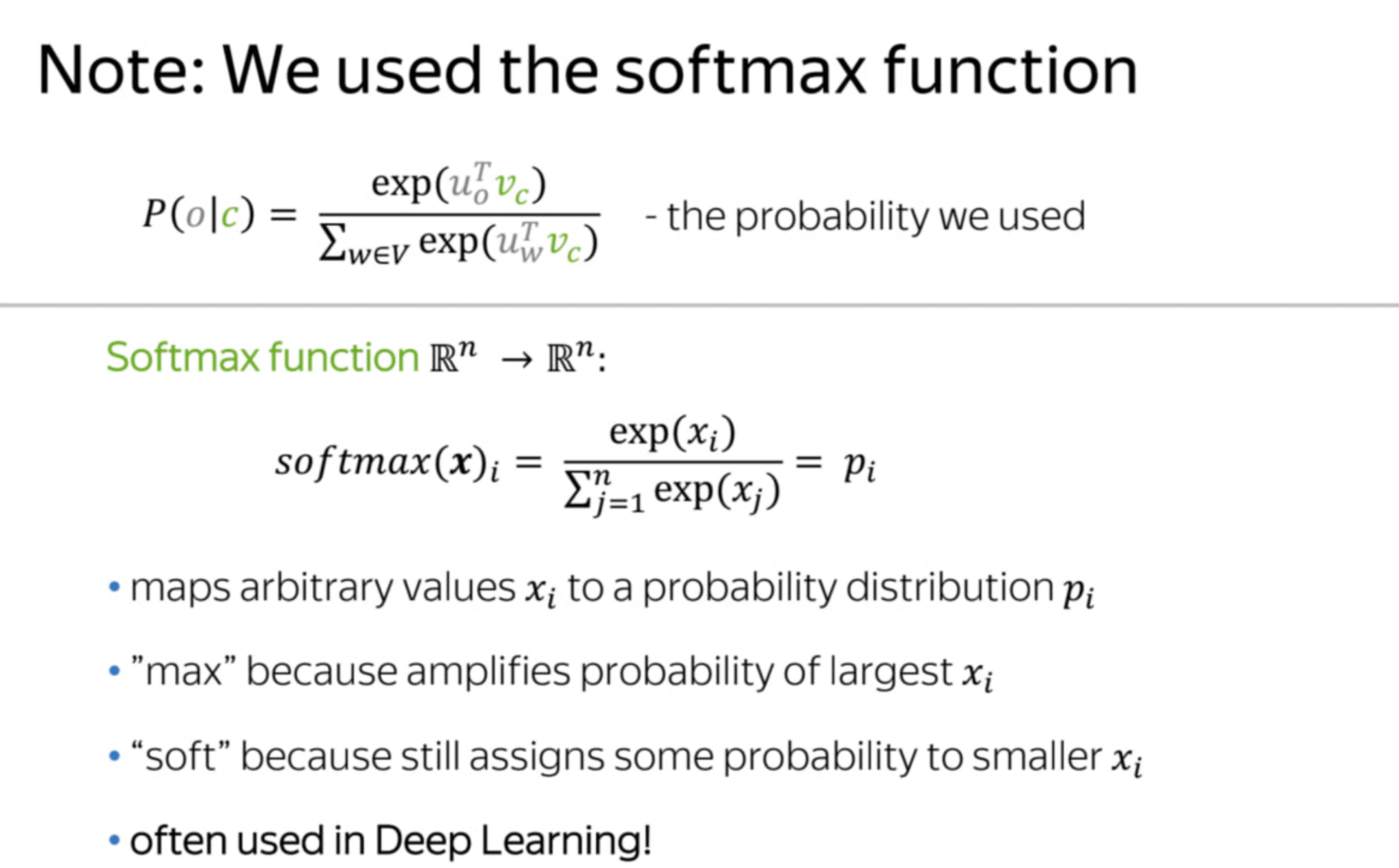
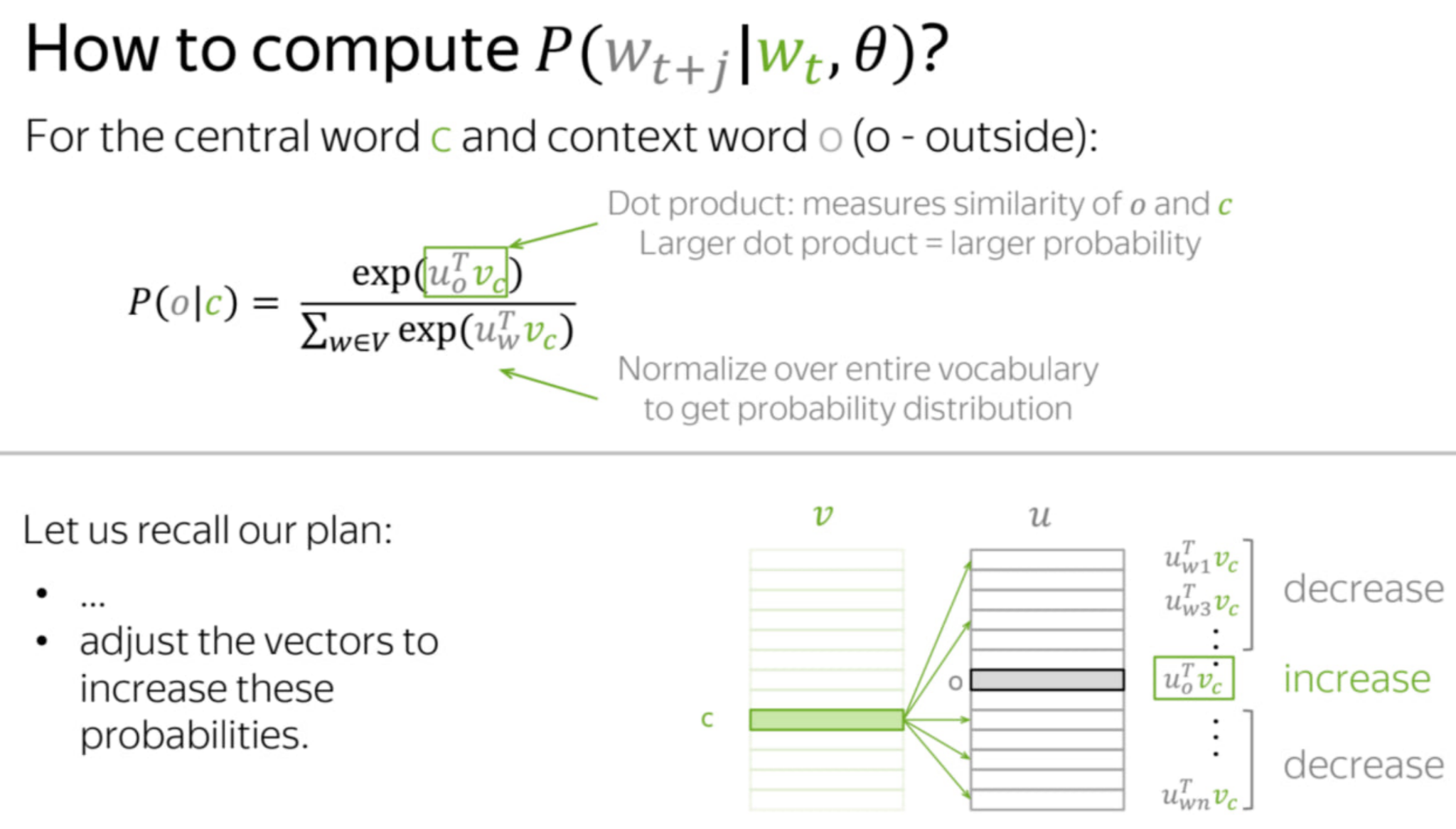
- Первый член в функции потерь максимизирует скалярное произведение векторов выходного и входного слов, поощряя их близость.

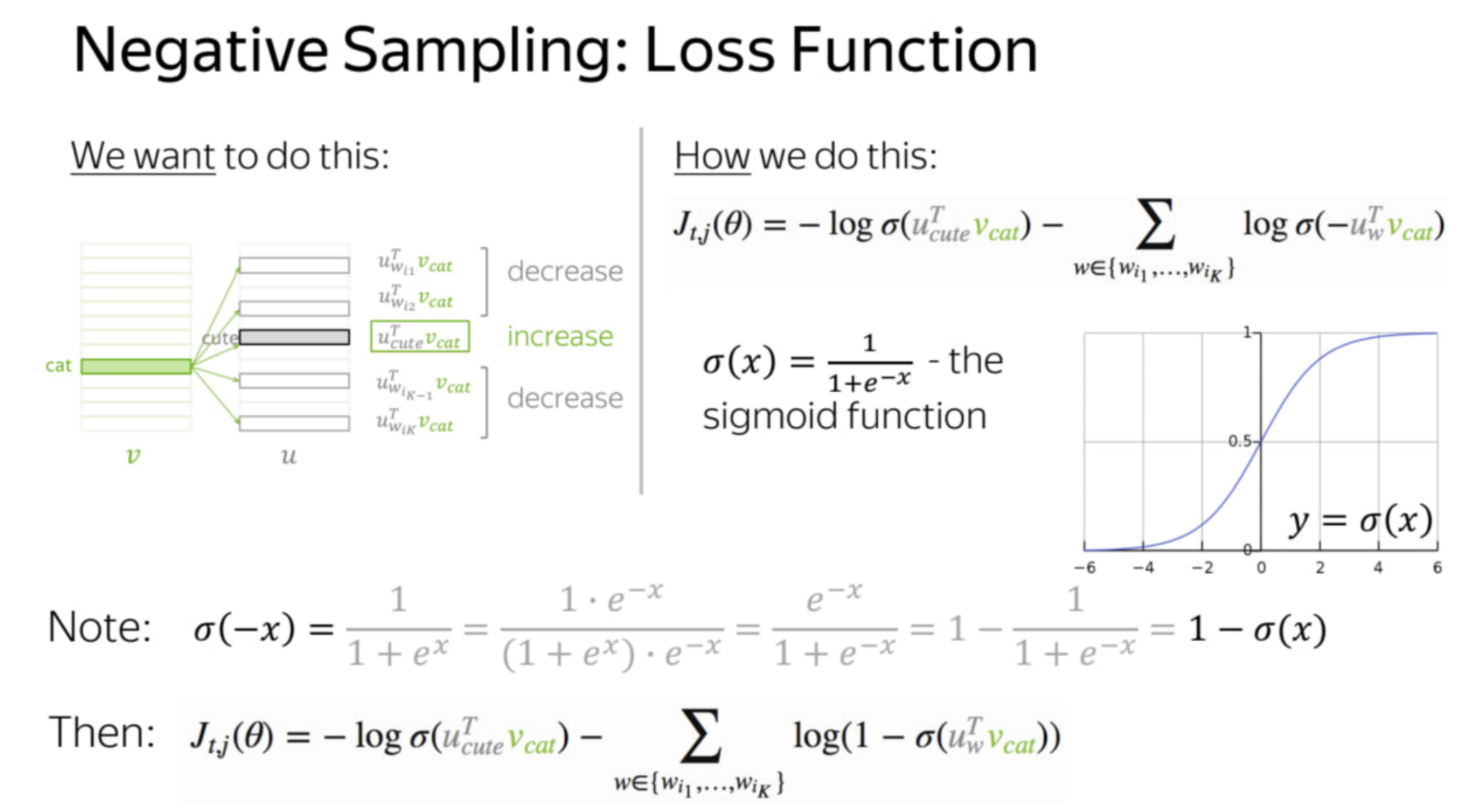
- Второй член минимизирует скалярное произведение векторов отрицательных примеров и входного слова, отталкивая их друг от друга.

- Сигмоидная функция преобразует скалярные произведения в вероятности.

- Минимизация этой функции потерь приводит к обучению векторных представлений слов, которые отражают их семантическую близость.







*код реализации:*

import numpy as np

class Word2Vec:

def init(self, vocab\_size, embedding\_size):

self.vocab\_size = vocab\_size

self.embedding\_size = embedding\_size

self.embeddings = np.random.uniform(-1, 1, (vocab\_size, embedding\_size))

self.context\_weights = np.random.uniform(-1, 1, (vocab\_size, embedding\_size))

def forward(self, center\_word, context\_word):

center\_embedding = self.embeddings[center\_word]

context\_weight = self.context\_weights[context\_word]

score = np.dot(center\_embedding, context\_weight)

return score

def backward(self, center\_word, context\_word, learning\_rate):

center\_embedding = self.embeddings[center\_word]

context\_weight = self.context\_weights[context\_word]

gradient = -context\_weight

self.embeddings[center\_word] -= learning\_rate \* gradient

self.context\_weights[context\_word] -= learning\_rate \* center\_embedding

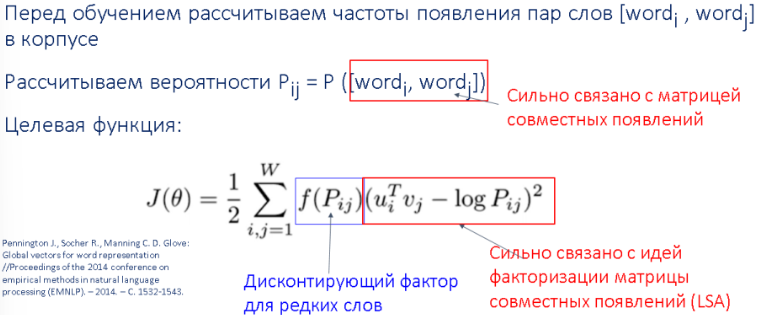
**GloVe (Global Vectors for Word Representation):**

- Разработан командой Stanford в 2014 году.

- Основан на матрице совместной встречаемости слов.

- Использует статистику совместной встречаемости слов для обучения эмбедингов.

- Функция потерь: минимизирует разницу между скалярным произведением векторов слов и логарифмом их совместной встречаемости.



*формула:*

- Функция потерь: L = Σ\_i Σ\_j f(X\_ij) (w\_i^T w\_j + b\_i + b\_j - log(X\_ij))^2

- где:

- X\_ij - количество совместных появлений слов i и j в заданном окне контекста

- w\_i, w\_j - векторные представления слов i и j

- b\_i, b\_j - смещения (biases) для слов i и j

- f - весовая функция, которая придает меньший вес редким совместным появлениям: f(x) = (x / x\_max)^α if x < x\_max, 1 otherwise

Объяснение:

- GloVe обучает векторные представления слов, минимизируя разницу между скалярным произведением векторов слов и логарифмом их совместной встречаемости.

- Весовая функция f уменьшает влияние редких совместных появлений, которые могут быть менее информативными.

- Смещения b\_i и b\_j учитывают различия в частоте появления отдельных слов.

- Минимизация этой функции потерь приводит к обучению векторных представлений слов, которые отражают их совместную встречаемость и семантическую близость.

*код реализации:*

import numpy as np

class GloVe:

def init(self, vocab\_size, embedding\_size):

self.vocab\_size = vocab\_size

self.embedding\_size = embedding\_size

self.word\_embeddings = np.random.uniform(-1, 1, (vocab\_size, embedding\_size))

self.context\_embeddings = np.random.uniform(-1, 1, (vocab\_size, embedding\_size))

self.biases = np.random.uniform(-1, 1, (vocab\_size, 2))

def forward(self, word, context):

word\_embedding = self.word\_embeddings[word]

context\_embedding = self.context\_embeddings[context]

bias\_word = self.biases[word][0]

bias\_context = self.biases[context][1]

score = np.dot(word\_embedding, context\_embedding) + bias\_word + bias\_context

return score

def backward(self, word, context, cooccurrence, learning\_rate):

word\_embedding = self.word\_embeddings[word]

context\_embedding = self.context\_embeddings[context]

bias\_word = self.biases[word][0]

bias\_context = self.biases[context][1]

error = np.dot(word\_embedding, context\_embedding) + bias\_word + bias\_context - np.log(cooccurrence)

grad\_word = error \* context\_embedding

grad\_context = error \* word\_embedding

self.word\_embeddings[word] -= learning\_rate \* grad\_word

self.context\_embeddings[context] -= learning\_rate \* grad\_context

self.biases[word][0] -= learning\_rate \* error

self.biases[context][1] -= learning\_rate \* error

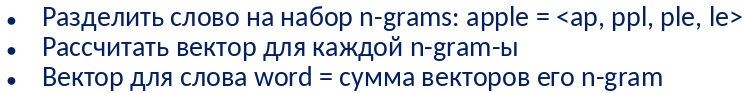
**FastText:**

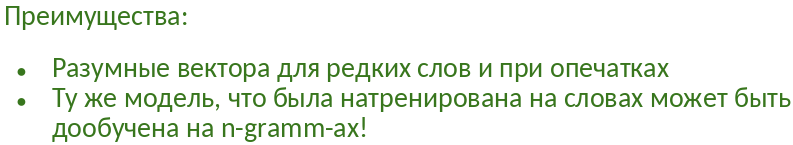
- Разработан командой Facebook в 2016 году.

- Расширение Word2Vec, учитывающее информацию о подсловах (n-граммах символов).

- Может генерировать эмбединги для несуществующих слов на основе их подслов.

- Использует ту же архитектуру и функции потерь, что и Word2Vec.





*код реализации:*

class FastText:

def init(self, vocab\_size, embedding\_size, ngram\_size):

self.vocab\_size = vocab\_size

self.embedding\_size = embedding\_size

self.ngram\_size = ngram\_size

self.word\_embeddings = np.random.uniform(-1, 1, (vocab\_size, embedding\_size))

self.ngram\_embeddings = np.random.uniform(-1, 1, (ngram\_size, embedding\_size))

self.context\_weights = np.random.uniform(-1, 1, (vocab\_size, embedding\_size))

def forward(self, word, context):

word\_embedding = self.word\_embeddings[word]

ngram\_embeddings = [self.ngram\_embeddings[ngram] for ngram in self.get\_ngrams(word)]

word\_embedding += np.sum(ngram\_embeddings, axis=0)

context\_weight = self.context\_weights[context]

score = np.dot(word\_embedding, context\_weight)

return score

def backward(self, word, context, learning\_rate):

word\_embedding = self.word\_embeddings[word]

ngram\_embeddings = [self.ngram\_embeddings[ngram] for ngram in self.get\_ngrams(word)]

context\_weight = self.context\_weights[context]

gradient = -context\_weight

self.word\_embeddings[word] -= learning\_rate \* gradient

for ngram, ngram\_embedding in zip(self.get\_ngrams(word), ngram\_embeddings):

self.ngram\_embeddings[ngram] -= learning\_rate \* gradient

self.context\_weights[context] -= learning\_rate \* (word\_embedding + np.sum(ngram\_embeddings, axis=0))

def get\_ngrams(self, word):

ngrams = []

for i in range(len(word) - self.ngram\_size + 1):

ngrams.append(word[i:i+self.ngram\_size])

return ngrams

Отличия:

- Word2Vec и FastText основаны на предсказании слов по контексту или наоборот, в то время как GloVe использует статистику совместной встречаемости слов.

- FastText учитывает информацию о подсловах, что позволяет ему работать с несуществующими словами и морфологически богатыми языками.

- GloVe обучается на глобальной статистике совместной встречаемости, в то время как Word2Vec и FastText обучаются на локальных контекстах.

**Методы классификации поверх текста до появления трансформеров**

1. **Наивный байесовский классификатор:**

- Формула: P(C|X) = (P(X|C) \* P(C)) / P(X)

- где:

- C - класс

- X - вектор признаков

- P(C|X) - вероятность класса C при наблюдении признаков X (апостериорная вероятность)

- P(X|C) - вероятность наблюдения признаков X при условии класса C (правдоподобие)

- P(C) - вероятность класса C (априорная вероятность)

- P(X) - вероятность наблюдения признаков X (доказательство)

*Объяснение:*

- Наивный байесовский классификатор основан на теореме Байеса и предположении о независимости признаков.

- Он вычисляет апостериорную вероятность каждого класса при наблюдении признаков и выбирает класс с наибольшей вероятностью.

- Априорные вероятности и правдоподобия оцениваются на основе обучающих данных.

- Несмотря на свою простоту, наивный байесовский классификатор может быть эффективным во многих реальных задачах, особенно при наличии большого количества признаков.

2. **Логистическая регрессия:**

- Формула: y = σ(w^T x + b), где σ(z) = 1 / (1 + exp(-z))

- где:

- y - предсказанная вероятность принадлежности к классу 1

- w - вектор весов

- x - вектор признаков

- b - смещение (bias)

- σ - сигмоидная функция

*Объяснение:*

- Логистическая регрессия - это линейный классификатор, который моделирует вероятность принадлежности к классу с помощью сигмоидной функции.

- Она находит оптимальные веса и смещение, минимизируя функцию потерь, обычно бинарную кросс-энтропию.

- Предсказанная вероятность интерпретируется как вероятность принадлежности к классу 1, а порог 0.5 используется для принятия решения о классе.

- Логистическая регрессия широко используется для задач бинарной классификации и может быть расширена для многоклассовой классификации (например, с помощью стратегии "один против всех").

3. **Опорные векторные машины (SVM):**

- Формула: y = sign(w^T x + b)

- где:

- y - предсказанный класс (-1 или 1)

- w - вектор весов

- x - вектор признаков

- b - смещение (bias)

- sign - функция знака, возвращающая -1 для отрицательных чисел и 1 для положительных чисел

*Объяснение:*

- SVM находит гиперплоскость, которая максимизирует зазор между классами в пространстве признаков.

- Оптимальная гиперплоскость определяется опорными векторами, которые являются ближайшими точками данных к гиперплоскости.

- SVM могут использовать ядерные функции (например, RBF или полиномиальное ядро) для неявного преобразования данных в пространство более высокой размерности, что позволяет находить нелинейные разделяющие гиперплоскости.

- SVM хорошо работают с высокоразмерными данными и могут быть эффективны даже при небольшом количестве обучающих примеров.

**Виды нейронных сетей для обработки последовательностей**

1. **Сверточные нейронные сети (CNN):**

- CNN специально разработаны для обработки данных с сеточной топологией, таких как изображения.

- Они состоят из сверточных слоев, которые применяют операцию свертки к входным данным с использованием обучаемых фильтров (ядер), за которыми следуют нелинейные функции активации (например, ReLU).

- Сверточные слои позволяют извлекать локальные признаки из входных данных и обеспечивают инвариантность к переносу и масштабированию.

- За сверточными слоями обычно следуют слои подвыборки (например, максимальная подвыборка), которые уменьшают пространственные размеры и обеспечивают инвариантность к небольшим сдвигам и искажениям.

- CNN также включают в себя полносвязные слои для классификации или регрессии на основе извлеченных признаков.

- CNN успешно применяются в различных задачах компьютерного зрения, таких как классификация изображений, обнаружение объектов и семантическая сегментация.

*Алгоритм:*

- Операция свертки: output[i, j] = sum(input[i+m, j+n] \* kernel[m, n])

- output[i, j] - значение выходного пикселя на позиции (i, j)

- input[i+m, j+n] - значение входного пикселя на позиции (i+m, j+n)

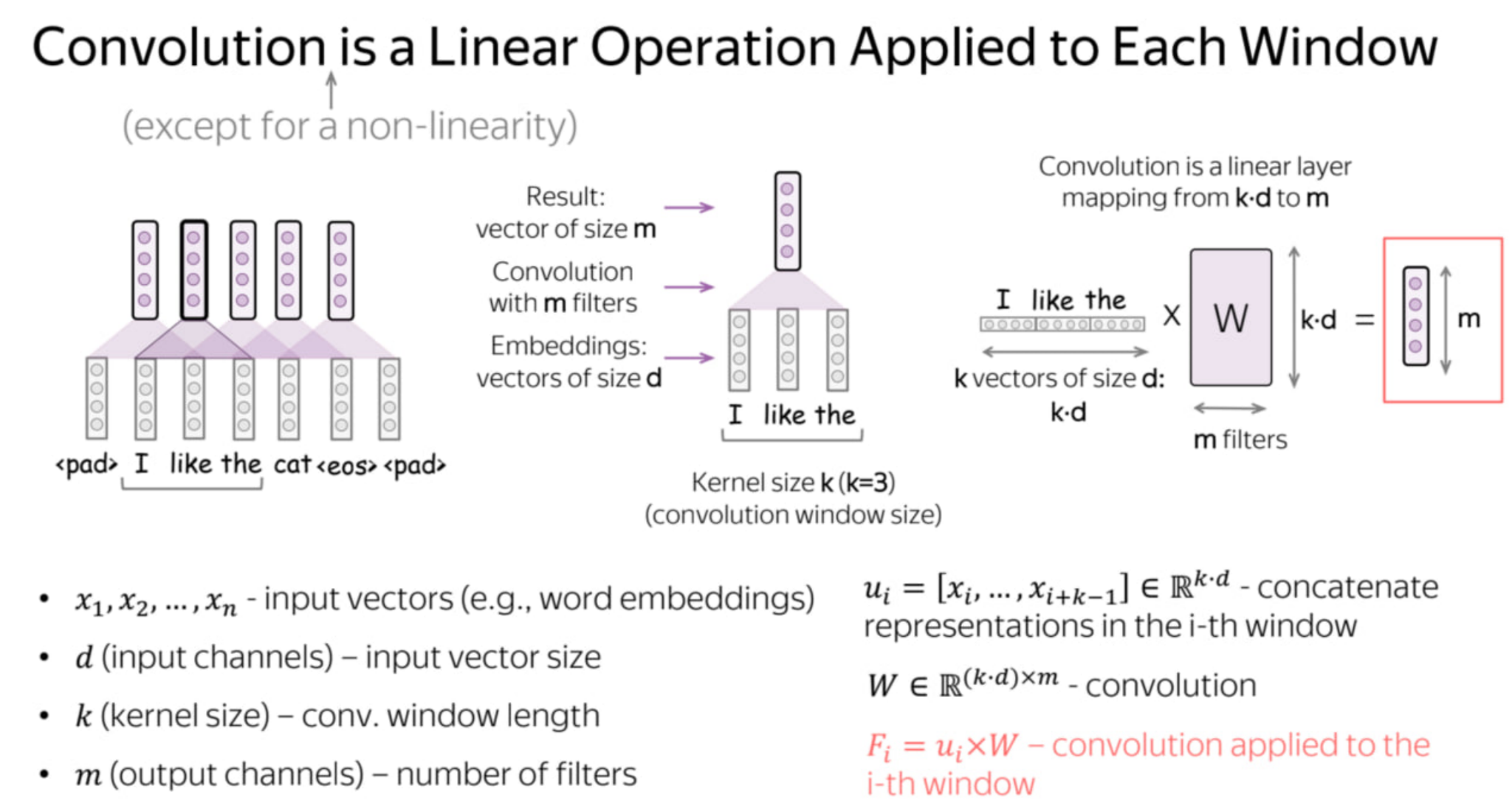
- kernel[m, n] - значение ядра (фильтра) на позиции (m, n)

- Функция активации ReLU: f(x) = max(0, x)

- Максимальная подвыборка: output[i, j] = max(input[i\*s:i\*s+k, j\*s:j\*s+k])

- s - шаг подвыборки

- k - размер окна подвыборки



2. **Рекуррентные нейронные сети (RNN):**

- RNN предназначены для обработки последовательных данных, таких как временные ряды или естественный язык.

- Они имеют рекуррентные связи, которые позволяют сети сохранять информацию о предыдущих входных данных и использовать ее для обработки текущего ввода.

- В простейшей форме RNN состоят из рекуррентного слоя, который применяет одну и ту же операцию к каждому элементу последовательности, передавая скрытое состояние от одного временного шага к другому.

- Однако простые RNN страдают от проблемы исчезающего или взрывающегося градиента, что затрудняет обучение долгосрочных зависимостей.

*Алгоритм:*

- Простой RNN: h[t] = f(W[hh] \* h[t-1] + W[xh] \* x[t] + b)

- h[t] - скрытое состояние на временном шаге t

- h[t-1] - скрытое состояние на предыдущем временном шаге t-1

- x[t] - входной вектор на временном шаге t

- W[hh], W[xh] - матрицы весов

- b - вектор смещения

- f - нелинейная функция активации (например, tanh или ReLU)

3. **Блоки долгой краткосрочной памяти (LSTM):**

- LSTM - это специальный тип RNN, разработанный для решения проблемы исчезающего градиента и улучшения способности сети изучать долгосрочные зависимости.

- Блок LSTM состоит из ячейки памяти, входных ворот, выходных ворот и ворот забывания.

- Ячейка памяти сохраняет информацию на протяжении длительных периодов времени, а ворота контролируют поток информации в ячейку и из нее.

- Входные ворота определяют, какую новую информацию следует сохранить в ячейке памяти, выходные ворота определяют, какую информацию следует выводить из ячейки, а ворота забывания определяют, какую информацию следует удалить из ячейки.

- LSTM успешно применяются в таких задачах, как моделирование языка, машинный перевод и распознавание речи.

*Алгоритм:*

- Входные ворота: i[t] = sigmoid(W[xi] \* x[t] + W[hi] \* h[t-1] + b[i])

- Ворота забывания: f[t] = sigmoid(W[xf] \* x[t] + W[hf] \* h[t-1] + b[f])

- Выходные ворота: o[t] = sigmoid(W[xo] \* x[t] + W[ho] \* h[t-1] + b[o])

- Кандидат на состояние ячейки: c\_tilde[t] = tanh(W[xc] \* x[t] + W[hc] \* h[t-1] + b[c])

- Состояние ячейки: c[t] = f[t] \* c[t-1] + i[t] \* c\_tilde[t]

- Скрытое состояние: h[t] = o[t] \* tanh(c[t])

4. **Управляемые рекуррентные блоки (GRU):**

- GRU - это упрощенная версия LSTM, которая объединяет входные и выходные ворота в единый управляющий вентиль и не имеет отдельной ячейки памяти.

- GRU состоит из управляющих ворот и ворот сброса. Управляющие ворота определяют, какую информацию следует передавать дальше, а ворота сброса определяют, какую информацию следует забыть из предыдущего скрытого состояния.

- GRU проще и быстрее в обучении, чем LSTM, при этом обеспечивая сопоставимую производительность во многих задачах.

- GRU часто используются в задачах обработки естественного языка, таких как моделирование языка, машинный перевод и анализ тональности текста.

Управляемые рекуррентные блоки (GRU):

*Алгоритм:*

- Управляющие ворота: z[t] = sigmoid(W[xz] \* x[t] + W[hz] \* h[t-1] + b[z])

- Ворота сброса: r[t] = sigmoid(W[xr] \* x[t] + W[hr] \* h[t-1] + b[r])

- Кандидат на скрытое состояние: h\_tilde[t] = tanh(W[xh] \* x[t] + W[hh] \* (r[t] \* h[t-1]) + b[h])

- Скрытое состояние: h[t] = (1 - z[t]) \* h[t-1] + z[t] \* h\_tilde[t]

В этих формулах W обозначает матрицы весов, b - векторы смещения, \* - поэлементное умножение, а sigmoid и tanh - нелинейные функции активации.

5. **State Space Models (SSM):**

1. Основные компоненты:

- Состояние системы (x)

- Входные данные (u)

- Выходные данные (y)

2. Ключевые уравнения:

- Уравнение состояния: x(t+1) = Ax(t) + Bu(t)

- Уравнение выхода: y(t) = Cx(t) + Du(t)

Где A, B, C, D - матрицы параметров модели.

3. Зависимости:

- Текущее состояние зависит от предыдущего состояния и текущего входа

- Выход зависит от текущего состояния и текущего входа

4. Непрерывное представление:

dx/dt = Ax(t) + Bu(t)

y(t) = Cx(t) + Du(t)

6. **Mamba:**

Mamba основана на SSM, но вводит несколько ключевых инноваций:

1. Селективный механизм состояний:

- Использует функцию выбора S(x), которая определяет, какие части состояния обновлять

- Уравнение состояния модифицируется: x(t+1) = S(x(t)) ⊙ (Ax(t)) + (1 - S(x(t))) ⊙ x(t) + Bu(t)

Где ⊙ обозначает поэлементное умножение

2. Структура блока Mamba:

- Входной слой: линейное преобразование входа

- SSM слой: обработка последовательности с использованием модифицированного SSM

- Выходной слой: еще одно линейное преобразование

3. Ключевые зависимости:

- Состояние зависит от предыдущего состояния, входа и функции выбора

- Функция выбора зависит от текущего состояния и входа

- Выход зависит от обработанного состояния

4. Оптимизации:

- Использование быстрого алгоритма для вычисления экспоненты матрицы

- Эффективная параллельная обработка на GPU

5. Обучение:

- Используется обратное распространение ошибки через время (BPTT)

- Градиенты вычисляются с учетом селективного механизма

*Ключевые особенности Mamba:*

1. Линейная сложность: O(n) по отношению к длине последовательности, в отличие от O(n^2) у трансформеров.

2. Эффективное использование памяти: благодаря селективному обновлению состояний.

3. Способность обрабатывать длинные последовательности: эффективна для последовательностей длиной до миллиона токенов.

4. Адаптивность: селективный механизм позволяет модели адаптироваться к различным типам входных данных.

5. Сочетание с другими техниками: Mamba может быть интегрирована с другими методами глубокого обучения.

**Виды позиционного кодирования**

1. **Sinusoidal Positional Encoding** (Синусоидальное позиционное кодирование):

- Это оригинальный метод, предложенный в статье "Attention Is All You Need".

- Позиционное кодирование вычисляется с использованием синусоидальных функций разных частот.

*Формула:*

PE(pos, 2i) = sin(pos / 10000^(2i/d\_model))

PE(pos, 2i+1) = cos(pos / 10000^(2i/d\_model))

где pos - позиция токена, i - измерение (индекс) позиционного кодирования, а d\_model - размерность эмбеддинга.

2. **Learned Positional Encoding (Обучаемое позиционное кодирование):**

- В этом подходе позиционные эмбеддинги являются обучаемыми параметрами модели.

- Каждой позиции соответствует свой уникальный эмбеддинг, который обновляется в процессе обучения.

*Формула:*

PE = Embedding(pos)

где pos - позиция токена, а Embedding - обучаемая матрица эмбеддингов.

3. **Relative Positional Encoding (RoPE) (Относительное позиционное кодирование)**:

- RoPE учитывает относительные позиции между токенами вместо абсолютных позиций.

- Он использует синусоидальные функции для вычисления относительных позиционных эмбеддингов.

*Формула:*

RoPE(pos\_q, pos\_k, 2i) = sin((pos\_q - pos\_k) / 10000^(2i/d\_model))

RoPE(pos\_q, pos\_k, 2i+1) = cos((pos\_q - pos\_k) / 10000^(2i/d\_model))

где pos\_q и pos\_k - позиции запроса (query) и ключа (key) соответственно.

4. **ALiBi (Attention with Linear Biases) (Внимание с линейным смещением):**

- ALiBi добавляет линейное смещение к вниманию (attention) на основе относительных позиций.

*Формула:*

ALiBi(pos\_q, pos\_k) = -|pos\_q - pos\_k| / m

где pos\_q и pos\_k - позиции запроса и ключа соответственно, а m - гиперпараметр, контролирующий силу смещения.

**Attention**

Глубоко эту тему не стала расписывать. Ниже слайдики из курса ШАДа. По сути мы определяем влияние одного эмбединга относительно другого. Q – место в последовательности где стоим, K и V – куда смотрим

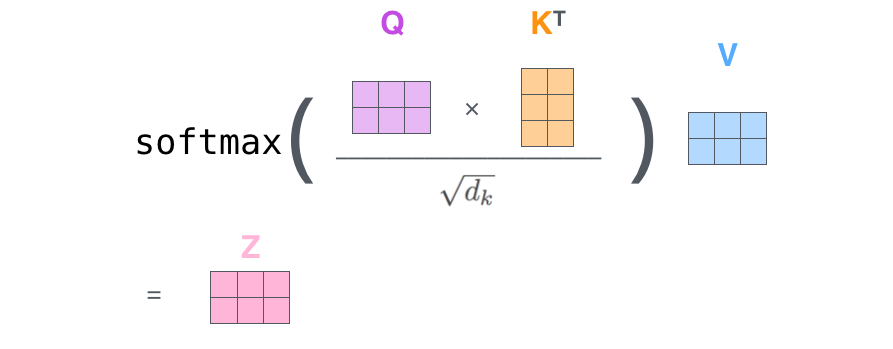
**Для более детального понимания ролей матриц:**

- Q (query): текущая позиция, для которой мы вычисляем внимание

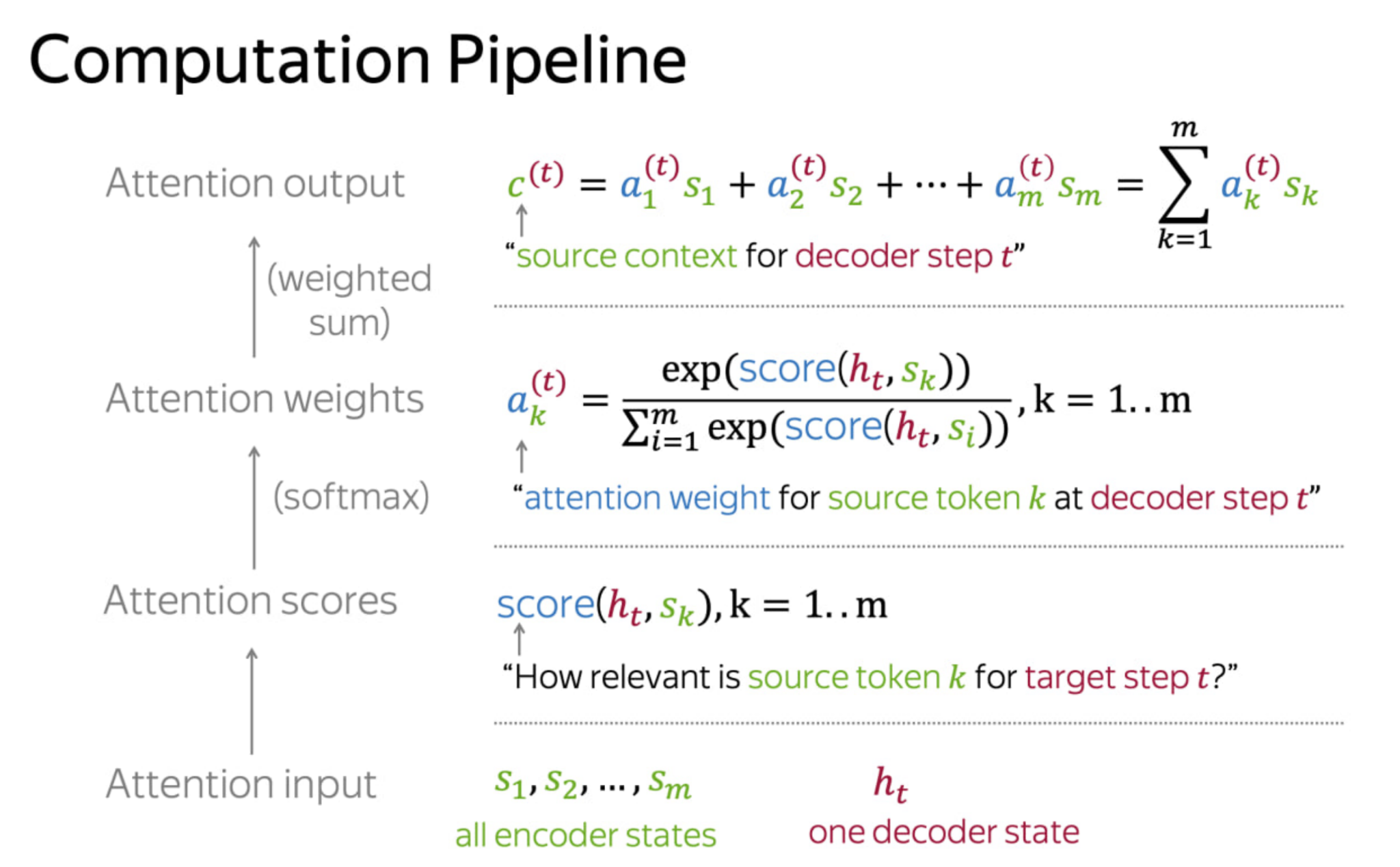
- K (key): используется для сравнения с Q для определения релевантности

- V (value): фактическая информация, которую мы извлекаем

Формула в трансформере:



*Для seq2seq объяснение из курса ШАДа:*



**Отличия моделей на основе трансформера:**

1. BERT (Bidirectional Encoder Representations from Transformers):

encoder-only 2018

- Двунаправленное обучение

- Использует маскированное языковое моделирование и предсказание следующего предложения

- Предобучен на большом корпусе текстов

2. RoBERTa (Robustly Optimized BERT Approach):

encoder-only 2019

- Улучшенная версия BERT

- Более длительное обучение на большем объеме данных

- Удален этап предсказания следующего предложения

- Динамическое маскирование

3. ALBERT (A Lite BERT):

encoder-only 2019

- Облегченная версия BERT

- Разделение параметров между слоями

- Факторизация матрицы встраивания

- Меньше параметров, но сопоставимая производительность

4. DistilBERT:

encoder-only 2019

- Дистиллированная версия BERT

- Меньше слоев и параметров

- Быстрее, но с небольшой потерей в производительности

5. XLNet:

decoder-only 2019

- Использует авторегрессивное предобучение

- Учитывает все возможные перестановки токенов

- Лучше справляется с длинными зависимостями

6. ELECTRA:

encoder-only 2020

- Использует дискриминативное предобучение вместо маскированного языкового моделирования

- Более эффективное использование вычислительных ресурсов

7. T5 (Text-to-Text Transfer Transformer):

encoder-decoder 2019

- Унифицированный подход к различным задачам NLP

- Рассматривает все задачи как преобразование текста в текст

8. GPT (Generative Pre-trained Transformer):

decoder-only 2018

- Однонаправленное (слева направо) обучение

- Автореггресивное языковое моделирование

- Хорошо подходит для генеративных задач

10. UniLM (Unified Language Model):

flexible architecture that can be used in various configurations 2019

- Объединяет однонаправленное, двунаправленное и seq2seq моделирование

- Может использоваться для широкого спектра задач

**Методы расширения возможностей и анализа нейронных сетей: NTK и интерполяция**

(тут мне уже просто было весело. Сомневаюсь, что это можно назвать базой)

**Neural Tangent Kernel (NTK)** - это теоретический инструмент, который связывает поведение нейронных сетей с ядрами (kernels) в теории машинного обучения. Вот как работает NTK:

1. Бесконечная ширина нейронной сети:

- NTK рассматривает нейронную сеть с бесконечной шириной, то есть с бесконечным количеством нейронов в каждом слое.

- В пределе бесконечной ширины поведение нейронной сети становится более простым и математически обоснованным.

2. Линеаризация около начальной инициализации:

- NTK линеаризует нейронную сеть вокруг ее начальной инициализации.

- Это означает, что NTK аппроксимирует нейронную сеть линейной моделью, которая касается (является касательной) к функции, представленной нейронной сетью, в точке начальной инициализации.

3. Ядро (Kernel) NTK:

- NTK определяет ядро (kernel), которое описывает подобие между двумя входными точками данных.

- Ядро NTK вычисляется как скалярное произведение градиентов выходов нейронной сети относительно параметров в начальной инициализации.

- Интуитивно, ядро NTK измеряет, насколько похоже нейронная сеть реагирует на два входных примера в терминах изменения ее параметров.

4. Эволюция во время обучения:

- В процессе обучения нейронной сети с помощью градиентного спуска NTK остается фиксированным.

- Это означает, что во время обучения нейронная сеть эффективно остается в линейном режиме, определяемом NTK.

- Обучение нейронной сети можно рассматривать как оптимизацию в пространстве функций, индуцированном ядром NTK.

5. Свойства и выводы:

- NTK обеспечивает теоретическое понимание поведения нейронных сетей в пределе бесконечной ширины.

- Он связывает обучение нейронных сетей с ядрами и позволяет анализировать такие свойства, как сходимость, обобщение и выразительность.

- NTK также дает представление о том, как архитектура нейронной сети влияет на ее способность к обучению и представлению функций.

Преимущества NTK:

1. Теоретическое понимание: NTK обеспечивает математическую основу для анализа поведения нейронных сетей и связывает их с установленной теорией ядер.

2. Анализ сходимости и обобщения: NTK позволяет изучать свойства сходимости и обобщения нейронных сетей с использованием инструментов теории ядер.

3. Интерпретируемость: Ядро NTK дает интерпретируемую меру подобия между входными примерами на основе реакции нейронной сети.

*Однако у NTK есть и некоторые ограничения:*

1. Предположение о бесконечной ширине: NTK основан на предположении о бесконечной ширине нейронной сети, что может не всегда соответствовать практическим сценариям.

2. Линейное приближение: NTK использует линейное приближение нейронной сети, которое может не полностью охватывать нелинейное поведение реальных нейронных сетей.

3. Вычислительная сложность: Вычисление ядра NTK может быть вычислительно сложным для больших наборов данных и глубоких нейронных сетей.

**Интерполяция** - это способ увеличения контекста в моделях обработки последовательностей путем комбинирования представлений (embeddings) из разных слоев модели. Вот как это работает:

1. Прямой проход (Forward Pass):

- Входная последовательность пропускается через модель, и представления вычисляются на каждом слое.

- Представления на разных слоях отражают различные уровни абстракции и контекстной информации.

2. Интерполяция представлений:

- Представления из разных слоев интерполируются (комбинируются) с использованием взвешенной суммы.

- Веса интерполяции могут быть фиксированными или настраиваемыми во время обучения.

- Интерполированное представление объединяет информацию из разных слоев, охватывая более широкий контекст.

3. Использование интерполированных представлений:

- Интерполированные представления используются для дальнейшей обработки, такой как внимание (attention) или классификация.

- Поскольку интерполированные представления содержат информацию из разных слоев, они обеспечивают более богатый и разнообразный контекст для последующих вычислений.

Преимущества интерполяции по сравнению с использованием ядер нейронных сетей (Neural Tangent Kernel, NTK):

1. Гибкость:

- Интерполяция позволяет комбинировать представления из разных слоев гибким и настраиваемым образом.

- Веса интерполяции могут быть адаптированы к конкретной задаче и настроены во время обучения.

- NTK, с другой стороны, полагается на фиксированную kernel-функцию, которая менее гибка.

2. Учет нелинейностей:

- Интерполяция работает с представлениями, полученными из нелинейных преобразований в нейронной сети.

- Это позволяет учитывать сложные нелинейные зависимости и взаимодействия между токенами.

- NTK основан на линейном приближении нейронной сети и может упустить важные нелинейные свойства.

3. Вычислительная эффективность:

- Интерполяция может быть выполнена эффективно с использованием простых операций взвешенного суммирования.

- Она не требует вычисления больших матриц Грама или kernel-функций, как в случае с NTK.

- Это делает интерполяцию более масштабируемой и применимой к большим моделям и наборам данных.

4. Интеграция с существующими архитектурами:

- Интерполяция может быть легко интегрирована в существующие архитектуры, такие как трансформеры, без существенных изменений в модели.

- Она совместима с различными механизмами внимания и может использоваться совместно с другими методами увеличения контекста.

- NTK обычно требует отдельной kernel-функции и может быть сложнее интегрировать с существующими моделями.

Интерполяция представляет собой простой и эффективный способ увеличения контекста в моделях обработки последовательностей. Она обеспечивает гибкость, учитывает нелинейности и может быть легко интегрирована в существующие архитектуры. Хотя NTK имеет свои преимущества в некоторых сценариях, интерполяция часто является более практичным и масштабируемым подходом для увеличения контекста в реальных приложениях.