Neural Networks

Semester 1

Practical works for GRIAT
RCSE master program

Teacher: Makhmutova Alisa Zufarovna
AZMakhmutova@kai.ru
@DarkAliceSophie

Probability theory

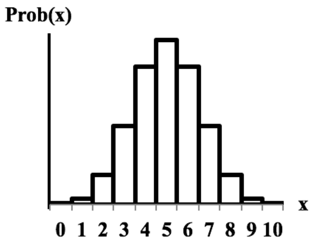
A **random variable** is a variable whose value is a numerical outcome of a random phenomenon.

- A random variable is denoted with a capital letter
- The probability distribution of a random variable *X* tells what the possible values of *X* are and how probabilities are assigned to those values
- A random variable can be discrete or continuous

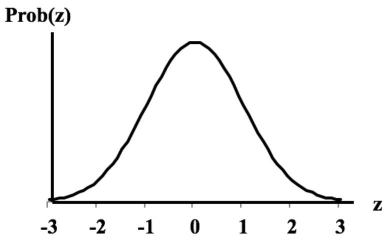
A **discrete random variable** *X* has a countable number of possible values. Examples: Throwing dice, experiments with decks of cards, random walk and tossing coins.

A **continuous random variable** *X* takes all values in a given interval of numbers.

- The probability distribution of a continuous random variable is shown by a **density curve**.
  - The probability that X is between an interval of numbers is the area under the density curve between the interval endpoints
  - The probability that a **continuous random variable** *X* is exactly equal to a number is zero. For example, if we let X denote the height (in meters) of a randomly selected maple tree, then X is a continuous random variable.



Binomial Distribution
Discrete Data & Discrete Probability Curve



Standard Normal Distribution
Continuous Data and Continuous Probability Curve

Probability theory

- **Expected value (математическое ожидание)** of a random variable, is the long-run average value of repetitions of the experiment it represents. The law of large numbers states that the *arithmetic mean* of the values almost surely converges to the expected value as the number of repetitions approaches infinity. The expected value is also known as the **expectation, mathematical expectation, EV, average, mean value, mean, or first moment: *E[X], M[X] – in Russian, μ***

- **Variance (дисперсия)** is the expectation of the squared deviation of a random variable from its mean. Informally, it measures how far a set of (random) numbers are spread out from their average value: ***Var(X), D[X] – in Russian, σ^2, s^2***

$$Var(X) = E[(X - \mu)^2]$$

- **Standard deviation (среднеквадратическое отклонение)** - is a measure that is used to quantify the amount of variation or dispersion of a set of data values: $\boldsymbol{\sigma}$, $\boldsymbol{s}$

$$\sigma = \sqrt{Var(X)}$$

- **The cumulative distribution function** (CDF) of a real-valued random variable X, or just distribution function of X, evaluated at x, is the probability that X will take a value less than or equal to x.

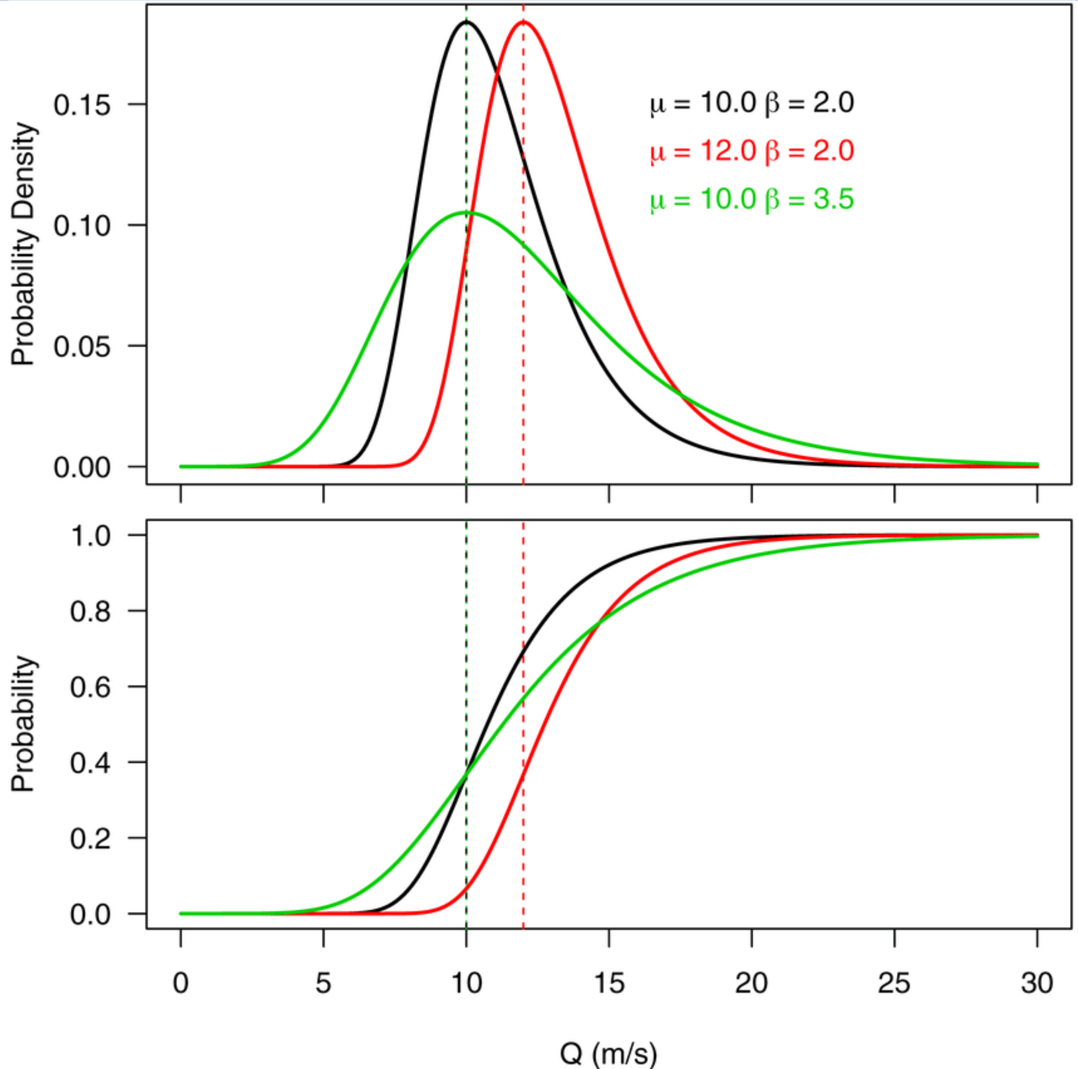
Cumulative distribution function is defined for discrete random variables as:

$$F(x) = P(X \leq x) = \sum_{t \leq x} f(t)$$

The cumulative distribution function ("c.d.f.") of a continuous random variable X is defined as:

$$F(x) = \int_{-\infty}^{x} f(t)dt, \qquad for -\infty < x < \infty$$

Probability Density Functions



- Probability density function (PDF), or density of a continuous random variable, is a function, whose value at any given sample (or point) in the sample space (the set of possible values taken by the random variable) can be interpreted as providing a *relative likelihood* that the value of the random variable would equal that sample.
- PDF shows the probability on some interval (a,b)

$$\int_{-\infty}^{\infty} p(x)dx = F(\infty) - F(-\infty) = 1$$

Probability theory

- **Joint probability** is a statistical measure that calculates the likelihood of two events occurring together and at the same point in time - $p(x, y)$ (example - throwing two dice)

To get probability of one event, we need to sum: $p(x) = \sum_y p(x, y)$ or $p(x) = \int_Y \; p(x, y)\, dy$

- **Conditional probability** - a measure of the probability of an event given that another event has occurred - $p(x \mid y)$; $p(x \mid y) = \frac{p(x,y)}{p(y)}$
- When two events are said to be **independent of each other**, what this means is that the probability that one event occurs in no way affects the probability of the other event occurring - $p(x, y) = p(x)p(y)$
- Two events x and y are **conditionally independent** given a third event z precisely if the occurrence of x and the occurrence of y are independent events in their conditional probability distribution given z –

$$p(x, y \mid z) = p(x \mid z)p(y \mid z)$$

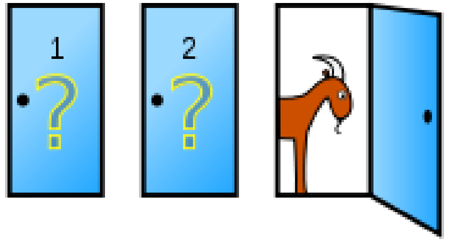
Bayes' theorem

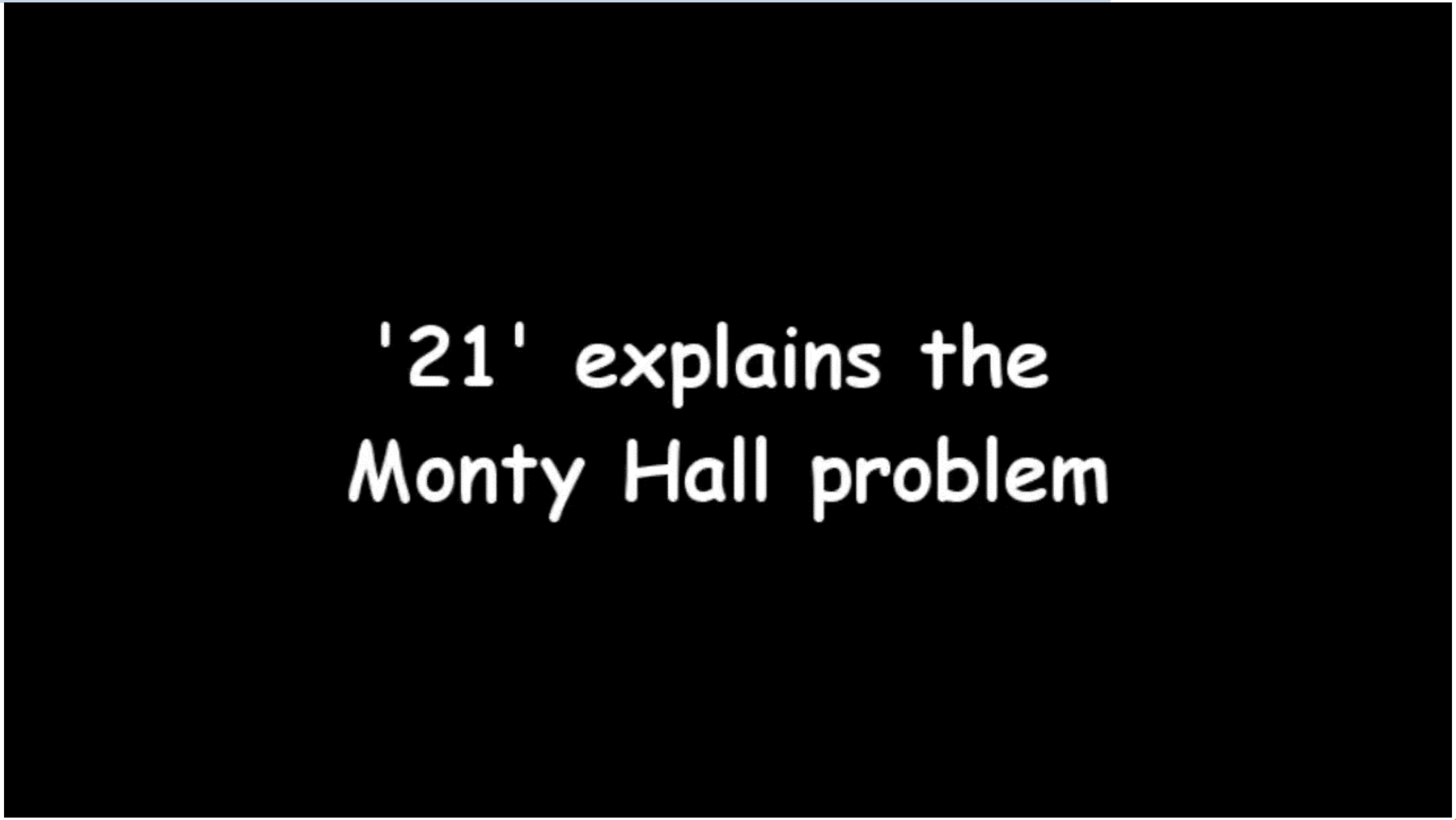
- From definition of conditional probability $p(x\,|y) = \frac{p(x,y)}{p(y)}$ => $p(x,y) = p(x\,|y)p(y) = p(y\,|x)p(x)$

$$p(y\,|x) = \frac{p(x \mid y)p(y)}{p(x)} = \frac{p(x \mid y)p(y)}{\sum_{y' \in Y} p(x \mid y')p(y')}$$

- Bayes formula allows us to overestimate our *apriori representation* about the world (in the formula above this is $p(y)$) based on partial information (data), which we received as observations (in the formula above, this is $p(x \mid y)$) as an output getting a new state of our representations is $p(y\,|x)$
- Probabilistic inference or Bayesian inference
- Inverse and direct probability

Monty Hall problem





Monty Hall problem

- Let's consider event of having car behind each door as x_1, x_2, x_3 respectively. At the beginning, before Ben choose door, the probability was equal: $p(x_1) = p(x_2) = p(\ x_3) = \frac{1}{3}$. If the presenter had just opened one of the door (let it be door 3, x_3), then the other two remain equally probable: $p(x_1 \mid x_3 = 0) = p(x_2 \mid x_3 = 0) = p(\ x_3) = \frac{1}{2}$

- But presenter chooses one of two not chosen by the player and it changes everything. The two events – "which from two doors to choose" and "is there a car behind that door" became dependent. Ben choose first door, and the door choose by presenter let's define as y . If the car really behind first door $(x_1 \ = 1)$, then the presenter selects one of two blank equally likely:

$$p(x_1 = 1, y = 2) = \frac{1}{6} \qquad p(x_2 = 1, y = 2) = 0 \qquad p(x_3 = 1, y = 2) = \frac{1}{3}$$

$$p(x_1 = 1, y = 3) = \frac{1}{6} \qquad p(x_2 = 1, y = 3) = \frac{1}{3} \qquad p(x_3 = 1, y = 3) = 0$$

Monty Hall problem

$$p(x_1 = 1 \mid y = 2) = \frac{p(x_1{=}1,y{=}2)}{p(x_1{=}1,y{=}2)+p(x_1{=}0,y{=}2)} = \frac{p(x_1{=}1,y{=}2)}{p(x_1{=}1,y{=}2)+p(x_2{=}1,y{=}2)+p(x_3{=}1\ y{=}2)} = \frac{1/6}{1/6+1/3} = \frac{1}{3}$$

$$p(x_1 = 0 \mid y = 2) = \frac{p(x_1 = 0, y = 2)}{p(x_1 = 1, y = 2) + p(x_1 = 0, y = 2)} =$$

$$= \frac{p(x_2 = 1, y = 2) + p(x_3 = 1\ y = 2)}{p(x_1 = 1, y = 2) + p(x_2 = 1, y = 2) + p(x_3 = 1\ y = 2)} = \frac{1/3}{1/6 + 1/3} = \frac{2}{3}$$

Bayesian statistics

- Bayes' theorem is one of the most important tool in machine learning task

$$p(\theta \mid D) = \frac{p(\theta)p(D \mid \theta)}{p(D)} = \frac{p(\theta)p(D \mid \theta)}{\int p(D \mid \theta)p(\theta)d\theta}$$

- $p(\theta)$ - prior probability – *априорная вероятность, распределение*
- $p(D \mid \theta)$ - likelihood - *правдоподобие*
- $p(\theta \mid D)$ - posterior probability – *апостериорная вероятность, распределение*
- $p(D) = \int p(D \mid \theta)p(\theta)d\theta$ - evidence – *вероятность данных*

Bayesian statistics

Almost all tasks in machine learning have some models with parameters θ. The goal is according to data *D* to choose the parameters θ describing them in the best way. In classical statistics it is the task of finding *maximum likelihood, ML:*

$$\theta_{ML} = \arg\max_{\theta} p(D \mid \theta),$$

where $\arg\max_{\theta} f(\theta)$ its value of vector θ, at which the maximum of $f(\theta)$ is reached.

In Bayesian approach and modern ML the task is in finding posterior:

$$p(\theta \mid D) \propto p(D \mid \theta) p(\theta)$$

And then, if needed, to find maximum a posteriori hypothesis, MAP

$$\theta_{MAP} = \arg\max_{\theta} p(\theta \mid D) = \arg\max_{\theta} p(D \mid \theta) p(\theta)$$

Bayesian statistics

In answer prediction tasks (for example, we learn to separate cats from dogs in base of photos in order to distinguish a cat from a dog on a new, not previously seen the picture) we are more interested on ***predictive distribution*** $p(y \mid D)$ or $p(y \mid D, x)$, where y – is next example or correct answer on new question x, instead of posterior probability $p(\theta \mid D)$:

$$p(y \mid D) = \int_{\theta} p(y \mid \theta) p(\theta \mid D) d\theta \propto \int_{\theta} p(y \mid \theta) p(\theta) p(D \mid \theta) d\theta$$

But it's quite hard task from even linear models point of view, so usually posterior probability is the goal. So, the task is in optimizing posterior probability $p(\theta \mid D)$:

$\theta_{MAP} = \arg\max_{\theta} p(\theta \mid D) = \arg\max_{\theta} p(D \mid \theta) p(\theta)$ or just a likelihood: $\theta_{ML} = \arg\max_{\theta} p(D \mid \theta)$

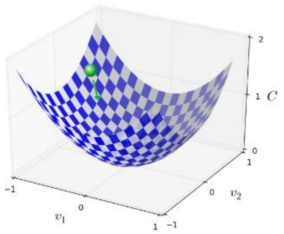
Bayesian statistics

$$\theta_{ML} = \arg\max_{\theta} p(D \mid \theta) \quad \text{->} \; p(D \mid \theta) = \prod_{d \in D} p(d \mid \theta) \; \text{->}$$

$$\theta_{MAP} = \arg\max_{\theta} p(D \mid \theta) p(\theta) = \arg\max_{\theta} p(\theta) \prod_{d \in D} p(d \mid \theta) =$$

$$= \arg\max_{\theta} \left(\log p(\theta) + \sum_{d \in D} \log p(d \mid \theta) \right)$$

Gradient descent

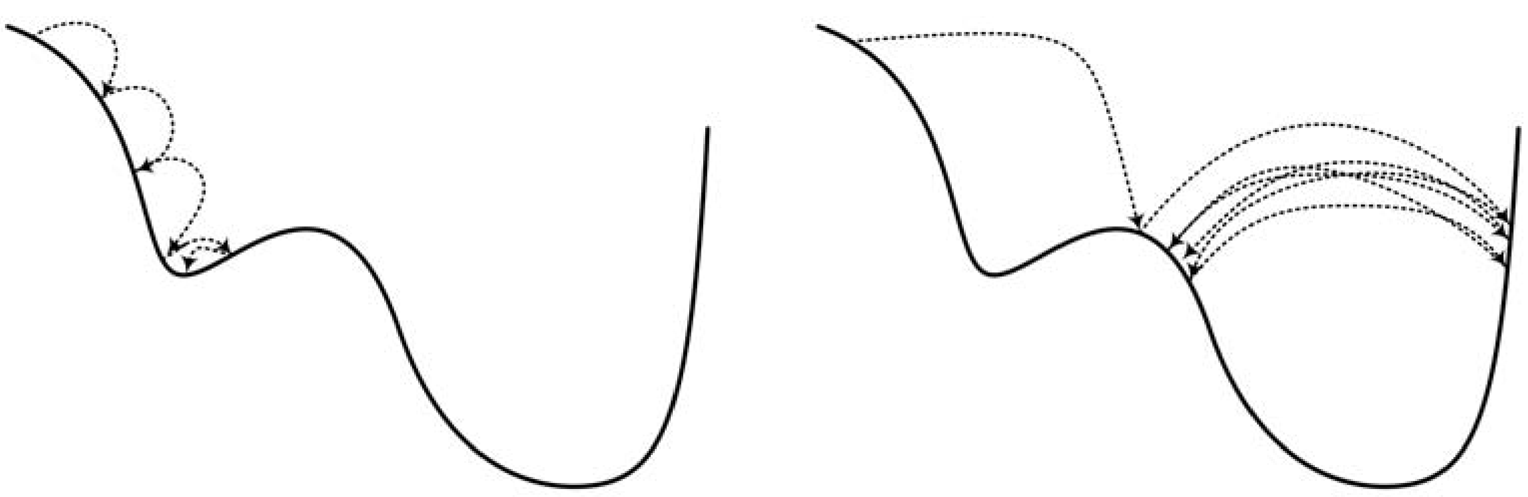


- We have a function $E(\theta) = E(\theta_1, \theta_2, \dots, \theta_n)$, then its *gradient* $\nabla_\theta E = \begin{pmatrix} \frac{\partial E}{\partial \theta_1} \dots \\ \frac{\partial E}{\partial \theta_{n-1}} \\ \frac{\partial E}{\partial \theta_n} \end{pmatrix}$
- The direction of descending is -$\nabla_\theta E$
- By discretizing time and moving forward step by step, we have θ_t on step *t*, so vector of updates will be $u_t = -\eta \nabla_\theta E(\theta_{t-1}), \quad \theta_t = \theta_{t-1} + u_t$

- u = - learning_rate * grad
- theta += u

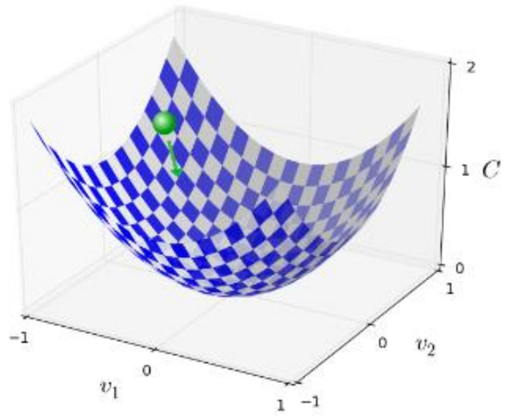
Gradient descent

- $u_t = -\eta \nabla_\theta E(\theta_{t-1}), \quad \theta_t = \theta_{t-1} + u_t$



а

б



Linear regression

linear regression is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The regression model:

$$y = f(x, w) + \varepsilon = w_0 + wx_1 + w_2x_2 + \cdots + w_jx_j + \varepsilon = w_0 + \sum_{j=1}^{p} x_jw_j + \varepsilon = x^Tw + \varepsilon,$$

where x – regressors, w - effects or regression coefficients, ε - error term, disturbance term, or sometimes noise

$$\hat{y} = \widehat{w_0} + \sum_{j=1}^{p} x_j\widehat{w_j} = x^T\widehat{w}$$

$$RSS(w) = \sum_{i=1}^{N} (y_i - x_i^T w)^2;$$

$$RSS(w) = (y - \mathcal{X}w)^T(y - \mathcal{X}w), \mathcal{X} - matrix\ N \times p \text{ -> } w^* = (\mathcal{X}^T\mathcal{X})^{-1}\mathcal{X}^Ty$$

Linear regression

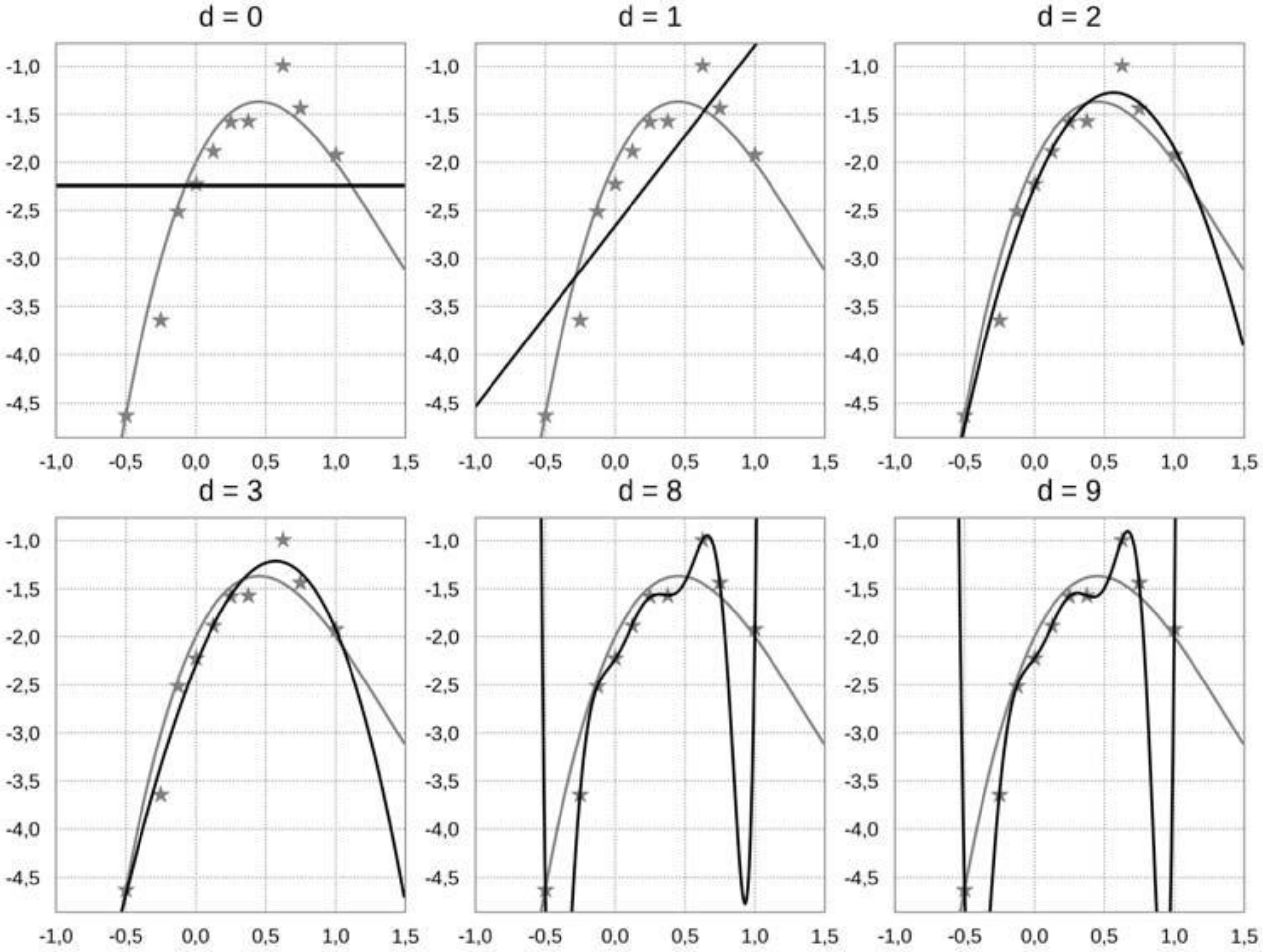
- In linear regression model we had noise ε has normal distribution $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- Our $t = y(x, w) + \varepsilon \; \rightarrow p(t \mid x, w, \sigma^2) = \mathcal{N}(t \mid y(x, w), \sigma^2)$

$$p(t \mid \mathcal{X}, w, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(t_n \mid w^T x_n, \delta^2)$$

$$p(\, x \mid \mu, \delta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

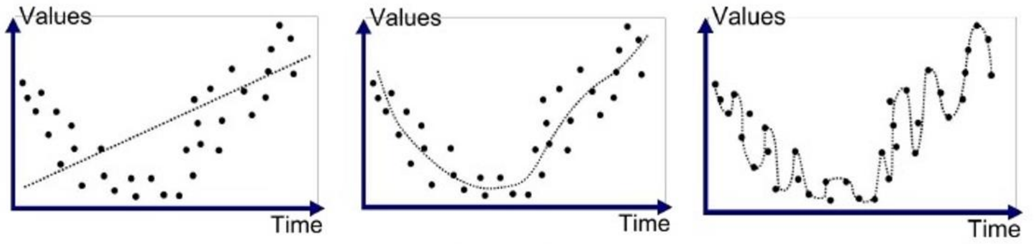
$$\ln p(t \mid w, \sigma^2) = -\frac{N}{2} \ln(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{n=1}^{N} (t_n - w^T x_n)^2$$

Regularization



$$\hat{y} = w_0 + \sum_{j=1}^{d} w_j x^j = \left(1\ x\ x^2 \dots x^d\right)^T w,$$

$$f(x) = x^3 - 4x^2 + 3x - 2$$



Underfitted

Good Fit/Robust

Overfitted

Regularization

$$f(x) = x^3 - 4x^2 + 3x - 2$$

$$f_0(x) = -2{,}2393,$$

$$f_1(x) = -2{,}6617 + 1{,}8775x,$$

$$f_2(x) = -2{,}2528 + 3{,}4604x - 3{,}0603x^2,$$

$$f_3(x) = -2{,}2937 + 3{,}5898x - 2{,}6538x^2 - 0{,}5639x^3,$$

$$f_8(x) = -2{,}2324 + 2{,}2326x + 6{,}2543x^2 + 15{,}5996x^3 - 239{,}9751x^4 + \\ + 322{,}8516x^5 + 621{,}0952x^6 - 1478{,}6505x^7 + 750{,}9032x^8,$$

$$f_9(x) = -2{,}22 + 2{,}01x + 4{,}88x^2 + 31{,}13x^3 - 230{,}31x^4 + \\ + 103{,}72x^5 + 869{,}22x^6 - 966{,}67x^7 - 319{,}31x^8 + 505{,}64x^9.$$

$$RSS(L(w)) = \frac{1}{2}\sum_{i=1}^{N}(f(x_i, w) - y_i)^2$$

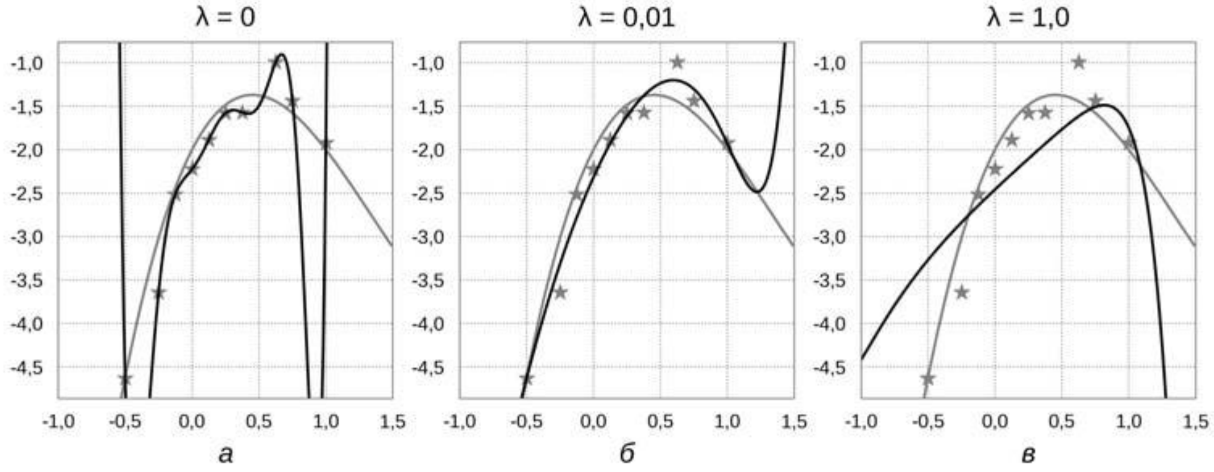
$$RSS(w) = \frac{1}{2}\sum_{i=1}^{N}(f(x_i, w) - y_i)^2 + \frac{\lambda}{2}\|w\|^2,$$

where λ - regularization coefficient

$\frac{\lambda}{2}\|w\|^2$ - ridge regression

Regularization

$$f(x) = x^3 - 4x^2 + 3x - 2$$



$$f_{\lambda=0}(x) = -2{,}22 + 2{,}01x + 4{,}88x^2 + 31{,}13x^3 - 230{,}31x^4 + \\ + 103{,}72x^5 + 869{,}22x^6 - 966{,}67x^7 - 319{,}31x^8 + 505{,}64x^9,$$

$$f_{\lambda=0{,}01}(x) = -2{,}32 + 3{,}40x - 2{,}33x^2 + 0{,}05x^3 - 0{,}51x^4 - \\ - 0{,}29x^5 - 0{,}22x^6 - 0{,}06x^7 + 0{,}09x^8 + 0{,}24x^9,$$

$$f_{\lambda=1}(x) = -2{,}46 + 1{,}45x - 0{,}19x^2 + 0{,}22x^3 - 0{,}13x^4 - \\ - 0{,}05x^5 - 0{,}14x^6 - 0{,}13x^7 - 0{,}16x^8 - 0{,}16x^9.$$

Regularization

$$p(w) = \mathcal{N}(\mathrm{w} \mid \mu_0, \Sigma_0) \text{ on } \mathcal{X} = \{x_1, \dots, x_N\}\ with\ t = \{t_1, \dots, t_N\}$$

$$p(t \mid \mathcal{X}, w, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(t_n \mid w^T x_n, \sigma^2)$$

$$p(w \mid t) \propto p(t \mid \mathcal{X}, w, \sigma^2) p(w) = \mathcal{N}(\mathrm{w} \mid \mu_0, \Sigma_0) \prod_{n=1}^{N} \mathcal{N}(t_n \mid w^T x_n, \sigma^2)$$

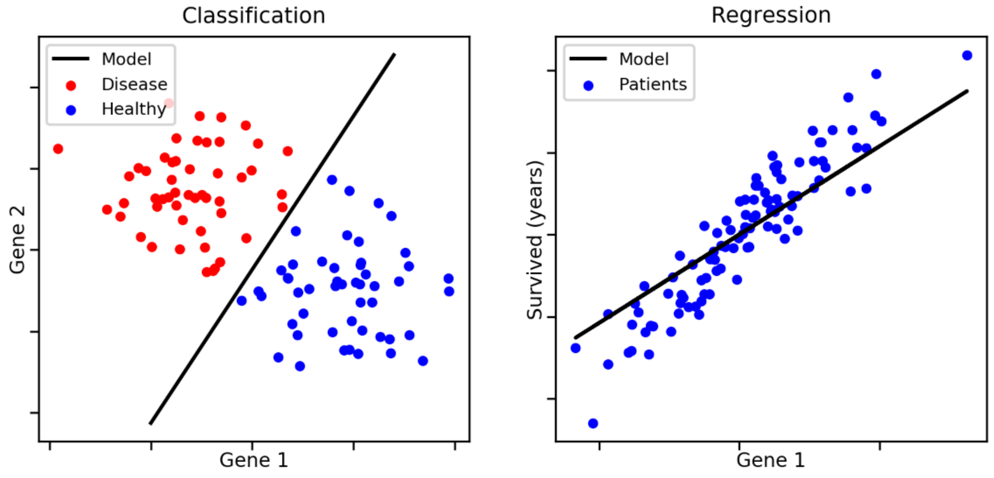
$$p(w \mid t) = \mathcal{N}(\mathrm{w} \mid \mu_N, \Sigma_N)$$

$$\mu_N = \Sigma_N \left(\sum\nolimits_0^{-1} \mu_0 + \frac{1}{\sigma^2} X^T t\right), \qquad \Sigma_N = \left(\sum\nolimits_0^{-1} \mu_0 + \frac{1}{\sigma^2} X^T X\right)^{-1}, \qquad p(w) = \mathcal{N}\left(\mathrm{w} \mid \mathbb{0}, \frac{1}{\alpha} \mathbb{I}\right)$$

$$\ln p(w \mid t) = -\frac{1}{2\sigma^2} \sum_{n=1}^{N} (t_n - w^T x_n)^2 - \frac{\alpha}{2} w^T w + const$$

Classification vs Regression

Property	Classification	Regression
Output type	Discrete (class labels)	Continuous (number)
What are trying to find?	Decision boundary	“Best fit line”
Evaluation	Accuracy	“Sum of squared error” or r^2



Kullback — Leibler divergence, KL divergence, relative entropy

Kullback-Leibler divergence (also called relative entropy) is a measure of how one probability distribution (P) is different from a second (Q), reference probability distribution.

$$KL(P \parallel Q) = \int \log \frac{dP}{dQ} dP$$

1) For discrete random variable on discrete set $X = \{x_1, \dots, x_N\}$ $KL(P \parallel Q) = \sum_i p(x_i) \log \frac{p(x_i)}{q(x_i)}$

2) For continuous random variable P and Q in space $\mathbb{R}^d$ $KL(P \parallel Q) = \int_{\mathbb{R}^d} p(x) \log \frac{p(x)}{q(x)} dx$

3) $KL(P \parallel Q) \neq KL(Q \parallel P),\ KL(P \parallel Q) \geq 0$

Kullback — Leibler divergence, KL divergence, relative entropy

- How to use Kullback-Leibler divergence for classification tasks?

We will try to calculate how distribution in the test examples generated by the classifier (let's call it *q*), it similar or dissimilarly with the “true” distribution defined by the data (let's call it *p*).

Let’s look at the binary classification with input data *(x,y)*, where *y* takes only two values – *0 and 1*. $p(y=1)=y, \quad p(y=0)=1-y$.

The classifier is trying to assess the probability of a positive response of $p(y \mid D, x)$ and we will call that probability *q(y)*

Cross-entropy

- We will use cross-entropy for minimization $H(p,q) = \mathbb{E}_p[-\log q] = -\sum_y p(y)\log q(y)$ which is connected with Kullback — Leibler divergence:

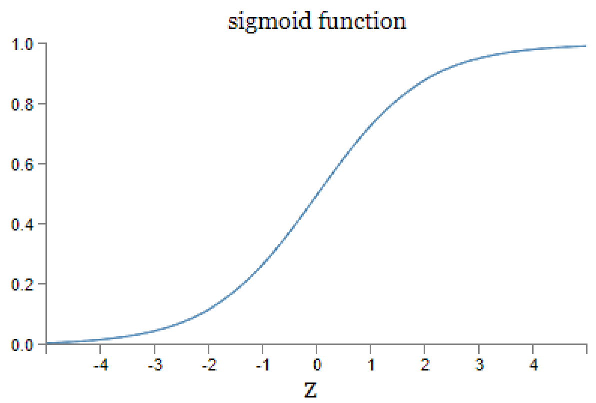
$$KL(P \parallel Q) = \sum_y p(y)\log\frac{p(y)}{q(y)} = \sum_y p(y)\log p(y) - \sum_y p(y)\log q(y) = H(p) + H(p,q)$$

So, binary classification, the objective function, which we will minimize on data set $D = \{(x_i, y_i)\}_{i=1}^{N}$:

$$L(\theta) = H\left(p_{data,}q(\theta)\right) = -\frac{1}{N}\sum_{i=1}^{N}(y_i\log\widehat{y}_i(\theta) + (1-y_i)\log(1-\widehat{y}_i(\theta))),$$

where $\widehat{y}_i(\theta)$ - estimation of the probability of response 1, obtained by the classifier

Logistic regression (logit model)



Let's consider the classification problem from a probabilistic point of view. We will associate with each class C_k density $p(x \mid C_k)$, unknown for us, will find prior probability $p(C_k)$ and then $p(C_k \mid x)$. For 2 classes:

$$p(C_1 \mid x) = \frac{p(x \mid C_1)p(C_1)}{p(x \mid C_1)p(C_1)+p(x \mid C_2)p(C_2)} = \frac{1}{1+e^{-a}} = \sigma(a), \ \ where\ a = \ln\frac{p(x \mid C_1)p(C_1)}{p(x \mid C_2)p(C_2)} \text{ and } \sigma = \frac{1}{1+e^{-a}}$$

$$a = w^T x \ \rightarrow \ p(C_1 \mid x) = y(x) = \sigma(w^T x), \qquad p(C_2 \mid x) = 1 - p(C_1 \mid x)$$

For learning we can directly optimize the likelihood on *w*. For input data $\{x_n, t_n\}$, where x_n - *inputs*, t_n - *corresponding correct answers*, $t_n \in \{0,1\}$ likelihood will be:

$$p(t \mid w) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{1-t_n} \text{ , where } \ y_n = p(C_1 \mid x_n)$$

Logistic regression (logit model)

$$E(w) = -\ln p(t \mid w) = -\sum_{n=1}^{N} [t_n \ln y_n + (1 - t_n) \ln(1 - y_n)]$$

For K classes:

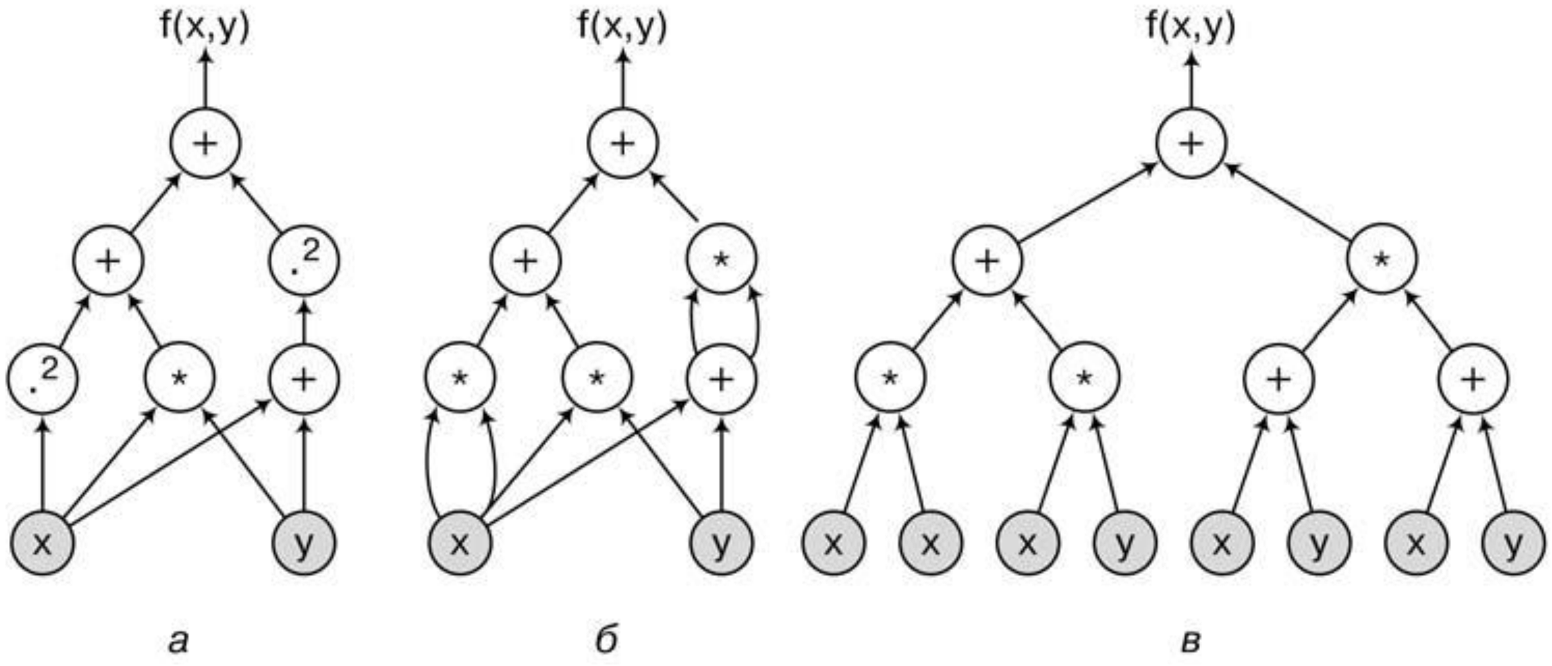
$$p(C_k \mid x) = \frac{p(x \mid C_k) p(C_k)}{\sum_{j=1}^{k} p(x \mid C_j) p(C_j)} = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}}, \quad a_k = \ln p(x \mid C_k) p(C_k)$$

$$p(T \mid w_1, \dots, w_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(C_k \mid x_n)^{t_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}}, \qquad y_{nk} = y_k(x_n)$$

$$E(w_1, \dots, w_K) = -\ln p(T \mid w_1, \dots, w_K) = -\prod_{n=1}^{N} \prod_{k=1}^{K} t_{nk} \ln y_{nk}$$

Derivatives of computational graph of function

A computation graph is a graph whose nodes are functions (usually fairly simple, taken from a predetermined set), and the edges associate functions with their arguments



$$f(x, y) = x^2 + xy + (x + y)^2$$

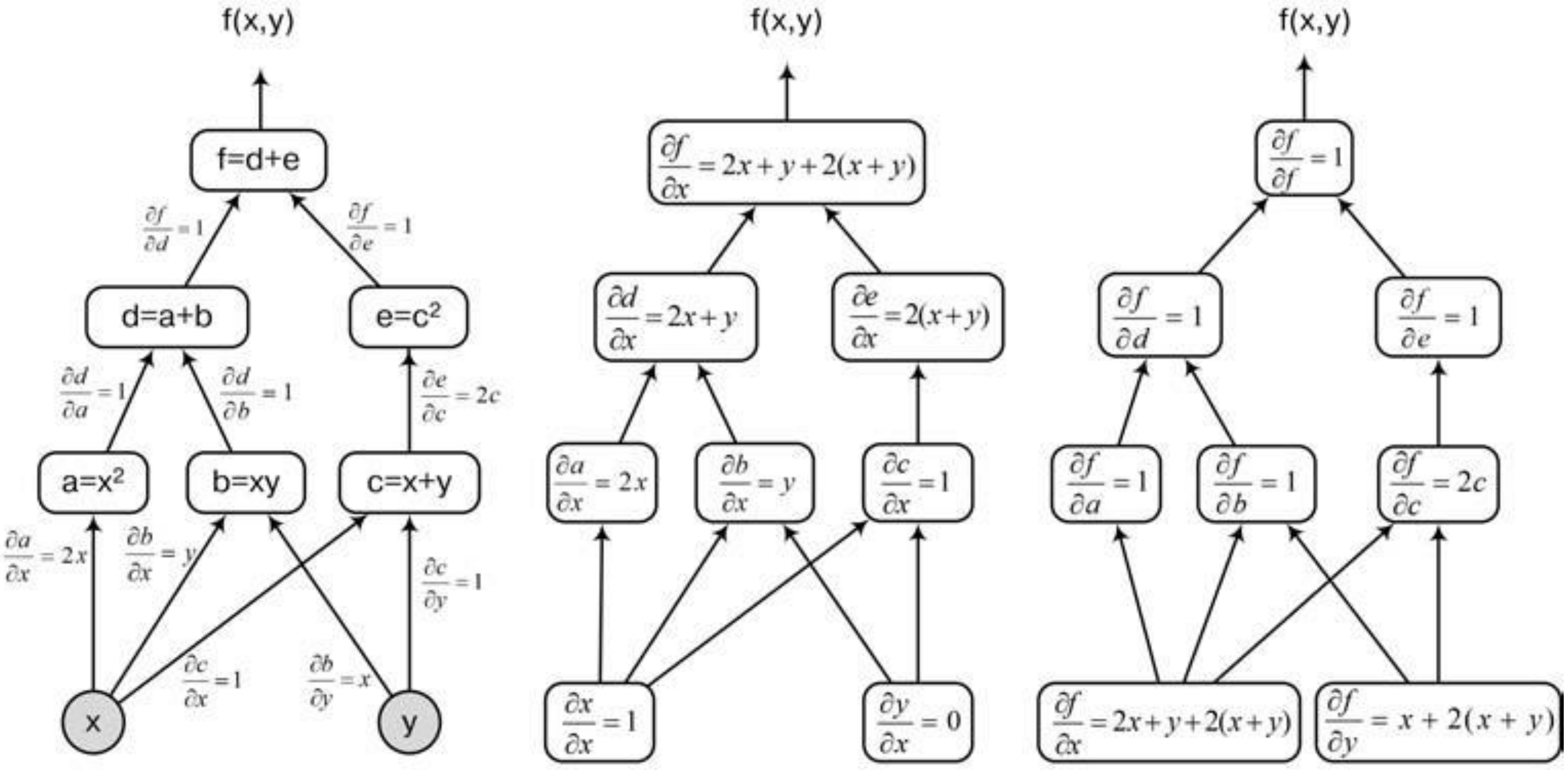
Derivatives of computational graph of function

$$\left(f\left(g(x)\right)\right)' = f'\left(g(x)\right)g'(x); \qquad \nabla_x f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

$$\nabla_x (f \circ g) = \begin{pmatrix} \frac{\partial f \circ g}{\partial x_1} \\ \vdots \\ \frac{\partial f \circ g}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial g}\frac{\partial g}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial g}\frac{\partial g}{\partial x_n} \end{pmatrix} = \frac{\partial f}{\partial g}\nabla_x g; \quad \nabla_x f = \frac{\partial f}{\partial g_1}\nabla_x g_1 + \cdots + \frac{\partial f}{\partial g_k}\nabla_x g_k = \sum_{i=1}^{k}\frac{\partial f}{\partial g_i}\nabla_x g_i$$

$$\nabla_x f = \nabla_x g \nabla_g f, where \nabla_x g = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_k}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_1}{\partial x_n} & \cdots & \frac{\partial g_k}{\partial x_n} \end{pmatrix}$$ - Jacobian matrix

Derivatives of computational graph of function



а б в

$$f(x, y) = x^2 + xy + (x + y)^2$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial x} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial x} =$$

$$= 2x + y + 2(x + y)$$

Linear regression training

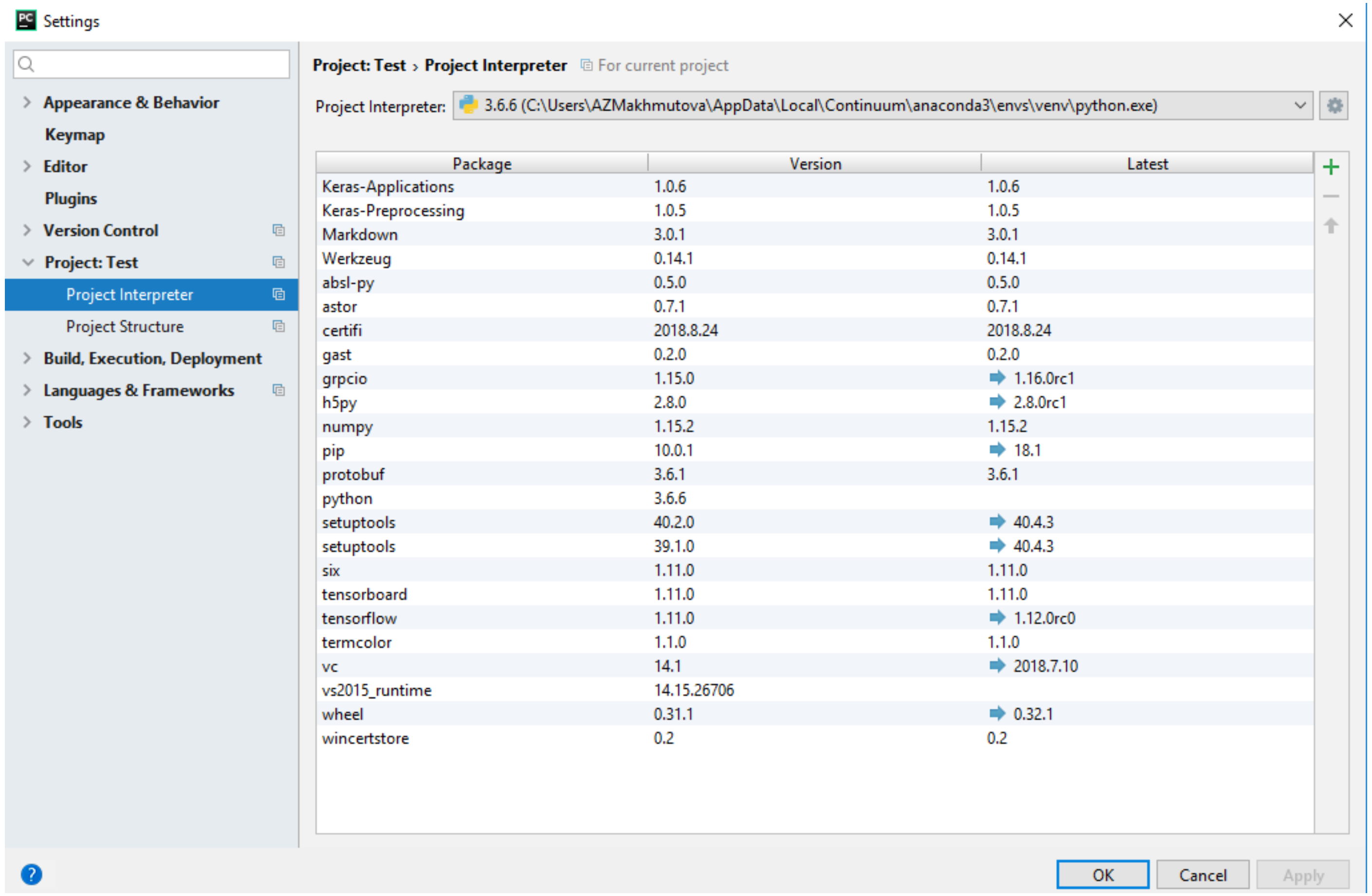
```python
import numpy as np,tensorflow as tf

n_samples, batch_size, num_steps = 1000, 100, 20000
X_data = np.random.uniform(1, 10, (n_samples, 1))
y_data = 2 * X_data + 1 + np.random.normal(0, 2, (n_samples, 1))
X = tf.placeholder(tf.float32, shape=(batch_size, 1))
y = tf.placeholder(tf.float32, shape=(batch_size, 1))

with tf.variable_scope('linear-regression'):
    k = tf.Variable(tf.random_normal((1, 1), stddev=0.0), name='slope')
    b = tf.Variable(tf.zeros((1,)), name='bias')

y_pred = tf.matmul(X, k) + b
loss = tf.reduce_mean((y - y_pred) ** 2)
optimizer = tf.train.GradientDescentOptimizer(0.001).minimize(loss)

display_step = 50

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(num_steps):
        indices = np.random.choice(n_samples, batch_size)
        X_batch, y_batch = X_data[indices], y_data[indices]
        _, loss_val, k_val, b_val = sess.run([ optimizer, loss, k, b ],
        feed_dict = { X : X_batch, y : y_batch })
    if (i+1) % display_step == 0:
        print('Epoch %d: %.8f, k=%.4f, b=%.4f' % (i+1, loss_val, k_val, b_val))
```

$$L = \sum_{i=1}^{N} (\hat{y} - y)^2 \to \min$$

f = kx + b, k=2, b=1

1) Create 1000 random points in [0:1]
2) Calculate for each point x the correct answer $y = 2x+1+\epsilon$, ϵ $N(\epsilon; 0; 2$- variance)
3) Initialization of k and b
4) Calculating the initial loss function
5) Calculating the optimizer, learning rate = 0.001
6) Optimization cycle

Keras


Settings
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Project: Test
Project Interpreter
Project Structure
Build, Execution, Deployment
Languages & Frameworks
Tools
Project: Test › Project Interpreter
For current project
Project Interpreter: 3.6.6 (C:\Users\AZMakhmutova\AppData\Local\Continuum\anaconda3\envs\venv\python.exe)
Package Version Latest
Keras-Applications 1.0.6 1.0.6
Keras-Preprocessing 1.0.5 1.0.5
Markdown 3.0.1 3.0.1
Werkzeug 0.14.1 0.14.1
absl-py 0.5.0 0.5.0
astor 0.7.1 0.7.1
certifi 2018.8.24 2018.8.24
gast 0.2.0 0.2.0
grpcio 1.15.0 1.16.0rc1
h5py 2.8.0 2.8.0rc1
numpy 1.15.2 1.15.2
pip 10.0.1 18.1
protobuf 3.6.1 3.6.1
python 3.6.6
setuptools 40.2.0 40.4.3
setuptools 39.1.0 40.4.3
six 1.11.0 1.11.0
tensorboard 1.11.0 1.11.0
tensorflow 1.11.0 1.12.0rc0
termcolor 1.1.0 1.1.0
vc 14.1 2018.7.10
vs2015_runtime 14.15.26706
wheel 0.31.1 0.32.1
wincertstore 0.2 0.2
OK
Cancel
Apply

Keras

```
import numpy as np
import tensorflow as tf
from tensorflow import keras

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Activation

logr = Sequential()
logr.add(Dense(1, input_dim=2, activation='sigmoid'))
logr.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

def sampler(n, x, y):
    return np.random.normal(size=[n, 2]) + [x, y]

def sample_data(n=1000, p0=(-1., -1.), p1=(1., 1.)):
    zeros, ones = np.zeros((n, 1)), np.ones((n, 1))
    labels = np.vstack([zeros, ones])
    z_sample = sampler(n, x=p0[0], y=p0[1])
    o_sample = sampler(n, x=p1[0], y=p1[1])
    return np.vstack([z_sample, o_sample]), labels


X_train, Y_train = sample_data()
X_test, Y_test = sample_data(100)

logr.fit(X_train, Y_train, batch_size=16, epochs=100, verbose=1, validation_data=(X_test, Y_test))
```

Handwriting digits recognition

```
import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

x = tf.placeholder(tf.float32, [None, 784])

W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

y = tf.nn.softmax(tf.matmul(x, W) + b)

y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Accuracy: %s" % sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

Handwriting digits recognition

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
pred = np.array([[31, 23,  4, 24, 27, 34],
                 [18,  3, 25,  0,  6, 35],
                 [28, 14, 33, 22, 20,  8],
                 [13, 30, 21, 19,  7,  9],
                 [16,  1, 26, 32,  2, 29],
                 [17, 12,  5, 11, 10, 15]])

y = np.array([[31, 23,  4, 24, 27, 34],
              [18,  3, 25,  0,  6, 35],
              [28, 14, 33, 22, 20,  8],
              [13, 30, 21, 19,  7,  9],
              [16,  1, 26, 32,  2, 29],
              [17, 12,  5, 11, 10, 15]])
```

```
tf.argmax(pred, 1) -> array([5, 5, 2, 1, 3, 0])
tf.argmax(y, 1) ->    array([5, 5, 2, 1, 3, 0])


tf.equal(tf.argmax(pred, 1),tf.argmax(y, 1))  ->
                          array(1,1,1,1,1,1)
```

- Add one ReLu layer
- Add dropout - this is the layer that throws out (resets) the outputs of some neurons, selected randomly and anew for each training example.
- Change model

```
keep_probability = tf.placeholder(tf.float32)
h_drop = tf.nn.dropout(h, keep_probability)

for i in range(2000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys, keep_probability: 0.5})
```

Practice

- https://playground.tensorflow.org/