Introduction to ML Methods and Applications

Rybin Dmitry

September 27, 2020

Table of Contents

Popularity and Growth

Why did Data Science and Machine Learning became so popular only during the last decade? Two possible triggers:

- Big data collection and storage became accessible to more businesses. Gigabytes of data is not a privilege of huge corporations now.
- Many high-profitable applications for businesses.

General Structure

Data + Goal $\rightarrow$ Objective function + Model + Optimization Process.

Data: some list of **objects** X_i (e.g. a house) with **label** y_i (e.g. price of a house) for each object. Object is described by its **features** (e.g. a house is described by its area, neighbourhood, number of rooms, etc).

Goal: given the data, get the algorithm that **generalizes to new objects**, namely, given new object X_{new} it should predict its true label y_{new} as good as possible (e.g. price of a house with small error).

Objective function: MSE, MAE, Accuracy, ROC AUC, F1, ..., arbitary function (but better if easily differentiable).

Model: Regression, Decision Trees, k-Nearest Neighbours, SVM, CNN, RNN, Transformers, ...

Optimization Process: usually gradient descent with some enhancements.

Linear Regression, honest MSE derivation

Suppose that our data consists of points (x_i, y_i) on $\mathbb{R}^2$, $i = 1, 2, ..., n$, and we know that they belong to some line $y = ax + b$, but the data is noisy:

$$y_i = a \cdot x_i + b + \varepsilon_i,$$

where ε_i is noise with Gaussian distribution $\mathcal{N}(0, \sigma^2)$, with probability density function:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}.$$

How to choose the best line?

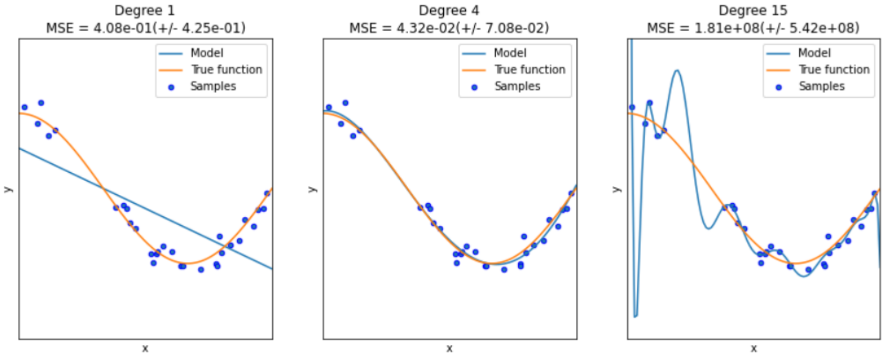
Idea: $\varepsilon_i = y_i - a \cdot x_i - b$, "likelihood" of error ε_i is $\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - a\cdot x_i - b)^2}{2\sigma^2}}$, thus:

$$L(X, y) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - a\cdot x_i - b)^2}{2\sigma^2}} = C_1 \cdot e^{-C_2 \cdot \sum (y_i - a\cdot x_i - b)^2} \to \max$$

If the noise is known to be of other nature, for example Cauchy or Geometric distribution, then the objective will change.

Underfit, Overfit

What if the relationship between x_i and y_i is less trivial? Let's generate more features: $1, x_i, x_i^2, ..., x_i^k$, and do *polynomial regression*.



We end up with the fundamental problem of overfitting/underfitting in ML. A lot of research is dedicated for general methods and special cases to efficently avoid overfitting/underfitting.

Underfit, Overfit, Vanishing and Exploding Gradients

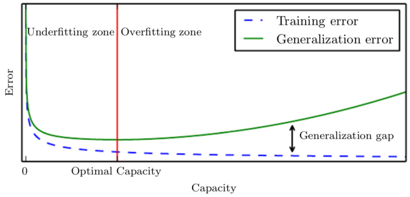


Figure: Fitting curve

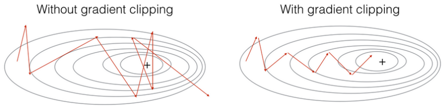


Figure: Exploding gradient

Apart from underfitting/overfitting problem, the most common one is vanishing/exploding gradients.

When the objective function or optimization method was poorly selected, the gradient can take arbitrary big values or stay arbitrary small for long periods.

Table of Contents

Decision Trees - a simple classification algorithm

Consider the famous dataset - "Titanic passengers".

Each object is a passenger.

Features are: Ticket class, Sex, Age, # of siblings / spouses, # of parents / children, Ticket number, Passenger fare, Cabin number, Port of embarkation.

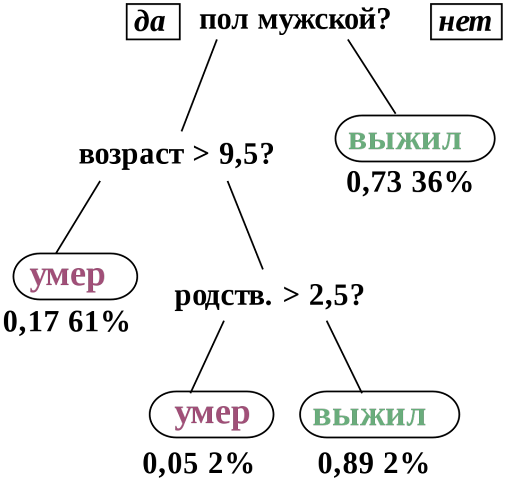
Target is survival (0/1).

How to solve this problem? Linear regression is definitely not the best approach.

Idea: divide the data into parts based on very important features e.g. "Age", "Sex", and if some class is dominant in part, predict it for all objects from the part.

Mathematically one can write split efficiency as combination of Variances/Entropy of target function at childs and root nodes.

Decision Trees



Gradient Boosting Decision Trees

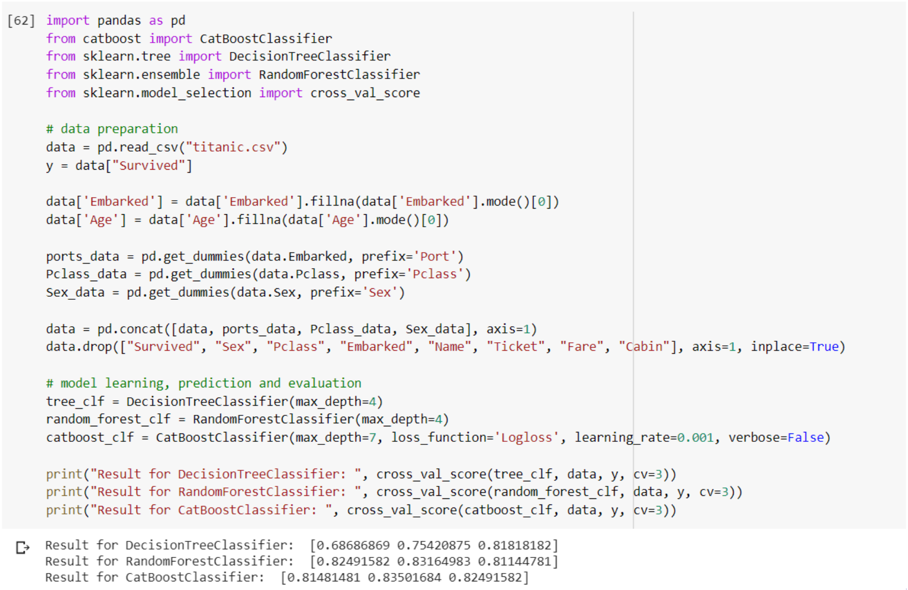
Basic enhancement of Decision Tree Classifier is Random Forest Classifier - just generate many trees and average their prediction.

Boosting is the following algorithm: train first model, subtract its prediction from the target, then train second model to predict the error produced by the first model and so on.

Boosting on Decision Trees is a very strong baseline in many Classification and Regression problems: Credit scoring, House pricing, malicious bot traffic, CERN experiments, selling prediction(combination of time-series analysis and gradient boosting), customer uplift...

Popular solutions: xgboost, LGBM(Microsoft), catboost(Yandex).

Decision Trees + Random Forest + GBDT in code



```
[62] import pandas as pd
     from catboost import CatBoostClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import cross_val_score

     # data preparation
     data = pd.read_csv("titanic.csv")
     y = data["Survived"]

     data['Embarked'] = data['Embarked'].fillna(data['Embarked'].mode()[0])
     data['Age'] = data['Age'].fillna(data['Age'].mode()[0])

     ports_data = pd.get_dummies(data.Embarked, prefix='Port')
     Pclass_data = pd.get_dummies(data.Pclass, prefix='Pclass')
     Sex_data = pd.get_dummies(data.Sex, prefix='Sex')

     data = pd.concat([data, ports_data, Pclass_data, Sex_data], axis=1)
     data.drop(["Survived", "Sex", "Pclass", "Embarked", "Name", "Ticket", "Fare", "Cabin"], axis=1, inplace=True)

     # model learning, prediction and evaluation
     tree_clf = DecisionTreeClassifier(max_depth=4)
     random_forest_clf = RandomForestClassifier(max_depth=4)
     catboost_clf = CatBoostClassifier(max_depth=7, loss_function='Logloss', learning_rate=0.001, verbose=False)

     print("Result for DecisionTreeClassifier: ", cross_val_score(tree_clf, data, y, cv=3))
     print("Result for RandomForestClassifier: ", cross_val_score(random_forest_clf, data, y, cv=3))
     print("Result for CatBoostClassifier: ", cross_val_score(catboost_clf, data, y, cv=3))
```

```
Result for DecisionTreeClassifier:  [0.68686869 0.75420875 0.81818182]
Result for RandomForestClassifier:  [0.82491582 0.83164983 0.81144781]
Result for CatBoostClassifier:  [0.81481481 0.83501684 0.82491582]
```

Table of Contents

A Neural Network is...

a mathematical model for neural brain structure, based on old(1943) and wrong concept by neurophysiologist Warren McCulloch and genius mathematician Walter Pitts.

For us it is directed acyclic graph (usually Layered) with Input Layer, Hidden Layers and Output Layer.

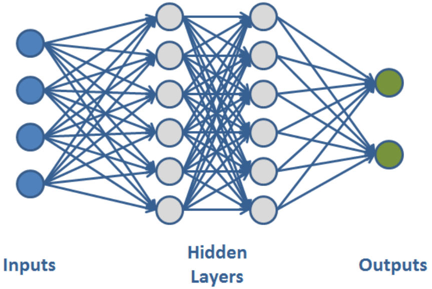


Figure: Dense Layered Neural Network

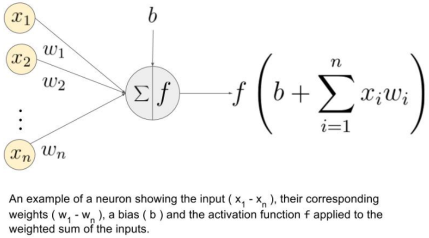

x_1
x_2
x_n
w_1
w_2
w_n
b
Σ f
$f\left(b + \sum_{i=1}^{n} x_i w_i\right)$
An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

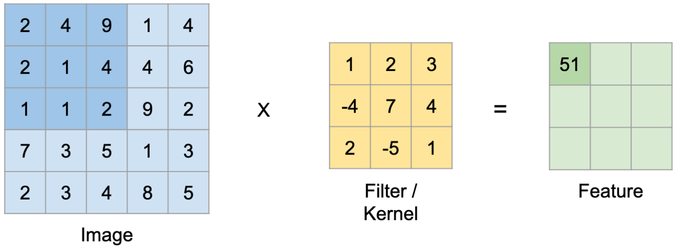
Figure: Neuron activation

Computer Vision

Standard problems from Computer Vision include:

- Classification e.g. recognize handwriting, ban inappropriate content.
- Object detection e.g. detect humans on the video in forbidden areas, detect forest fires / car traffic / new buildings based on satellite images.
- Face recognition.

The common architecture is Convolutional Neural Network which, apart from usual layers, contains Convolutional Layers.

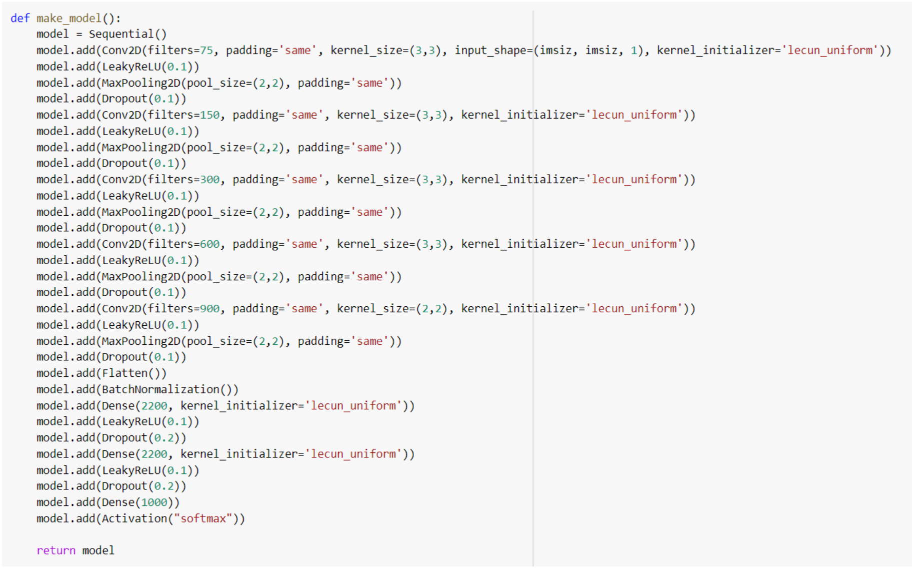


Training Neural Networks

How to do gradient descent with 10 million dimensions and difficult intertwined objective? How to do a single step when the number of objects is hundreds of thousands or millions? How to handle overfitting/underfitting? How to handle vanishing/exploding gradients?

- Optimization is performed by batches (e.g. by sample of 256 images) otherwise one step would take too long.
- Gradient descent is done with special optimizers (Adam, RMSProp).
- The gradient is computed via special technique - backpropagation. Tensorflow and Pytorch are easy-to-use libraries for it.
- Activation function historically was the logistic curve, but due to vanishing gradients many others were tried. Currently mostly used are ReLU and LeakyReLU.
- To avoid overfitting, some neurons are randomly dropped from the learning scheme each step (Dropout layers). Other way to prevent overfitting is to use cropping/rotation/translation/scaling to generate more data.

Convolutional Neural Network Initialization code



```python
def make_model():
    model = Sequential()
    model.add(Conv2D(filters=75, padding='same', kernel_size=(3,3), input_shape=(imsiz, imsiz, 1), kernel_initializer='lecun_uniform'))
    model.add(LeakyReLU(0.1))
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.1))
    model.add(Conv2D(filters=150, padding='same', kernel_size=(3,3), kernel_initializer='lecun_uniform'))
    model.add(LeakyReLU(0.1))
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.1))
    model.add(Conv2D(filters=300, padding='same', kernel_size=(3,3), kernel_initializer='lecun_uniform'))
    model.add(LeakyReLU(0.1))
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.1))
    model.add(Conv2D(filters=600, padding='same', kernel_size=(3,3), kernel_initializer='lecun_uniform'))
    model.add(LeakyReLU(0.1))
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.1))
    model.add(Conv2D(filters=900, padding='same', kernel_size=(2,2), kernel_initializer='lecun_uniform'))
    model.add(LeakyReLU(0.1))
    model.add(MaxPooling2D(pool_size=(2,2), padding='same'))
    model.add(Dropout(0.1))
    model.add(Flatten())
    model.add(BatchNormalization())
    model.add(Dense(2200, kernel_initializer='lecun_uniform'))
    model.add(LeakyReLU(0.1))
    model.add(Dropout(0.2))
    model.add(Dense(2200, kernel_initializer='lecun_uniform'))
    model.add(LeakyReLU(0.1))
    model.add(Dropout(0.2))
    model.add(Dense(1000))
    model.add(Activation("softmax"))

    return model
```

Chinese character recognition

This CNN is from one of our homework on recognition of 1000 different chinese characters. Full dataset contains millions of black and white 64×64 images. Model has $\approx 2 \cdot 10^7$ parameters and final accuracy of 98.07%.

Training on CPU takes > 8 hours, on GPU it takes ≈ 40 minutes. Google has invented special ASIC hardware TPU (Tensor Processing Unit) which allows to train huge Neural Networks within minutes.

Remark: a program can be faster on GPU if its "bottleneck" is a simple operations as addition, or multiplication, or FFT, on a data that is 2 times larger than the CPU available memory limit.

Remarks on Computer Vision

Deep Learning is a general term used describe Deep Neural Networks solutions. In CV Deep Learning is one of the ways to achieve State-of-the-Art classification results.

In 2012 ImageNet competition one particular Deep CNN called AlexNet has beaten other models by more than 10% in accuracy, this was the moment researchers and the industry started to pay much more attention to CV.

Such models became much better than humans in image classification. And since ground truth is produced by humans, further research is focused on more difficult tasks such as segmentation or model compressing from millions of parameters to thousands thus making the inference faster.

Right now (2020) the demand for CV ML Engineers and Research is overwhelming in the industry.

Table of Contents

Tasks in NLP

Wide range of applications:

- Text classification.
- Text translation, even multi-lingual translation is doable.
- Text generation.
- Document clusterization.
- Detection of simple bugs in code.

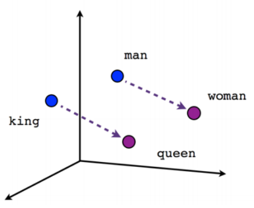
We always consider text as a sequence of letters or a sequence of words.

First approaches were based on Markov chains with hidden parameters (1950s) and prediction of the next word in the text. They were too weak.

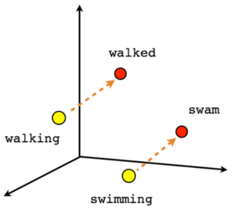
Simple solutions in NLP

Fast and simple solution of sufficient quality for Text classification and document clusterization can be produced by taking text to vector based on the word count (dimension is incredibly big!) and perform projection on a subspace of small dimension by SVM (Principal Component Analysis), then do linear regression.

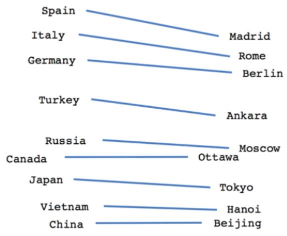
Another Interesting solution is Word2Vec - pre-trained embedding of English words into high-dimensional vector space with great behaviour. Close words are close vectors, and even some semantic relationships hold:



Male-Female



Verb tense



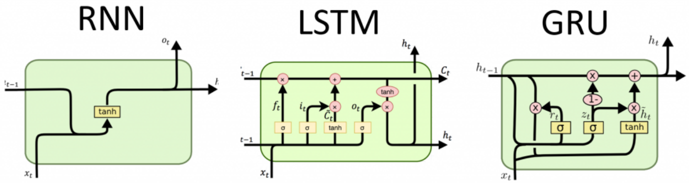
Country-Capital

State of the art solutions in NLP

First breakthroughs started with realization that large data allows generation of sentences by predicting the next *character* instead of the next word.

Recurrent Neural Networks (RNNs) and very similar models were dominating in the NLP a few years ago. Simplified description is the following:

The model has a hidden state(vector in $\mathbb{R}^N$) which should "contain" information about current "context". With each new character the hidden state is adjusted and output is produced according to the last input and the current hidden state.



Transformers

Another major breakthrough was the paper "Attention Is All You Need" (June 2017), where authors introduced "Transformer" architecture (Deep Neural Networks focused on Attention mechanism). This led to significant improvements of objectives in nearly all NLP tasks. Soon a variety of modifications like BERT and RoBERTa were produced.

In June 2020 Generative Pre-trained Transformer 3 (GPT-3) was released by OpenAI. It has 175 billion parameters (human brain has 85-86 billion neurons on average). The training costed millions of dollars.

Can do text summarization and answering questions. With small corrections by a human it can produce essays and articles on a variety of topics (indistinguishable from human). Capable of producing code for simple Python and Ruby tasks.

Remark: one can use pre-trained BERT or GPT for his research or adjustment for his tasks.

Table of Contents

Problem formulation

Recommendation systems are used in Yandex, Spotify, Netflix, Amazon, Google.

Given a set of users U, a set of items I and some known results r_{ui} with $(u, i) \in K$ - the interaction between user u and item i (for example, $r_{ui} = 1$ if user liked the movie and -1 if he disliked it, or $r_{ui} = \log_2(1 + N)$, where N is the number of times user bought some item on Amazon), we wish to approximate r_{ui} for the unseen pairs and do best recommendations for users.

Solutions from simplest to hardest: average r_{ui} along users and make constant prediction; K-Nearest Neighbours(Collaborative Filtering); Matrix Factorization; Factorization Machines.

Simple solutions description

Fix i and let U' be the set of $(u, i) \in K$.

Constant prediction: Predict $\overline{r}_{ui} = \frac{1}{|U'|} \sum\limits_{r_{u'i} \in K} r_{u'i}$.

K-Nearest Neighbours:
Consider users u, u' as vectors in $\mathbb{R}^{|I|}$.
Normalize vectors: $\overline{u} = (r_{ui} - \overline{r}_{ui})_{i \in I}$
Define metric between them as cosine metric between normalized vectors: $d(u, u') = \cos(\overline{u}, \overline{u}')$.
Find K-Nearest Neighbours to the current vector and average their predictions.

Matrix Factorization: Fix k. Suppose that $\exists v_i, v_u \in \mathbb{R}^k, i \in I, u \in U$ such that $r_{ui} = \langle v_u, v_i \rangle$. Learn v_u, v_i via gradient descent(with appropriate reguralization). It is the same as learning matrix factorization $R = V_U \cdot V_I$.

Factorization Machines

Factorization Machines is a generalization of Matrix Factorization. Suppose that apart from the $r_{ui} \in K$ we know some other user and item features, which is often the case. For example we know that user marked Rock as his favorite music genre, or that a song belongs to album from trendings.

Let $x \in \mathbb{R}^d$ denote the vector and let us find the prediction y as the second order polynomial regression:

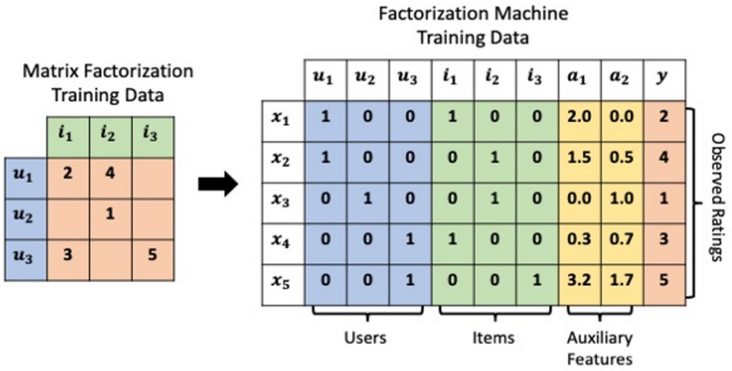
$$y(x) = w_0 + \sum w_s \cdot x_s + \sum w_{st} \cdot x_s x_t.$$

The number of weights is $\frac{d^2+d+2}{2}$. Let us restrict to the case $w_{st} = \langle v_s, v_t \rangle$, where $v_s, v_t \in \mathbb{R}^r$. Then the number of weights becomes $rd + d + 1$. To recover FM model we should put $x_i, x_u = 1$ for user and item, then

$$y(x) = w_0 + w_u + w_i + \langle v_u, v_i \rangle.$$

Good Open Source realization: LightFM.

Gradient descent


Matrix Factorization Training Data
Factorization Machine Training Data
Observed Ratings
Users
Items
Auxiliary Features

Matrix Factorization Training Data

	i_1	i_2	i_3
u_1	2	4	
u_2		1	
u_3	3		5

Factorization Machine Training Data

	u_1	u_2	u_3	i_1	i_2	i_3	a_1	a_2	y
x_1	1	0	0	1	0	0	2.0	0.0	2
x_2	1	0	0	0	1	0	1.5	0.5	4
x_3	0	1	0	0	1	0	0.0	1.0	1
x_4	0	0	1	1	0	0	0.3	0.7	3
x_5	0	0	1	0	0	1	3.2	1.7	5

Two most popular approaches to do optimization with Matrix Factorization are:

- Stochastic Gradient Descent: batch optimization with batch size equal to 1.
- EM algorithm: Fix V_U, optimize V_I by regression, then fix V_I, optimize V_U and so on.

Table of Contents

RL is very challenging and very popular among researchers. General problem formulation is the following: we are given an **environment** (a game, a map) and an **agent** (computer bot playing a game or trying to navigate on the map). Each moment of time t game is in some state S_t and agent can perform **action** A_t and for each his action we give him a reward $R(S_t, A_t)$.

Mathematics behind: One way is to learn Markov chain probabilities of performing some action which maximizes the reward, namely, we learn a policy function

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s),$$

to optimize the reward function

$$\mathbb{E}_\pi(\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s).$$

note that the sum is calculating the future rewards with an exponential decay γ. This way we can either try to learn immediate reward (γ is small) or long-term rewards (γ is close to 1).

Underfit/Overfit

As usual, we face the classical difficulties - underfitting and overfitting. In Reinforcement Learning both of these are still very challenging.
Overfitting examples:

- People tried to teach a car model to drive without crashes, the result was a car learning to drive backwards because this way crashes were not registered.
- Tetris bot learned to pause the game when he is in a definite losing position.
- Sometimes bots learn unexpected game behaviour unknown even to developers and experts.

Underfitting examples:

- All the tasks mentioned on previous slides RL can't handle.

Games dominated by RL models

- Chess
- Atari games dataset - humans still need less frames to learn, but SotA is close to beating us.
- Go - model description is available but it takes several millions of dollars to train.
- Dota - RL models have won against world champion team.
- Starcraft - models lose to pro players if click speed is bounded and even if it is unbounded they still cannot beat best players. Note that the number of degrees of freedom is essentially infinite. The learning again costs millions of dollars.