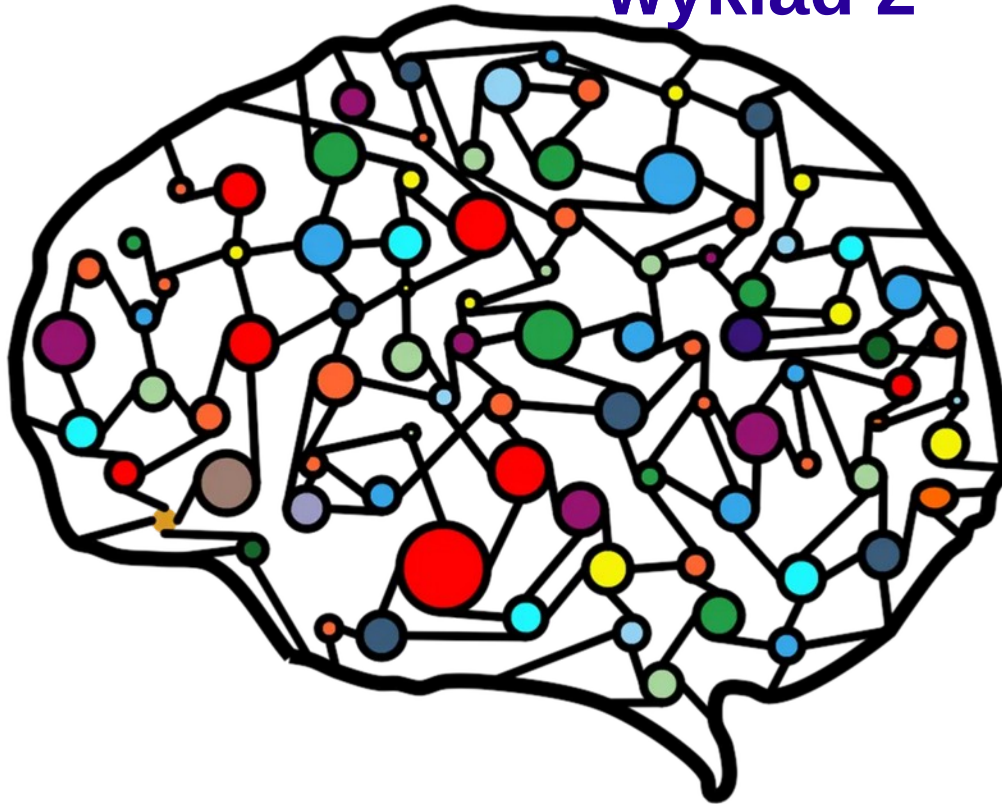
Deep learning
wykład 2



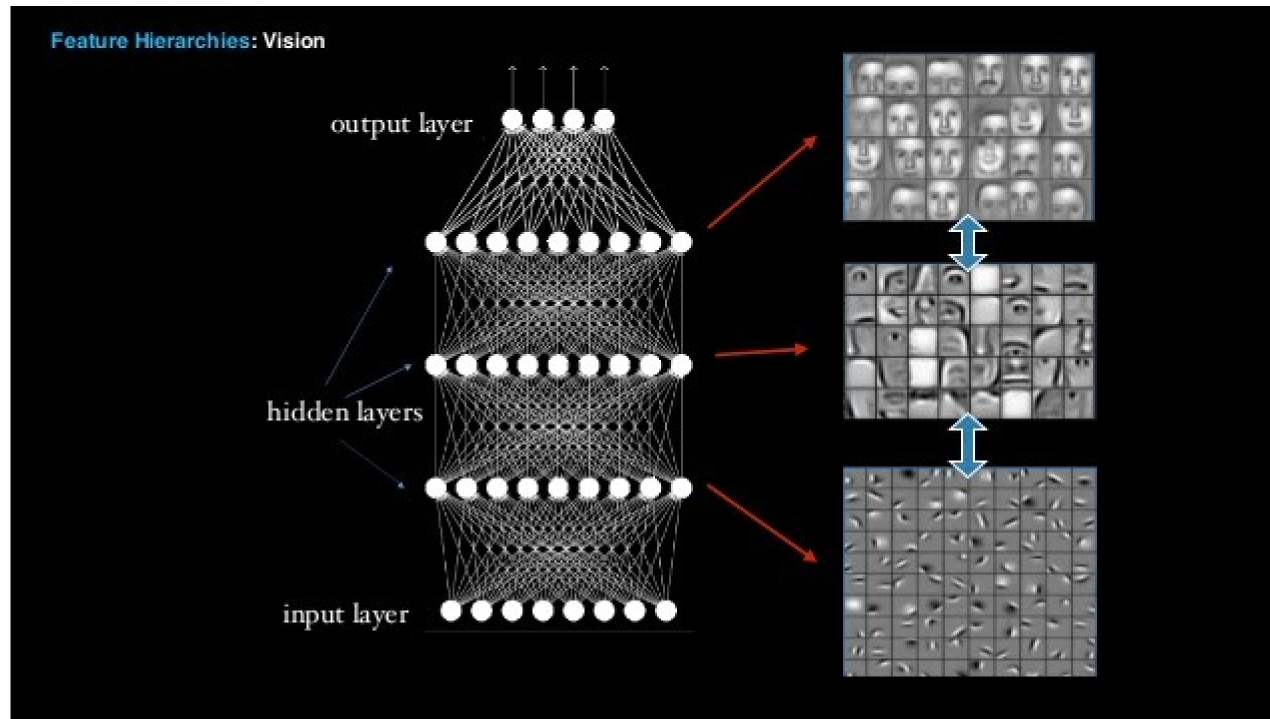
- Głębokie sieci neuronowe
- Keras tutorial – jak zbudować głęboką sieć neuronową

Marcin Wolter

IFJ PAN

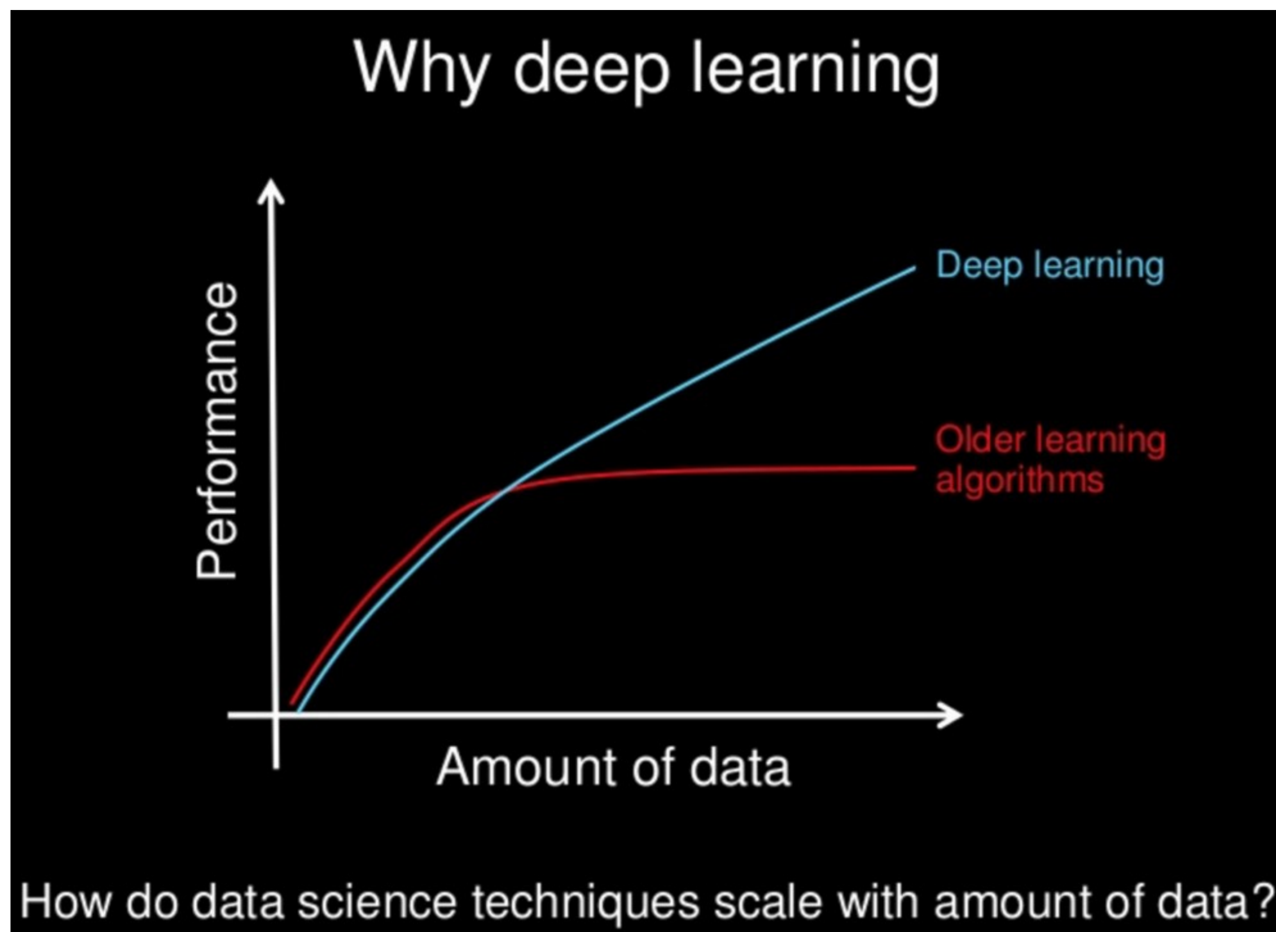
25 listopada 2020

Deep Learning
Głebokie Uczenie



Głębokie uczenie

- Co to znaczy “głębokie uczenie”?
- Dlaczego daje lepsze rezultaty np. przy rozpoznawaniu obrazu, mowy?

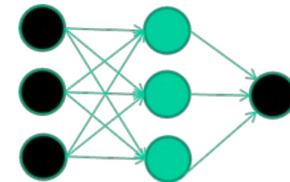


Krótka odpowiedź:

‘Deep Learning’ – oznacza użycie sieci neuronowej z wieloma warstwami ukrytymi

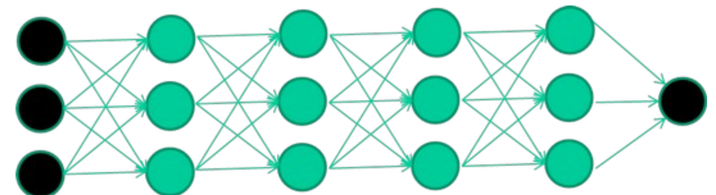
Pierwsze warstwy ukryte identyfikują „cechy” (features): identyfikacja cech → identyfikacja bardziej złożonych cech → selekcja (ostatnie warstwy)

Ale sieci neuronowe są znane od lat 80-tych? Zawsze mieliśmy dobre algorytmy uczenia płytkich sieci. Niestety one zawodziły dla sieci głębszych.



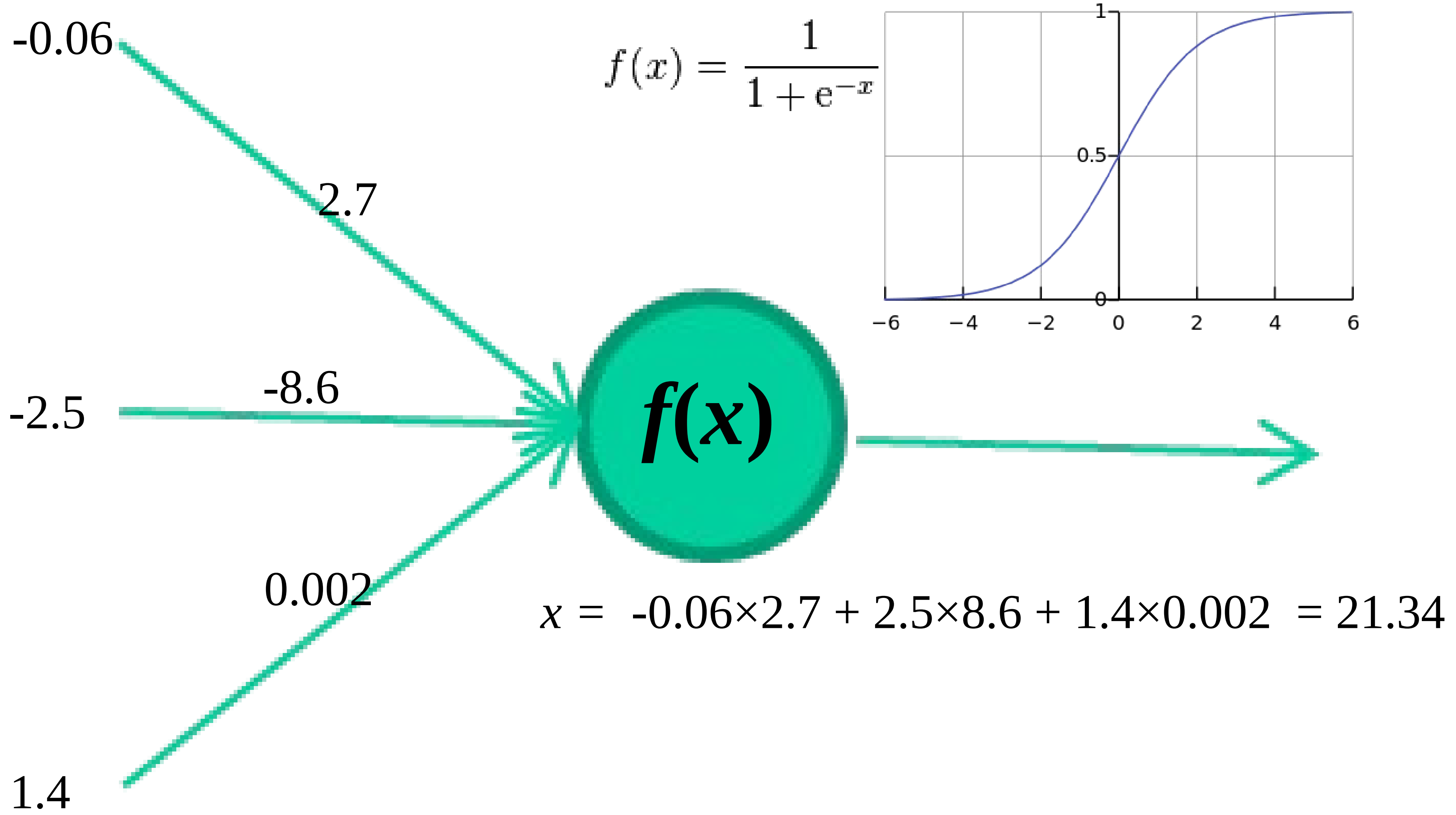
Co mamy nowego: algorytmy uczące

dużo większa moc obliczeniowa



Jak trenujemy sieć neuronową?

$$f(x) = \frac{1}{1+e^{-x}}$$

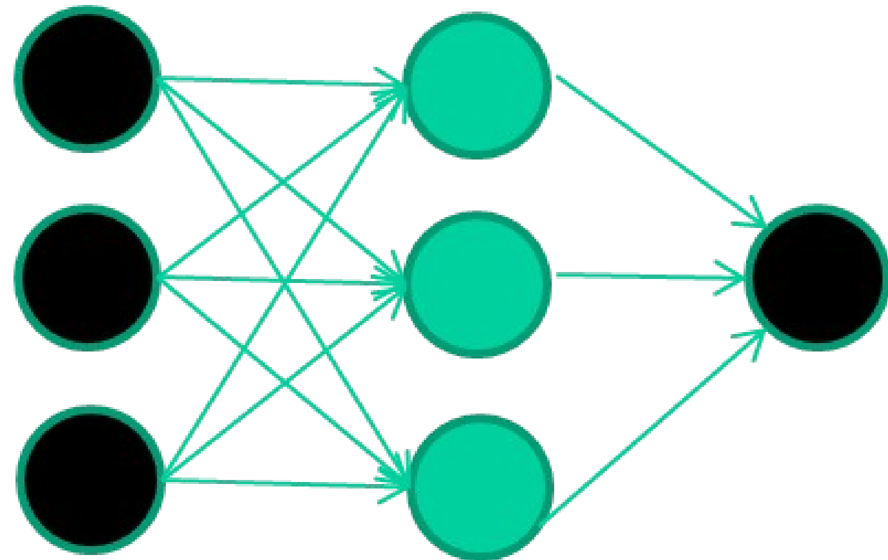


-0.06
2.7
-2.5
-8.6
0.002
1.4
f(x)
f(x) = 1 / (1 + e^-x)
1
0.5
0
−6
−4
−2
0
2
4
6
x = -0.06×2.7 + 2.5×8.6 + 1.4×0.002 = 21.34

Trening sieci

Inputs			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

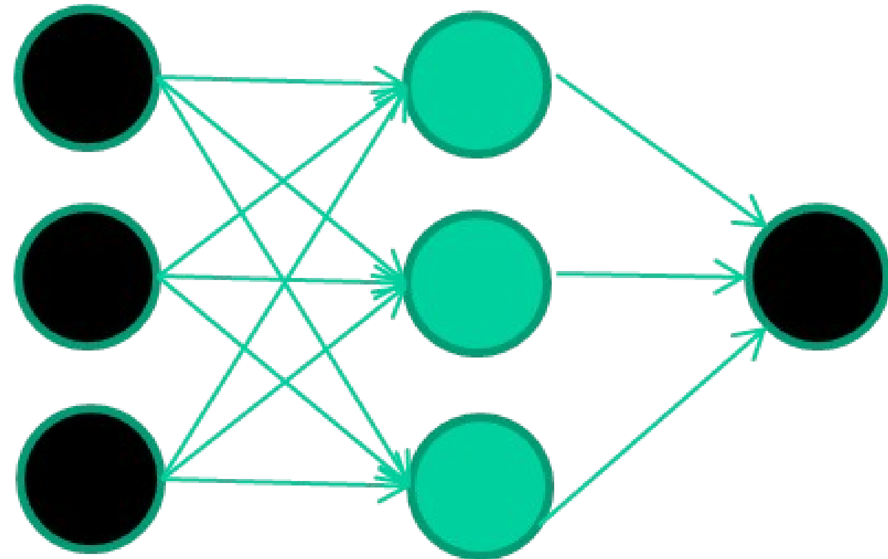
etc …



Trening sieci

Inputs			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

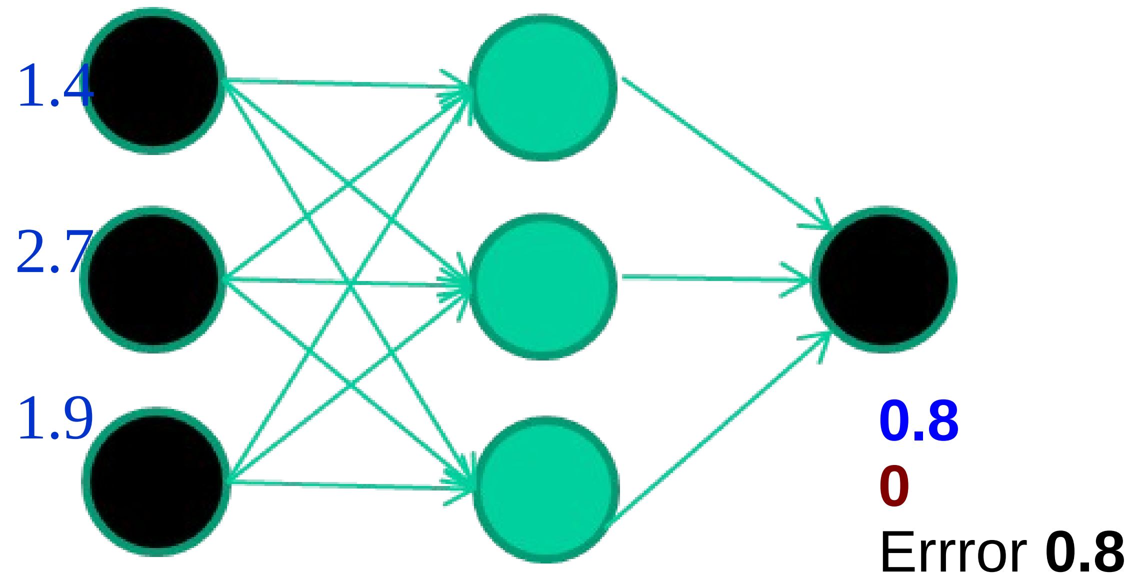
Inicjalizacja przypadkowymi wagami



Trening

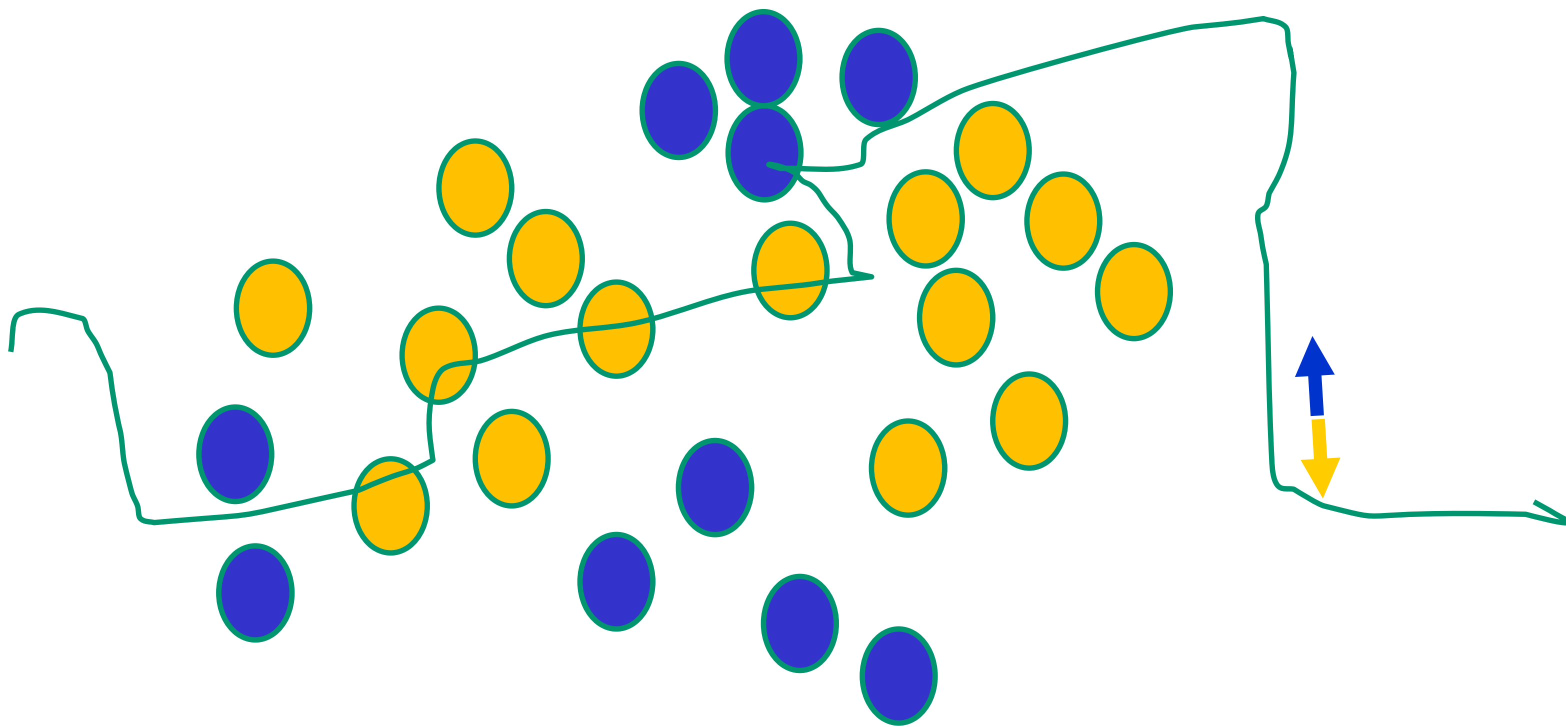
Inputs			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

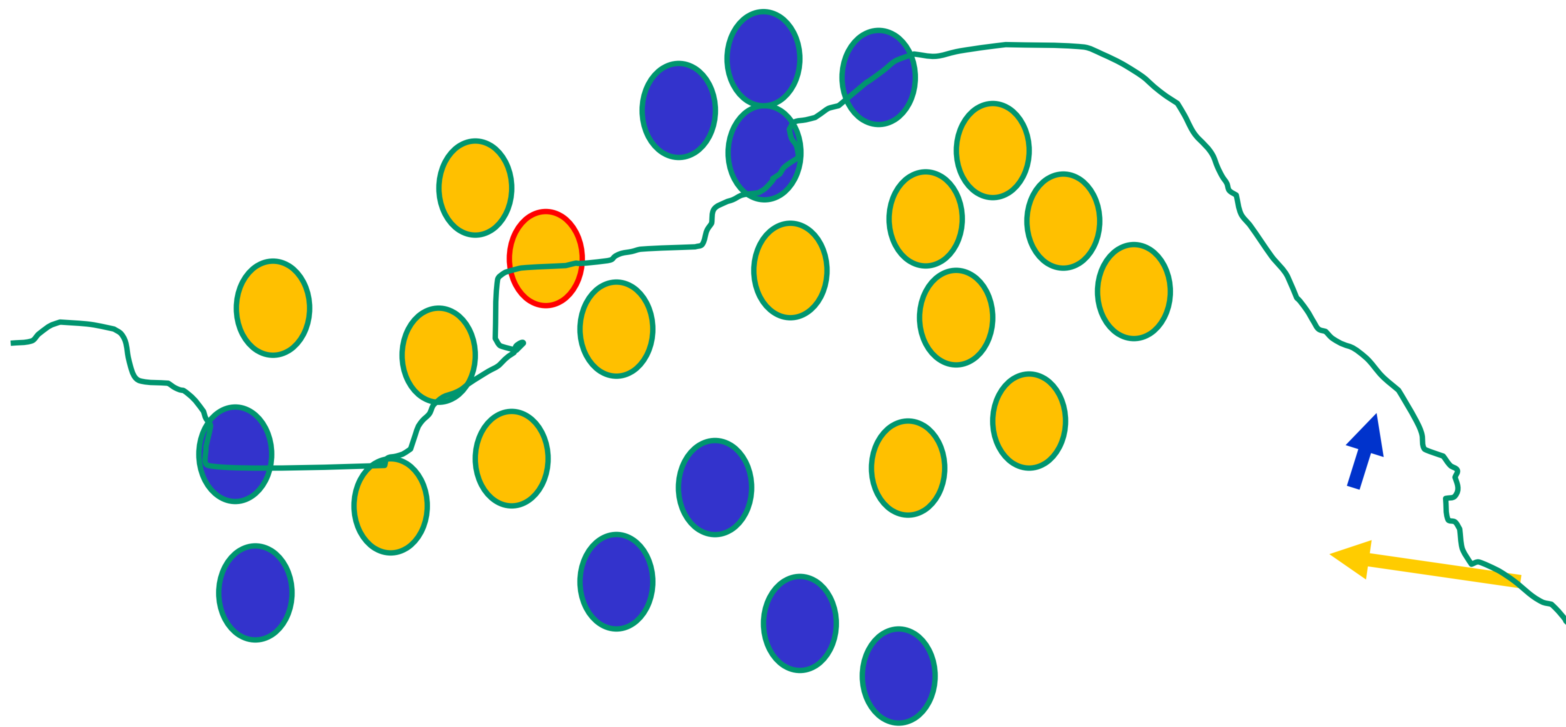
Czytanie danych
Przetwarzanie przez sieć
Wynik porównywany z prawdziwymi wynikami

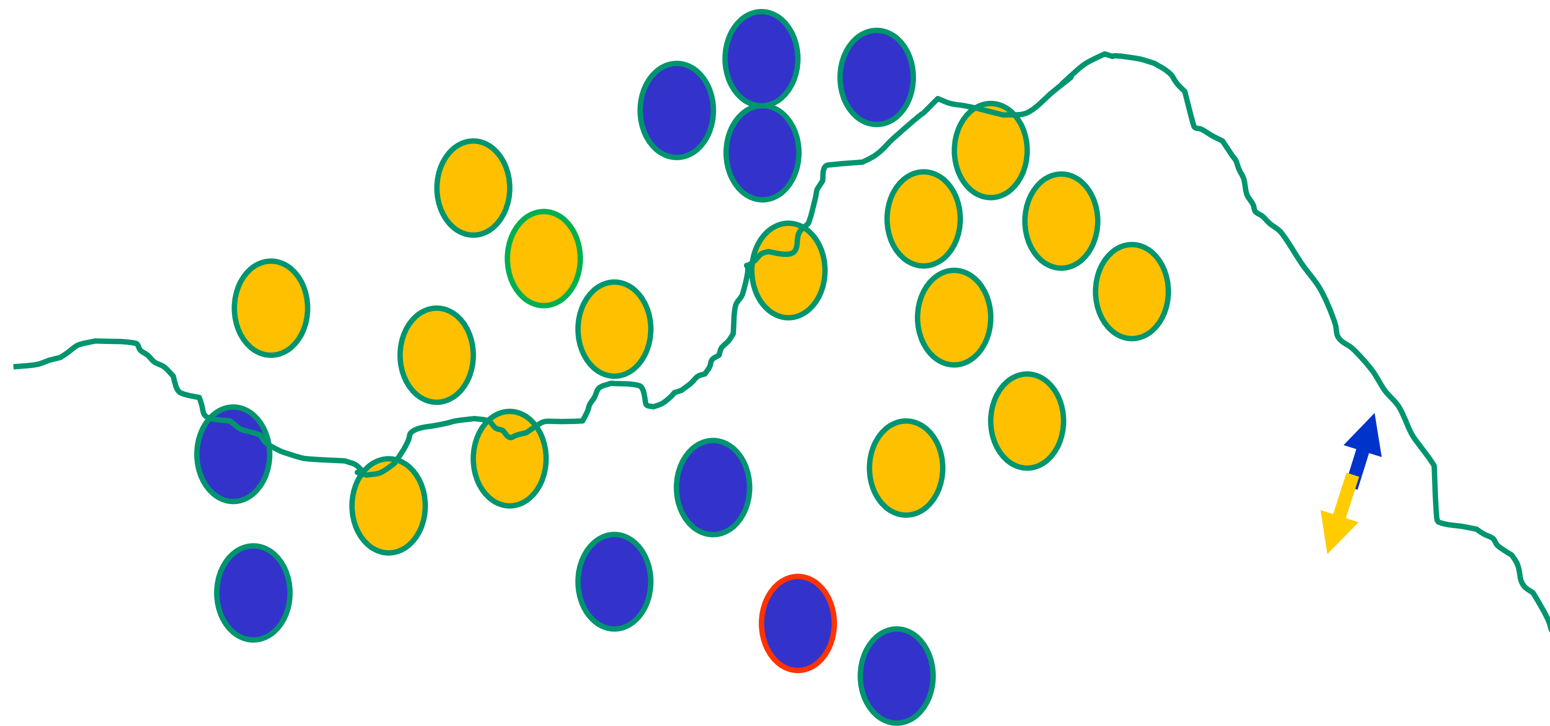


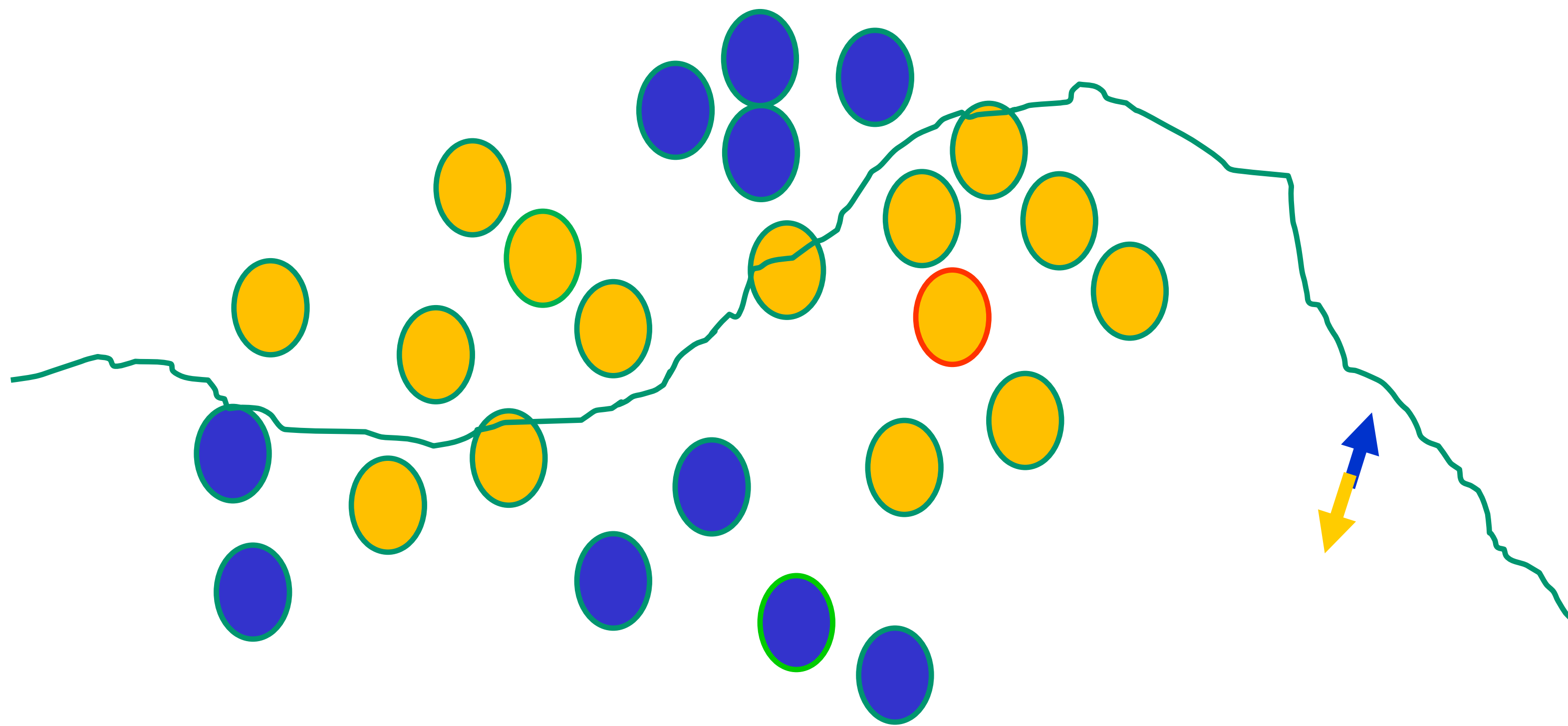
Wagi są modyfikowane (modyfikacja zależy od wartości błędu.

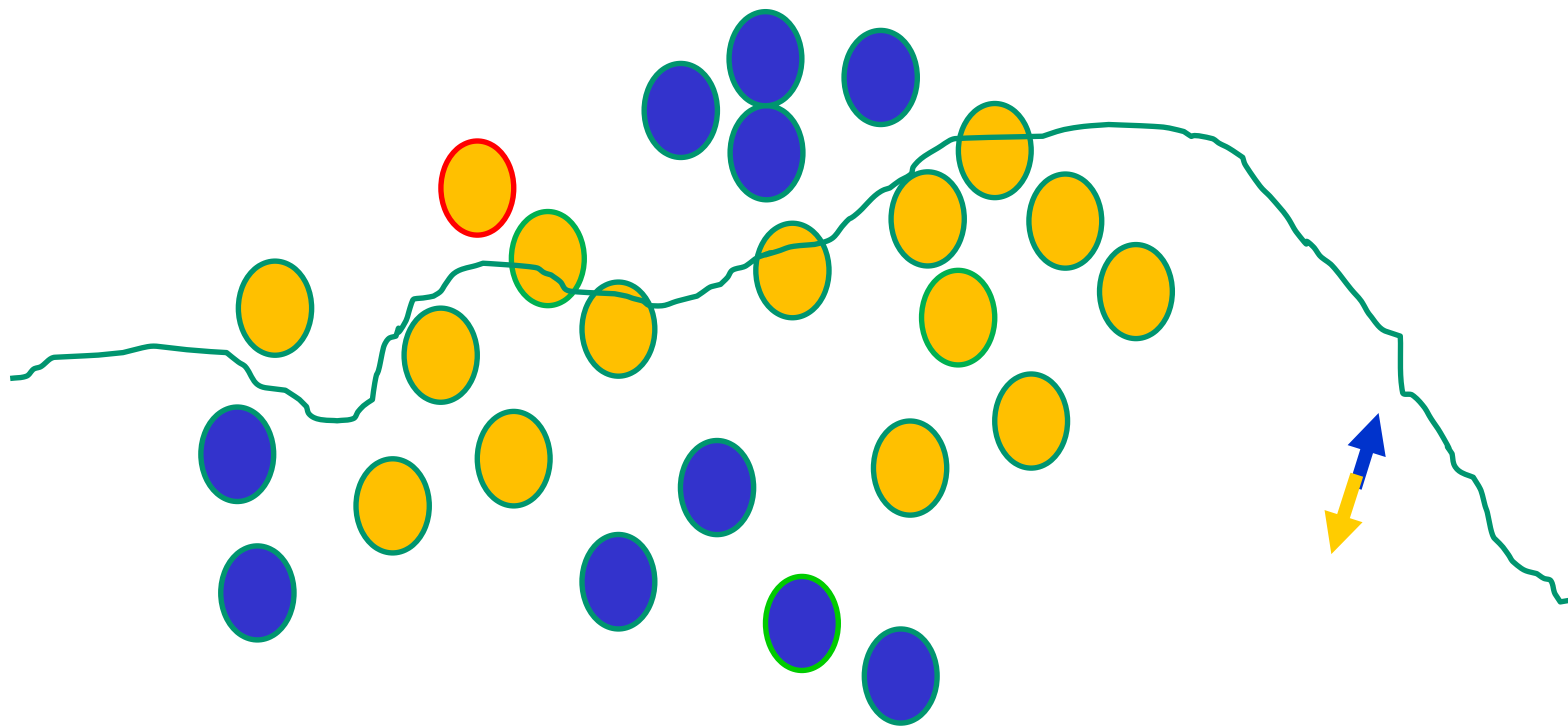
Powtarzamy wiele razy
Algorytmy trenujące troszczą się, aby błędy malały

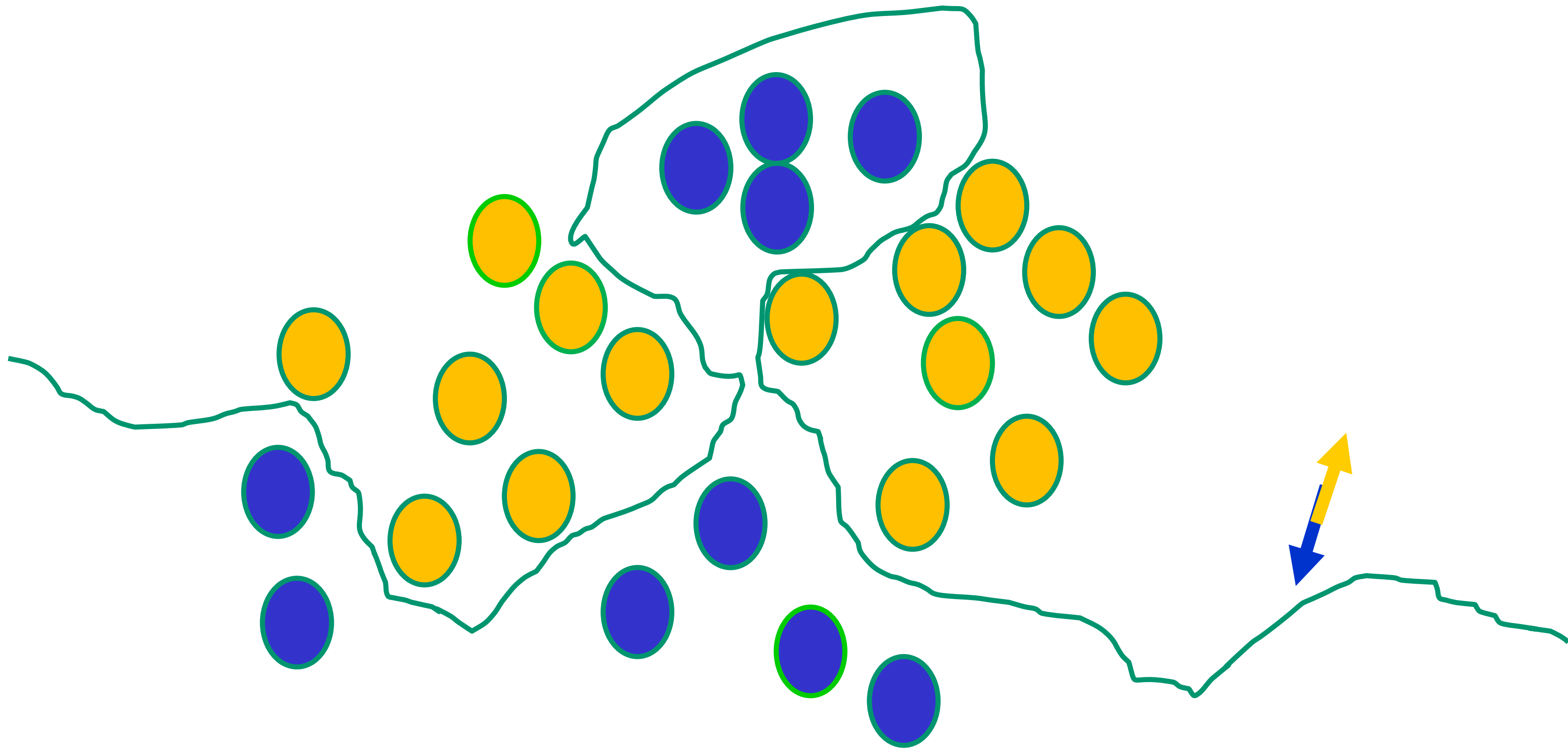












Uwagi

- **Jeśli funkcja aktywacji jest nieliniowa, wtedy sieć neuronowa z jedną warstwą ukrytą może klasyfikować dowolny problem (lub dopasować dowolną funkcję).**
- **Istnieje zestaw wag, który na to pozwala. Problemem jest znalezienie tego zestawu wag...**

Jak identyfikujemy cechy??

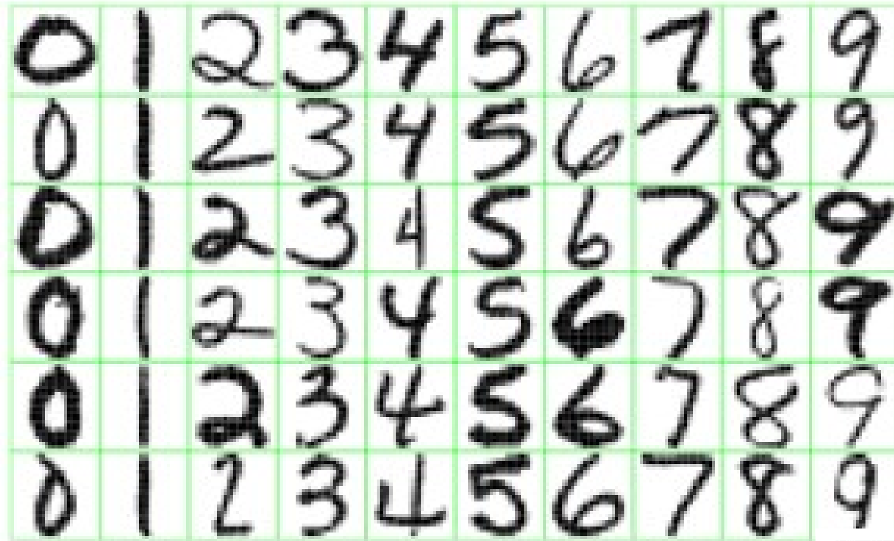
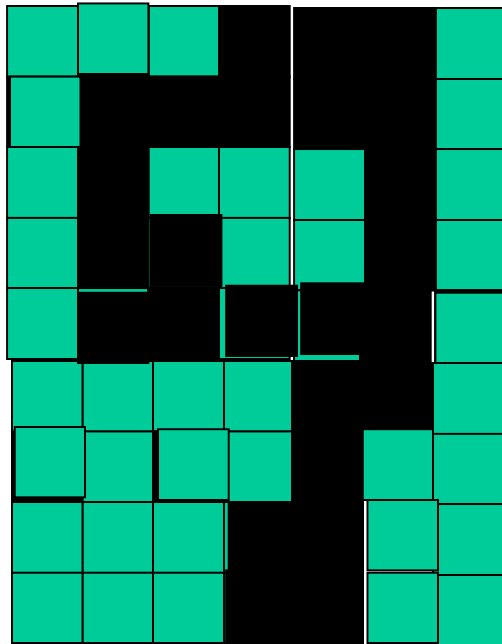
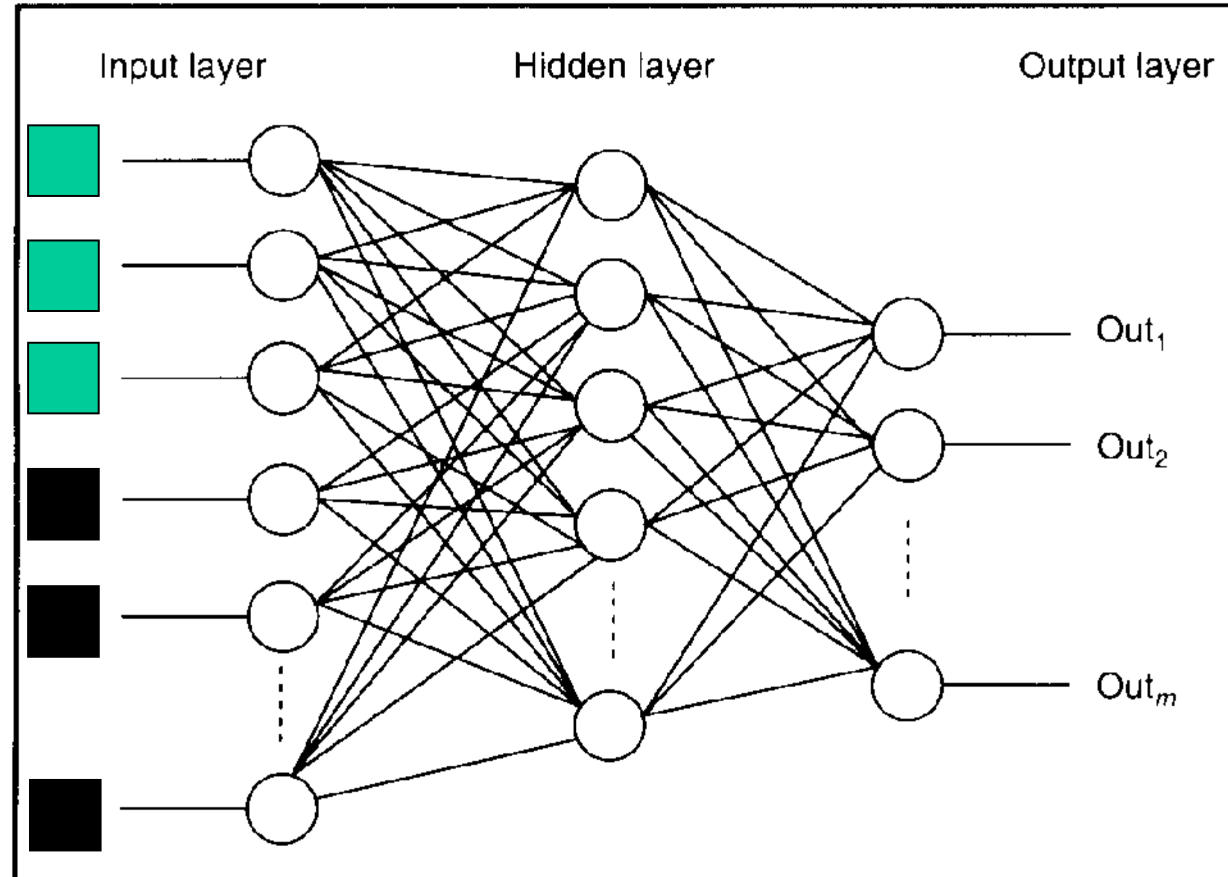


Figure 1.2: *Examples of handwritten digits from postal envelopes.*





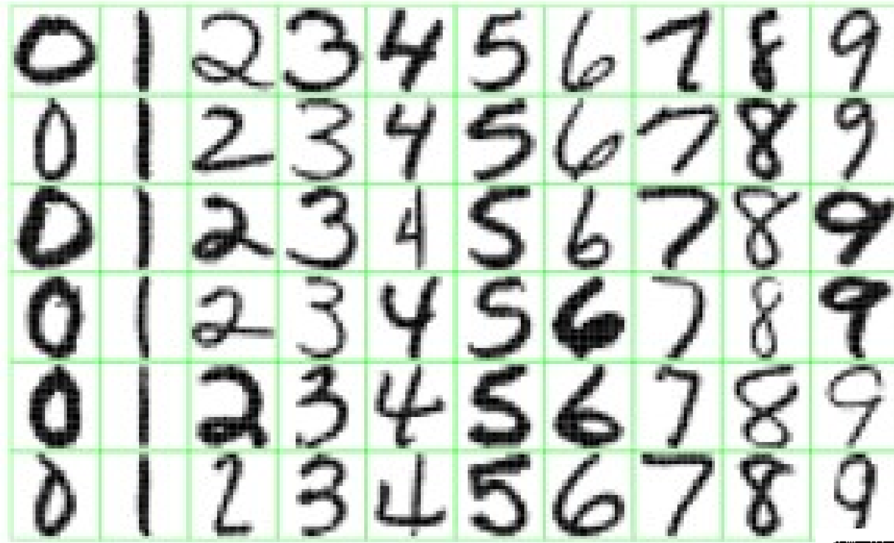
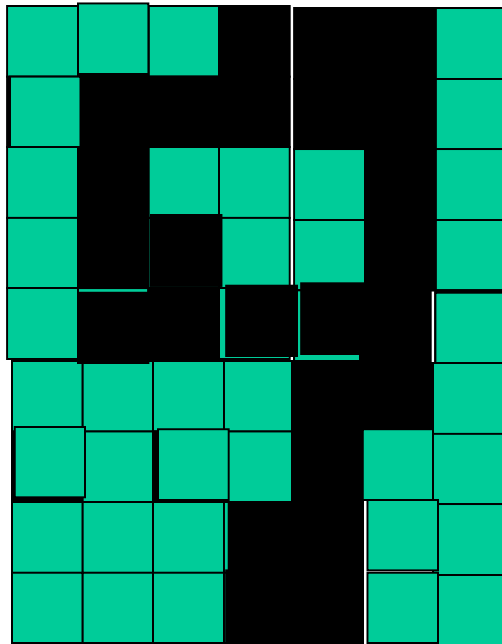
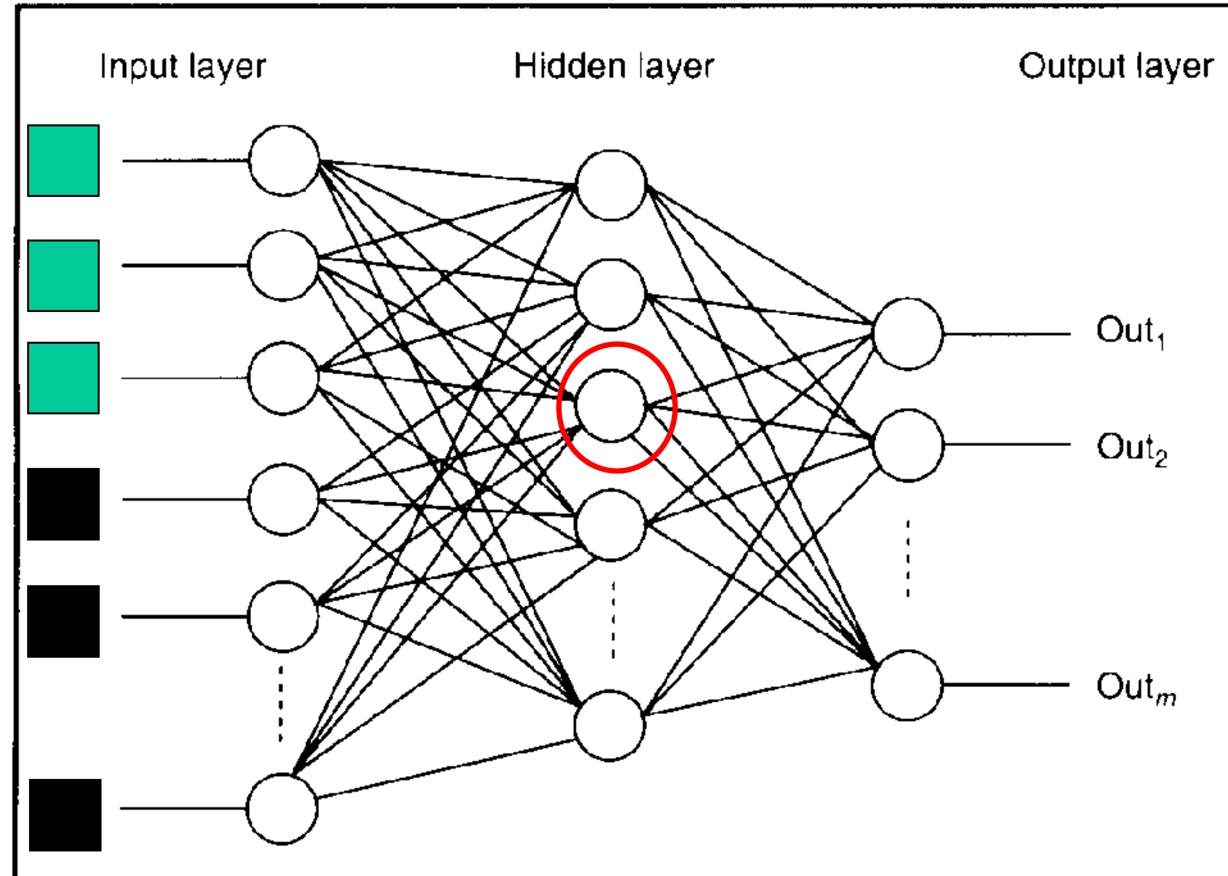


Figure 1.2: *Examples of handwritten digits from postal envelopes.*

Co robi ten neuron?





Neurony w warstwach ukrytych są samoorganizującymi się detektorami cech
(self-organizing feature detectors)

1 5 10 15 20 25 ...

...

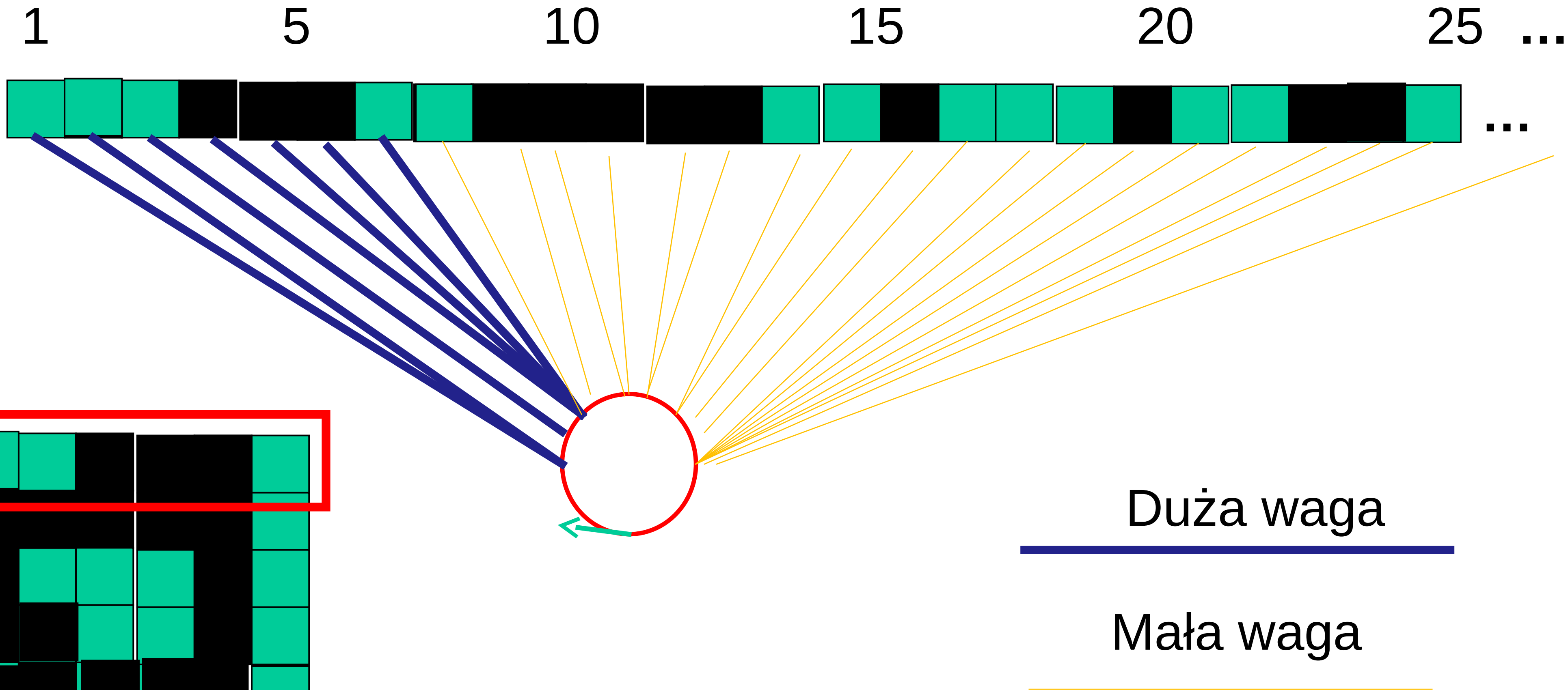
1

63

Duża waga

Mała waga

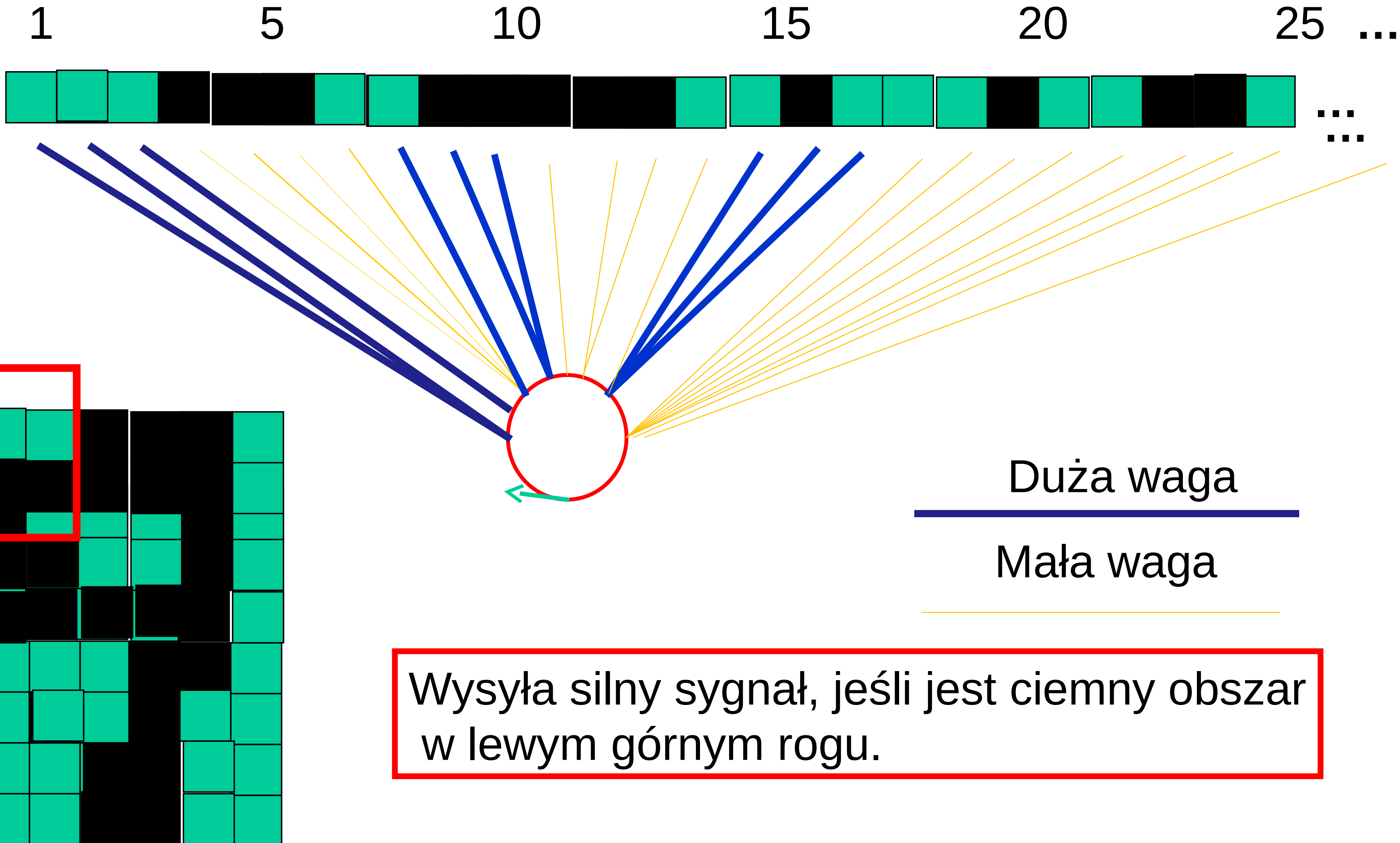
Co może wykryć?



Ten neuron wysyła silny sygnał, kiedy znajdzie Poziomą linię w pierwszym rzędzie pikseli (ignoruje wszystko inne).

63

Co może wykryć?



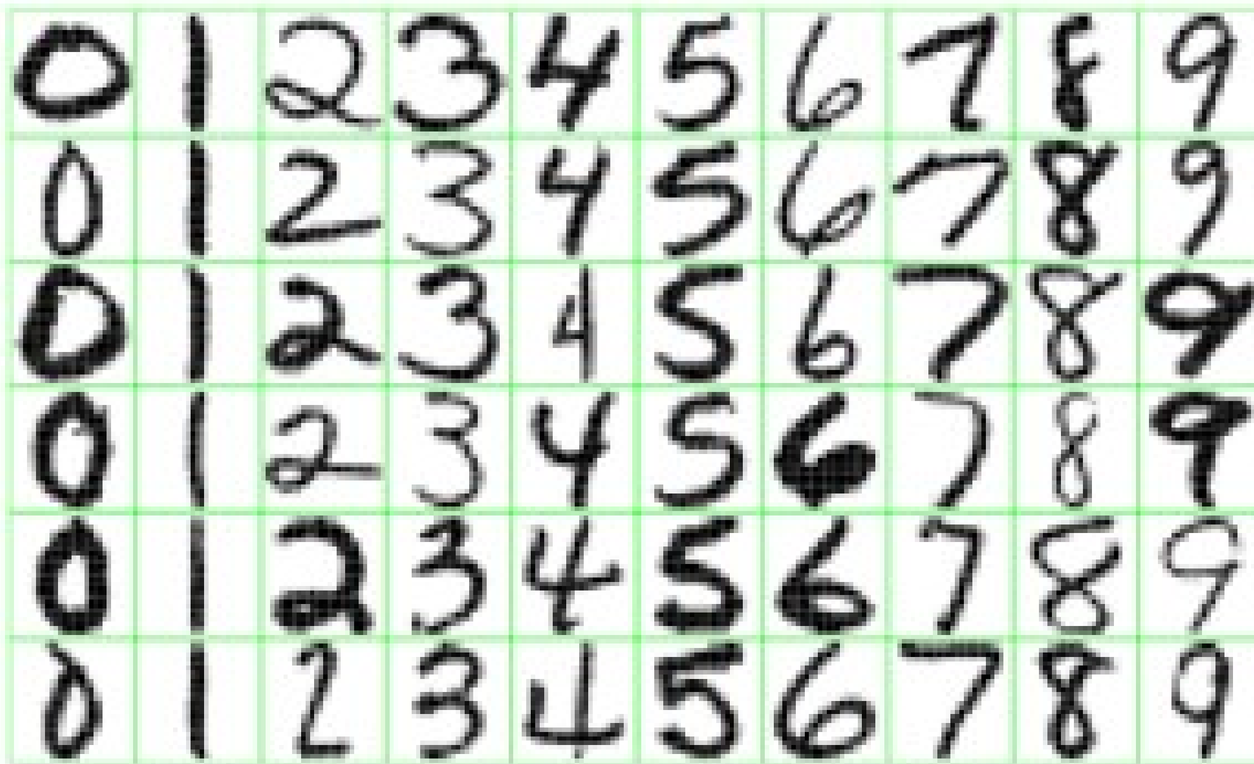


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

Jakie cechy powinna wykrywać sieć klasyfikująca ręcznie pisane cyfry?

Pionowe linie

1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

Poziome linie

1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

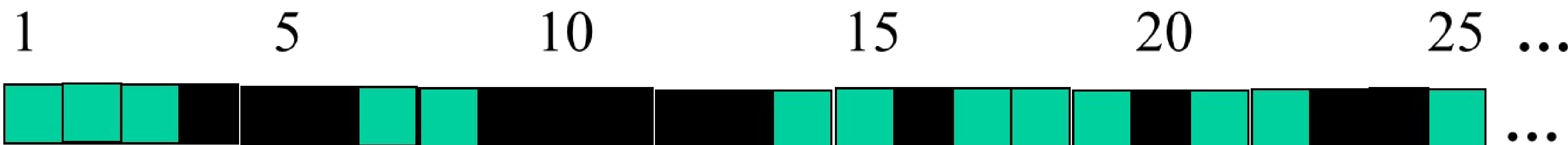
Okręgi

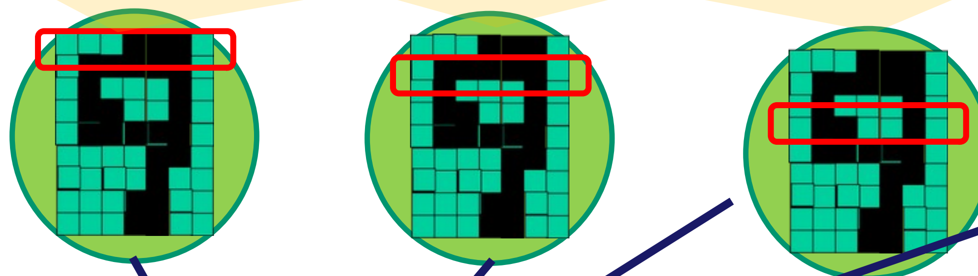
1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

25

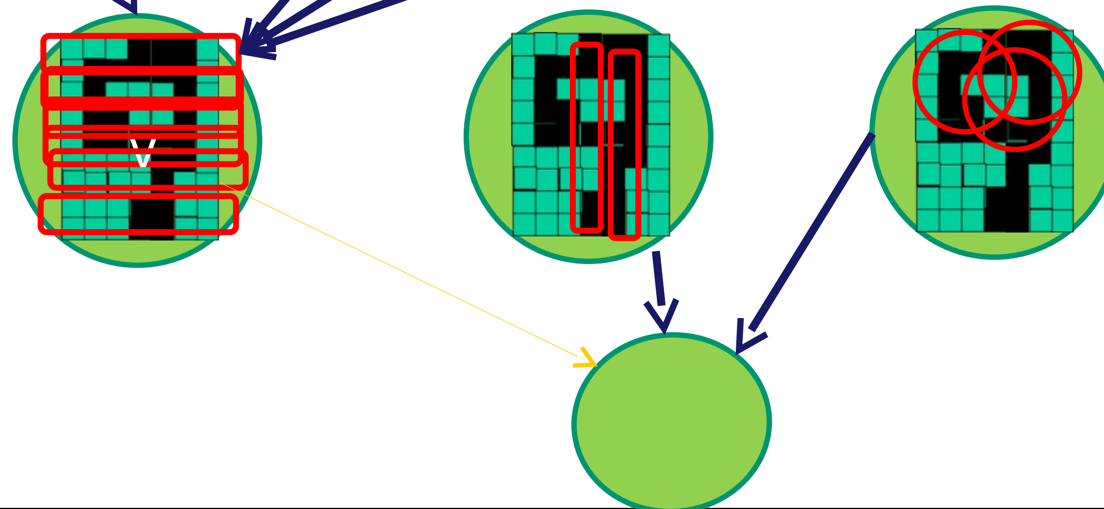
Następne warstwy wykrywają cechy wyższego poziomu





Wykrywają linie
w danych obszarach

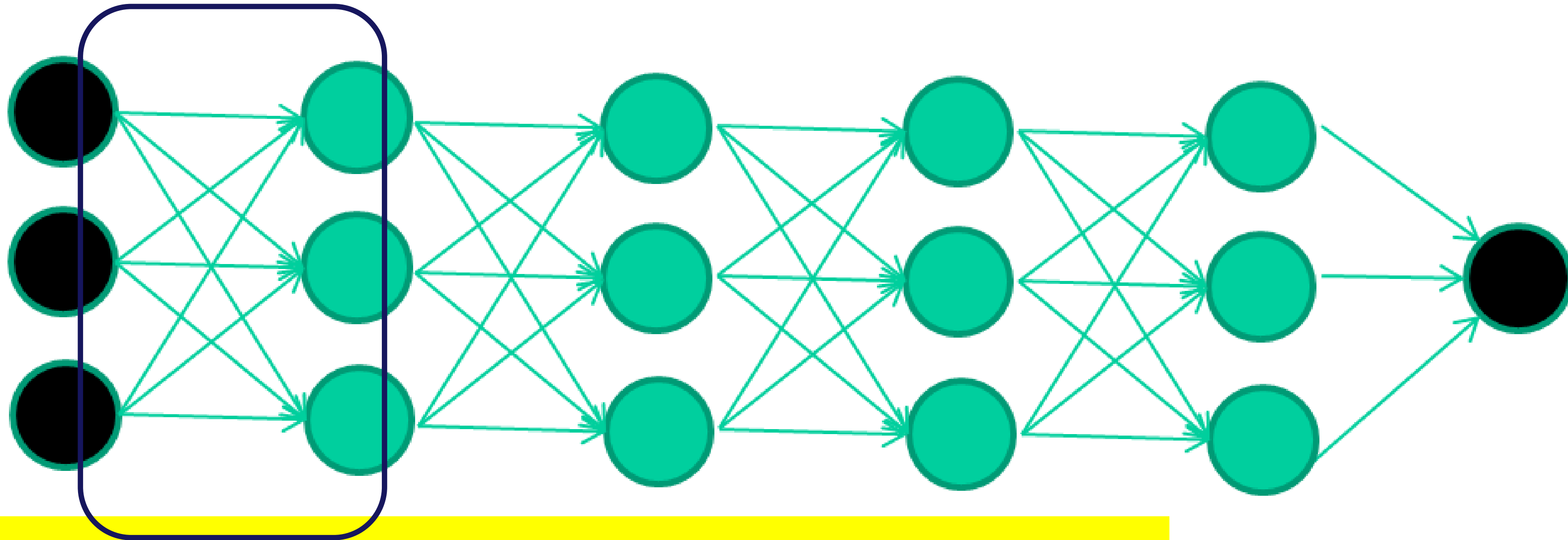
etc ...



Detektory cech
wyższego rzędu

etc ...

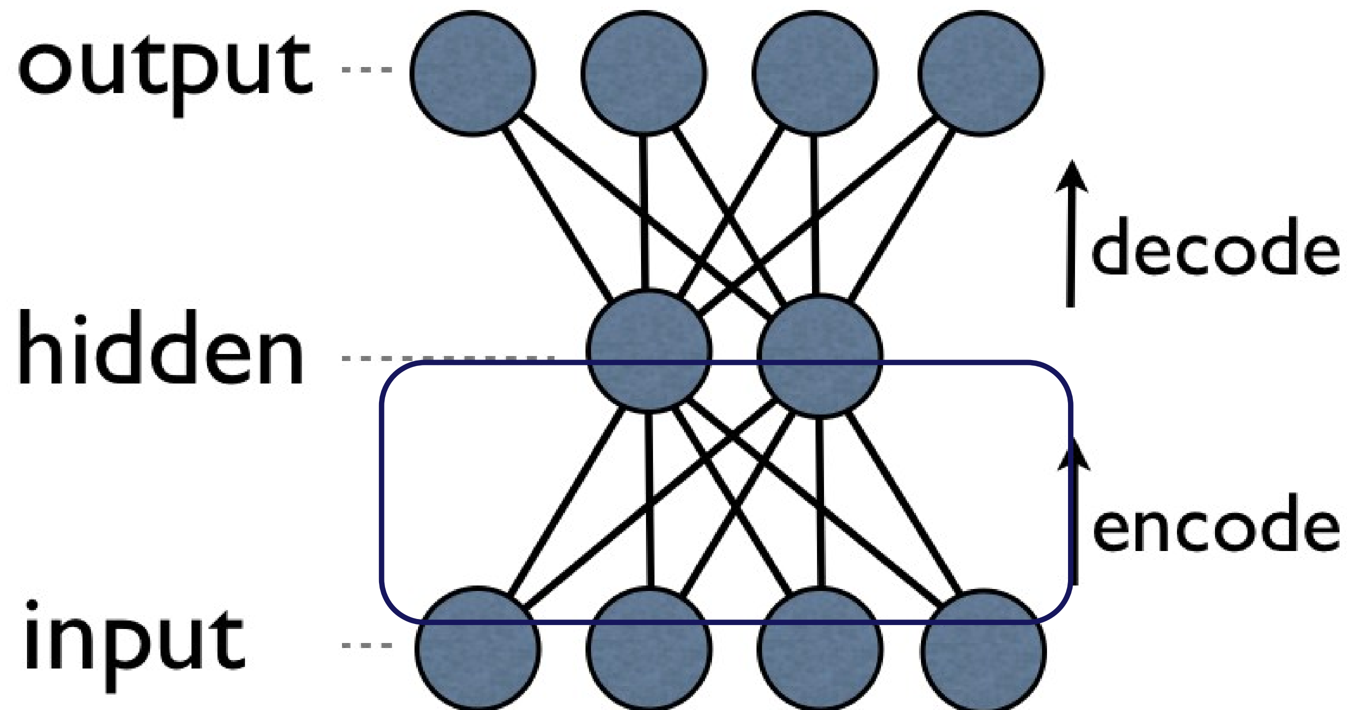
Głęboka sieć



Każda warstwa jest detektorem cech

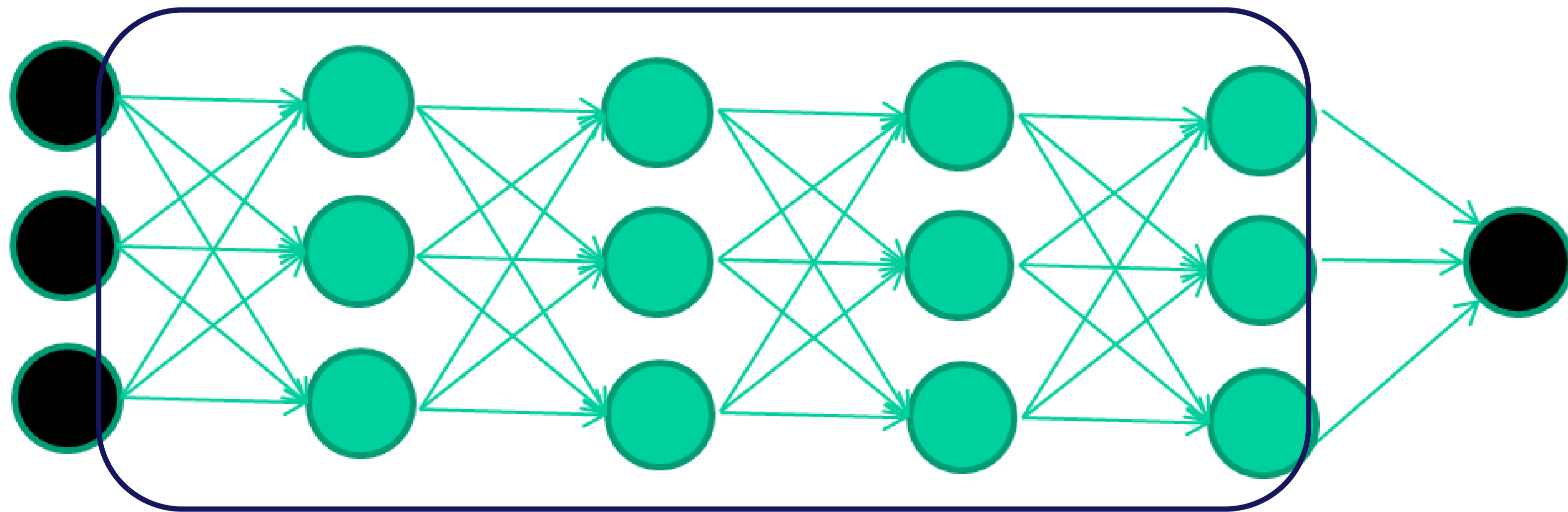
auto-encoder

An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.

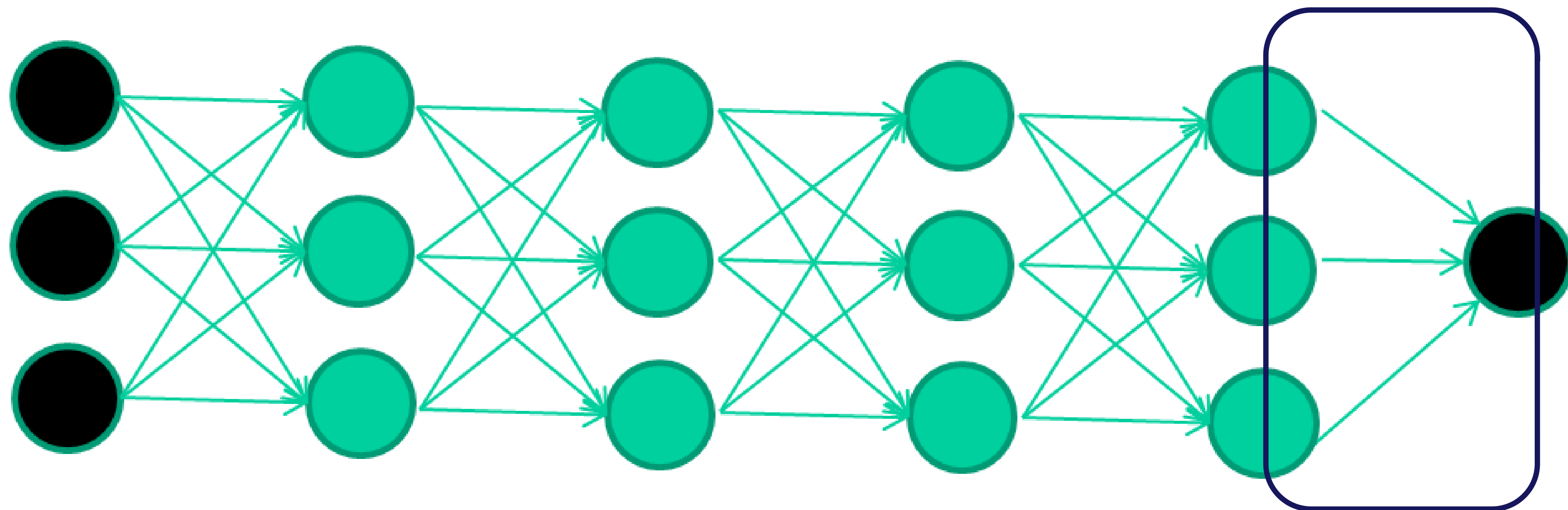


The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.
If there is a structure in the data, than it should find features.

Warstwy ukryte są wytrenowane w celu rozpoznawania cech

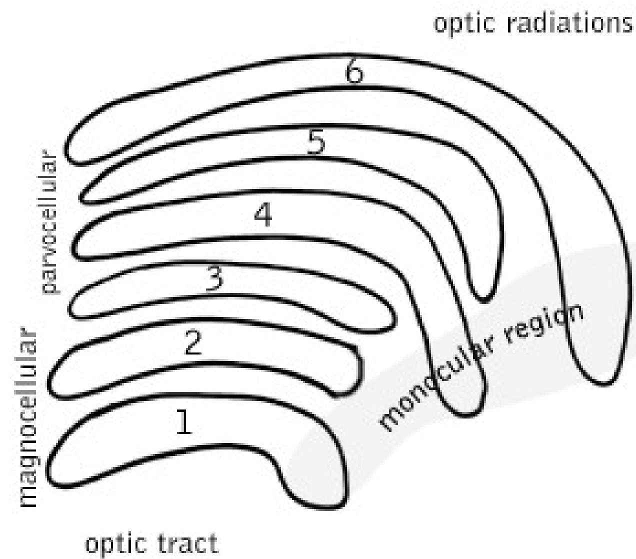


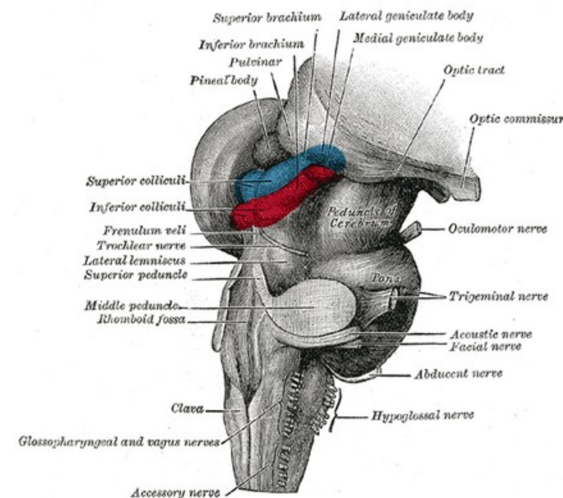
Ostania warstwa przeprowadza klasyfikację



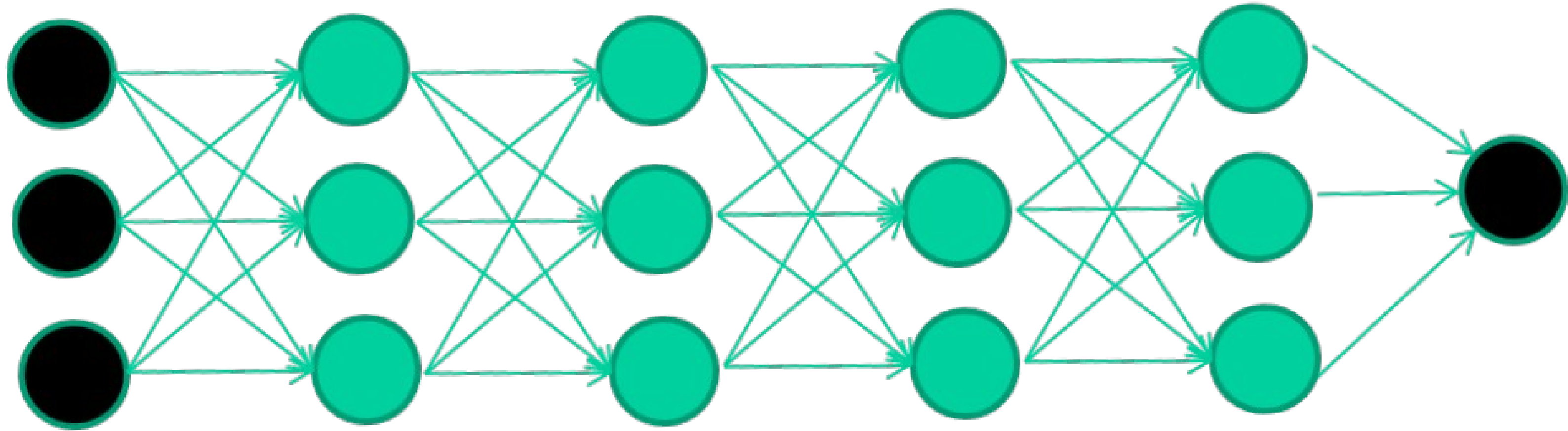
Tak zorganizowana sieć ma sens

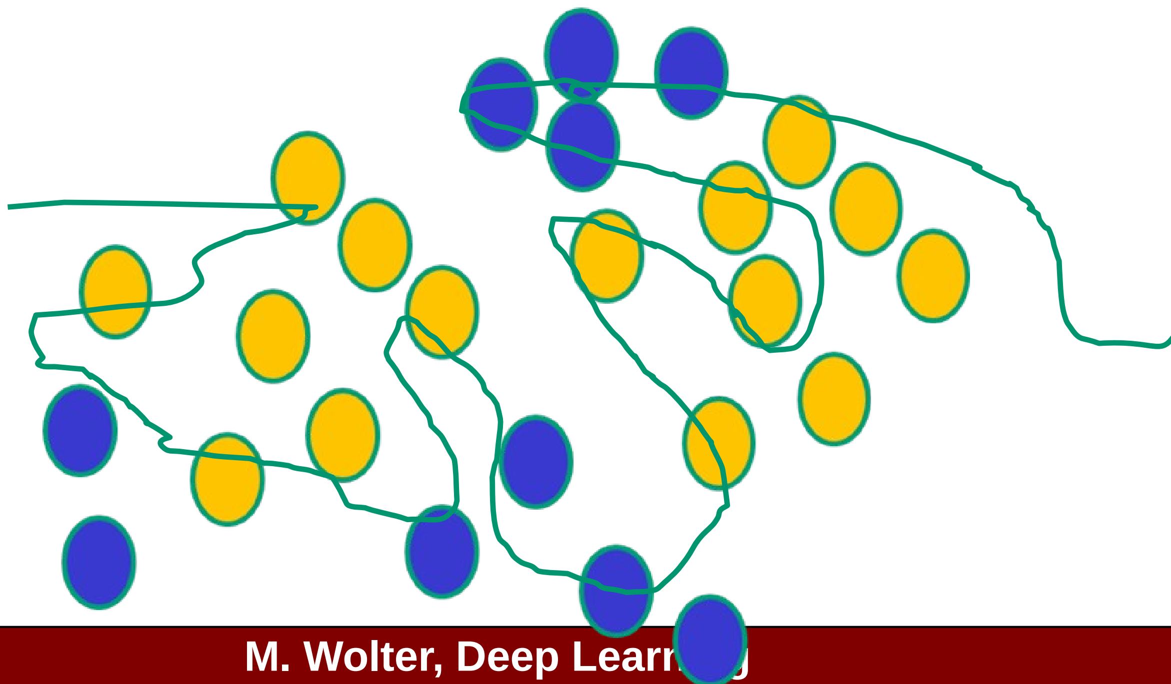
Prawdopodobnie nasz mózg pracuje w ten sposób





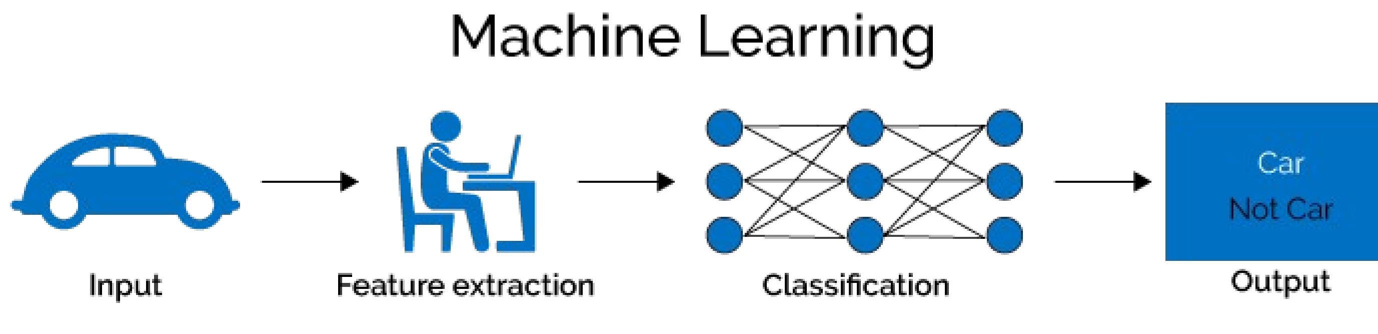
Niestety, do około 2010 nie potrafiliśmy trenować takiej sieci ...

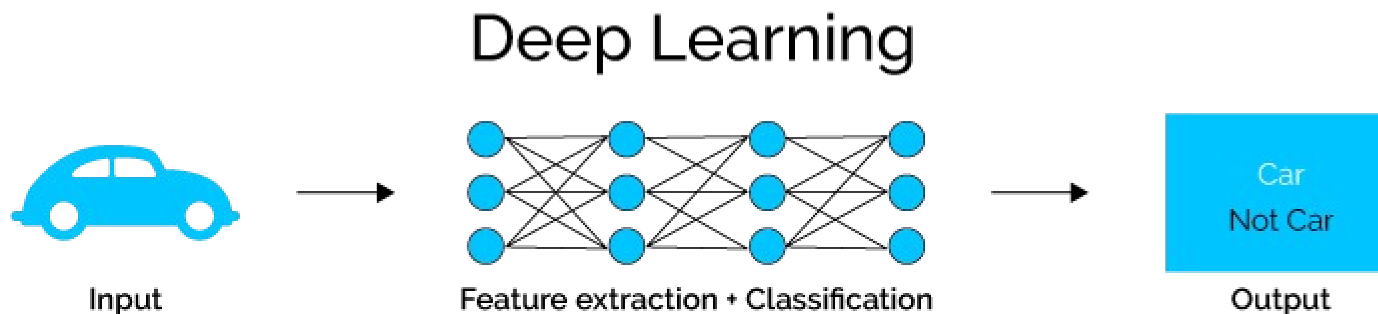




Płytkie i głębokie uczenie

- **Tradycyjne uczenie maszynowe (BDT, sieć neuronowa etc)** – człowiek znajduje dobre zmienne, dobrze różnicujące sygnał i tło (~10), nazywane “features” (cechy), i przeprowadza klasyfikację używając ich jako wejścia do sieci neuronowej.
- **Głębokie uczenie (Deep Learning)** – tysiące lub miliony zmiennych (np. piksele na zdjęciu), cechy są znajdowane automatycznie podczas treningu.





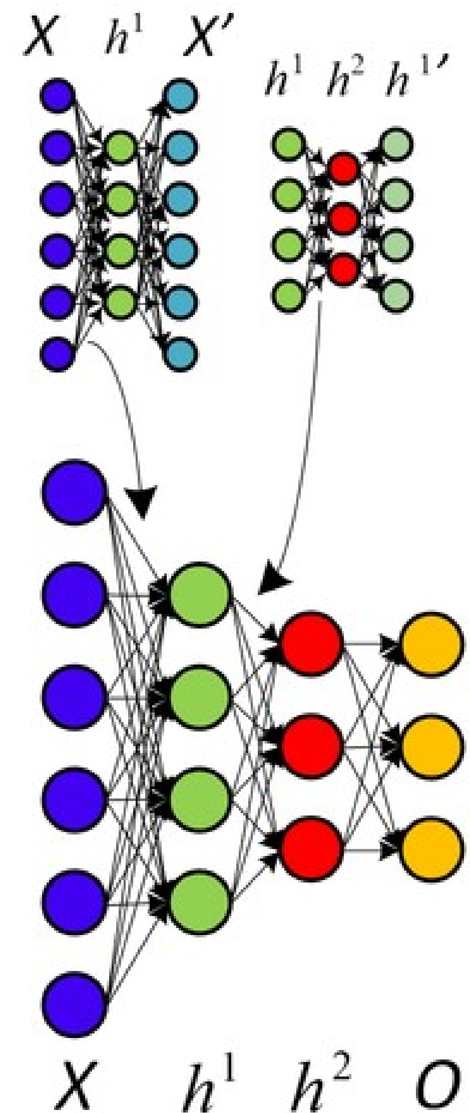
Głębsze sieci?

Płytka sieć neuronowa ma jedną lub dwie warstwy ukryte.

Głęboka sieć neuronowa: stos sekwencyjnie trenpwanych **autoenkoderów (autoencoders)**, which recognize different features (more complicated in each layer) and automatically prepare a new representation of data. This is how our brains are organized.

Parę słów o autoenkoderach:
https://miroslawmamczur.pl/czym-sa-autoenkodery-autokodery-i-jakie-maja-zastosowanie/

Ale jak wytrenować taki stos?



http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning-part-4/

Trenowanie głębokiej sieci neuronowej

- Na początku naszego wieku podejmowano próby trenowania głębokich sieci neuronowych (propagacja wsteczna, „gradient descent”), niestety dobrze znane algorytmy zawodziły. Wielu badaczy twierdziło, że sieci neuronowe są już „passe”, należy używać tylko maszyn wektorów nośnych SVM i wzmocnionych drzew decyzyjnych BDT!

- W 2006 roku Hinton, Osindero i Teh[1] po raz pierwszy odnieśli sukces w trenowaniu głębokiej sieci neuronowej, inicjując najpierw jej parametry sekwencyjnie, warstwa po warstwie. Każda warstwa została wytrenowana do tworzenia reprezentacji danych wejściowych, które z kolei służyły jako dane wejściowe dla następnej warstwy. Następnie tak zainicjowana sieć została ulepszona przy użyciu „gradient descent” (standardowy algorytm).

- **Wierzono, że głęboka sieć neuronowa wymaga starannej inicjalizacji parametrów i bardzo rozbudowanych algorytmów uczenia.**

[1]Hinton, G. E., Osindero, S. and Teh, Y., A fast learning algorithm for deep belief nets, Neural Computation 18, 1527-1554.

Trening z użyciem „brute force”

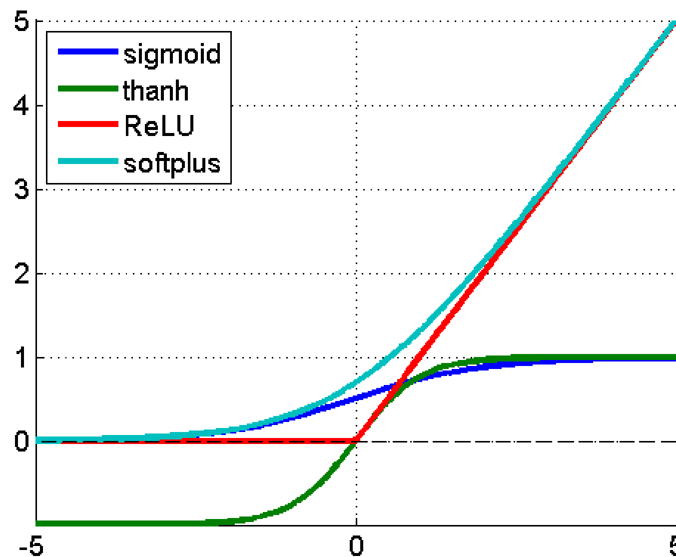
- W 2010 roku zademonstrowano zaskakujący kontrprzykład[1].
- Głęboka sieć neuronowa została wytrenowana do klasyfikowania odręcznych cyfr w zestawie danych MNIST[2] obejmującego 60 000 obrazów 28 × 28 = 784 pikseli do treningu i 10 000 obrazów do testowania.
- Autorzy pokazali, że zwykła DNN (Deep Neural network) z architekturą (784, 2500, 2000, 1500, 1000, 500, 10 – OGROMNA SIEĆ !!!), trenowana przy użyciu standardowego algorytmu „stochastic gradient descent” (Minuit na sterydach!) przewyższała wszystkie inne zastosowane metody. Współczynnik błędu tej sieci z 12 milionami parametrów wyniósł 35 źle sklasyfikowanych obrazów na 10 000.

Obrazy użyte do treningu były losowo i nieznacznie zdeformowane przed każdą epoką uczenia. Wtedy cały zestaw 60 000 nie zaburzonych obrazów może być użyty jako zestaw walidacyjny podczas uczenia, ponieważ żaden z nich nie był używany jako dane uczące.

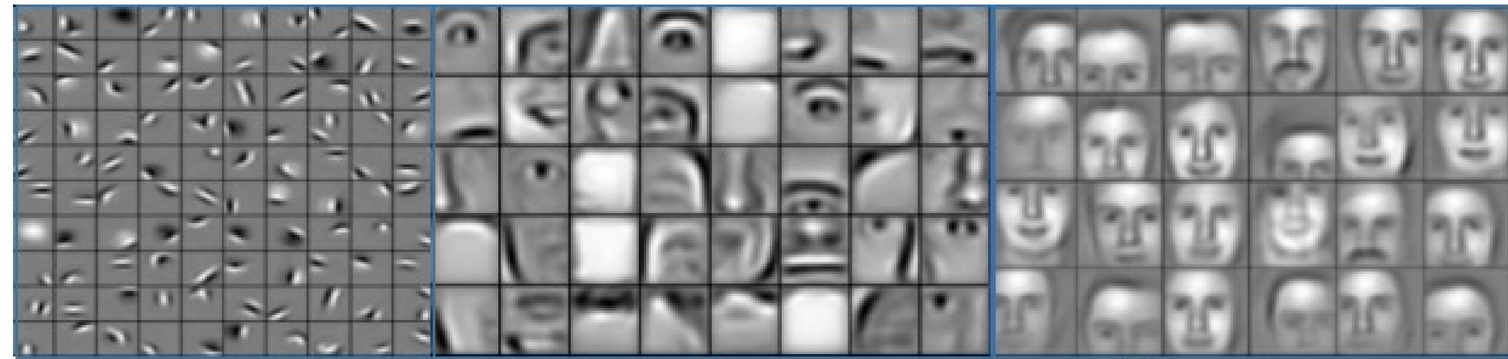
[1] Cireşan DC, Meier U, Gambardella LM, Schmidhuber J. ,Deep, big, simple neural nets for handwritten digit recognition. Neural Comput. 2010 Dec; 22 (12): 3207-20.
[2] http://yann.lecun.com/exdb/mnist/

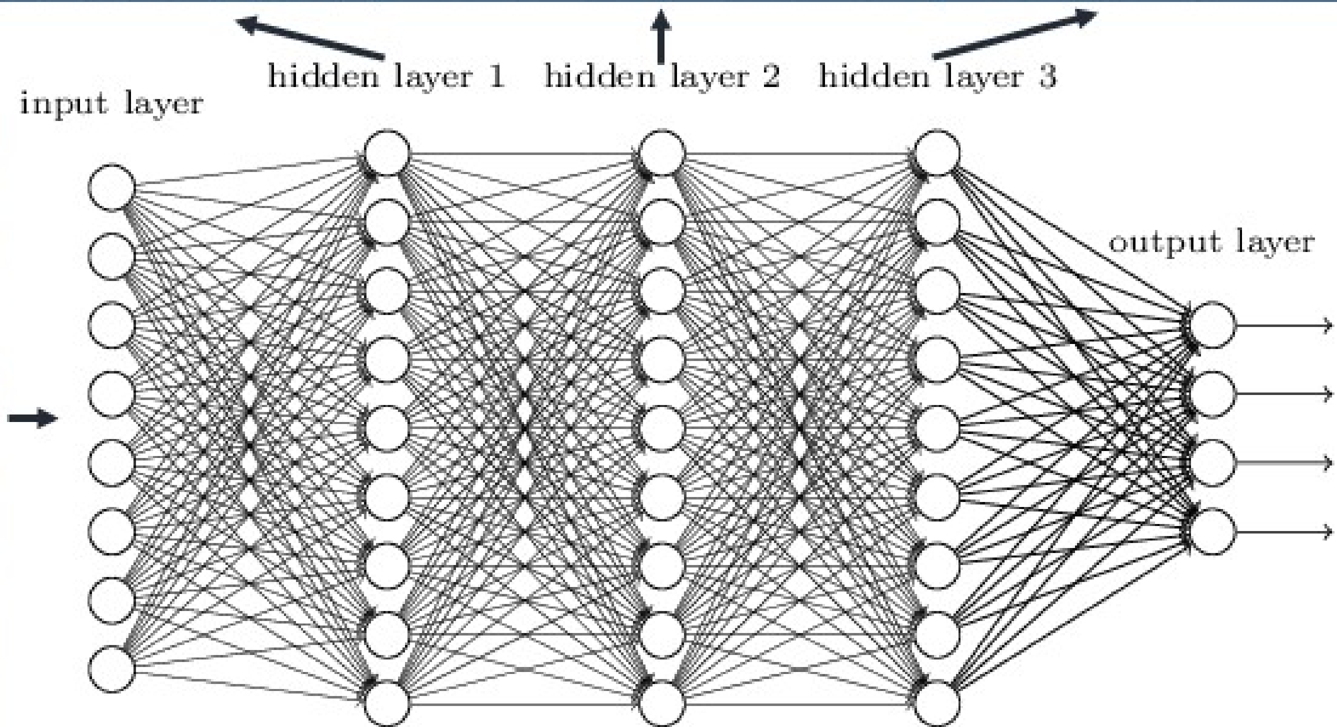
Dlaczego wcześniej uczenie nie działało?

- Więcej danych, więcej mocy obliczeniowej, klastry GPU/CPU
- Konkretna nieliniowa funkcja aktywacji dla neuronów w sieci neuronowej ma duży wpływ na wydajność, a ta często używana domyślnieokazała się nie być najlepszym wyborem.
- Problem „znikającego gradientu” pojawia się głównie dlatego, że wsteczna propagacja obejmuje sekwencję mnożeń, które niezmiennie skutkują mniejszymi pochodnymi dla wcześniejszych warstw. Przynajmniej dopóki wagi nie są wybierane z różnymi skalami w zależności od warstwy, w której się znajdują - ta prosta zmiana skutkuje znaczną poprawą.



Deep neural networks learn hierarchical feature representations





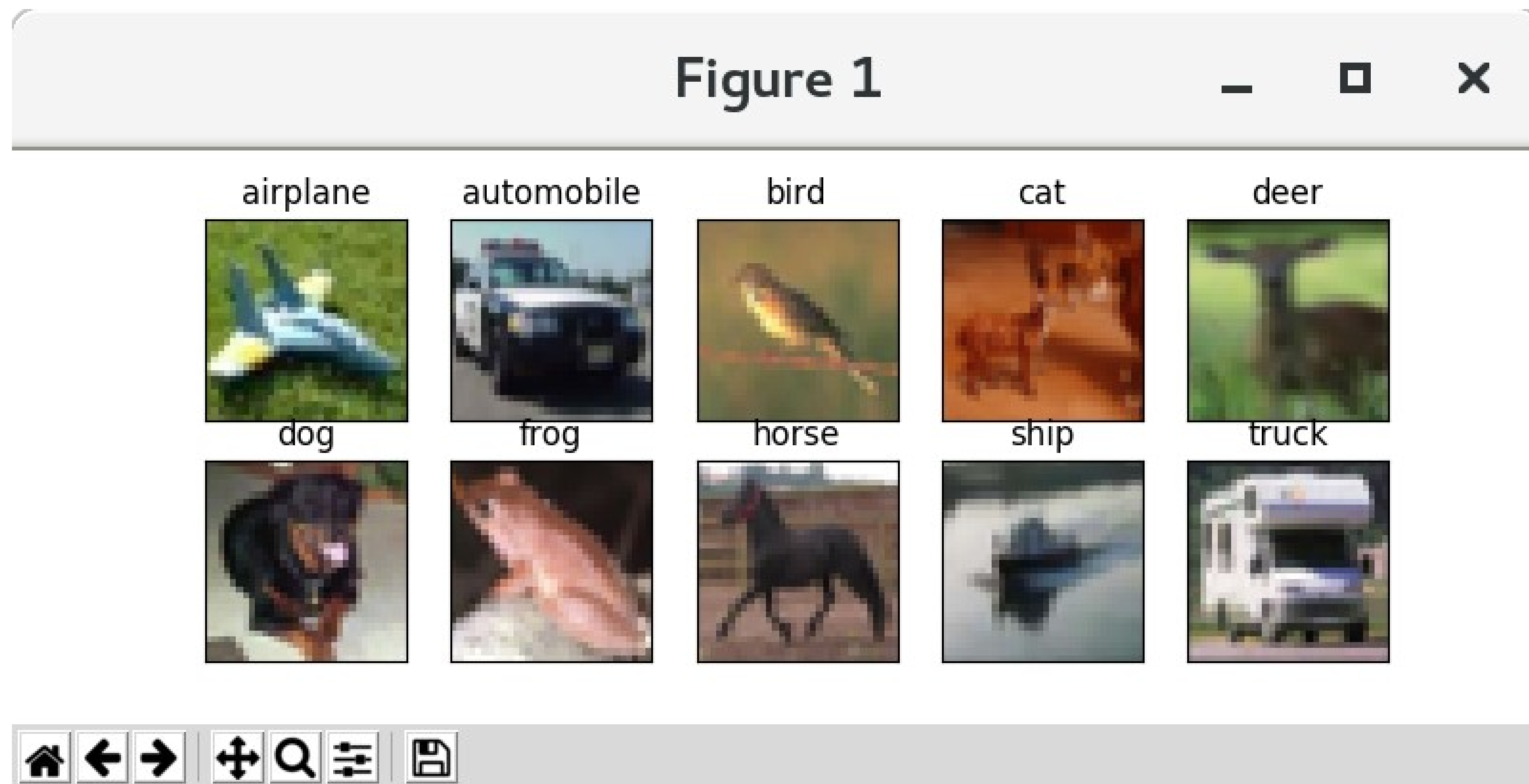
Applet demonstrujący działanie DNN (deep neural network)

Applet:

http://cs.stanford.edu/people/karpathy/convnetjs/

Przykład – rozpoznawanie obrazów KERAS & TensorFlow

- CIFAR10 – małe obrazy. Dane: 50,000 obrazów 32x32 pikseli, kolorowe obrazy treningowe, tagowane 10 kategorii, 10,000 obrazów testowych.



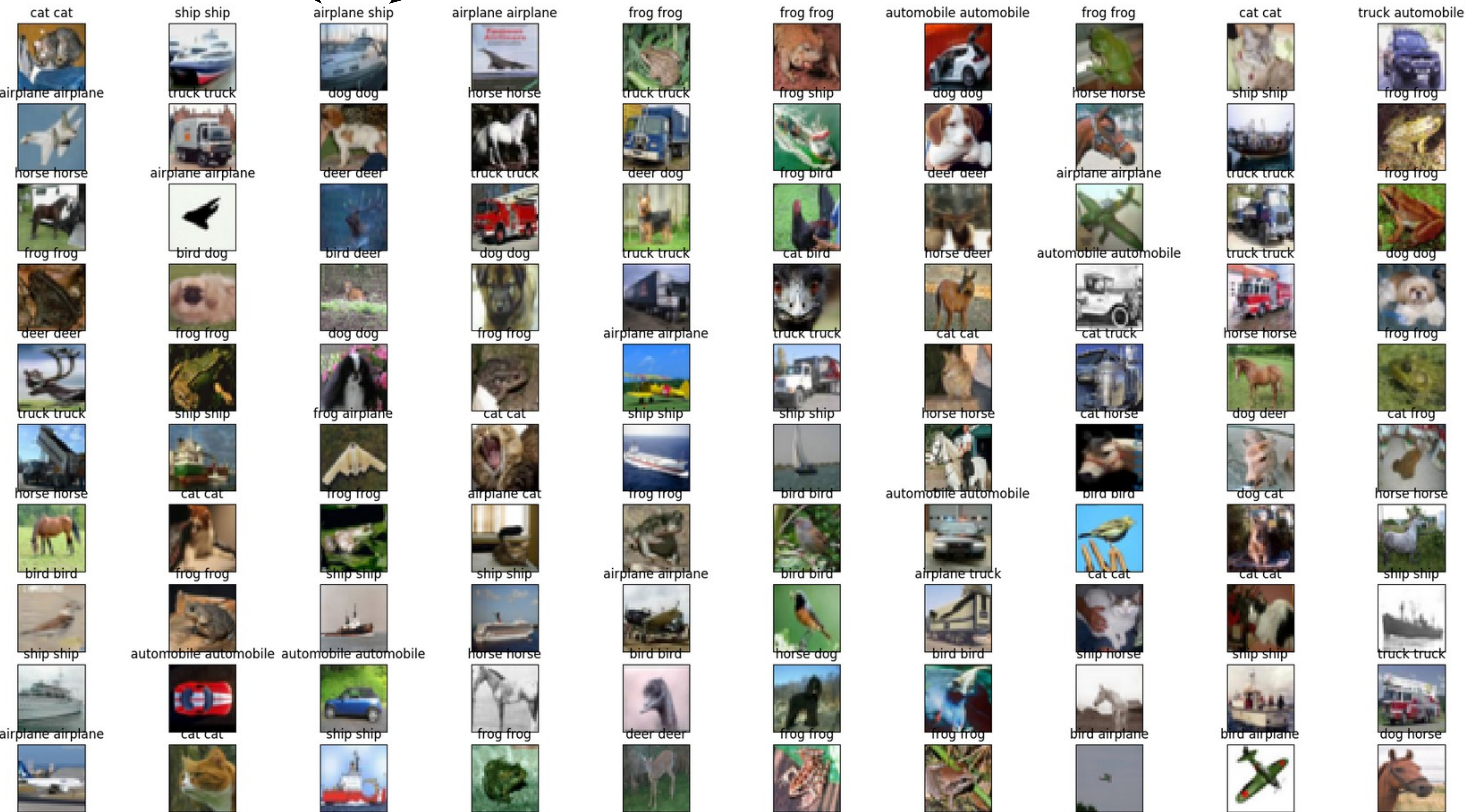
Deep Neural Network

```
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
           OPERATION           DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

               Input   #####      3   32   32
              Conv2D    \|/  -------------------       896     0.0%
                relu   #####     32   32   32
              Conv2D    \|/  -------------------      9248     0.0%
                relu   #####     32   30   30
        MaxPooling2D   Y max -------------------         0     0.0%
                       #####     32   15   15
             Dropout    | || -------------------         0     0.0%
                       #####     32   15   15
              Conv2D    \|/  -------------------     18496     1.0%
                relu   #####     64   15   15
              Conv2D    \|/  -------------------     36928     2.0%
                relu   #####     64   13   13
        MaxPooling2D   Y max -------------------         0     0.0%
                       #####     64    6    6
             Dropout    | || -------------------         0     0.0%
                       #####     64    6    6
             Flatten   ||||| -------------------         0     0.0%
                       #####        2304
               Dense   XXXXX -------------------   1180160    94.0%
                relu   #####         512
             Dropout    | || -------------------         0     0.0%
                       #####         512
               Dense   XXXXX -------------------      5130     0.0%
             softmax   #####          10
Train on 50000 samples, validate on 10000 samples
```

Wyniki

Recognized as
Really was

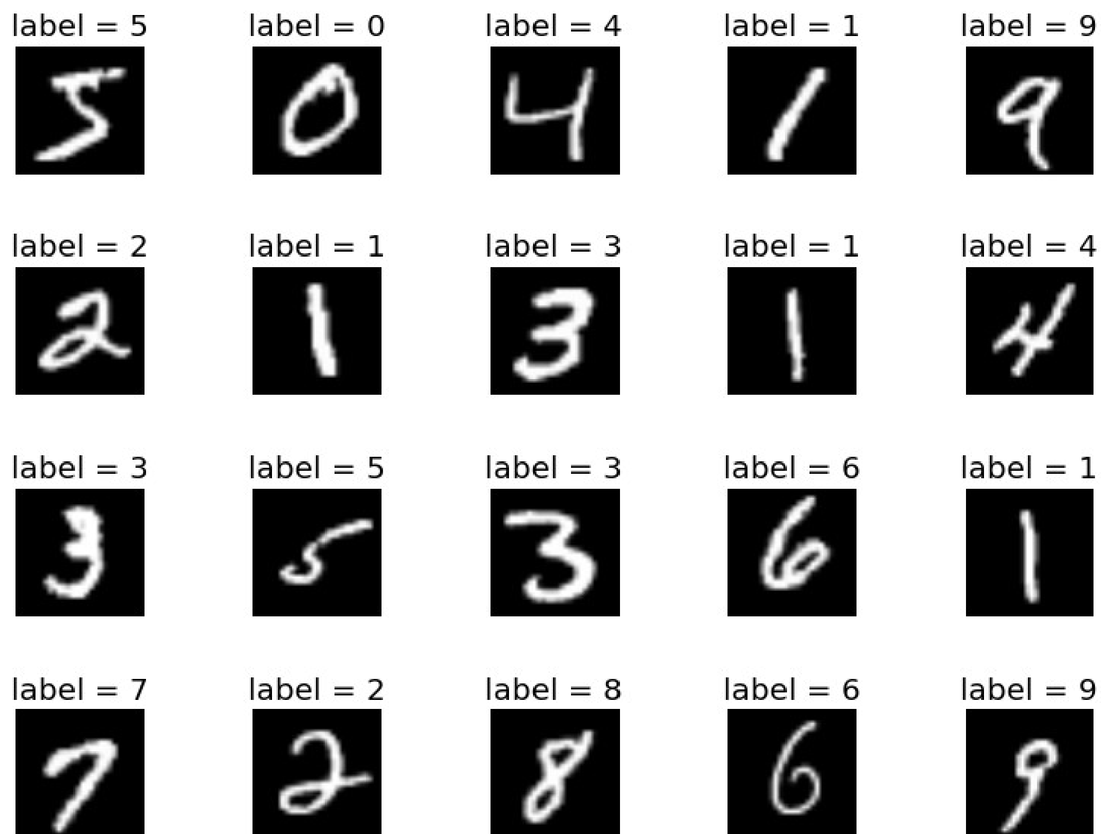


Macierz konfuzji

```
[[754   8  74  47  32   4   9   4  50  18]
 [ 10 875   6  24   3   1  10   0  19  52]
 [ 38   2 678  79  64  52  54  18  11   4]
 [ 14   3  52 647  70 121  51  30   7   5]
 [  7   2  49  85 753  15  39  43   3   4]
 [  6   0  44 192  45 650  26  30   3   4]
 [  3   1  31  70  35   8 837   4   9   2]
 [ 10   0  31  70  57  31   6 790   3   2]
 [ 54  13  15  30  15   4  14   1 831  23]
 [ 31  50   6  36  11   4  12   4  18 828]]
```

Tutorial – jak zaprogramować prostą sieć Keras DNN

- Zadanie – rozpoznać ręcznie pisane cyfry



28 x 28   pixels

```
60000 train samples
10000 test samples
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_7 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_11 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_12 (Dense)             (None, 10)                5130
=================================================================
Total params: 932,362
Trainable params: 932,362
Non-trainable params: 0
```

Inicjalizacja

Pierwszy krok – zdefiniowanie funkcji i klas których użyjemy. Użyjemy NumPy library i kilku klas z Keras library:

```
import matplotlib.pyplot as plt  # matplotlib plotting
import numpy as np

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop, Adam
```

Załadowanie danych

Możemy teraz załadować dane

```
# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

MNIST database of handwritten digits

Dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.
Usage:

from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

Returns:
2 tuples:
x_train, x_test: uint8 array of grayscale image data with shape (num_samples, 28, 28).
y_train, y_test: uint8 array of digit labels (integers in range 0-9) with shape (num_samples,).

Dane MNIST

6	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0

784 liczb

Zróbmy teraz „numpy array” o kształcie (6000, 784) z „python tuple”

```
# reshape dataset
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

# convert to float32
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

#normalize to one
x_train /= 255
x_test /= 255
```

Przygotowanie danych ”convert to categorical”

Chcemy używać NN z 10 wyjściami (każde odpowiada jednej cyfrze) aby otrzymać 10 prawdopodobieństw odpowiadającym poszczególnym cyfrom. Przekształcamy więc y_train z liczby na wektor:

- 7 → (0, 0, 0, 0, 0, 0, 0, 1, 0)
- 0 → (1, 0, 0, 0, 0, 0, 0, 0, 0)
- 9 → (0, 0, 0, 0, 0, 0, 0, 0, 1)

```
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

num_classes = 10
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Zdefiniujmy Keras Model

Model sieci definiujemy jako sekwencję warstw.

Tworzymy „Sequential model” i dodajemy do niego warstwy.

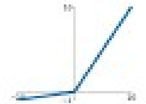
Po pierwsze musimy zapewnić, że pierwsza warstwa ma odpowiednią liczbę wejść (u nas 784).

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
```

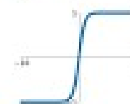
Funkcja aktywacji: relu (Rectified Linear)

Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$

Leaky ReLU
$\max(0.1x, x)$

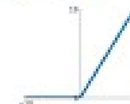


tanh
$\tanh(x)$

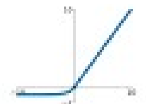


Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ReLU
$\max(0, x)$



ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Keras Model

Dodawanie kolejnych warstw. Skąd znamy liczbę warstw i ich typy?

To trudne pytanie. Istnieją heurystyki, których możemy użyć, ale najlepszą strukturę sieci można znaleźć w procesie eksperymentowania metodą prób i błędów. Ogólnie rzecz biorąc, potrzebujemy sieci wystarczająco dużej, aby uchwycić strukturę problemu.

W tym przykładzie użyjemy w pełni połączonej struktury sieci (fully connected) z trzema ukrytymi warstwami.

W pełni połączone warstwy są definiowane za pomocą klasy **Dense**. Możemy określić liczbę neuronów lub węzłów w warstwie jako pierwszy argument i określić funkcję aktywacji za pomocą argumentu aktywacji.

Użyjemy funkcji aktywacji ReLU w pierwszych trzech warstwach:

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='relu'))
```

Keras Model

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

Dodanie warstwy wyjściowej z *num_classes = 10* i funkcją aktywacji *softmax* (patrz następny slajd). Używamy *softmax* na warstwie wyjściowej, aby zapewnić, że wartości wyjściowezmieszczą się w zakresie od 0 do 1 i jest łatwe do zmapowania na prawdopodobieństwa lub też może być użyte do twardej klasyfikacji.

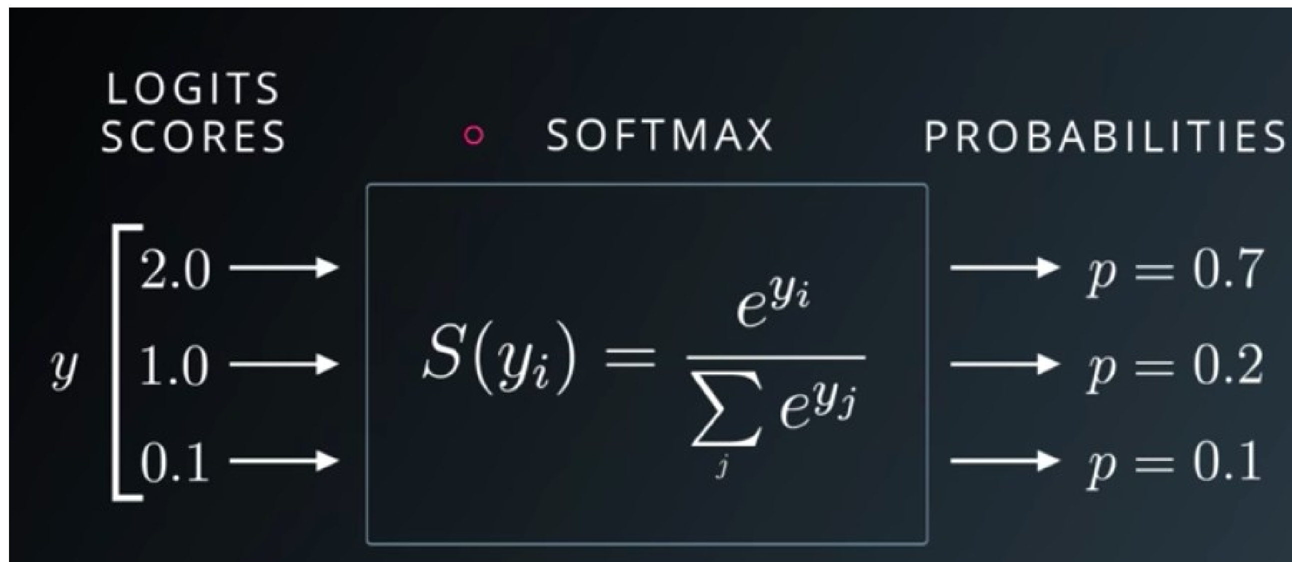
Między warstwami dodajemy warstwę *Dropout*, aby uniknąć przetrenowania: *Dropout* polega na losowym ustawianiu pewnej liczby wejśćna 0 przy każdej aktualizacji w czasie treningu, co pomaga zapobiegać przetrenowaniu.

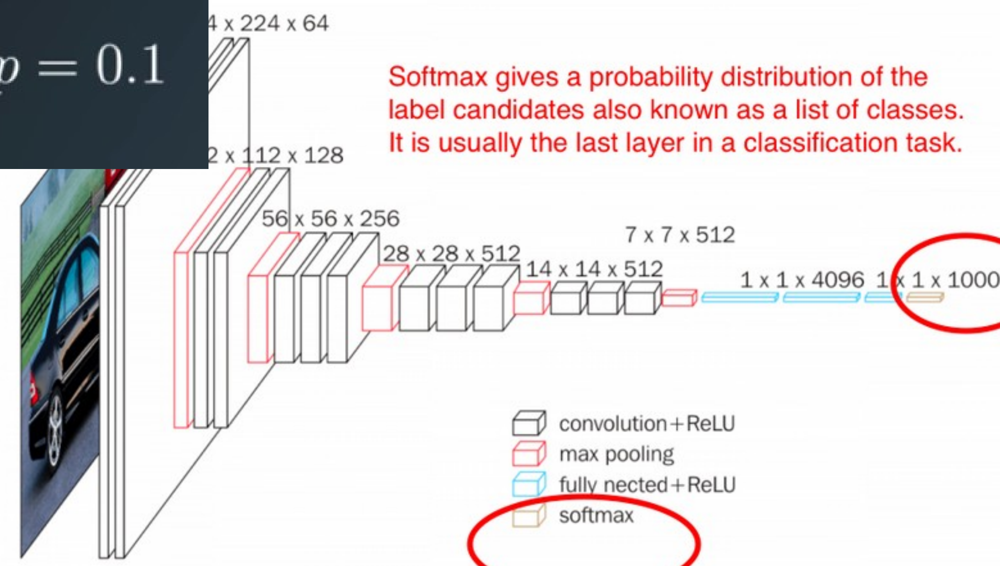
http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

`model.summary()` - wypisuje strukturę sieci

Funkcja aktywacji softmax

Funkcja *softmax*: zamienia liczby zwane *logitami* na prawdopodobieństwa, które sumują się do jednego. Funkcja *softmax* generuje wektor, który reprezentuje rozkłady prawdopodobieństwa. Jest to podstawowy element używany w zadaniach klasyfikacyjnych uczenia głębokiego.





Kompilacja sieci

```
model.compile(loss='categorical_crossentropy',
      optimizer=RMSprop(),
      metrics=['accuracy'])
```

Podczas kompilacji musimy określić dodatkowe parametry wymagane podczas uczenia sieci. Uczenie sieci oznacza znalezienie najlepszego zestawu wag do mapowania danych wejściowych na dane wyjściowe.
Musimy określić funkcję straty, która ma być używana do oceny zestawu wag, oraz optymalizator służący ajdowania tych wag.

Przyjęta funkcja straty związana jest z klasyfikacją kategoryczną i jest zdefiniowana w Keras jako „categorical_crossentropy”. Więcej o wyborze funkcji straty:

https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

Optymalizator wybieramy jako wydajny „stochastic gradient descent” nazywany “*RMSprop*“. Możemy również użyć optymalizatora “adam”, popularnej wersji „gradient descent”. Dopasowuje się on automatycznie do problemu i przeważnie daje dobre wyniki.

Trening sieci

```
batch_size = 128
epochs = 10

history = model.fit(x_train, y_train,
         batch_size=batch_size,
         epochs=epochs,
         verbose=1,
         validation_data=(x_test, y_test))
```

Możemy wytrenować nasz model na danych treningowych, wywołując funkcję *fit ()*.
Trenowanie odbywa się w epokach, a każda epoka jest dzielona na partie „batches”.

Epoch: jeden przebieg przez wszystkie wiersze w zestawie danych uczących.

Batch: co najmniej jedna próbka brana pod uwagę przez model w okresie przed aktualizacją wag.

Jedna epoka składa się z jednej lub więcej partii „bstches”. W przypadku tego problemu będziemy trenować przez niewielką liczbę epok (10) i użyć rozmiaru „batch” 128.

Te konfiguracje można wybrać eksperymentalnie metodą prób i błędów. Chcemy wystarczająco wytrenować model, aby nauczył się dobrego (lub wystarczająco dobrego) odwzorowania danych wejściowych do znanej klasyfikacji wyjściowej. Model zawsze będzie miał jakieś błędy, ale ilość błędów będzie spadać i w końcu osiągnie plateau.

Ewaluacja modelu

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy:', score[1])
```

We have trained our neural network on the entire dataset and we can evaluate the performance of the network on another “test” dataset.
You can evaluate your model on a dataset using the evaluate() function.
This will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics you have configured, such as accuracy.
The evaluate() function will return a list with two values. The first will be the loss of the model on the dataset and the second will be the accuracy of the model on the dataset.

Ewaluacja modelu

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy:', score[1])
```

Wytrenowaliśmy naszą sieć neuronową na całym zestawie danych i możemy ocenić wydajność sieci na innym, „testowym” zestawie danych.
Możemy ocenić swój model na zbiorze danych za pomocą funkcji *evaluate()*.
Spowoduje to wygenerowanie prognozy dla każdej pary wejścia i wyjścia i zebranie wyników, w tym średniej straty oraz dokładności „accuracy”.
Funkcja *evaluate()* zwróci listę z dwiema wartościami. Pierwszym będzie strata modelu, a drugim dokładność.

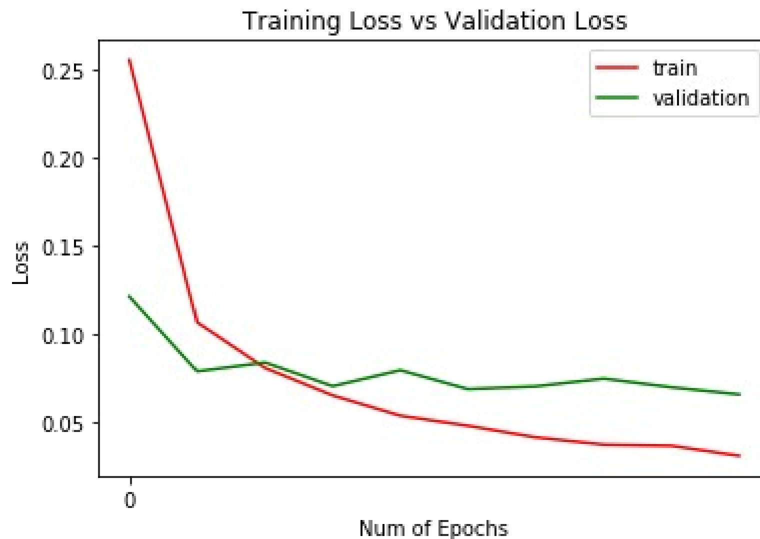
Zbierzmy wszystko razem

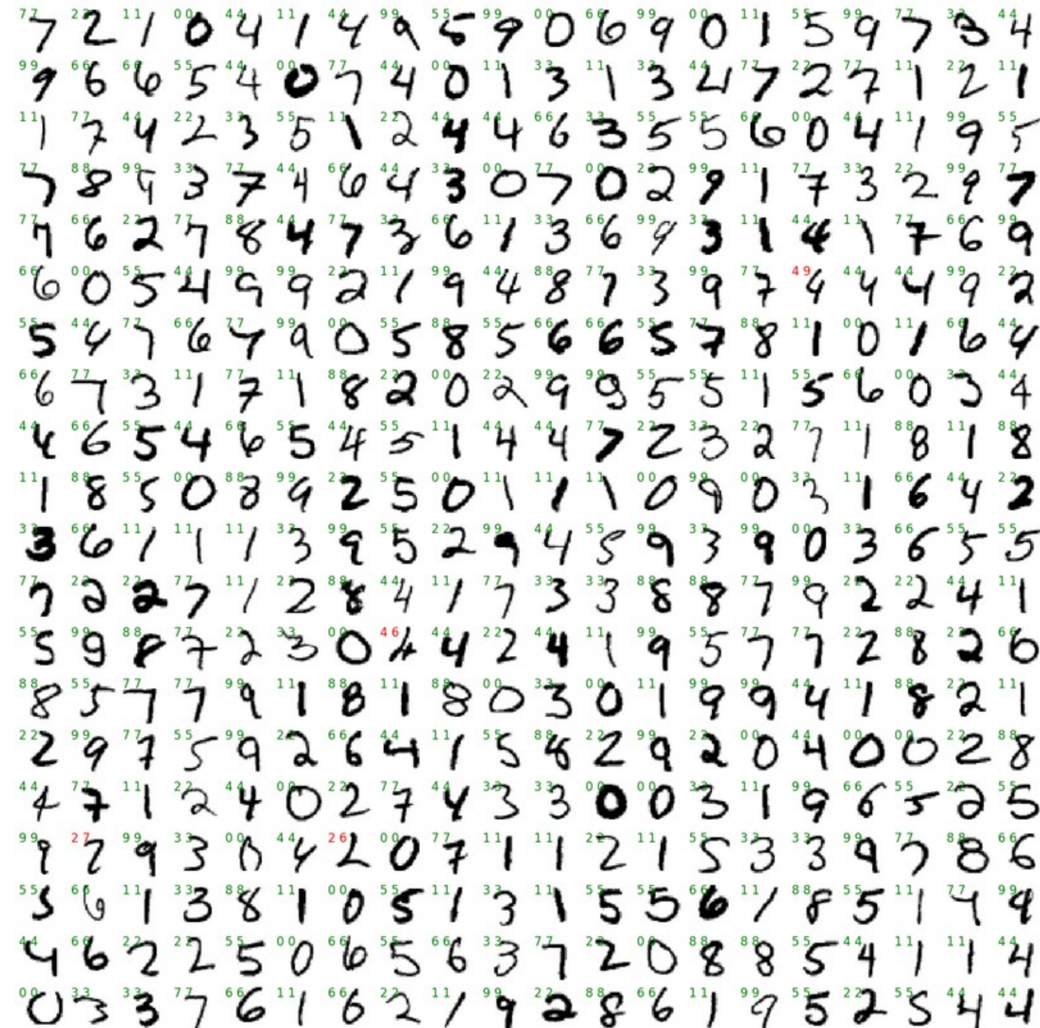
https://github.com/marcinwolter/DeepLearning_2020/blob/main/mnist_mlp_minimal.ipynb

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                5130
=================================================================
Total params: 932,362
Trainable params: 932,362
Non-trainable params: 0
```

Więcej dodatków

- https://github.com/marcinwolter/MachineLearning2020/blob/main/mnist_mlp.ipynb
- Visualization of results
- Plotting the Neural Network structure





Podsumowanie

- Zbudowaliśmy naszą pierwszą głęboką sieć neuronową!!!