Study of Orbit Correction by Neural Networks In Taiwan Photon Source

Mau-Sen Chiu

2024/03/07

Beam Dynamics Group, NSRRC

Abstract

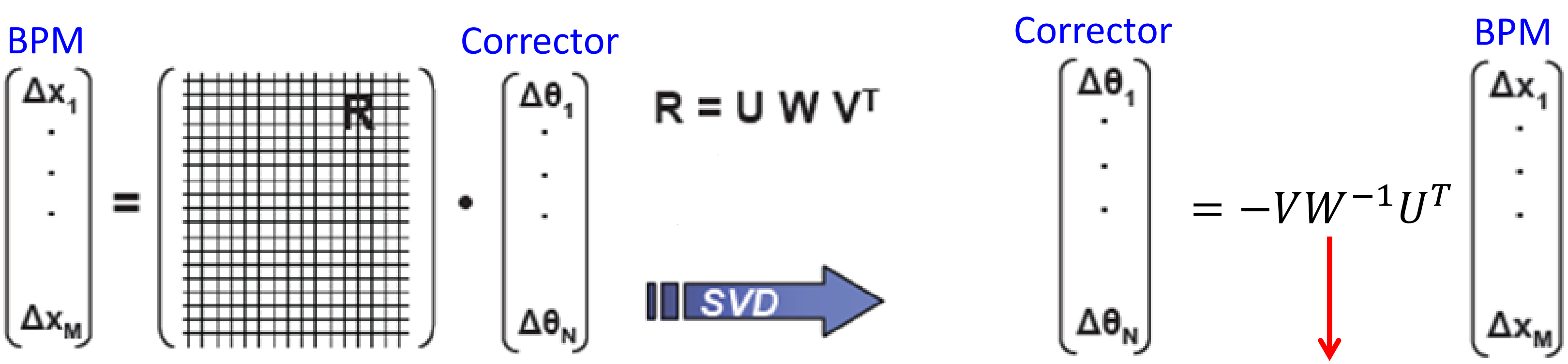
The Taiwan Photon Source is designed as a 3 GeV synchrotron light source, encompassing a 518.4 m circumference. The lattice structure of the storage ring consists of 24 Double-Bend Achromat cells. The storage ring is equipped with 172 BPMs and 72/96 correctors to do orbit correction and control in horizontal and vertical planes, respectively. The correction algorithm uses a measured orbit response matrix and singular value decomposition (SVD) algorithm at present. This traditional method is rooted in physics and well-established principles of beam dynamics in particle accelerators. In this study, we use neural network model to do orbit correction. The training data for the neural networks is generated by accelerator toolbox (AT).

Orbit Correction by SVD (Traditional)

1. Establish reference orbit (Target Orbit)
2. Measure Orbit Response Matrix R between BPMs and correctors.
3. Apply SVD to decompose R, and select the number of singular values
4. Measure actual orbit - check for bad readings
5. Compute difference orbit
6. Compute corrector strength from $\Delta\theta = -V \cdot diag(1/w_j) \cdot (U^T \cdot \Delta X)$

   ΔX: Difference Orbit

7. Check for corrector currents in reasonable range
8. Apply corrector currents

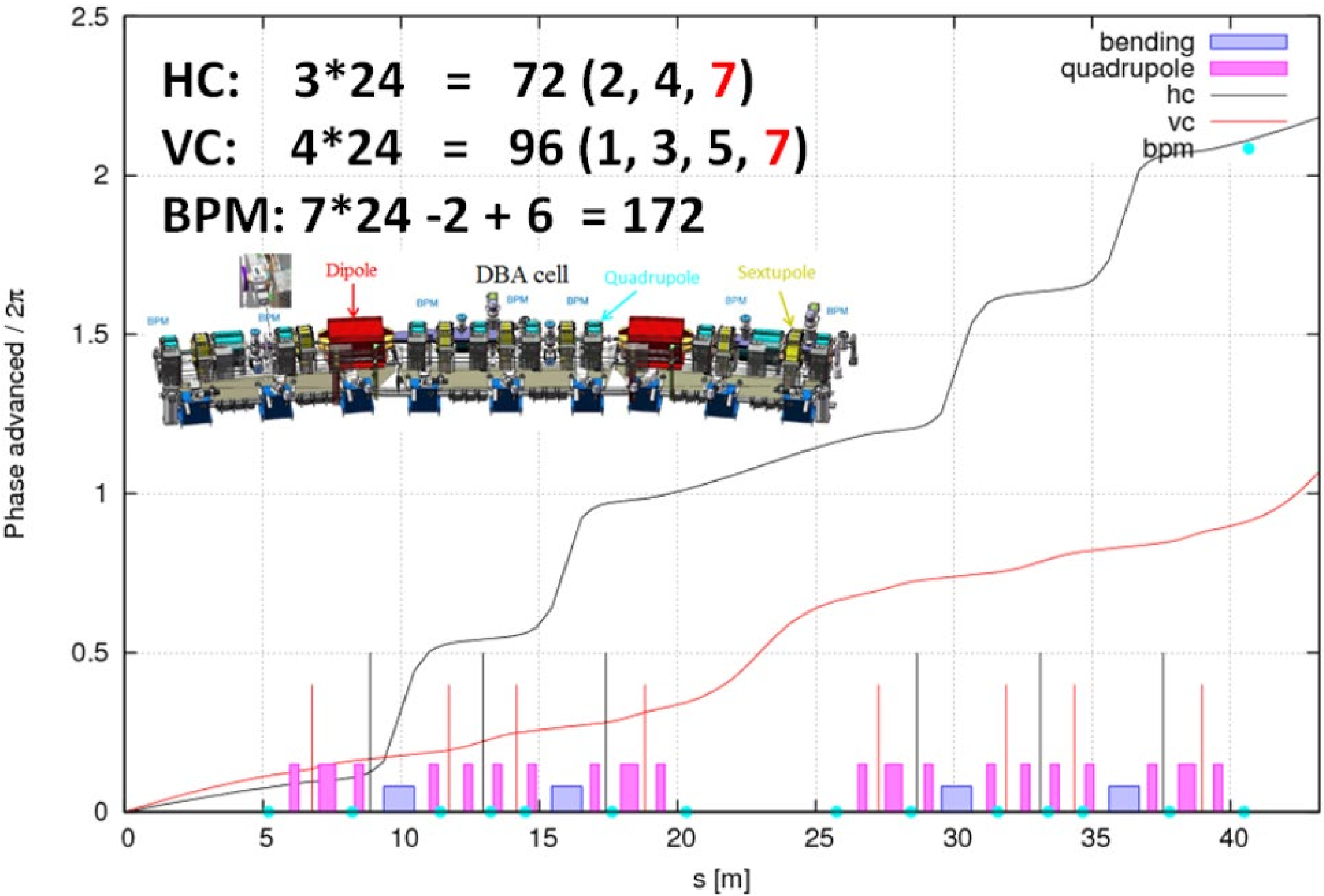


Singular values

$$R_{ij} = \frac{\sqrt{\beta_i \beta_j}}{2\sin \pi\nu} \cos(|\phi_i - \phi_j| - \pi\nu)$$

It work with difference orbit and corrector changes rather than the absolute orbit and corrector values.

Orbit Correction Scheme in TPS Storage Ring



Simulation of Orbit Correction by Neural Networks

■ Training:

1. 72 horizontal correctors (HC) strengths within +/- 2.5 μrad are randomly assigned and then get orbits (172 BPMs) by AT: repeat 3000 times.
2. Build Model by keras: input layer is 172 nodes, hidden layer is 172 nodes, output layer is 72 nodes.
3. Train the model with AT simulation data.
4. Save the well-trained model of the neural networks.

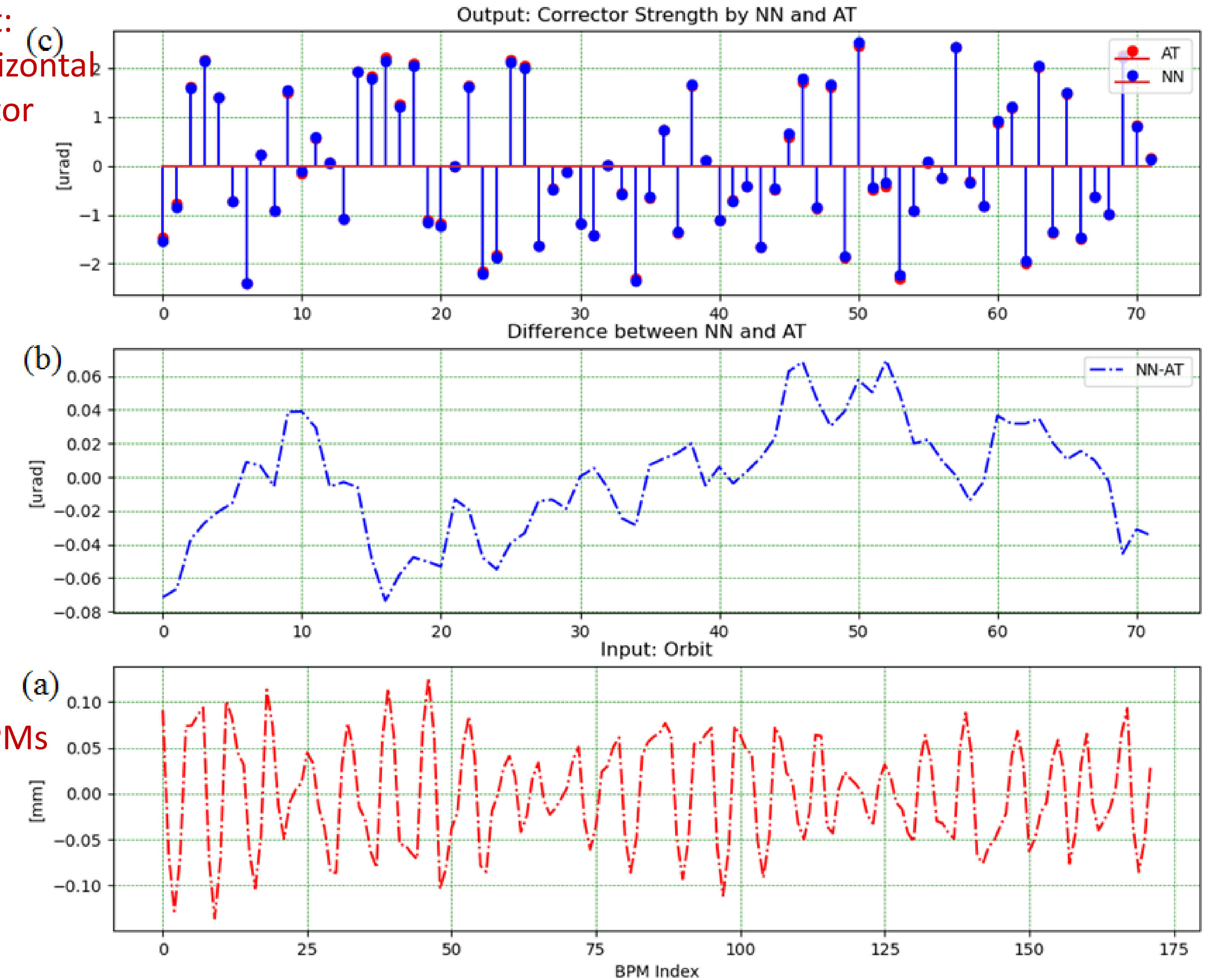
■ Test:

5. Generate many orbit distortions by randomly shifting 249 quadrupoles within +/- 3 μm in horizontal plane.
6. Load the well-trained model of the neural networks
7. Input the orbit distortions to the neural networks to get the predicted corrector strength
8. Use the predicted corrector strength to correct the orbit distortion generated by quadrupole misalignment
9. Iterate step 7 ~ 8: 3 times

Training Neural Networks (NN)

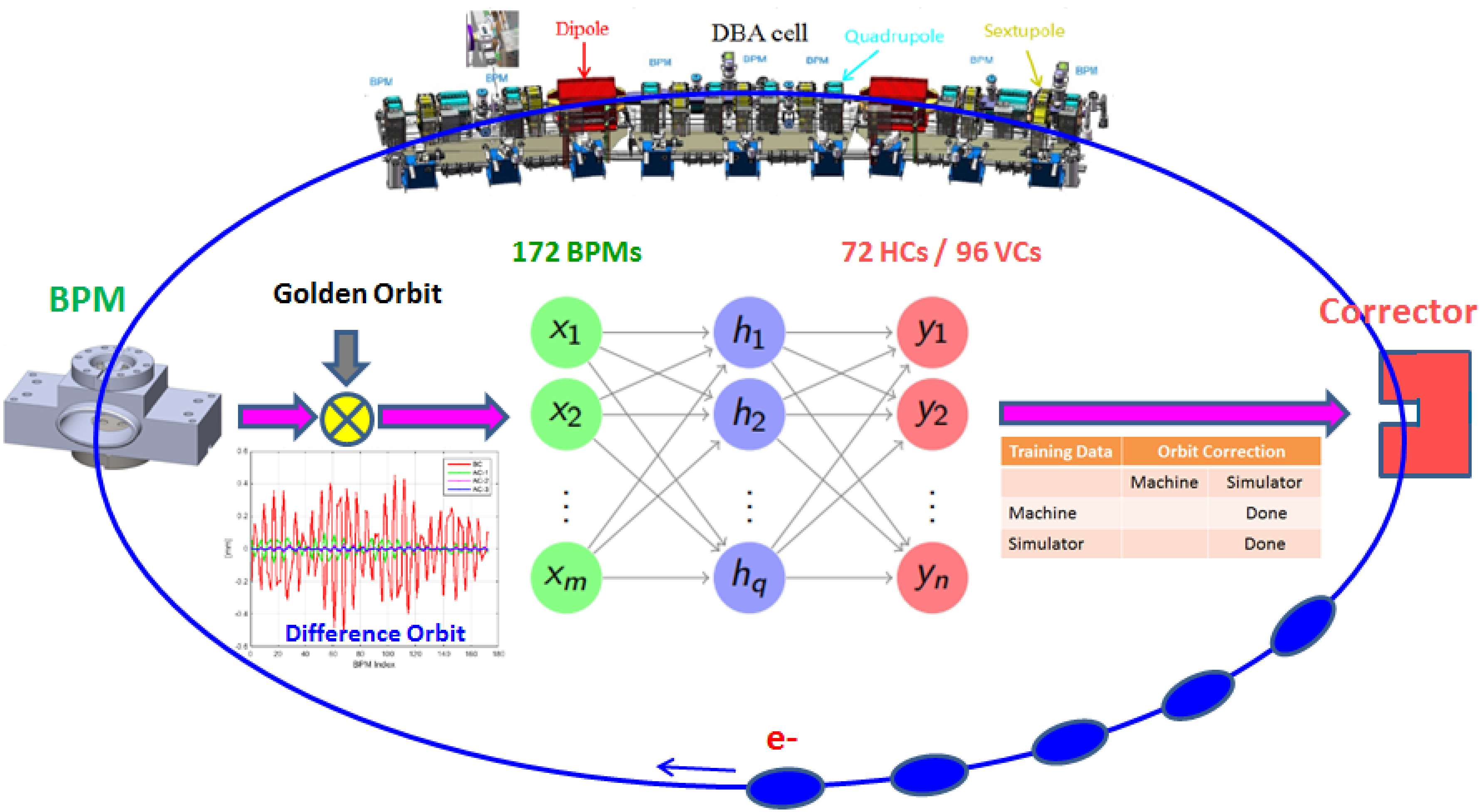
Output:
72 Horizontal
corrector

Input:
172 BPMs



Simulation of Orbit Correction by Neural Networks

In TPS Storage Ring

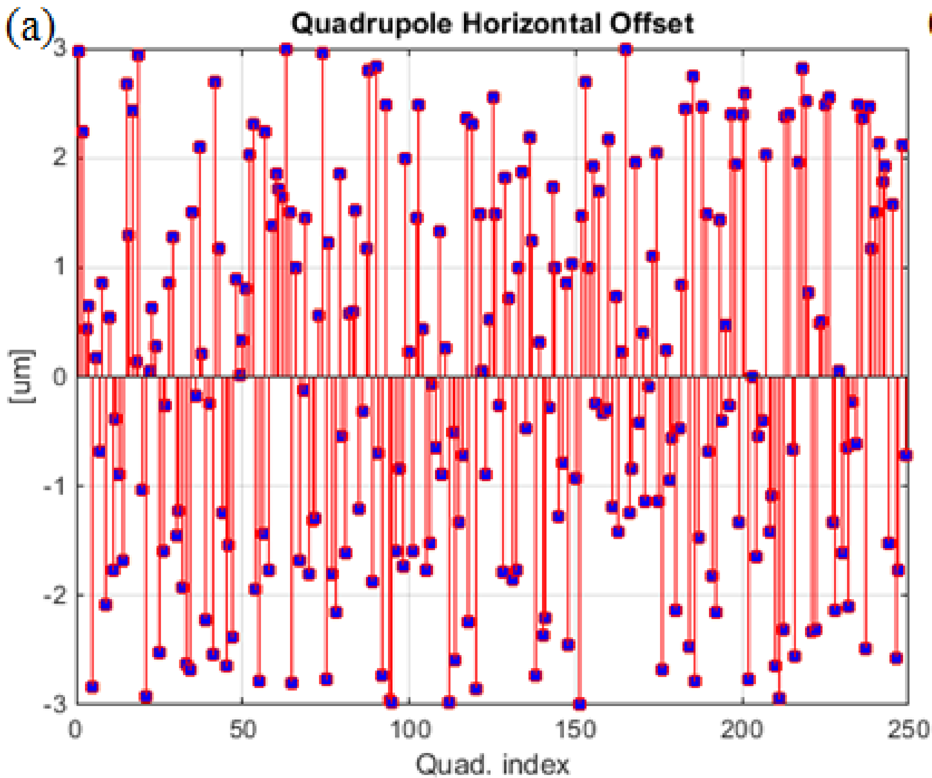


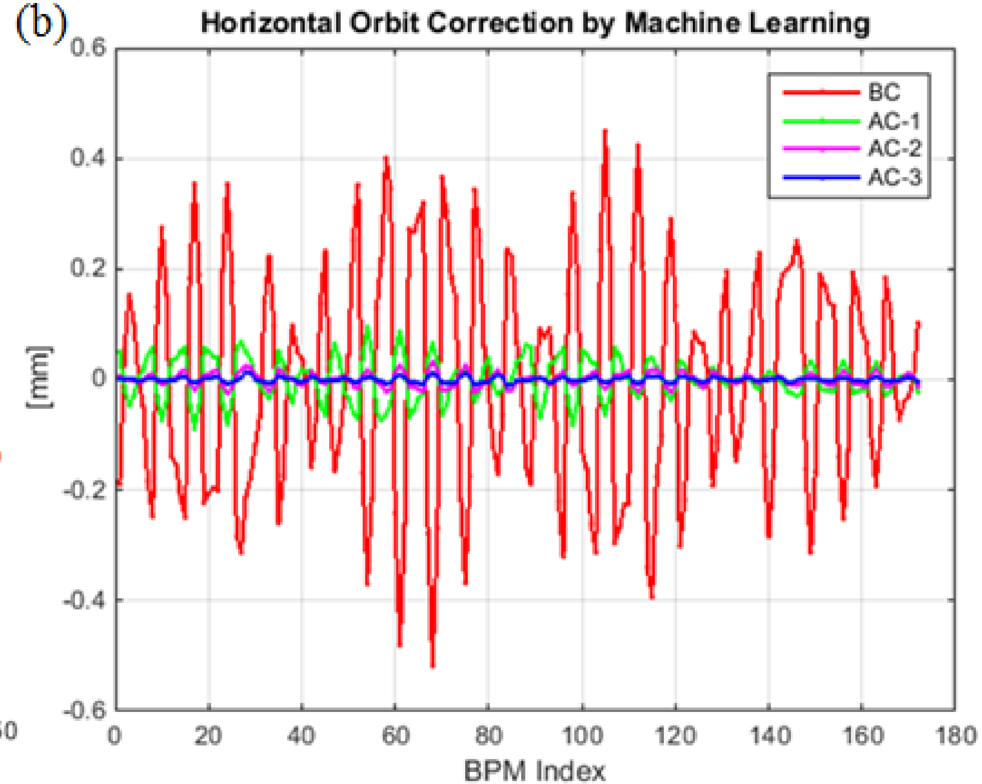
Training Data	Orbit Correction	
	Machine	Simulator
Machine		Done
Simulator		Done

Misalignment quantities of 249 quadrupole magnets within +/- 3 µm to generate orbit distortion in TPS storage ring simulated by AT.

Simulation of Orbit Correction by Neural Networks

In TPS Storage Ring





Misalignment quantities of 249 quadrupole magnets within +/- 3 μm to generate orbit distortion in TPS storage ring simulated by AT.

Orbit correction by neural network: Red is the orbit before correction (BC), green, magenta, and blue are the orbit after correction (AC), iterate 3 times (AC-1, AC-2, AC-3).

Demonstration

APPENDIX

17th Int. Conf. on Acc. and Large Exp. Physics Control Systems ICALEPCS2019, New York, NY, USA JACoW Publishing
ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEPHA138

ORBIT CORRECTION WITH MACHINE LEARNING TECHNIQUES AT THE SYNCHROTRON LIGHT SOURCE DELTA

115 m, 1.5 GeV

D. Schirmer*
Center for Synchrotron Radiation (DELTA), TU Dortmund University, Germany

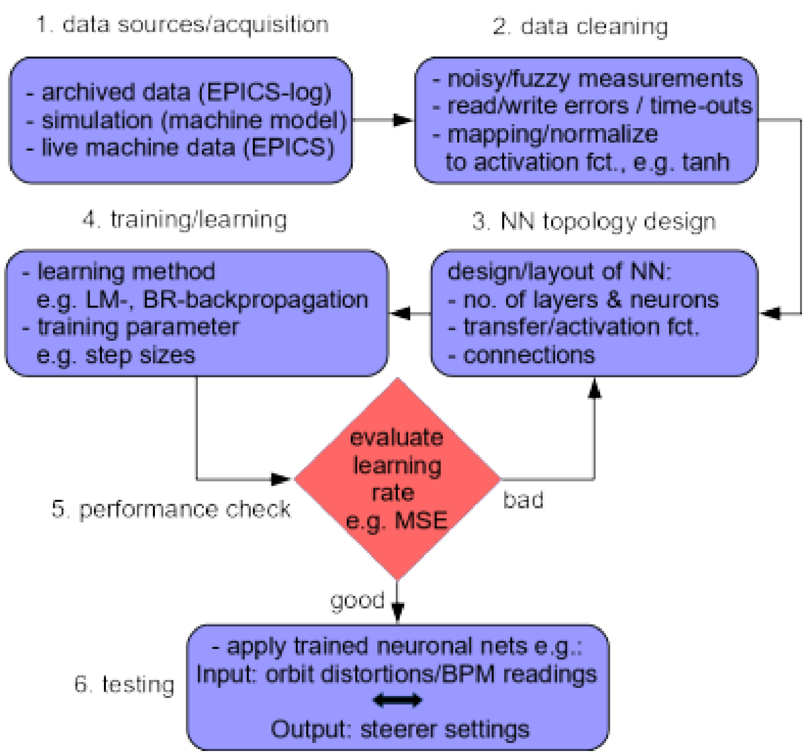


Figure 1: Development stages for an ML-based OC.

30 HC (±200 to 300 mA), 54 BPM, 1500 data sets

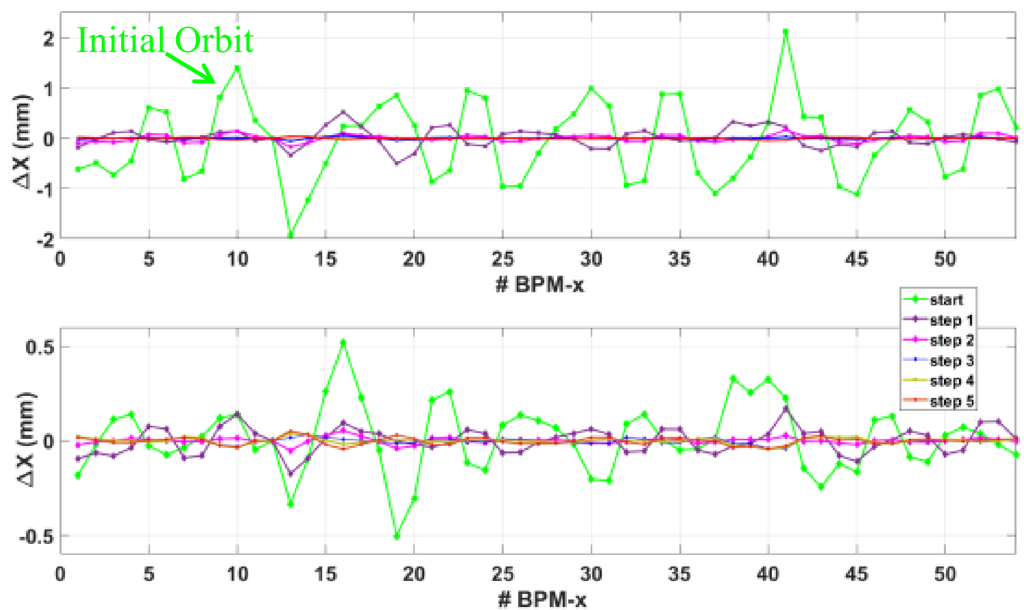
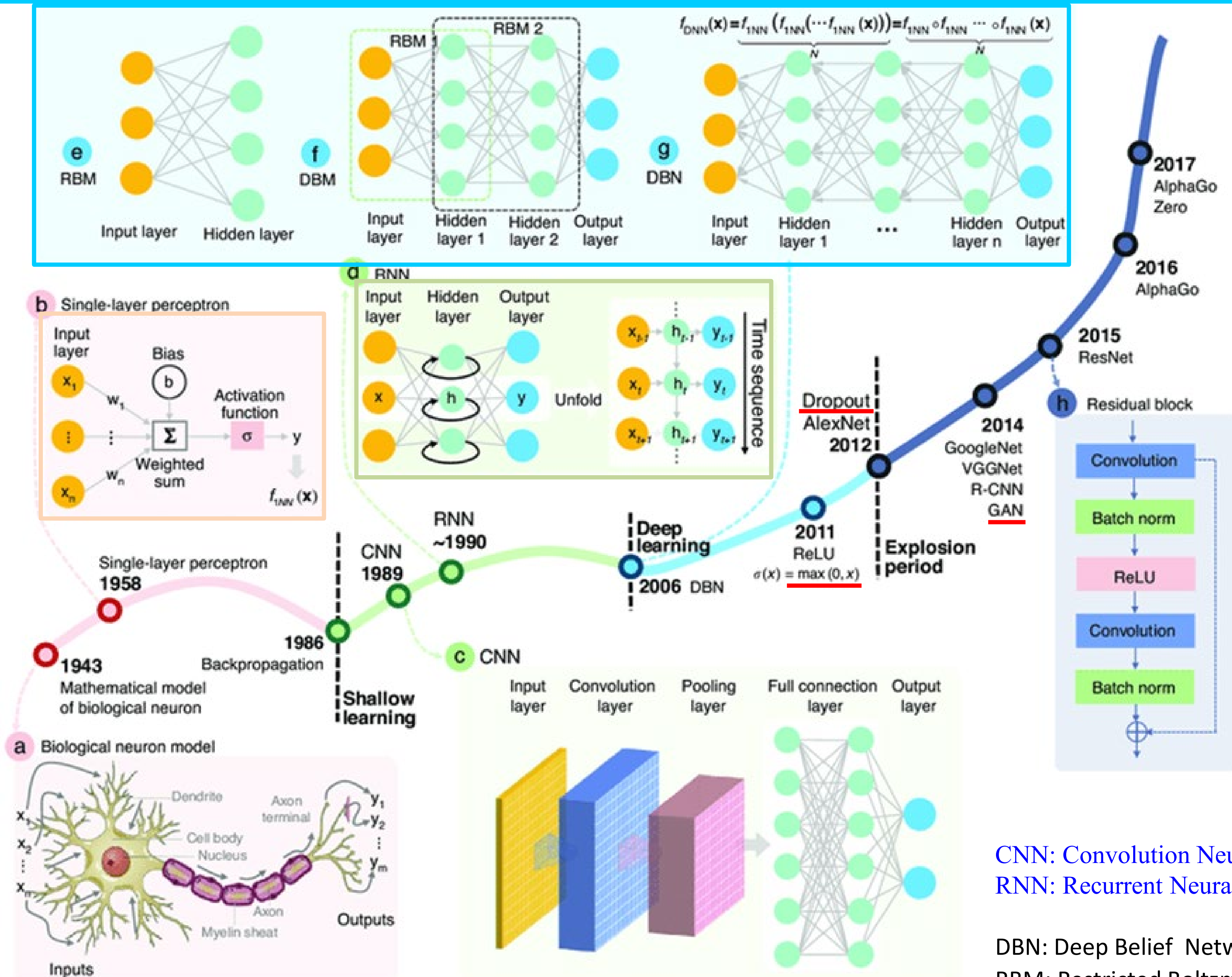


Figure 6: Iterative application of the pretrained FFNN referred to the previously corrected orbit, starting from a randomly disturbed orbit (start). After 3 successive correction steps, an error of < 200 µm was achieved.

History of Neural Networks



CNN: Convolution Neural Network
RNN: Recurrent Neural Network

DBN: Deep Belief Network
RBM: Restricted Boltzmann Machine

Ref: Deep learning in optical metrology: a review, Chao Zuoet al. Light: Science & Applications (2022) 11:39

Popular Deep Learning & Software

TABLE 2 | List of popular deep learning models, available learning algorithms (unsupervised, supervised) and software implementations in R or python.

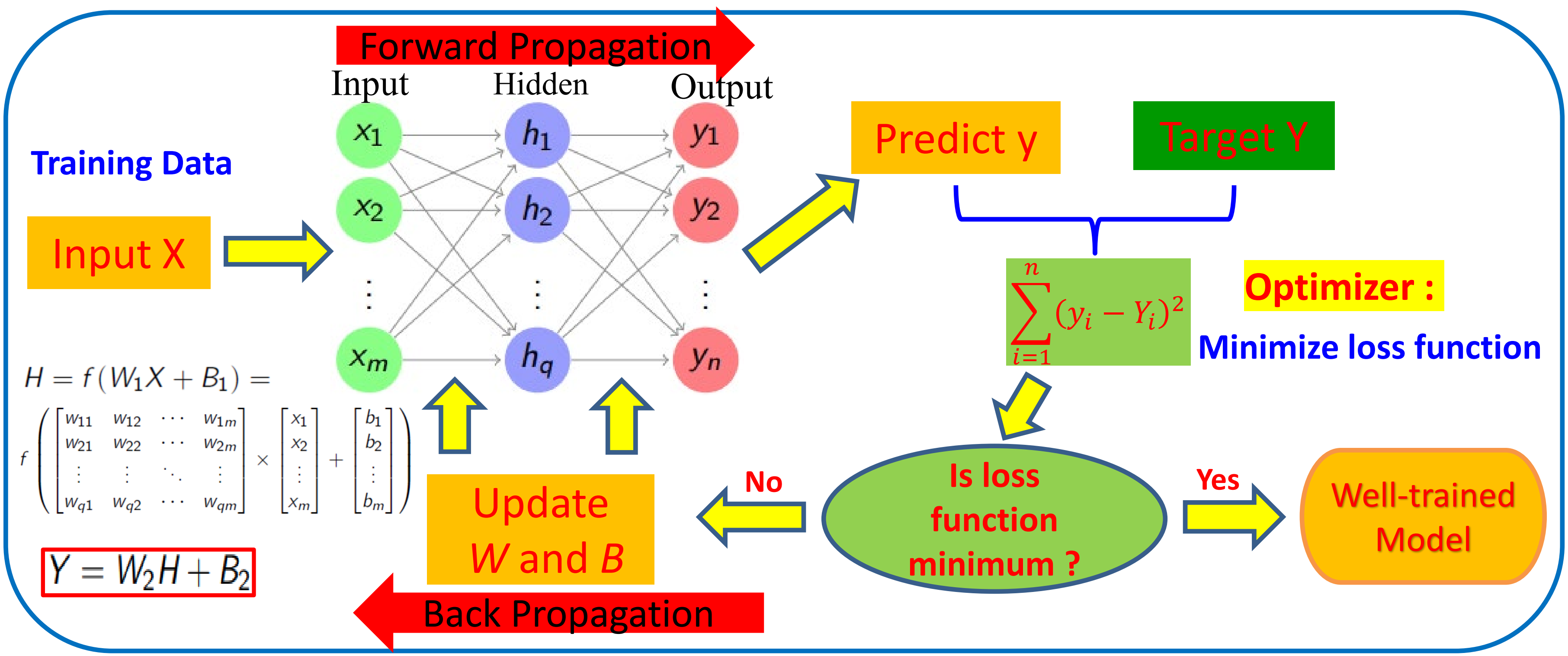
Model	Unsupervised	Supervised	Software
Autoencoder	✓		Keras (Chollet, 2015), R: dimRed (Kraemer et al., 2018), h2o (Candel et al., 2015), RcppDL (Kou and Sugomori, 2014)
Convolutional Deep Belief Network (CDBN)	✓	✓	R & python: TensorFlow (Abadi et al., 2016), Keras (Chollet, 2015), h2o (Candel et al., 2015)
Convolutional Neural Network (CNN)	✓	✓	R & python: Keras (Chollet, 2015) MXNet (Chen et al., 2015), Tensorflow (Abadi et al., 2016), h2O (Candel et al., 2015), fastai (python) (Howard and Gugger, 2018)
Deep Belief Network (DBN)	✓	✓	RcppDL (R) (Kou and Sugomori, 2014), python: Caffee (Jia et al., 2014), Theano (Theano Development Team, 2016), Pytorch (Paszke et al., 2017), R & python: TensorFlow (Abadi et al., 2016), h2O (Candel et al., 2015)
Deep Boltzmann Machine (DBM)		✓	python: boltzmann-machines (Bondarenko, 2017), pydbm (Chimera, 2019)
Denoising Autoencoder (dA)	✓		Tensorflow (R, python) (Abadi et al., 2016), Keras (R, python) (Chollet, 2015), RcppDL (R) (Kou and Sugomori, 2014)
Long short-term memory (LSTM)		✓	rnn (R) (Quast, 2016), OSTSC (R) (Dixon et al., 2017), Keras (R and python) (Chollet, 2015), Lasagne (python) (Dieleman et al., 2015), BigDL (python) (Dai et al., 2018), Caffe (python) (Jia et al., 2014)
Multilayer Perceptron (MLP)		✓	SparkR (R) (Venkataraman et al., 2016), RSNNS (R) (Bergmeir and Benítez, 2012), keras (R and python) (Chollet, 2015), sklearn (python) (Pedregosa et al., 2011), tensorflow (R and python) (Abadi et al., 2016)
Recurrent Neural Network (RNN)		✓	RSNNS (R) (Bergmeir and Benítez, 2012), rnn (R) (Quast, 2016), keras (R and python) (Chollet, 2015)
Restricted Boltzmann Machine (RBM)	✓	✓	RcppDL (R) (Kou and Sugomori, 2014), deepnet (R) (Rong, 2014), pydbm (python) (Chimera, 2019), sklearn (python) (Chimera, 2019), Pylearn2 (Goodfellow et al., 2013), TheanoLM (Enarvi and Kurimo, 2016)

Ref: An Introductory Review of Deep Learning for Prediction Models With Big Data, Frontiers in Artificial Intelligence, 28 Feb. 2020

Training by Backpropagation

How to determine weights and bias ?

- Initialize weights "randomly"
- For all training epochs
  - for all input-output in training set
    - using input and compute output : forward propagation
    - compare computed output with training output -> calculate loss function
    - update weights (backpropagation) to improve output -> minimize loss function
  - if accuracy is good enough, stop



https://www.analyticsvidhya.com/blog/2016/08/evolution-core-concepts-deep-learning-neural-networks/

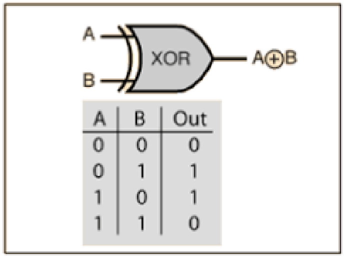
Workflow of Neural Networks

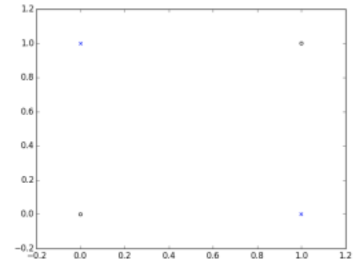
- **Software Packages**: Keras, Tensorflow, Python.
- **Data collection**: Scaling and normalizing data, then splitting data into training, validation and test sets.
- **Build a neural network**: Select an appropriate neural network architecture (e.g. feedforward, recurrent, convolution, *et al*) based on problem type (e.g. regression, classification, *et al*.), and assign the number of layers, neuron number in each layer, activation function (e.g. sigmoid, tanh, ReLu, *et al*.).
- **Compile the Model**: Specify the loss function (e.g. mean square error, *et al*.), optimizer (e.g. adam, sgd, *et al*.) that adjusts the model's weights and bias.
- **Fit (Training) Model (minimize loss function)**: Specify the batch size, the number of epochs (training iteration times), and using training set of data.
- **Evaluate Model**: Evaluate the model's performance by using validation data set.
- **Fine-Tuning Hyperparameter**: Training model with different learning rate (step size during training), batch size (number of data sets used in each iteration of training, , number of layers, neurons per layer, Epoch (training times of passing data sets through network model), to avoid underfitting and overfitting.
- **Make Predictions**: Use the trained model to make prediction on test data.

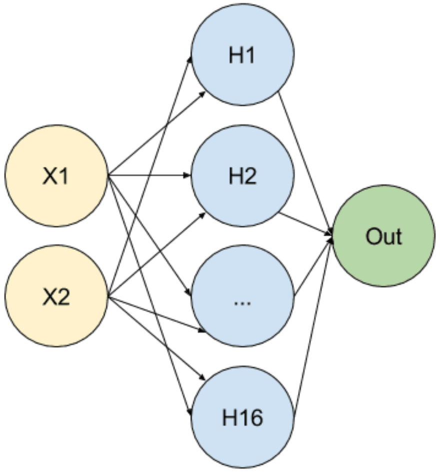
Python Code by Keras for XOR

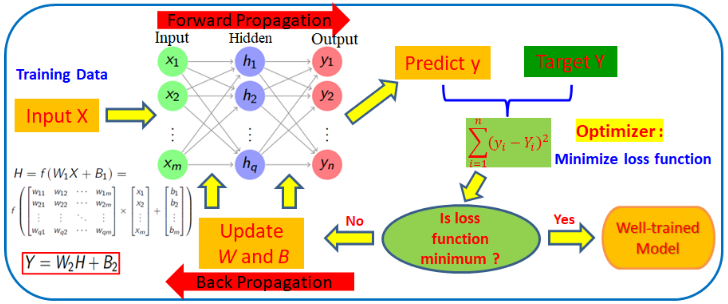
```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Dense

# the four different states of the XOR gate
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")

# the four expected results in the same order
target_data = np.array([[0],[1],[1],[0]], "float32")
# Build a model
model = Sequential()
model.add(Dense(16, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['binary_accuracy'])
# start to train
model.fit(x=training_data, y=target_data, nb_epoch=500, verbose=2)
# Prediction
print model.predict(training_data).round()
```

Dense: Fully connected









https://keras.io/api/metrics/

https://keras.io/api/optimizers/

https://keras.io/api/models/model_training_apis/

https://blog.thoughtram.io/machine-learning/2016/11/02/understanding-XOR-with-keras-and-tensorlow.html