**Билет№1) Функции активации. Зачем нужны нелинейности. Почему отĸазались от сигмоиды.**

**Функция активации** определяет выходное значение нейрона в зависимости от результата взвешенной суммы входов и порогового значения.

**Ступенчатая функция активации**: . Проблема: разделяет 2 класса, а если несколько?

**Линейная функция активации**: Линейная функция представляет собой прямую линию и пропорциональна входу (то есть взвешенной сумме на этом нейроне). Такой выбор активационной функции позволяет получать спектр значений, а не только бинарный ответ. Можно соединить несколько нейронов вместе и, если более одного нейрона активировано, решение принимается на основе применения операции max (или softmax).

**Проблема**: **1)** Производная от  по*x* равна *с*. Это означает, что градиент никак не связан с x. Градиент является постоянным вектором, а спуск производится по постоянному градиенту. **2)** Рассмотрим связанные слои. Каждый слой активируется линейной функцией. Значение с этой функции идет в следующий слой в качестве входа, второй слой считает взвешенную сумму на своих входах и, в свою очередь, включает нейроны в зависимости от другой линейной активационной функции. Если все они по своей природе линейные, то финальная функция активации в последнем слое будет просто линейной функцией от входов на первом слое! Это означает, что два слоя (или N слоев) могут быть заменены одним слоем. Мы потеряли возможность делать наборы из слоев.

## Сигмоида: . Сигмоида выглядит гладкой и подобна ступенчатой функции. Во-первых, сигмоида — нелинейна по своей природе, а комбинация таких функций производит тоже нелинейную функцию. Теперь мы можем стэкать слои. Сигмоиду удобно применять для задач классификации благодаря свойству «прижимания» к асимптотам - для значений и «прижимается» к одной из асимптот, что позволяет делать чёткие предсказания классов. Также сигмоида даёт нормализацию выходного значения каждого нейрона, так как она ограничена двумя горизонтальными асимптотами  и  (например, это необходимо, когда итоговое значение слоя есть вероятномсть).

## Проблема: производная такой функции мала во всех точках, кроме сравнительно небольшого промежутка, что сильно усложняет процесс улучшения весов с помощью градиентного спуска. Более того, эта проблема усугубляется в случае, если модель содержит много слоёв; данная проблема называется проблемой исчезающих градиентов.  Нейросеть отказывается обучаться дальше или делает это крайне медленно.

## Гиперболический тангенс: - скорректированная сигмоидная функция. – сохраняет преимущества и недостатки сигмоиды, но уже для диапазона (-1,1). Если нет необходимости в нормализации, то эта функция предпочтительнее сигмоиды, так как её область определения центрирована относительно 0, что снимает ограничение при подсчёте градиента для перемещения в определённом направлении. Также производная гиперболического тангенса значительна выше вблизи нуля, что даёт большую амплитуду градиентному спуску более быструю сходимость.

## ReLU: . ReLU нелинейна по своей природе, а комбинация ReLU также нелинейна! Это означает, что мы можем стэкать слои. Здесь легко считается производная: 1 – для положительных значений, 0 – для отрицательных значений. Также преимуществом является разрежённость активации – не все нейроны активируются, сеть становится легче.

**Проблема**: есть недостаток, называемый проблемой умирающего ReLU. Так как часть производной функции равна нулю, то и градиент для неё будет нулевым веса не изменяются во время спуска и нейросеть перестаёт обучаться.

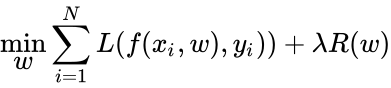
**Модификации ReLU**: – получаем ненулевой градиент для отрицательных значений, имеет недостатки: 1) сложнее считать производную по сравнению с ReLU; 2) угловой коэффициент – отдельный гиперпараметр, который нужно подбирать; 3) На практике результат не сильно лучше ReLU; 4) проблема затухания градиента в глубоких сетях всё ещё актуальна. – аналогичен , но угол наклона регулируется нейросетью.

**Билет№2) Регуляризация. Capacity модели. Dropout и BatchNorm. Дистилляция\*.**

Regularization, Dropout and Batch Normalization – методы борьбы с переобучением.

**Регуляризация** – это техника, ограничивающая модель в идеальной подгонке результатов, делающая её обощаемой.

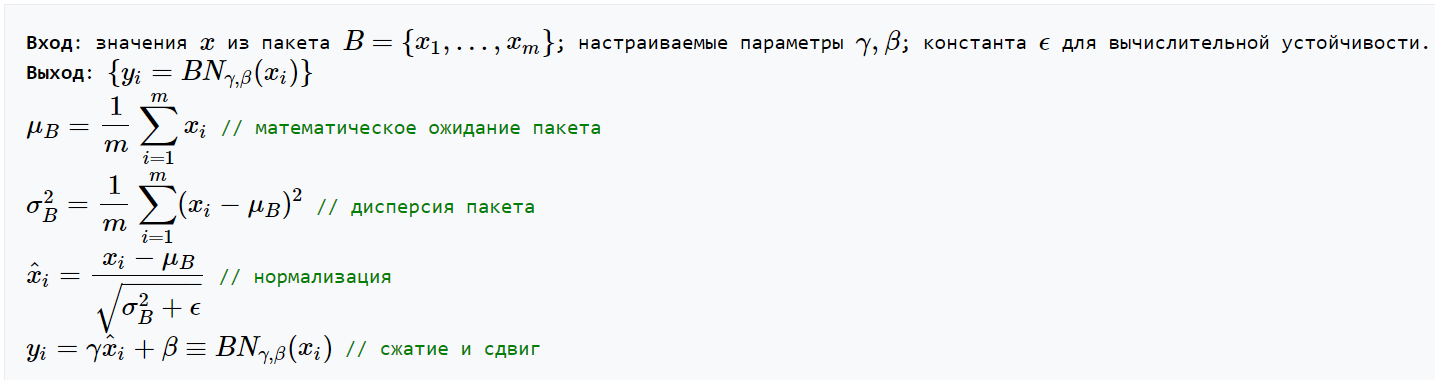
Список из пяти наиболее распространенных дополнительных **методов регуляризации**:

* **Регуляризация веса**: Оштрафовать модель во время тренировки на основе величины веса.  . L1, L2 регуляризация, их комбинация: подробнее почитать [здесь](https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning).
* **Ограничение веса**: Ограничьте величину весов, чтобы быть в пределах диапазона или ниже предела.
* **Шум**: Добавьте статистический шум к входам во время обучения.
* **Ранняя остановка**: Мониторинг производительности модели на проверочном наборе и остановка обучения при снижении производительности.

**Capacity модели**: Ёмкость модели — это её способность выполнять широкий спектр функций. Модель с меньшей ёмкостью может быть не в состоянии в достаточной степени изучить обучающий набор данных. Модель с большей ёмкостью может моделировать больше различных типов функций и может быть способна изучать функцию, чтобы в достаточной степени сопоставлять входные данные с выходными данными в обучающем наборе данных, но она может запомнить обучающий набор данных и не суметь обобщить, заблудиться или застрять в поиске подходящей функции отображения. Емкость нейронной сети может контролироваться двумя аспектами модели: 1) Количество узлов. 2) Количество слоев.

**BatchNorm**: Нормализация данных – это нормализация входных объектов таким образом, чтобы все объекты были в одном масштабе. Чтобы привести все значения активации к одной шкале, мы нормализуем значения активации таким образом, чтобы скрытое представление не менялось кардинально, а также помогало нам улучшить скорость обучения. Почему это называется пакетной нормализацией: поскольку мы вычисляем среднее и стандартное отклонение из одной партии, а не из всех данных. Пакетная нормализация выполняется индивидуально для каждого скрытого нейрона в сети.

Данные могут иметь различное мат ожидание и дисперсию, такая разница статических характеристик называется внешним ковариационным сдвигом (привести в пример цвета машин в обучающей выборке и на тестовой). Внутренним ковариационным сдвигом называется – это когда статическая картинка меняется внутри сети от слоя к слою. Задача Batch Normalization – уменьшение внутреннего ковариационного сдвига в распределении признаков через их стандартизирование (N(0,1)). Вышеописанная нормализация может изменить представление данных, созданную в предыдущем слое, поэтому вводятся два параметра сжатия и сдвига нормализованной величины, действующие как обучаемые параметры для денормализации входных данных



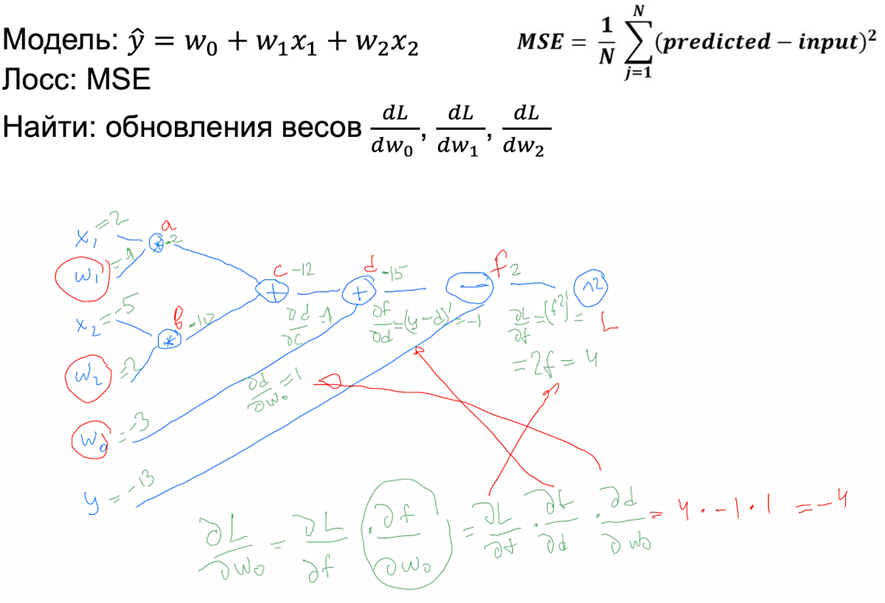
**Dropout**: — это метод регуляризации, который “выбрасывает” или “деактивирует” несколько нейронов в нейронной сети случайным образом, чтобы избежать проблемы переобучения. Таким образом, применяя отсев, мы можем смоделировать ансамбль нейронной сети с различными архитектурами и уменьшить сложность модели.

**Дистилляция** — это метод перегонки знаний из большой нейронной сети в меньшую нейронную сеть, как правило, позволяющий достичь аналогичной производительности (т.е. точности) с меньшей сетью. Этот метод работает так, что у вас есть две модели, одна из которых является большой моделью (моделью учителя) и меньшей моделью (моделью студента). На этапе обучения модель студента пытается предсказать значения и изменяет свои веса на основе потери, которая сочетает в себе как ошибку прогнозирования модели студента, так и ошибку между предсказанием модели учителя и модели студента. Идея заключается в том, что таким образом модель учителя помогает направлять модель студента, чтобы сделать правильные прогнозы.

**Билет№3) Backpropagation. Вычислительные графы. Дополнительная память при обучении. Вывод формул для градиента параметров и входа на примере линейного слоя.**

1) **Метод обратного распространения ошибки** (backpropagation) - метод вычисления градиента, который используется при обновлении весов многослойного перцептрона. Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении обратном прямому распространению сигналов в обычном режиме работы.

2) **Вычислительный граф** — это иллюстрированная запись какой-либо функции, состоящая из вершин и рёбер. **Вершины** (иногда их ещё называют узлы) — вычислительные операции, которые необходимо выполнить, а **рёбра** связывают их в определённую последовательность.



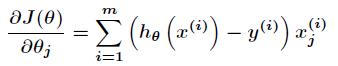
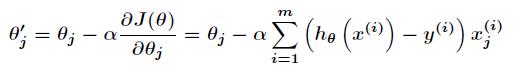
**Преимущество графов** в том, что они позволяют использовать так называемый метод обратного распространения ошибки (backpropagation). Этот алгоритм рекурсивно использует правило дифференцирования сложной функции для вычисления градиента каждой переменной в графе. Метод становится очень полезным для действительно сложных функций, которые применяются в свёрточных нейросетях.

3) **Большие объемы** занимаемой памяти – одна из самых больших проблем глубинных нейронных сетей. Нейронным сетям нужна память для того, чтобы хранить входные данные, весовые параметры и функции активации, как вход распространяется через сеть. В обучении активация на входе должна сохраняться до тех пор, пока ее нельзя будет использовать, чтобы вычислить погрешности градиентов на выходе.

**Дополнительная память** также требуется для хранения входных данных, временных значений и инструкций программы, а также градиенты скаляров.

**Билет№4) Оптимизация. Стохастичесĸий градиентный спусĸ. Adam, RMSProp, моментум:**

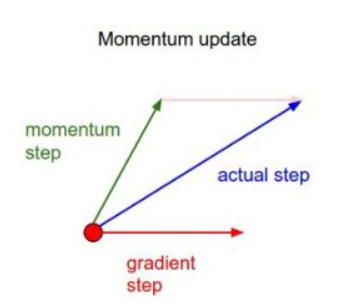
1. **Градиент функции**— это направление наискорейшего роста функции, а градиент, взятый с минусом – это направление наискорейшего убывания. Обучение нейронной сети  —  это такой процесс, при котором происходит подбор оптимальных параметров модели, с точки зрения минимизации функционала ошибки. Иными словами, осуществляется поиск параметров функции, на которой достигается минимум функционала ошибки, с помощью градиентного спуска.
2. **Виды градиентного спуска**:
3. *Пакетный градиентный спуск* - при этом подходе градиент функционала обычно вычисляется как сумма градиентов, учитывая каждый элемент обучения сразу. Это хорошо работает в случае выпуклых и относительно гладких функционалов, как например в задаче [линейной](http://www.machinelearning.ru/wiki/index.php?title=%D0%A0%D0%B5%D0%B3%D1%80%D0%B5%D1%81%D1%81%D0%B8%D1%8F) или логистической регрессии. Поверхность, задаваемая функционалом ошибки нейронной сети, зачастую негладкая и имеет множество локальных экстремумов, в которых мы обречены застрять, если двигаться пакетным градиентным спуском. Также при большом числе выборок процесс обучения будет медленным.

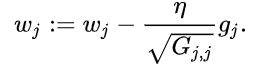
Функция потерь: . Итерационная формула пакетного градиентного спуска: .

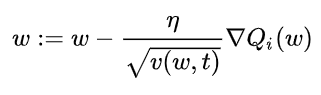
2. *Стохастический градиентный спуск (SGD)* - этот подход подразумевает корректировку весов нейронной сети, используя аппроксимацию градиента функционала, вычисленную на одном подмножестве из полного набора. Метод привносит «шум» в процесс обучения, что позволяет (иногда) избежать локальных экстремумов. Также в этом варианте шаги обучения происходят чаще, и не требуется держать в памяти градиенты всех обучающих примеров.

**Модификации SGD**:

1) **SGD с импульсом** (SGD with momentum) на каждой новой итерации оптимизации используется скользящее среднее градиента. Движение в направлении среднего прошлых градиентов добавляет в алгоритм оптимизации эффект импульса, что позволяет скорректировать направление очередного шага, относительно исторически доминирующего направления.



2) **Adagrad (*adaptive gradient algorithm*)** - базовая идея здесь в том, что некоторые параметры могут быстрее достигать своего оптимума, чем другие. Метод устанавливает разные скорости обучения для разных параметров, основываясь на прошлых градиентах, рассчитанных для каждого параметра. На каждом временном шаге t метод Адаграда выбирает различную скорость обучения для каждого параметра, обновляет соответствующие параметры и затем выполняет векторизацию. То есть, те параметры, что близки к оптимуму менять с меньшим шагом, а другие – с большим. Считается, что это заметно ускоряет сходимость при сильно разреженных данных. Вектор обновления: 

4) **RMSProp** (сейчас Adadelta это тоже самое) – Идея заключается в делении скорости обучения для весов на [скользящие средние](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D0%BE%D0%BB%D1%8C%D0%B7%D1%8F%D1%89%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D1%8F%D1%8F) значения недавних градиентов для этого веса. Параметры обновляются так: 

5) **Adam** – улучшенный RMSProp, использующий сглаженные версии среднего и среднеквадратического градиентов. Самый часто используемы сейчас.

**Билет№5) Свёрточные сети. Представление изображения. Операция свёртĸи. Residual connection.**

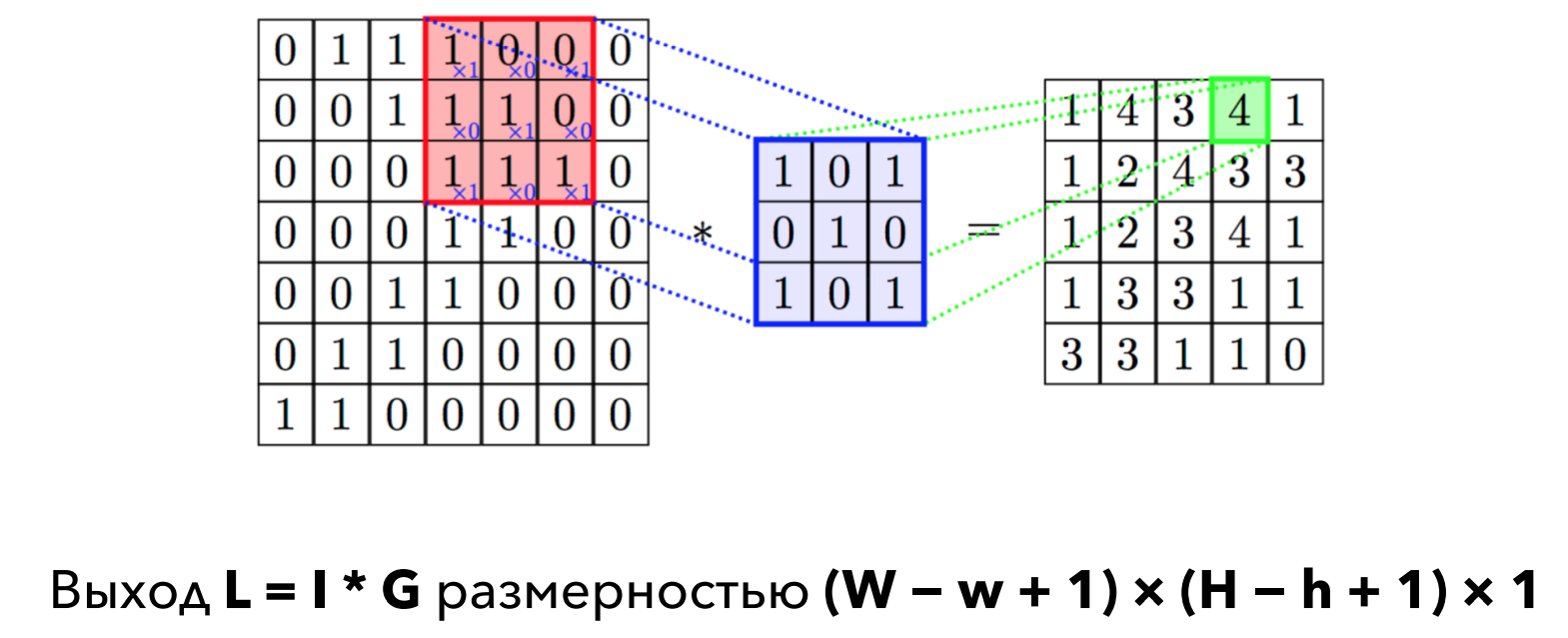
**Представление изображений:** чтобы упросить процесс создания признаков, мы можем представить изображение в табличной форме, то есть когда каждый пиксель преобразуется в признак. Однако результат неутешительный: не остаётся практически никакой информации, которую может использовать нейросеть/алгоритм МО — отсюда плохая производительность. Чтобы автоматизировать извлечение признаков придумали сверточные нейронные сети (Convolutional neural network).

**Свёрточная нейронная сеть** (ConvNet/CNN) — это просто нейронные сети, в которых вместо общеӗ операции умножения на матрицу, по крайней̆ мере в одном слое, используется операция свёртки. Отличие от простого разглаживания изображение, состоит в том, что сеть CNN способна с успехом схватывать пространственные и временные зависимости в изображении через применение соответствующих фильтров.

Можно представить, насколько интенсивными будут вычисления, когда изображения достигнут размеров, например, 8 K (76804320). Роль CNN заключается в том, чтобы привести изображения в форму, которую легче обрабатывать, без потери признаков, имеющих решающее значение в получении хорошего прогноза.

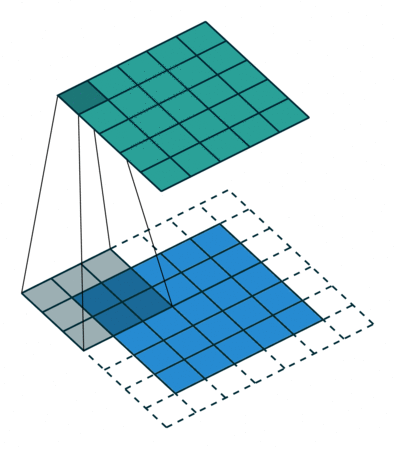
У CNN есть различные архитектуры, сыгравшие ключевую роль в построении алгоритмов, на которых стоит и в обозримом будущем будет стоять искусственный интеллект в целом: **LeNet, AlexNet, VGGNet, GoogLeNet, ResNet.**

Свёрточный слой представляет из себя набор карт, у каждой карты есть ядро (или синаптическое ядро, или сканирующее ядро, или фильтр). Операция свёртки выглядит так:

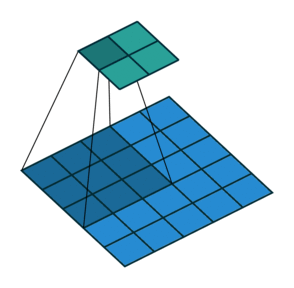


Параметры свёртки:

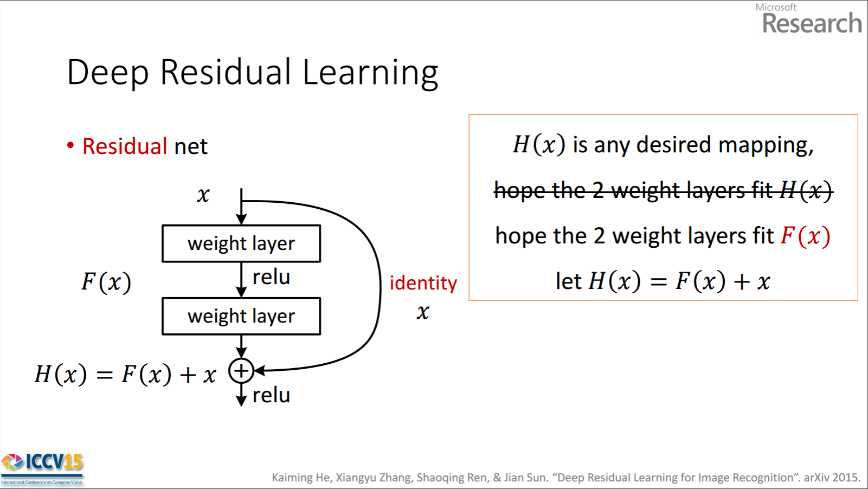
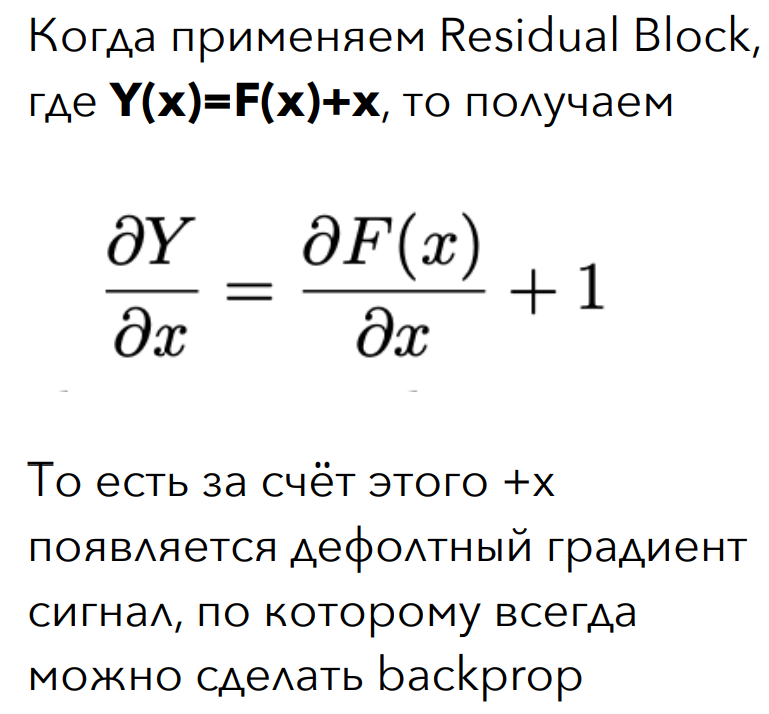
* Количество ядер.
* Размерность ядра.
* **Padding** - добавляет к краям поддельные (fake) пиксели (обычно нулевого значения).



* **Striding** - пропускает некоторые области, над которыми скользит ядро. На картинке можно увидеть свёртку со stride=2:

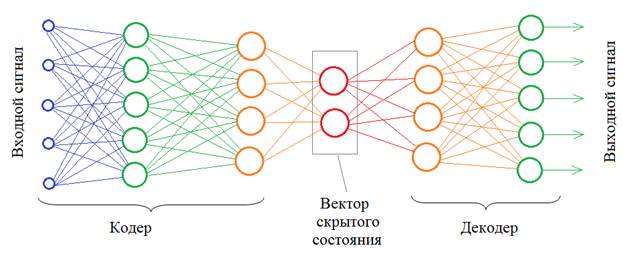


**Residual connection:** появилось как решение проблемы затухания градиентов в 2015. Теперь к производные каждые несколько блоков прибавляется 1.

**Билет№6) Автоэнĸодеры. Морфинг. Генерация ĸартиноĸ. Вариационные автоэнĸодеры\***

1) **Автоэнкодер** – это нейросеть, которая сначала кодирует входной сигнал в некоторое скрытое (латентное) состояние, размерность которого, как правило, меньше размерности входного сигнала, а затем, из скрытого состояния снова разворачивает (декодирует) данные в другое, новое состояние:



**Отличительная особенность автоэнкодеров** — количество нейронов на входе и на выходе совпадает.

Автоэнкодер состоит из двух частей:

* **Энкодер**: отвечает за сжатие входа в latent-space.
* **Декодер**: предназначен для восстановления ввода из latent-space.

Практические задачи автоэнкодеров: удаление шума из изображения, повышение чёткости изображения; уменьшение размерности, кластеризация, визуализация.

2) **Морфинг** —плавный переход между объектами. Обозначим уже обученный на данных энкодер как функцию e, а декодер как функцию g. Тогда морфинг между изображениями A и B мы можем сделать так: переведём изображения A и B в скрытые состояния a=e(A) и b=e(B), а затем каждый кадр генерируется как

C=d((1−t)\*a+t\*b)

где t равномерно изменяется от 0 до 1. Иными словами, мы берём все точки на отрезке ab и последовательно декодируем.

3) **Вариационный автоэнкодер** - генеративная модель, которая находит применение во многих областях исследований: от генерации новых человеческих лиц до создания полностью искусственной музыки.

## Проблема классических автоэнкодеров: Всё замечательно, если вам необходимо просто воспроизвести исходные изображения. Однако, если вы строите генеративную модель, ваша цель не простое дублирование изображений. В этом случае вы хотите получить случайное или видоизмененное изображение, восстановленное из непрерывного скрытого пространства.

## 

## Вариационный автоэнкодер (VAE) имеет одно уникальное свойство: их скрытое пространство по построению является непрерывным, позволяя выполнять случайные преобразования и интерполяцию. Непрерывность скрытого пространства достигается неожиданным способом: энкодер выдаёт не один вектор размера n, а два вектора размера n – вектор средних значений µ и вектор стандартных отклонений σ.

## 

## Эта так называемая стохастическая генерация означает, что даже для одинаковых входных данных результат кодирования будет разным вследствие случайности выбора вектора кодирования

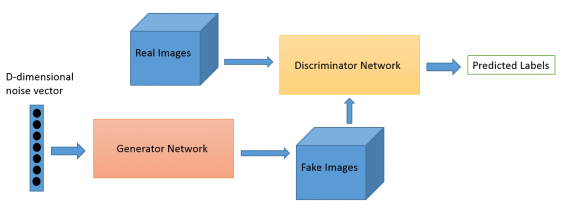
## 

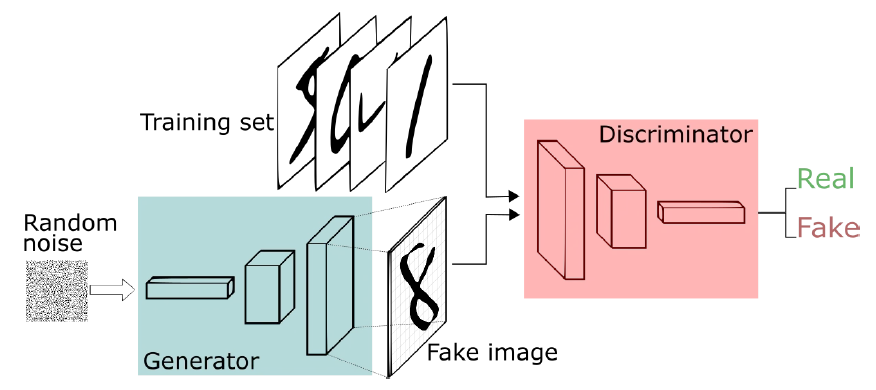
## Теперь модель обладает вариативностью даже в пределах одного вектора кодирования, так как скрытое пространство локально непрерывно, т.е. непрерывно для каждого образца входных данных.

## 

**Билет№7) GAN-ы. Фунĸция потерь. Каĸ оценивать ĸачество. Циĸлоганы\*.**

1) **Генеративно-состязательная нейросеть** (Generative adversarial network, [GAN](https://neurohive.io/ru/osnovy-data-science/gan-rukovodstvo-dlja-novichkov/)) — архитектура, состоящая из генератора и дискриминатора, настроенных на работу друг против друга. Отсюда GAN и получила название генеративно-созтязательный.



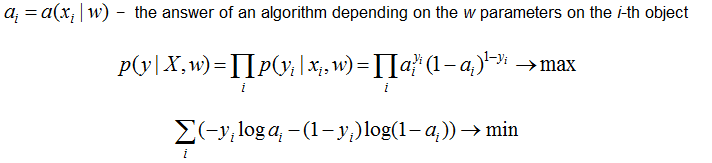


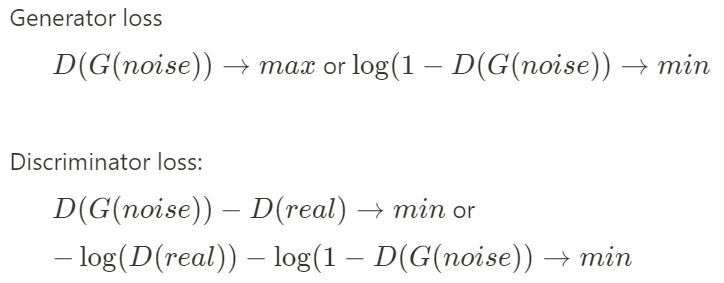
**Дискриминатор** пытается классифицировать входные данные, учитывая их особенности (например, понять, сгенерировано изображение или нет), то есть изучает границы между классами.

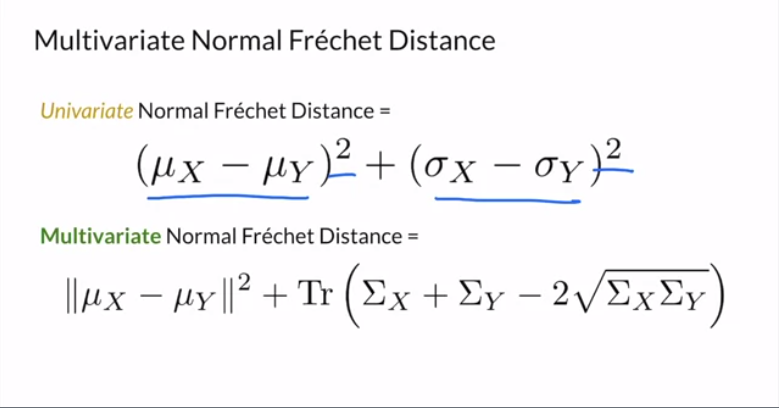
**Генеративные алгоритмы** заняты обратным: вместо того, чтобы предсказывать категорию по имеющимся образам, они пытаются подобрать образы к данной категории (например, генерируют изображение из рандомного шума), то есть моделируют распределение между классами.

**Как работает GAN:** Одна нейронная сеть, называемая генератором, генерирует новые экземпляры данных, а другая — дискриминатор, оценивает их на подлинность; т.е. дискриминатор решает, относится ли каждый экземпляр данных, который он рассматривает, к набору тренировочных данных или нет.

При обучении **дискриминатора** мы игнорируем потери генератора и используем только функцию потерь дискриминатора, которая штрафует дискриминатор за неверную классификацию, например, сгенерированных лиц. При обучении **генератора** мы используем функцию потерь генератора, штрафующую генератор за неудачный обман дискриминатора, то есть, например, за генерацию лица, которое дискриминатор признал фейковым.

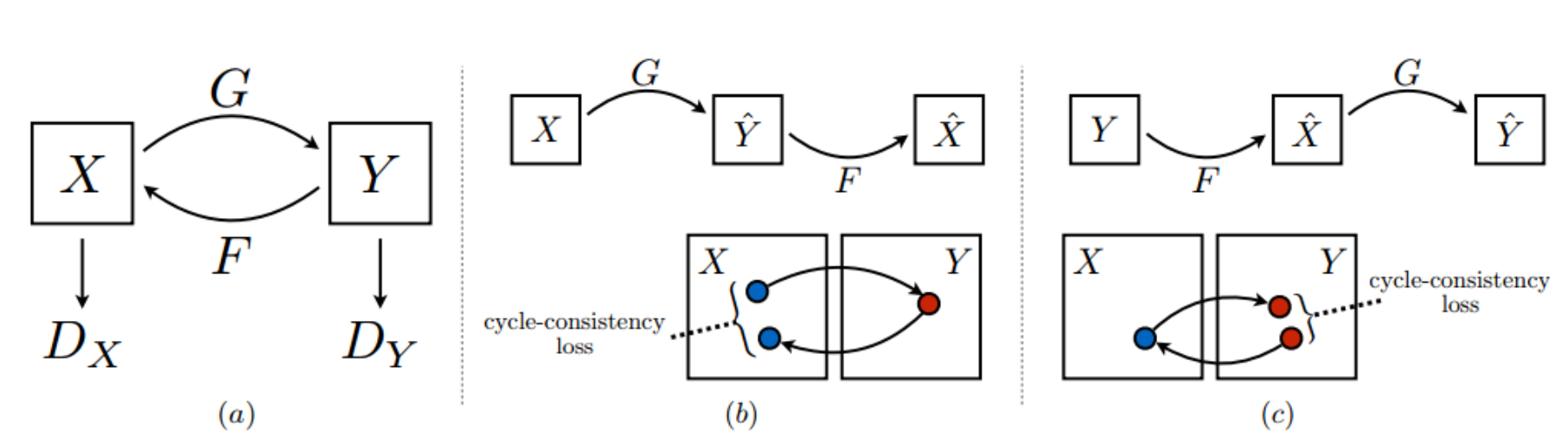
1. **Функция потерь:** Log Loss 

Например, для DCGAN: 

1. Как оценивать качество: метрика **FID**: 

Чем ниже FID, тем лучше качество. Другими словами, сходство между реальными и сгенерированными изображениями близко. FID сравнивает статистику сгенерированных выборок с реальными выборками, а не оценивает сгенерированные выборки в вакууме.

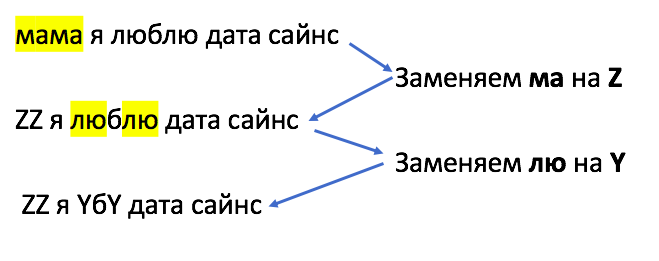
4) **Циĸлоганы** – состоит из двух обучаемых функций-генераторов. Одна — G — учится по входному изображению из домена X генерировать изображение из домена Y. Другая — F — наоборот, из Y в X. Соответствующие дискриминаторы Dy и Dx им в этом помогают, как это свойственно GAN'ам. Дополнительно авторы вводят так называемый Cycle Consistensy Loss: Суть его в том, чтобы изображение из домена X, пройдя через генератор G, а потом через генератор F, было максимально похоже на оригинал, то есть 



**Билет№8) Тоĸенизация и представление теĸста. Byte-pair-encoding\***

Токенизация – важный шаг в предварительной работе над текстом. Виды токенизации:

1. **Сhar-level tokenization**: для каждой буквы есть свой индекс. Минусы: текст после токенизации становится очень длинным, что усложняет обучение; не собрать потом нормальные слова из вероятностей букв.
2. **Word-based tokenization**: или для каждого слова есть свой индекс, или просто One-Hot-Encoding, CountVectoriser (- преобразовывает входной текст в матрицу, значениями которой, являются количества вхождения данного ключа(слова) в текст)
3. **Subword tokenization**: Основан на принципе, что часто используемые слова не должны быть разделены на более мелкие под слова, но редкие слова должны быть разложены на значимые под слова. Например, "annoyingly" может считаться редким словом и может быть разложено на "annoying" и "ly", то есть получим [‘\_annoying’, ‘ly’].
4. **Byte-pair encoding**.

.

Имеем следующий алгоритм:

Шаг 0. Создаём словарь.

Шаг 1. Представляем слова из текста как списки букв.

Шаг 2. Считаем количество вхождений каждой пары букв.

Шаг 3. Объединяем самые частотные в токен и добавляем в словарь.

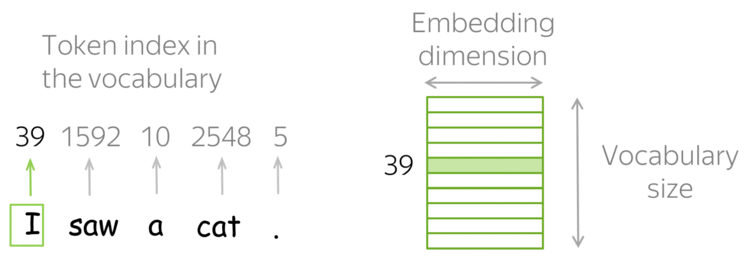
Шаг 4. Повторяем шаг 3 до тех пор, пока не получим словарь заданного размера.

1. **Модификация BPE – токенизатор WordPiece**.  Основная идея остается прежней: мы разбиваем слова на буквы и объединяем символы попарно. Однако в словарь добавляются не самые частотные пары, а пары, максимально увеличивающие вероятность на данных для обучения.

**Билет№9) Эмбеддинги. Методы обучения. Почему получаются геометричесĸие свойства. Fasttext\*.**

ML models не знает, что это за слово - мы должны как-то его закодировать, например, мы могли бы представить его в виде вектора (что и называется «embedding of word»).

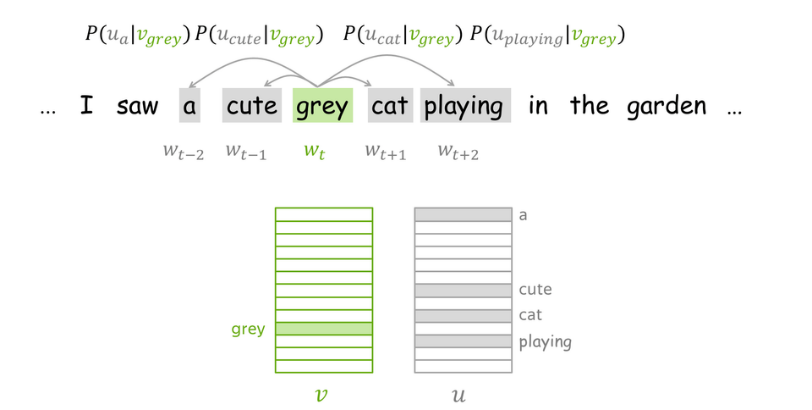
1. **Матрица эмбеддингов** — это матрица размера (слов в словаре, длина вектора), в которой i-тая строка - эмбеддинг i-того слова в словаре. Если слова нет в словаре - ему сопоставляется особый токен <UNK> (таких токенов может быть несколько, например, <START>, <END>, <PAD>, для них заранее резервируются места в словаре).



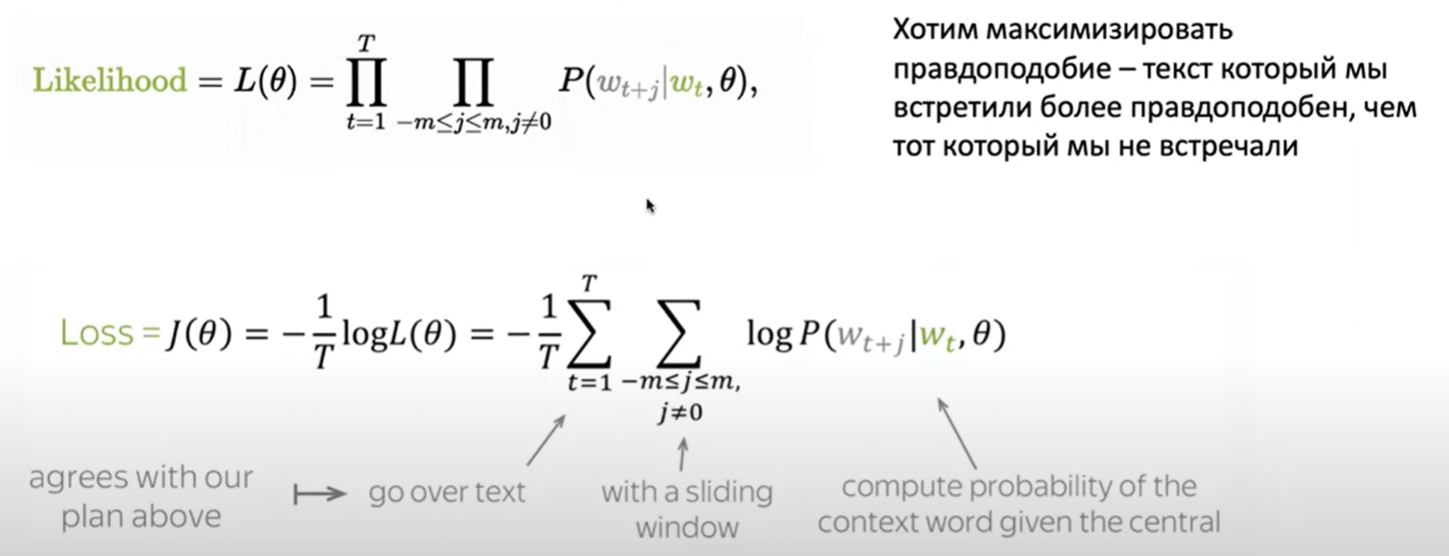
1. **Как обучаются эмбеддинги:** Основная идея: мы должны поместить информацию о контекстах в векторы слов.Методы, основанные на подсчёте, восприняли эту идею буквально, Word2Vec использует ее по-другому, как: изучите векторы слов, научив их предсказывать контексты.

**Word2Vec** — это итерационный метод. Его основная идея заключается в следующем:

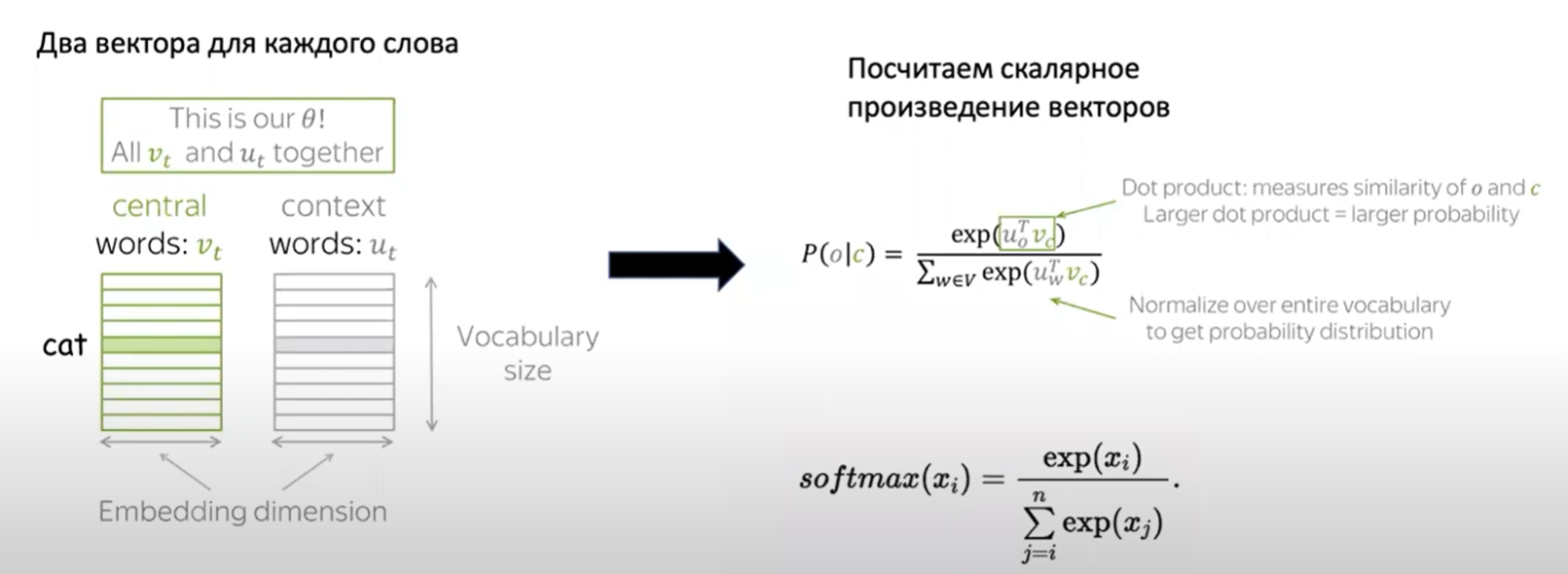
* взять огромный текстовый корпус
* пройтись по тексту с помощью скользящего окна, перемещая по одному слову за раз. На каждом шаге есть центральное слово и контекстные слова (- другие слова в этом окне)



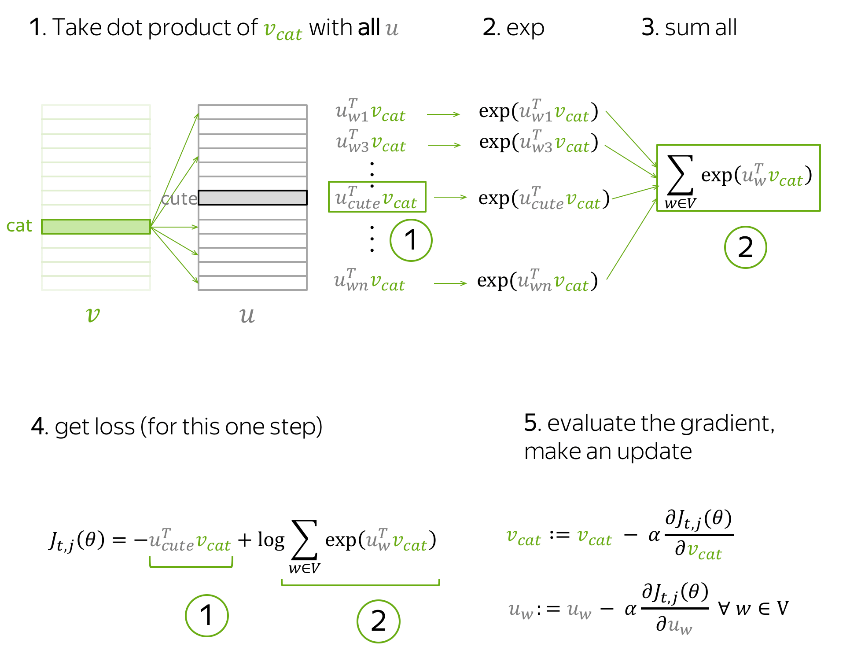
* для центрального слова вычислить вероятности контекстных слов
* отрегулировать векторы, чтобы увеличить эти вероятности, используя принцип максимума правдоподобия (чтобы вероятности контекстного слова в скользящем окне увеличивались, а вероятности других слов уменьшались – можно брать подмножество слов, чтобы обучение было быстрее (Negative Sampling))



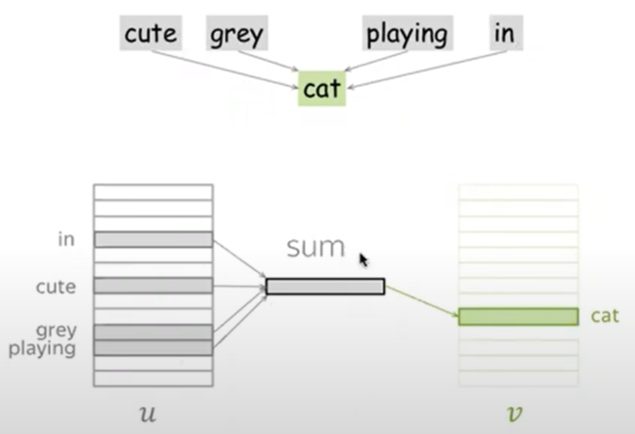
где вероятности будут считаться следующим образом (начальное задание векторов случайное)



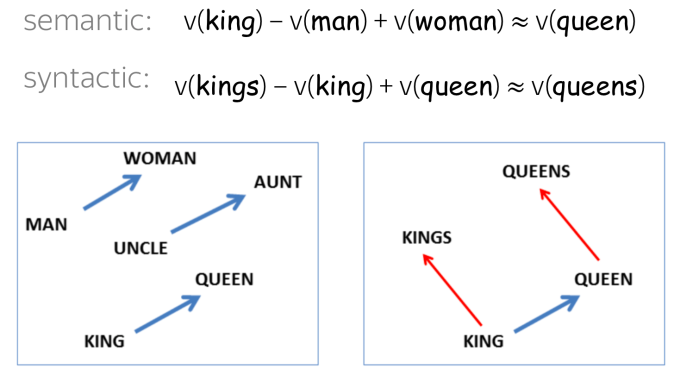
Ну и тогда получим следующий порядок действий



Из двух получившихся матриц оставляют одну (чаще с центральным словом). Это была Skip-Gram вариация Word2Vec, когда из центрального слова предсказывается контекст. Есть ещё вариация CBOW, обучающаяся аналогично, когда из суммы контекста предсказывается центральное слово:



1. **Геометрические свойства**: близкие по смыслу слова дают одинаковый контекст

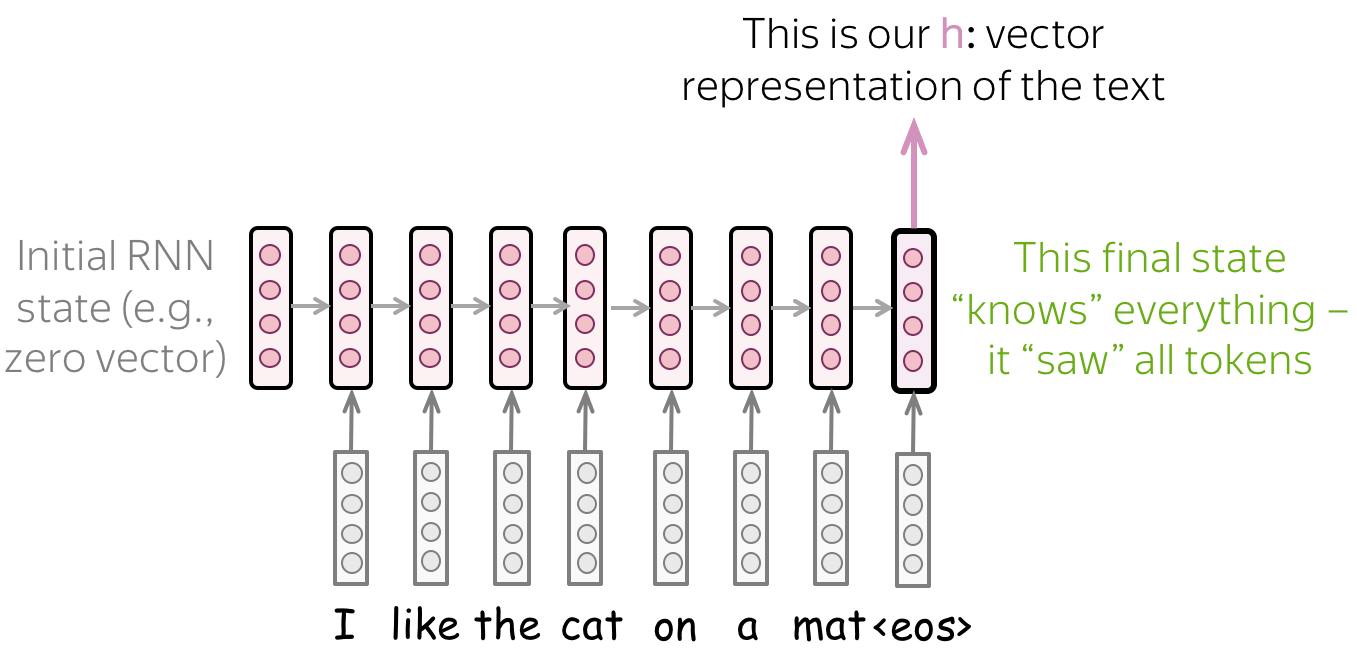


1. **FastText** - это библиотека для эффективного изучения представлений слов и классификации предложений. Эта библиотека переводит слова в векторное преобразование + может обучать модель для классификации разных типов текстов;

**Билет№10) Реĸуррентные сети. GRU, LSTM. Затухающий / взрывающийся градиент и способы с ним бороться.**

**1) Рекуррентные нейронные сети (RNN)** — это класс нейронных сетей, которые хороши для моделирования последовательных данных, таких как временные ряды или естественный язык.

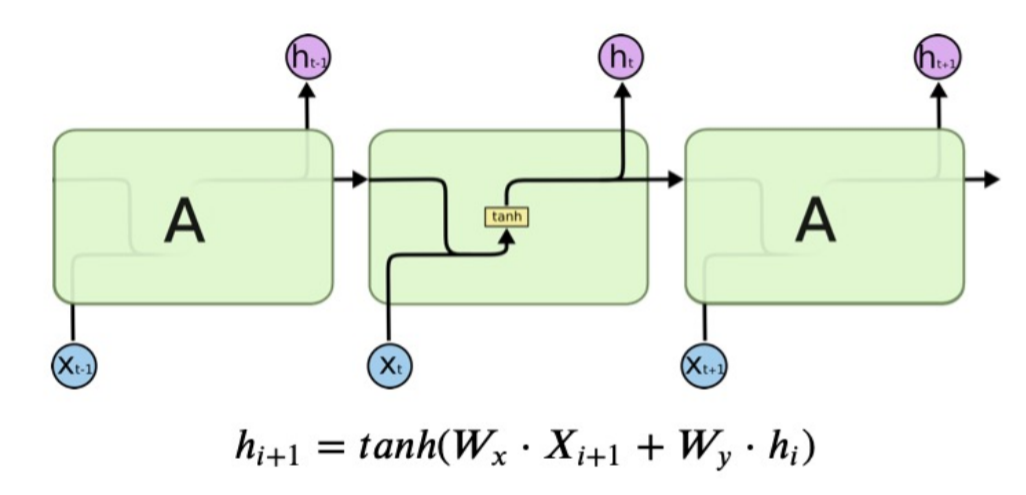
* позволяют обрабатывать последовательности событий/обьектов
* используют внутреннюю память для обработки последовательностей произвольной длины
* позволяют представлять события в виде одного вектора



Примеры:

* **Vanilla RNN**

Самый простой архитектура RNN - линейный слой + активация; Проблема - надо каждый раз умножать на одну и ту же 𝑊, и норма градиента может расти или убывать экспоненциально, могут возникнуть взрывающиеся и затухающие коэффициенты.



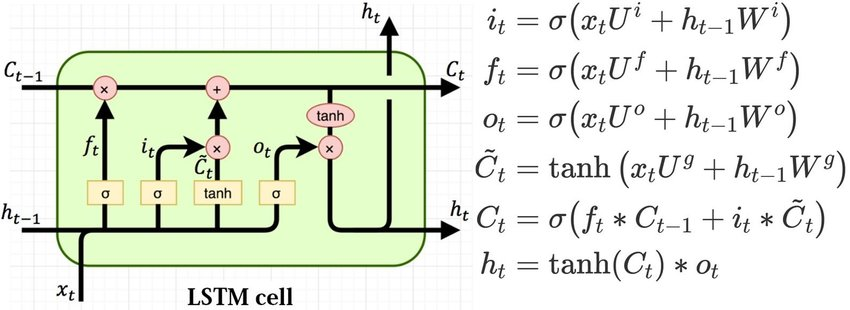
* **LSTM (Long Short Term Memory)**

На практике люди поняли, что RNN не способно на долгосрочное запоминание информации и придумали LSTM. **У нее есть свои слои**:

1) **Forget gate layer**. Сколько информации мы выкинем из cell state; Пример – встретили объект нового пола, надо забыть старый;

2) **Input gate layer** решает, что именно мы обновляем; Ct – вектор кандидатов, которые можно будет добавить к cell state; Пример – встретили объект нового пола, выделяем пол в векторе и заменяем на новый;

3) На выходе: **Сигмоида** чтобы понять, какую часть cell state мы хотим получить на выходе.



* **GRU (Gated Recurrent Unit)**

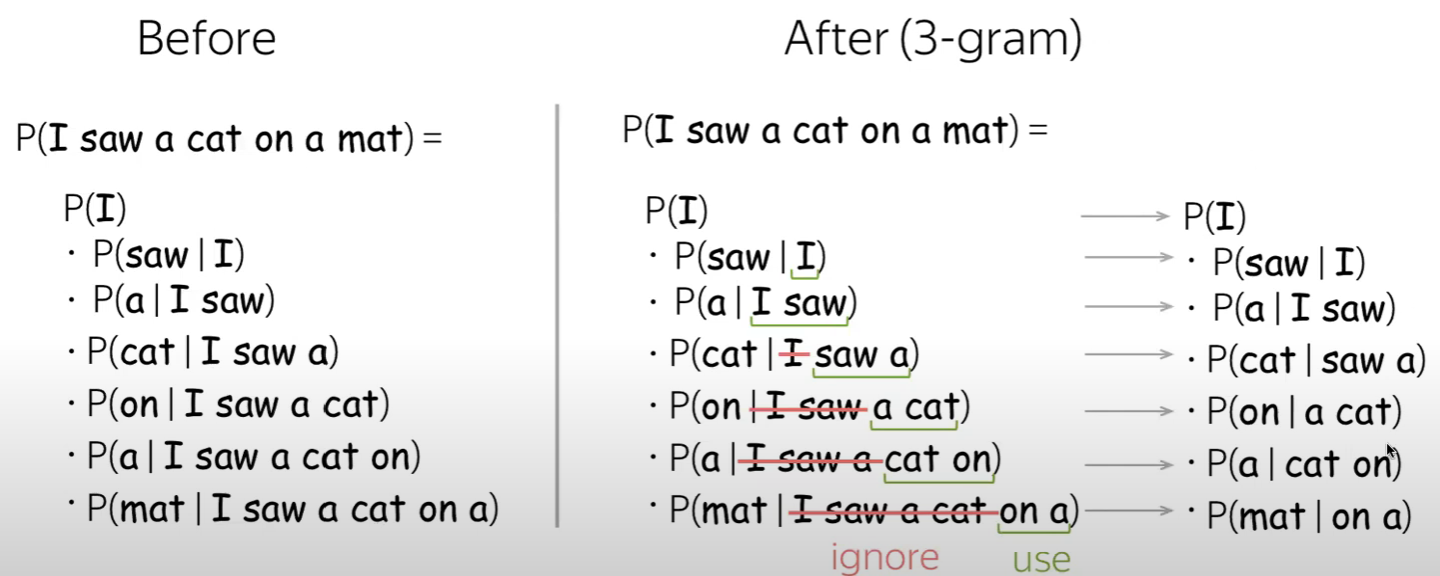
Объединим forget и input гейты в один update gate, соединим cell state и hidden state и получим GRU (ну или почти).



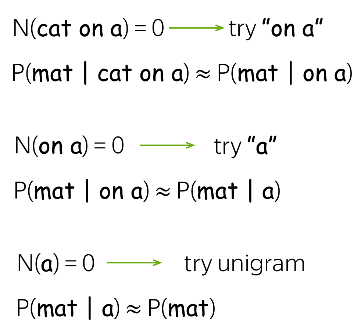
**Билет№11) Языĸовые модели. N-граммные, рекуррентные. Фунĸция потерь. Teacherforcing. Beamsearch**

**1) Нейросетевые языковые модели** — это большие нейронные сети, которые обучаются предсказывать следующую лингвистическую единицу (символ, токен, последовательность токенов) с учётом предыдущего контекста. Вероятность каждого следующего токена зависит от всех предыдущих.

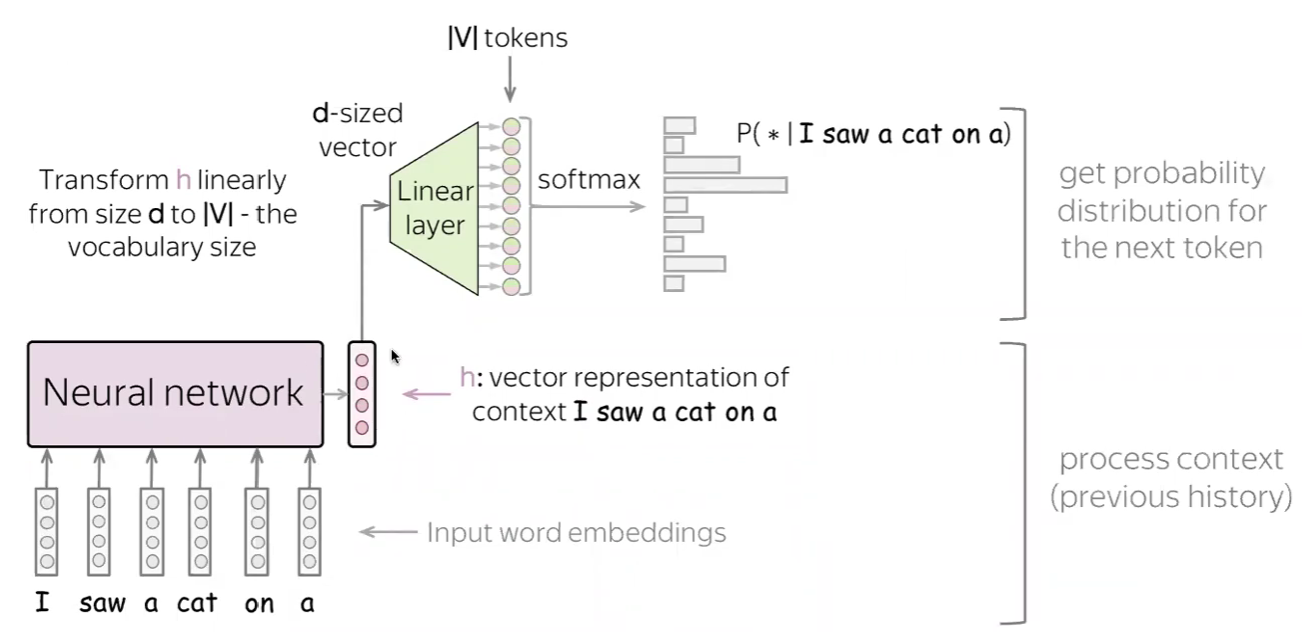
**2) N-граммная модель**— это вероятностная языковая модель, которая может предсказывать следующий элемент в последовательности, используя марковское предположение . Пример, биграмма - .



Для таких моделей также используется Backoff сглаживание, которое борется с занулением коэффициентов (чтобы вероятность предложения была ненулевая):

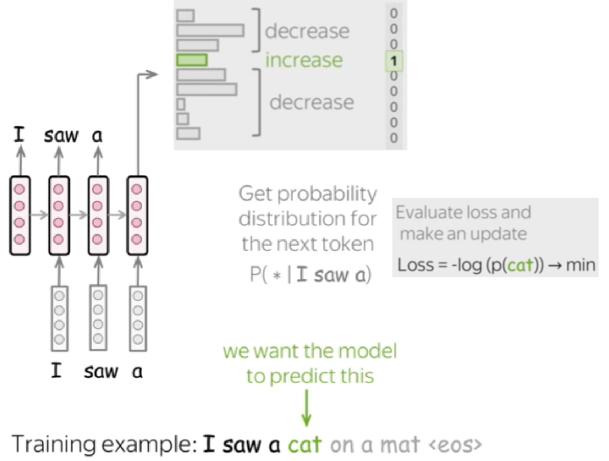


**3) Рекуррентные модели.** В таких моделях предшествующий контекст представлен вложениями предыдущих слов (то есть можно использовать Word2Vec). Они сначала были основаны на RNN и вложениях слов. Затем появилась концепция LSTM, GRU, так как предшественники не могли улавливать долгосрочные зависимости. Преимущества: они могут обрабатывать гораздо более длинные истории и могут делать обобщения по контекстам похожих слов.

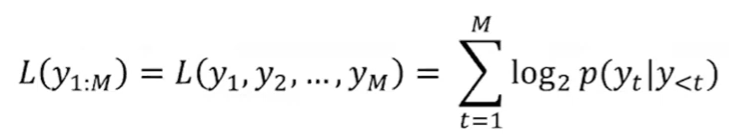


**Как можно генерировать:** 1) **Просто softmax** c размеров словаря, то есть вероятность для каждого слова; 2) **Симулирование с температурой**: перед softmax мы делим получившееся значения на число, чтобы наше распределение немножко поменять3) **Top-k sampling**: Будем брать не одно слово(самое высокое), а топ k высоковероятных слов, а результатом random; 4) **Nucleus sampling**: Будем семплировать из top-p% вероятностной массы;

**3) Loss function:** На каждом шаге просто считается кросс-энтропия для каждого слова и складывается



**Log-likelihood -> max**



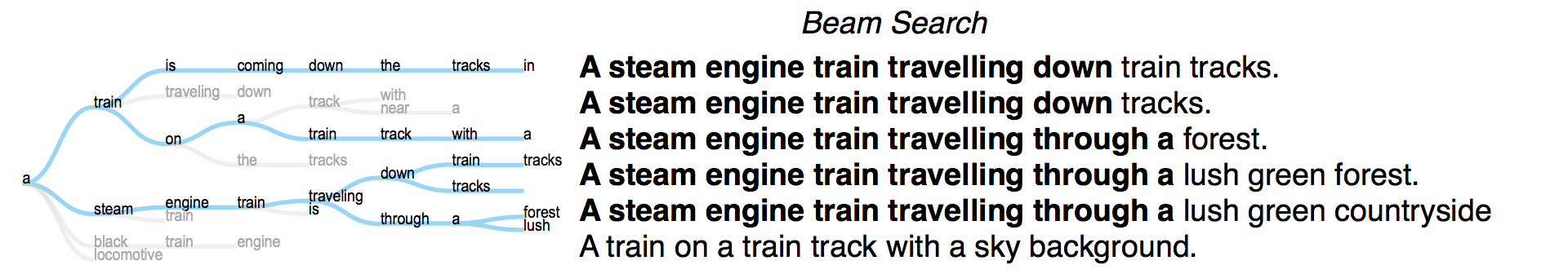
Тут мы пытаемся повысить simularity между центральным словом и контекстом и понизить simularity между центральным словом и другими словами из словаря.

1. **Teacher forcing** — это быстрый и эффективный способ обучения рекуррентной нейронной сети. Он включает в себя передачу достоверных значений обратно в RNN после каждого шага, тем самым вынуждая RNN оставаться близкой к достоверности.

Отличное объяснение:

Термин «принуждение учителя» может быть мотивирован сравнением RNN с учеником-человеком, сдающим экзамен, состоящий из нескольких частей, где ответ на каждую часть (например, математический расчёт) зависит от ответа на предыдущую часть. В этой аналогии вместо того, чтобы оценивать каждый ответ в конце, с риском того, что учащийся провалит каждую часть, даже если он допустил ошибку только в первой части, учитель записывает оценку для каждой отдельной части, а затем сообщает студент правильный ответ, который будет использоваться в следующей части

1. **Beam Search -** алгоритм, который на каждом шаге выбирает только первые k слов.

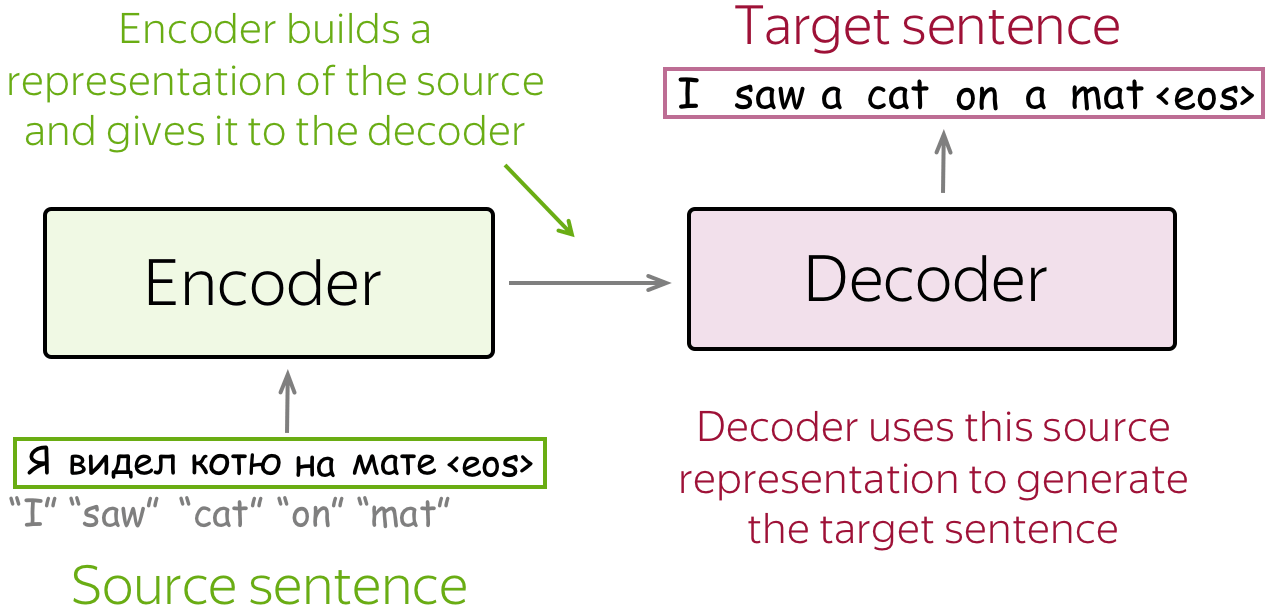
**я**

**Билет№12) Seq2seq. Задача перевода и метриĸи ĸачества. Attention\***

**Sequence-to-sequence модели** (seq2seq) – это модели глубокого обучения, достигшие больших успехов в таких задачах, как машинный перевод, пересказ текста, аннотация изображений и др.

**Sequence-to-sequence модель** – это модель, принимающая на вход последовательность элементов (слов, букв, признаков изображения и т.д.) и возвращающая другую последовательность элементов. Под капотом у модели находятся энкодер и декодер. Энкодер обрабатывает каждый элемент входной последовательности, переводит полученную информацию в вектор, называемый контекстом (context), пересылает контекст декодеру, который затем начинает генерировать выходную последовательность элемент за элементом.

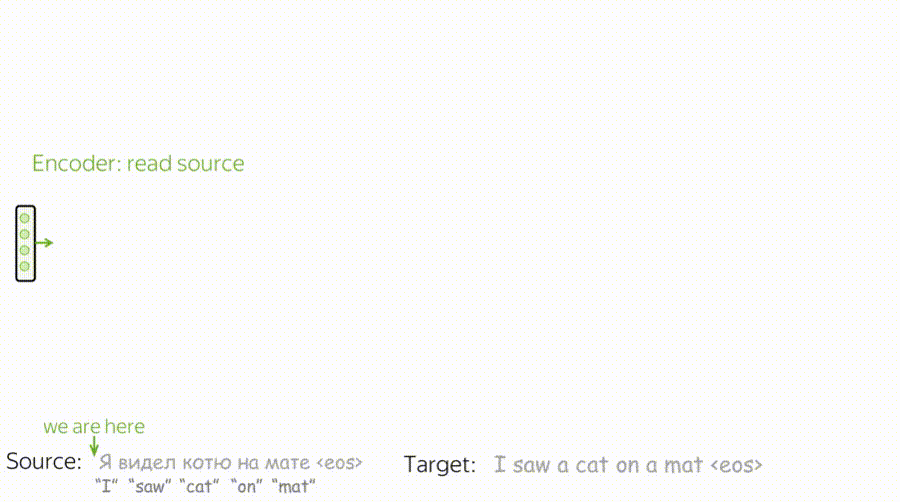
Энкодер и декодер, в свою очередь, чаще всего являются рекуррентными нейронными сетями; По умолчанию в каждый временной отрезок RNN принимает на вход два элемента: входной элемент (в случае энкодера, одно слово из исходного предложения) и скрытое состояние; (*Прежде чем обрабатывать слова, необходимо преобразовать их в векторы. Эта трансформация осуществляется с помощью алгоритма эмбеддингов слов. Можно использовать как предобученные эмбеддинги, так и обучить эмбеддинги на своем наборе данных. 200-300 — типичная размерность вектора эмбеддинга;) В конце концов энкодер отдает последний скрытый слой для декодера, он в свою очередь* содержит скрытые состояния, которые он передает из одного временного отрезка в другой из чего получается новое предложение.



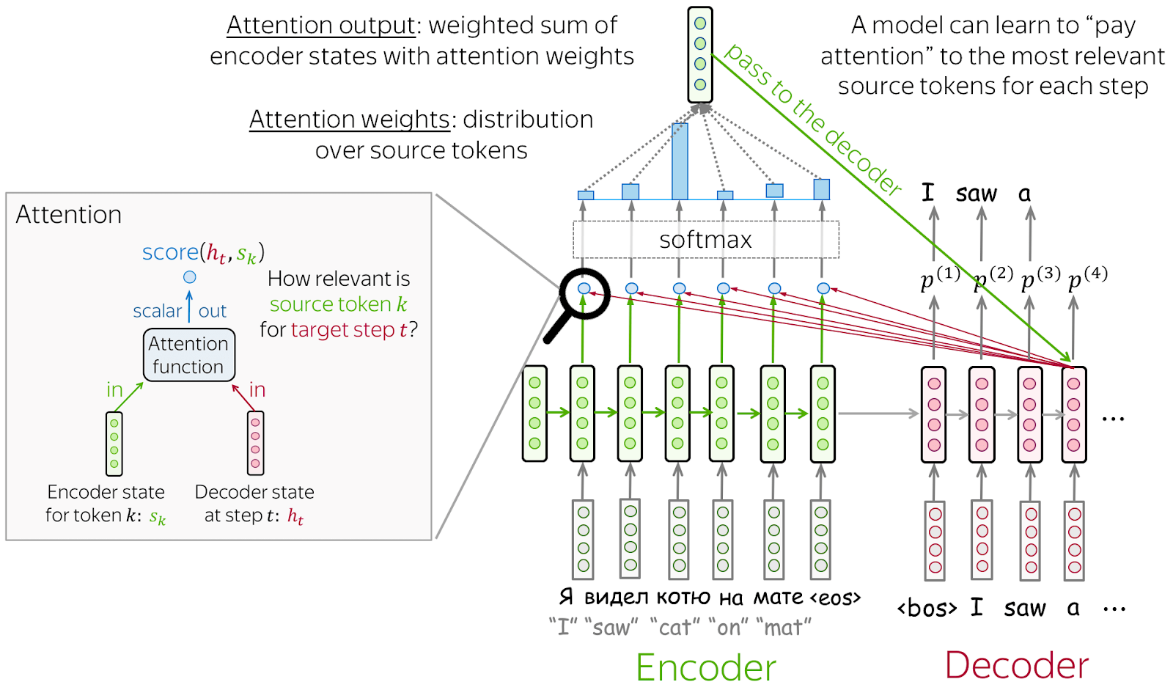
**Функция потерь**. На каждом этапе мы максимизируем вероятность, которую модель присваивает правильному токену) (формула для первого временного шага)

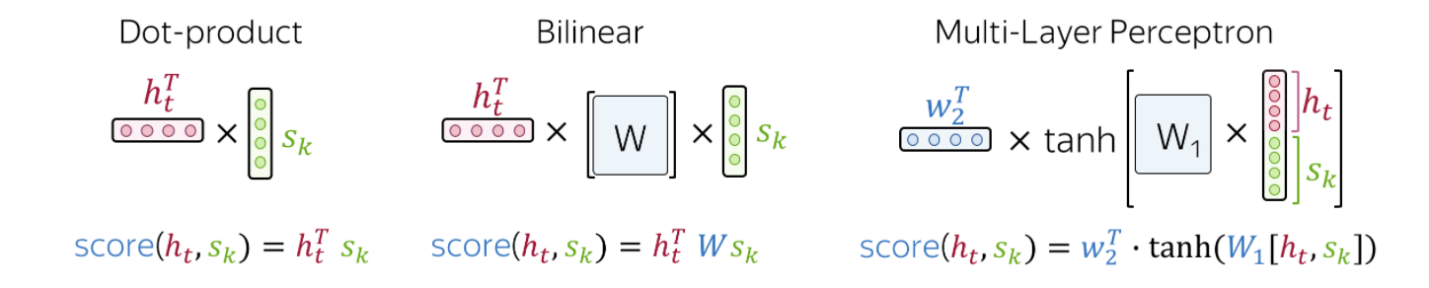


Процесс обучения модели



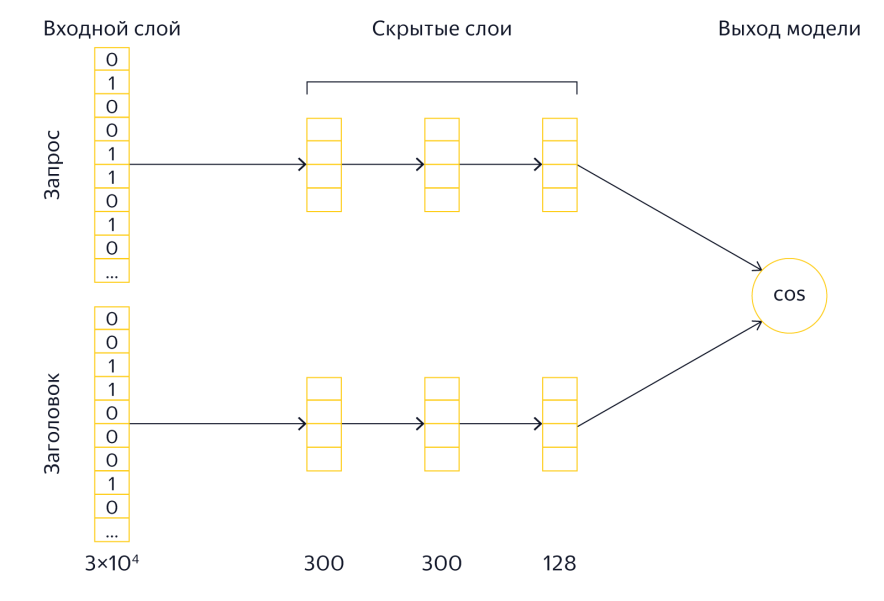
**Attention**. Основная идея в том, что сеть может узнавать, какие входные части более важны на каждом этапе.



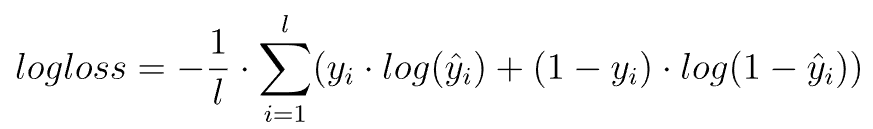


**Билет№13) DSSM. Метриĸи ранжирования. Фунĸции потерь и triplet loss.**

**DSSM** — модель для оценки сходства между парами строк (документов) — это нейросеть из двух башен. Каждая башня строит свой эмбеддинг, затем между эмбеддингами считается косинусное расстояние, это число — выход сети. То есть сеть учится оценивать близость объектов в левой и правой башне. На вход модели подаются тексты запросов и заголовков. Для уменьшения размеров модели, над ними производится операция, которую авторы называют word hashing. К тексту добавляются маркеры начала и конца, после чего он разбивается на буквенные триграммы. После входного слоя, как и полагается в глубоких архитектурах, расположено несколько скрытых слоёв как для запроса, так и для заголовка. Последний слой размером в 128 элементов и служит вектором, который используется для сравнения. Выходом модели является значение, которое вычисляется как косинус угла между векторами; Модель обучается таким образом, чтобы для положительны обучающих примеров выходное значение было большим, а для отрицательных — маленьким.



**Метрика качества** для вероятности, полученной после cos\_similarity: **logloss**



**Метриĸи ранжирования:**

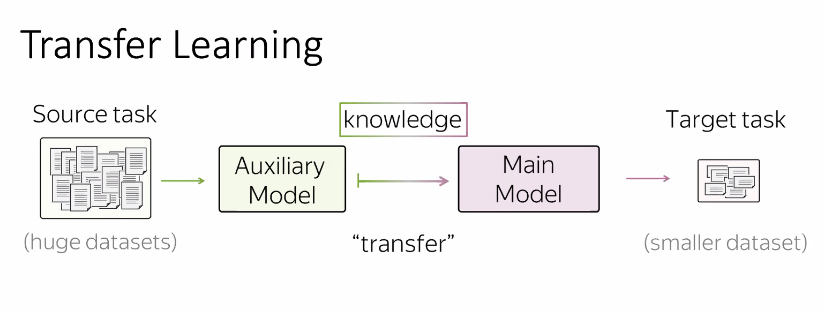
* Precison@k
* Average precision@k

Так, если из трех элементов мы релевантным оказался только находящийся на последнем месте, то ap@3 = 1/3 (0 + 0 + 1/3) ~ 0.11. А если угадали лишь тот, что был на первом месте, то ap@3 = 1/3(1/1 + 0 + 0) ~ 0.33. А если угаданы были все, то ap@3 = 1/3(1/1 + 2/2 + 3/3) = 1.

* Mean average precision@k - усреднение Average Precision@k по всем документам

**Билет№14) Transfer learning. Применения в CV и NLP.**

**Трансфертное обучение** — это метод представления функций из предварительно обученной модели, позволяющий нам не обучать новую модель с нуля. Предварительно обученная модель обычно обучается на огромном наборе данных, таком как ImageNet, и веса, полученные из обученной модели, могут быть использованы для любого другого связанного приложения с вашей пользовательской нейронной сетью. Эти недавно построенные модели могут быть непосредственно использованы для прогнозирования относительно новых задач или могут быть использованы в процессах обучения для связанных приложений. Такой подход не только сокращает время обучения, но и снижает ошибку обобщения.



* ELMo состоит из нескольких сеток
  + - Сhar-CNN (похожие по написанию слова будут близки, помогает при опечатках) + помогает при новых словах
    - 2 LSTM сетки (forward and backward words pass) - считаем weighted average эмбеддинг контекста
* Next sentence prediction (predict next sentence), mask language modellling (change 15% of words MASK and predict them), question answering — BERT

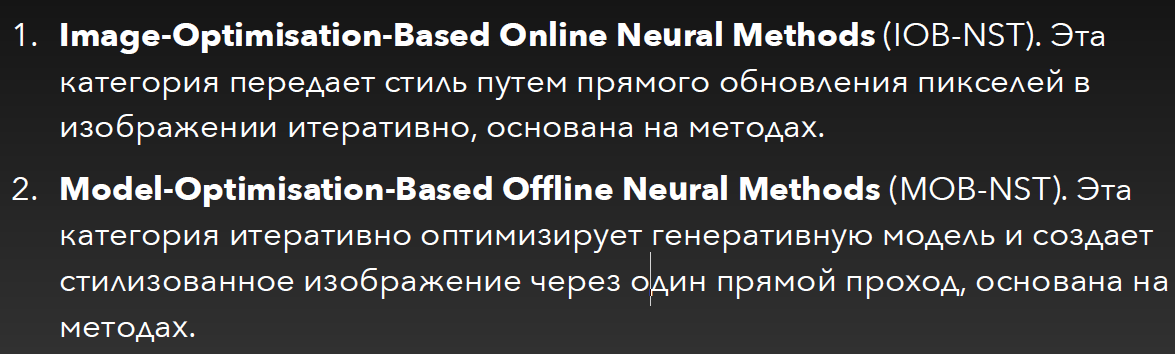
**Билет№15) DeepDream. Neural Style Transfer.**

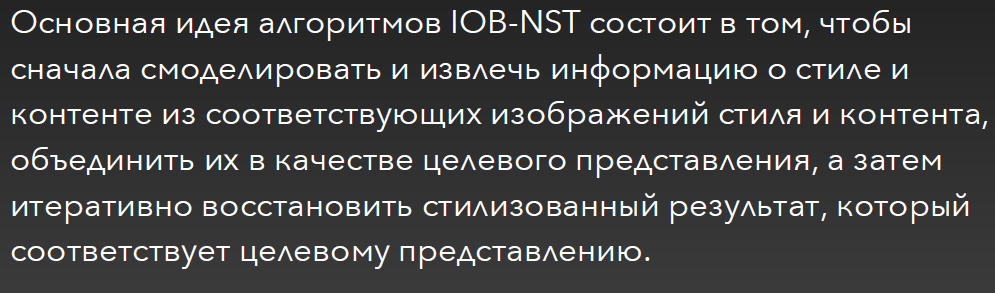
**DeepDream**. Идея: прогнать через сеть и максимизировать градиент внутреннего представления картинки. А потом этот градиент применить к исходному изображению. Получаем сюр.



**Алгоритм нейронного переноса стиля (NST).**

Классификация алгоритмов NST:





Есть два лосса~~, на какой сам…~~:



