Can we harness Machine Learning for (storage ring) beam dynamics applications and for PETRA IV automation?

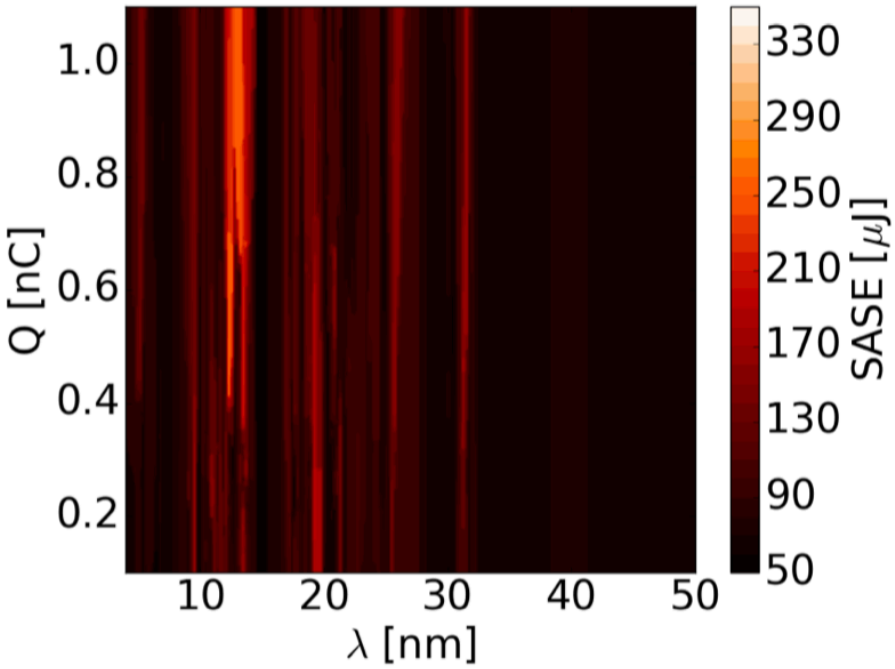
Ilya Agapov

M Betriebsseminar
Travemuende, 22 Feb 2019

Content

- Introduction — needs of 4G light sources
- Very high level “overview” of ML/Deep learning
- What can we expect to achieve with PETRA IV?

Some historical context



ML-assisted optimisation

Promise of faster and more efficient hyperparameter tuning

K-means prediction of FEL power at FLASH
Analysis of machine setting files for 2 years from *Agapov et al. DESY 17-054*

Data visualisation. Might be nice to have
But in no way critical

Needs of 4G sources

- ML first looked at in the FEL optimisation (and later failure prediction) context
  - FEL tuning has a direct impact on user operation and plays central role
  - FELs are newer machines, more adequate diagnostics and DAQ
- 3G storage rings much more stable in operation than FELs. Some optimisation is required but effort incomparable to an FEL. Most optimisation work (lifetime, Dynamic aperture, injection efficiency) has little direct impact on users
- Many labs including us (plan to) upgrade to 4G storage rings
- To proceed we should answer following questions (this talk is a first attempt)
  - 1.What difference can we expect in operation?
  - 2. Where does ML fit? — understand what to expect

Needs of 4G sources - difference to 3G

- Reliability demands grow (95% -> 99%)
- But machines are more sensitive with larger number of components
- We would like to meet availability goals and provide required beam-hours to users
- But at the same time we would like to keep doing accelerator physics and spend dedicated machine time on studies rather than machine setup
- This could only be successful if all standard procedures are highly automated
  - Startup of components (magnet cycling etc.)
  - Orbit correction
  - BBA
  - Optics measurement and correction
- Methods are well understood but (high-level) controls software not designed for autonomous operation

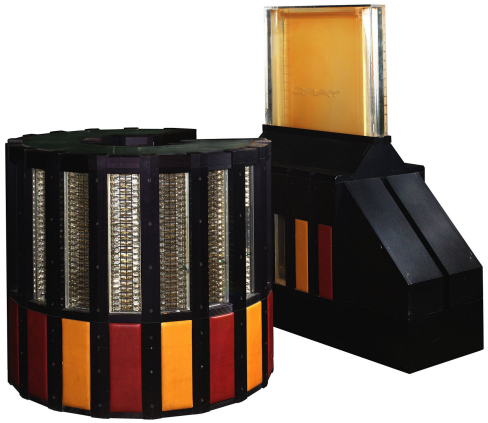
Big data and machine learning

- HEP pioneered BIG data (HERA, LHC) along with finance and insurance
- Internet businesses caught up and outperformed in 2010s
- Lots of pioneering AI research in the 60s and 70s, NN only a small subset
- Deep learning (large multilayered NN) largely impractical until about 10 years ago (don't listen to HEP physicist who did PhDs in the 90s)
- At the core of deep learning – statistical fitting but quantity becomes quality, in combination with big data (usually > 10000 distinct training examples) and computing power
- Key to success
  - Well-posed problem (from general AI to tangible problems in automation, advertisement, finance)
  - High performance computing (many orders of magnitude)
  - Ubiquitous data
  - Some improvement on algorithms resulting from sufficient experimentation
  - Economics (increase in 5% in performance/cost will wipe competitors out in most large-scale industries, not the case with light sources, FELs, or HEP)

Infrastructure – computing power and GPUs

5 orders of magnitude between most powerful computer in 1985 and a 5K consumer card (5 min training -> 1 year)

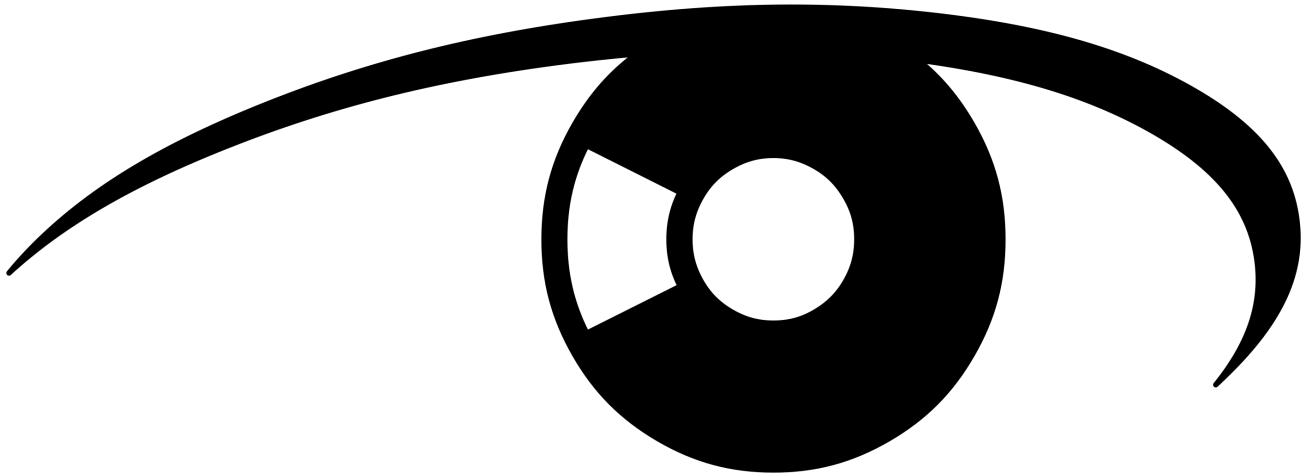
Cray-2 1.9 GFLOPS (1985)



	Tesla V100 for NVLink
PERFORMANCE with NVIDIA GPU Boost™	DOUBLE-PRECISION 7.8 teraFLOPS
	SINGLE-PRECISION 15.7 teraFLOPS
	DEEP LEARNING 125 teraFLOPS

Intel Pentium III (2000)
2 GFLOPS

Infrastructure – data collection



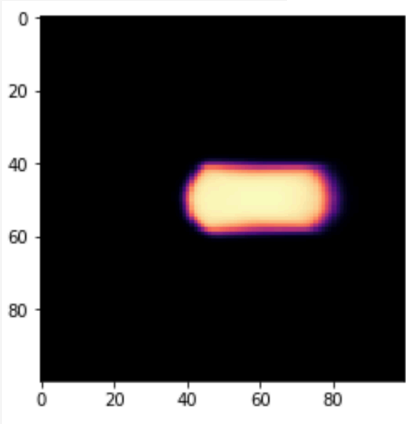
Infrastructure – Software stack

- ML industry-standard software stack is now freely available and easy to learn (data is the asset, not the software)
  - Python, pylab, Jupyter
  - Scikit-learn
  - Pandas
  - Tensorflow and keras
  - MATLAB used initially for ML application prototyping but now more or less completely abandoned

Primitive tensorflow example

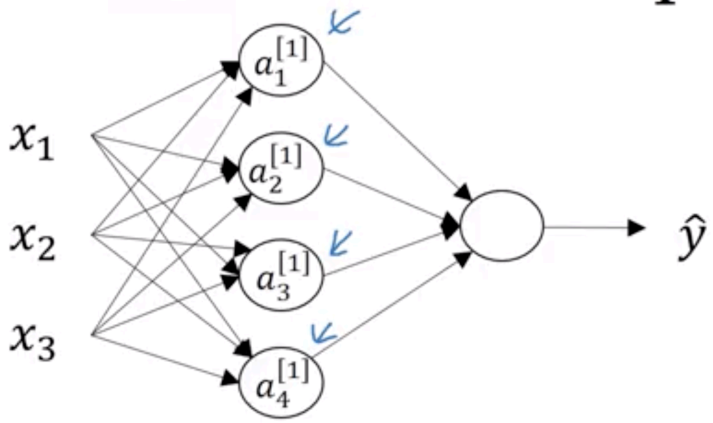
In [1]:
```
#0th order NN example: binary classification
import tensorflow
import tensorflow.keras as keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense, Dropout
from tensorflow.keras.utils import to_categorical


from pylab import *

#fixing a duplicate openmp dylib on mac???
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

model = Sequential()
model.add(Dense(16, input_dim=2, activation='relu'))
#model.add(Dropout(0.5))
model.add(Dense(8, activation='relu'))
#model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd',metrics=['accuracy'])

#primitive function : all inputs positive - 1, otherwise 0 (classification)
def f(x1,x2):
    if (x1**2 + x2**2)<1 or ((x1-2.0)**2 + x2**2)<1:
        return 0
    return 1
x_train = 8.*(np.random.random((50000,2)) - 0.5)
n_train = shape(x_train)[0]
y_train = [ f(x_train[i,0],x_train[i,1]) for i in range(n_train)]
#print(x_train)
#print(y_train)
#sys.exit(0)
model.fit(x_train,y_train, epochs=20)
x = np.array([[3,3],[2,-1],[-1,-1],[-10,-10],[0,0],[0,0.6]])
y=model.predict(x)
print(y)
```

Learns a simple 2D function



NN training is highly scalable

Neural Network Representation

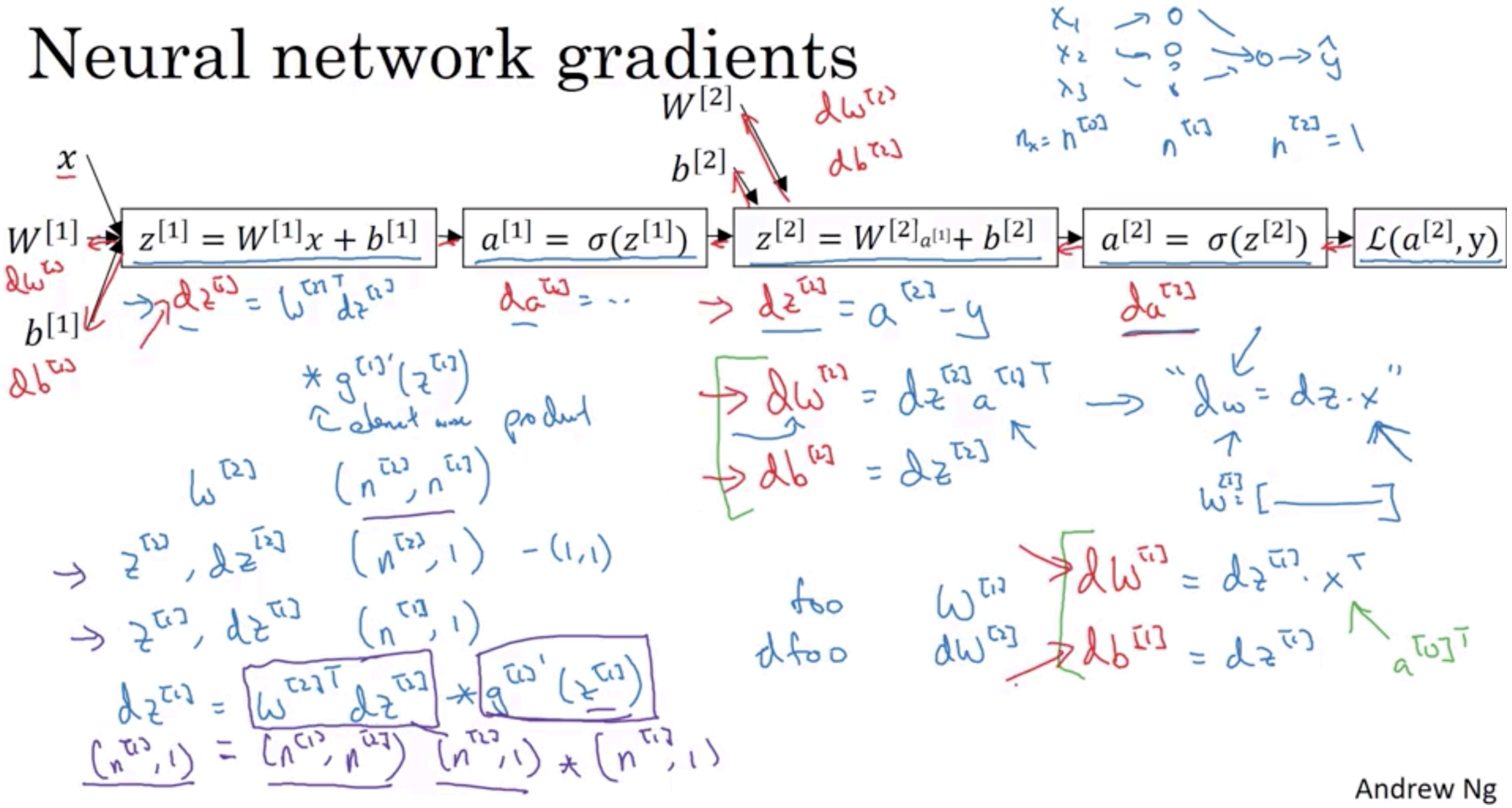


$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}_{(4,3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

Andrew Ng

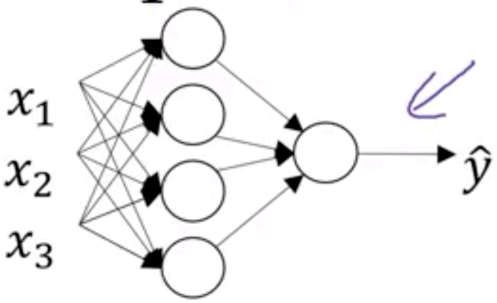
Neural network gradients



Andrew Ng

Training (backprop) — lots of parallelizable linear algebra

Recap of vectorizing across multiple examples



$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

```
for i = 1 to m
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$A^{[0]} \quad X = a^{[0]} \quad x^{(i)} = a^{[0](i)}$

$$Z^{[1]} = W^{[1]}X + b^{[1]} \quad \leftarrow \quad w^{[1]}A^{[0]} + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Andrew Ng

Can feed many training examples in parallel
Perfect scalability

ML and beam dynamics

- Although single-particle beam dynamics well developed, several directions still considered art, such as:
  - Multi-parameter matching in high dimensions (e.g. MBA. cell design)
  - Nonlinear aberrations (beyond simple ideas like low-order achromats, reducing sextuple strength, -I)
- A common line of reasoning — build fast “surrogate models” based on NN trained on simulated data

Sanity check — FODO

In [2]:
```
# try to learn FODO stability
import os,sys
from ocelot import *
from pylab import *
QF = Quadrupole(l=0.1, k1=0.1)
QD = Quadrupole(l=0.1, k1=-0.14)
D = Drift(l=1.0)
fodo = (QF,D,QD,QD,D,QF)
lat = MagneticLattice(fodo)
```

```
n_train = 10000

x_train = np.zeros([n_train,3])
y_train = np.zeros([n_train,1])


for i in range(n_train):
    QF.k1 = np.random.rand()
    QD.k1 = -np.random.rand()
    D.l = 2.0 * np.random.rand()

    x_train[i,0] = QF.k1
    x_train[i,1] = QD.k1
    x_train[i,2] = D.l
    lat.update_transfer_maps()
    tws = twiss(lat, Twiss())
    if tws is None:
        y_train[i,0] = 0
    else:
        y_train[i,0] = 1
```

```
import keras
from keras.models import Sequential
from keras.layers import Input, Dense, Dropout
from keras.utils import to_categorical
from pylab import *

#fixing a duplicate openmp dylib on mac???
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'

model = Sequential()
model.add(Dense(64, input_dim=3, activation='relu'))
#model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='sgd',metrics=['accura
model.fit(x_train,y_train, epochs=30, batch_size=16)
```

```
Using TensorFlow backend.
```

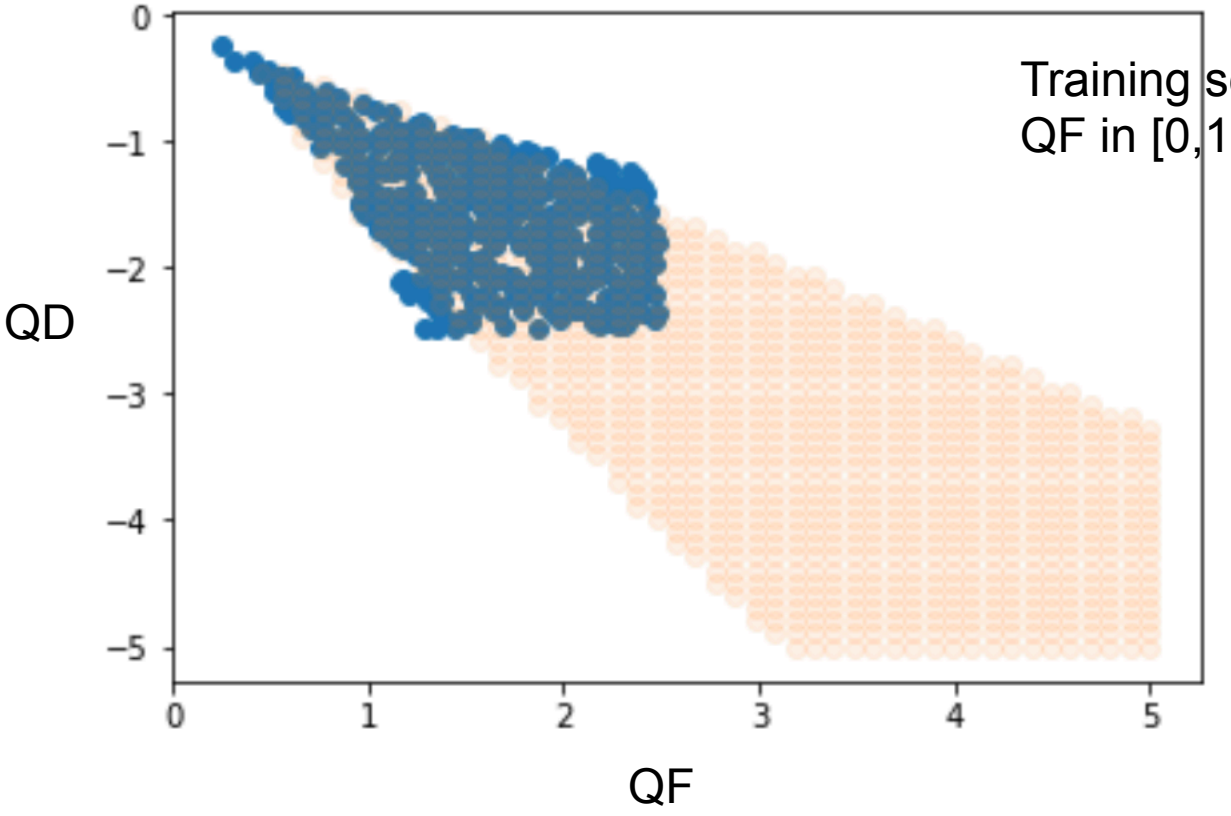
```
Epoch 1/30
10000/10000 [==============================] - 2s 211us/step - loss: 0.325
Epoch 2/30
10000/10000 [==============================] - 2s 153us/step - loss: 0.2748 - acc: 0.9313
Epoch 3/30
10000/10000 [==============================] - 2s 152us/step - loss: 0.2614 - acc: 0.9313
```

```
n_test = 5000

x_test = np.zeros([n_test,2])
y_test = np.zeros([n_test,1])

D.l = 3.9

for i in range(n_test):
    QF.k1 = 5*np.random.rand()
    QD.k1 = -5*np.random.rand()

    x_test[i,0] = QF.k1
    x_test[i,1] = QD.k1
    lat.update_transfer_maps()
    tws = twiss(lat, Twiss())
    if tws is None:
        y_test[i,0] = 0
    else:
        y_test[i,0] = 1
```

Sanity check — FODO

Stability diagram check on the test set
QF in [0,5] QD in [-5,0] L=3.9

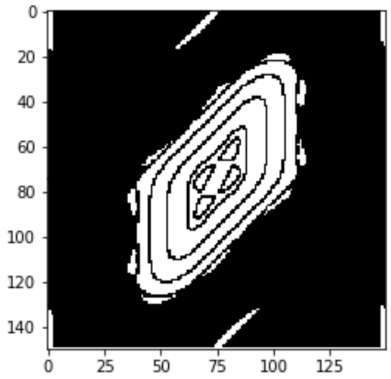
Training set was
QF in [0,1] QD in [-1,0] L in [0,2]

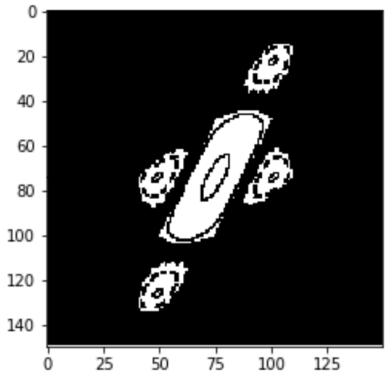


The trained NN generalises surprisingly well beyond training set parameters!
However only within a certain range, and validity range needs checks
Practical application not clear — some real-time FPGA-based response matrix calculation for non-linear, time-dependent situations etc. ?

Sanity check — Henon map

2-layer NN trained to generate Henon map based on nonlinearity parameter as input

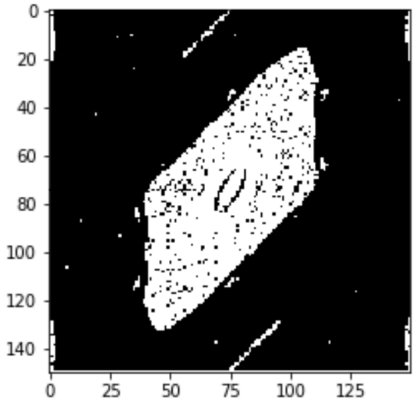


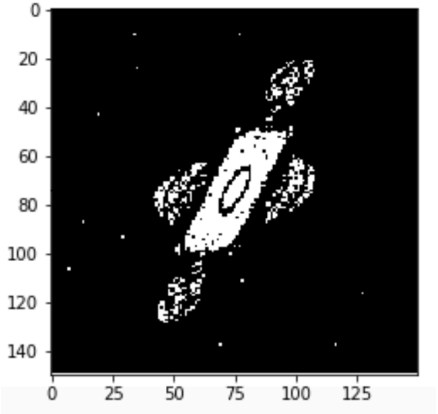


Phase portraits

NN can generate images which look like phase portraits also capturing rough dependency on the non-linearity parameter
It is however very hard to train the NN to reproduce fine features
Practical application not clear (unless sold as art)

Generated by NN





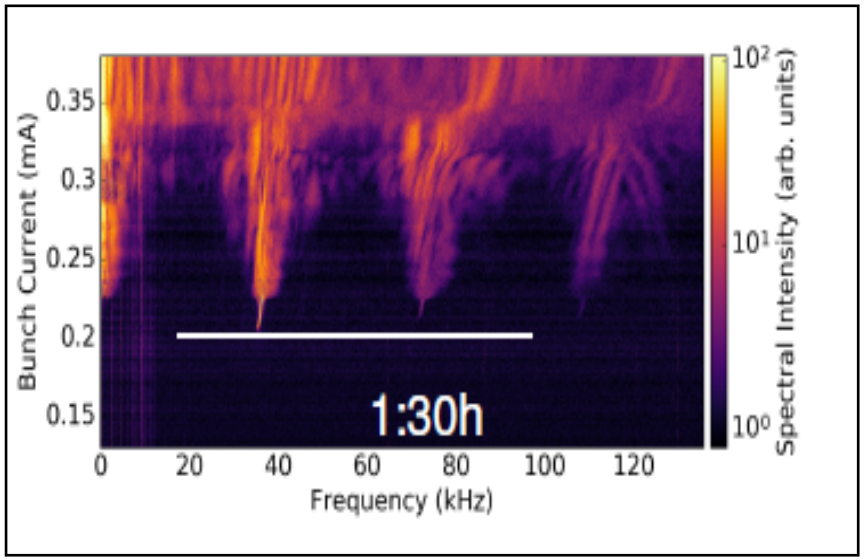
Portrait of Edmond Belamy
Generated by GAN
Sold at Christies for $432,500, 2018

Single-particle dynamics perspectives

- Deep learning might be useful for single-particle dynamics analysis
- The key is however in computation speed, not directly related to ML
- Questions of calculating DA, MA, low-dimensional sextuple strength optimisation etc. are now routinely solved within minutes to hours on ~1000-core machines (required lengthy dedicated studies in the 1990s)
- We could gain another 1-2 orders of magnitude in performance with tracking on GPU — GPU version of *Sixtrack* under development at CERN and tests with PETRA IV lattice ongoing
- With super-massive computations NN might be a way to represent simulated results along with other statistical tools

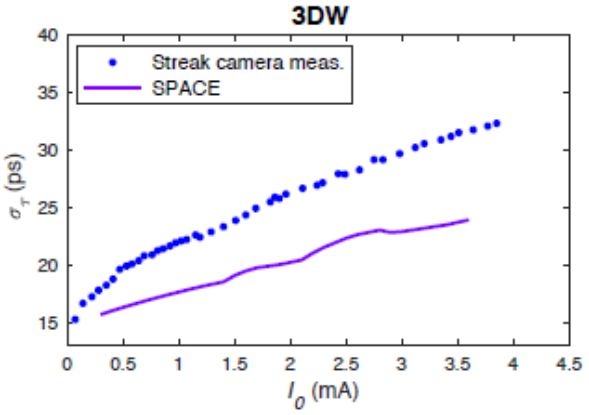
ML and collective effects

- One of less understood areas of storage rings is high-charge limitations
- Relevant for PETRA IV timing mode
- Relevant for many lower-energy machines pushing for higher current (500 mA NSLS-II, MAX-IV) or coherent radiation modes
- Large-scale simulation/measurement campaigns + model fitting could be useful (some work done at KIT)



Experimental microbunching instability studies at KARA/KIT

NLSL-II bunch lengths vs. intensity
from A. Blednych, BCTLESR 2019



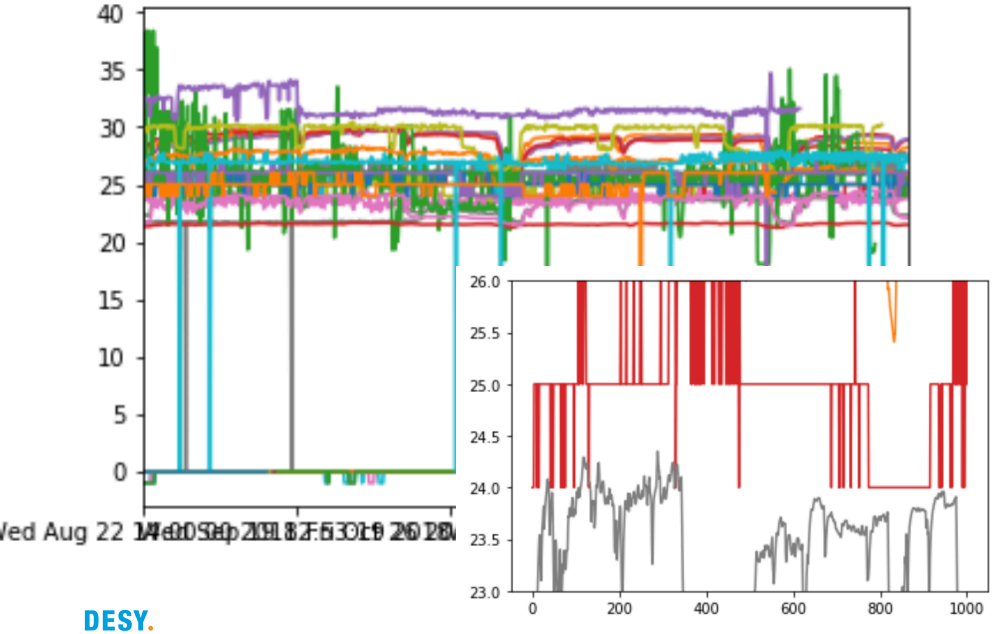
- We believe, the instability thresholds can be detected from the bunch length measurements with a streak camera resolution better than $0.5ps$.

ML and optics measurement

- Straightforward applications of optics measurement (especially all quick methods like multi turn and AC) often performs poorly
- Need some expertise to process data, e.g. remove bad BPM readings
- ML techniques are looked at to speed that up
- We collaborate with the CERN group on optics measurement and further machine studies at PETRA III planned

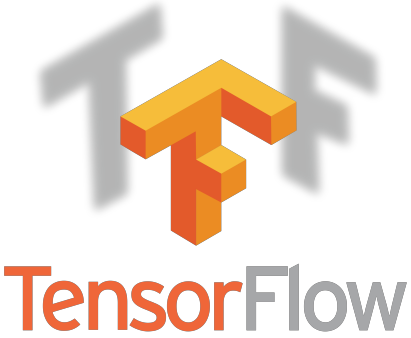
Big data and operational statistics

- A typical activity is to figure out what correlates to what through archive mining and try to improve things
- Problems:
  - Unprocessed data from P3 archive mostly garbage (G. in g. out for any algorithm).
  - Lengthy manual processing required to arrive at a meaningful dataset
  - Amount and quality of data after processing could be surprisingly low in comparison to the raw data



Typical pull from archive
All temperature sensors, Aug 22-Nov30 2018
Fraction of channel data missing (25%)
Unmatched timestamps
Data archived on thresholds
Bad readings/datapoints

Prototyping ML workflow — software stack



PyTine

Conclusion and Outlook

- Exciting opportunities in ML, but need to boost controls, diagnostics (monitoring) and expertise in “data science” to make use of it
- Expect that the outcome is unexpected, work should be well aligned with lower-risk R&D and more critical machine issues
- For next generation (PETRA IV) need to completely rework data acquisition and curation
- ML should be a part of larger automation campaign
- The goal (and challenge) of automation is to free personnel and machine time for studies given high pressure on availability and reliability
- Activities being launched to address that (some prototyping ongoing; postdoc hiring in progress. PETRA IV HLC group launched from late March 2019)
- Some level of collaboration pursued with other labs (BESSY, KIT/KARA, SLAC/ SPEAR3, CERN, MAXIV)
- In ML-centric programming paradigm **data is the most important asset**
- Human brain 10^{11} neutrons. Deep nets 10^{2} -10^{3} . Don’t expect too much intelligence (from NN)