Лекция 1: Основы глубинного обучения

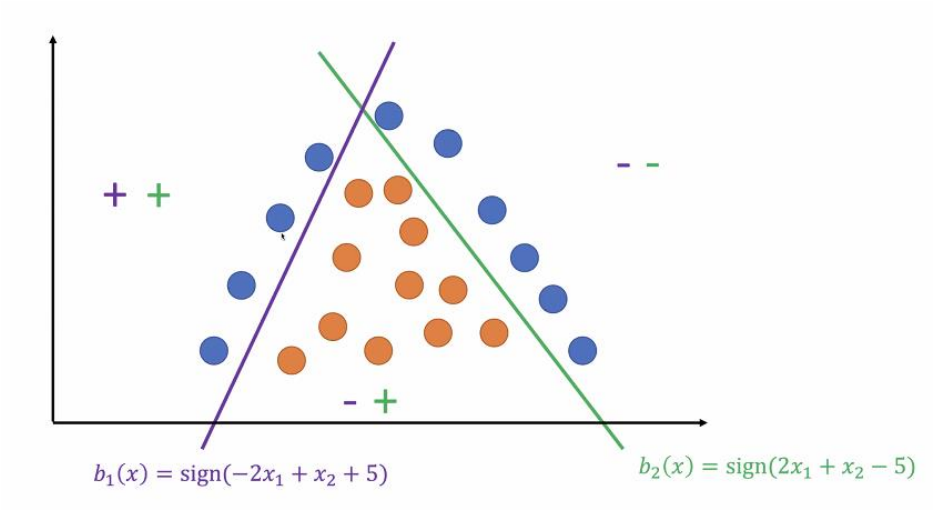
Мотивация

- Анализ изображений
  - Классическое компьютерное зрение
    - Считаем признаки → очень долго
    - Обучаем на них градиентный бустинг
  - Делаем сложную модель
    - Сама преобразует картинку в вектор признаков
    - Выдает ответ
- NLP
  - Классическое NLP
    - Подсчитываем статистику, как часто то или иное слово встречается после данного
    - Генерируем слово из этого распределения
  - GPT-3
    - Огромный корпус текста

Зачем нужны нейронные сети

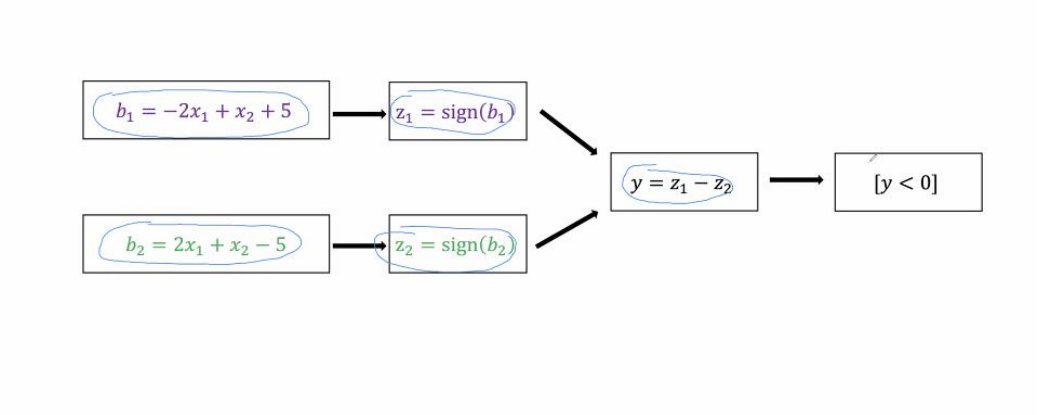
- Линейные модели слишком простые
  - Предполагаем, что **признаки независимы**
  - При введении полиномиальных признаков **не очень интерпретируемы**
- Градиентный бустинг → работает хорошо
  - Тяжело обучать

Нелинейный закономерности



Если знаки двух функций одинаковы → **синий класс**

- Если $b_1 - b_2 < 0$, то это оранжевый класс



Это граф вычислений (computation graph)
Везде линейные модели
Признаки x_1, x_2преобразовали в признаки $\boldsymbol{z_1}, \boldsymbol{z_2}$→ на них обучили модель → пришли к ответу

Нейрон

Приходят сигналы → модифицируются → идут в следующие нейроны
$a(x) = \sum_{j=1}^{d} w_j x_j$
Линейная модель имитирует 1 нейрон → соединяем их друг с другом
$x^{(0)} - \text{признаки}$
$h_1(x) - \text{преобразование (слой)}$
$x^{(1)} - \text{результат}$
$x^{(0)} \to h_1(x^{(0)}) \to x^{(1)} \to h_2(x^{(1)}) \to \ldots \to y$
Графы вычислений могут быть **значительно сложнее**

Полносвязный слой (FC)

На вход подаем $x_1, \ldots, x_n$
На выходе $z_1, \ldots, z_m$

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

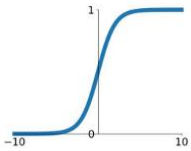
В полносвязном слое nm параметров
Нужно **много данных** для обучения

Нелинейность

Добавляем **ReLU**

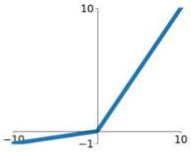
Sigmoid

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



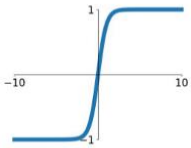
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

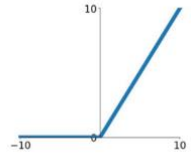


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

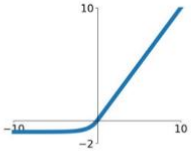
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Без нелинейности у нас будет просто большая **линейная модель**
После каждого слоя нужна **нелинейная функция**

Обучение нейронных сетей

Все слои обычно **дифференцируемы**

$$a(x_i) = FC_2(FC_1(x))$$

Минимизируем функцию ошибки от $(y_i, a(x_i))$

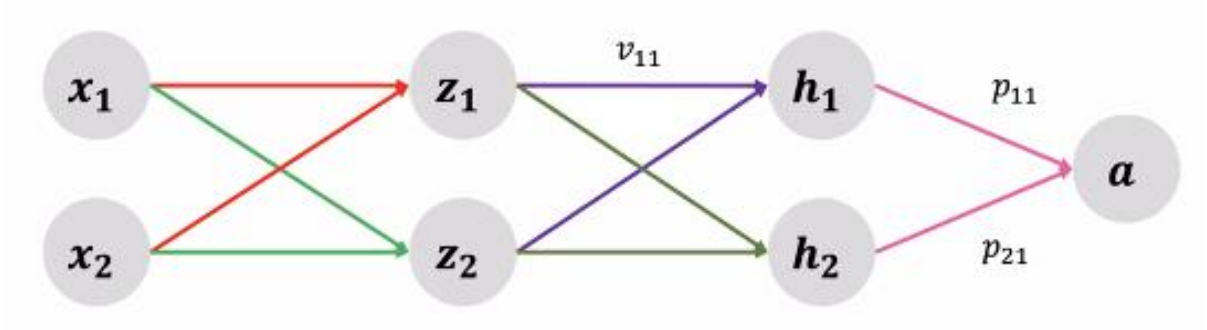
Теорема Цыбенко

Если взять любую непрерывную функцию $g(x)$, можно создать двухслойную нейронную сеть, приближающую $g(x)$ с любой заранее заданной точностью

Лекция 2: Backprop, Сверточные сети

В FC находятся веса → их надо настроить
Обучаем на функцию потерь

Обратное распространение ошибки (backprop)



$$a(x) = p_{11}h_1(x) + p_{21}h_2(x)$$

$\frac{\partial a}{\partial p_{11}} = h_1(x) \rightarrow$ чем больше $h_1(x)$, тем сильнее p_{11}влияет на a

v_{11}влияет на h_1

$$a(x) = p_{11} f(v_{11} z_1(x) + v_{21} z_2(x)) + p_{21} h_2(x)$$

$$\frac{\delta a}{\delta v_{11}} = \frac{\delta a}{\delta h_1} \frac{\delta h_1}{\delta v_{11}}$$

w_{11}влияет на v_{11}

$\frac{\delta a}{\delta w_{11}} = \frac{\delta a}{\delta h_1} \frac{\delta h_1}{\delta z_1} \frac{\delta z_{11}}{\delta w_{11}} + \frac{\delta a}{\delta h_2} \frac{\delta h_2}{\delta z1} \frac{\delta z_1}{\delta w_{11}}$ - перебрали все пути

3: $\frac{\partial p}{\partial h_1}$ $\frac{\partial p}{\partial h_2}$

2: $\frac{\partial p}{\partial z_1} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_1} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_1}$ $\frac{\partial p}{\partial z_2} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_2} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_2}$

1: $\frac{\partial p}{\partial x_1} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_1} \frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_2} \frac{\partial z_2}{\partial x_1} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial x_1}$

$\frac{\partial p}{\partial x_2} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_1} \frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_2} \frac{\partial z_2}{\partial x_2} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial x_2}$

Распознавание изображений

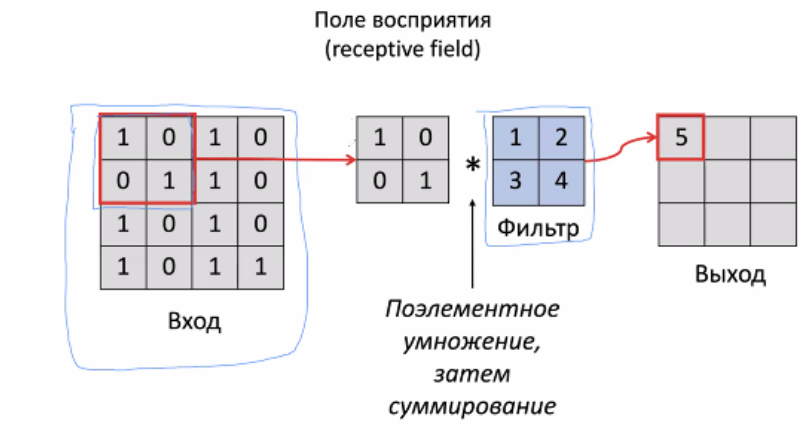
MNIST → распознаем рукописные цифры

Полносвязная сеть

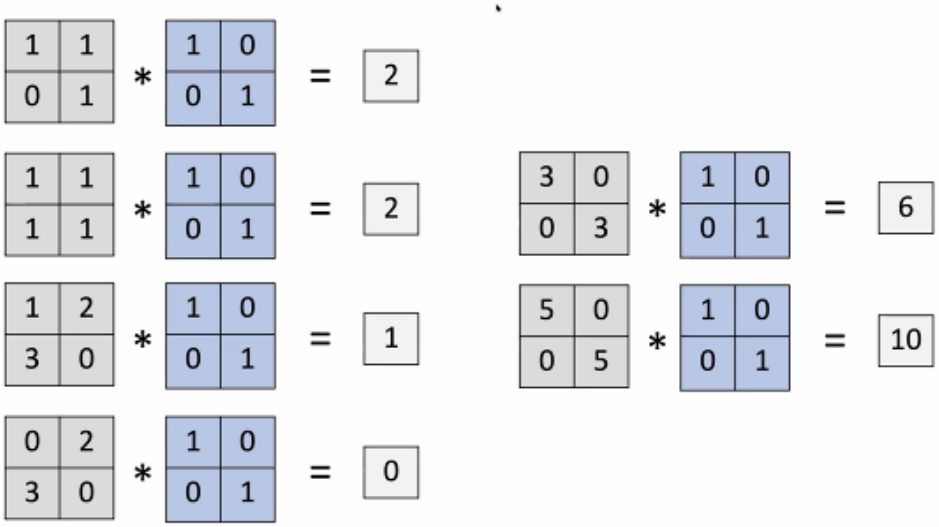
1. Взять пискели как векторы → картинка 28 на 28 содержит 784 входных нейрона
2. Скрытый полносвязный слой - ищет связи в пискелях
    a. Если сдвинуть цифру → **то нейрон не будет на нее реагировать**
    b. Каждый нейрон отвечает за конкретное положение пикселей
    c. Если в полносвязном слое 1000 нейронов, то нужно обучить **785000 параметров**
3. Выходной слой
    a. Плюс еще (1000 + 1) * 10 параметров тут → 10.010
4. **Весов сильно больше, чем входов**
5. **Сильно переобучаются**
6. **Не учитывают специфику изображений**
7. 99.68% качество на 12 млн. Параметров
    a. 114 часов обучения
    b. Применяли аугментацию

Сверточные нейросети

Надо детектить на картинке **конкретный паттерн**



Двигаем красный квадрат по всей матрице
Фильтр дает большие значения, если находит паттерн в куске изображения



Не важно меняется ли положение паттерна
Фильтр Собеля → Детекция краев

$Im^{out}(x,y) = \sum_{i=-d}^{d} \sum_{j=-d}^{d} K(i,j) Im^{in}(x+i, y+j)$ → **Свертка**

$\boldsymbol{K(i,j)}$ — это фильтр + координаты нумеруются от -d до d
Im^{In}- изображение на входе
Чтобы получить (i,j) ячейку вывода - фильтром проходимся по окрестности точки (x,y)
Выполнены два свойства:

1. Пискель в результирующем изображении зависит от <u>небольшого участка исходного изображения</u> **(локальная связность)**
2. Веса одни и те же для всех пикселей результирующего изображения → Shared weights → **очень мало параметров**

Лекция 3: Сверточные нейросети 2

Обычно в изображении несколько каналов из-за цветов → **RGB**

- Добавляется еще одно измерение в матрицу

$$Im^{out}(x, y, t) = \sum_{i=-d}^{d} \sum_{j=-d}^{d} \sum_{c=1}^{C} K_t(i, j, c) Im^{in}(x + i, y + j, c)$$

  - Добавляем сумму по каналу → Фильтр трехмерный
  - Используем несколько фильтров, чтобы выделять одновременно несколько паттернов в изображении
  - Обучаем только фильтры
    - $(2d + 1)^2 \times C \times T$ параметров

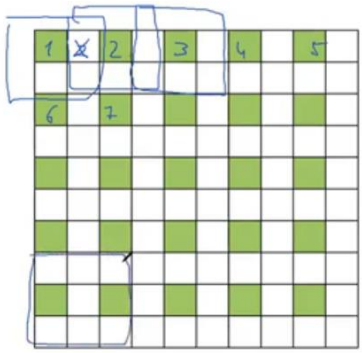
Receptive field

- Пиксель в Im^{Out} видит только размер фильтра
- Можно применить свертку несколько раз → увеличит receptive field
- Наслаивать друг на друга свертки неэффективно на больших изображениях

Три способа

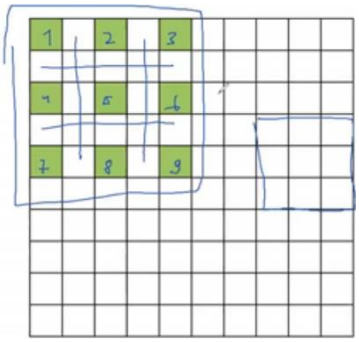
Convolutions with strides

- Пропускаем s клеток между центрами фильтрованных значений
- Размер страйда = **гиперпараметр**
- Увеличивают поле восприятия



Dilated convolutions (Раздутые свертки)

- Фильтр не с непрерывным изображением → пропускаем между пикселями другие пиксели
- Receptive field = (f + l; f + l)



Pooling

- Не имеет параметров
- Гиперпараметр - размер фильтра
    - Считаем внутри max **= max-pooling**
- Уменьшает картинку

Важно следить за тем, чтобы последние слои имели receptive field как вся картинка

Padding

- **Valid mode** - Если считаем свертку с фильтром без паддинга - теряем края изображения, так как там нельзя поставить центр фильтра
- Уменьшается размер изображения через фильтр → не всегда хорошо для вычислений

Zero-padding

- Покрываем края изображения нулями
- Нейросеть может переобучиться на края

Reflection padding

- С краев ставим отражения
- Тут получаются симметрии → нейросеть может это выучить

Replication padding

- ...

Резюме

- Padding может помочь контролировать размер изображений
- Учитываем объекты на краях
- Разные типы паддингов допускают переобучение на края

Архитектура нейронных сетей

1. Conv

2. Pooling
3. Conv
4. Pooling
5. …
6. Flattening
7. FC - Если взять это как признаки, то это будут хорошие признаки картинки
    a. Неинтерпретирумы
8. FC
9. Out

Лекция 4

Оптимизация

Градиентный спуск

- Очень много считать, так как в нейросетях слишком много весов
    - Для линейной регрессии идет суммирование всех элементов выборки
    - $Q(w) = \frac{1}{l}\sum_{i=1}^{l} L(y_i, a(x_i))$

Стохастический градиентный спуск

- Оцениваем градиент на одном случайном объекте
    - Оценка несмещенная
- Под конец начинаются сильные осцилляции
    - Как фиксить?
        - Длина шага зависит от итерации
            - $w_{t+1} = w_t - \eta_t \nabla L(y_{i_t}, a(x_{i_t}))$
            - η_t убывает
                - Должна стремиться к 0
- Гарантия сходимости есть только для выпуклых функций - у нейросетей не выпуклые
    - Есть локальные минимумы

Mini-batch SGD

- Стоит брать степень 2
    - Так эффективнее для памяти
- Может сделать оценку градиента стабильной
    - Нет гарантий
- Обычно также эффективно, как и для 1 объекта
- Лучше всего брать от 2 до 32
    - Маленькие шаги

Проблемы

- В случае вытянутых линий уровня - градиентный спуск требует аккуратного подбора длины шага и будет долго сходится
    - Используем momentum
- Разреженные данные → редко оптимизируем по редкой категории + длина шага убывает, независимо от того какой раз обновляем параметр
- Признаки с разным масштабом - нужно учесть в скорости сходимости
    - Adagrad

Momentum

- Инициализируем $h_0 = 0$
- $h_t = \alpha h_{t-1} + \eta \nabla Q(w^{t-1})$
    - Усредненное направление движения
    - α- экспоненциальное затухание, обычно берут побольше
- $w_t = w_{t-1} - h_t$

Nesterov Momentum

- $h_t = \alpha h_{t-1} + \eta \nabla Q(w^{t-1} - \alpha h_{t-1})$
    - Сначала делаем шаг по моменту, потом оптимизируем
    - Ускоряет сходимость

Adagrad

- Инициализируем ${G_j}^t = 0$

$$ {G_j}^t = {G_j}^{t-1} + (\nabla Q(w^{t-1}))^2 $$

$$ {w_j}^t = {w_j}^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}} g_{tj} $$

$$ g_{tj} = \nabla Q(w^{t-1}) $$

RMSProp

$$ G_j^t = \alpha G_j^{t-1} + (1 - \alpha)\left(\nabla Q(w^{t-1})\right)_j^2 $$

$$ {w_j}^t = {w_j}^{t-1} - \frac{\eta_t}{\sqrt{G_j^t + \epsilon}} g_{tj} $$

- Знаменатель слишком быстро убывает в AdaGrad → берем альфу, так что скорость зависит только от недавних шагов

Adam

- …
- И momentum и RMSProp

Регуляризация

- Можно снизить число параметров
  - Сделать PCA
  - Сверточные нейросети
- Добавление штрафа к ошибке
- Dropout

Dropout

- Выкидываем некоторые нейроны и оптимизируем как будто случайных нейронов нет
- Отдельный слой
- *Inverted dropout*

$$d(x) = \frac{1}{p} m \times x$$

- m – вектор размера x из распределения Бернулли
- Делим на p, чтобы сохранить масштаб слоя после выкидывания
- *Dropout*

$$d(x) = m \times x \ - \text{ на этапе обучения}$$
$$d(x) = xp \ - \text{ на этапе применения}$$

- *Интерпретация*
  - Используем множество нейросетей с выкинутыми нейронами → как будто используем композицию

Batch Norm

- *Covariate shift* – классы в тестовой выборке смещены по отношению к обучающей выборке

Domain adaptation

$$\sum_{i=1}^{l} s_i(a(x_i) - y_i) \ \rightarrow min$$

  - Большие веса для объектов, которые похожи на объекты в тестовой выборке
- *Internal covariate shift*
  - Изменение первого слоя → Следующие слои работают хуже
  - Оцениваем среднее и дисперсию каждой компоненты входного вектора

$$\mu_B = \frac{1}{n}\sum_{j=1}^{n} x_{B,j}$$

$$\sigma_B^2 = \frac{1}{n}\sum_{j=1}^{n}\left(x_{B,j} - \mu_B\right)^2$$

$$\widetilde{x_{B,J}} = \frac{x_{B,j} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$z_{B,j} = \gamma\widetilde{x_{B,J}} + \beta,$$

$$\gamma \text{ и } \beta - \text{ обучаемые параметры}$$

- Обычно вставляется между полносвязным слоем и нелинейностью
- Позволяет увеличить длину шага в градиентном спуске
- Не факт, что действительно устраняет covariance shift

Лекция 5

Инициализация весов

- Можно инициализировать нулями – очень плохо
  - Все нейроны одинаковые → Все обучаться будут одинаково
- Пытаемся масштабировать веса
  - Иначе будут взрывы градиентов или затухание
- *He initialization*

$$w_j \sim \frac{2}{\sqrt{n}} N(0,1)$$

- Xavier initialization
  - …

Аугментация

- Создаем новые изображения
  - Блюрим
  - Растягиваем
  - Поворачиваем
  - Двигаем
  - Блики
  - …
  - Библиотека albumentations
- Бесплатное расширение обучающей выборки
- Регуляризация модели
- На этапе применения модели можно применить аугментацию – усреднить ответы модели по аугментированным картинкам

Архитектуры сверточных сетей

Le Net (1998)

    - Вход 32x32
    - Свертка с 6 каналами
    - Max Pooling
    - Еще свертки
    - Max Pooling
    - 16 фильтров размера 5 x 5
    - Вытягиваем в полносвязный слой
    - 3 FC слоя
- End-to-End обучение – сама решает задачу от начала до конца
- Около 60к параметров

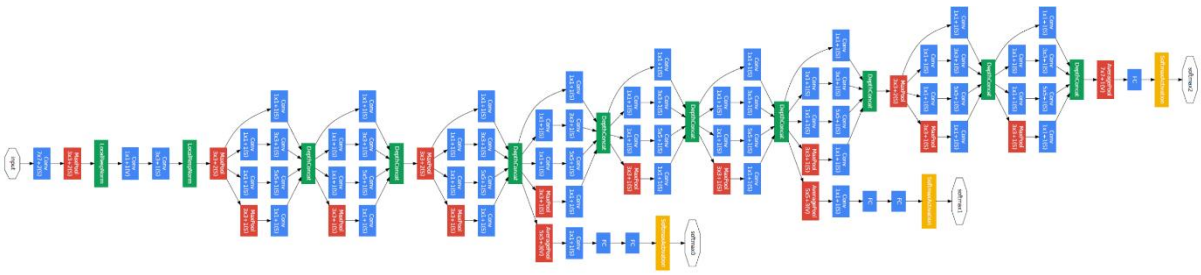
AlexNet (2012)

- Модель разбили на 2 одинаковые части – из-за того, что не влезала на 1 видеокарту
- Архитектура
    - Свертка 5 x 5 - 48 каналов
    - Max Pooling
    - Свертка 3 x 3 – 128 каналов
    - Max Pooling
    - Свертка 3 x 3 – 192 канала
    - Свертка 3 x 3 – 192 канала
    - Свертка 3 x 3 – 128 канала
    - Max Pooling
    - Dense 2048
    - Dense 2048
- Тут используется ReLU, Dropout, аугментация, momentum
- 60 млн. параметров

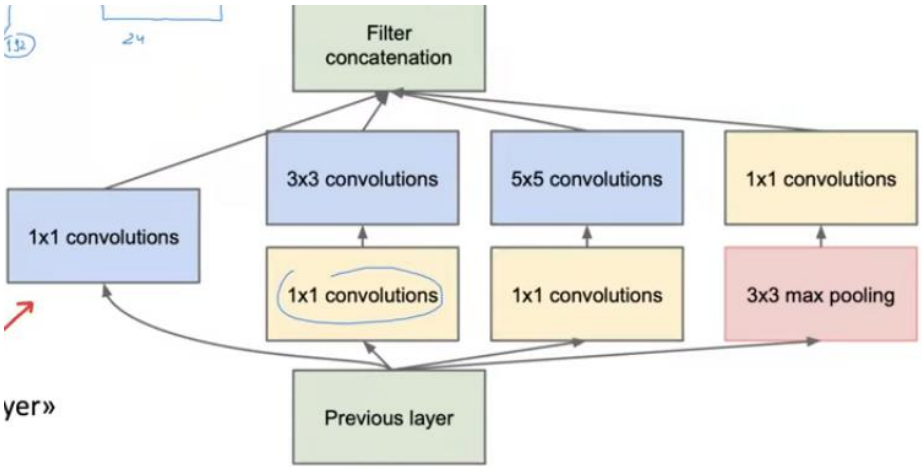
VGG (2014)

- 5 вариантов архитектур
- Маленькие свертки (везде 3x3), но больше каналов и более глубокая сеть
- Между Max Pooling много слоев
- Каналов на каждой итерации в 2 раза больше
- Число параметров
- Dropout
- Хитрая инициализация

GoogLeNet



Inception module



- Берем признаки и сверткой 1x1 считаем их линейные комбинации → Схлопываем каналы
    - Projection layer
- Делаем конкатенацию с картами одного размера с разными размерами фильтров

- Дополнительные слои на выходе, чтобы градиенты не затухали

ResNet (2015)

- Количество слоев дает ошибку выше и на тесте, и на обучении
    - Теряются градиенты – К началу они маленькие → параметры не обучаются
- **Skip connections**
    - Выход слоя = преобразования + выход предыдущего слоя
    - Позволяют обучать глубокие нейросети
- Дает низкую ошибку даже с 1000 слоями

Transfer learning

- Если данных мало
- Берем модель с другой задачи
- Выкидываем полносвязный слой
- Вставляем свой и обучаем его

- По сути, обучаем линейную модель
- Вся остальная сеть – Feature Extractor
- Чем больше отличается задача, тем больше слоев переучивать

Лекция 6

Интерпретация нейросетей

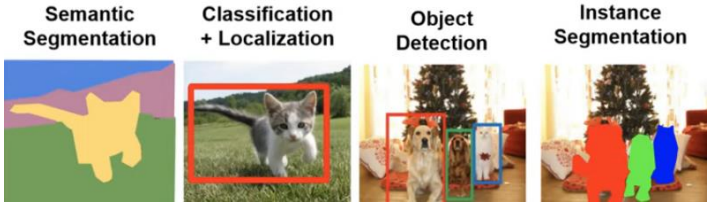
- Очень много параметров → непонятный вклад
- Как интерпретировать?
  - Ищем элемент, на котором фильтр имеет максимальное значение

Максимизация вероятности класса

- $\alpha_y(x) - \lambda\left|\left|x\right|\right|^2 \rightarrow max_x$
  - Инициализируем картинку случайным шумом
  - Ищем оптимальную картинку для данного класса градиентным спуском

Задачи компьютерного зрения

- Semantic Segmentation
- Classification + Localization
- Object detection
- Instance segmentation



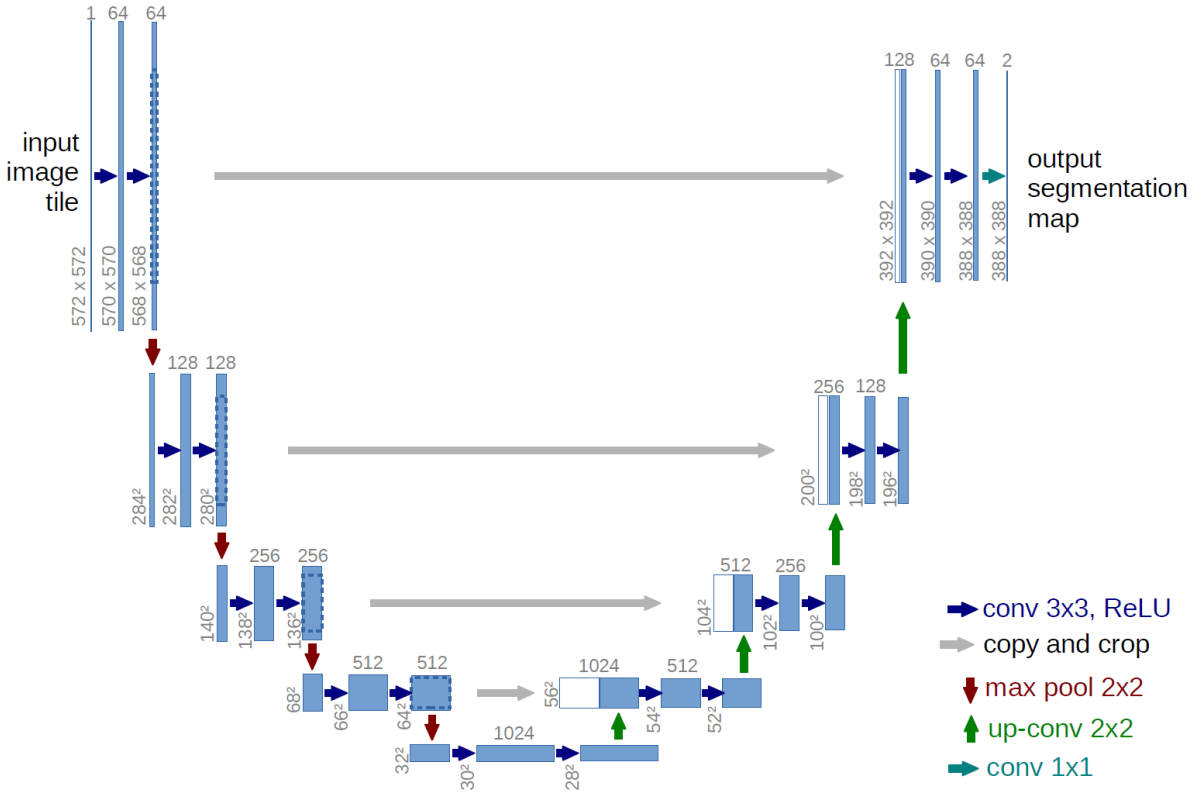
Все эти задачи сводятся к классификации

Семантическая сегментация

- Не разделяем на отдельные объекты, выделяем просто к какому классу относится область изображения
- Задача классификации на отдельных пикселях
  - Для каждого пикселя есть ответ
- Хорошо работает на маленьких выборках, так как обучаемся на пиксели
- **Метрики**
  - Попиксельная accuracy
    - $\boldsymbol{L(y,a) = \frac{1}{n}\sum_{i=1}^{n}[y_i = a_i]}$
  - Мера Жаккара (IOU)
    - $\boldsymbol{Jk(y,a) = \frac{\sum_{i=1}^{n}[y_i=k][a_i=k]}{\sum_{i=1}^{n} max([y_i=k][a_i=k])}}$

- Считается для каждого класса отдельно
- Площадь пересечения положения класса и оценки на площадь объединения класса и оценки

- **Loss**
  - $L(y, a) = \sum_{i=1}^{n} \sum_{k=1}^{K} [y_i = k] log(a_{ik})$
    - Превращаем числа в вероятности через softmax
      - $a_{ik} = \frac{exp(b_{ik})}{\sum_{m=1}^{K} exp(b_{im})}$
- **Как делать предсказания**
  - **Fully Convolutional Network**
    - Убираем полносвязные слои
    - В конце свертками 1x1 делаем предсказания по каждому пикселю
    - Тензор на последнем слое маленький
    - У объектов нечеткие границы
  - **U-Net**
    - Свертки
    - Max-pooling
    - Расширяем картинку с помощью up-convolutions
    - Copy and crop – добавляем к картинке на этапе разворачивания, ее же на том же по порядку этапе сворачивания



Up-convolution

- Upsampling – вставляем в новые пиксели 0

Dilated convolutions

- Увеличивается receptive field + с правильным паддингом может стать по размеру как исходная картинка

Задача детекции

- Не можем просто перебрать все прямоугольники
- Не сможем стоящие рядом объекты детектить

Non-maximum suppression

- Модель выдает много прямоугольников с некоторой уверенностью
    - Идем по прямоугольникам с высокой уверенностью
    - Если другие пересекаются сильно (IOU > 0.5) → Удаляем

Метрики качества

- Модель выдает для класса k список прямоугольников с уверенностями
- Считаем прямоугольник корректным:
    - IoU(y, z) = Jaccard(y, z) > t
        - Большое пересечение с правильным прямоугольников
- Строим PR-кривую для всех порогов → Считаем под ней площадь → Average precision
- Усредняем по всем объектам → Mean average precision (mAP)

Two-shot detection

- Медленнее, но точнее
- Решаем задачу в два этапа
    - Находим прямоугольники где что-то есть
    - Классифицируем эти прямоугольники
- **R-CNN**
    - Выделяем 2к прямоугольников классическим компьютерным зрением → Region proposals
        - Концентрируются там, где перепады цветов
    - Закидываем их в AlexNet
    - Минусы этой схемы
        - Блоки учатся независимо друг от друга → Долго
- **Fast R-CNN**
    - Кандидаты генерируются отдельным слоем
    - В последнем слое вырезаем кандидатов из тензора
    - Парой полносвязных слоев предсказываем классы и насколько нужно сдвинуть прямоугольник чтобы было лучше
- **Faster R-CNN**
    - Одна из веток сверточной сети генерирует кандидатов
    - И передает их дальше
    - Region Proposal Network
        - Берем последний сверточный слой сети
            - 256 каналов
        - В каждой точке тензора ставим 9 прямоугольников разного размера, с разным соотношением сторон
        - Для каждого такого прямоугольника предсказываем есть ли объект и корректировки

- Выбираем топ прямоугольников по вероятности, корректируем, потом уже классифицируем

YOLO (You only look once)

- Разбиваем картинку на S x S прямоугольников
- Прогоняем через сверточную архитектуру
- В каждом прямоугольнике может быть не более b объектов
- Предсказываем
  - Координаты
  - Ширина, высота anchor box
  - Есть ли объект
  - Какой объект

Лекция 8: Идентификация, обучение без учителя

Deep Face

- Обучаем некоторую архитектуру
- Делаем классификацию с большим количеством классом для предсказания разных людей
- Используем предпоследний слой как признаки, так как он хорошо описывает лица
- Нормируем признаки
- Считаем близость вектора фотографии, которая поступает и других векторов по какой-то метрике
- Можно сравнивать разницу в векторах с порогом
- Двухэтапная задача

Face Net

- Сразу обучаем нейросеть помещать лица так, чтобы лица одного человека были близко в пространстве
- Берем батч
- Прогоняем через нейросеть
- Нормируем вектор
- Обучаем на **триплетный лосс**
  - Считается для трех изображений
  - Anchor – исходная
  - Positive – Тот же человек
  - Negative – Другой человек
  - $\sum_{i=0}^{N}[\left|\left|f(x_i^a) - f(x_i^p)\right|\right|_2^2 - \left|\left|f(x_i^a) - f(x_i^n)\right|\right|_2^2 + \alpha]$

- Выбираем positive и ищем semi-hard negatives (на близком расстоянии, $> \alpha$)
    - Тогда лучше учиться

Построение представлений изображений

- В сверточных архитектурах вывод предпоследнего слоя = хороший набор признаков (представление изображения)
- Но для её обучения нужны изображения с разметкой
- Надо научиться строить векторы без разметки!

Автокодировщики

- Надо обучить такой эмбеддинг, чтобы up-convolutions были попиксельно максимально близко к исходной картинке
- $\frac{1}{l}\sum_{i=1}^{l} L(x_i, g(x_i)) \to min$, где L – расстояние между картинками
- Недостатки
    - Восстанавливают изображения с потерями
    - Переобучаются
- Решение
    - Нам нужно восстанавливать не картинку, а ее смысл
    - **Perceptual loss**
        - Ранние слои воспринимают стиль картинки, поздние ее смысл
        - Измеряем расстояния разных слоев
            - Если интересуют только объекты, используем последние слои
            - Берем какую-то предобученную сеть для получения результатов слоев

Denoising autoencoder

- Добавляем в картинку шум, требуем чтобы выход был без шума
- Берем исходную картинку + Добавляем шум
- Кодируем
- Получаем эмбеддинг
- Декодируем
- Требуем, чтобы выход был близок к исходному изображению

Зачем это все?

- Сжатие данных – аналог PCA
- Поиск похожих картинок
- Трансформация, генерация изображений
    - Морфинг изображений
        - Создаем два вектора для картинок
        - Усредняем их
        - Подаем вектор в декодировщик
        - Получаем

Лекция 9: Работа с последовательностями

Bag of words

- Заводим словарь, состоящий из всех слов в выборке
- Делаем признак-индикатор для каждого слова из словаря
- Можно добавлять n-граммы
- Классический подход
- **Проблемы подхода**
  - Много признаков
  - Не учитывает смысл слов
  - Семантически похожие тексты могут иметь разные представления
    - Не учитываются синонимы

Word2vec

- Требуем:
  - Если выкинуть слово, то оно должно хорошо восстанавливаться по представлениям соседних слов – Self-Supervised Learning
  - **CBOW**
    - Последовательность из 5 слов
    - Выкидываем среднее слово
    - Оно должно хорошо предсказываться по соседним словам
  - **Skip-Gram**
    - По вектору центрального слова предсказываем векторы соседних слов
    - Обучаем два вектора для каждого слова - v_w, v'_w
      - На случай если слово центральное (центральное представление слова), и на случай, если слово – сосед (контекстное представление слова)
    - Центральное слово w_I
    - Соседнее слово w_O

$$p(w_O|w_I) = \frac{exp(< v'_{w_O}, v_{w_I} >)}{\sum_{w \in W} exp(< v'_w, v_{w_I} >)}$$

$$\sum_{T} \sum_{i=1}^{n} \sum_{-c \le j \le c} log\left(p(w_{i+j}|w_i)\right) \to max$$

    - Считать знаменатель очень затратно → Производные тоже затратно считать

Negative sampling

$$log\sigma(< v'_{w_O}, v_{w_I} >) + \sum_{i=1}^{k} log\,\sigma(-< v'_{w_i}, v_{w_I} >) \to max$$

- Второе слагаемое позволяет сохранить эффект минимизации знаменателя

- w_i – случайное слово
- Слово генерируется с вероятность P(w) – шумовое распределение

$$P(w) = \frac{U(w)^{3/4}}{\sum_{v \in V} U(v)^{3/4}}$$

  - U – частота слова

Word2vec: Особенности обучения

- Важно сэмплировать в SGD слова с отрицательным учетом их популярности – иначе не будем обращать внимание на редкие слова

Как это использовать

- Можно искать похожие слова
- Можно менять формы слов
  - Обычно отличаются на одно направление

Проблемы Word2vec

- Считаем каждое слово независимым - не учитываем структуру слов
- Не учитываем опечатки

FastText

- Учим представления не для отдельных слов, а для токенов
- Заменяем каждое слово на мешок
- Заменяем слово на набор токенов из мешка
- Вектор слова – сумма векторов всех токенов
- В остальном как word2vec

Работа с текстом

- Усреднить предложения по словам?
- Усреднить с весами?
- Есть ли возможность поумнее
- **Сверточные сети**
  - Сворачиваем последовательности с одномерными фильтрами
    - Берем свертки с разной шириной
  - Max pooling
  - Полносвязный слой
- CNN-rand – обучаем векторы для слов, инициализируем случайно
- CNN-static – Берем word2vec как векторы
- CNN-non-static – Берем word2vec как векторы и потом дообучаем
- **Минусы**
  - Нет определения глобального смысла текста, не анализируем весь текст

Рекуррентные нейронные сети

- Последовательность - $x_1, x_2, \dots, x_n$
- Читаем слева направо
- $\mathrm{h}_t = f(W_{xh}x_t + W_{hh}h_{t-1})$ – накапливаем информацию в векторе h
- Если хотим выдавать что-то на каждом шаге
  - $\mathrm{o}_t = f_o(W_{ho}h_t)$
- Можно стэкать рекуррентные слои
- **Можно обучить по t символам предсказывать t+1 слово**
  - Выбираем слово с максимальной вероятностью
  - Генерируем слово из распределения
- К последнему токену прикручиваем полносвязные слои и выдаем предсказания

Backpropagation through time (BPTT)

- На каждом этапе предсказываем элемент последовательности
- Измеряем ошибку
- Бэкпроп через всю модель
- Происходит затухание градиентов
  - Ошибки в начале очень дорогие для остальной модели

LSTM

- Прокидываем градиенты

$$\mathrm{h_t} = (1 - \mathrm{z_t}) \times \mathrm{h_{t-1}} + \mathrm{z_t} \times \widetilde{\mathrm{h_t}}$$
$$\widetilde{h_t} = tanh(W \times [r_{t-1} \times h_{t-1}, x_t])$$
$$r_t = \sigma(W_r \times [h_{t-1}, x_t])$$
$$z_t = \sigma(W_z \times [h_{t-1}, x_t])$$

- Residual connection

Bidirectional LSTM

- Две сети – одна читает справа налево, другая слева направо
- Берем скрытые состояния, конкатенируем
- Предсказываем часть речь по сконкатенированному вектору
- Теперь сеть знает весь контекст

Seq2Seq

- Выдаем последовательность, которая не равна по длине входной последовательности
- Прогоняем последовательность через Encoder – получаем вектор, который описывает всю последовательность
  - RNN – получаем h_n
- Дальше декодируем через Decoder
  - Другая RNN
- Ставим в конце <EOS>, чтобы сеть знала что текст закончился

- **Проблема**
  - Последние слова более значимы
  - В декодере то же самое
  - BiLSTM не спасает ситуацию – не учитывает центр

Beam Search

- Выбираем B наиболее вероятных вариантов для первого слова, запоминаем
- Пробуем продолжить первое слово для всех B вариантов – опять выбираем B вариантов
- …
- Выбираем тот который на последнем шаге получил наиболее высокую вероятность

Механизм внимания

- Во время генерации каждого слова смотрим на всю входную последовательность
- Усреднить скрытые состояния?
  - Нужен не весь текст, а только некоторая часть
- Усреднять со специфичными весами значительно лучше!
  - Скрытое состояние декодировщика $h_t^d = g\left(\widehat{y_{t-1}}, h_{t-1}^d, c_t\right)$
    - c_t – вектор контекста из входной последовательности
  - Релевантность скрытого j-го входного слоя t-тому выходному слою
    - Полносвязная нейросеть

$$s\left(h_j^e, h_{t-1}^d\right)$$

  - Нормируем s

$$\alpha_{jt} = softmax\left(s\left(h_j^e, h_{t-1}^d\right)\right)$$

  - Усредняем входные векторы

$$c_t = \sum_j \alpha_{jt} h_j^e$$