



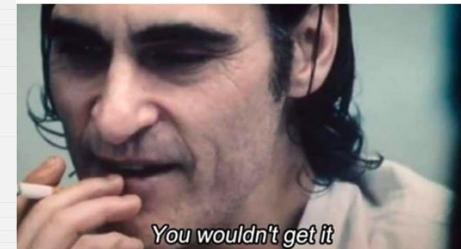
Деплоймент ML моделей

Intro

Обо мне

- 24 года
- 5 лет опыта промышленной разработки
- Начинал с бекенда
- Пишу на Scala/Go/Python
- Люблю C++
- Вместе с небольшой командой ML-Инженеров пилим ML-модели для бизнеса (в основном AdTech)
- Этой же командой разрабатываем и поддерживаем production-сервисы, внутри которых эти модели живут
- Сервисы в основном привязаны к Hadoop-стеку (Flink, Spark), но им не ограничиваются
- Примеры задач:
 - предсказание соц. дем. атрибутов юзеров(скиньте 4 лабу, у кого скор хороший?) (Spark, Xgboost, Catboost)
 - построение “интересов” конкретного пользователя (Flink, Tf-Idf, экспериментируем с BERT)
 - предсказание вероятности клика внутри AdServer'a с жесткими ограничениями по времени (Golang, tensorflow, factorization machines)
 - модели классификации данных с Wi-Fi-сенсоров (Flink, DI4j, LSTM)

When a noob asks me what I do as an ML Engineer



You wouldn't get it

When an expert asks me what I do as an ML Engineer



@debo

Обо мне

Очень не люблю слайды и очень люблю писать
код

Основной тезис

Не все ML-фреймворки подходят работы на вашей
продакшн-нагрузке/объемах данных/совместимы с
вашим стеком технологий

План

1. Intro
2. Recap & Deployment:
 - Линейные модели
 - Бустинги над деревьями
 - Deep learning
3. Практика

План

Что повторим/узнаем:

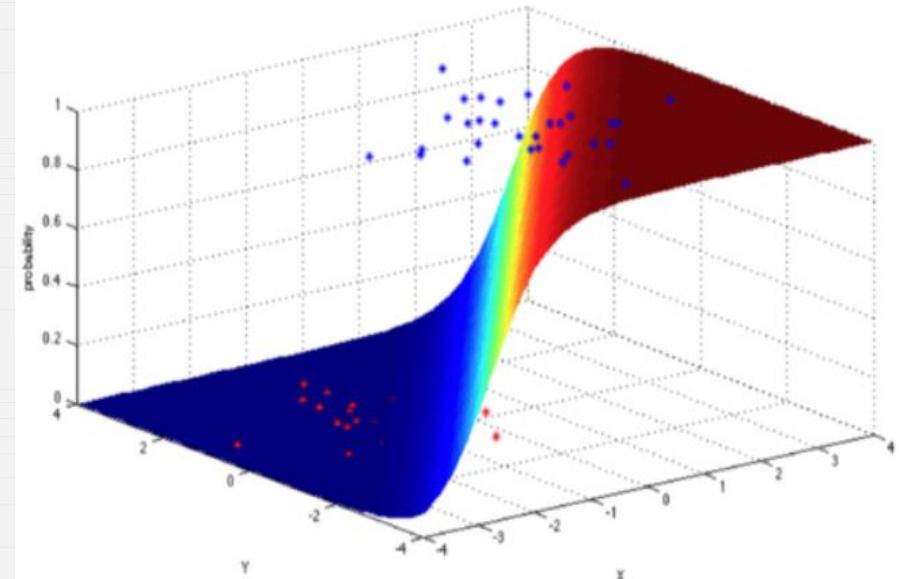
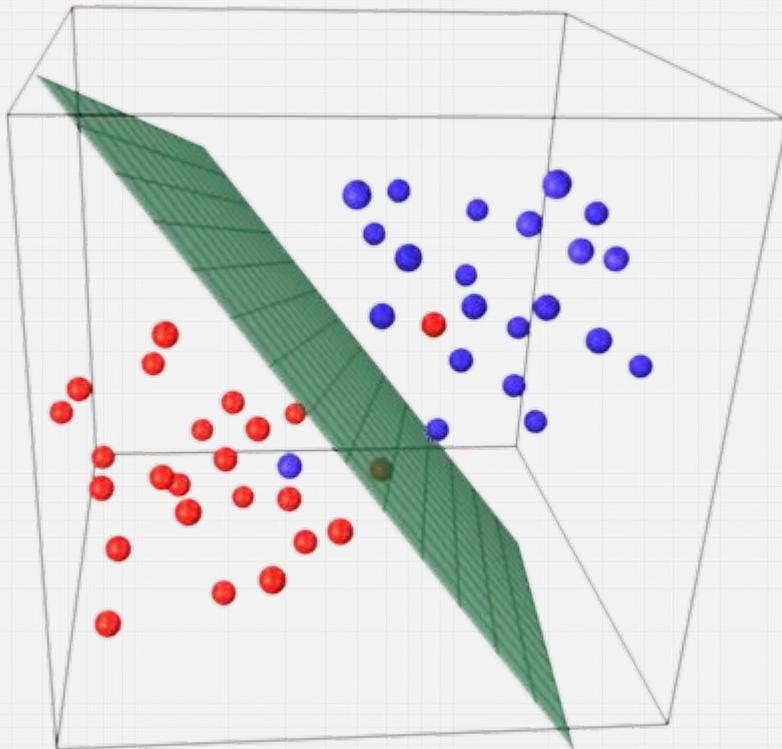
- Пробежимся по самым популярным supervised-моделям(без матана)

Чему научимся:

- деплоить Keras-модели на JVM через DL4j
- обучать LSTM
- пройдем вводный курс(на пару слайдов слайда) "NLP для чайников от чайника"

Линейные модели

Линейные модели

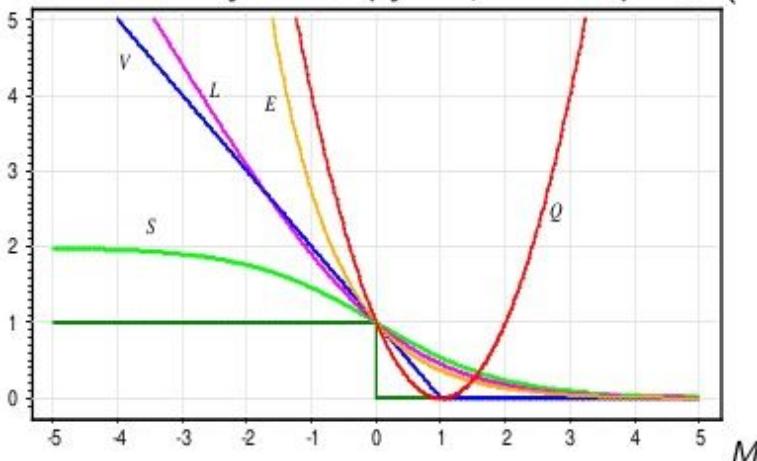


Линейные модели

- Приближают таргет как взвешенную сумму независимых признаков, к которым применяется link function
 - e.g sigmoid
- Решением задачи является подбор веса каждого признака

Непрерывные аппроксимации пороговой функции потерь

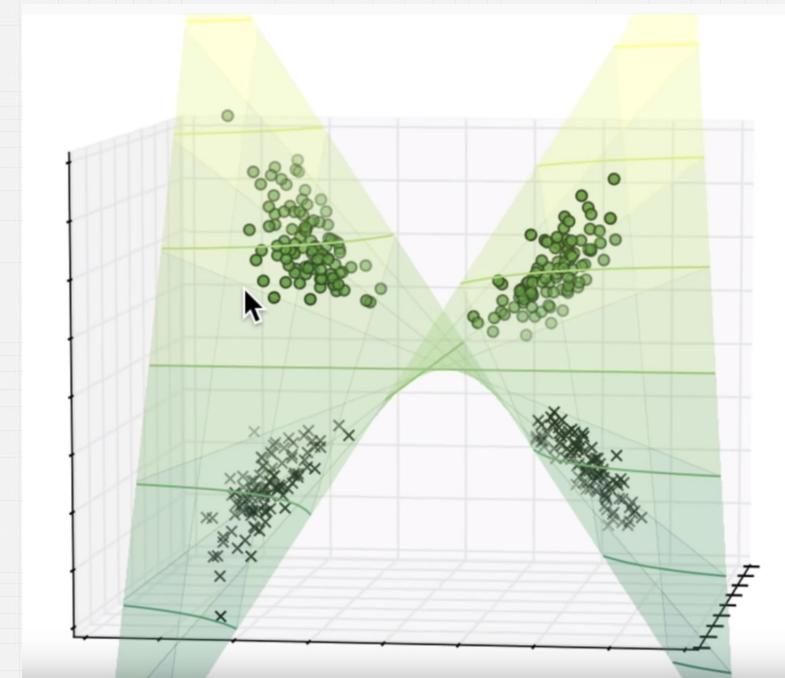
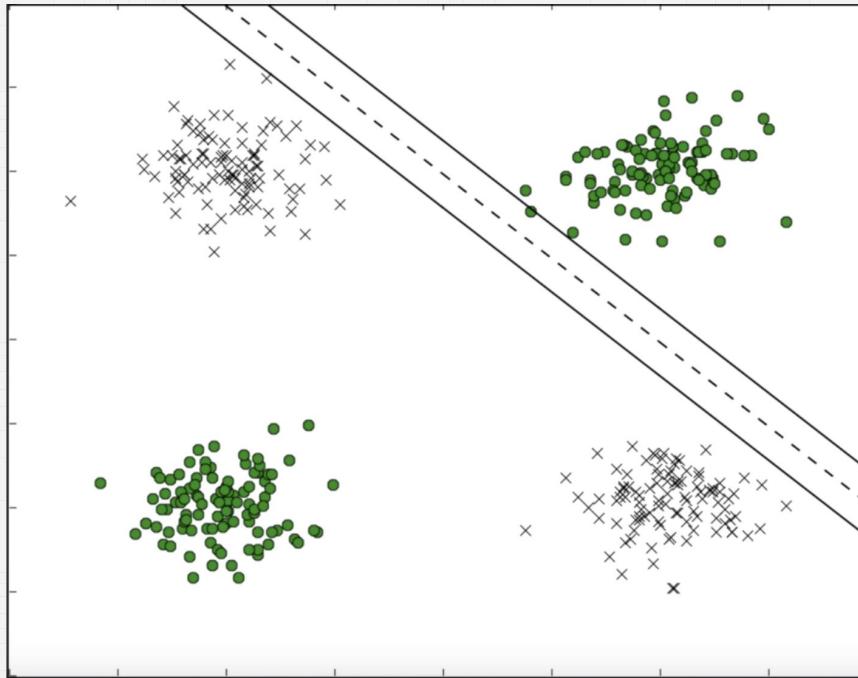
Часто используемые функции потерь $\mathcal{L}(M)$:



- | | |
|-----------------------------|--------------------------------|
| $Q(M) = (1 - M)^2$ | — квадратичная (ЛДФ); |
| $V(M) = (1 - M)_+$ | — кусочно-линейная (SVM); |
| $S(M) = 2(1 + e^M)^{-1}$ | — сигмоидная (нейросети); |
| $L(M) = \log_2(1 + e^{-M})$ | — логарифмическая (LR); |
| $E(M) = e^{-M}$ | — экспоненциальная (AdaBoost). |

Kernel trick

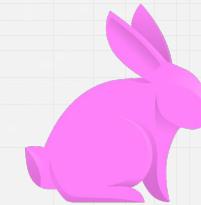
Заменяем скалярное произведение $\langle w, x \rangle$ ядром $K(w, x)$ - симметричной неотрицательной функцией, нелинейно трансформирующей исходное пространство признаков



Линейные модели



MLlib



VOWPAL WABBIT



Линейные модели

Как катим?

В лоб: микросервис на питоне: HTTP/GRPC/Kafka

минусы:

- надо мониторить еще один сервис
- производительность

плюс: просто

Линейные модели

Как еще катим?

В лоб: PySpark UDF

плюс: все просто и быстро работает

минусы:

- python
- pyspark API
- python

Линейные модели

Как еще катим?

```
clf.fit(df[X], df[target])  
json.dumps(list(zip(clf.coef_[0], X)) + ("bias", clf.intercept_[0]))
```

минусы:

- пишем код с нуля, можно ошибиться -> пишем тесты!
- надо поддерживать код в продакшне

плюс: прозрачно, понятно

Линейные модели

Как еще катим?

“Взрослые” фреймворки глубинного обучения:

LR - Нейрон с сигмоидом

SVM - $|w|/2 + C \sum \max[0, 1 - y (wx - b)]^2$

<http://bytepawn.com/svm-with-pytorch.html>

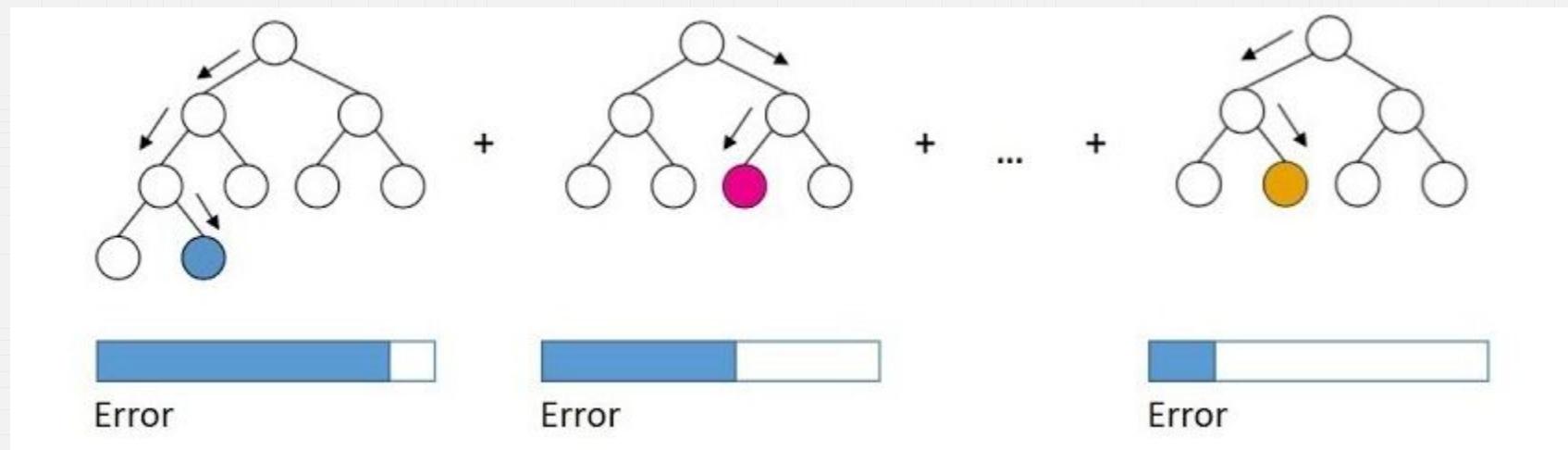
SVM с нелинейным ядром иногда эквивалентен нейросети

минус: иногда попахивает извращением

плюс: оно точно будет production-ready

Бустинг над деревьями

Бустинг над деревьями





Бустинг над деревьями



Yandex
CatBoost

LightGBM

dmlc
XGBoost

Бустинг над деревьями

- 1: **ПРОЦЕДУРА** LearnID3 ($U \subseteq X^\ell$);
- 2: **если** все объекты из U лежат в одном классе $c \in Y$ **то**
- 3: **вернуть** новый лист v , $c_v := c$;
- 4: **найти предикат с максимальной информативностью:**
$$\beta := \arg \max_{\beta \in \mathcal{B}} I(\beta, U);$$
- 5: разбить выборку на две части $U = U_0 \sqcup U_1$ по предикату β :
$$U_0 := \{x \in U : \beta(x) = 0\};$$
$$U_1 := \{x \in U : \beta(x) = 1\};$$
- 6: **если** $U_0 = \emptyset$ или $U_1 = \emptyset$ **то**
- 7: **вернуть** новый лист v , $c_v := \text{Мажоритарный класс}(U)$;
- 8: создать новую внутреннюю вершину v : $\beta_v := \beta$;
построить левое поддерево: $L_v := \text{LearnID3 } (U_0)$;
построить правое поддерево: $R_v := \text{LearnID3 } (U_1)$;
- 9: **вернуть** v ;

Бустинг над деревьями

Аналогично GLM:

- Можно деплоить микросервисами
- Использовать в качестве UDF на питоне

XGBoost

- XGBoost4j
- XGBoost4j-spark

Все работает!(почти)

*<https://github.com/dmlc/xgboost/issues/3689>

Catboost

- JVM-интерфейс сделан в этом году

Наконец-то можно использовать в Spark/Flink!

*<https://github.com/catboost/catboost/issues/858> : до сих пор не сделали predict_proba

*<https://github.com/catboost/catboost/issues/1032> : валит JVM на больших батчах, очищая память

и другие открытые issue.



LightGBM



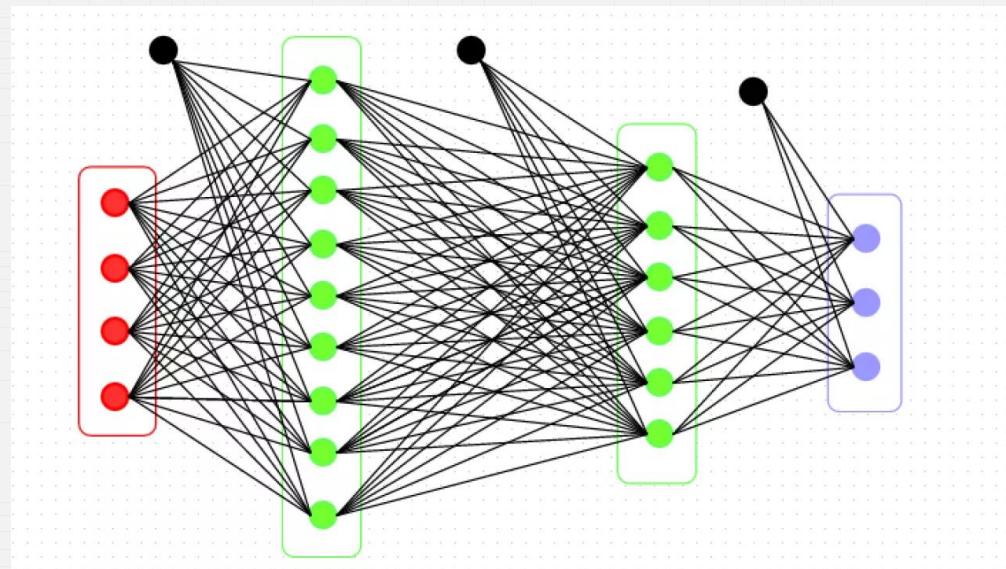
- [https://github.com/Azure/mmlspark ??](https://github.com/Azure/mmlspark)
- udf

Универсальный способ

```
-0.9107080101966857 ) { if ( arr[3] <= -0.7786628901958464 ) { return -0.4999823185309175; } else { return 0.42165444271150354; } }  
else { if ( arr[0] <= -0.9998325407505034 ) { if ( arr[0] <= -1.0000185966491697 ) { return -0.4954312790056301; } else { return  
0.4999910695257724; } } else { if ( arr[3] <= -0.42420934140682226 ) { return -0.4242342444287882; } else { if ( arr[1] <=  
-0.03954122774302959 ) { return -0.2984805929795958; } else { return -0.033533716255187175; } } } } else { if ( arr[3] <=  
0.34858489036560064 ) { return -0.02412296918518087; } else { return 0.3560399278707141; } } } } else { if ( arr[3] <=  
0.6689711511135102 ) { if ( arr[1] <= 0.40496174991130834 ) { if ( arr[0] <= -0.9090864658355712 ) { if ( arr[0] <= -1.00001859664916  
) { return -0.4698252543867944; } else { return 0.4188906855159819; } } } else { if ( arr[2] <= 0.5713316500186921 ) { if ( arr[1] <=  
-0.9107080101966857 ) { return 0.37417867081069234; } else { if ( arr[3] <= 0.5449194312095643 ) { return -0.36618168371363347; } els  
{ if ( arr[1] <= -0.19759044051170346 ) { return -0.23066804159882354; } else { return 0.05297149559702035; } } } else { if ( arr[2]  
<= 0.7123272418975831 ) { if ( arr[3] <= -0.18318788707256314 ) { return -0.46950643287041044; } else { return 0.46143692712645806; }  
else { return -0.17542118662041722; } } } else { return 0.30750447635134753; } } else { if ( arr[3] <= 0.10098057985305788 ) { if ( arr[0] <=  
-0.8649789094924926 ) { return 0.4776185338206315; } else { if ( arr[1] <= -0.1258977726101875 ) { if ( arr[1] <=  
-0.9107080101966857 ) { return 0.4983021622663243; } else { if ( arr[2] <= 0.5324387550354005 ) { return -0.021087915912730442; } el  
{ if ( arr[2] <= 0.748778522014618 ) { return 0.30838429152487445; } else { return 0.05297466911526139; } } } else { return  
0.41578548500643386; } } } else { return 0.4714204567291775; } } } } else { return  
double predictTree1(double[] arr) { if ( arr[1] <= -0.7112968564033507 ) { if ( arr[3] <= 0.37980069220066076 ) { if ( arr[3] <=  
-0.7786628901958464 ) { return -0.31396688972748044; } else { if ( arr[1] <= -0.9993717074394225 ) { return 0.4237110613372529; } els  
{ return -0.1271748691022109; } } } else { if ( arr[1] <= -0.9107080101966857 ) { if ( arr[0] <= -1.0000185966491697 ) { return  
-0.4166733540893253; } else { return 0.4239174463427051; } } else { if ( arr[2] <= 0.748778522014618 ) { if ( arr[3] <=  
1.0260902643203738 ) { if ( arr[0] <= -0.9998325407505034 ) { if ( arr[0] <= -1.0000185966491697 ) { return -0.473356035329457; } els  
{ return 0.4206303330726302; } } else { if ( arr[2] <= 0.6563940644264222 ) { if ( arr[3] <= 1.0177175402641299 ) { return  
-0.1907179349552964; } else { return 0.43172682701641; } } else { if ( arr[3] <= 1.0177175402641299 ) { return 0.305062947183727; }  
else { return -0.490616609274715; } } } } else { if ( arr[2] <= 0.7336338162422183 ) { return 0.4320994560610357; } else { return  
-0.3219973554036538; } } } else { if ( arr[0] <= -0.8502908349037169 ) { return 0.27075216943054614; } else { return  
-0.24551515952565; } } } } else { if ( arr[1] <= 1.1146433353424074 ) { if ( arr[0] <= -0.8649789094924926 ) { if ( arr[3] <=  
0.8585446774959565 ) { if ( arr[0] <= -0.9587144851684569 ) { return 0.2955053145802371; } else { return -0.12715321555638903; } } el  
{ return 0.30263738490576464; } } else { if ( arr[3] <= 0.7630092799663545 ) { if ( arr[2] <= 0.6563940644264222 ) { return  
-0.2557548249035224; } else { if ( arr[2] <= 0.7123272418975831 ) { return 0.4310338148186092; } else { return -0.2699269465228268;  
} } else { if ( arr[2] <= 0.7123272418975831 ) { if ( arr[3] <= 1.0177175402641299 ) { return -0.11457810973080383; } else { return  
0.41855201303664313; } } else { if ( arr[1] <= -0.4930653125047683 ) { return -0.2762778003089093; } else { if ( arr[3] <=  
1.1660366058349612 ) { if ( arr[0] <= 1.6538881063461306 ) { return -0.19232262476449624; } else { return -0.44140231217621667; } }  
else { return -0.08146017943745655; } } } } else { if ( arr[3] <= 1.2241393923759463 ) { if ( arr[1] <= 3.6959618330001835 ) { if  
arr[3] <= 0.6200790405273439 ) { return -0.20216593296440877; } else { return 0.013380837180160357; } } else { return  
0.20250123575455795; } } else { return 0.2934574404901449; } } } }  
  
double predictTree2(double[] arr) { if ( arr[2] <= 0.112373694774888 ) { if ( arr[2] <= -0.17052836716175077 ) { return  
-0.29366942965238446; } else { if ( arr[0] <= -0.8649789094924926 ) { return 0.15153335196415718; } else { return -0.1605794151577604  
} } } else { if ( arr[0] <= -0.559778243032988 ) { if ( arr[0] <= -0.575709074356414 ) { if ( arr[0] <= -0.9998325407505034 ) { if  
arr[0] <= -1.0000185966491697 ) { return -0.26584748764774163; } else { return 0.293413317207701; } } else { if ( arr[1] <=  
-0.8070870041847228 ) { return 0.20622834713480573; } else { if ( arr[3] <= 1.2145292162895205 ) { if ( arr[1] <= -0.5780228674411773  
{ if ( arr[1] <= -0.893078144788742 ) { return -0.2954692684713277; } else { if ( arr[1] <= -0.7122968564033507 ) { return  
0.08309003831926516; } else { return -0.0670250959786568; } } } else { if ( arr[0] <= -0.824189931154251 ) { return  
0.17175870653509395; } else { return 0.029139873131576614; } } } else { if ( arr[1] <= 0.12156610265710701 ) { if ( arr[1] <= 0.6005511602260102 ) { return 0.2551021010072206; } else { return 0.021162215062071211. 11  
26
```

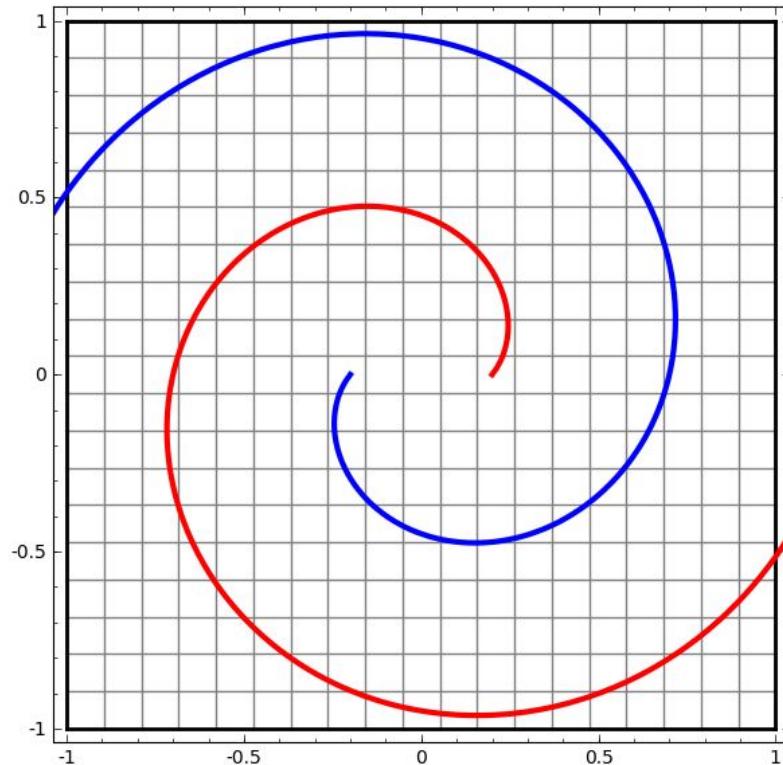
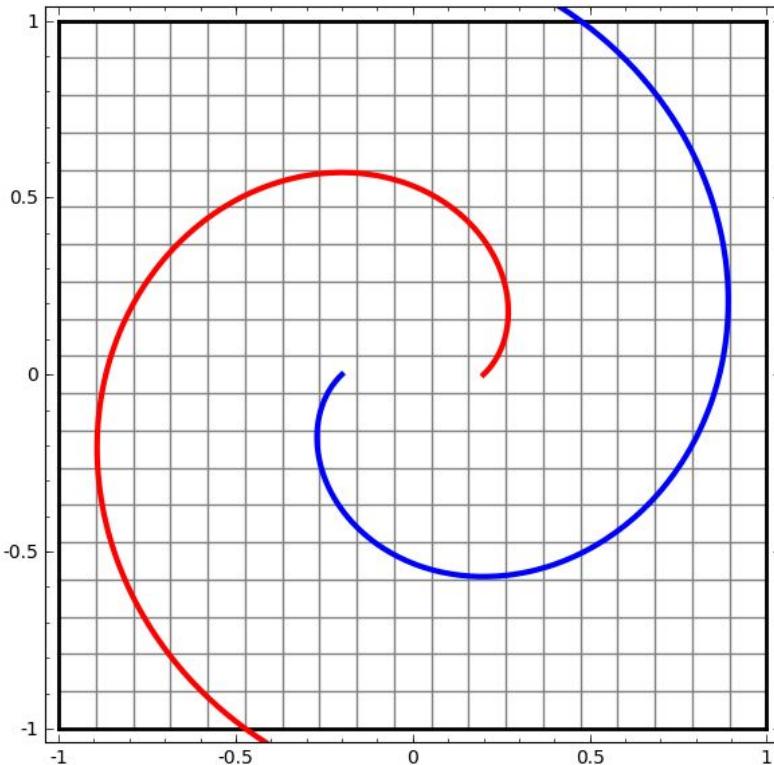
Deep learning

Полносвязник

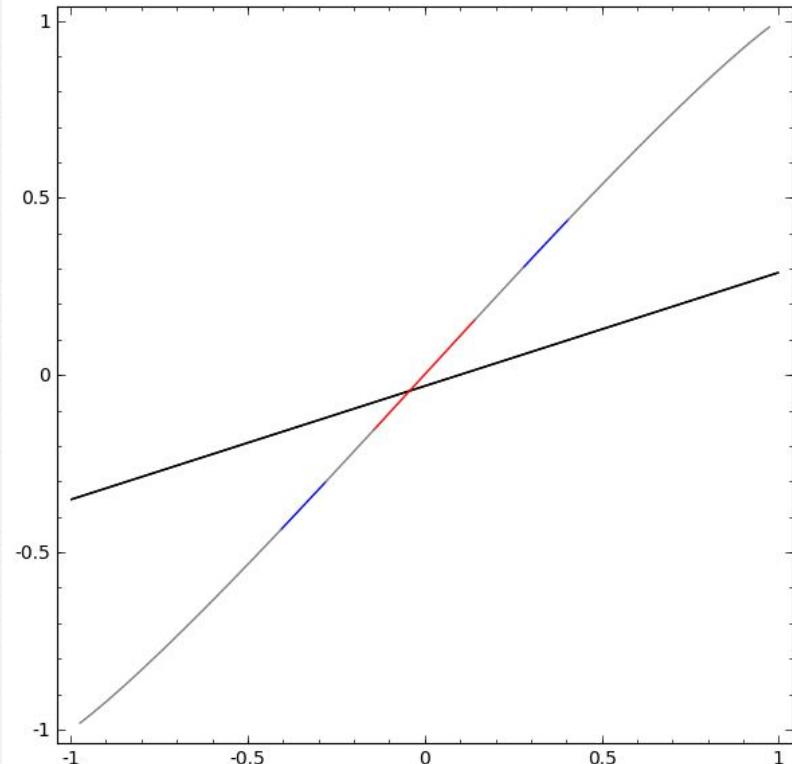
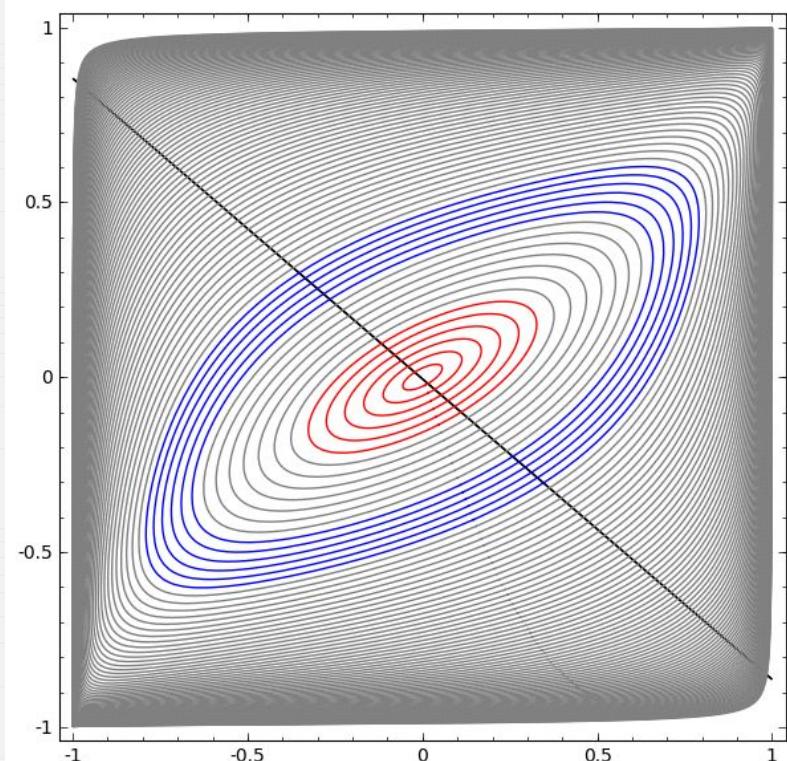


- Стек логистических регрессий(не совсем)

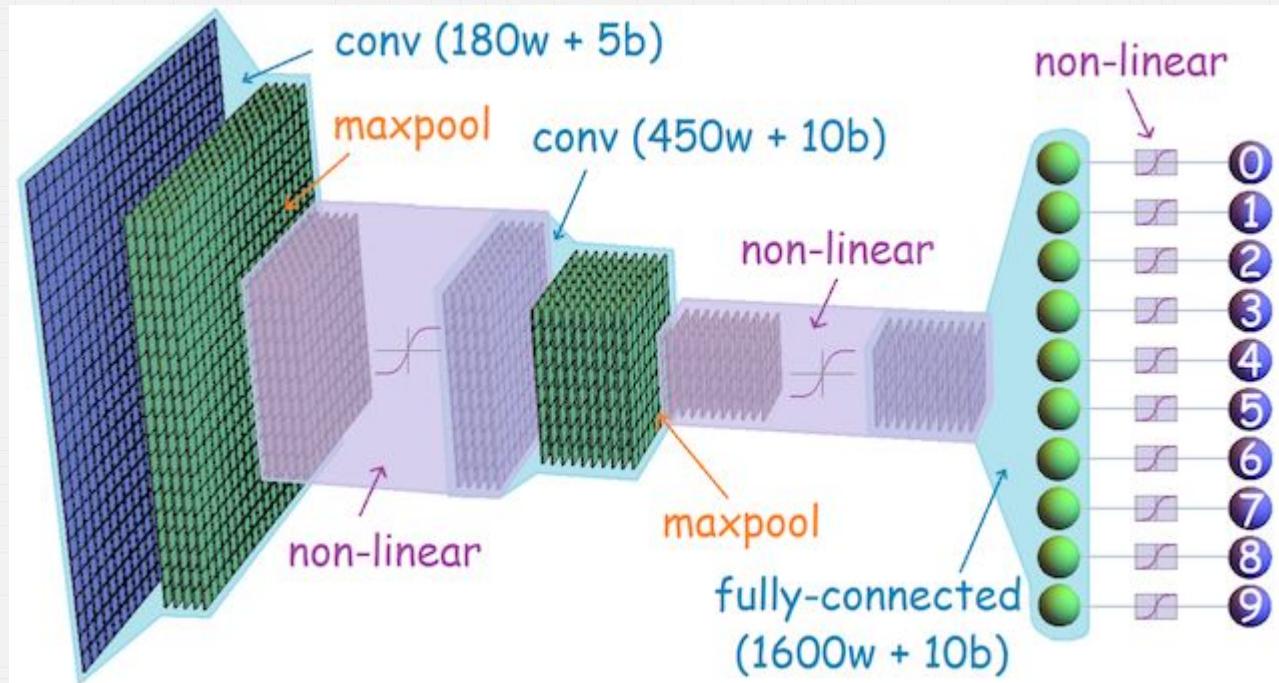
Полносвязник



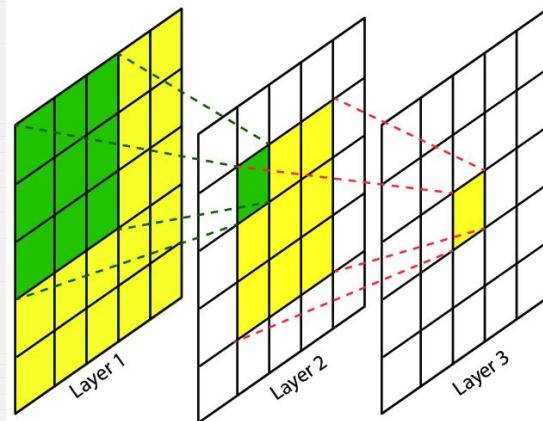
Полносвязник



CNN



- Придуманы, чтобы не переобучаться на больших тензорах(чаще всего это картинки)
- Ограничивают receptive field каждого нейрона, сокращая количество весов
- Эквивалентны применению маленькой полносвязной сети со входной размерностью = размер фильтра



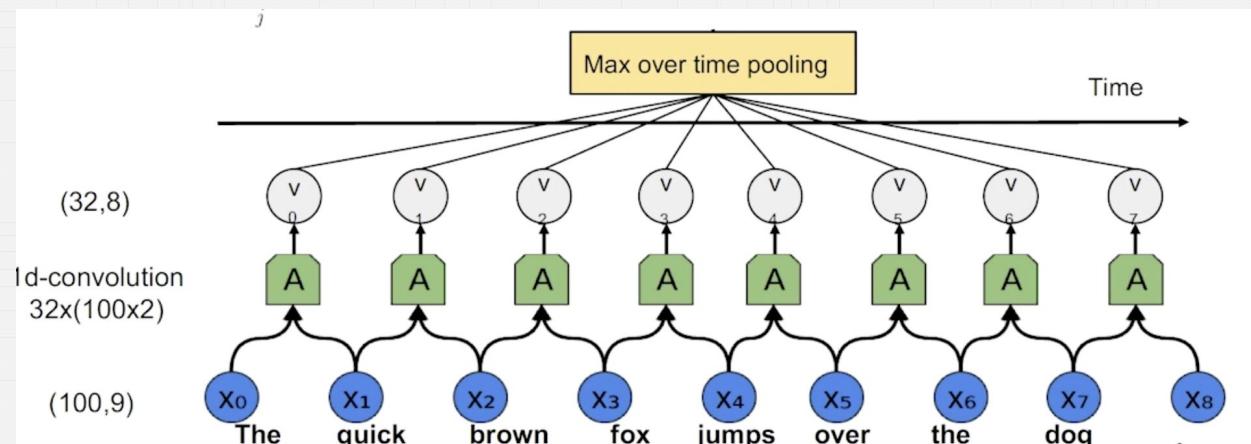
1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

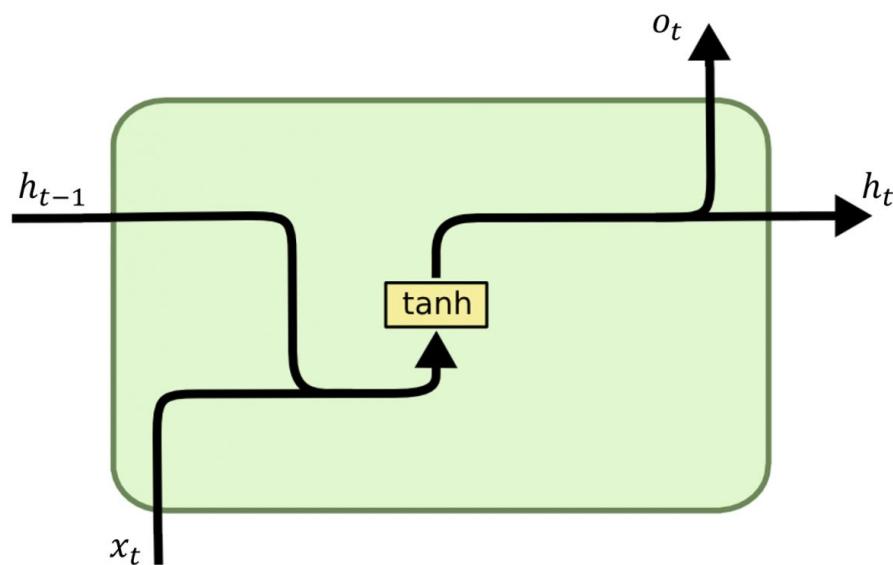
- Бывают одномерные свертки
- В случае с текстами находят словосочетания/устойчивые выражения
- Max over time pooling



RNN

- Все предыдущие сетки не подходят для обработки последовательностей(например, предложений), потому что ничего не знают о контексте. (Свертки знают, но только о локальном)
- Для этой задачи нужна конструкция, которая может “запомнить” информацию о предыдущем значении

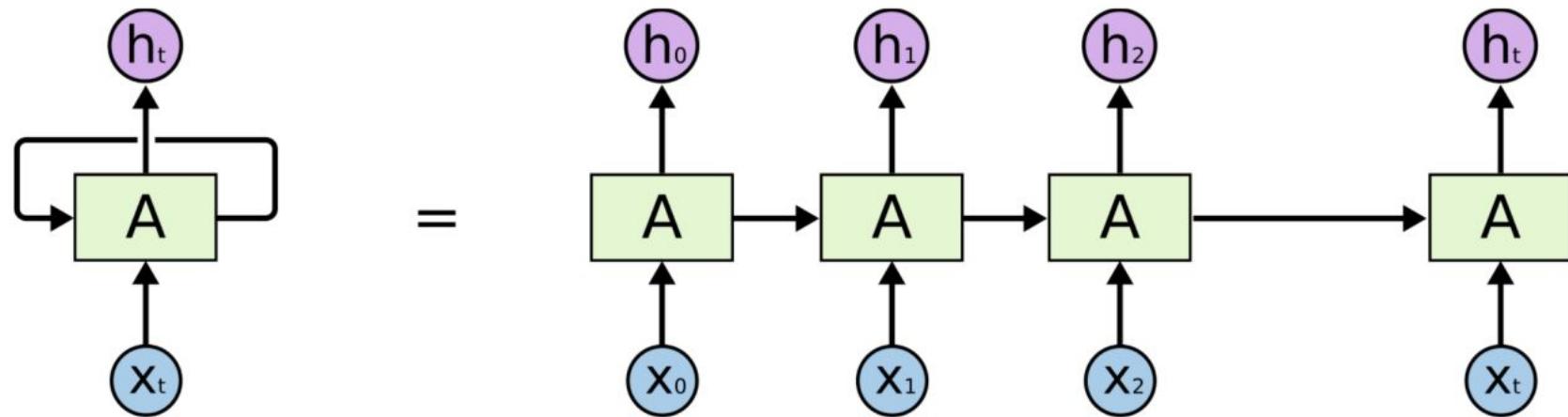
RNN



$$h_t = \sigma_h(i_t) = \sigma_h(U_h x_t + V_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(a_t) = \sigma_y(W_y h_t + b_y)$$

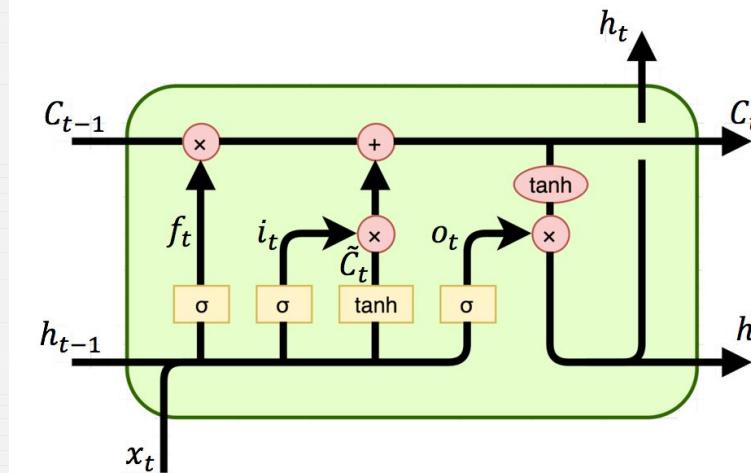
RNN



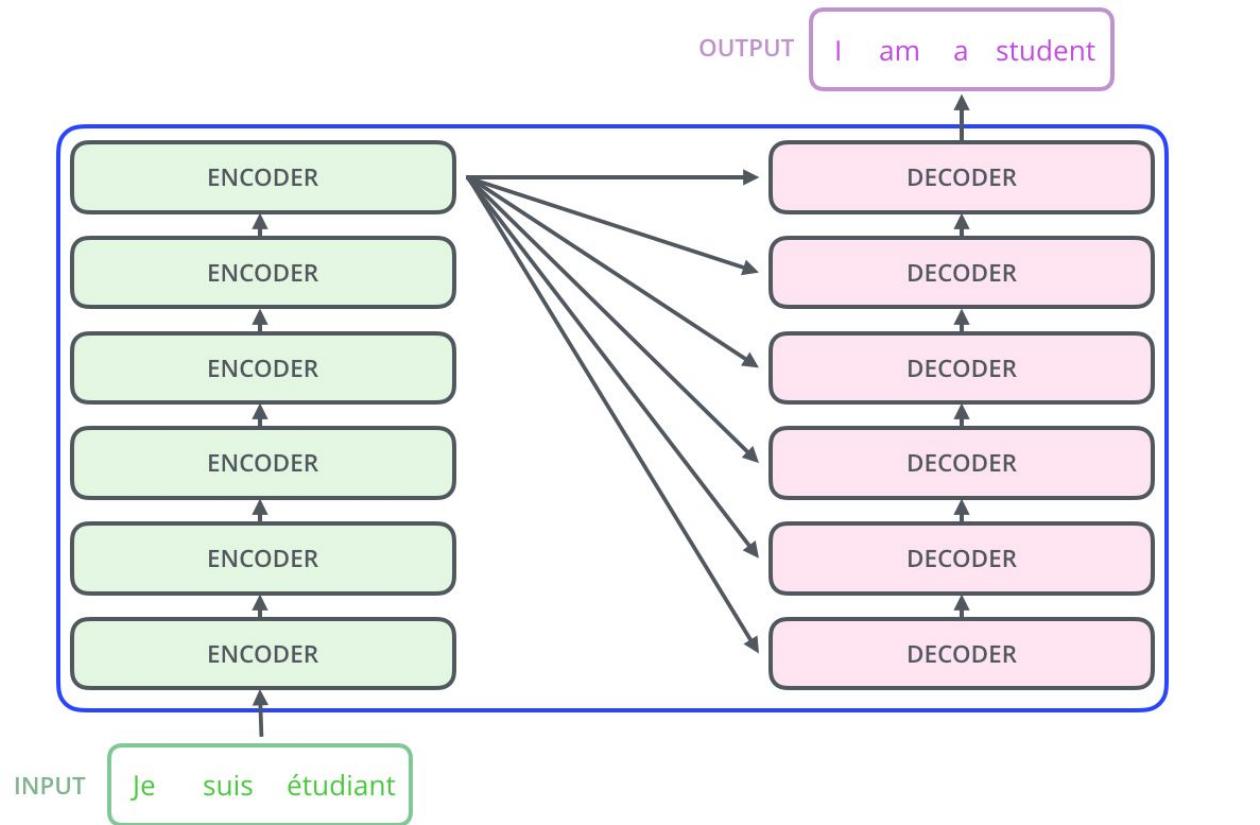
Современные RNN (LSTM, GRU) имеют немного больше “весов” и “вентилей”

примеры:

- контроль “забывания” текущего hidden state
- контроль потоков информации на входе и на выходе



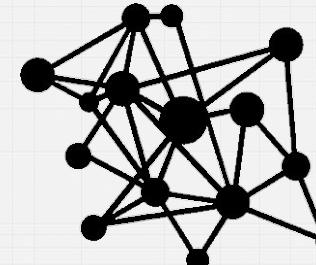
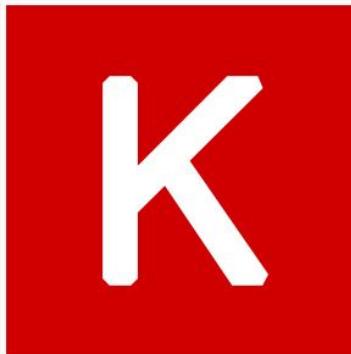
Трансформеры



Deep Learning Frameworks



TensorFlow



DL4J

- Майнстрим
- Низкоуровневый
- Статический граф вычислений
- Есть биндинги ко многим языкам: Java, Go, C++
- TF Serving:
 - Несколько моделей одновременно
 - А/В тестирование
 - HTTP/GRPC API
 - Горячая подмена моделей

- High-level фреймворк над бекендами(TF, Theano, CNTK)
- Все плюсы TF Соответственно
- Построить сетку также просто, как построить пирамидку из детского конструктора

- Динамический граф вычислений
- Легче отлаживать, чем blackbox
- Дает на порядок больше контроля(при желании)
- Для продакшна сыроват, есть хорошие биндинги только на C++ и Python
- Конвертируем модель на другой бекенд или не используем

PyTorch

```
model = model.train()
iteration = 0

for epoch in range(max_epochs):
    for batch in train_loader:
        optimizer.zero_grad()
        input, target = batch
        output = model(input)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

        if iteration % validate_every == 0:
            binary_accuracy = validate(model, val_loader)
            print("After {} iterations, binary accuracy = {:.2f}"
                  .format(iteration, binary_accuracy))

        if iteration % checkpoint_every == 0:
            checkpoint(model, optimizer, checkpoint_dir)
        iteration += 1
```

- Полноценный Deep Learning на JVM
- Неудобный Deep Learning на JVM
- Для инференса пойдет
- Как и все джависты, очень медленно завозят новые фичи
- Может читать модельки Keras!

```
log.info("Building model...")
val conf: MultiLayerConfiguration = new NeuralNetConfiguration.Builder().seed(seed)
  .weightInit(WeightInit.XAVIER)
  .updater(new Nadam(learningRate)).list
  .layer(
    new DenseLayer.Builder()
      .nIn(numInputs)
      .nOut(numHiddenNodes)
      .activation(Activation.RELU)
      .build
  ).layer(
    new OutputLayer.Builder(LossFunction.XENT)
      .nIn(numHiddenNodes)
      .nOut(numOutputs)
      .activation(Activation.SIGMOID)
      .build
  ).build

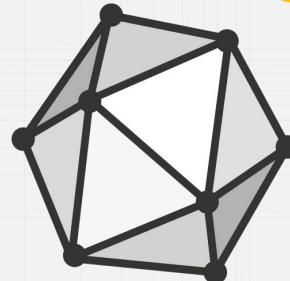
val model: MultiLayerNetwork = new MultiLayerNetwork(conf)
model.init()

val uiServer: UIServer = UIServer.getInstance
val statsStorage = new InMemoryStatsStorage
uiServer.attach(statsStorage)

model.setListeners(new ScoreIterationListener(100), new StatsListener(statsStorage)) //Print score every 100 parameter update

for(_ <- 1 to nEpochs){
  model.fit(trainIter)
}
```

Будущее: ONNX



ONNX

- Единый формат сохранения графов вычислений
- e.g. : можно будет обучить модель в Keras и выкатить ее на IOS
- очень сырь
- поддержка базовых слоев и преобразований есть в большинстве современных DL фреймворков.
- Кроме DL4j :(
- Хороший способ раскатать PyTorch на TF-Бекенд(будет)

Будущее: ONNX

Frameworks

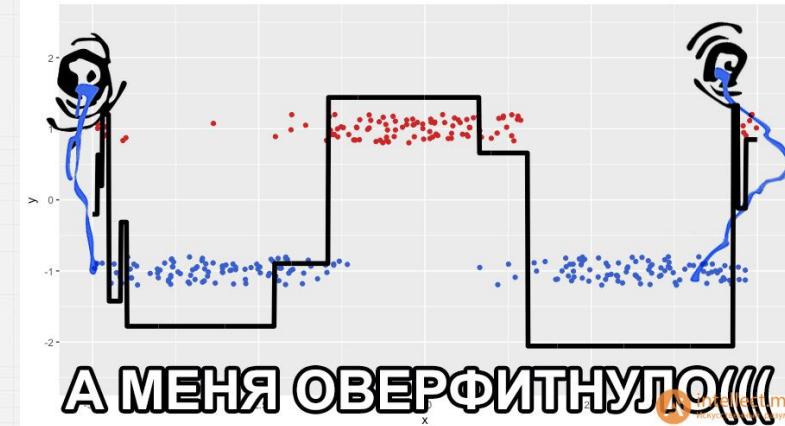
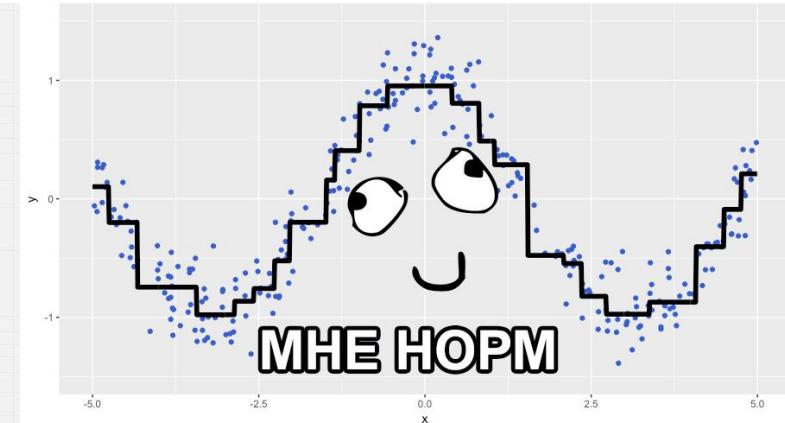


Converters



Практика

When your binary classification outputs 51.5% test accuracy



Практика

wheels of a shopping trolley
be like



Apple proposes 13 new emoji to represent people with disabilities

With a total of 45 options if you count skin tones

By [Jdykstra](#) on March 23, 2018 6:00 pm



Практика: Word Embeddings

The diagram illustrates word embeddings as vectors in a high-dimensional space. Four words are shown with their corresponding vector representations:

- Rome = [1, 0, 0, 0, 0, 0, ..., 0]
- Paris = [0, 1, 0, 0, 0, 0, ..., 0]
- Italy = [0, 0, 1, 0, 0, 0, ..., 0]
- France = [0, 0, 0, 1, 0, 0, ..., 0]

Arrows point from the words to their respective vectors. A curved arrow also points from the word "Paris" to its vector component at index 1.

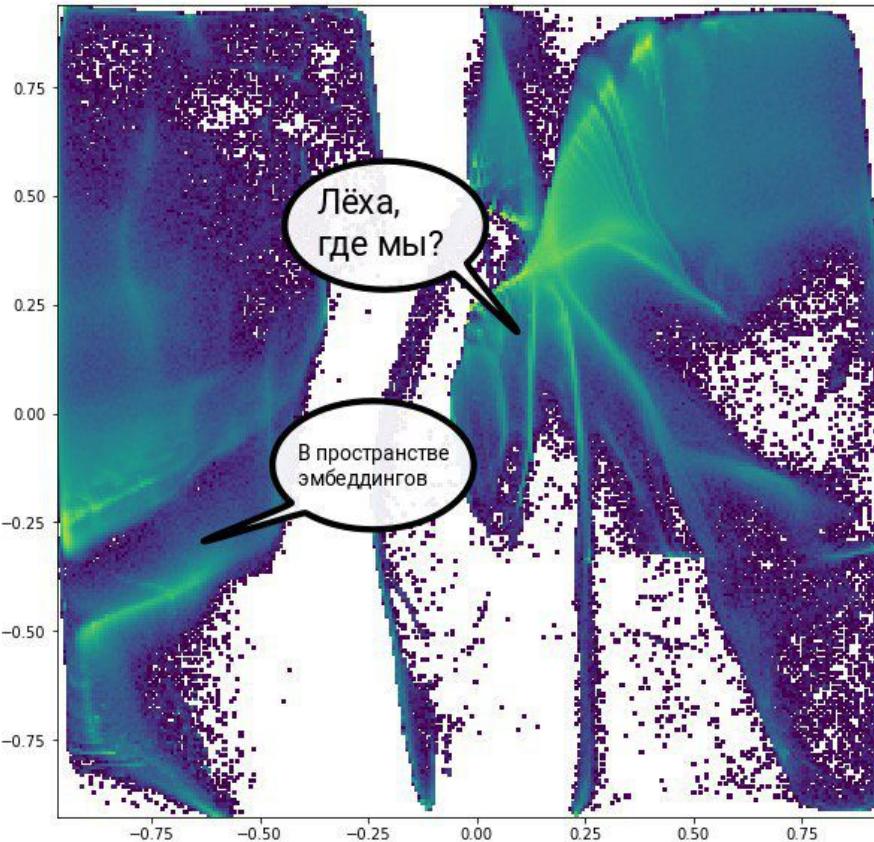
Можно измерить расстояние между предложениями, посчитав косинусную меру схожести между векторами предложений.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

Практика: Word Embeddings

- One-Hot Encoding/Bag of Words ничего не говорит нам о семантике слов, их схожести.
- Хотелось бы уметь делать со словами то же, что и с предложениями

Практика: Word Embeddings



Практика: Word Embeddings

Семантический калькулятор

Здесь можно вычислять отношения: например, «найти слово D, связанное со словом C таким же образом, как слово A связано со словом B». Таким образом можно определять семантические связи между понятиями и решать задачи на аналогии. В форме ввода приведен пример: какое слово относится к слову «Лондон», так же, как «Россия» относится к «Москве»? Ответ — «Великобритания»: Лондон столица Великобритании, а Москва — столица России.

[Подробнее...](#)

новичок_NOUN



профессионал_NC

студент_NOUN



Новостной корпус

1. **рабочие**_{VERB} 0.40
2. **сельхозинstitута**_{NOUN} 0.40
3. **колин мадсон**_{PROPN} 0.39
4. **java-программирования**_{NOUN} 0.38
5. **преподаватель**_{NOUN} 0.38

Семантический калькулятор

Здесь можно вычислять отношения: например, «найти слово D, связанное со словом C таким же образом, как слово A связано со словом B». Таким образом можно определять семантические связи между понятиями и решать задачи на аналогии. В форме ввода приведен пример: какое слово относится к слову «Лондон», так же, как «Россия» относится к «Москве»? Ответ — «Великобритания»: Лондон столица Великобритании, а Москва — столица России.

[Подробнее...](#)

Москва_PROPN



Россия_PROPN

Лондон_PROPN



???

Частотность слова

Высокая Средняя Низкая

НКРЯ и Wikipedia

1. **англия**_{PROPN} 0.58
2. **европа**_{PROPN} 0.54
3. **великобритания**_{PROPN} 0.52
4. **страна**_{NOUN} 0.48
5. **франция**_{PROPN} 0.47

Практика

Давайте экономить ресурсы кластера

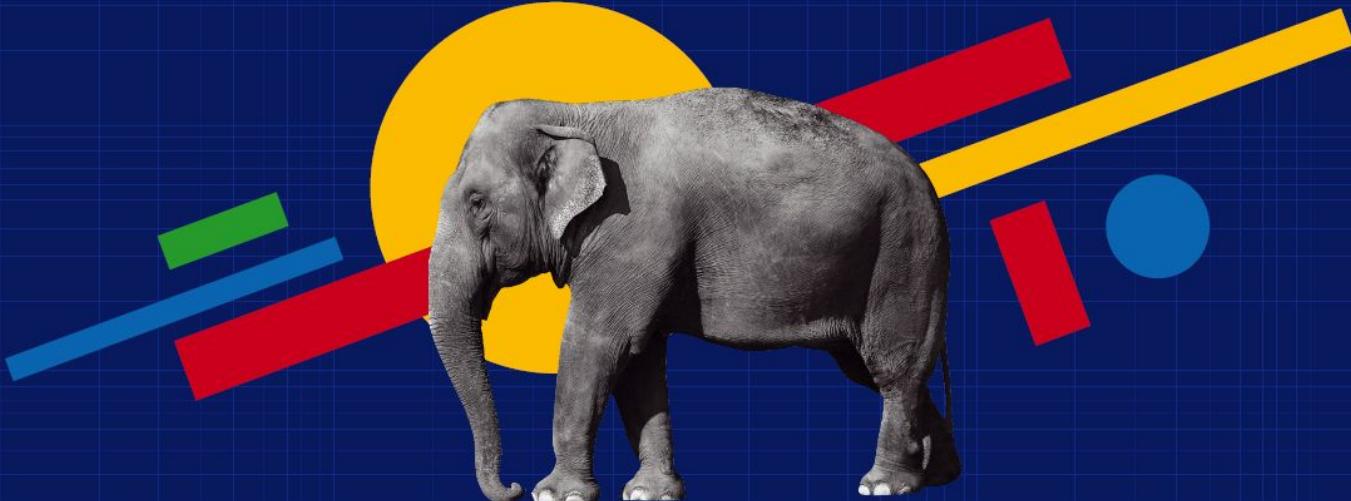
Не забудьте выполнить
`./configureTopics.sh`
для установки небольшого retention

Заключение

Сегодня мы:

- Узнали/освежили интуитивное понимание самых популярных supervised-моделей
- Узнали немного о Deep Learning в обработке естественного языка
- Сами обучили свою модель
- Сделали из нее масштабируемое streaming-приложение

Алексей Бабенков
babenkov25@gmail.com



BIG DATA IS LOVE

NEWPROLAB.COM