Neural Networks

Semester 1

Practical works for GRIAT
RCSE master program

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

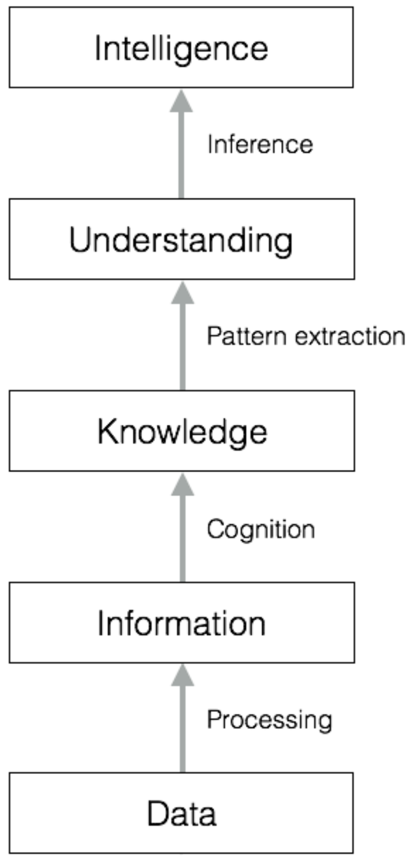
Requirements

- Basic Python (2, 3) knowledge: syntax, operations, packages etc. (SoloLearn application, courses on edX, tutorials)
- Python IDE (https://www.jetbrains.com/pycharm/)
- Basic matrix calculation

Motivation for Artificial Intelligence (AI)

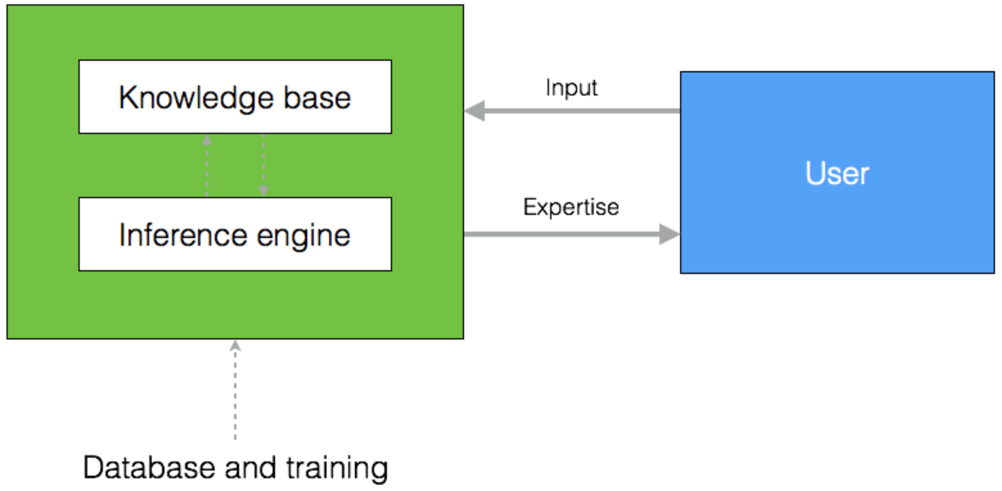
- Big amount of data for processing
- Multiple simultaneous sources of data
- Unorganized, mixed data
- Evolving of data (knowledge update)
- Fast, “human alike” response in real-time

AI techniques can partially cover described problems.



Which tasks Artificial Intelligence (AI) can solve?

- Computer Vision (CV)
- Natural Language Processing (NLP)
- Speech Recognition
- Expert Systems
- Robotics



AI itself include several branches: search and optimization (e.g. mathematical optimization and evolutionary computation), classifiers and statistical learning methods (machine learning), artificial neural networks

Types of learning

1) Knowledge acquisition from expert

2) Knowledge acquisition from data:

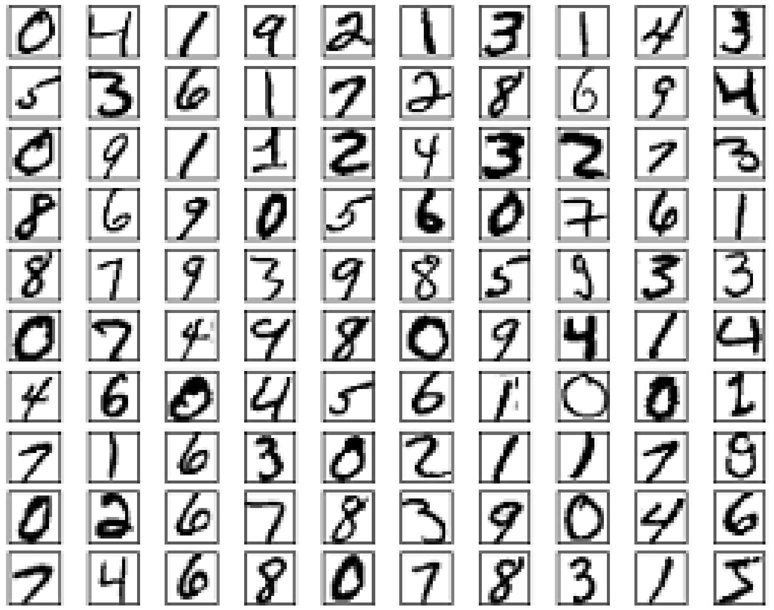
- *Supervised learning* – the system is supplied with a set of training examples consisting of inputs and corresponding outputs, and is required to discover the relation or mapping between them.
- *Unsupervised learning* – the system is supplied with a set of training examples consisting only of inputs. It is required to discover what appropriate outputs should be.

Artificial Neural Networks (ANN) for handwritten digits recognition

Everyday we face with task of recognition of handwritten symbols

Idea – create neuron network, that will recognize handwritten digits

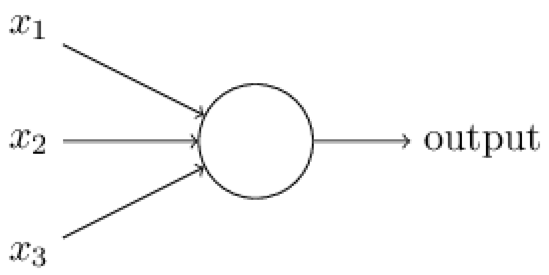
Training examples – large number of handwritten digits



PERCEPTRONS

Perceptrons were developed in the **1950s** by the scientist **Frank Rosenblatt.**

Binary inputs



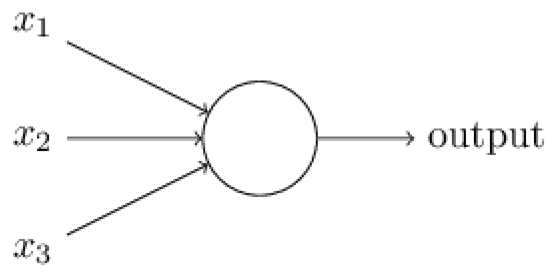
Idea of Rosenblatt was in weights, $w_1, w_2, \ldots$, real numbers expressing the importance of the respective inputs to the output

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

PERCEPTRONS – example

You’re thinking about going to the cinema. You need to make a decision. You might make your decision by weighing up three factors:

1) Is there a convenient time for the exact movie?
2) Is there a good price?
3) Do you have a company to go?



Ley’s represent these three factors by corresponding binary variables $x_{1,}x_2$, and x_3. For instance, we'd have x_1=1 if there is a good time, and x_1=0 if there is no time. For x_2 and x_3 we will do the same.

You want to go to see this movie so badly, so you don’t care about company or price so much. You can use perceptrons to model this kind of decision-making. One way to do this is to choose a weight w_1= 6 for the time, and w_2=2 and w_3=2 for the other conditions.

Complex network of perceptron

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

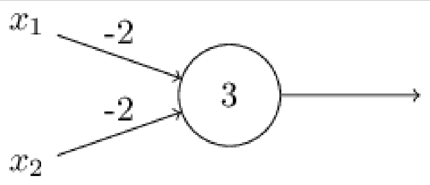
$w \cdot x \equiv \sum_j w_j x_j$, where *w* and *x* are vectors whose components are the weights and inputs

$\sum_j w_j x_j > threshold$

$bias,\ b \equiv -threshold$

$$output = \begin{Bmatrix} 0 & if\ w^T \cdot x + b \ \leq 0 \\ 1 & if\ w^T \cdot x + b \ > 0 \end{Bmatrix}$$

Small example



Perceptrons can be used is to compute the elementary logical functions we usually think of as underlying computation, functions such as AND, OR, and NAND. For example, suppose we have a perceptron with two inputs, each with weight −2, and an overall bias of 3.

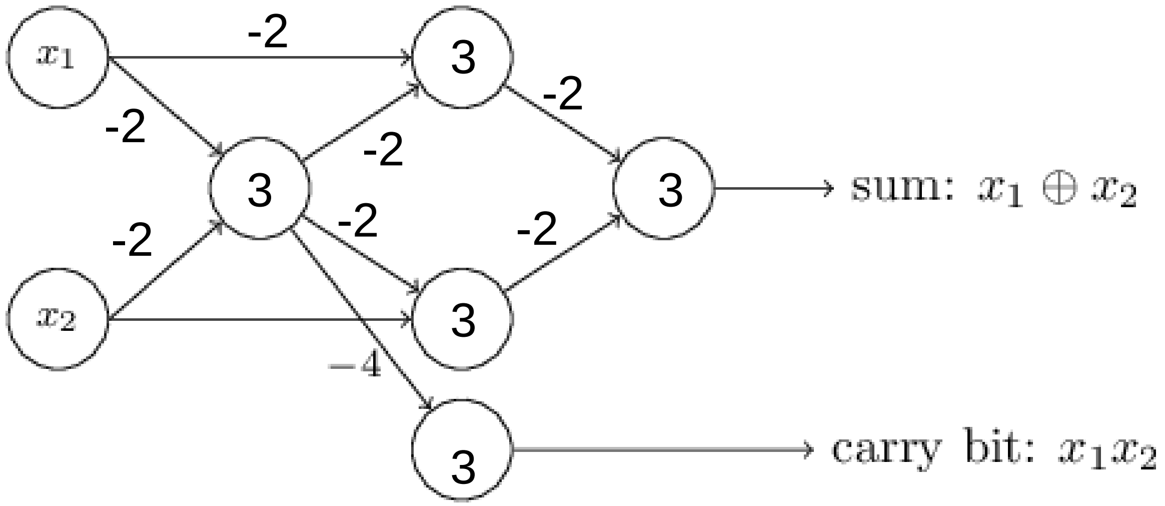
$$W = \begin{pmatrix} -2 \\ -2 \end{pmatrix}$$

Input 00 produces output 1, since (−2)∗0+(−2)∗0+3=3 is positive. Similar calculations show that the inputs 01 and 10 produce output 1. But the input 11 produces output 0, since (−2)∗1+(−2)∗1+3=−1 is negative.

$$W = \begin{pmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ w_{21} & w_{22} & \dots & w_{2m} \\ \dots & \dots & \dots & \dots \\ w_{n1} & w_{n2} & \dots & w_{nn} \end{pmatrix}$$

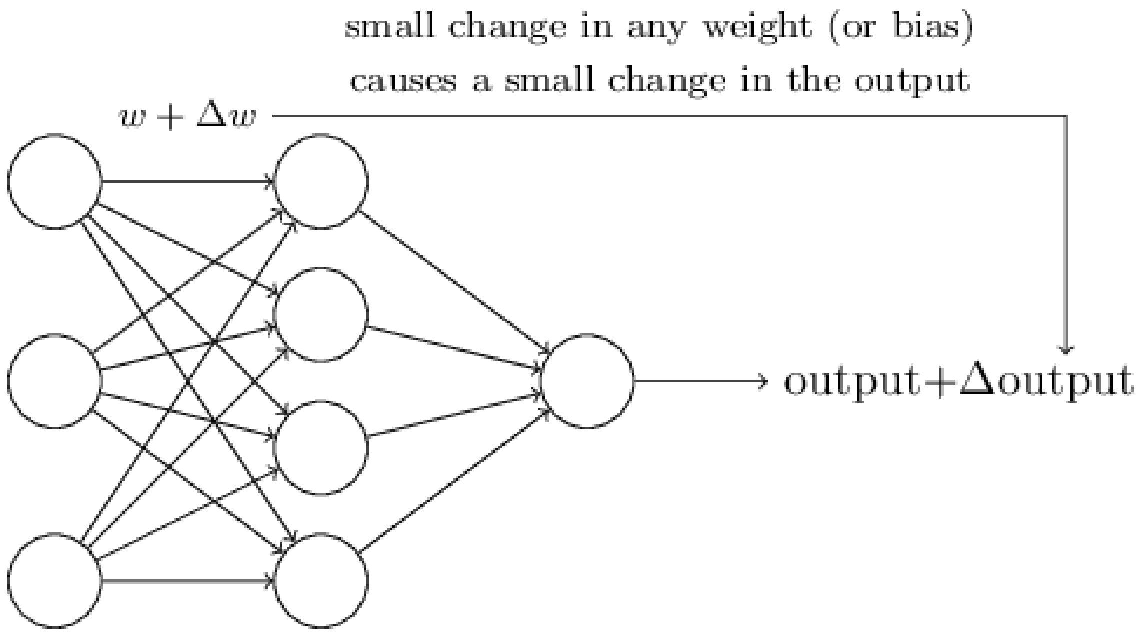
Practice

We can use networks of perceptrons to compute any logical function at all. For example, we can use NAND gates to build a circuit which adds two bits, x_1 and x_2. This requires computing the bitwise sum, $x_1 \oplus x_2$. Let's calculate outputs of the network.

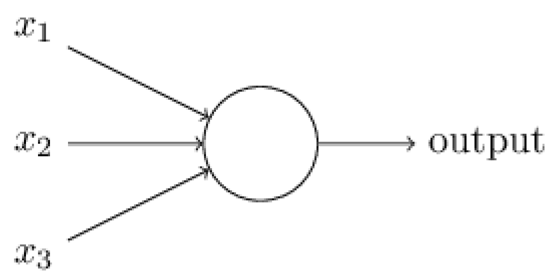


a	b	$a \oplus b$
0	0	0
1	0	1
0	1	1
1	1	0

SIGMOID NEURONS


small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

SIGMOID NEURONS



Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output.

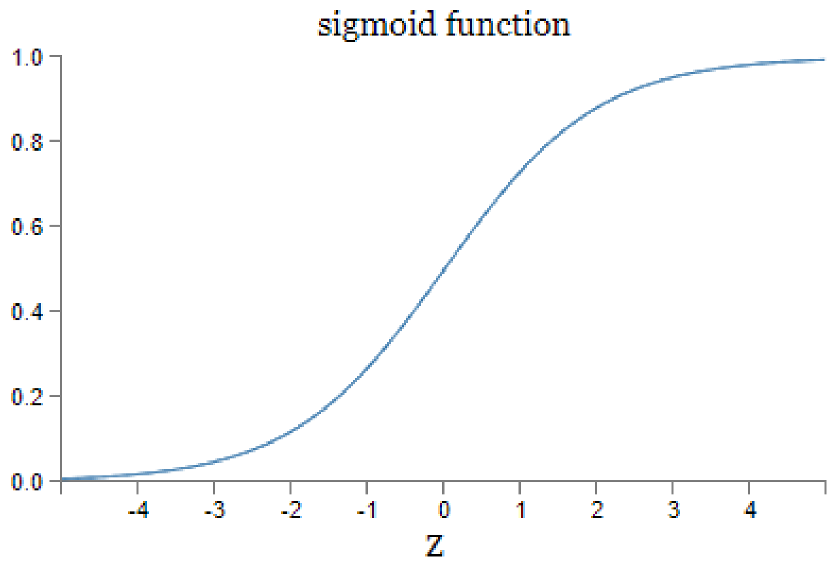
1) Just like a perceptron, the sigmoid neuron has inputs, $x_1, x_2, \ldots$. But instead of being just 0 or 1, these inputs can also take on any values between 0 and 1. So, for instance, 0.638 is a valid input for a sigmoid neuron.

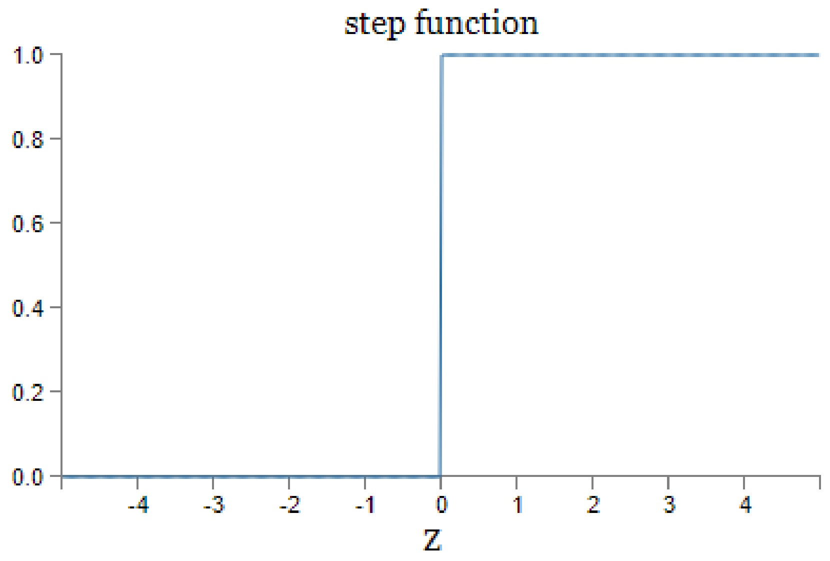
2) The sigmoid neuron has weights for each input, $w_1, w_2, \ldots$, and an overall *bias*, b. But the output is not 0 or 1. Instead, it's $\boldsymbol{\sigma(z)}$, *where* $z \equiv w \cdot x + b$ *and* $\boldsymbol{\sigma}$ is called the ***sigmoid function****

$$\sigma(z) \equiv \frac{1}{1+e^{-z}} \quad \Longrightarrow \quad \sigma(z) \equiv \frac{1}{1+\exp\left(-\sum_j w_j x_j - b\right)}$$

*sometimes called the logistic function, and this class of neurons called logistic neurons

SIGMOID NEURONS

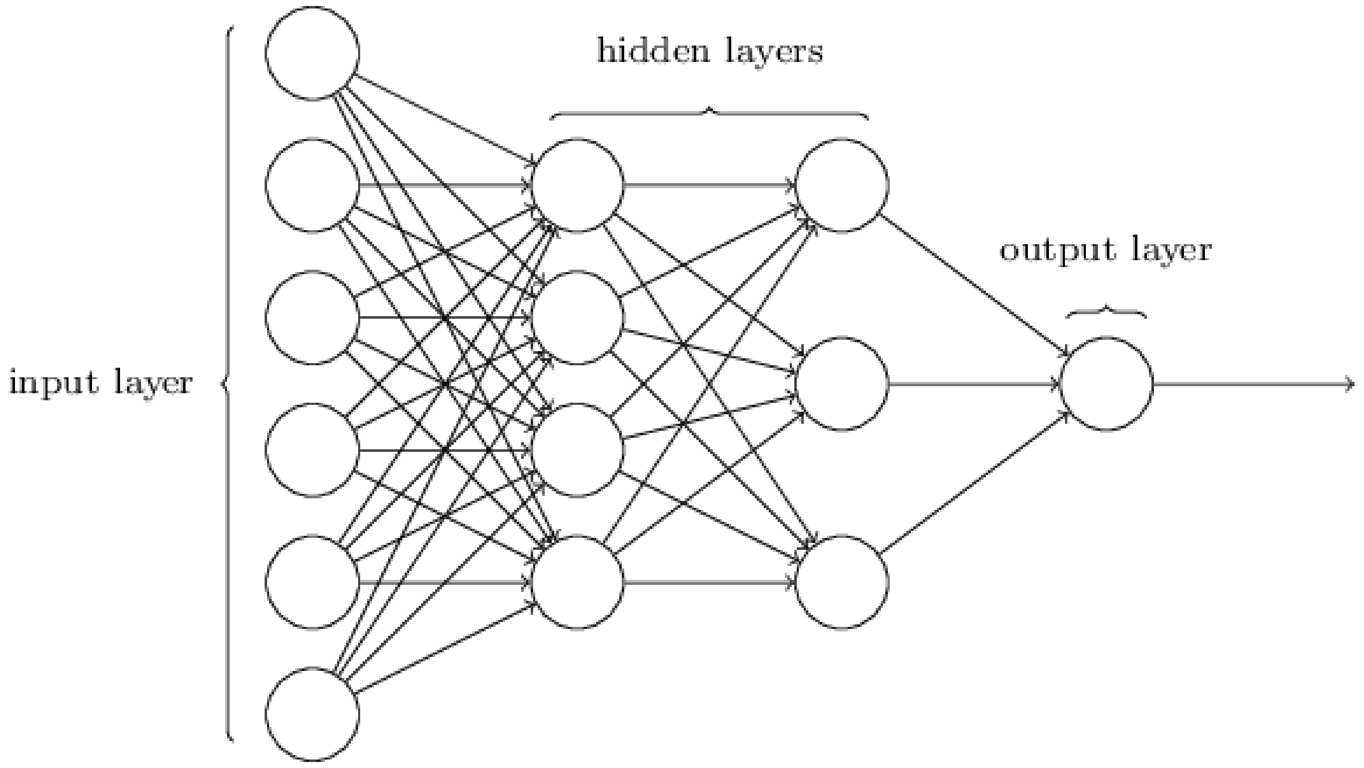




$$\Delta output \approx \sum_j \frac{\partial\, output}{\partial\, w_j}\Delta w_j + \frac{\partial\, output}{\partial\, b}\Delta b$$

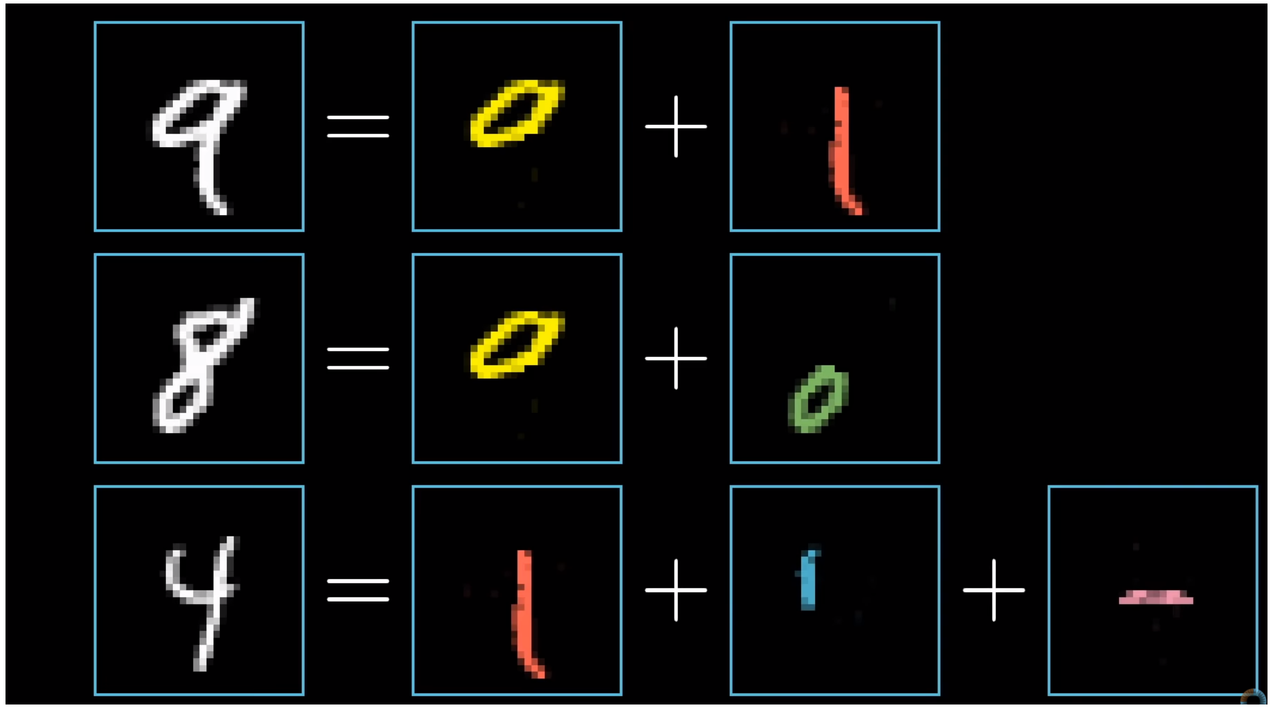
$\Delta output$ is a **linear function** of the changes Δw_j and Δb in the weights and bias

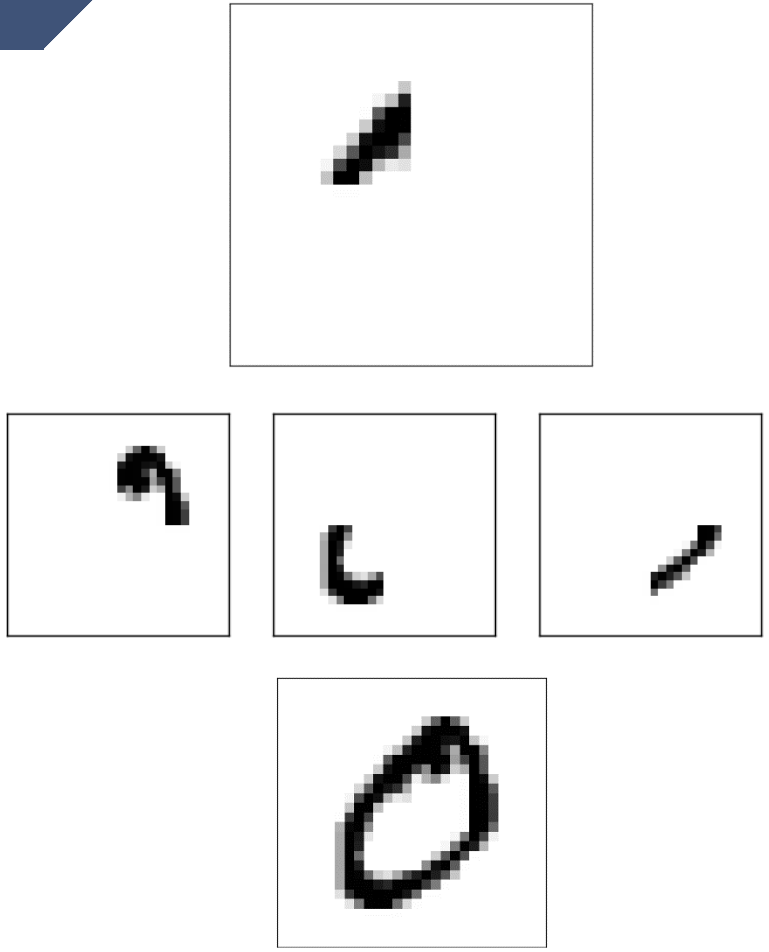
The architecture of neural networks



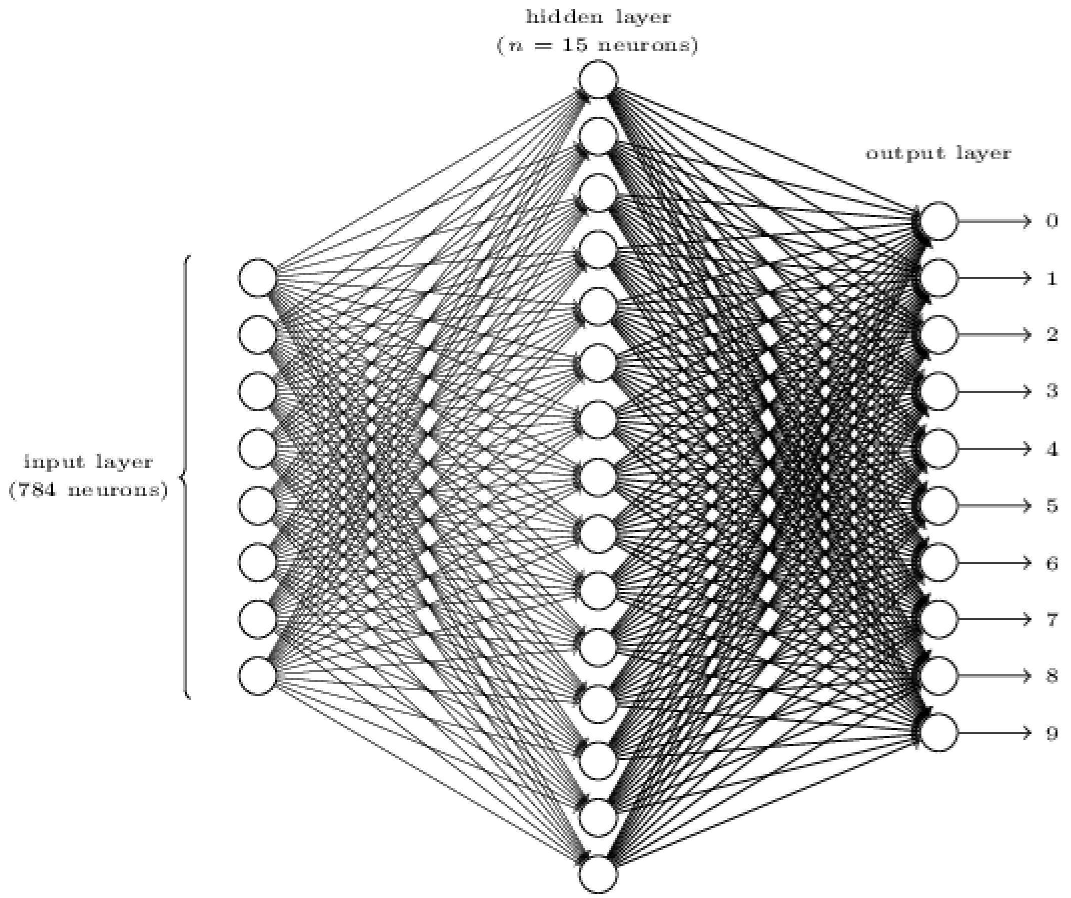
multilayer perceptrons or MLPs

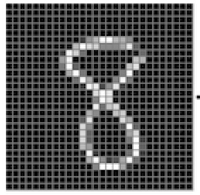
Handwritten digits recognition task

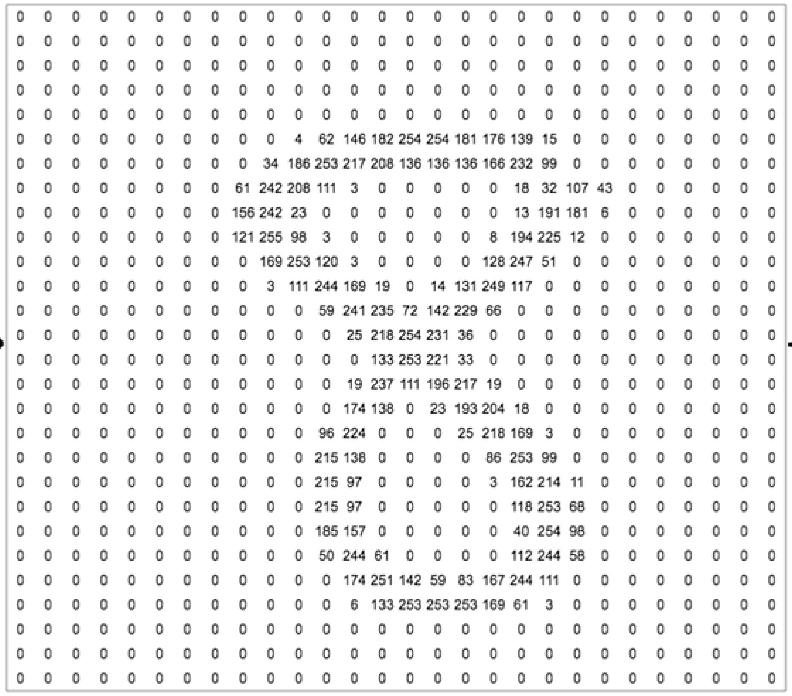




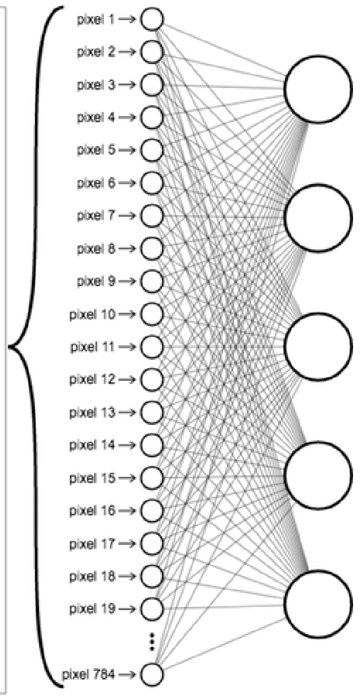
Handwritten digits recognition task




28 x 28
784 pixels

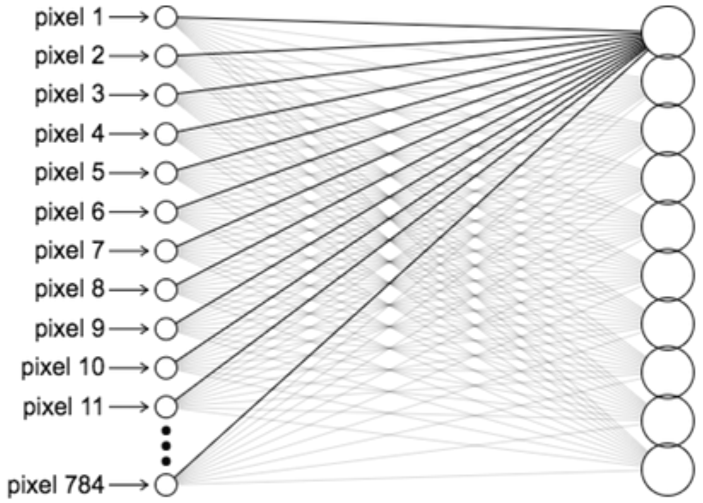


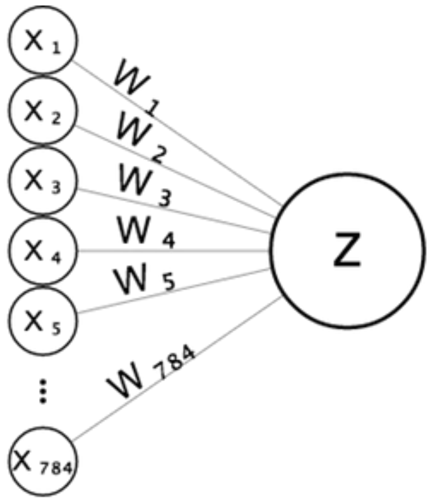
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Handwritten digits recognition task

$$\sigma(w_1a_1+w_2a_2+w_3a_3+w_4a_4+\ldots+w_na_n + bias)$$





Only activate meaningfully
when weighted sum > **threshold**

Learning with gradient descent

x is training input - 28×28=784-dimensional vector, where each entry in the vector represents the grey value for a single pixel in the image

y = *y(x)* is a desired output - 10-dimensional vector (*y(x)*=$(0,0,0,0,0,0,1,0,0,0)^T$ for digit 6)

For determination of how good we are in approximating *y(x)* we will define cost function*:

$$C(w,b) = \frac{1}{2n}\sum_x \|y(x) - a\|^2$$

where *w* denotes the collection of all weights in the network, *b* all the biases, *n* is the total number of training inputs, *a* is the vector of outputs from the network when *x* is input, and the sum is over all training inputs, x. In the other words – the smaller $C(w,b)$≈0 means that training algorithm has done a good job

*There are 3 different notions: loss function, cost function and objective function.
Loss function is usually a function defined on a data point, prediction and label, and measures the penalty.
Cost function is usually more general. It might be a sum of loss functions over your training set plus some model complexity penalty (regularization). Objective function is the most general term for any function that you optimize during training.

Learning with gradient descent

	Neural network function	Cost function
Input	784 numbers (pixels)	11 935
Output	10 numbers	1 number (the cost or the error)
Parameters	11 935	Many training examples

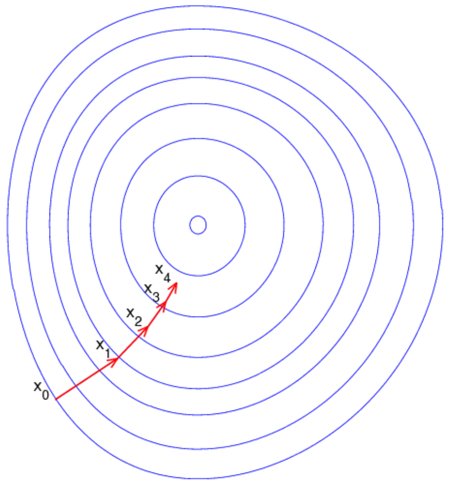
$$\underbrace{(784\text{x}15 + 15\text{x}10)}_{\text{weights}} + \underbrace{(15+10)}_{\text{biases}} = 11\ 935$$

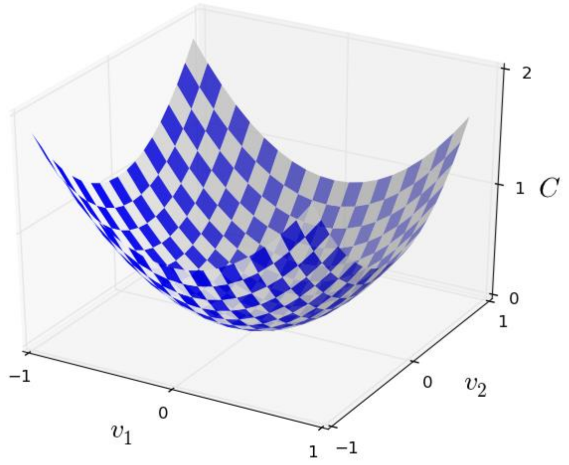
Learning with gradient descent

So our goal in training a neural network is to find weights and biases, which minimize the quadratic cost function

$$C(w,b) = \frac{1}{2n}\sum_x \|y(x) - a\|^2$$

Let's replace w, b with some variables $v=v_1,v_2,\ldots$ => $C(v)$. To minimize $C(v)$ it helps to imagine C as a function of just two variables, which is v_1 *and* v_2:



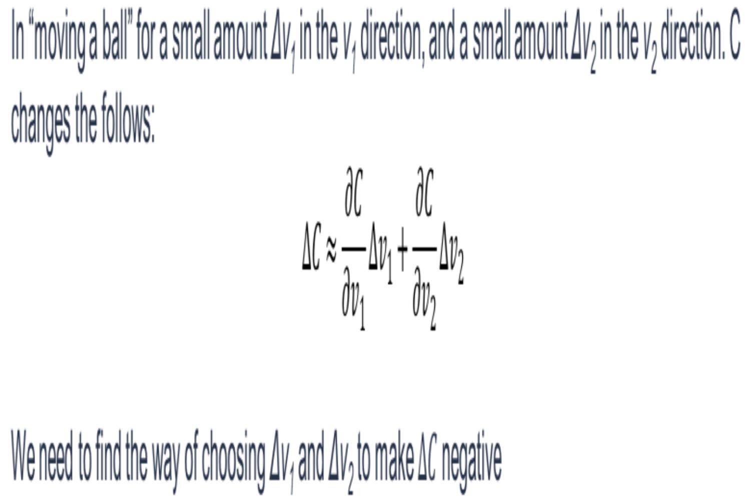


Learning with gradient descent

In "moving a ball" for a small amount Δv_1 in the v_1 direction, and a small amount Δv_2 in the v_2 direction. C changes the follows:

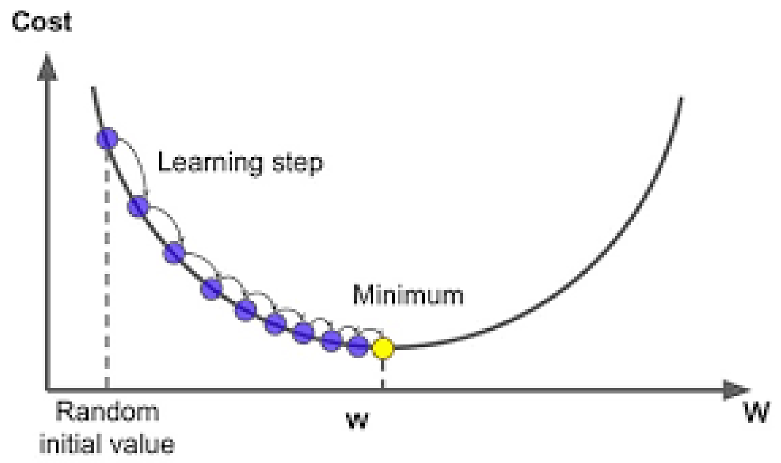
$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

We need to find the way of choosing Δv_1 and Δv_2 to make ΔC negative


In "moving a ball" for a small amount Δv_1 in the v_1 direction, and a small amount Δv_2 in the v_2 direction. C changes the follows:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

We need to find the way of choosing Δv_1 and Δv_2 to make ΔC negative



Learning with gradient descent

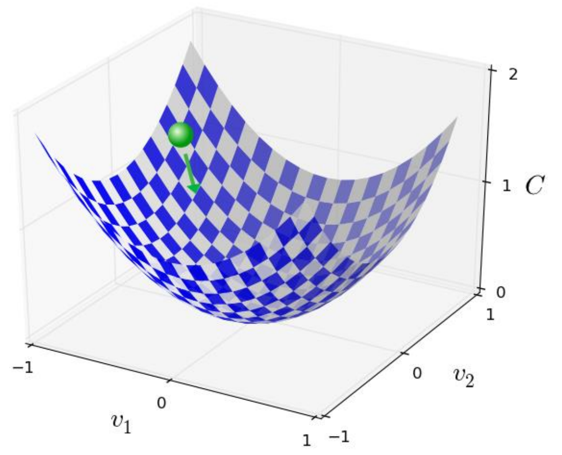
We can re-write this in vector form:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 \;\rightarrow \Delta C \approx \nabla C \cdot \Delta v$$

$$\Delta v = (\Delta v_1, \Delta v_2)^T$$ - vector of changes

$$\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$$ - gradient vector

We need to find the way of choosing Δv_1 and Δv_2 to make ΔC negative.



Δv = - $\eta \cdot \nabla C \rightarrow \Delta C \approx$ - $\eta \cdot ||\nabla C||^2$

η – learning rate

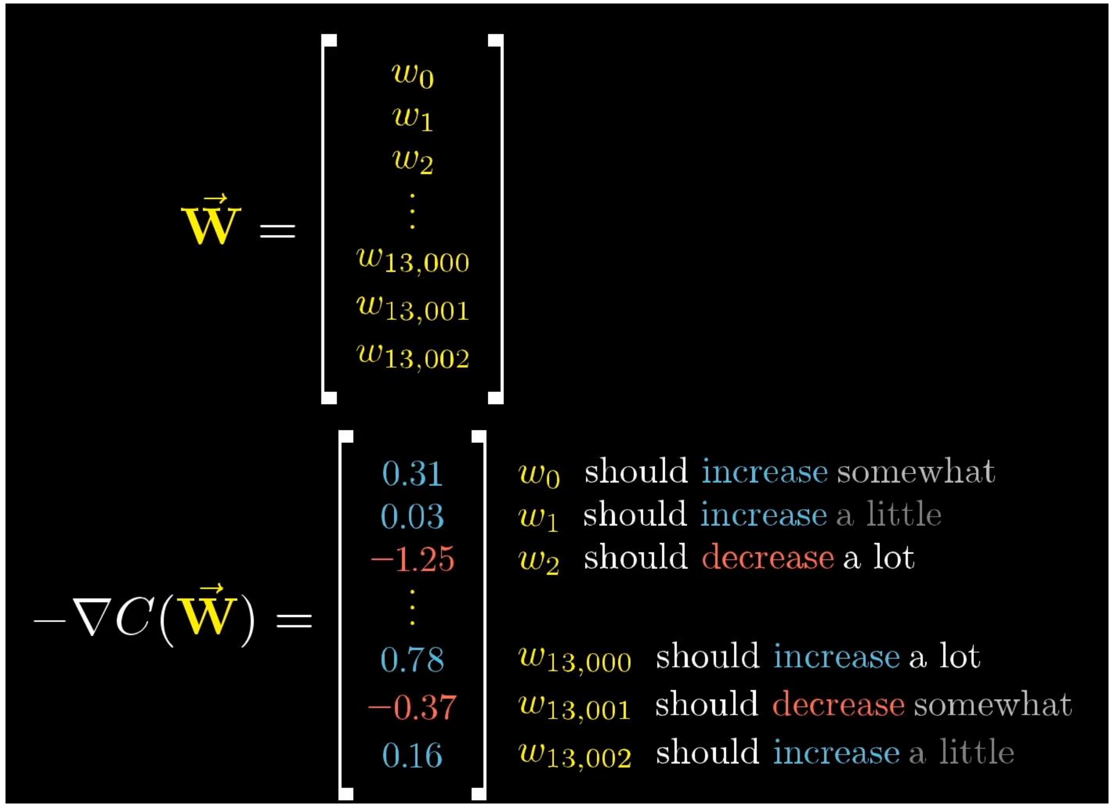
v' =v - $\eta \cdot \nabla C$ – update rule

Δv = - $\eta \cdot \nabla C \rightarrow \Delta C \approx - \eta \cdot ||\nabla C||^2$

η – learning rate

v' =v - $\eta \cdot \nabla C$ – update rule

Learning with gradient descent

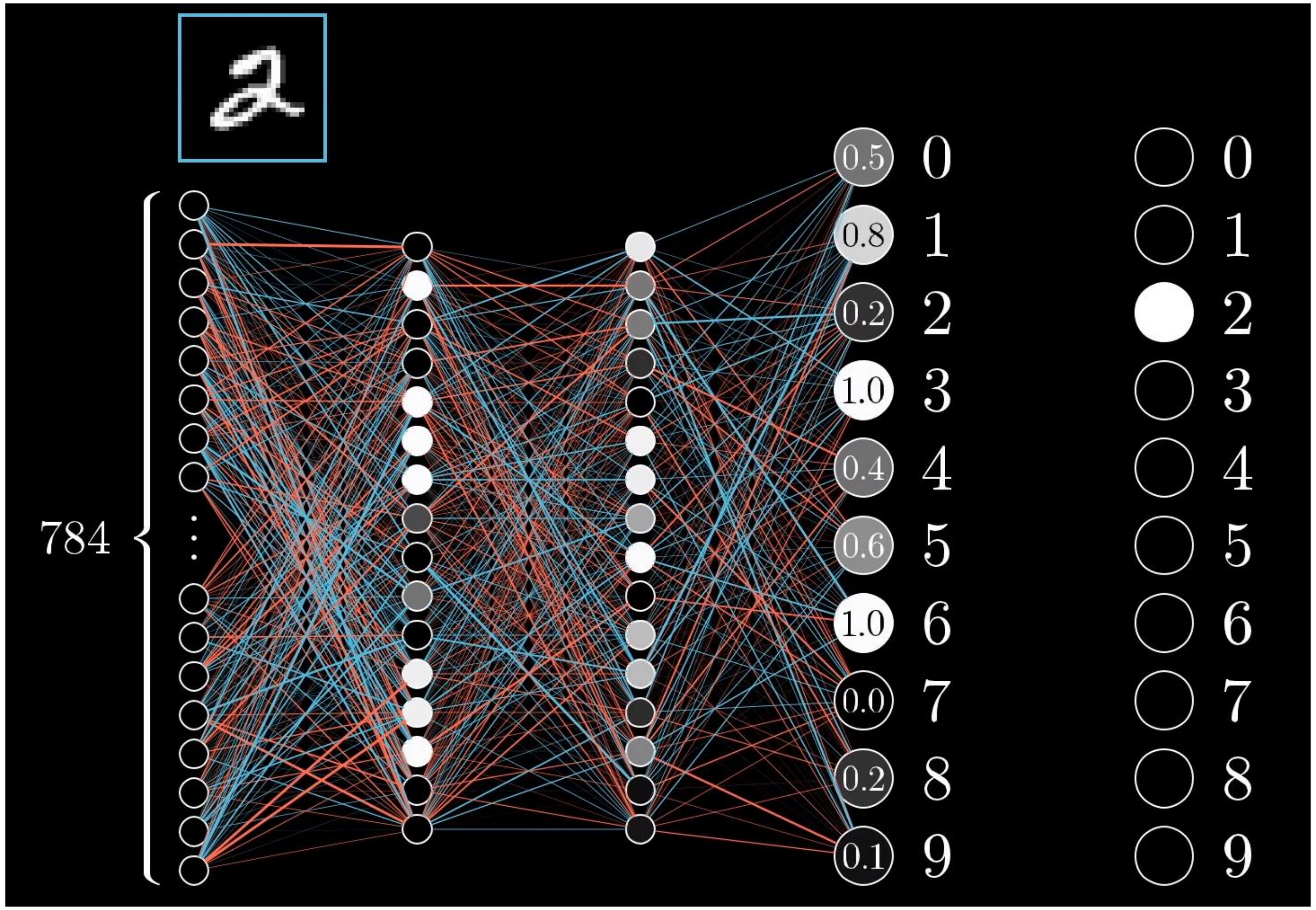

$\vec{\mathbf{W}} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{13,000} \\ w_{13,001} \\ w_{13,002} \end{bmatrix}$
$-\nabla C(\vec{\mathbf{W}}) = \begin{bmatrix} 0.31 \\ 0.03 \\ -1.25 \\ \vdots \\ 0.78 \\ -0.37 \\ 0.16 \end{bmatrix}$
w_0 should increase somewhat
w_1 should increase a little
w_2 should decrease a lot
$w_{13,000}$ should increase a lot
$w_{13,001}$ should decrease somewhat
$w_{13,002}$ should increase a little

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$
$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

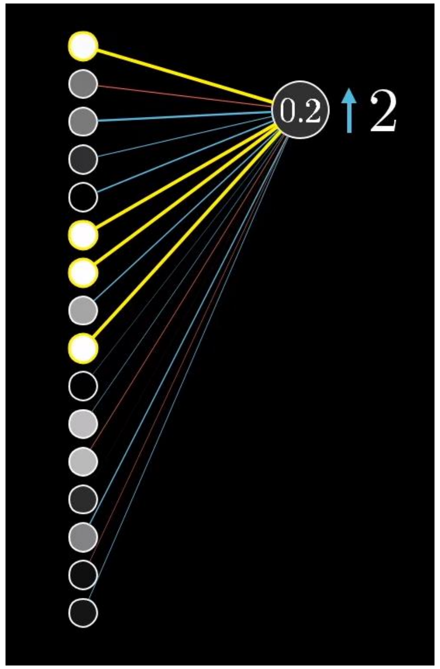
$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$
$$b_l \rightarrow b_l' = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

After first training we pick out another randomly chosen mini-batch and train with those. And so on, until we've exhausted the training inputs, which is said to complete an ***epoch*** of training. At that point we start over with a new training epoch.

Backpropagation

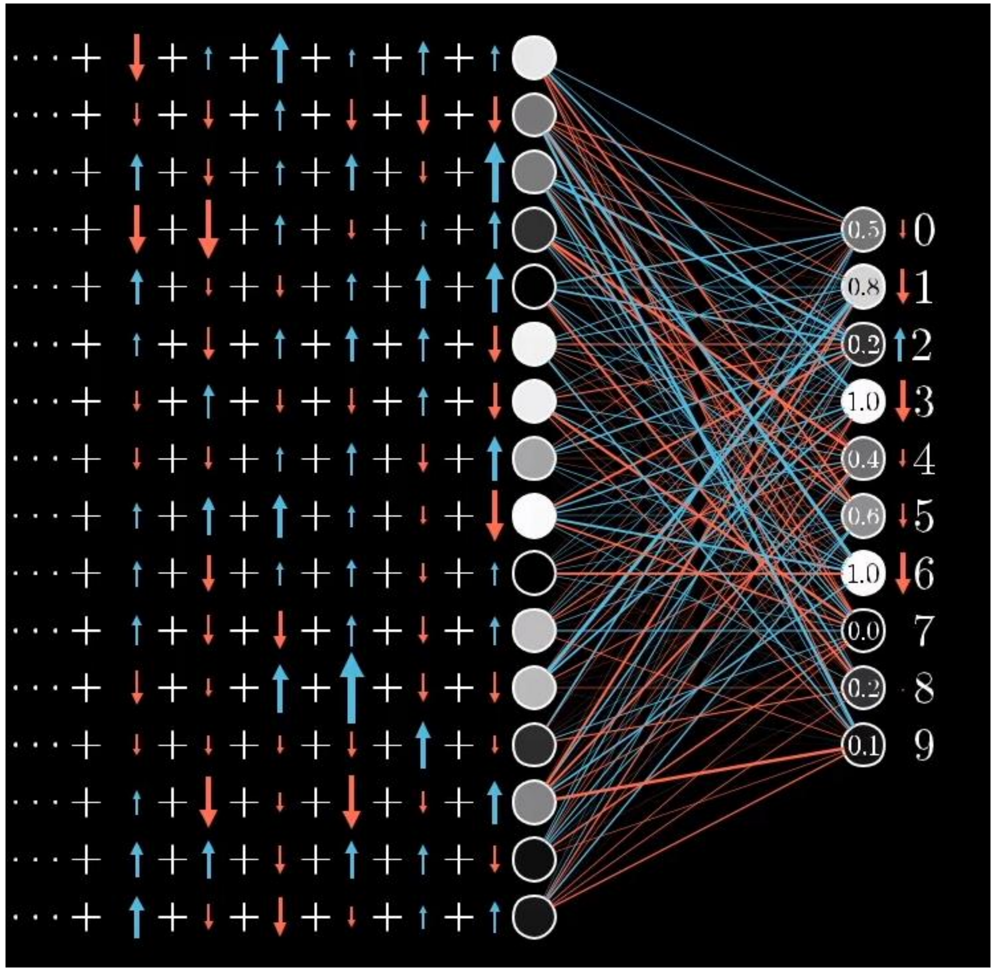


$$0.2 = \sigma(w_1 a_1 + w_2 a_2 + w_3 a_3 + w_4 a_4 + \ldots + w_n a_n + bias)$$



- Increase b
- Increase w_i in proportion to a_i
- Change a_i in proportion to w_i

Backpropagation

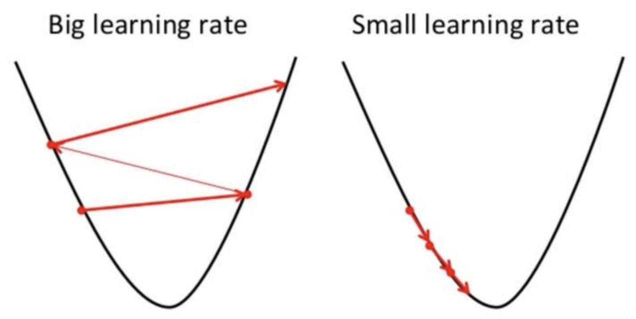


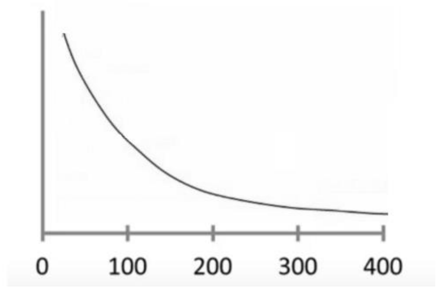
- $\eta \cdot \nabla C(W)$ calculates over all training data

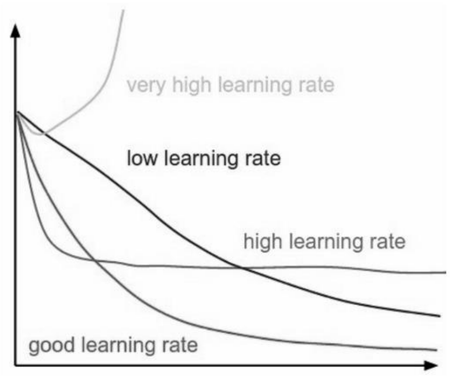
Learning with gradient descent

Types of Gradient Descent:

- **Batch** - calculates the error for each example within the training dataset, but only after all training examples have been evaluated, the model gets updated. This whole process is like a cycle and called a training epoch.
- **Stochastic** - it updates the parameters for each training example, one by one. This can make SGD faster than Batch Gradient Descent
- **Mini Batch** - it's a combination of the concepts of SGD and Batch Gradient Descent. It simply splits the training dataset into small batches and performs an update for each of these batches. Therefore it creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

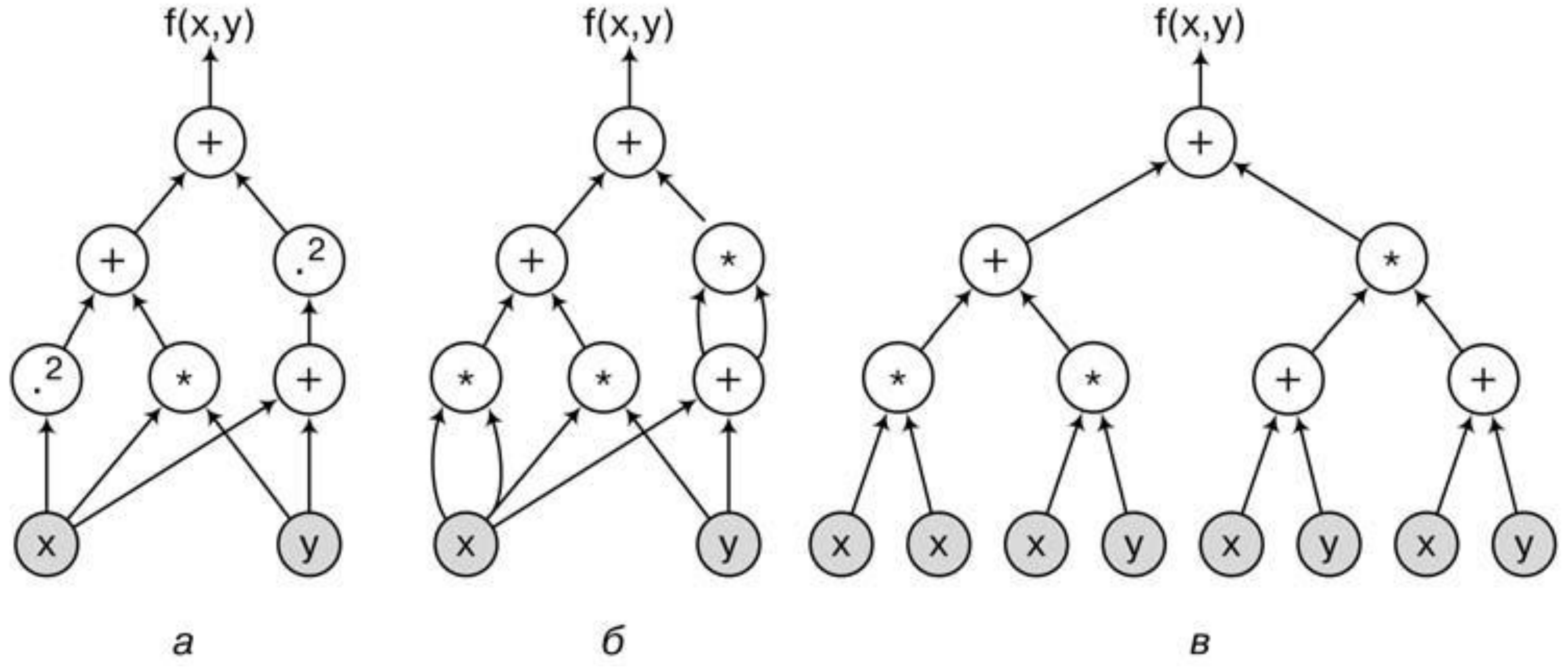






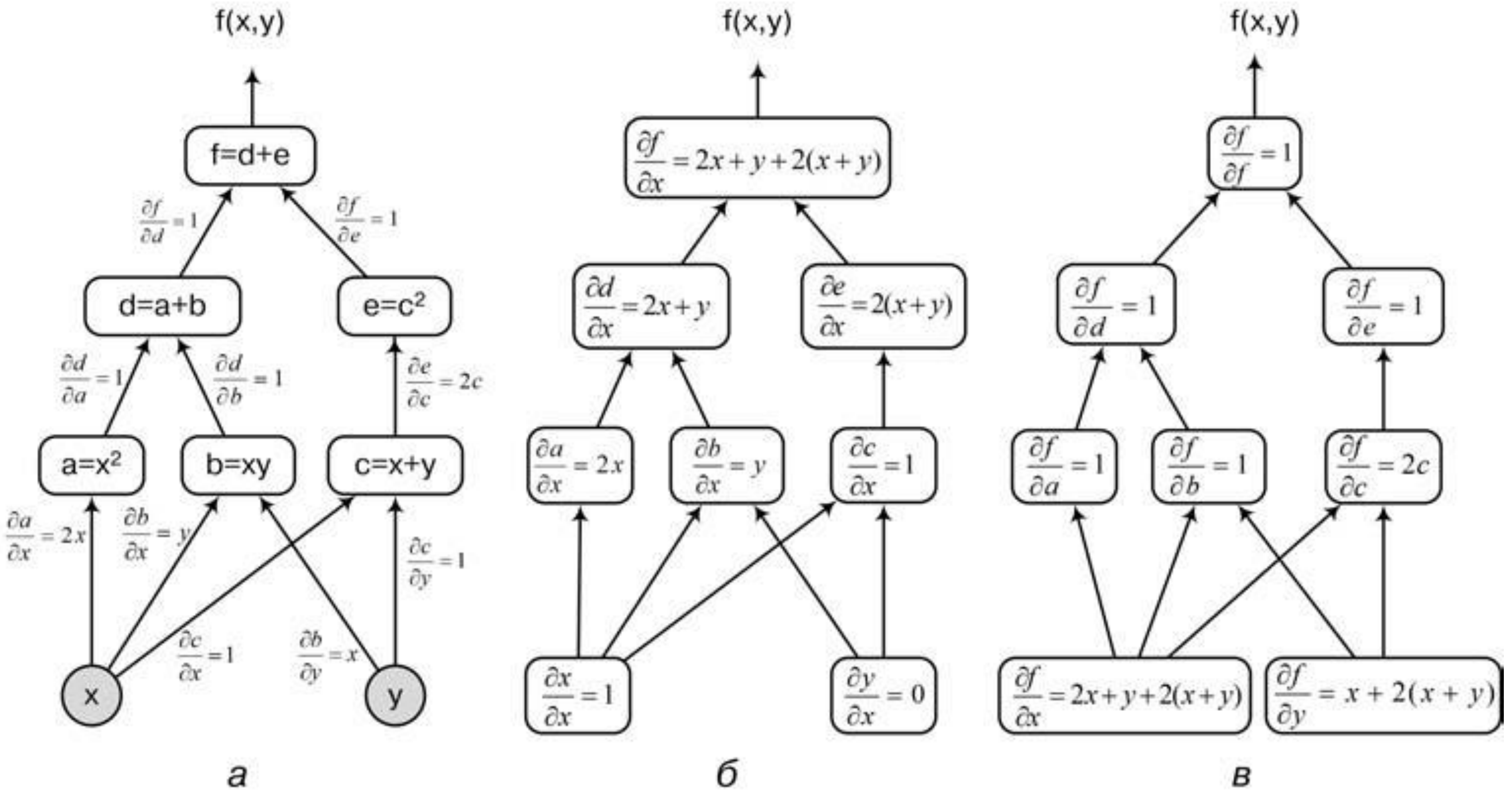
Derivatives of computational graph of function

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



$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial x} + \frac{\partial f}{\partial c}\frac{\partial c}{\partial x} = 2x + y + 2(x+y)$$

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

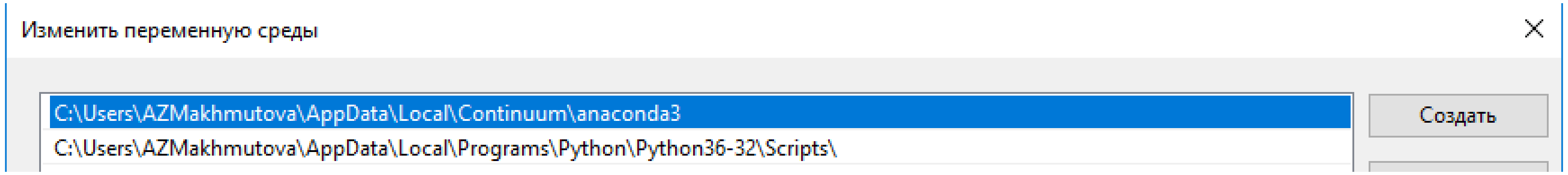


f(x,y)
f=d+e
$\frac{\partial f}{\partial d}=1$
$\frac{\partial f}{\partial e}=1$
d=a+b
e=c²
$\frac{\partial d}{\partial a}=1$
$\frac{\partial d}{\partial b}=1$
$\frac{\partial e}{\partial c}=2c$
a=x²
b=xy
c=x+y
$\frac{\partial a}{\partial x}=2x$
$\frac{\partial b}{\partial x}=y$
$\frac{\partial c}{\partial y}=1$
$\frac{\partial c}{\partial x}=1$
$\frac{\partial b}{\partial y}=x$
x
y
а
f(x,y)
$\frac{\partial f}{\partial x}=2x+y+2(x+y)$
$\frac{\partial d}{\partial x}=2x+y$
$\frac{\partial e}{\partial x}=2(x+y)$
$\frac{\partial a}{\partial x}=2x$
$\frac{\partial b}{\partial x}=y$
$\frac{\partial c}{\partial x}=1$
$\frac{\partial x}{\partial x}=1$
$\frac{\partial y}{\partial x}=0$
б
f(x,y)
$\frac{\partial f}{\partial f}=1$
$\frac{\partial f}{\partial d}=1$
$\frac{\partial f}{\partial e}=1$
$\frac{\partial f}{\partial a}=1$
$\frac{\partial f}{\partial b}=1$
$\frac{\partial f}{\partial c}=2c$
$\frac{\partial f}{\partial x}=2x+y+2(x+y)$
$\frac{\partial f}{\partial y}=x+2(x+y)$
в

Setting up the environment

- Download and install Anaconda Data Science Libraries 3.6
- https://www.anaconda.com/download/

- Install Anaconda and add it to the Path variable


Изменить переменную среды

C:\Users\AZMakhmutova\AppData\Local\Continuum\anaconda3
C:\Users\AZMakhmutova\AppData\Local\Programs\Python\Python36-32\Scripts\

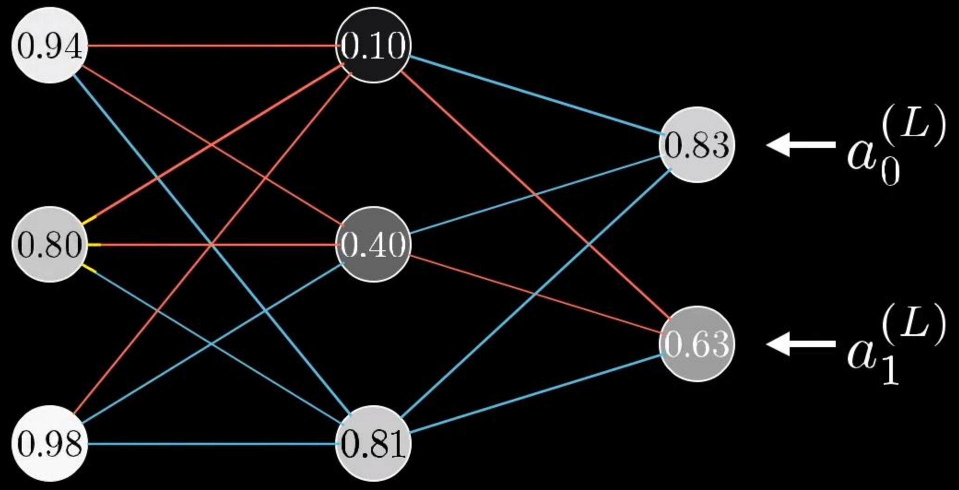
Backpropagation

$$\boxed{\frac{\partial C_0}{\partial a_k^{(L-1)}}} = \underbrace{\sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}}}_{\text{Sum over layer L}}$$

$$z_j^{(L)} = \cdots + w_{jk}^{(L)} a_k^{(L-1)} + \cdots$$

$$a_j^{(L)} = \sigma(z_j^{(L)})$$

$$C_0 = \sum_{j=0}^{n_L-1} (a_j^{(L)} - y_j)^2$$

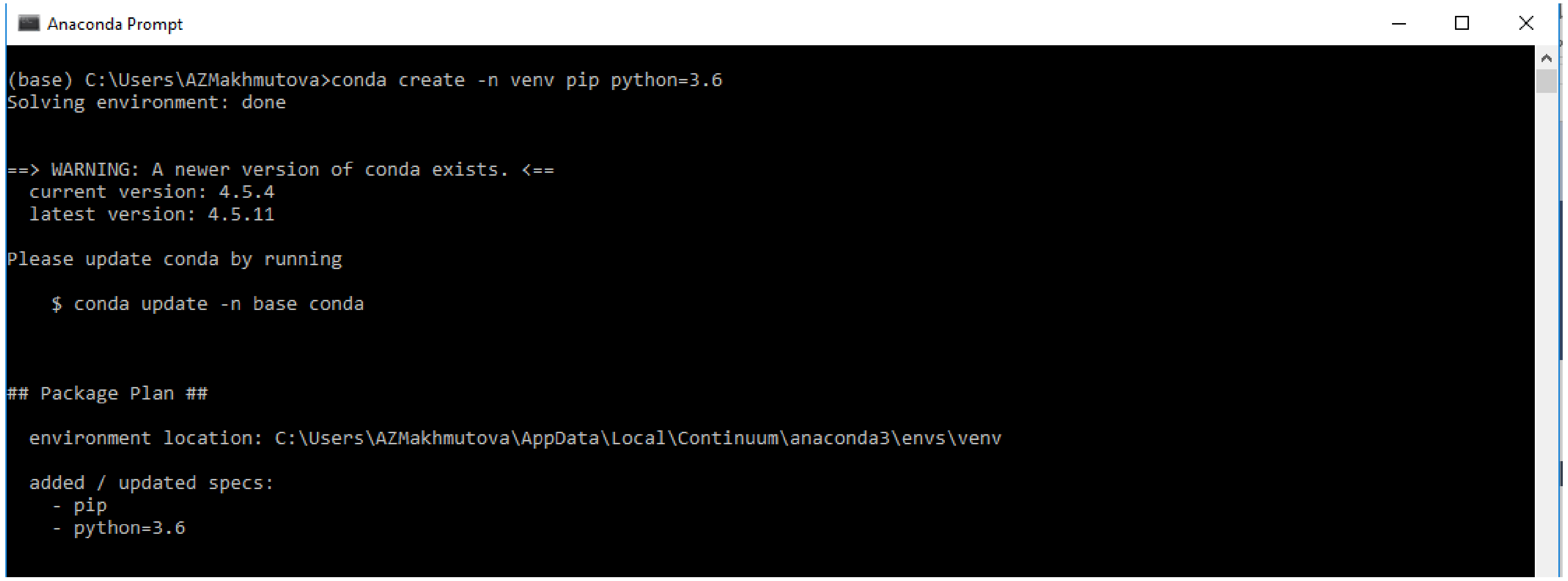


Setting up the environment

- Go to the Anaconda Prompt (Start->Programs->Anaconda3->Anaconda Prompt)

conda create -n venv pip python=3.6

We are creating virtual enviroment to isolate package installation from the system

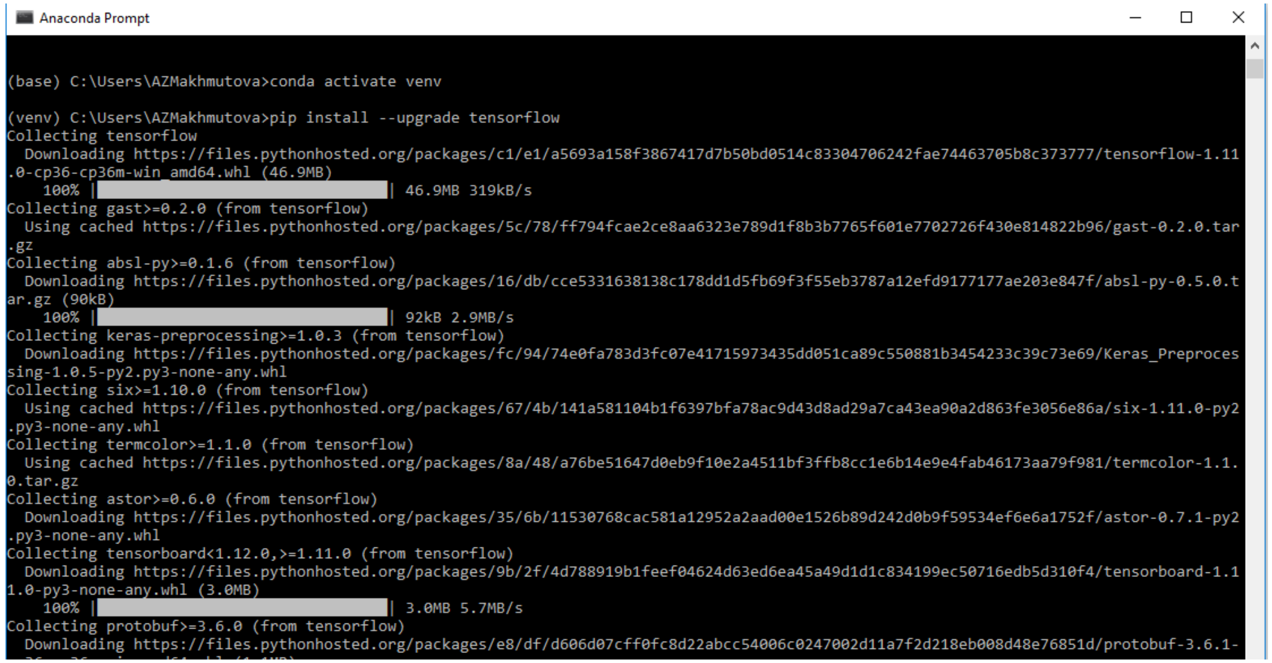


```
(base) C:\Users\AZMakhmutova>conda create -n venv pip python=3.6
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.5.4
  latest version: 4.5.11

Please update conda by running

    $ conda update -n base conda



## Package Plan ##

  environment location: C:\Users\AZMakhmutova\AppData\Local\Continuum\anaconda3\envs\venv

  added / updated specs:
    - pip
    - python=3.6
```

Setting up the environment

- Then activate the enviroment by *conda activate venv*
- And then install simple version of tensorflow (for CPU only) by

conda --upgrade tensorflow



```
(base) C:\Users\AZMakhmutova>conda activate venv

(venv) C:\Users\AZMakhmutova>pip install --upgrade tensorflow
Collecting tensorflow
  Downloading https://files.pythonhosted.org/packages/c1/e1/a5693a158f3867417d7b50bd0514c83304706242fae74463705b8c373777/tensorflow-1.11
.0-cp36-cp36m-win_amd64.whl (46.9MB)
    100% |████████████████████████████████| 46.9MB 319kB/s
Collecting gast>=0.2.0 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/5c/78/ff794fcae2ce8aa6323e789d1f8b3b7765f601e7702726f430e814822b96/gast-0.2.0.tar
.gz
Collecting absl-py>=0.1.6 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/16/db/cce5331638138c178dd1d5fb69f3f55eb3787a12efd9177177ae203e847f/absl-py-0.5.0.t
ar.gz (90kB)
    100% |████████████████████████████████| 92kB 2.9MB/s
Collecting keras-preprocessing>=1.0.3 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/fc/94/74e0fa783d3fc07e41715973435dd051ca89c550881b3454233c39c73e69/Keras_Preproces
sing-1.0.5-py2.py3-none-any.whl
Collecting six>=1.10.0 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/67/4b/141a581104b1f6397bfa78ac9d43d8ad29a7ca43ea90a2d863fe3056e86a/six-1.11.0-py2
.py3-none-any.whl
Collecting termcolor>=1.1.0 (from tensorflow)
  Using cached https://files.pythonhosted.org/packages/8a/48/a76be51647d0eb9f10e2a4511bf3ffb8cc1e6b14e9e4fab46173aa79f981/termcolor-1.1.
0.tar.gz
Collecting astor>=0.6.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/35/6b/11530768cac581a12952a2aad00e1526b89d242d0b9f59534ef6e6a1752f/astor-0.7.1-py2
.py3-none-any.whl
Collecting tensorboard<1.12.0,>=1.11.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/9b/2f/4d788919b1feef04624d63ed6ea45a49d1d1c834199ec50716edb5d310f4/tensorboard-1.1
1.0-py3-none-any.whl (3.0MB)
    100% |████████████████████████████████| 3.0MB 5.7MB/s
Collecting protobuf>=3.6.0 (from tensorflow)
  Downloading https://files.pythonhosted.org/packages/e8/df/d606d07cff0fc8d22abcc54006c0247002d11a7f2d218eb008d48e76851d/protobuf-3.6.1-
```

- After successfull installation deactivate the enviroment by

conda deactivate

Setting up the environment

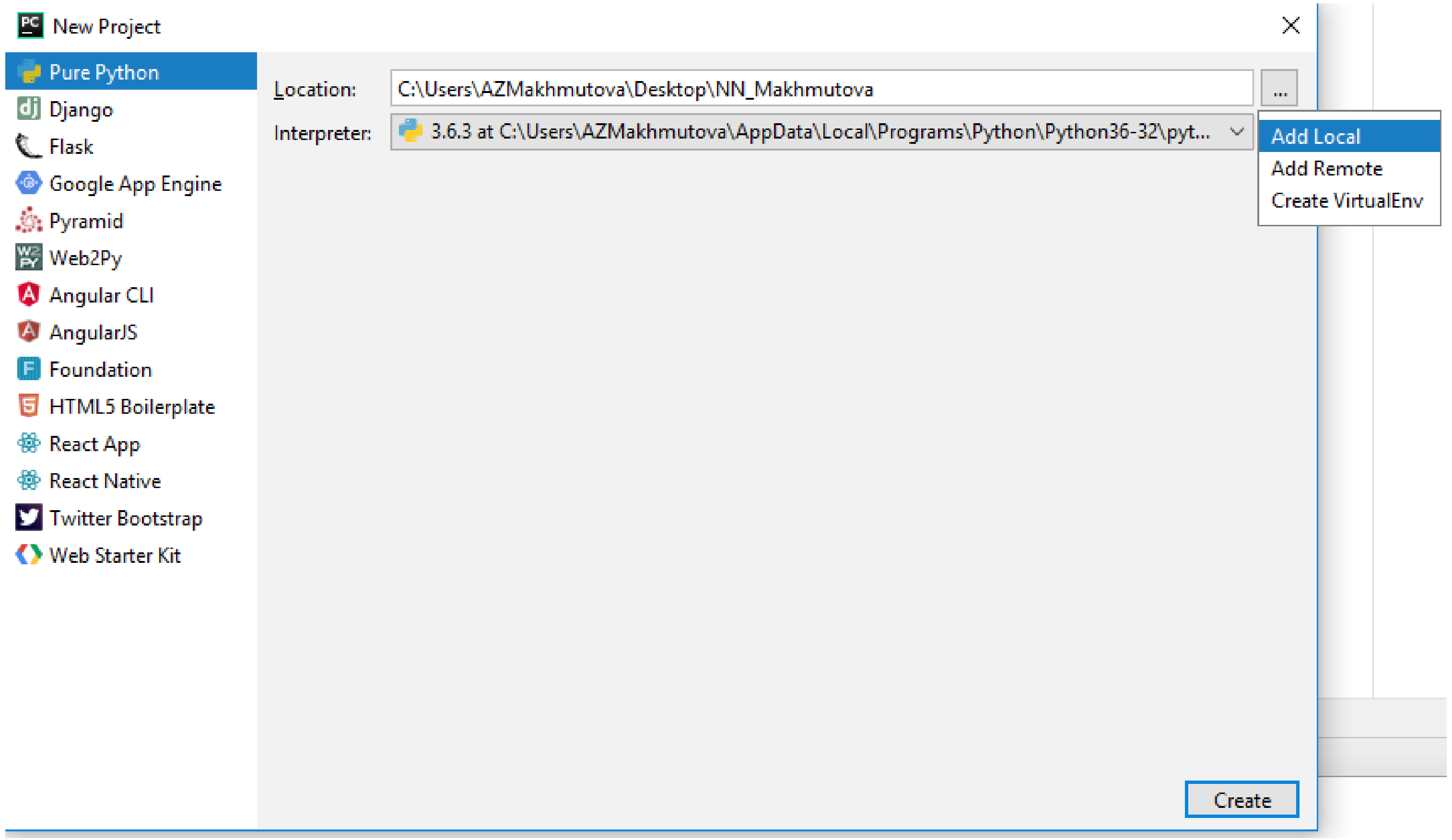
- Install PyCharm - Python IDE from https://www.jetbrains.com/pycharm/download/#section=windows

As a student you can have free individual licenses (fastest way with your KAI email) https://www.jetbrains.com/student/

Install PyCharm by default settings

Setting up the environment

- Create a new project with Project Interpreter from virtual environment



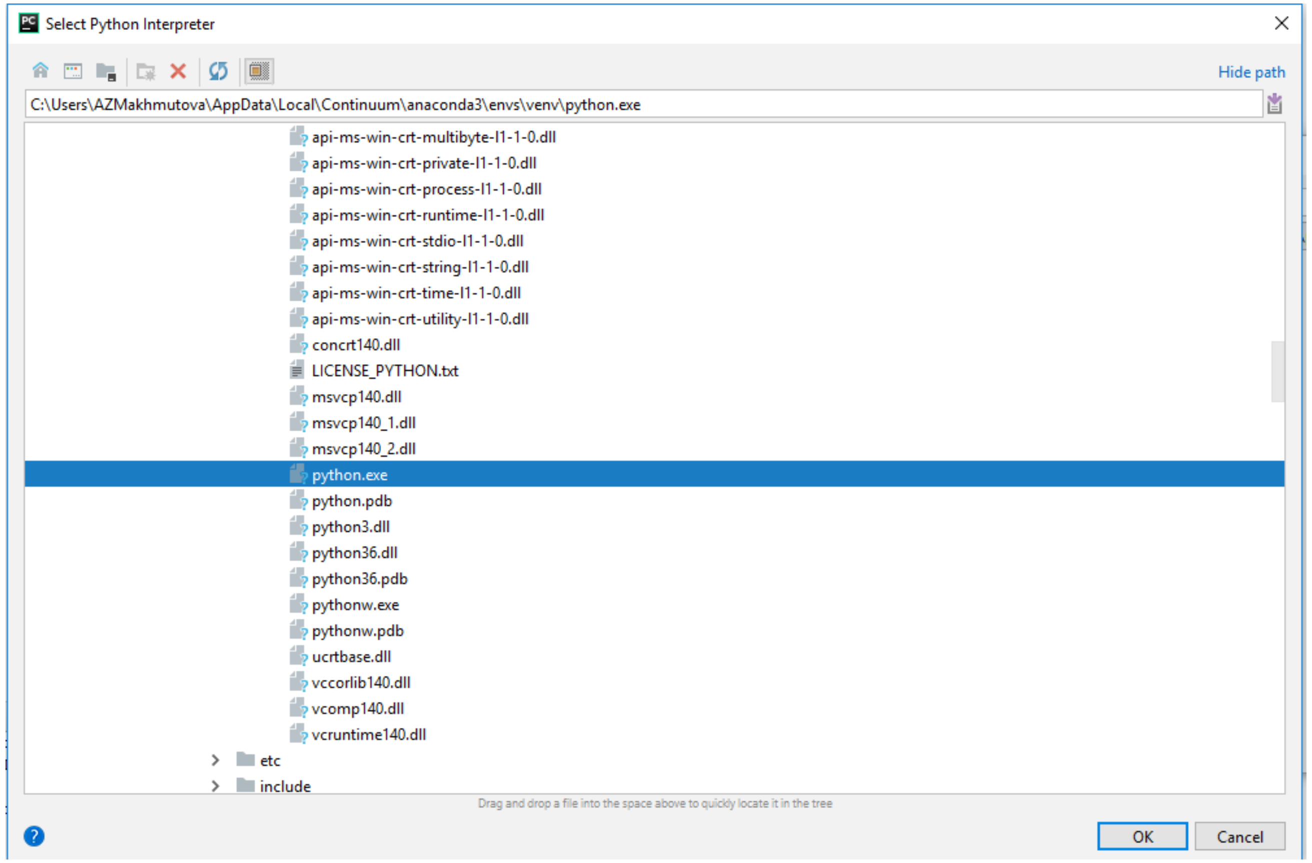
New Project

Location: C:\Users\AZMakhmutova\Desktop\NN_Makhmutova

Interpreter: 3.6.3 at C:\Users\AZMakhmutova\AppData\Local\Programs\Python\Python36-32\pyt...

Setting up the environment

- Choose Project Interpreter from virtual environment that we just created



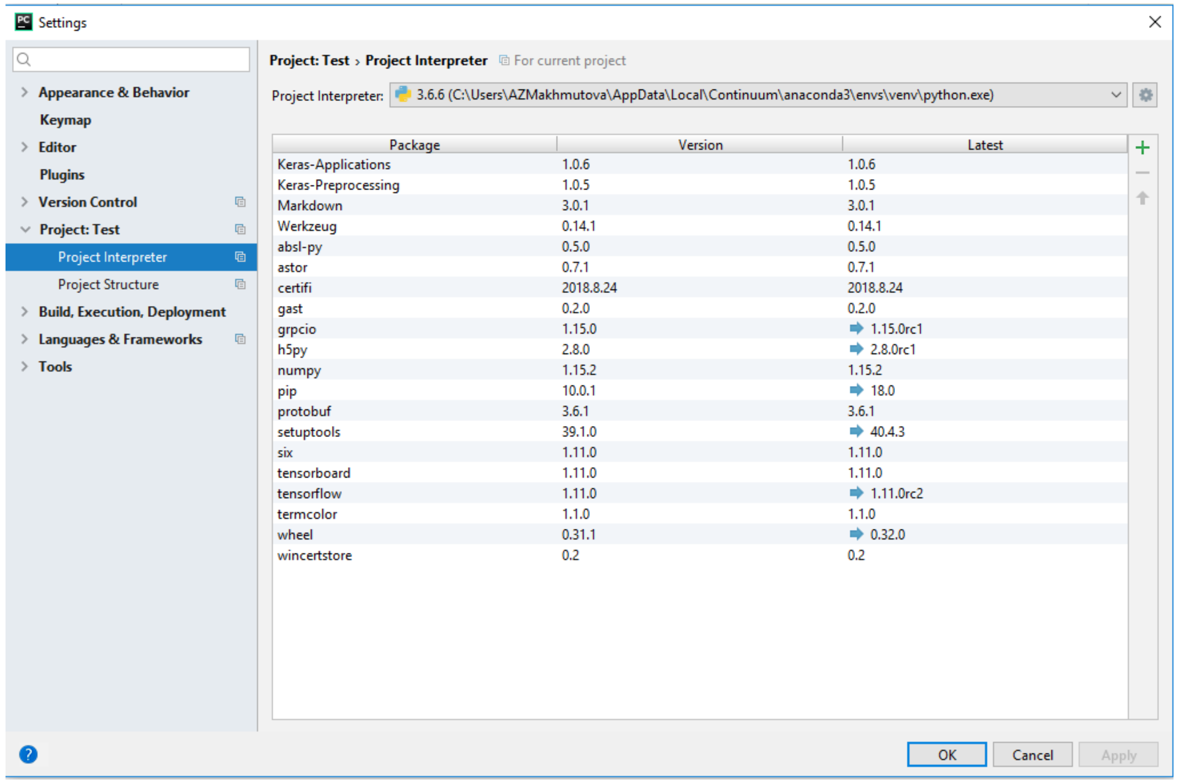
```
C:\Users\AZMakhmutova\AppData\Local\Continuum\anaconda3\envs\venv\python.exe
```

```
api-ms-win-crt-multibyte-l1-1-0.dll
api-ms-win-crt-private-l1-1-0.dll
api-ms-win-crt-process-l1-1-0.dll
api-ms-win-crt-runtime-l1-1-0.dll
api-ms-win-crt-stdio-l1-1-0.dll
api-ms-win-crt-string-l1-1-0.dll
api-ms-win-crt-time-l1-1-0.dll
api-ms-win-crt-utility-l1-1-0.dll
concrt140.dll
LICENSE_PYTHON.txt
msvcp140.dll
msvcp140_1.dll
msvcp140_2.dll
python.exe
python.pdb
python3.dll
python36.dll
python36.pdb
pythonw.exe
pythonw.pdb
ucrtbase.dll
vccorlib140.dll
vcomp140.dll
vcruntime140.dll
> etc
> include
```

Drag and drop a file into the space above to quickly locate it in the tree

Setting up the environment

- You can always check/change the current interpreter in the project
- *File -> Settings -> Project:Somename ->Project Interpreter*


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	➡ 1.15.0rc1
h5py	2.8.0	➡ 2.8.0rc1
numpy	1.15.2	1.15.2
pip	10.0.1	➡ 18.0
protobuf	3.6.1	3.6.1
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	➡ 1.11.0rc2
termcolor	1.1.0	1.1.0
wheel	0.31.1	➡ 0.32.0
wincertstore	0.2	0.2

Setting up the environment

- Let’s check the work of tensorflow. Create simple python file in the project and run following code:

```
import tensorflow as tf


hello = tf.constant ('Hello, TensorFlow!')

sess = tf.Session()
print(sess.run(hello))

a = tf.constant(10)
b = tf.constant(32)

print (sess.run(a+b))
```

- The output should be 42. If so, you set up the environment!

Getting familiar with tensorflow as tf

Variables in tf :

```
w = tf.Variable(tf.random_normal([3, 2], mean=0.0, stddev=0.4), name='weights')
b = tf.Variable(tf.zeros([2]), name='biases')
```

```
with tf.device('/gpu:0'):
w = tf.Variable(tf.random_normal([3, 2], mean=0.0, stddev=0.4), name='weights')
```

```
with tf.device('/job:ps/task:0'):
w = tf.Variable(tf.random_normal([3, 2], mean=0.0, stddev=0.4), name='weights')
```

Getting familiar with tensorflow as tf

- Initialization of the variables in tf :

```
init = tf.initialize_global_variables()

w2 = tf.Variable(w.initialized_value(), name='w2')



saved = saver.save(sess, 'model.ckpt')

saver.restore('model.ckpt')
```

Getting familiar with tensorflow as tf

- tf.placeholder:

```
import tensorflow as tf

x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
output = tf.multiply(x, y)
with tf.Session() as sess:
    result = sess.run(output, feed_dict={x: 2, y: 3})

print(result)
```

Getting familiar with tensorflow as tf

- broadcasting:

```
m = tf.Variable(tf.random_normal([10, 100], mean=0.0, stddev=0.4), name='matrix')
v = tf.Variable(tf.random_normal([100], mean=0.0, stddev=0.4), name='vector')
result = m + v

print(result)
```

Getting familiar with tensorflow as tf

- variable scopes:
- tf.get_variable(<name>, <shape>, <initializer>) creates and returns a variable with the specified name;
- • tf.variable_scope(<scope_name>) manages the namespaces used in tf.get_variable();

```
def linear_transform(vec, shape):
    with tf.variable_scope('transform'):
        w = tf.get_variable('matrix', shape,
        initializer=tf.random_normal_initializer())
    return tf.matmul(vec, w)


with tf.variable_scope('linear_transformers') as scope:
       result1 = linear_transform(vec1, shape)
       scope.reuse_variables()
       result2 = linear_transform(vec2, shape)
```

Linear regression training

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

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

```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))
```

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

Network implementation

MNIST data set - MNIST's name comes from the fact that it is a modified subset of two data sets collected by NIST, the United States' National Institute of Standards and Technology

http://yann.lecun.com/exdb/mnist/

Training data (set) is used for train the NN, but tests are made on test data!

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}))
```