



# Нейросети и искусственный интеллект на платформе .NET

**Дмитрий Сошников**

Технологический евангелист, Microsoft

[dmitryso@microsoft.com](mailto:dmitryso@microsoft.com)

[fb.com/shwars](https://fb.com/shwars)



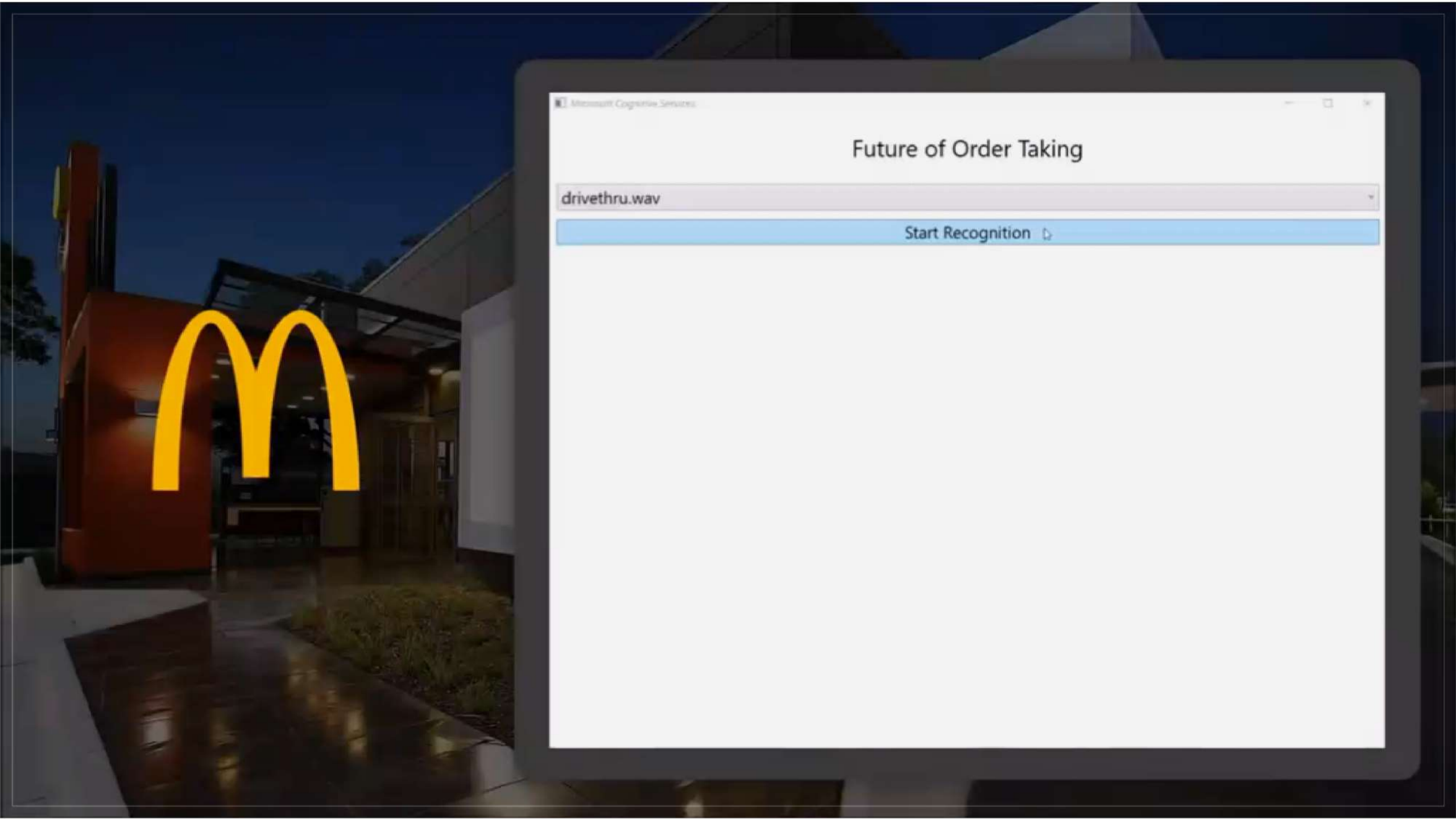
## Искусственный интеллект в .NET

[dotnext-moscow.ru](http://dotnext-moscow.ru)

Доклад от эксперта из  
Microsoft на .NET-  
конференции DotNext.







Microsoft Cognitive Services

## Future of Order Taking

drivethru.wav

Start Recognition ▶

# Microsoft и AI

- Achieving human parity for Speech Recognition

- <https://arxiv.org/abs/1610.05256>

- Winners in ImageNet 2015 Challenge

- <http://image-net.org/challenges/LSVRC/2015/results>
- Proposal of deep residual networks => human parity in image recognition

- Project Brainwave

- Running AI on FPGA chips in the cloud
- <https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/>



# Стратегия Microsoft в области ИИ



Демократизация  
ИИ

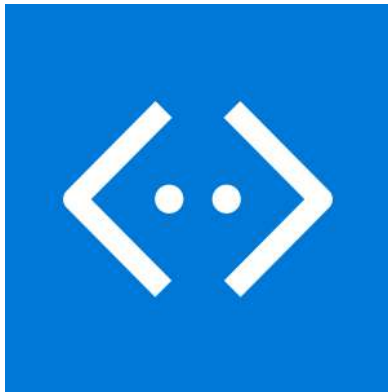


Внедрение ИИ в  
продукты



Исследования  
(FPGA в облаке, ..)

# Демократизация искусственного интеллекта



Bot Framework

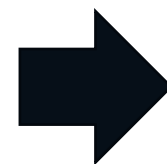
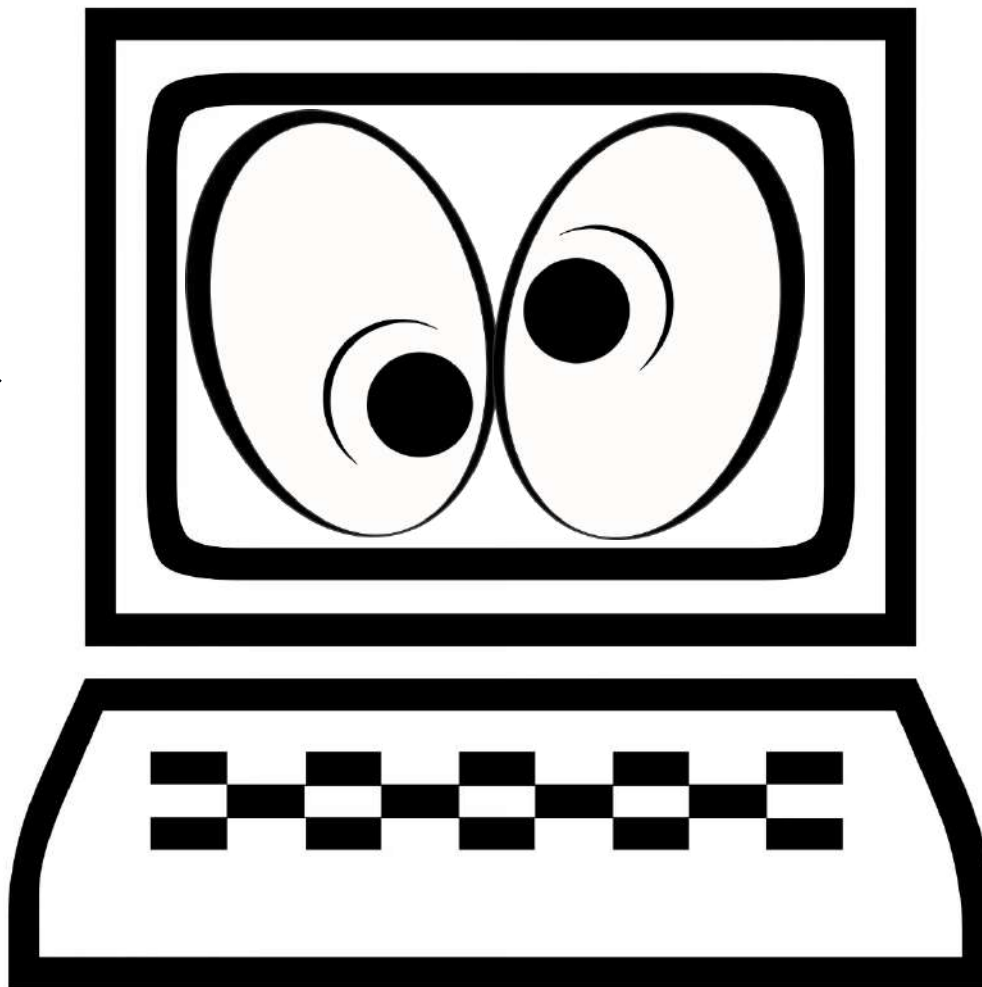




# Машинное обучение

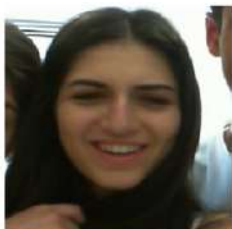


Data



Model

Happy

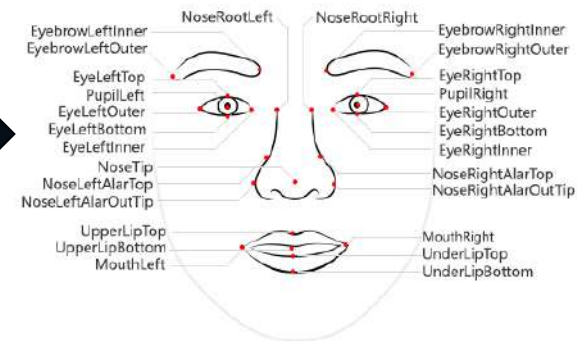


Surprise

Sad



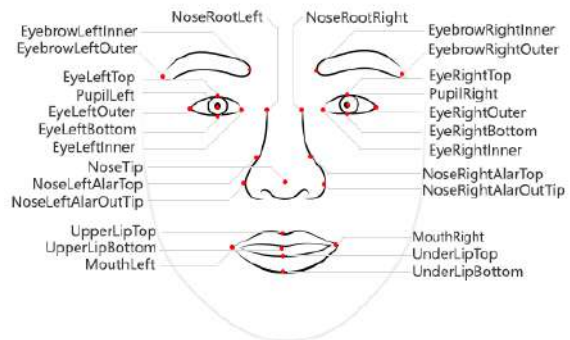
Anger



Copyright (c) Microsoft. All rights reserved.



Feature  
Extraction



Copyright (c) Microsoft. All rights reserved.



Model



Happy: 80%

# Немного про терминологию



*Почти всё, что вы слышали в последние годы под названием Искусственный интеллект – это нейросети*

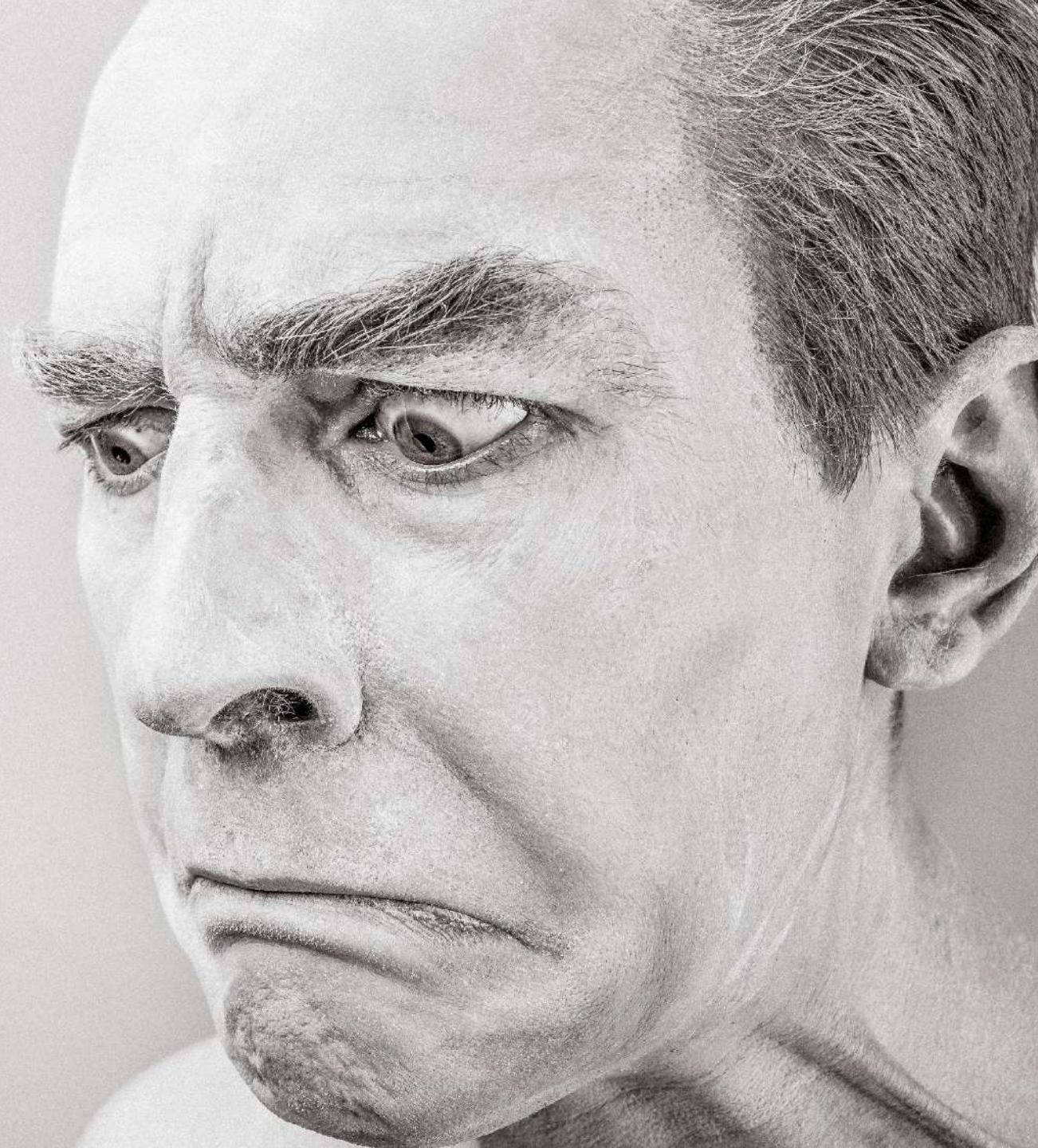
# Как живут Data Scientist-ы



== All Inclusive (CRAN) ==



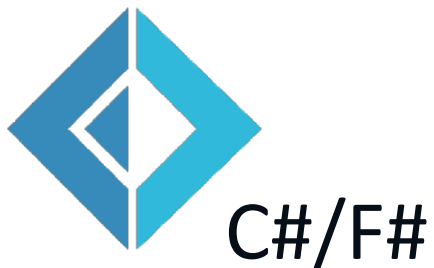
Все вокруг говорят  
про глубокое  
обучение, а я не  
знаю Python...



# Как живут Data Scientist-ы



== All Inclusive (CRAN) ==



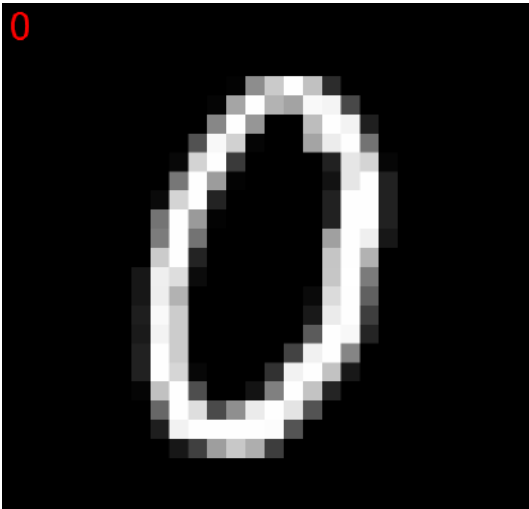
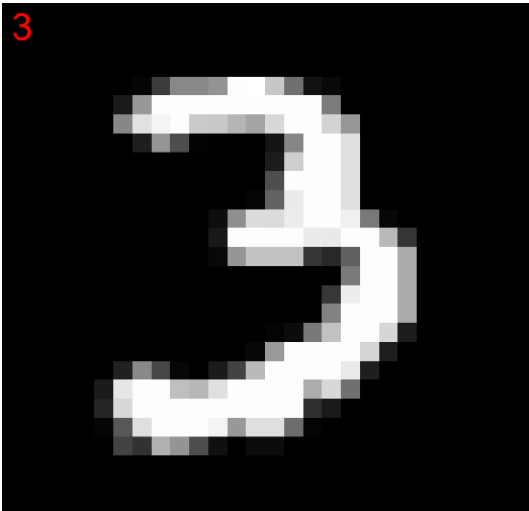


# Эпизод 1

Сложно ли самому  
запрограммировать машинное  
обучение?

# MNIST DataSet

<http://www.kaggle.com/c/digit-recognizer>

[illegible]

# 28 x 28 пикселей

Градации серого: 0 = чёрный, 255 - белый

## Плоские записи: цифра + 784 пикселей

## Формат CSV

## Обучающая выборка - 50,000 примеров

<https://notebooks.azure.com/sosh/libraries/fsharpdojo>

## Azure Notebooks: F#

jupyter KNNClassifier\_Solution Last Checkpoint: 10 minutes ago (unsaved changes) IF[#] Azure Notebooks My Libraries F# Coding Dojo

File Edit View Insert Cell Kernel Data Widgets Help Not Trusted F#

In [19]:

```
let convert (input:string) =  
    let a = input.Split(',')  
    (a.[0] |> int, a.[1..] |> Array.map (int))  
  
let train_data = train_sample |> Array.map convert
```

In [20]:

```
let test_data = test_sample |> Array.map convert
```

The method we will use to predict the correct number is called **K Nearest Neighbour Classifier** (KNN).

The simplest version is when  $K = 1$ . Suppose we have training data  $\text{train\_data} = \{(d_i, I_i)\}_{i=1}^N$ , where  $d_i \in [0..9]$  is the digit, corresponding to image  $I_i$ . Given the input image  $X$  we look for the image  $I_i$ , which is closest to  $X$  according to some metric  $\|\cdot\|$ , i.e.

$$(d_i, I_i) = \operatorname{argmin}_j \|X - I_j\|$$

In this case, we return  $d_i$  as the result.

Metric  $\|\cdot\|$  can be defined in a few different ways, the simplest being euclidian distance

$$\|A - B\| = \sum_{j=1}^{768} (A_j - B_j)^2$$

When  $K > 1$ , we look for  $K$  images in the training set that are closest to  $X$ , and select the digit which most frequently occurs among digits that correspond to those images.

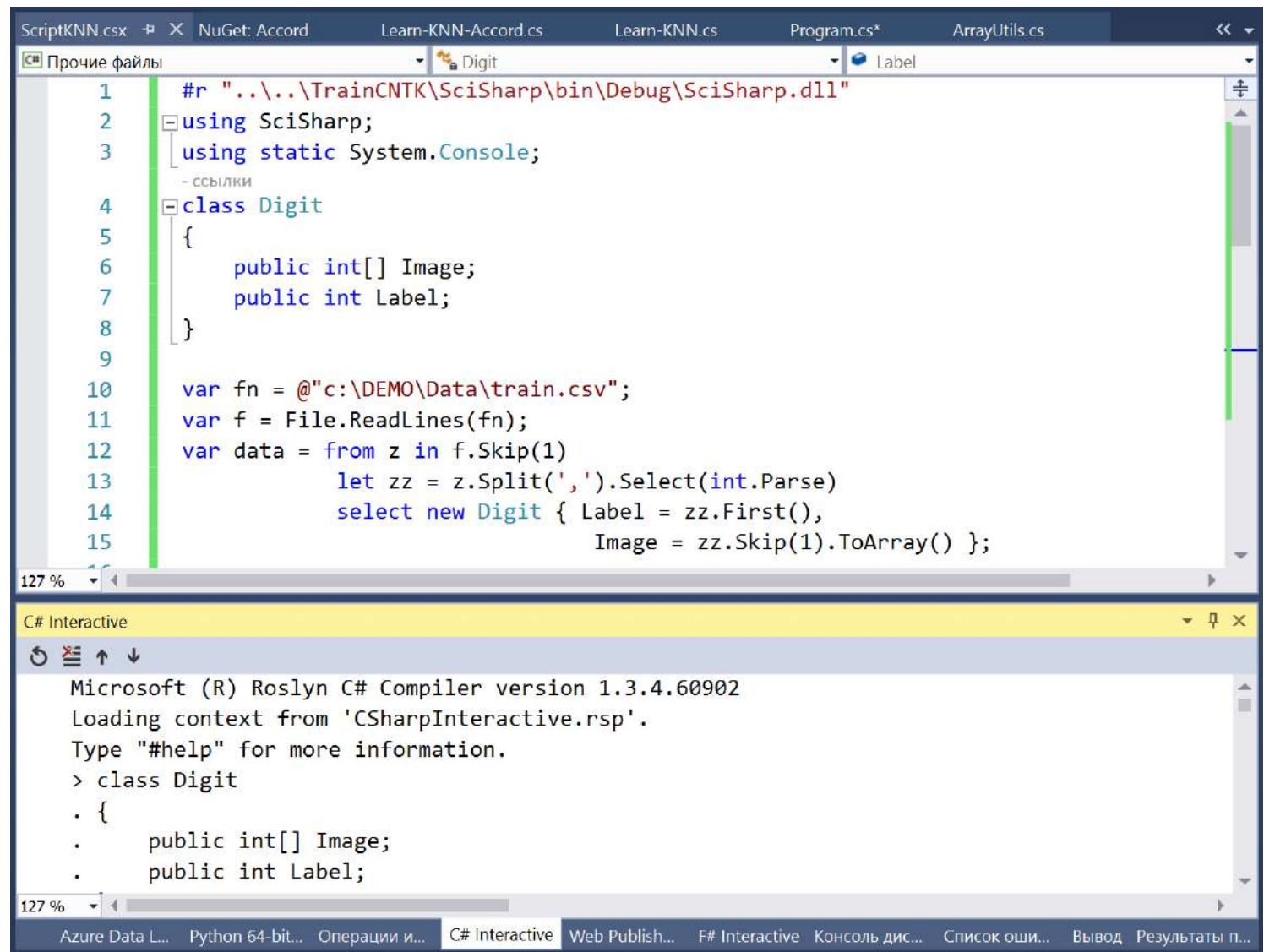
In our case, we need to define the function distance that computes the distance between two images (represented by int arrays).

In [22]:

```
let distance A B = Seq.map2 (fun a b -> (a-b)*(a-b)) A B |> Seq.sum
```

Having defined distance function, let's finally define the classifier classify, which will return the digit, given the image (array of pixels)

# C# Interactive



The screenshot displays the Visual Studio IDE with a C# script file and the C# Interactive window. The script file, named 'Digit', contains the following code:

```
1 #r "..\..\TrainCNTK\SciSharp\bin\Debug\SciSharp.dll"
2 using SciSharp;
3 using static System.Console;
4 class Digit
5 {
6     public int[] Image;
7     public int Label;
8 }
9
10 var fn = @"c:\DEMO\Data\train.csv";
11 var f = File.ReadLines(fn);
12 var data = from z in f.Skip(1)
13             let zz = z.Split(',').Select(int.Parse)
14             select new Digit { Label = zz.First(),
15                               Image = zz.Skip(1).ToArray() };
16
```

The C# Interactive window shows the output of the code execution:

```
Microsoft (R) Roslyn C# Compiler version 1.3.4.60902
Loading context from 'CSharpInteractive.rsp'.
Type "#help" for more information.
> class Digit
. {
.     public int[] Image;
.     public int Label;

```

The bottom of the image shows the Visual Studio status bar with tabs for 'Azure Data L...', 'Python 64-bit...', 'Операции и...', 'C# Interactive', 'Web Publish...', 'F# Interactive', 'Консоль дис...', 'Список оши...', 'Вывод', and 'Результаты п...'.

Select file:///c:/users/dmitryso/documents/visual studio 2015/Projects/Accord/Accord/bin/Debug/Accord.exe

0 => 0  
0 => 0  
7 => 7  
2 => 2  
4 => 4  
1 => 1  
0 => 0  
1 => 1  
2 => 2  
1 => 1  
4 => 4  
1 => 1  
1 => 1  
2 => 2  
6 => 6  
2 => 2  
5 => 5  
9 => 9  
0 => 0  
4 => 4  
0 => 0  
6 => 6  
3 => 3  
7 => 9  
8 => 8  
0 => 0  
1 => 1  
9 => 9  
Done, 943 of 1000 correct (94,3%)

# Подводим итоги

```
Func<int[], int[], int> dist = (a, b) =>  
    a.Zip(b, (x, y) => { return (x - y) * (x - y); }).Sum();
```

```
Func<int[], int> classify = (im) =>  
    train.MinBy(d => dist(d.Image, im)).Label;
```

Алгоритм	Точность
KNN	94%





Эпизод 2:

Accord.NET

# Accord.NET

Математические функции

- Матрицы
- Статистика

Машинное обучение

- Классические алгоритмы
- Нейронные сети
- Глубокое обучение

Обработка звука

Обработка изображений

- Выделение объектов и лица (Haar Cascade)

Элементы визуализации

<http://accord-framework.net>



# Подводим итоги

```
var svm = new MulticlassSupportVectorLearning<Linear>();  
var classifier = svm.Learn(  
    (from x in train select x.Image.Select(z => (double)z).ToArray()).ToArray(),  
    (from x in train select x.Label).ToArray());
```

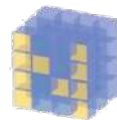
Алгоритм	Точность
KNN (Accord)	94%
SVM (Accord)	92%

- Взаимозаменяемые различные алгоритмы обучения
- Не очень быстрые и эффективные реализации нейронных сетей

# Как живут Data Scientist-ы



SciPy



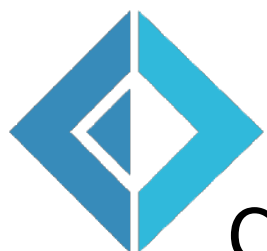
NumPy



Caffe



== All Inclusive (CRAN) ==



C#/F#



Эпизод 3:

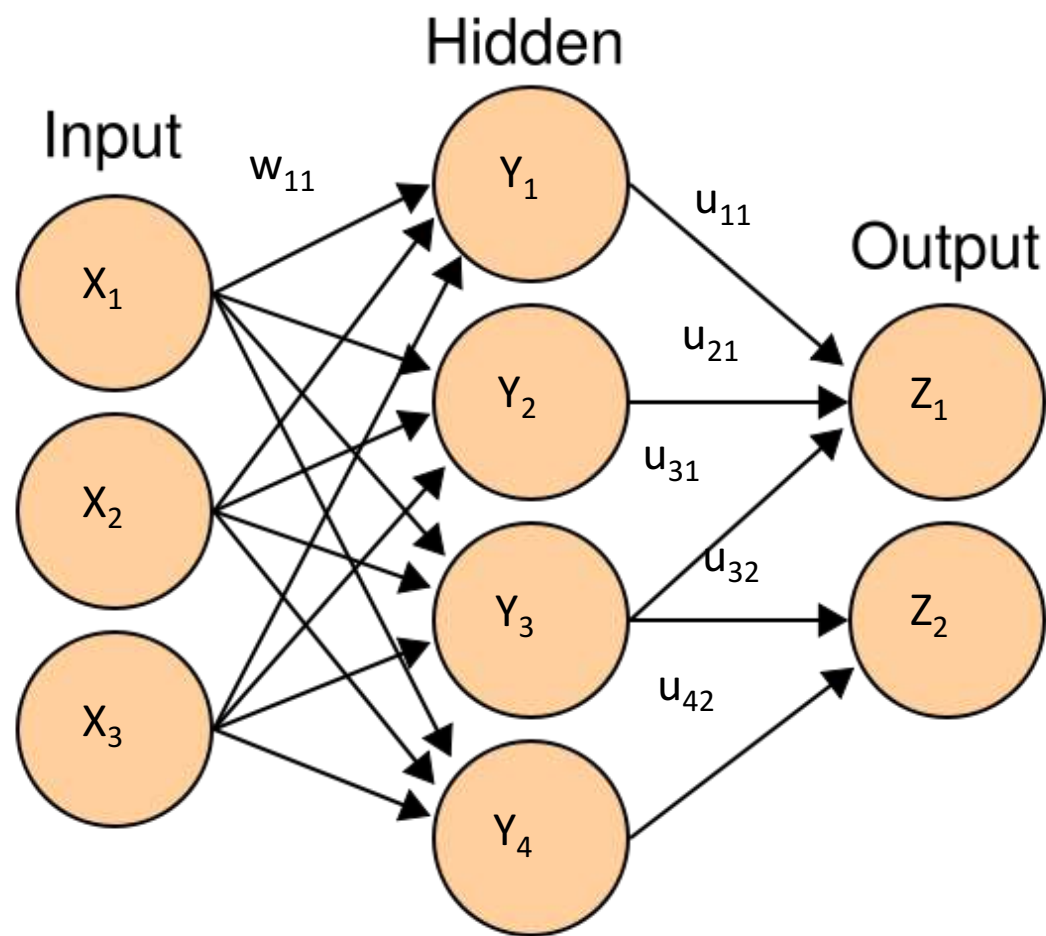
Нейронные сети и CNTK

# Самое важное про нейронные сети

- В последние годы нейросети считают синонимом ИИ
- Всё, что вы видели в когнитивных сервисах – нейронные сети
- Нейросеть – это не магия, а всего лишь способ оптимизации функций
- Часто работа нейросети выглядит как магия
- Для серьезных экспериментов нужны большие данные и вычислительные ресурсы => **облако**



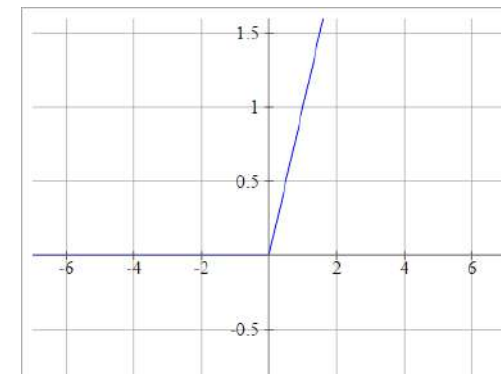
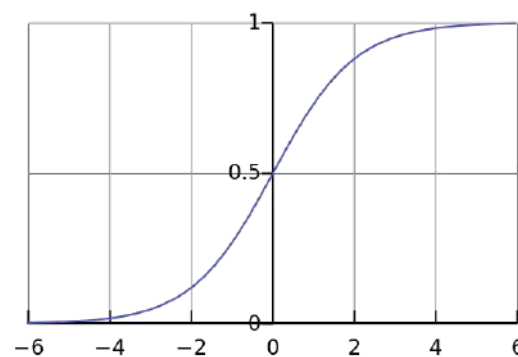
# Нейронные сети



$$Y_k = f\left(\sum_{i=1}^N w_{ik}X_i + b_k\right)$$

$$Y = f(WX + B)$$

$$Z = f(UY + C)$$



# Реализация нейронной сети

## Вручную

Надо явно программировать алгоритм обратного распространения

Для сложных сетей хорошо бы поддерживать вычисления на GPU/кластерах

## Использование фреймворков

Задаётся только «прямая» формула для вычисления сети

Обратное распространение и обучение производится автоматически

Поддерживаются различные среды вычислений

## Наиболее популярные фреймворки

TensorFlow (Google), Cognitive Toolkit / CNTK (Microsoft), Caffe, Torch, Theano

# Microsoft Cognitive Toolkit / CNTK

Инструментарий для обучения и использования нейронных сетей, в т.ч. глубокого обучения

Может использовать

CPU, GPU

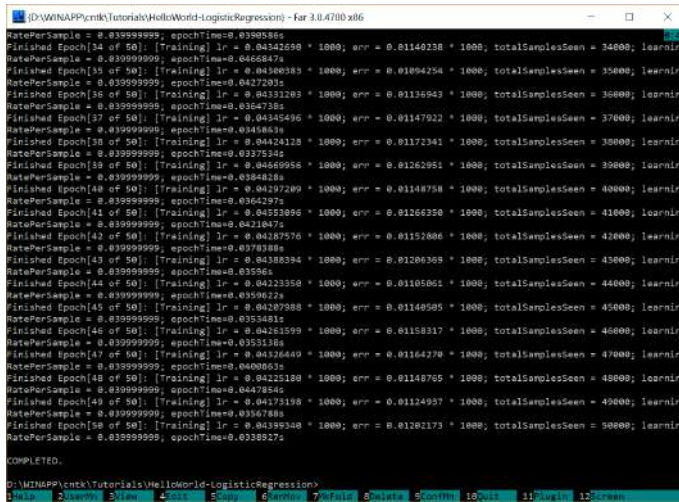
Несколько CPU

Несколько GPU на одном компьютере

Несколько GPU на нескольких компьютерах

<http://cntk.ai>, <http://github.com/microsoft/cntk>

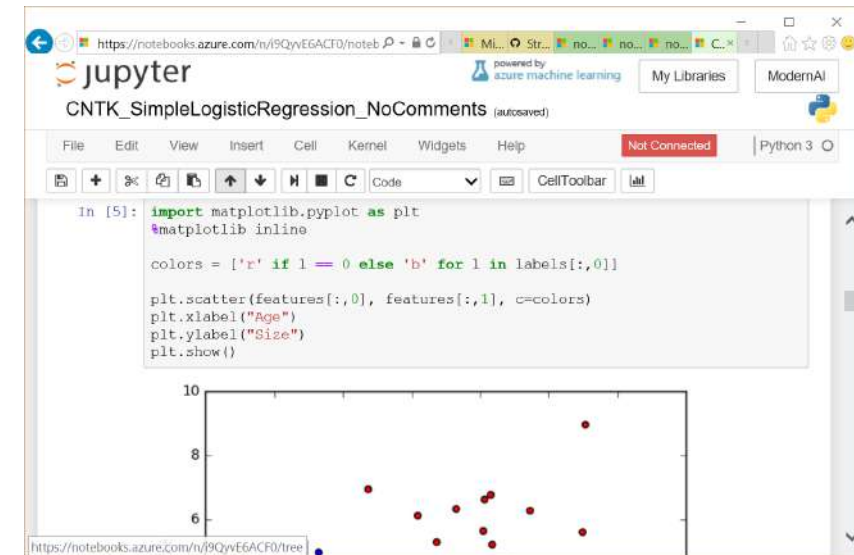
# Microsoft Cognitive Toolkit CNTK



```
(D:\WINAPP\cntk\tutorials\HelloWorld-LogisticRegression) - Far 3.0.4700 x64
RatePerSample = 0.039999999, epochTime=0.03905885
Finished Epoch[34 of 50]: [Training] lr = 0.04342690 * 1000; err = 0.01140238 * 1000; totalSamplesSeen = 34000; learning
RatePerSample = 0.039999999, epochTime=0.04068479
Finished Epoch[35 of 50]: [Training] lr = 0.04340385 * 1000; err = 0.01094254 * 1000; totalSamplesSeen = 35000; learning
RatePerSample = 0.039999999, epochTime=0.04272031
Finished Epoch[36 of 50]: [Training] lr = 0.04331203 * 1000; err = 0.01136943 * 1000; totalSamplesSeen = 36000; learning
RatePerSample = 0.039999999, epochTime=0.03647285
Finished Epoch[37 of 50]: [Training] lr = 0.04345496 * 1000; err = 0.01147922 * 1000; totalSamplesSeen = 37000; learning
RatePerSample = 0.039999999, epochTime=0.03458634
Finished Epoch[38 of 50]: [Training] lr = 0.04424128 * 1000; err = 0.01172341 * 1000; totalSamplesSeen = 38000; learning
RatePerSample = 0.039999999, epochTime=0.03775345
Finished Epoch[39 of 50]: [Training] lr = 0.04469956 * 1000; err = 0.01262951 * 1000; totalSamplesSeen = 39000; learning
RatePerSample = 0.039999999, epochTime=0.03648284
Finished Epoch[40 of 50]: [Training] lr = 0.04397200 * 1000; err = 0.01148758 * 1000; totalSamplesSeen = 40000; learning
RatePerSample = 0.039999999, epochTime=0.03642979
Finished Epoch[41 of 50]: [Training] lr = 0.04353096 * 1000; err = 0.01266350 * 1000; totalSamplesSeen = 41000; learning
RatePerSample = 0.039999999, epochTime=0.04210474
Finished Epoch[42 of 50]: [Training] lr = 0.04187576 * 1000; err = 0.0152806 * 1000; totalSamplesSeen = 42000; learning
RatePerSample = 0.039999999, epochTime=0.03783884
Finished Epoch[43 of 50]: [Training] lr = 0.04388394 * 1000; err = 0.01206309 * 1000; totalSamplesSeen = 43000; learning
RatePerSample = 0.039999999, epochTime=0.035965
Finished Epoch[44 of 50]: [Training] lr = 0.04223350 * 1000; err = 0.01105061 * 1000; totalSamplesSeen = 44000; learning
RatePerSample = 0.039999999, epochTime=0.03506224
Finished Epoch[45 of 50]: [Training] lr = 0.04207988 * 1000; err = 0.01148585 * 1000; totalSamplesSeen = 45000; learning
RatePerSample = 0.039999999, epochTime=0.03534815
Finished Epoch[46 of 50]: [Training] lr = 0.04215199 * 1000; err = 0.0158317 * 1000; totalSamplesSeen = 46000; learning
RatePerSample = 0.039999999, epochTime=0.03511384
Finished Epoch[47 of 50]: [Training] lr = 0.04326449 * 1000; err = 0.01164278 * 1000; totalSamplesSeen = 47000; learning
RatePerSample = 0.039999999, epochTime=0.04008638
Finished Epoch[48 of 50]: [Training] lr = 0.04215199 * 1000; err = 0.01148705 * 1000; totalSamplesSeen = 48000; learning
RatePerSample = 0.039999999, epochTime=0.04478445
Finished Epoch[49 of 50]: [Training] lr = 0.04175198 * 1000; err = 0.01124957 * 1000; totalSamplesSeen = 49000; learning
RatePerSample = 0.039999999, epochTime=0.03647888
Finished Epoch[50 of 50]: [Training] lr = 0.04398348 * 1000; err = 0.01202173 * 1000; totalSamplesSeen = 50000; learning
RatePerSample = 0.039999999, epochTime=0.03389271
COMPLETED.
D:\WINAPP\cntk\tutorials\HelloWorld-LogisticRegression>
```

- Утилита для тренировки сетей cntk.exe
- Библиотека для использования (C#, C++)
- Язык описания сетей BrainScript
- Надо устанавливать CNTK на ПК (с GPU или без)
- Спец. входной формат для данных

- Интерфейс с Python для обучения и использования сетей
- Конфигурация сети описывается программой на Python
- Можно использовать Azure Notebook или Jupyter Notebook на машине с GPU



# Как обучать нейросети

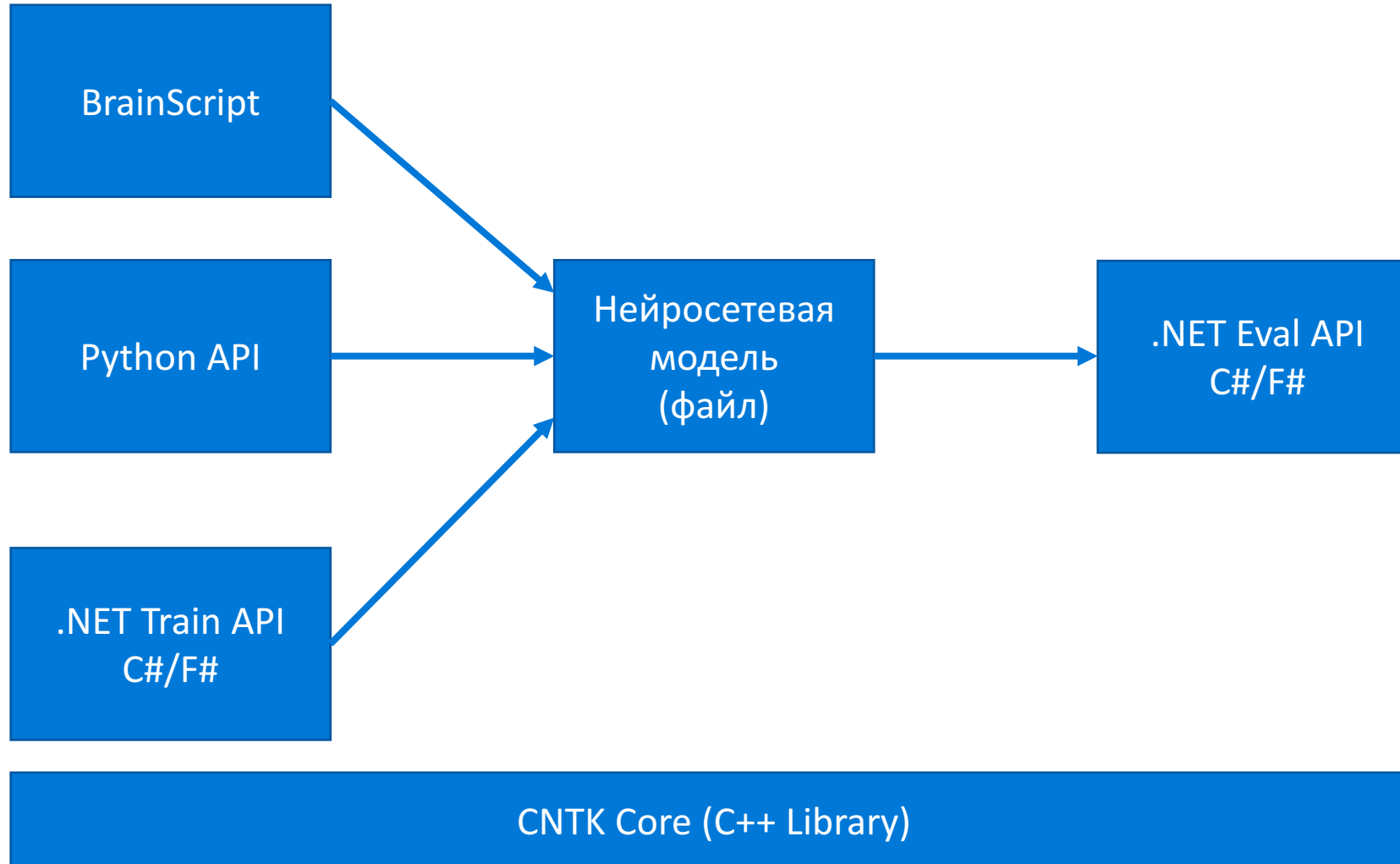
## VM с GPU в Azure

- N-Series VM
  - NC – специально для вычислений (дефицит!)
  - NV – для визуализации
- Готовые конфигурации
  - Data Science Virtual Machine
  - Windows 2016, Linux
  - Deep Learning Virtual Machine
- Доступность в дата-центрах
  - <https://azure.microsoft.com/en-us/pricing/details/virtual-machines/>

## Выделенный ПК

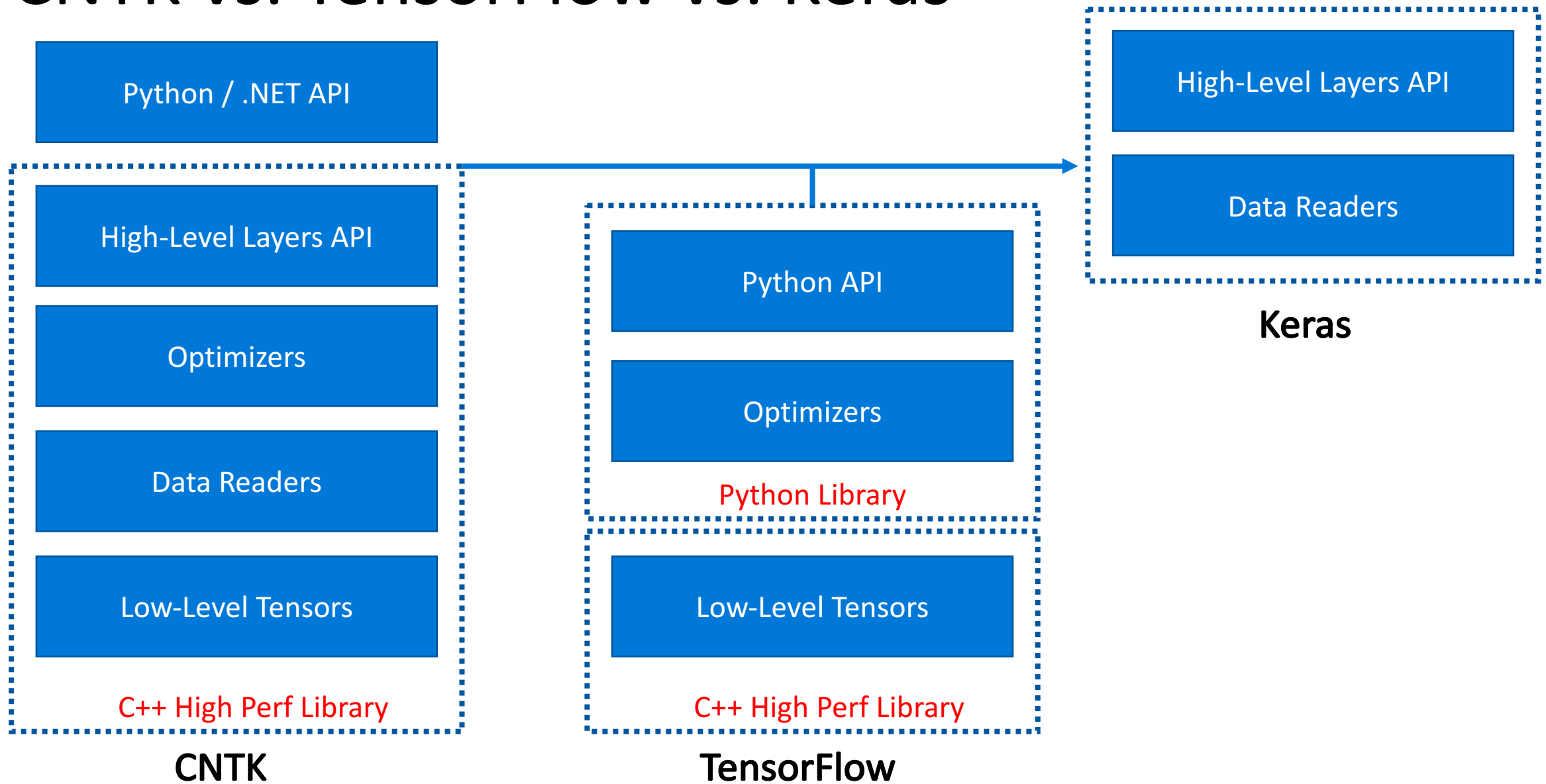
- Nvidia GPU
  - GTX 1070
  - GTX 1080 (Ti)
  - Titan X
- <http://bit.ly/deeplearnbox>

# Использование CNTK в .NET-проектах

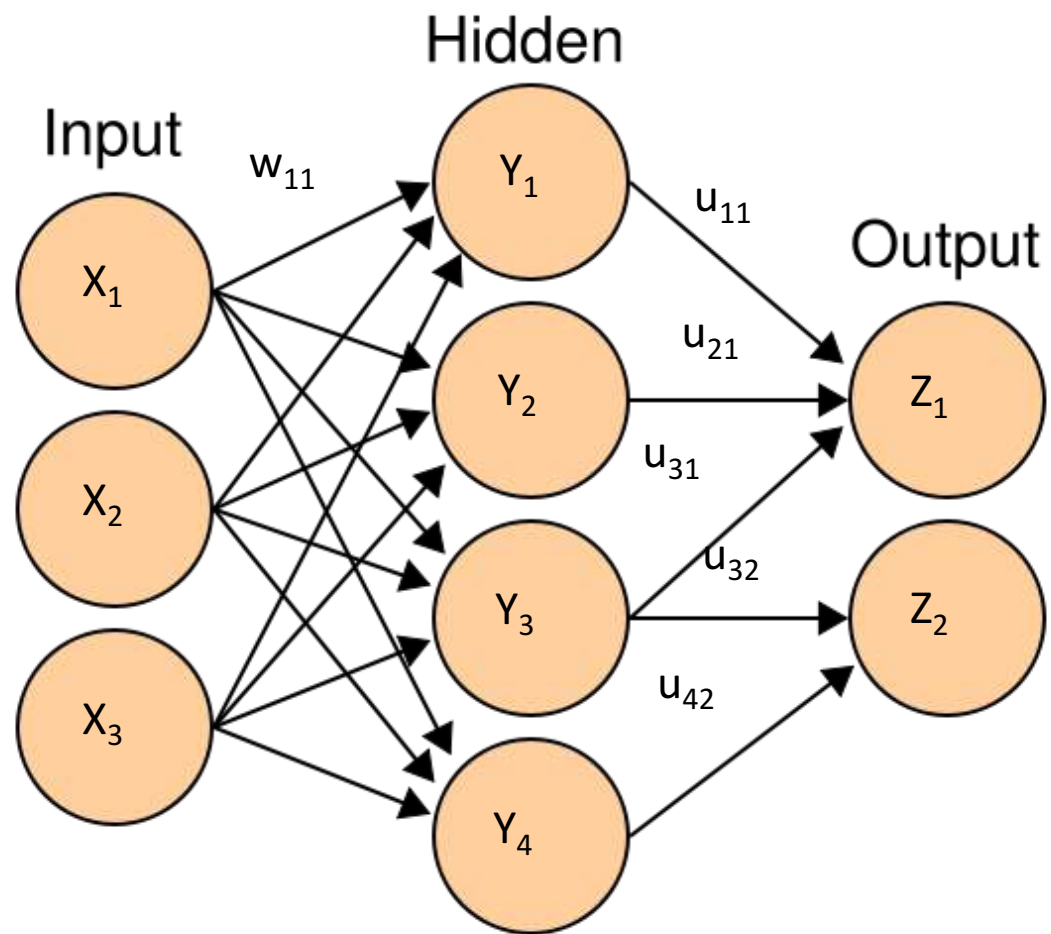




# CNTK vs. TensorFlow vs. Keras



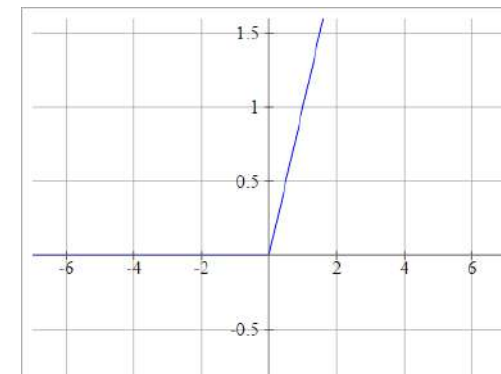
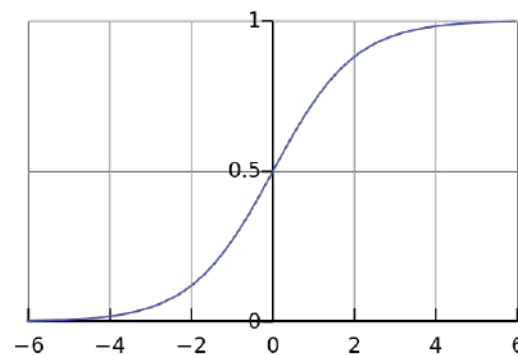
# Нейронные сети



$$Y_k = f\left(\sum_{i=1}^N w_{ik}X_i + b_k\right)$$

$$Y = f(WX + B)$$

$$Z = f(UY + C)$$



# Обучение нейронной сети

## Описание архитектуры сети

```
Variable features = Variable.InputVariable(768,...);  
Variable label = Variable.InputVariable(10,...);  
  
var W = new Parameter({ 10, 768 }, ...);  
var b = new Parameter({ 10 }, ...);  
  
var z = CNTKLib.Times(W,features)+b;
```

## Описание архитектуры сети

```
features = input_variable(input_dim, np.float32)  
label = input_variable(output_dim, np.float32)  
  
W = parameter(shape=(input_dim, output_dim))  
b = parameter(shape=(output_dim))  
  
z = times(features,W)+b
```

## Функции потерь и оптимизатор

```
var loss = CNTKLib.CrossEntropyWithSoftmax(z, label);  
var evalErr = CNTKLib.ClassificationError(z, label);  
var l = new Learner[] {Learner.SGDLearner(z.Parameters)}  
var trainer = Trainer.CreateTrainer(z,loss, evalErr,l);
```

## Функции потерь и оптимизатор

```
loss = cntk.cross_entropy_with_softmax(z, label)  
eval_error = cntk.classification_error(z, label)  
learner = sgd(z.parameters,...)  
trainer = Trainer(z, (loss, eval_error), [learner])
```

## Цикл обучения

```
foreach (epoch) {  
    ft = NextBatchFeatures(); lb = NextBatchLabels();  
    trainer.TrainMinibatch(  
        new Dictionary<Variable, Value>() {  
            { features, ft }, { label, lb } });  
}
```

## Цикл обучения

```
for (epoch in ...)  
    ft = NextBatchFeatures(); lb = NextBatchLabels()  
    trainer.train_minibatch({features: ft},{label:lb})
```

# Демонстрация

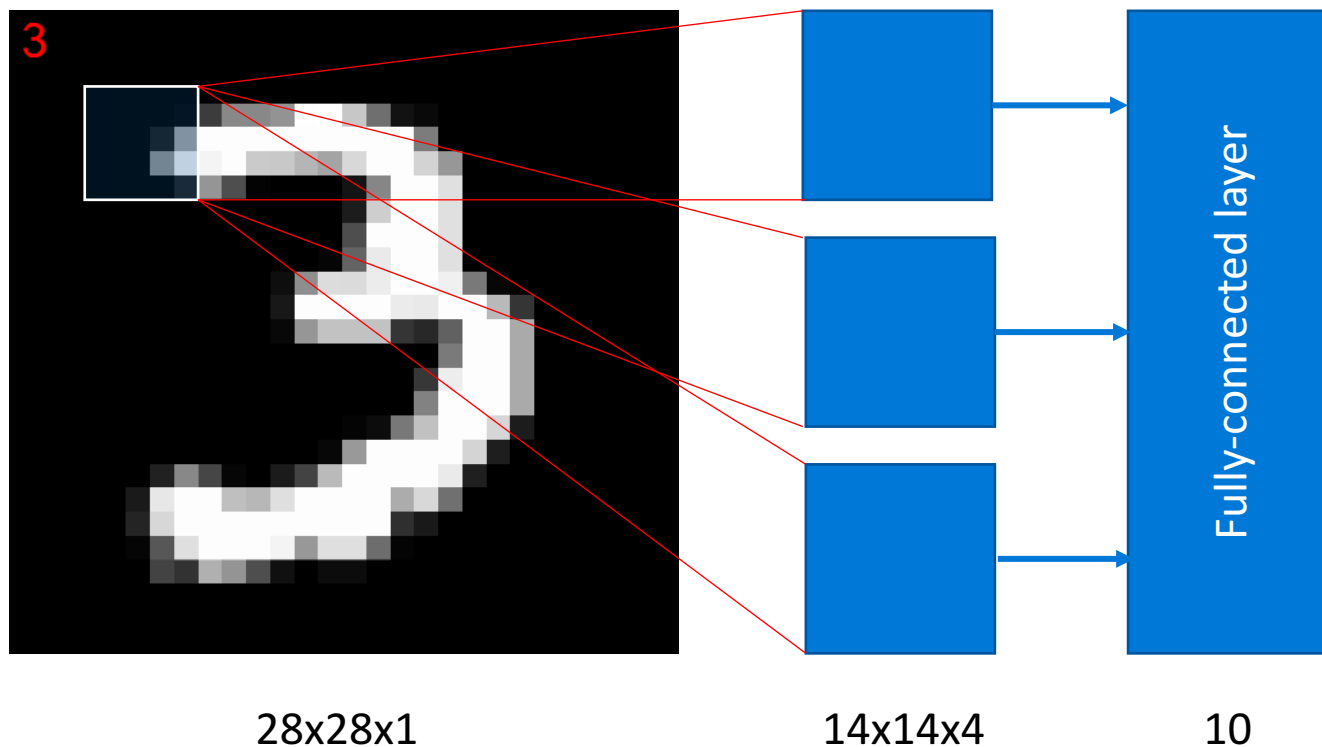
```
Reading data
Creating network
Epoch=0, loss=2,594153881073, eval=0,953125
Epoch=50, loss=2,30613112449646, eval=0,875
Epoch=100, loss=2,28647422790527, eval=0,921875
Epoch=150, loss=2,30462002754211, eval=0,84375
Epoch=200, loss=2,30903220176697, eval=0,90625
Epoch=250, loss=2,29002237319946, eval=0,859375
Epoch=300, loss=2,3040030002594, eval=0,9375
Epoch=350, loss=2,30656504631042, eval=0,921875
Epoch=400, loss=2,30595445632935, eval=0,9375
Epoch=450, loss=2,30992889404297, eval=0,9375
Epoch=500, loss=2,28662514686584, eval=0,921875
Epoch=550, loss=2,24828433990479, eval=0,734375
Epoch=600, loss=1,21208095550537, eval=0,3125
Epoch=650, loss=0,525042057037354, eval=0,1875
Epoch=700, loss=0,339907109737396, eval=0,140625
Epoch=750, loss=0,356711745262146, eval=0,109375
Epoch=800, loss=0,259305834770203, eval=0,03125
```

Алгоритм	Точность
KNN	94%
SVM	92%
Neural Net (1L)	90%
Neural Net (2L)	95,7%

# Основные задачи и архитектуры сетей

Задача	Класс сетей
Обычная классификация / регрессия	Feed-forward / Fully Connected (DNN)
Распознавание изображений	Convolutional / Свёрточная (CNN)
Выделение объектов на изображении	Fast R-CNN (Region CNN)
Анализ последовательностей (речь, текст)	Рекуррентная (RNN)
Машинный перевод	Sequence-to-Sequence, RNN+RNN
Описание изображения	CNN + RNN
Стилизация изображения	Style Transfer

# Свёрточная сеть (CNN)



Алгоритм	Точность
KNN	94%
SVM	92%
Neural Net (1L)	90%
Neural Net (2L)	95,7%
CNN	98%

# Неразрешимая задача





# Кошки против собак



[ 0.3418062 0.65819377]



[ 0.63091594 0.36908403]



[ 0.19271372 0.80728626]



# Что смотреть дальше

## По нейронным сетям

- <http://aka.ms/neuroworkshop>
- <http://github.com/shwars/neuroworkshop>

## F# API для CNTK:

- <https://github.com/mathias-brandewinder/CNTK.FSharp>

1

На .NET есть инструменты, позволяющие использовать машинное обучение и нейронные сети: Accord.NET, CNTK .NET API

2

Для работы с данными полезно использовать F#, Azure Notebooks (F#), C# Interactive, Xamarin Workbook

3

Для обучения нейросетей в облаке Microsoft Azure существуют настроенные виртуалки с GPU



Q&A

# Нейросети и ИИ на платформе .NET

Дмитрий Сошников

<http://fb.com/shwars>

