pawel.kasprowski@polsl.pl

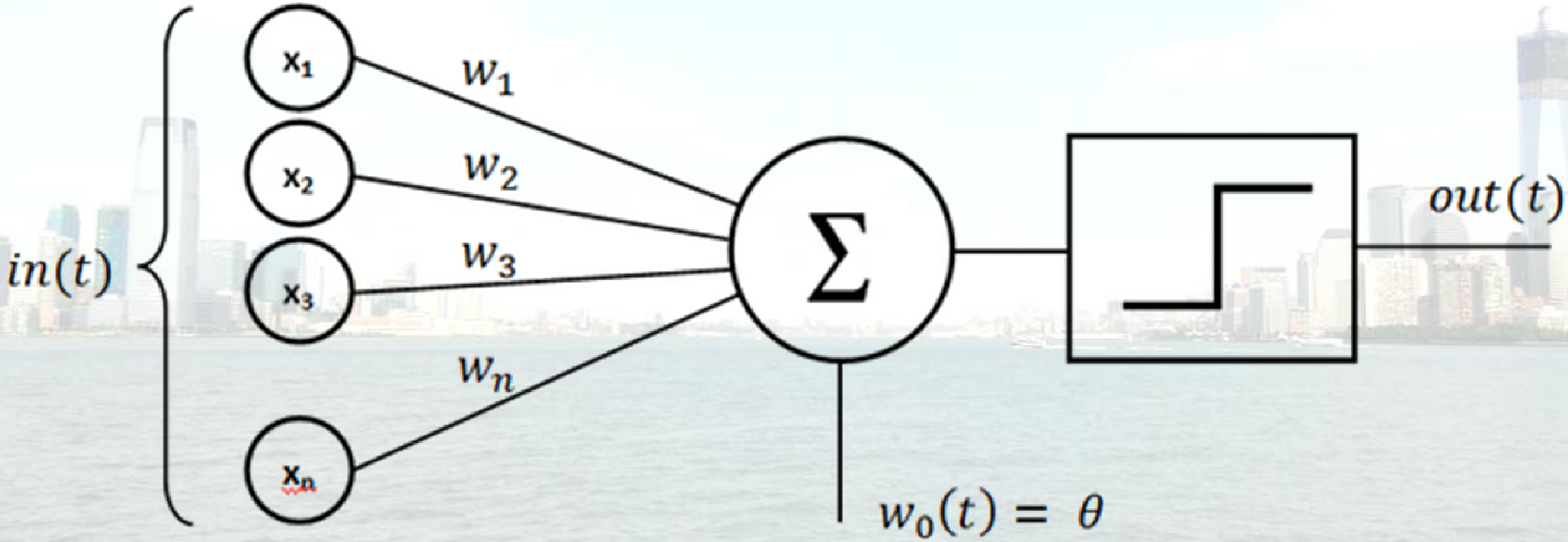
photo: New Jersey/New Jork

Artificial Intelligence

Neural Networks

Paweł Kasprowski, PhD, DSc.

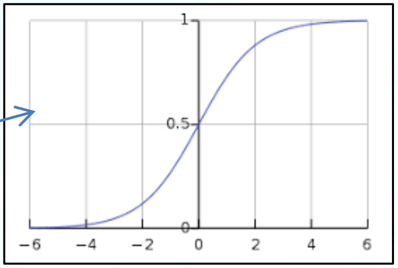
Perceptron (Rosenblatt 1957)

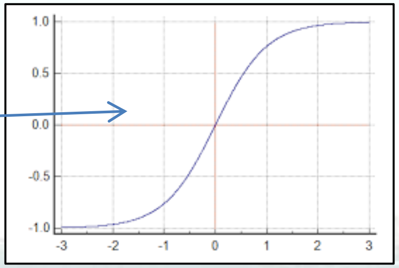


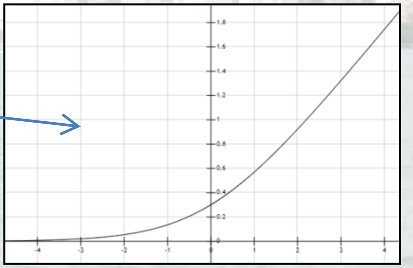
pawel.kasprowski@polsl.pl

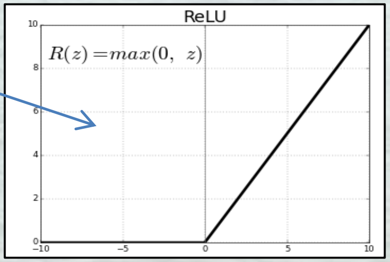
Activation function

- Linear combination of inputs
- Activation functions may be different
  - threshold function
  - sigmoid function
  - tanh function
  - softplus
  - RELU





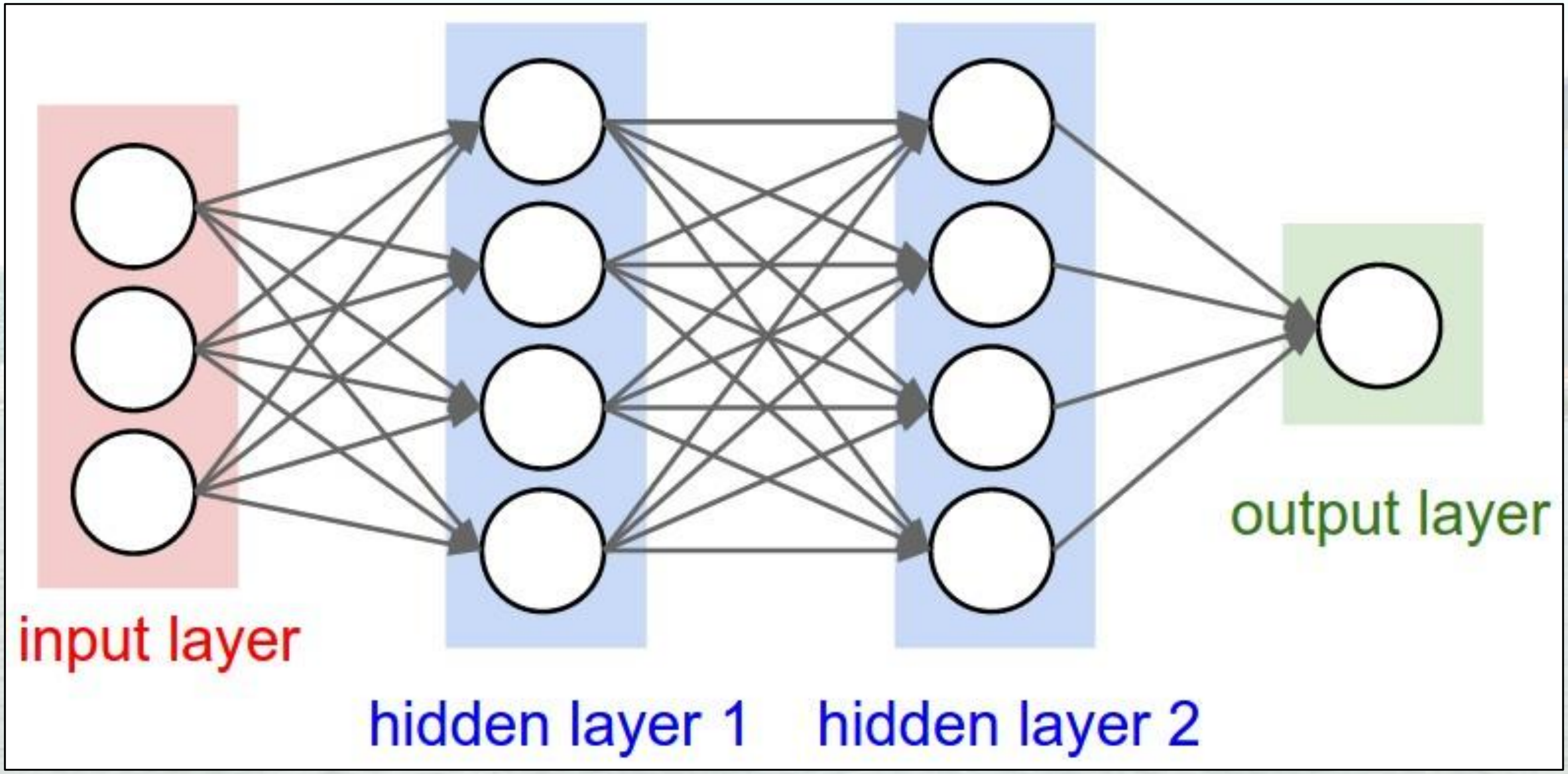




pawel.kasprowski@polsl.pl

Artifical Neural Network (ANN)

- input layer > hidden layers > output layer



http://cs231n.github.io/convolutional-networks/

pawel.kasprowski@polsl.pl

Training the network

- Back propagation
  - the state-of-the art method to train the network
  - utilizes the already mentioned Gradient Descent algorithm
- The algorithm:

initialize weights

repeat many times:

use network to calculate output (Y_{pred}) for some examples (X)

calculate error (loss function) using Y_{pred} and Y (real)

update weights to minimize loss (using gradient for direction)

pawel.kasprowski@polsl.pl

Tuning - hyperparameters

- Network structure
  - Number of layers
  - Number of neurons for layer
  - Connections
  - Activation functions for layers
- Loss function
- Optimization algorithm
  - how to change the weights
  - learning rate (how big changes of weights)

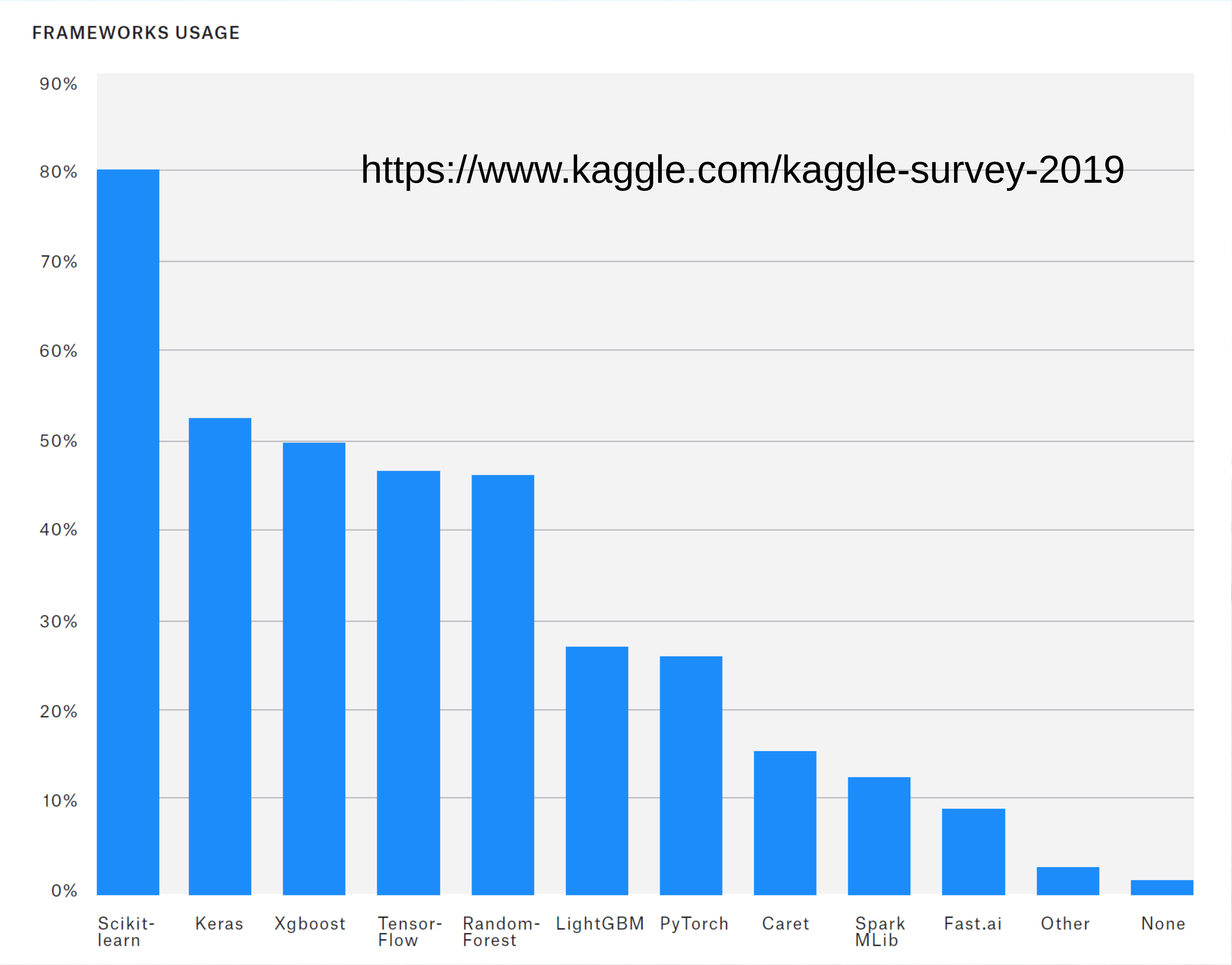
pawel.kasprowski@polsl.pl

Implementations

- Many classification libraries like scikit-learn or WEKA implement the neural networks
  - but only the simplest models
- For Deep Learning there are many libraries developed by leading companies:
  - Tensorflow, Google
  - PyTorch, Facebook
  - CNTK, Microsoft
  - ...
- We will use the most popular: Keras/Tensorflow

Keras

- General interface to use deep networks
- Works with Tensorflow, Theano, CNTK...
- User-friendly, modular, and extensible
- Created in Google
- Works in Python
- Tensorflow library includes Keras by default
  - that is why we installed Tensorflow and we use Keras
- Tensorflow 2 is integrated with Keras



https://www.kaggle.com/kaggle-survey-2019

pawel.kasprowski@polsl.pl

Table 3: Major Deep Learning Platforms

Platform	2019 % share	2018 % share	% change
Tensorflow	31.7%	29.9%	5.8%
Keras	26.6%	22.2%	19.7%
PyTorch	11.3%	6.4%	75.5%
Other Deep Learning Tools	5.6%	4.9%	15.2%
DeepLearning4J	2.5%	3.4%	-25.6%
Apache MXnet	1.7%	1.5%	13.1%
Microsoft Cognitive Toolkit	1.6%	3.0%	-45.5%
Theano	1.6%	4.9%	-67.4%
Torch	0.9%	1.0%	-6.1%
TFLearn	0.7%	1.1%	-34.7%
Caffe	0.6%	1.5%	-58.3%

https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html

pawel.kasprowski@polsl.pl

Keras basics

- Build the model (network)
  - contains layers with neurons and connections
- Compile the model (model.compile)
  - define loss function and optimizer
- Train the model (model.fit)
  - provide samples with known labels
- Use the model for predictions (model.predict)
  - predict labels of unknown samples

The simplest example

```
# import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
# build model
model = Sequential()
model.add(Dense(50, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
# compile model
model.compile(loss='binary_crossentropy',
    optimizer="adam",metrics=['accuracy'])
# train model
model.fit(samples, labels, epochs=100, batch_size=10)
# use model
predicted = model.predict(sample)
```

pawel.kasprowski@polsl.pl

Decoder

decoder1.ipynb

- Decodes a binary list:
  - [1,0,0,1,0,0,0,1]
- to the number:
  - 145
- Input: 8 values (0 or 1)
- Output: one value (the number)

pawel.kasprowski@polsl.pl

model.compile parameters

- loss – loss function
  - *binary_crossentropy* for 2-class problem
  - *categorical_crossentropy* for many classess (one-hot encoded)
  - *mean_squared_error* for regression
- optimizer – algorithm for back propagation
  - SGD
  - adam
  - RMSprop
- metrics – metrics to measure after each iteration
  - *accuracy* for classification
  - *mean squared error* for regression

pawel.kasprowski@polsl.pl

model.fit parameters

- samples – array of samples
- labels – array of labels (one for each sample)
- epochs – number of iterations
- batch_size – number of samples per batch (each back propagation pass)
  - stochastic (1 – backprop after every sample)
  - mini-batch (from 2 to N-1)
  - batch (N – backprop after all samples – one per epoch)
- validation_split – percent of validation samples
- validation_data = (samples, labels)

pawel.kasprowski@polsl.pl

Input parameters

- samples – a list of samples
  - every sample may be multidimensional (e.g. 3D for images)
- labels – a list of correct labels for each sample
  - a list of values [0,1,4,2,1,4,2...] (one output node)
  - one-hot encoded:
    - 0 > [1,0,0,0,0]
    - 1 > [0,1,0,0,0]
    - 4 > [0,0,0,0,1]

    (five output nodes)

Example

iris-keras.ipynb

- Create model:

```
model = Sequential()
model.add(Dense(50, input_dim=4, activation='sigmoid'))
model.add(Dense(50, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

- Use model

```
model.compile(loss='binary_crossentropy',
    optimizer="adam",metrics=['accuracy'])
model.fit(trainSamples, trainLabels,
    epochs=10,batch_size=10)
```

pawel.kasprowski@polsl.pl

Multinomial classification

- More than two classes
  - E.g. cats, dogs, snakes and elephants
- The model returns a probability for each class
- We choose the class with the highest probability
- Typical output:
  - binary class matrix (one-hot encoding)
    - 1 -> [1,0,0,0,0]
    - 2 -> [0,1,0,0,0]
    - 3 -> [0,0,1,0,0]
    - …
- The simplest method:

```
onehot_labels = tf.keras.utils.to_categorical(labels)
```

  - *(it requires integer labels!)*

pawel.kasprowski@polsl.pl

Dataset with tree classes

iris-keras.ipynb / Multinomial

- Encoding labels

```
data.loc[data['iris']=='Iris-versicolor'] = 0
data.loc[data['iris']=='Iris-setosa'] = 1
data.loc[data['iris']=='Iris-virginica'] = 2
labels = tf.keras.utils.to_categorical(labels)
```

  - shape: (150,3)

- Output from the network:

```
model.add(Dense(3, activation='softmax'))
```

Conversion of labels

- LabelEncoder – encodes labels to int numbers

```
encoder = sklearn.preprocessing.LabelEncoder()
int_labels = encoder.fit_transform(labels)
```

- It is possible to find the original label for integer

```
encoder.inverse_transform(int_label))
```

pawel.kasprowski@polsl.pl

One-hot encoding

- oh_labels = tf.keras.utils.to_categorical(labels)
- Problems:
  - original labels must be integer
  - no inverse transform
- More sophisticated way – LabelBinarizer

```
lb = sklearn.preprocessing.LabelBinarizer()
oh_labels = lb.fit_transform(labels)
```

decoding:

```
lb.inverse_transform(oh_label))
```

pawel.kasprowski@polsl.pl

Labeling example

binarizer.ipynb

- LabelBinarizer
- Problem with two classes
- Combining LabelEncoder and to_categorical

pawel.kasprowski@polsl.pl

EMVIC 2012 data

- Identification of people based on their eye movements
- Part of EMVIC 2012[1] dataset:
  - 155 samples
  - 3 classes (persons)
  - 16384 attributes (position-vel-acc)
- unbalanced distribution
  - {'a25': 104, 'a41': 39, 'a37': 11}

[1] Kasprowski, Komogortsev, Karpov: First eye movement verification and identification competition at BTAS 2012, IEEE Fifth International Conference on Biometrics: Theory, Applications and Systems (BTAS)

pawel.kasprowski@polsl.pl

Loading and preparing data

emvic.ipynb

- Loading

```
dataframe = pandas.read_csv(file)
dataset = dataframe.values
samples = dataset[:,1:]
labels = dataset[:,0]
```

- Another useful class - Counter

```
from collections import Counter
print("Class distribution:", Counter(labels))
```

pawel.kasprowski@polsl.pl

Dataset preparation (1)

- Choose 100 best attributes

```
newSamples = SelectKBest(k=100)
                        .fit_transform(samples, labels)
```

- Add weights to classes (dictionary!)

```
class_weights = class_weight.compute_class_weight(
   'balanced',np.unique(labels),labels)

d_class_weights = dict(enumerate(class_weights))
```

- Normalize the dataset

```
normalize(samples)
```

Dataset preparation (2)

- One-hot encoding

```
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
classesNum = labels.shape[1]
```

- Division to training and testing

```
(trainSamples, testSamples, trainLabels, testLabels) =
    train_test_split(samples, labels, test_size=0.25)
```

pawel.kasprowski@polsl.pl

The model

- Simple MLP model:

```
model = Sequential()
model.add(Dense(250, activation='sigmoid'))
model.add(Dense(250, activation='sigmoid'))
model.add(Dense(250, activation='sigmoid'))
model.add(Dense(classesNum, activation='softmax'))

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

Training, testing, reporting

- Training:

```
H = model.fit(trainSamples, trainLabels
    ,class_weight=d_class_weights
    ,validation_data=(testSamples,testLabels)
    batch_size=BATCH, epochs=EPOCHS, )
```

- Testing:

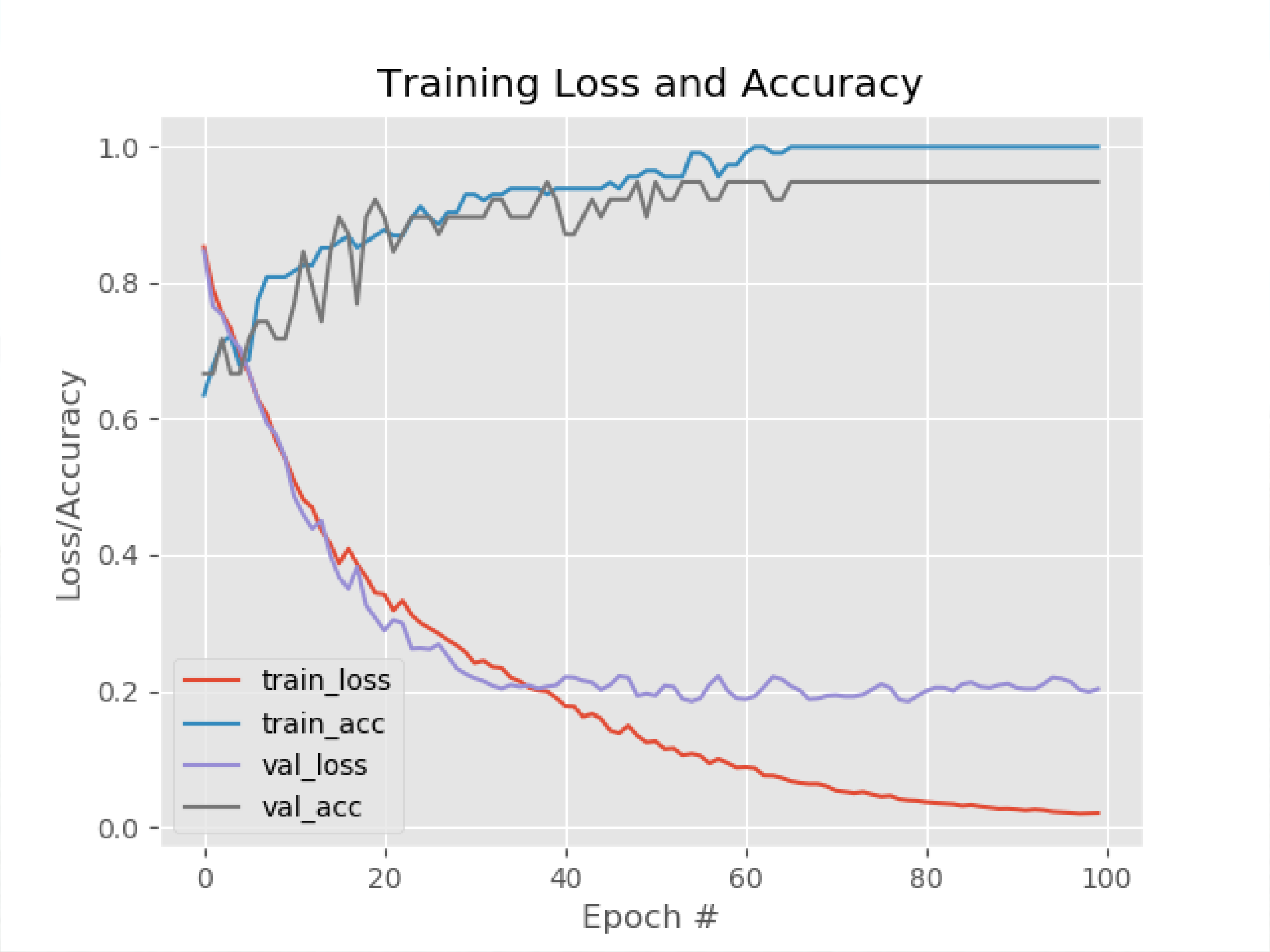
```
mlpResults = model.predict(testSamples)
```

- Reporting (using methods from sklearn):

```
print(confusion_matrix(testLabels.argmax(axis=1),
    mlpResults.argmax(axis=1)))
print(classification_report(testLabels.argmax(axis=1),
    mlpResults.argmax(axis=1),target_names=lb.classes_))
```

pawel.kasprowski@polsl.pl

Training results

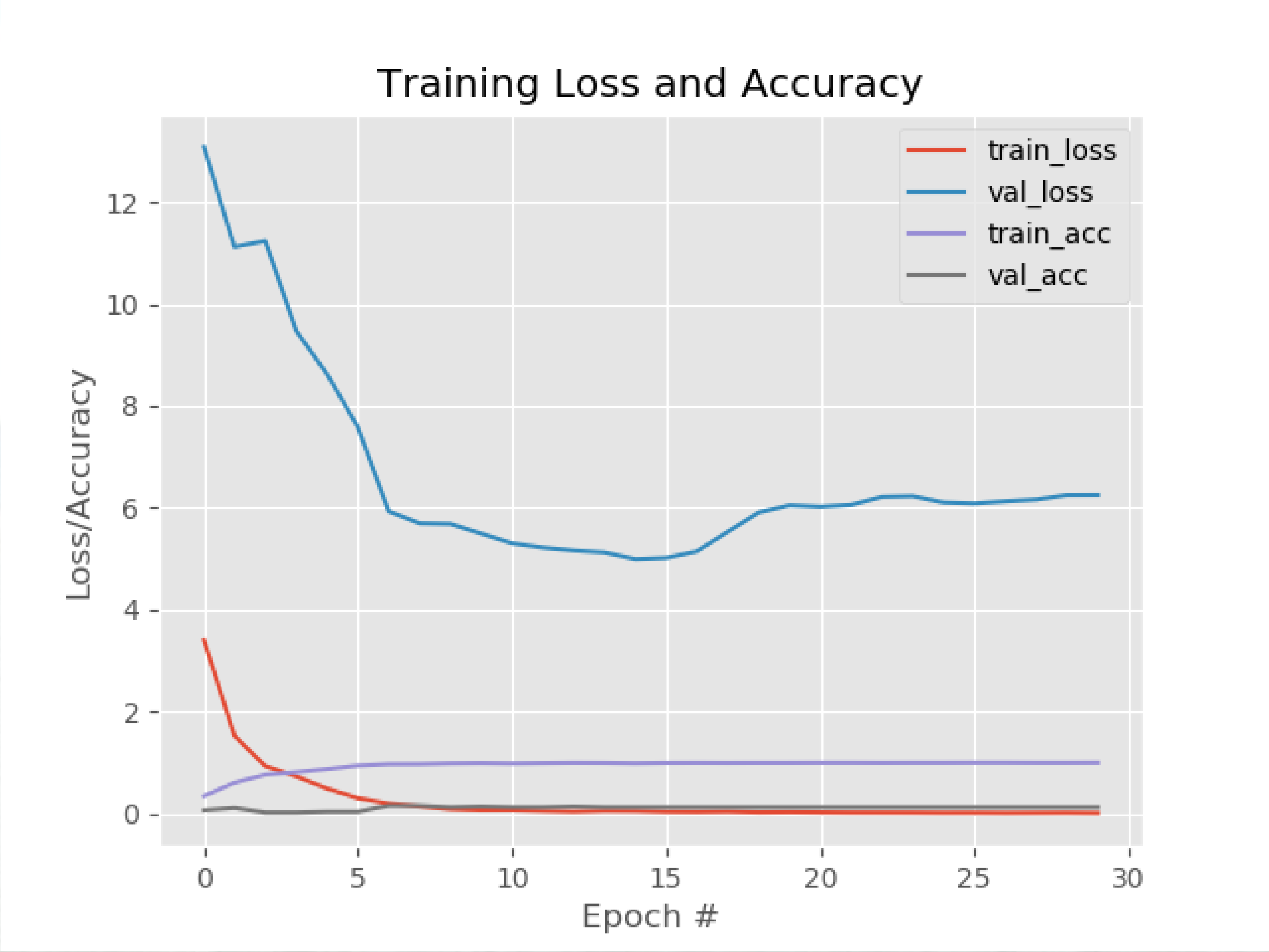


It may be worse...

pawel.kasprowski@polsl.pl

It may be worse...

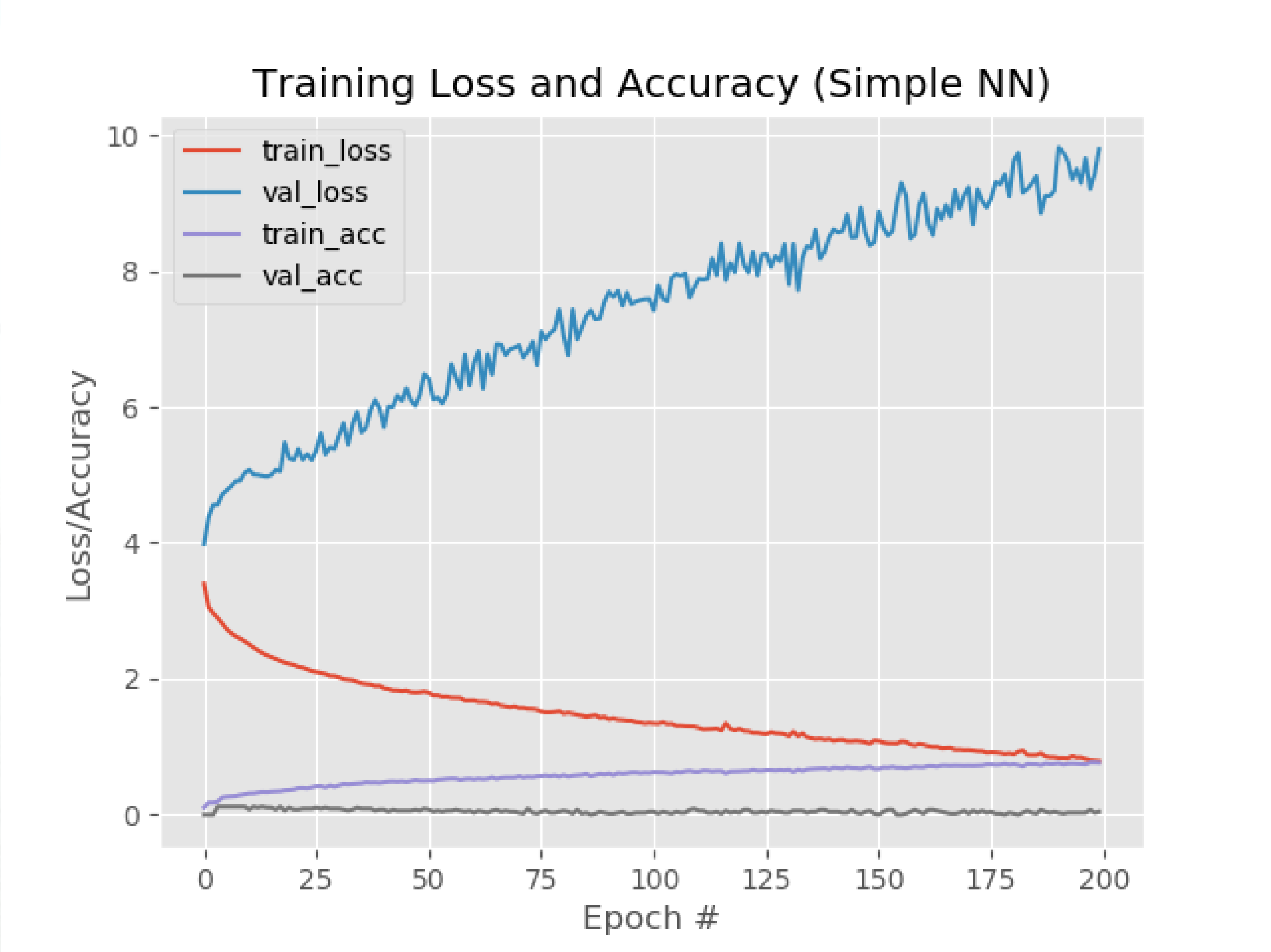
It may be worse...



pawel.kasprowski@polsl.pl

It may be even worse...

It may be even worse...

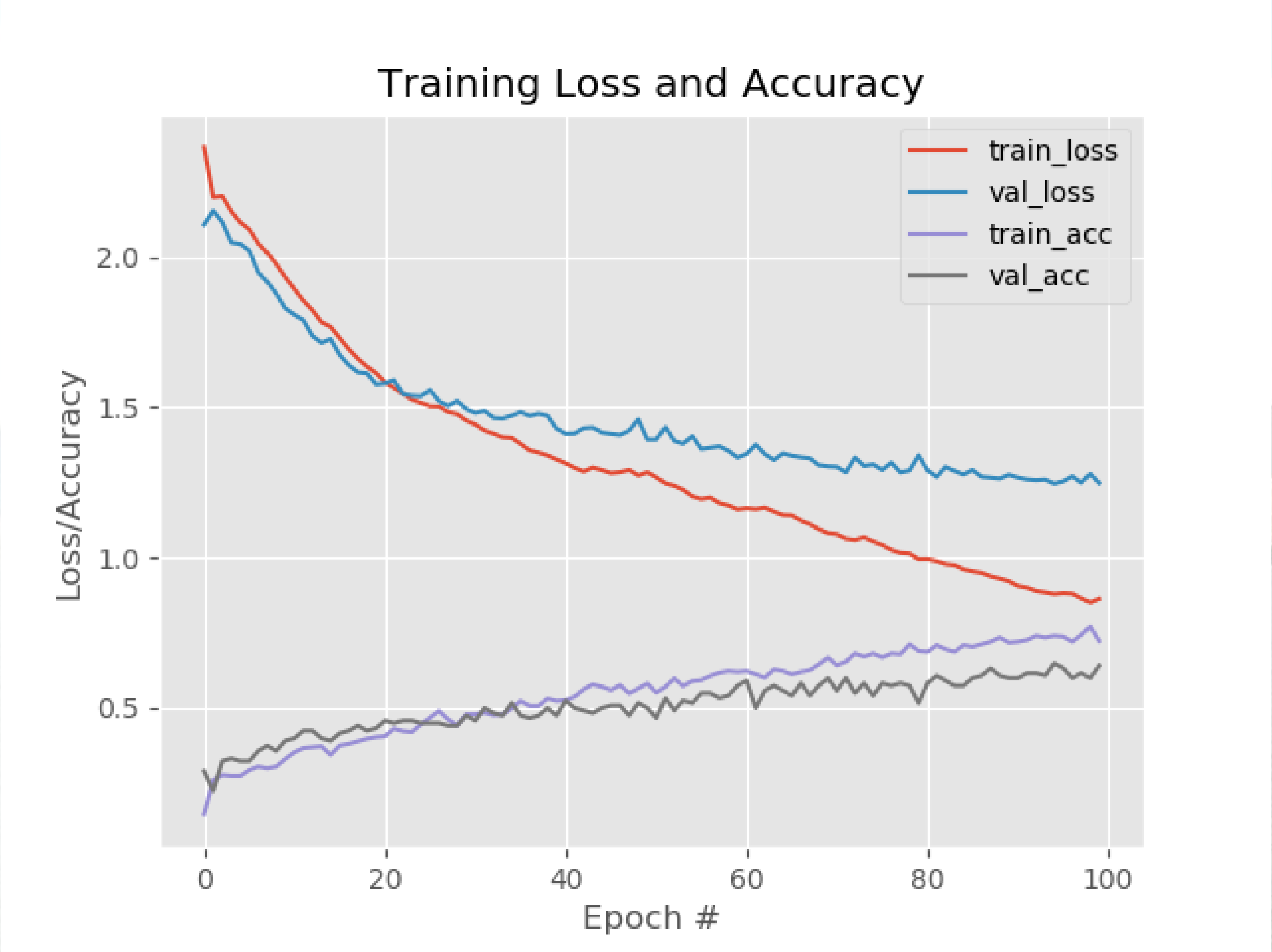


pawel.kasprowski@polsl.pl

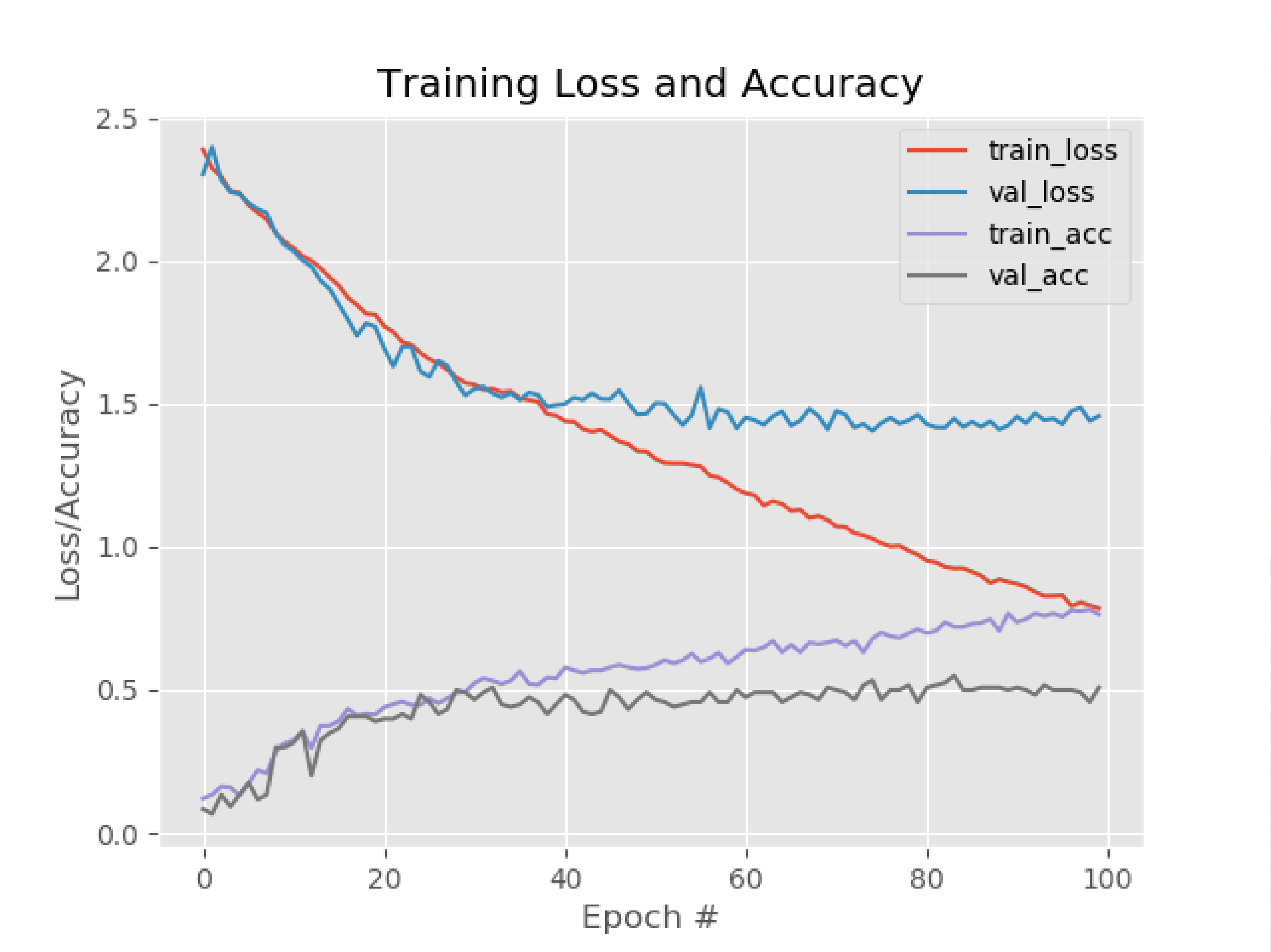
Sometimes even loss is not decreasing...

pawel.kasprowski@polsl.pl

So ,this is quite OK



This is worse



Comparison with Decision Tree

- Code for Decision Tree:

```
from sklearn.tree import DecisionTreeClassifier
treemodel = DecisionTreeClassifier()
treemodel.fit(trainSamples, trainLabels)
treeResults = treemodel.predict(testSamples)
```

- Results:
  - Comparable...

pawel.kasprowski@polsl.pl

Problems with neural networks

- Configuration is complicated (many hyperparameters)
  - layers, number of neurons
  - activation functions
  - optimizers
- Training is challenging
  - A lot of computations – a lot of weights
  - A lot of examples needed
- Training of many layers may fail
  - Vanishing gradient problem
  - Exploding gradient problem

pawel.kasprowski@polsl.pl

Question for today

- So why Deep Learning has become so popular?

- **Next part is about it!**