Applications of computer vision and forecasting to the CERN accelerators

F. M. Velotti, B. Goddard, V. Kain
with many inputs from ML community forum

Outline

- Introduction
- CNN for beam dump system analysis
  - Brief intro to CNNs and AE
  - VAEs for beam dump screen analysis
- LSTMs for kicker temperature predictions
  - Brief intro to LSTMs
  - Application of LSTMs models to kicker temperature prediction
- Physics Informed Neural Networks
  - Brief intro to PINN
  - Application of PINN to hysteresis predictions
  - Simple example of PINN
- Summary

Introduction

➔ How do we use Convolutional Neural Networks (CNN)?

Introduction

➔ How do we use Convolutional Neural Networks (CNN)? Obviously, to recognise a cat! [1]

Introduction

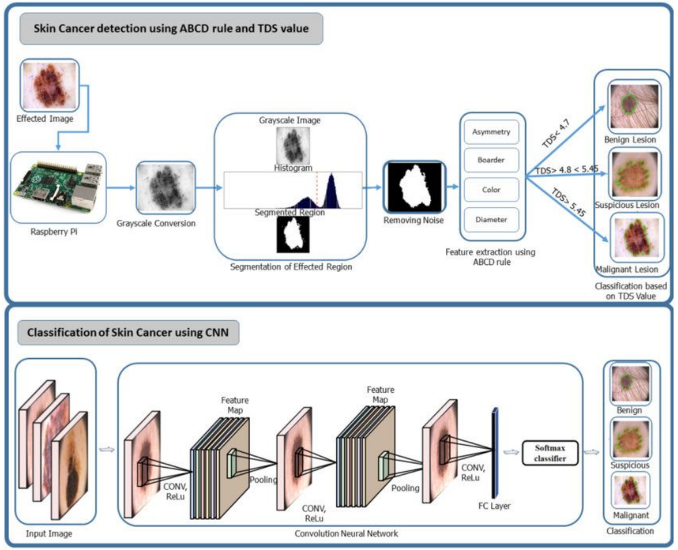
➔ How do we apply time-series forecasting?

Introduction

➔ How do we apply time-series forecasting? Obviously, to predict stock market!

Introduction (serius now)

- Huge achievements in image analysis with Convolutional Neural Networks (CNN)
  - Cancer tumores diagnostic from images (e.g. [2], [3]): in use in many institutes



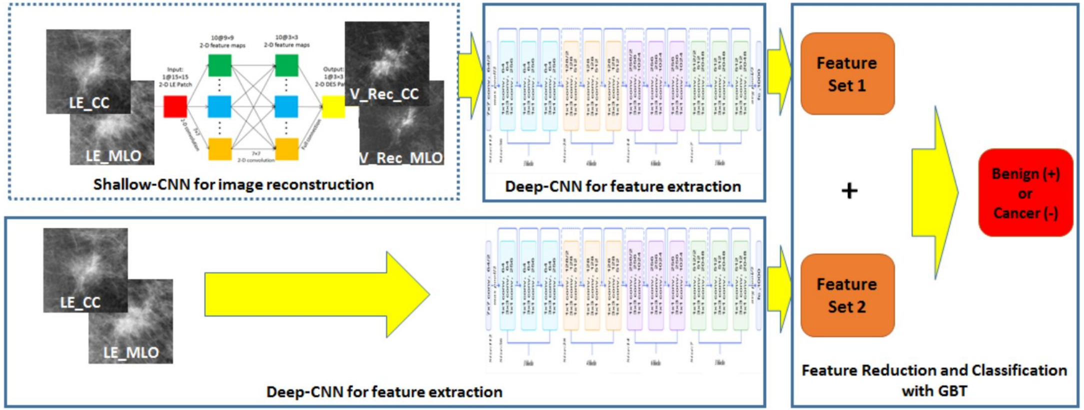
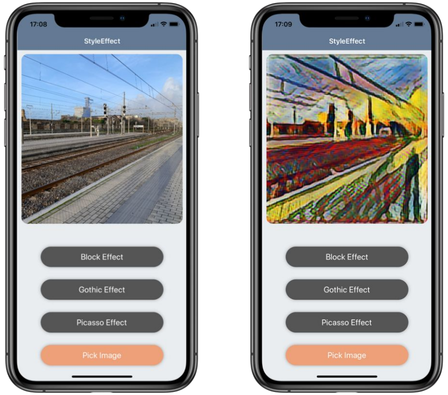
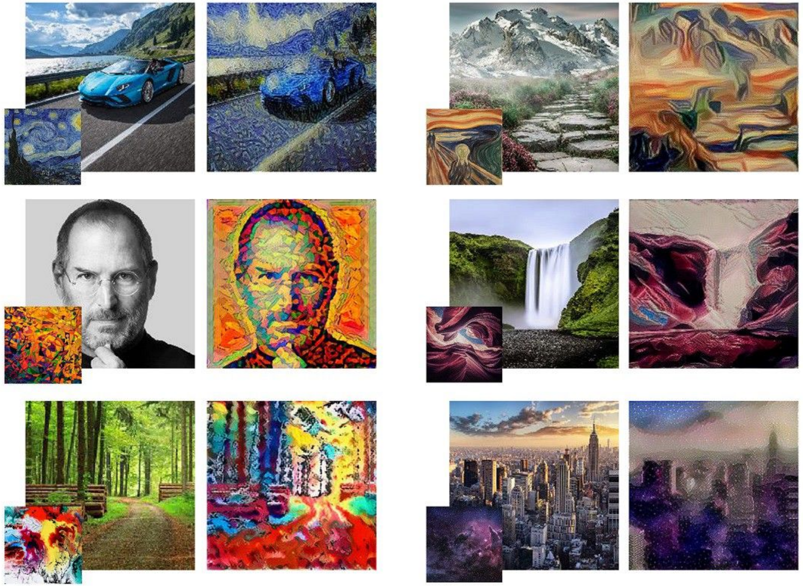


Fig. 3. Architecture of Shallow-Deep CNN

Introduction

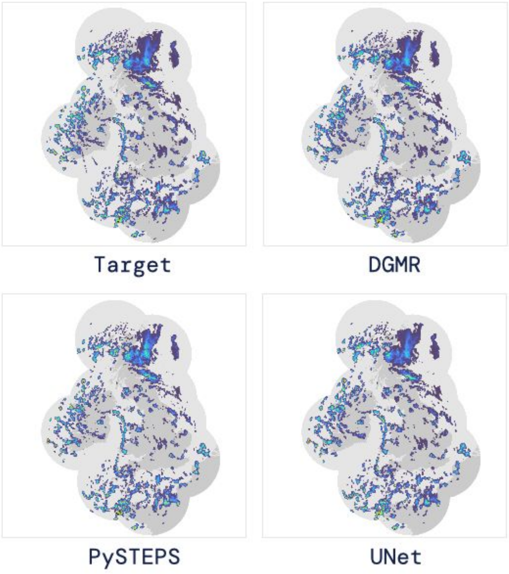
➔ Huge achievements in image analysis with Convolutional Neural Networks (CNN)
- Cancer tumores diagnostic from images (e.g. [2], [3]): in use in many institutes
- Here is a guide how to make a style changer app





Introduction

- ➔ Time-series prediction is another extremely active research topic
- ➔ Weather nowcast is a perfect example of how the 2 model types can work together [4]

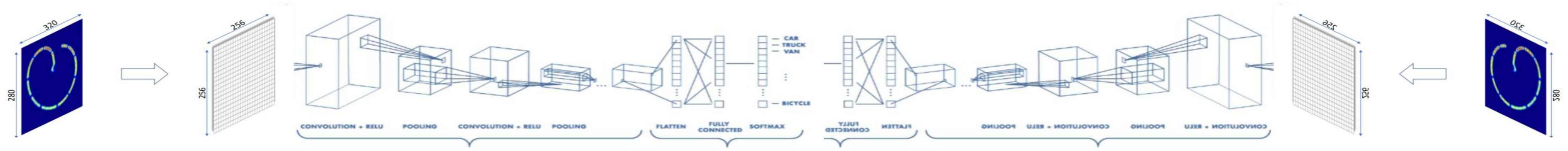


CNN for beam dump system analysis

Example of CNNs @ CERN accelerators

- ➔ **SPS and LHC beam dump systems:**
  - BTV just before absorber block => image of the dumped beam
- ➔ **GOAL: infer the state of the dump system from image and extract anomalous system**

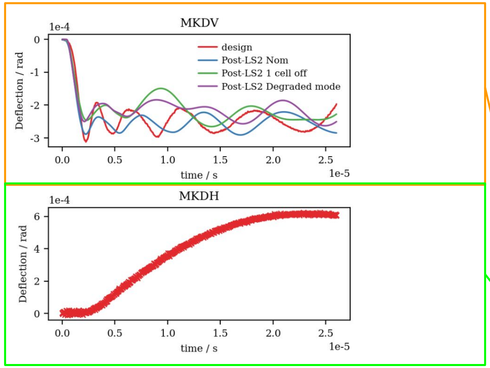
Physical system: $C[k_v, k_h, \tau, ...]$ Input: $X[m, n]$ Output: $\bar{C}, \bar{X}$



Feature extraction Regression

Just a little step back: SBDS

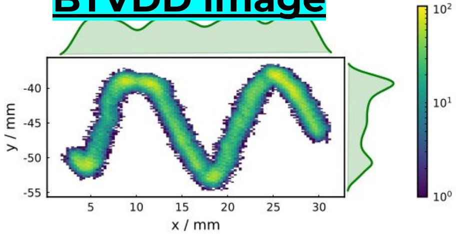
➔ The SPS dump system in a nutshell



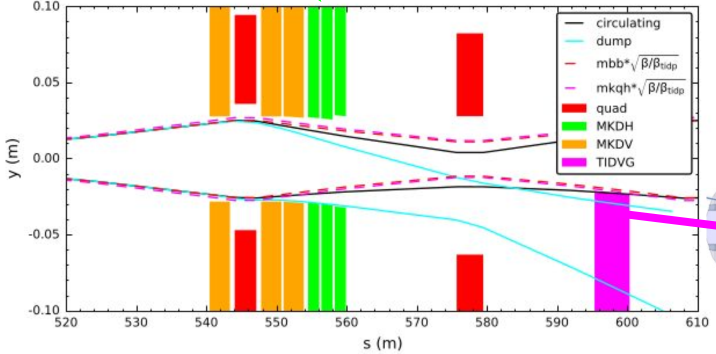
→ **Dump positon monitoring:**

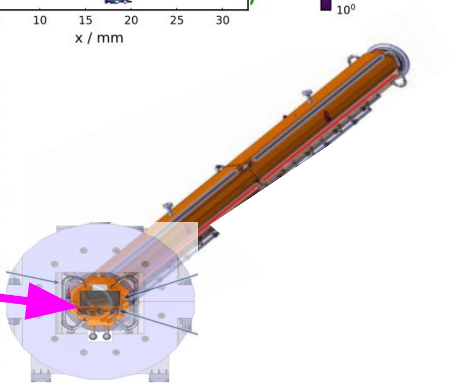
↳ Old SEM grids will be replaced with BTV in front of the TIDVG

BTVDD image



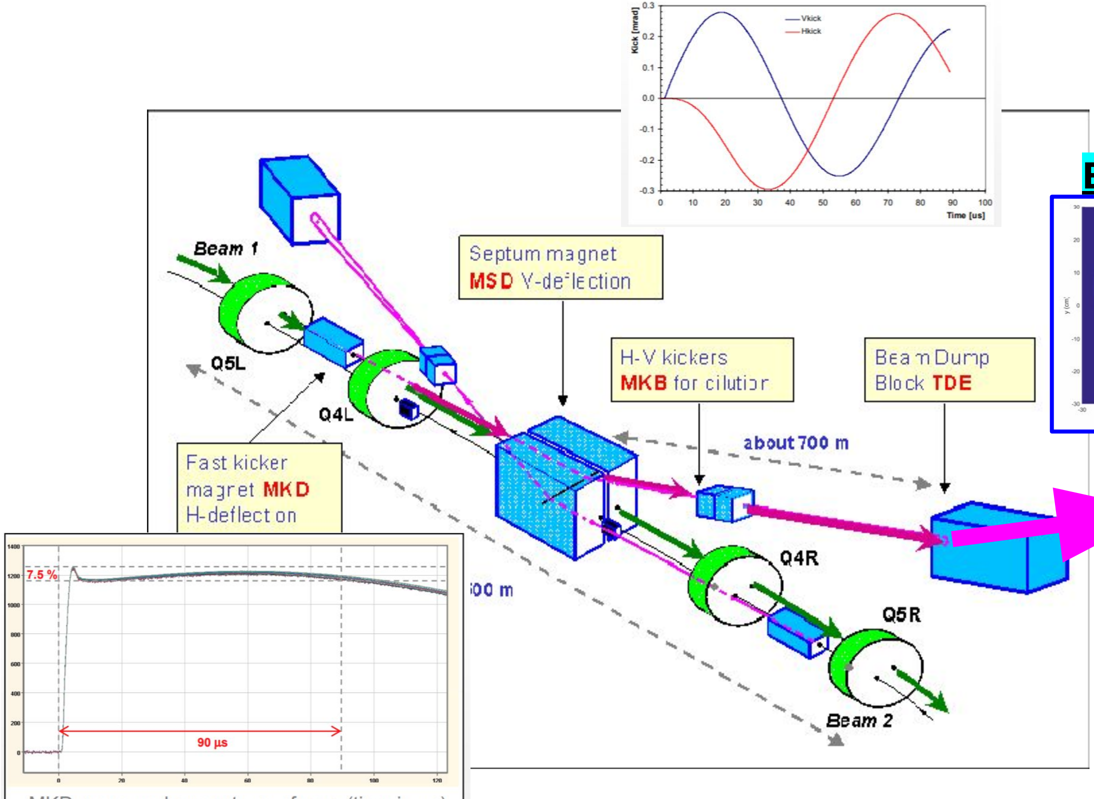
14 GeV SFTPRO





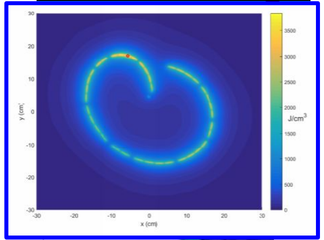
Just a little step back: LBDS

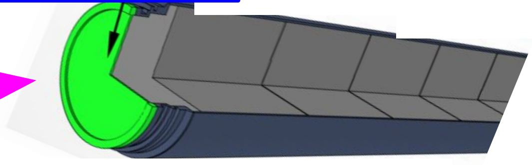
➔ The LHC beam dump system in a nutshell



MKD measured current waveforms (time in µs)

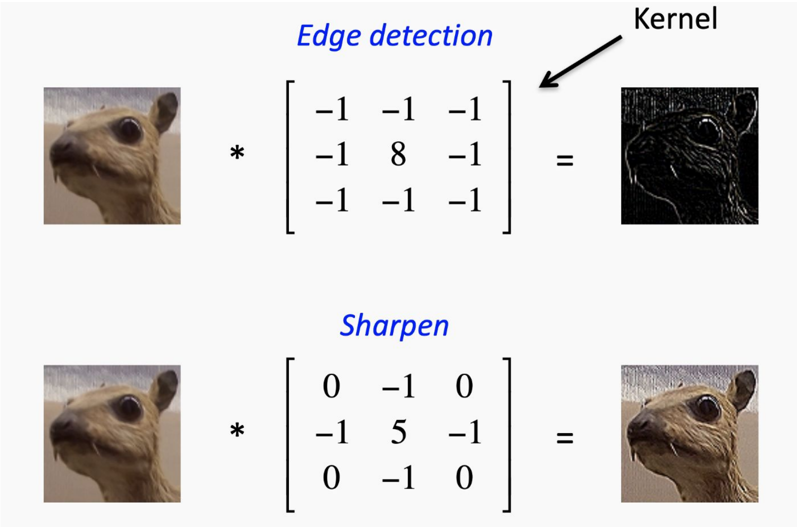
BTVDD image

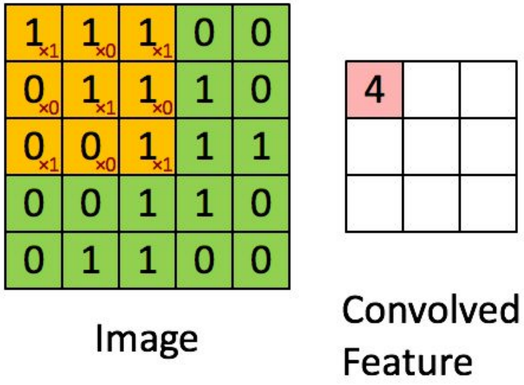




Convolutional NN

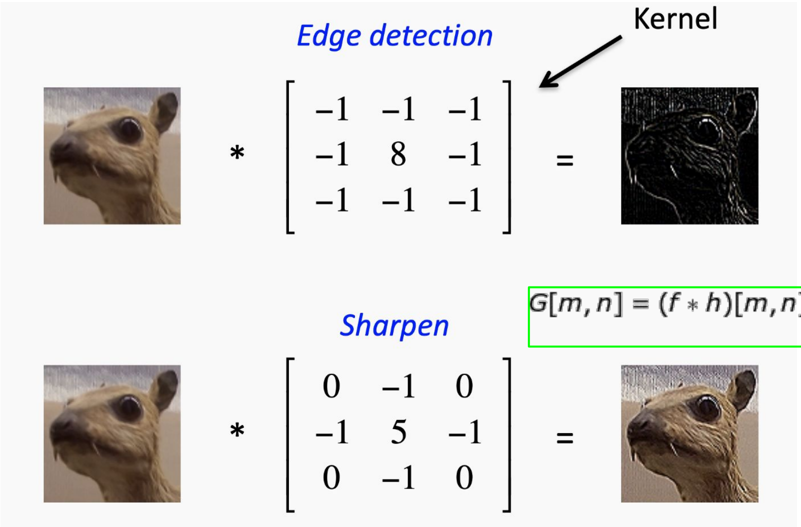
➔ CNN are neural networks that are mainly used for image processing
➔ We can see it as a sliding filter on the image

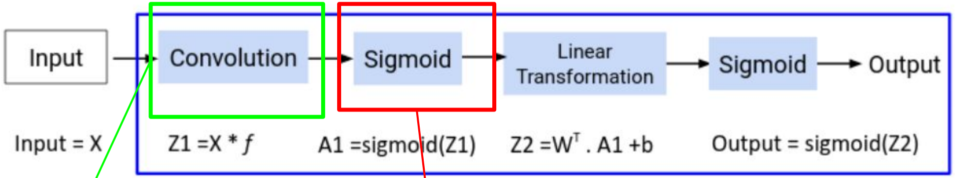




Convolutional NN

- ➔ CNN are neural networks that are mainly used for image processing
- ➔ We can see it as a sliding filter on the image => **not a black box but just a complicated function on many dimensions!**
  - ◆ "Looking at the a function's surroundings to make better/accurate predictions of its outcome" [Dr Prasad Samarakoon]





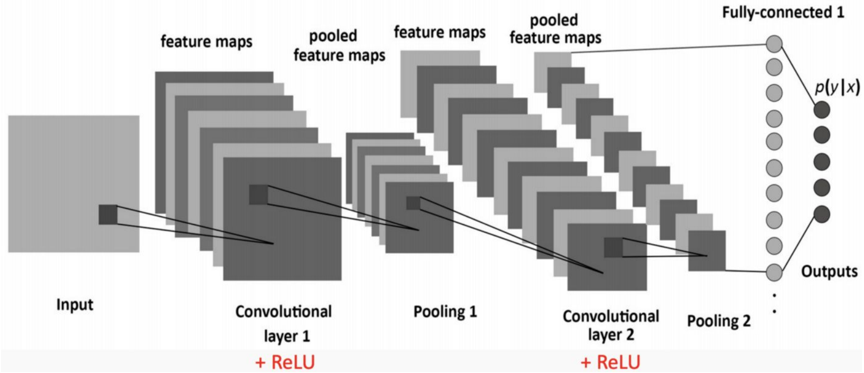
$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k]$$

$$A1 = \frac{1}{1+e^{-Z1}}$$

(*) This can be any other non-linearity

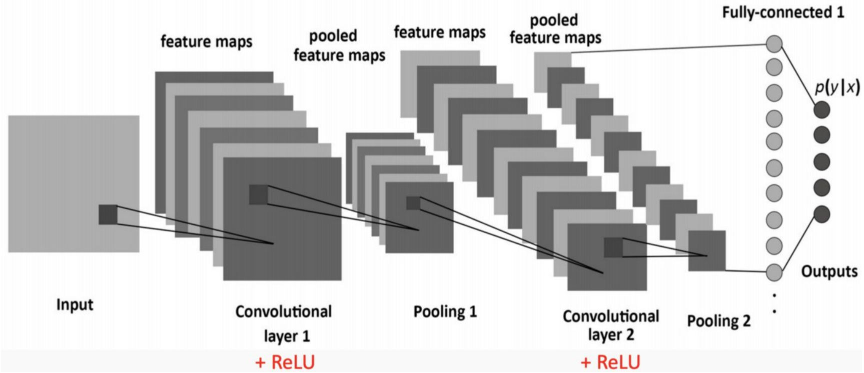
Convolutional NN models

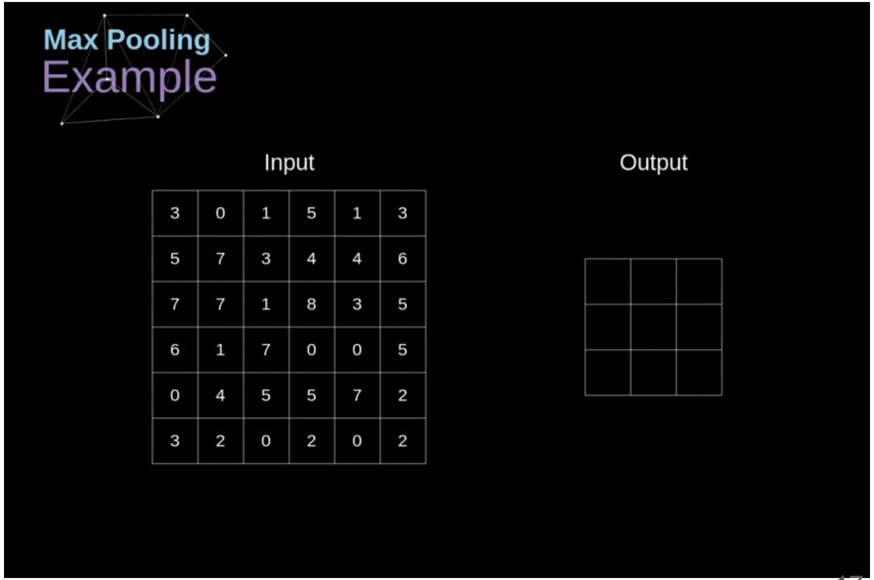
➔ CNN models are a sequence of CNN layers, but not only...



Convolutional NN models

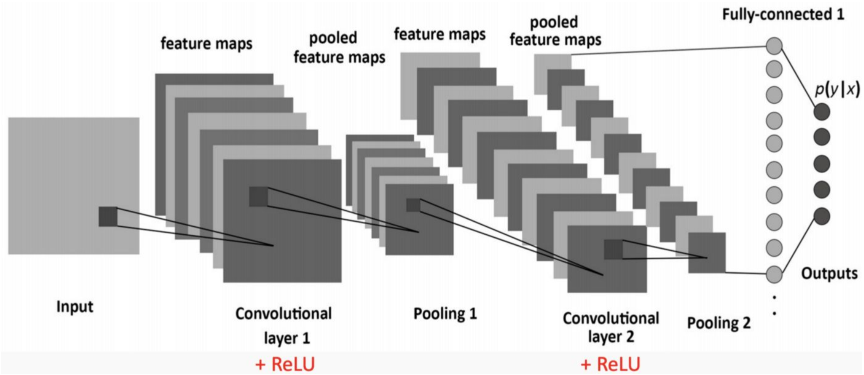
- ➔ CNN models are a sequence of CNN layers, but not only…
  - Max pooling

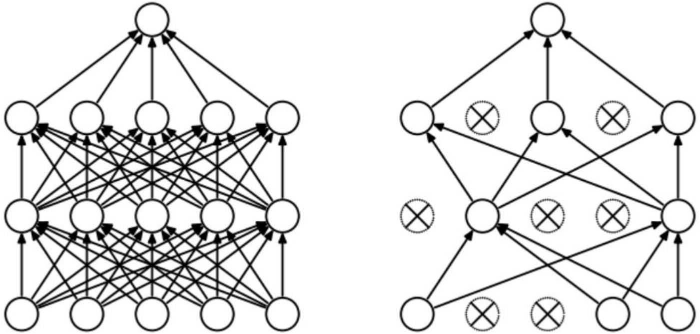




Convolutional NN models

- ➔ CNN models are a sequence of CNN layers, but not only...
  - ◆ Max pooling, dropout



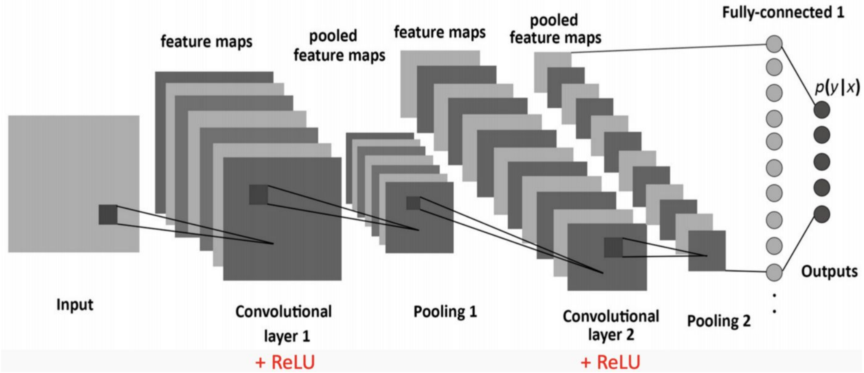


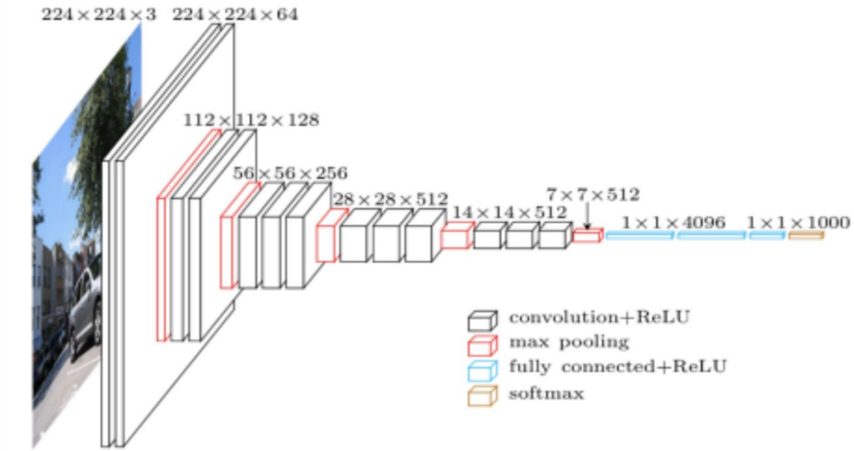
(a) Standard Neural Net

(b) After applying dropout.

Convolutional NN models

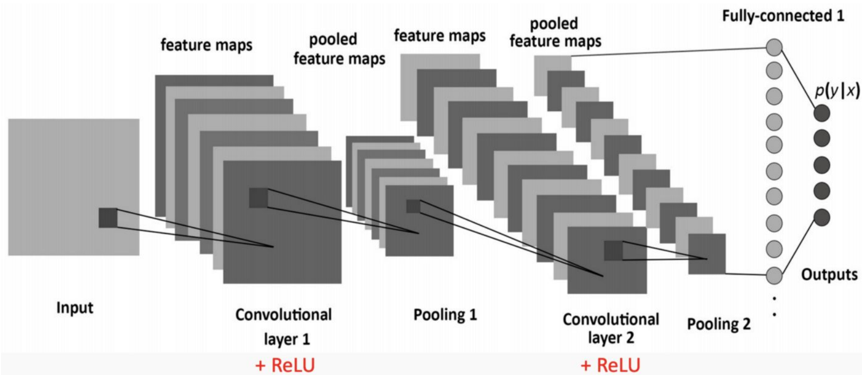
➔ CNN models are a sequence of CNN layers, but not only…
- Max pooling, dropout, linear layers…
- They can be used for classification or regression

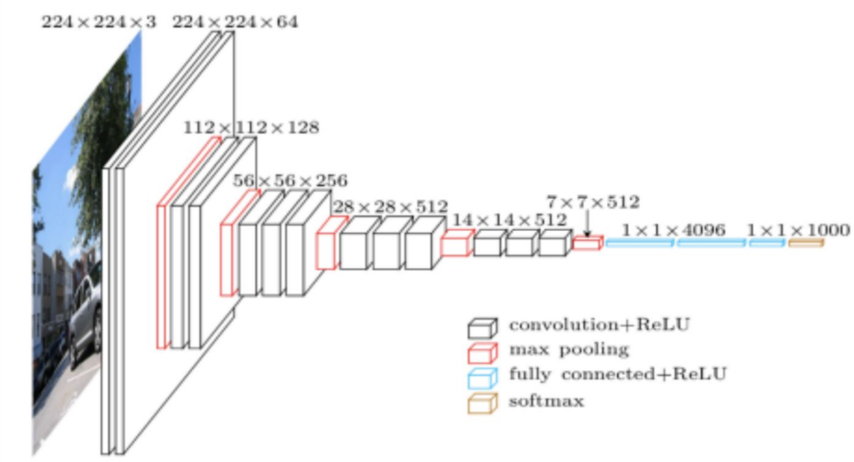




Convolutional NN models

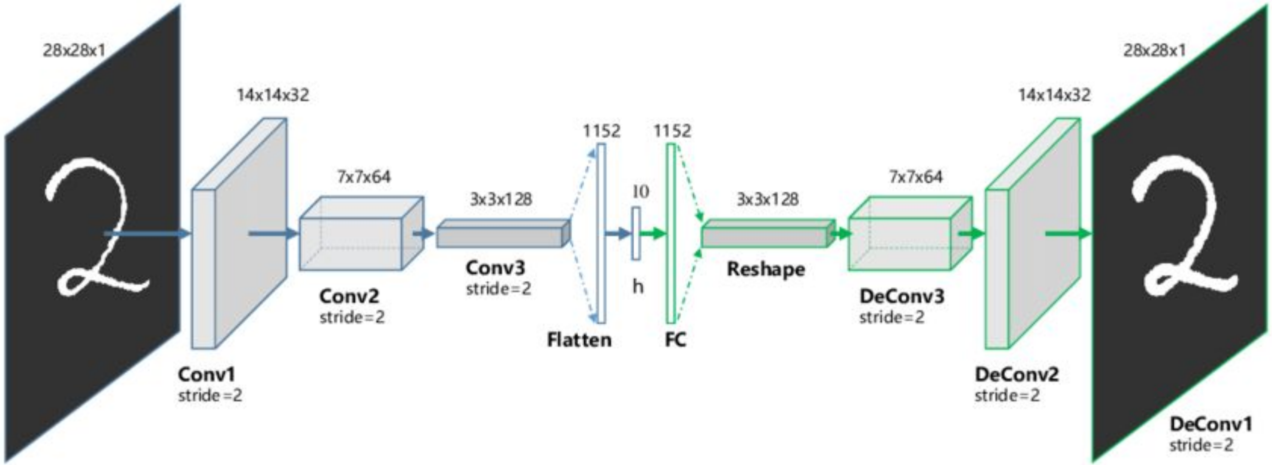
- ➔ CNN models are a sequence of CNN layers, but not only…
  - Max pooling, dropout, linear layers…
  - They can be used for classification or regression
  - Very clear explanation how CNN work here





Auto Encoders and Variational AE

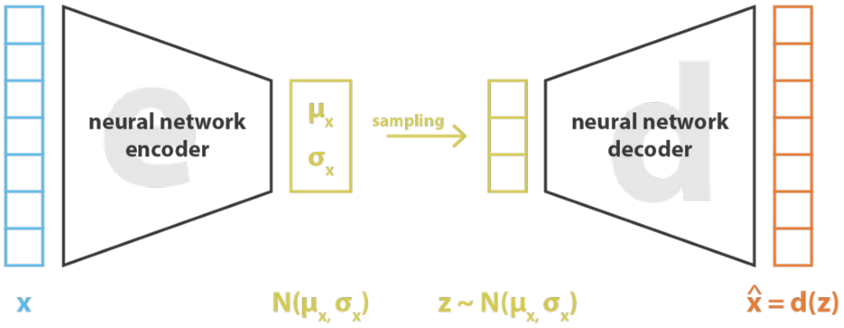
- ➔ Auto Encoders are just a type of NN that aims to learn efficient encoding of the unlabelled data (unsupervised learning)
  - ◆ This is done regenerating the input parameters (images, vectors, scalars), e.i. minimising the reconstruction error of the input
- ➔ Usually used for dimensionality reduction (kind of non-linear PCA), denoising, generative models, translation...



CERN Academic Lecture, Applications of computer vision and forecasting to the CERN accelerators, 5th May 2022

Auto Encoders and Variational AE

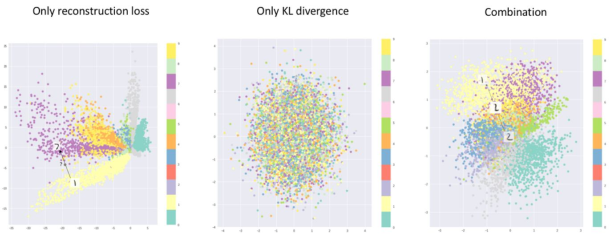
- Variational Auto Encoders (VAE) [12] are special type of encoder
  - Express the latent attributes as probability distribution
- This leads to smooth latent state representation of the input => towards generative interpolating models



$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

Kullback-Leibler Divergence

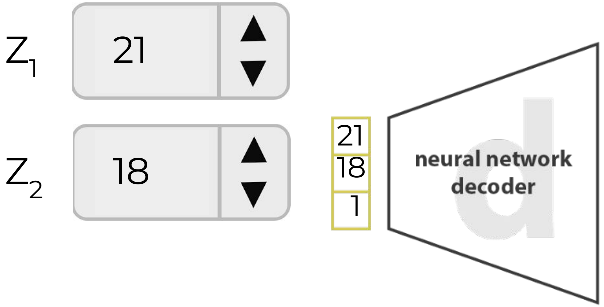
$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot log \frac{p(x_i)}{q(x_i)}$$

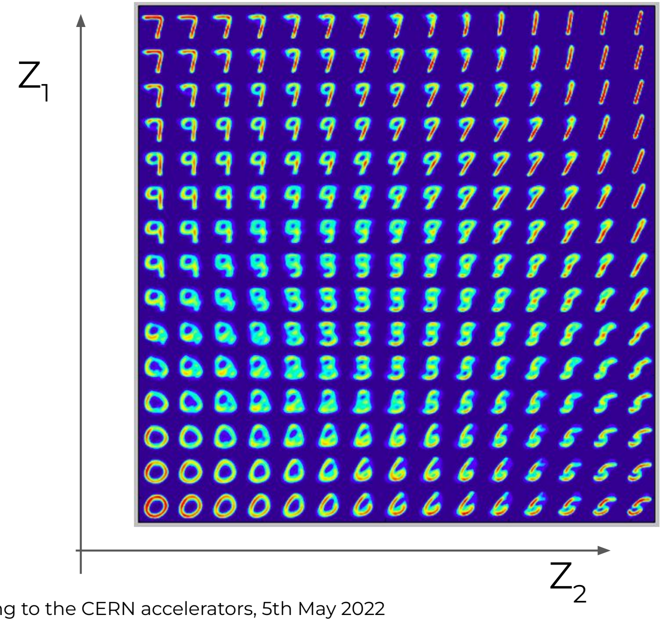


Source

Auto Encoders and Variational AE

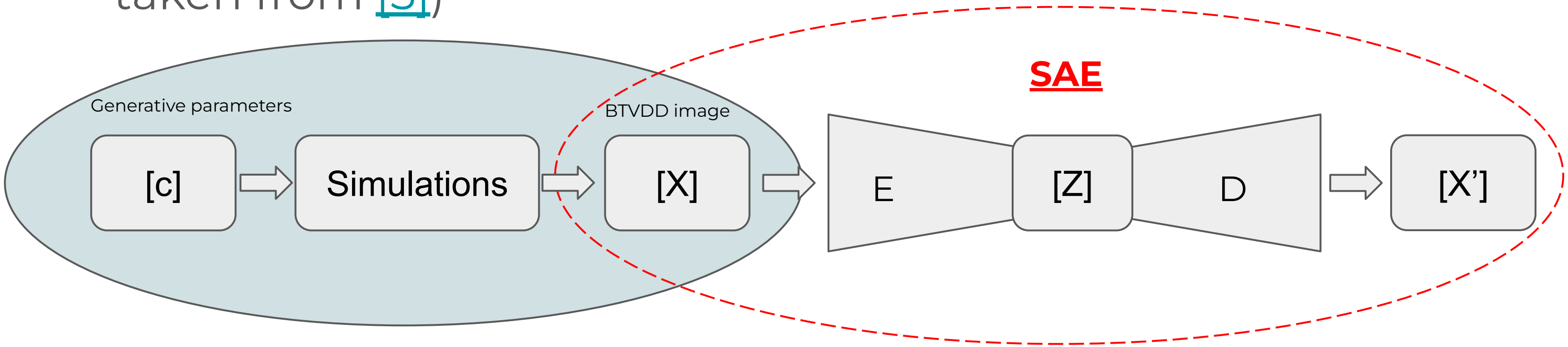
- Variational Auto Encoders (VAE) [12] are special type of encoder
  - Express the latent attributes as probability distribution
- This leads to smooth latent state representation of the input => towards generative interpolating models





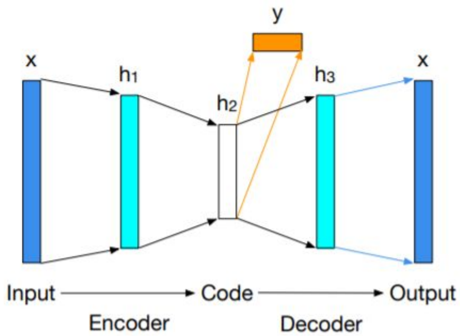
VAE for BTVD image reconstruction

➔ Special case of VAE => Supervised [Variational] Auto Encoder (idea taken from [5])



$$L_i(\theta,\phi) = -E_{z\sim q_\theta(z|x_i)}[\log_\phi(x_i|z)] + w_{KL}\, KL(q_\theta(z|x_i), p(z)) + w_g\, MSE(c, Z)$$

w_{KL}:
- =0
- !=0



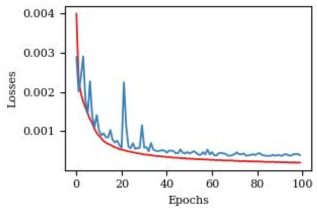
VAE for BTVD image reconstruction

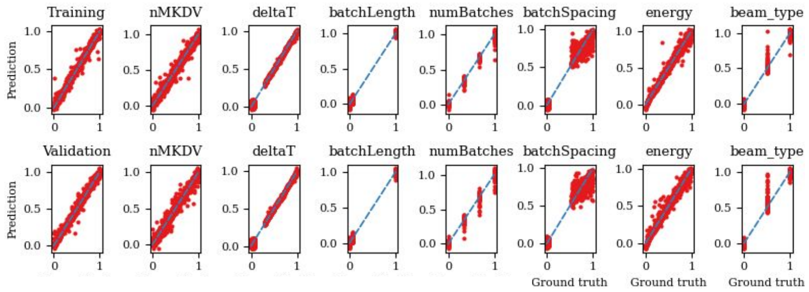
- ➔ All the snippet prested developed in [Pytorch](#)
- ➔ Started from the VAE [6]
- ➔ Many modification to the model were made to make it tunable at need
  - ◆ Our model is available [7] for the LBDS and very similar for SBDS
  - ◆ Custom loss function

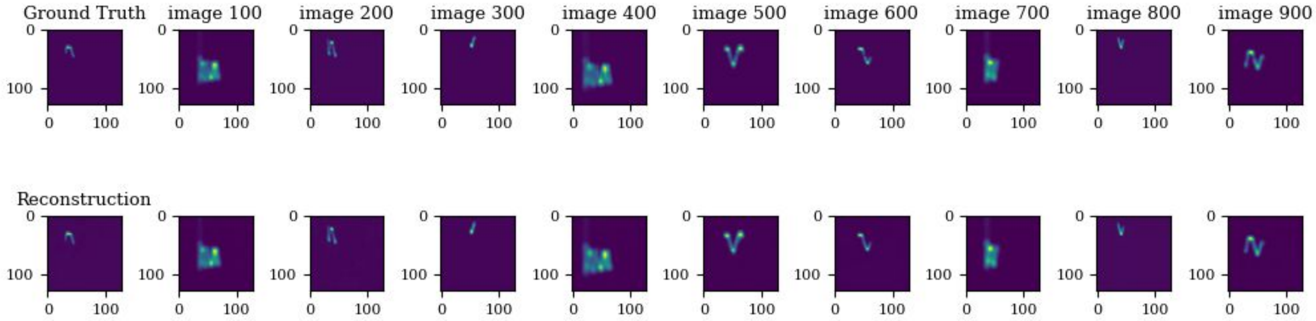
```python
def loss_function(
    self,
    out_vae: List,
    generative: Tensor = None,
    gen_w: float = 10.0,
    kl_w: float = 0.0,
) -> dict:

    recons = out_vae[0]
    input = out_vae[1]
    mu = out_vae[2]
    log_var = out_vae[3]

    # Xentropy loss of images
    recons_loss = F.mse_loss(recons, input)

    # KL divergence
    kld_weight = kl_w  # Account for the minibatch samples from the dataset
    kld_loss = torch.mean(
        -0.5 * torch.sum(1 + log_var - mu ** 2 - log_var.exp(), dim=1),
        dim=0,
    )

    # Generative factors
    gen_loss = F.mse_loss(mu, generative) if generative is not None else 0.0

    loss = recons_loss + kld_weight * kld_loss + gen_w * gen_loss
    return {
        "loss": loss,
        "Reconstruction_Loss": recons_loss,
        "KLD": -kld_loss,
        "gen_loss": gen_loss,
    }
```

BTVDD image reconstruction in SPS

- ➔ Very accurate prediction from simulations
- ➔ Batch spacing reconstruction not obvious (very difficult to see)
- ➔ Reconstructed images almost indistinguishable

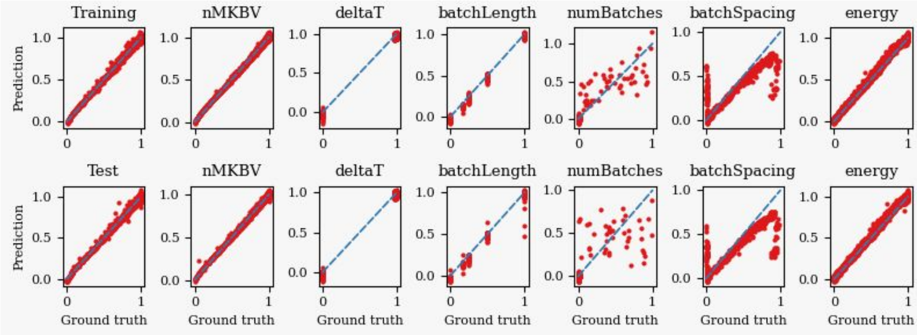


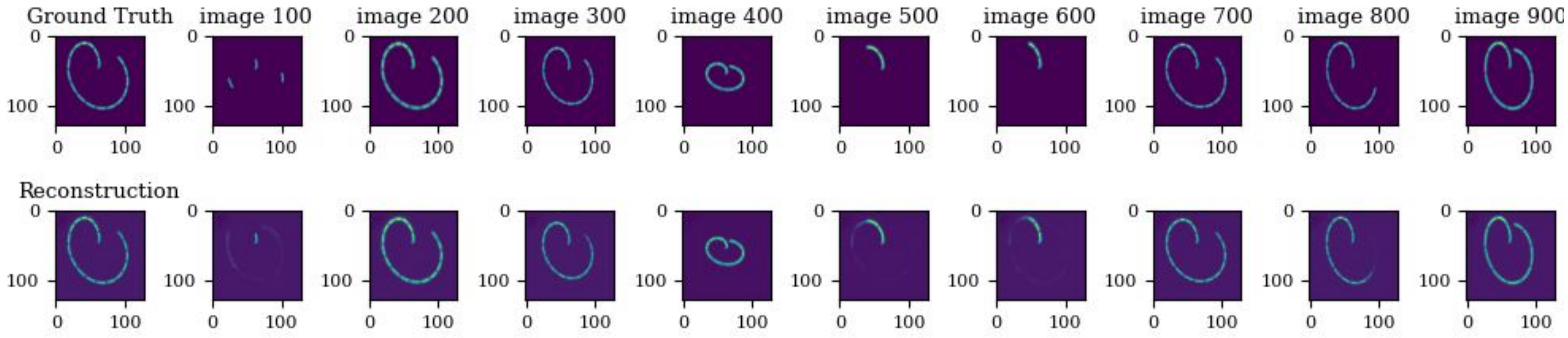




BTVDD image reconstruction in LHC

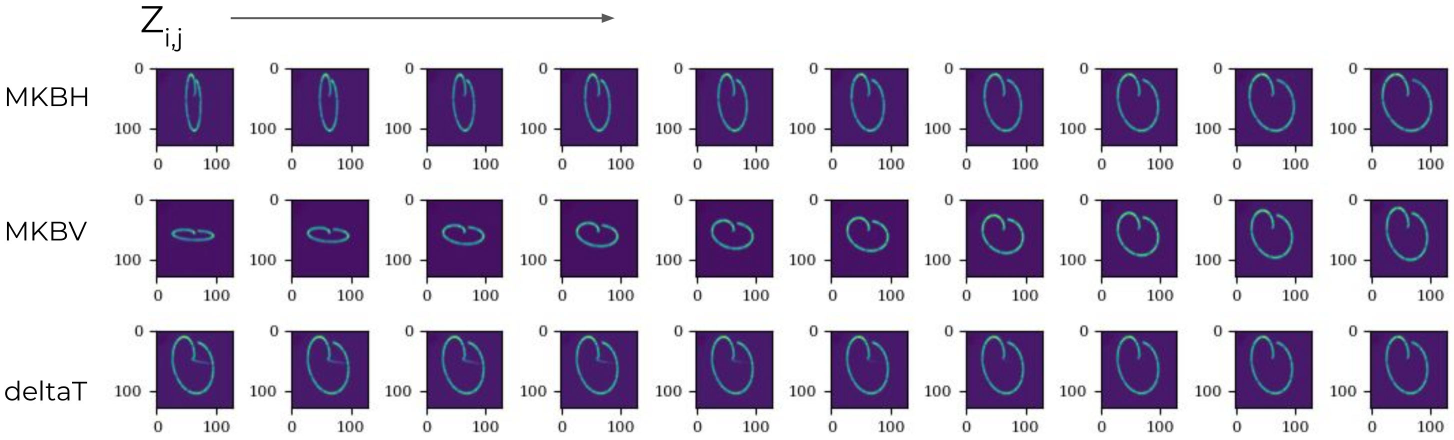
- ➔ Similar results for LHC
- ➔ Here the most complicated part is to simulate different filling patterns
  - ◆ Number for batches very difficult for many single bunches
  - ◆ batch spacing very difficult for single bunches





Latent space scan

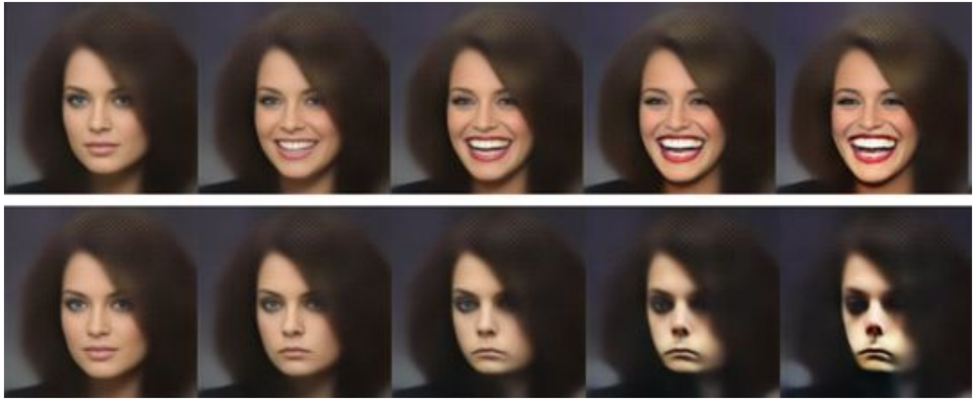
- ➔ With this architecture, we can generate BTVDD images from generative parameters (number of kickers...) using the decoder by itself
- ➔ Orthogonal scan possible

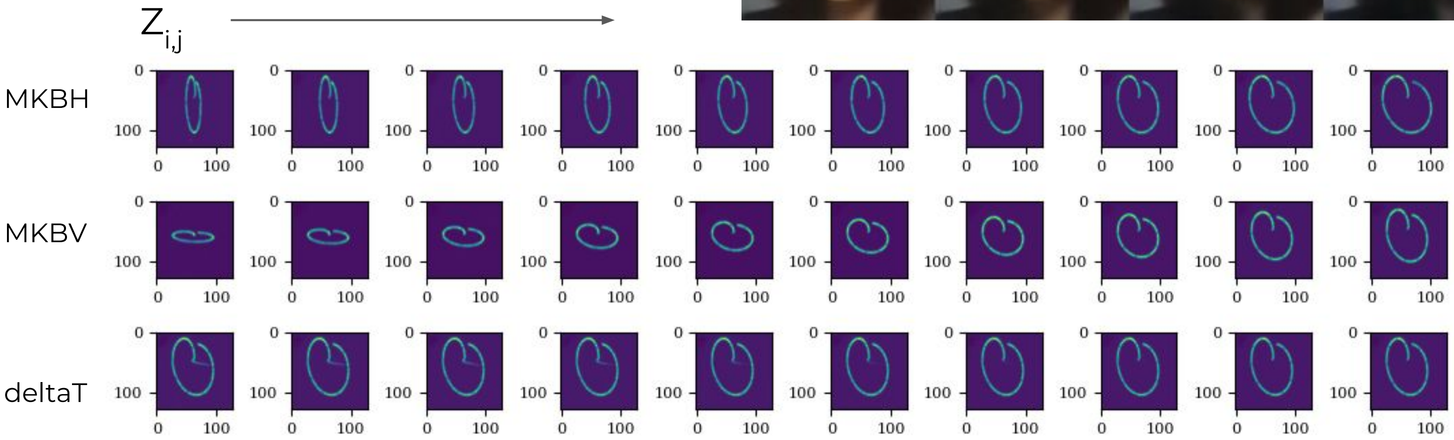


Latent space scan

➔ With this architecture, we can generate BTVDD images from generative parameters (number of kickers...) using the decoder by itself

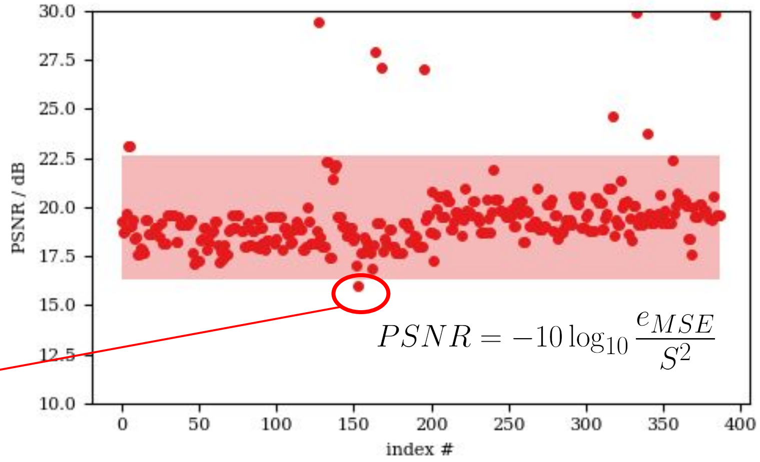
➔ Orthogonal scan possible

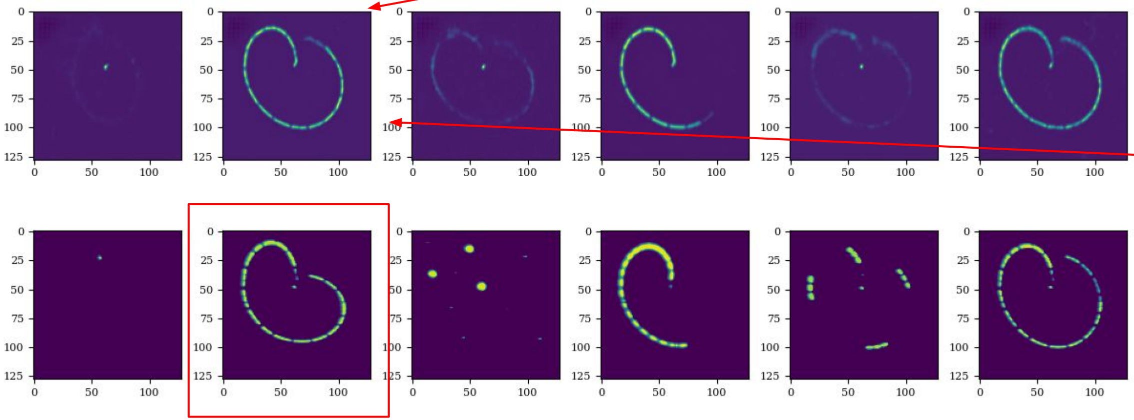


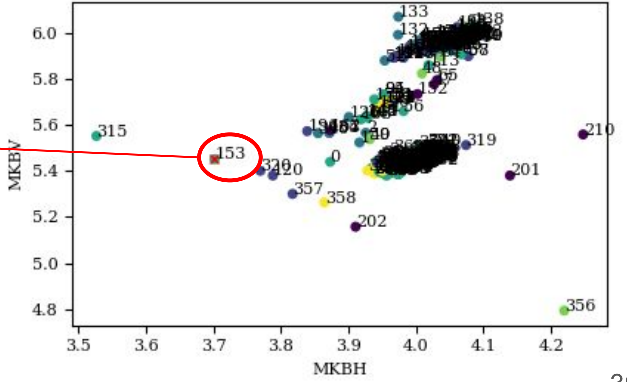


Deploy on real data

- Of course the final goal is to predict real images...
- Using both generative parameters and image reconstruction score, anomalous case found!

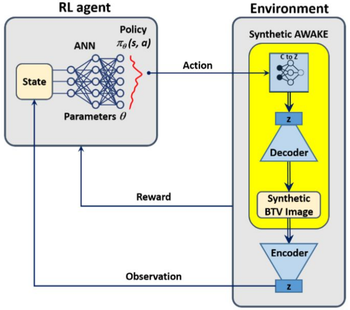


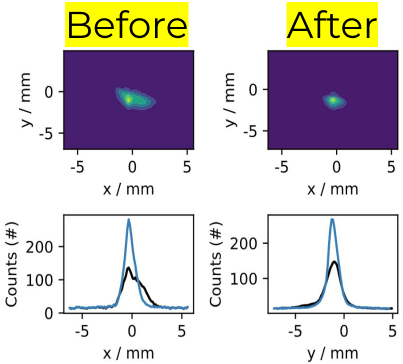


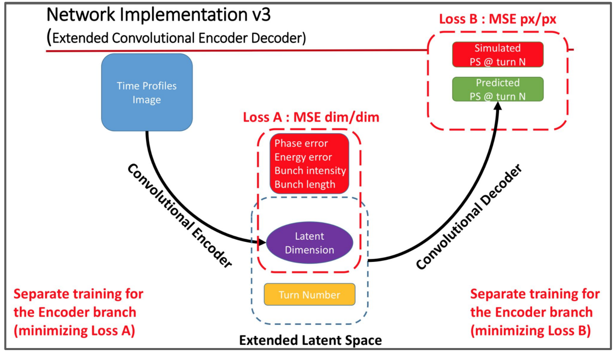


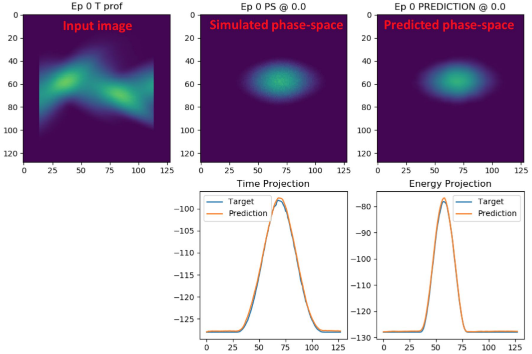
Other examples

- Neural Longitudinal tomography in the LHC
  - Classically limited to single bunch => with ML no limits!
- Unsupervised stated encoding for RL applied on AWAKE transfer line matching agent
  - Use of the encoded information of BTV image to match beam size to requested one







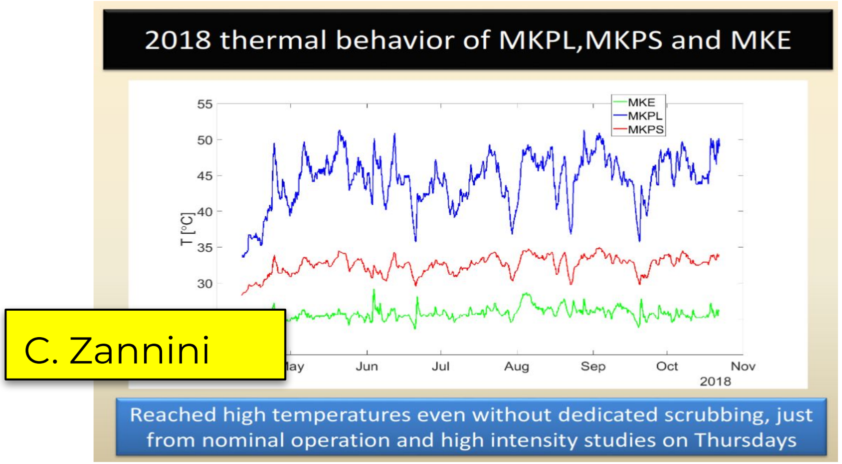


G. Trad and T. Argyropoulos

LSTMs for kicker temperature predictions

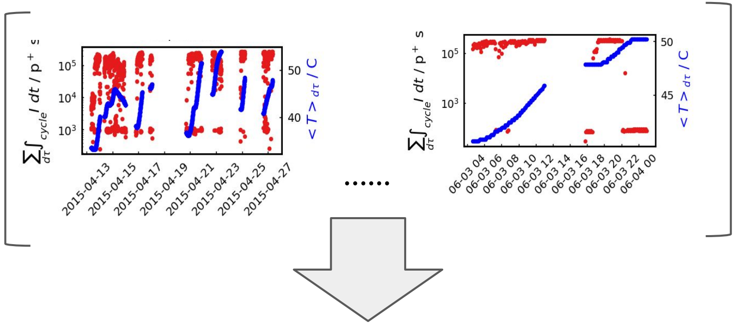
Introduction to the problem

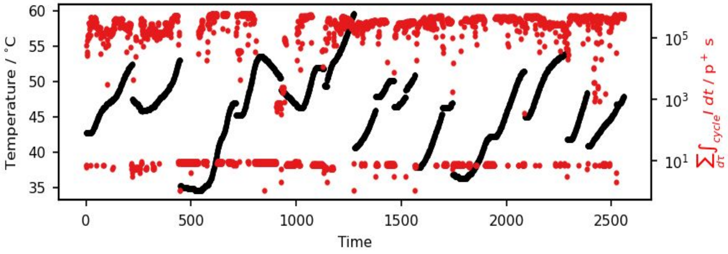
- The MKP-L is one of the main limiting element for high intensity
  - Beam induced heating is directly related to the beam power loss through the real part of the longitudinal impedance
- Temperature observed to be much higher than normal operation also during 2018's HI MDs



Model for the MKP-L heating evolution

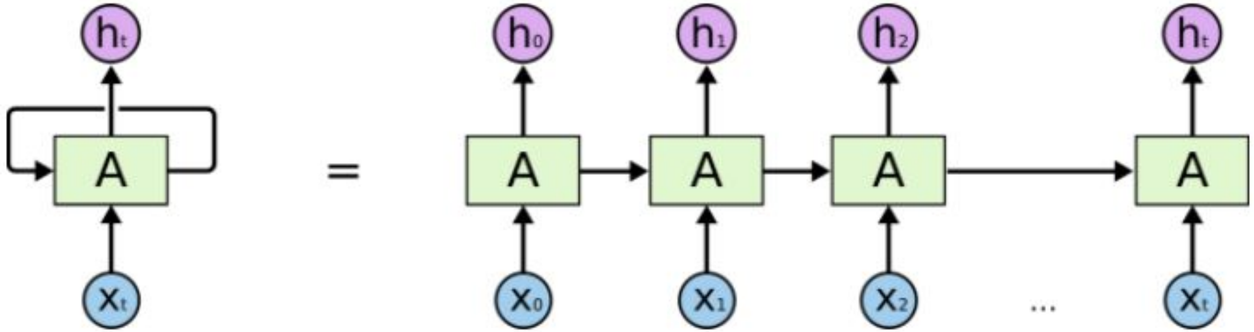
- ➔ Neural networks to **<u>estimate the temperature evolution of the MKP as a function of the intensity and history</u>**
  - ◆ Should be able to suggest the best strategy to optimise scrubbing
  - ◆ Keep MKP temperature below limits
  - ◆ Reduce idle time
- ➔ **<u>This is a time series!!</u>**
  - ◆ LSTMs are a very good choice for these kind of problems
- ➔ **<u>Input data:</u>** Intensity integrated over 5 min, bunch length, peak intensity and temperature history





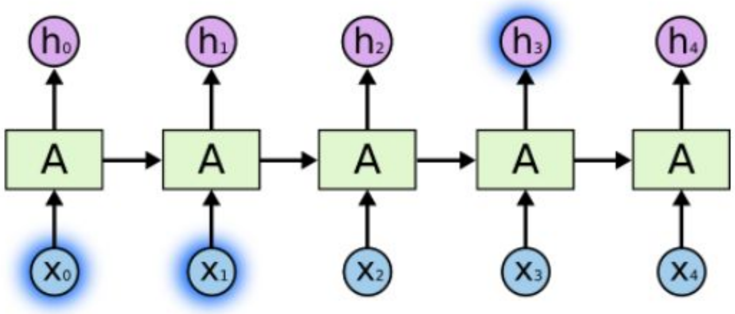
Recurrent Neural Networks

- ➔ DNN (as seen in one of the first lectures) cannot “remember” previous estimations as they deal with instantaneous data
- ➔ Recurrent NN (RNN) address this issue (source [6])
  - ◆ The input is passed to the same NN and the output is then recursively injected in the following prediction



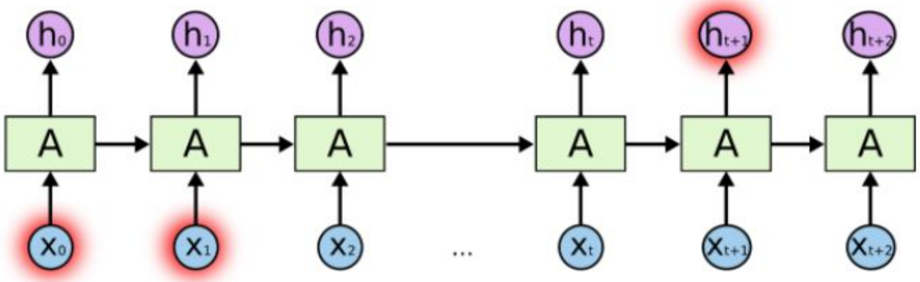
Recurrent Neural Networks

- DNN (as seen in one of the first lectures) cannot “remember” previous estimations as they deal with instantaneous data
- Recurrent NN (RNN) address this issue (source [6])
  - The input is passed to the same NN and the output is then recursively injected in the following prediction
- It works great for “recent” predictions



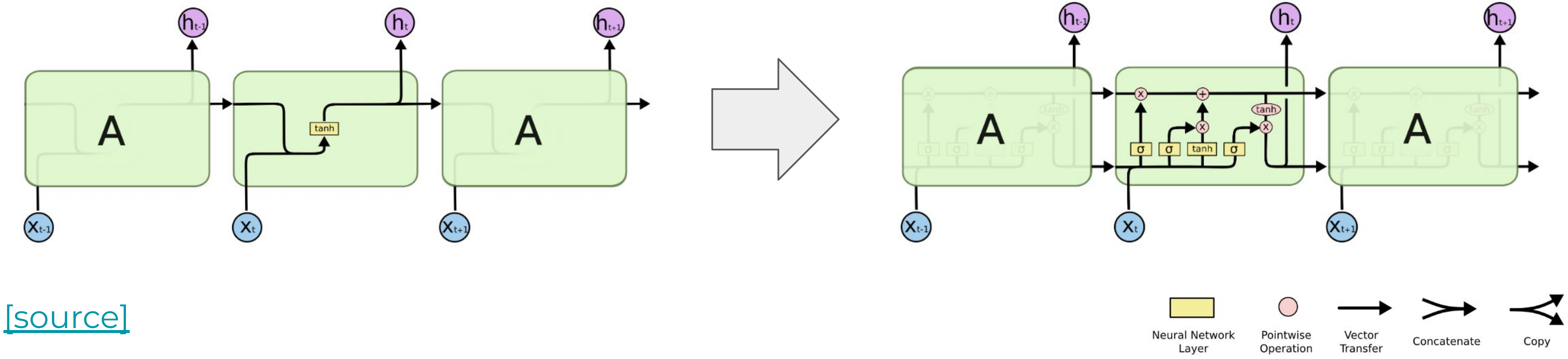
Recurrent Neural Networks

- ➔ DNN (as seen in one of the first lectures) cannot “remember” previous estimations as they deal with instantaneous data
- ➔ Recurrent NN (RNN) address this issue (source [6])
  - The input is passed to the same NN and the output is then recursively injected in the following prediction
- ➔ It works great for “recent” predictions
- ➔ But it struggles for information further back in time [7]



Long Short Term Memory NN

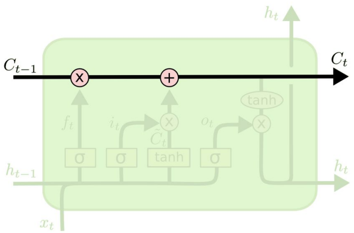
- ➔ In rescue of the RNN and their exploding/vanishing gradient issues (see [7] for more details) come the LSTMs
- ➔ Capable of “remembering” information for long sequences
- ➔ Intuition:
  - ◆ Select important part of sequence to remember



[source]

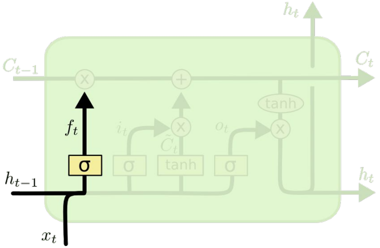
Long Short Term Memory NN

➔ Information flows via cell state from one time stamp to another (with some linear interaction with other gates)

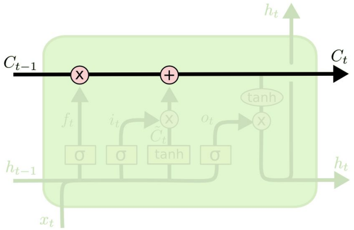


Long Short Term Memory NN

- ➔ Information flows via cell state from one time stamp to another (with some linear interaction with other gates)
- ➔ The “forget gate” decides how much of the cell state C_{t-1} we keep

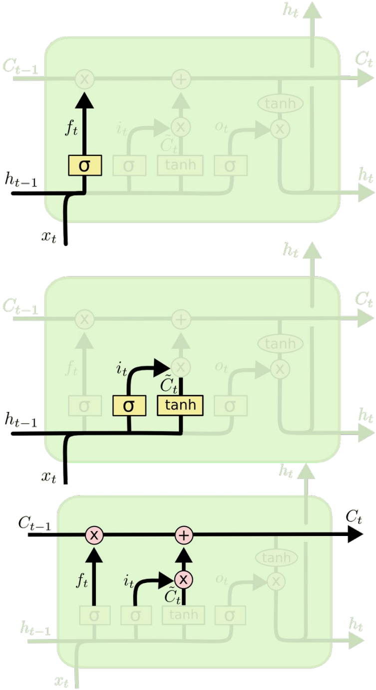


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



Long Short Term Memory NN

- ➔ Information flows via cell state from one time stamp to another (with some linear interaction with other gates)
- ➔ The “forget gate” decides how much of the cell state C_{t-1} we keep
- ➔ The input gate processes the input and proposes a new C_t

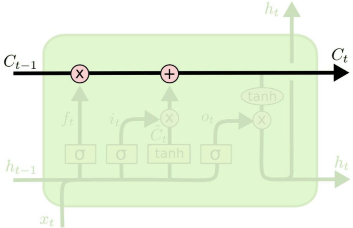


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

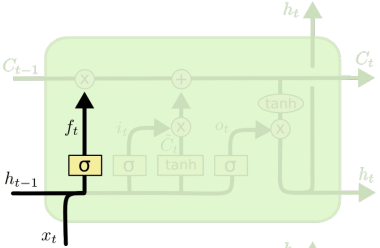
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

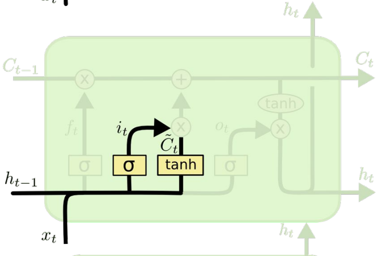


Long Short Term Memory NN

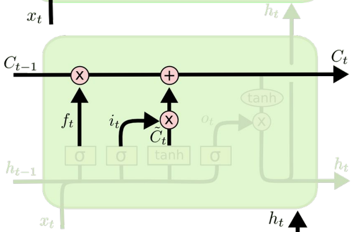
- ➔ Information flows via cell state from one time stamp to another (with some linear interaction with other gates)
- ➔ The “forget gate” decides how much of the cell state C_{t-1} we keep
- ➔ The input gate processes the input and proposes a new C_t
- ➔ Finally, we output h_t for the next cell or to be used as it is



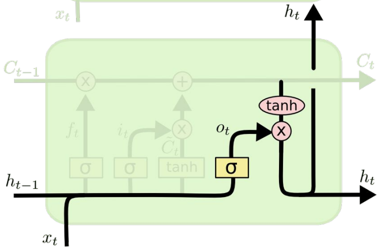
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



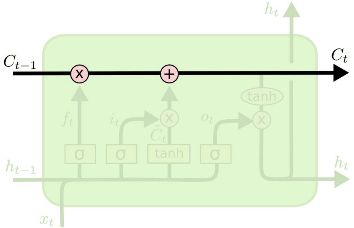
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



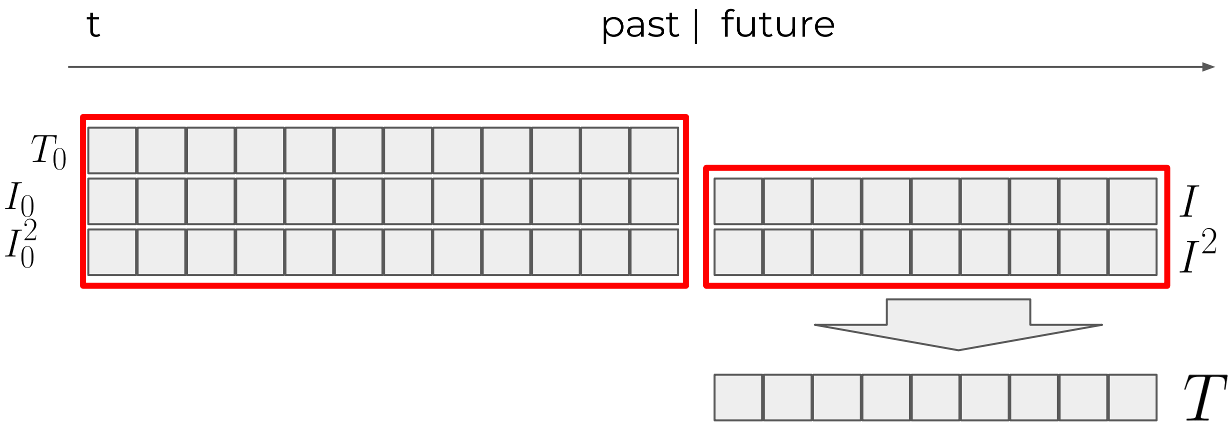
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$



Source

LSTM model for MKP temperature

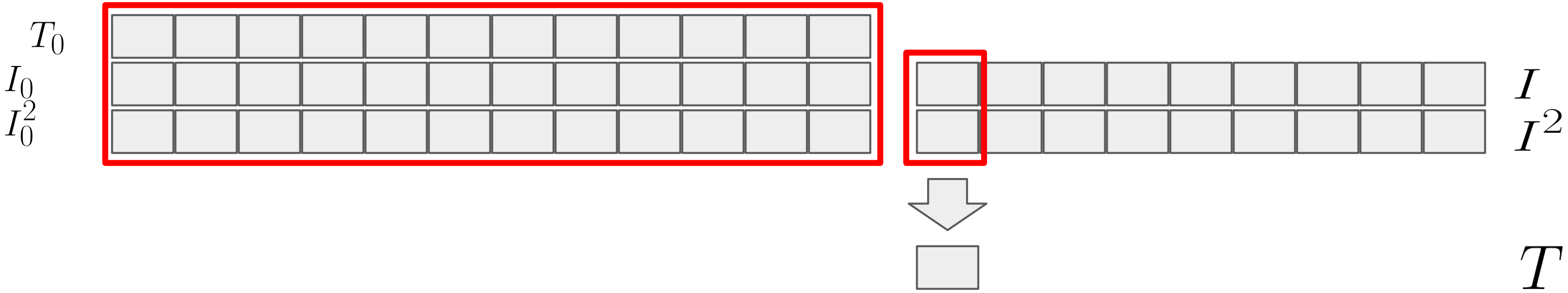
- ➔ Very simple architecture: basically one LSTM layer and a dropout layer before a linear one
- ➔ Add known future input (main difference wrt classic time-series prediction models)




```python
class LSTM_FB(nn.Module):
  def __init__(
      self,
      rnn_num_layers=1,
      input_feature_len=2,
      sequence_len=35,
      hidden_dim=100,
      max_output_size=30,
      device="cpu",
      dropout=0.2,
  ):
      super().__init__()
      self.sequence_len = sequence_len
      self.hidden_dim = hidden_dim
      self.input_feature_len = input_feature_len
      self.num_layers = rnn_num_layers
      self.lstm = nn.LSTM(
          num_layers=rnn_num_layers,
          input_size=input_feature_len,
          hidden_size=hidden_dim,
          batch_first=True,
          dropout=dropout,
      )

      self.max_output_size = max_output_size
      self.out_layer = nn.Linear(self.hidden_dim, 1)
      self.device = device
```

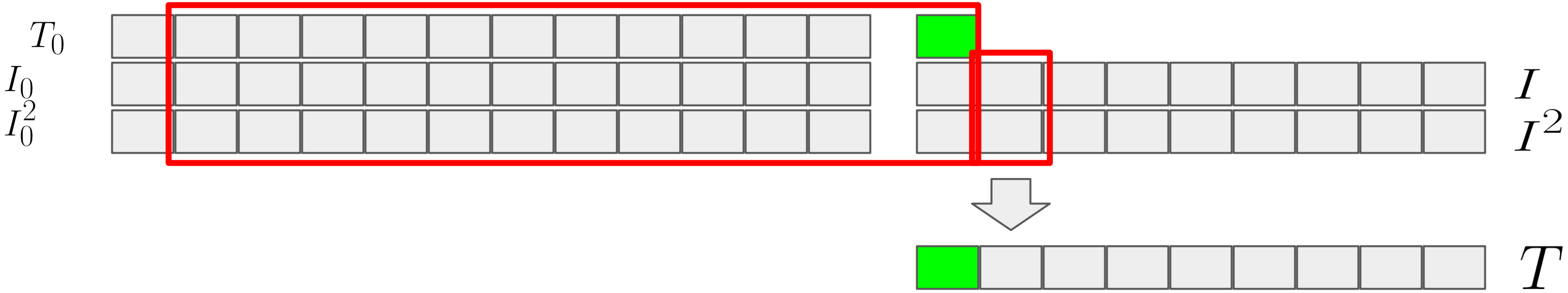
Model training

- Our idea: iterative prediction => teacher forcing for all samples
  - Losses calculated on a fixed sequence length and not value by value
- Advantages:
  - NN already exposed to its noise in the training phase already
  - The output sequence is obtained in one call of the NN (see later for the implementation)
  - Arbitrary output length



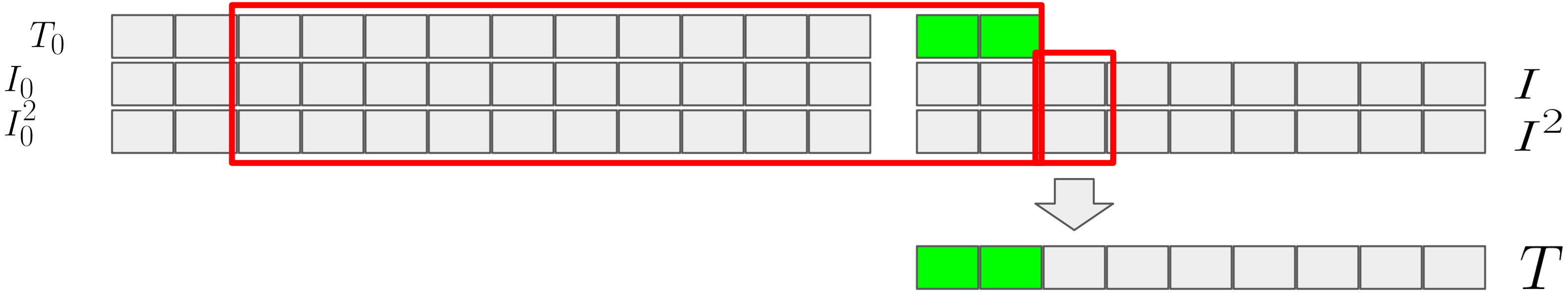
Model training

- ➔ Our idea: iterative prediction => teacher forcing for all samples
  - ◆ Losses calculated on a fixed sequence length and not value by value
- ➔ Advantages:
  - ◆ NN already exposed to its noise in the training phase already
  - ◆ The output sequence is obtained in one call of the NN (see later for the implementation)
  - ◆ Arbitrary output length



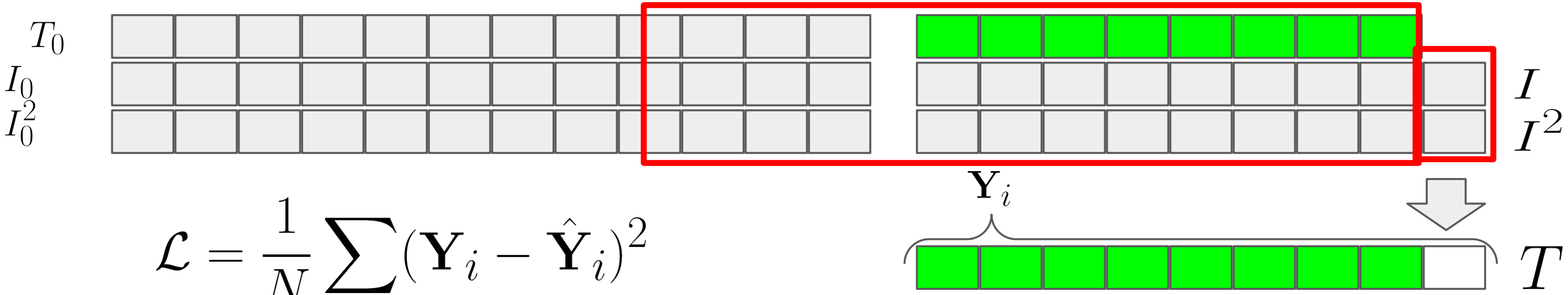
Model training

- Our idea: iterative prediction => teacher forcing for all samples
  - Losses calculated on a fixed sequence length and not value by value
- Advantages:
  - NN already exposed to its noise in the training phase already
  - The output sequence is obtained in one call of the NN (see later for the implementation)
  - Arbitrary output length



Model training

- ➔ Our idea: iterative prediction => teacher forcing for all samples
  - Losses calculated on a fixed sequence length and not value by value
- ➔ Advantages:
  - NN already exposed to its noise in the training phase already
  - The output sequence is obtained in one call of the NN (see later for the implementation)
  - Arbitrary output length

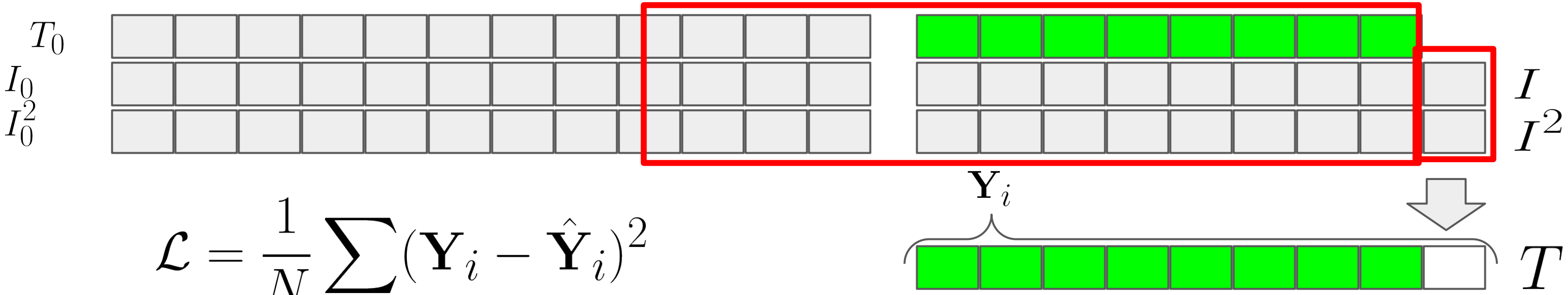


$$\mathcal{L} = \frac{1}{N} \sum_i (\mathbf{Y}_i - \hat{\mathbf{Y}}_i)^2$$

Model training

- ➔ Our idea: iterative prediction => teacher forcing for all samples
  - ◆ Losses calculated on a fixed sequence length and not value by value
- ➔ Advantages:
  - ◆ NN already exposed to its noise in the training phase already
  - ◆ The output sequence is obtained in one call of the NN (see later for the implementation)
  - ◆ Arbitrary output length

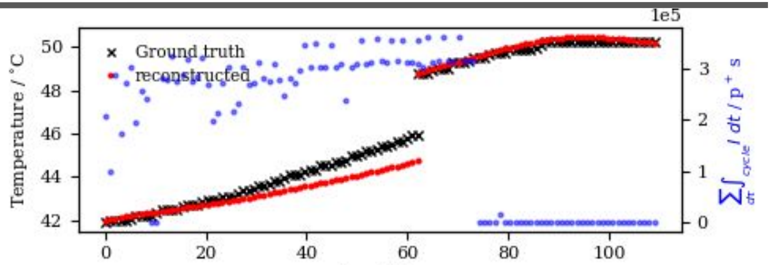
Then backpropagation step using this predicted sequence

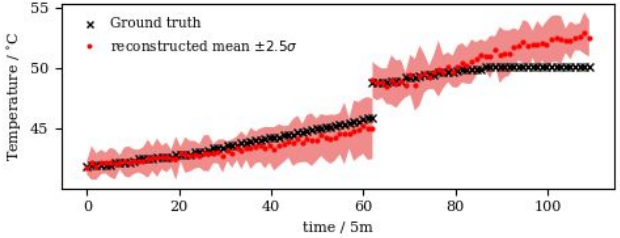


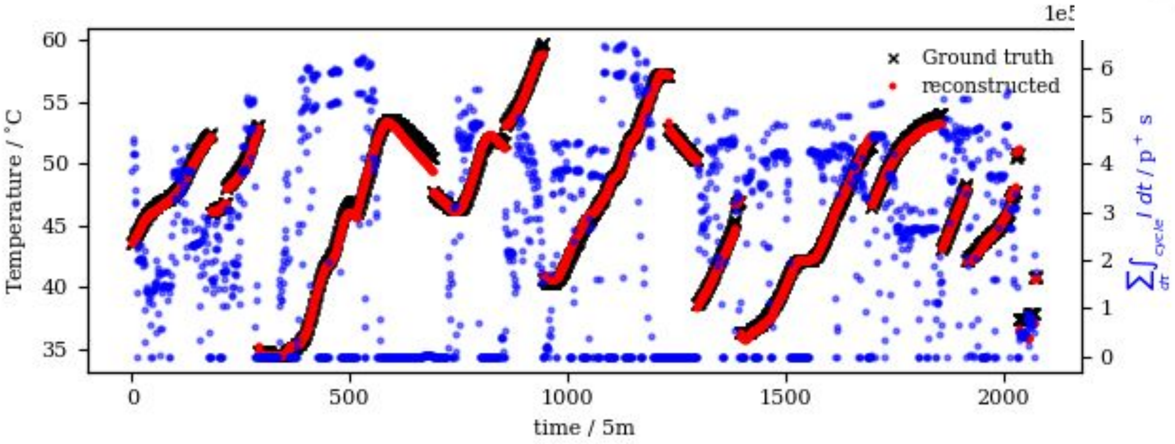
$$\mathcal{L} = \frac{1}{N}\sum_i (\mathbf{Y}_i - \hat{\mathbf{Y}}_i)^2$$

LSTM model for MKP: results

- ➔ Trained model repreduced training and validation data set almost perfectly
  - ◆ Error in the order of a couple of degrees on test dataset
- ➔ Bayesian version looking also promising



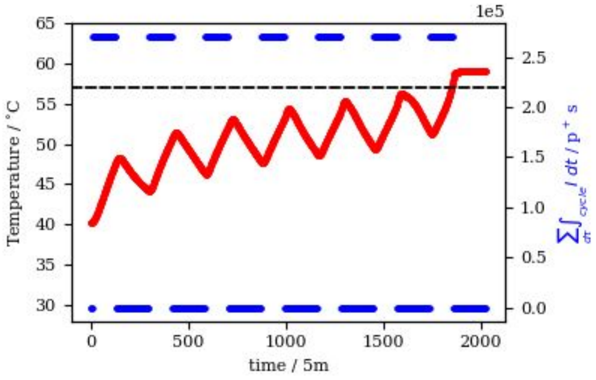


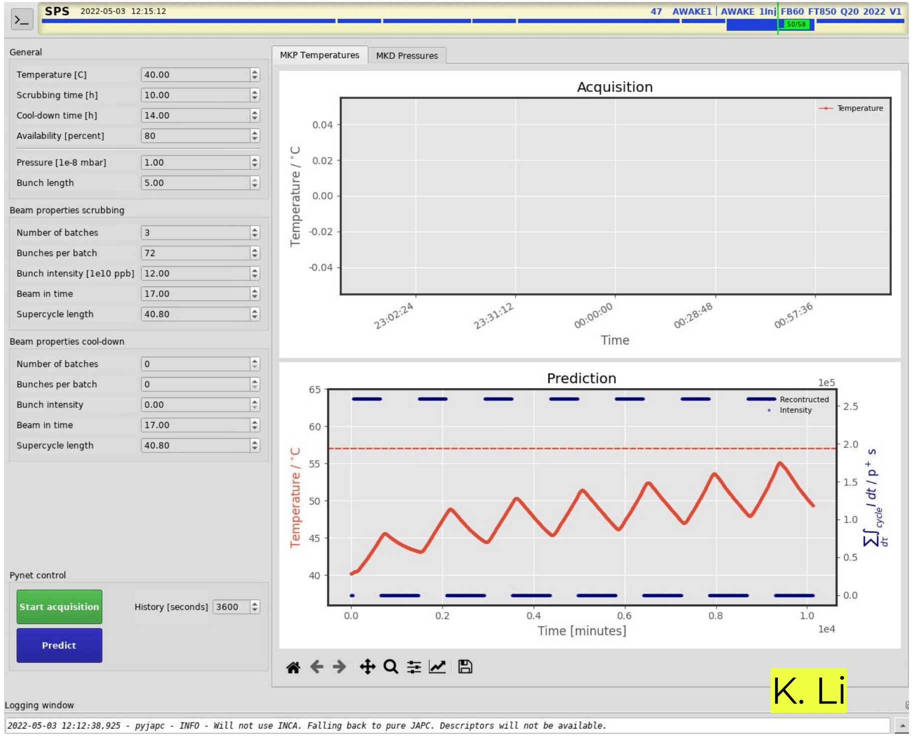


Summary and prediction

- Testing prediction on different scenarios
- Summary:
  - Model results very promising
  - Model ready and used in CCC to make estimation of time left for HI beams
  - Model not capable to extrapolate
- Need to include physics in the model…

Case 4



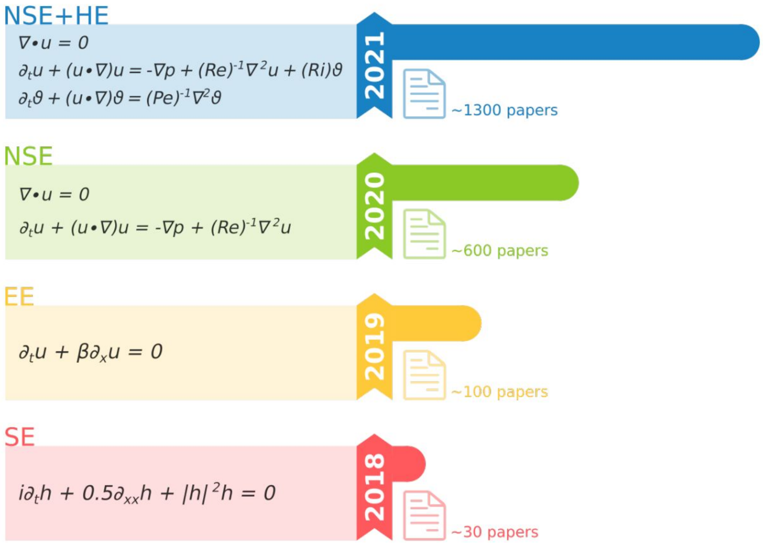


K. Li

Physics Informed Neural Networks (PINN)

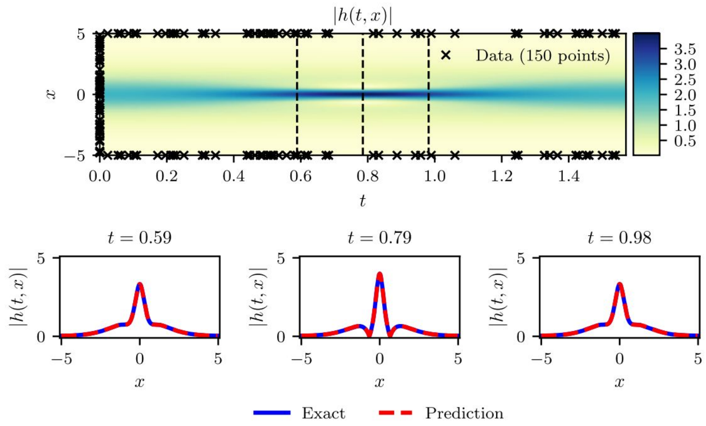
Physics Informed Neural Networks

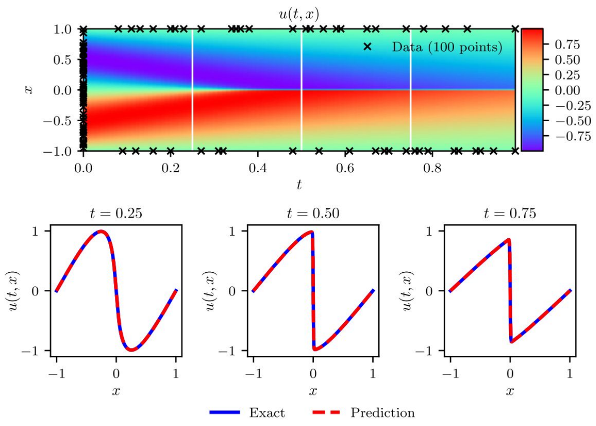
- ➔ Embedding physics knowledge in NN is becoming very common
- ➔ Very complete summary of applications [11] (figure taken from [11])



Physics Informed Neural Networks

- ➔ First proposed to solve nonlinear PDE [10] (all plots from [10])
- ➔ Basically using boundary and initial conditions values, NN can interpolate the whole system dynamics "knowing" the PDE that describe the system
  - ◆ At the same time though, one can just use a physics loss term…it doesn't have to be a PDE system (**IMO**)

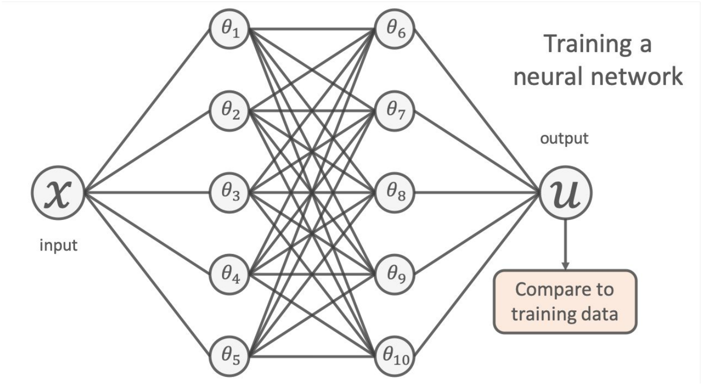




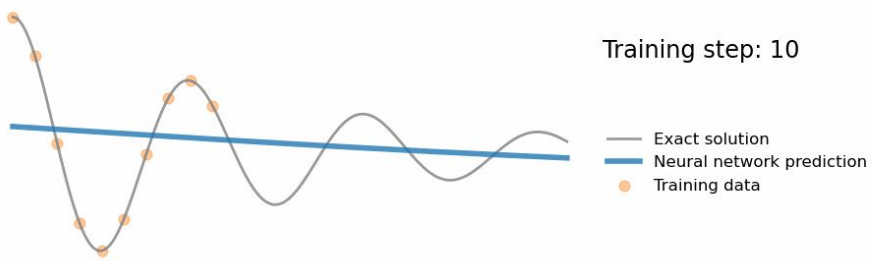
Physics Informed Neural Networks

➔ DNN cannot extrapolate beyond the training domain...which is exactly what we would expect from interpolation function

min(Loss) => Loss = Mean(data - prediction)^2



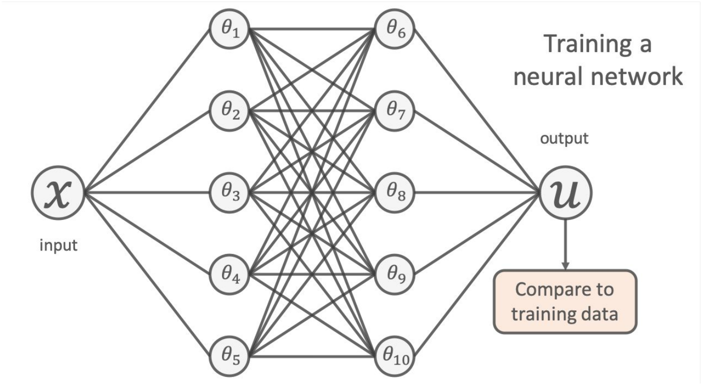
Source: [8]



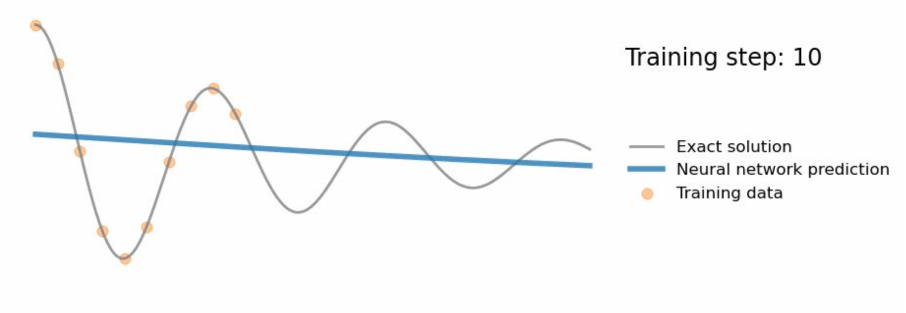
Physics Informed Neural Networks

➔ DNN cannot extrapolate beyond the training domain...which is exactly what we would expect from interpolation function

$$\mathcal{L} = \sum_{i}^{N} (u(x_i) - \hat{u}(x_i, \theta))^2$$



Source: [8]



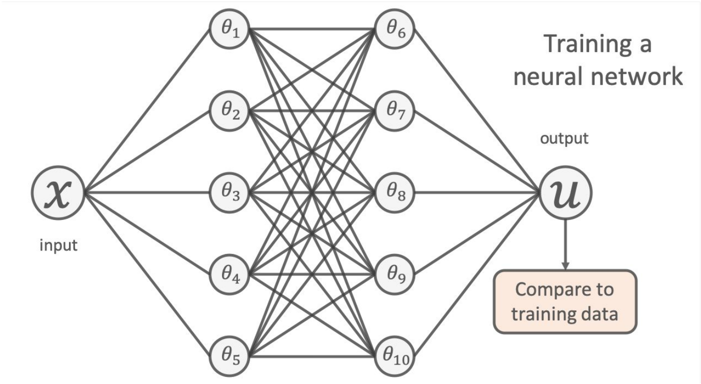
Physics Informed Neural Networks

➔ DNN cannot extrapolate beyond the training domain...which is exactly what we would expect from interpolation function

$$\mathcal{L} = \sum_{i}^{N} (u(x_i) - \hat{u}(x_i, \theta))^2$$

➔ Go beyond data domain => more information needed:

min(Loss) => Loss = Mean(data - prediction)^2
+ Additional_info(prediction)

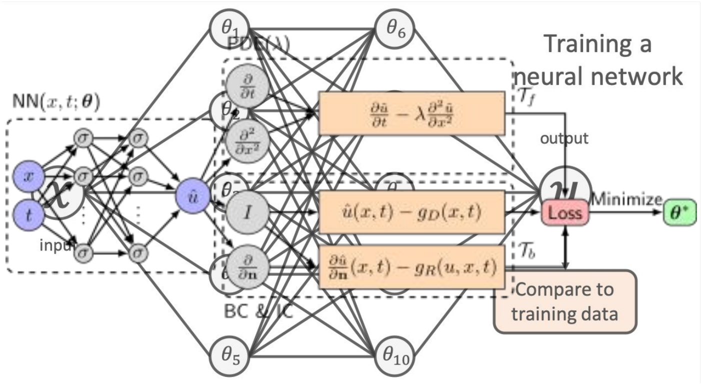


Source: [8]

Physics Informed Neural Networks

- DNN cannot extrapolate beyond the training domain...which is exactly what we would expect from interpolation function

$$\mathcal{L} = \sum_{i}^{N} (u(x_i) - \hat{u}(x_i, \theta))^2$$



Source: [8]

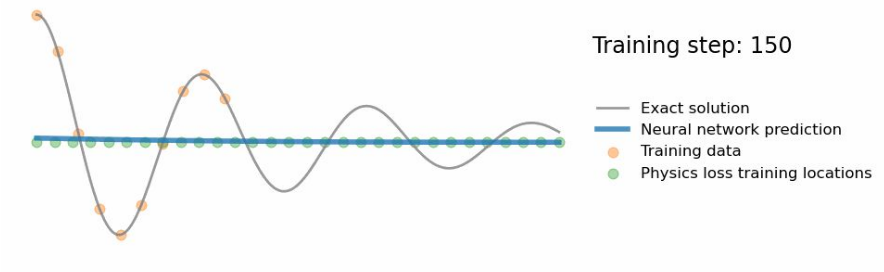
- Go beyond data domain => more information needed:

min(Loss) => Loss = Mean(data - prediction)^2
\+ Additional_info(prediction)

$$\mathcal{L}_2 = 1/M \sum_{j}^{M} \left(\frac{\partial^2 \hat{u}}{\partial x^2} - \frac{\partial \hat{u}}{\partial t}\right)^2$$

$$\mathcal{L}_3 = \hat{u}(x, t=0) - f(x)$$

$$\mathcal{L}_4 = \hat{u}(x=0, t) - u_0$$

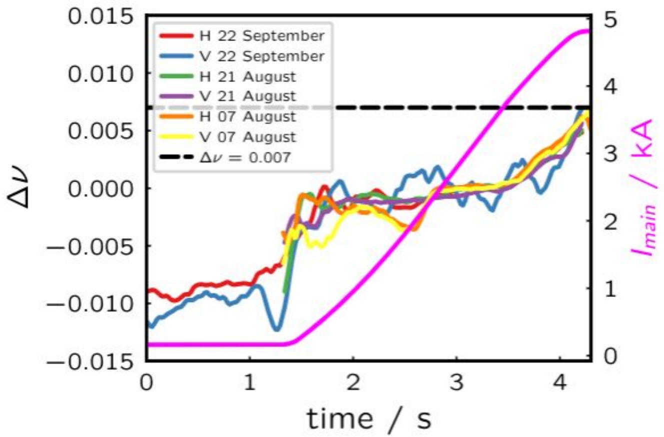


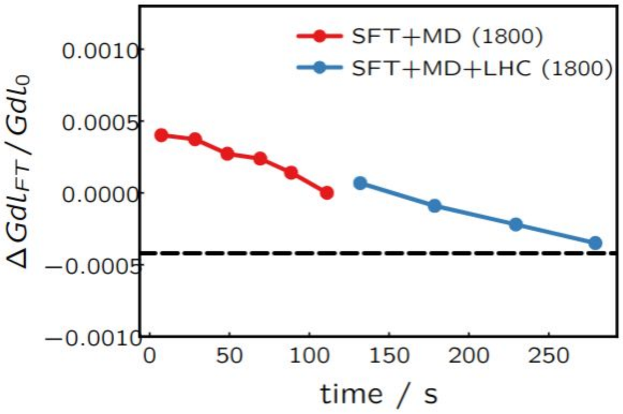
$$\mathcal{L}_{tot} = \alpha \mathcal{L}_1 + \beta \mathcal{L}_2 + \gamma \mathcal{L}_3 + \eta \mathcal{L}_4$$

Hysteresis prediction for slow extraction

➔ Hysteresis on the main SPS quadrupoles responsible for extracted beam quality degradation [9]

- Beam based measurements highlighted tune variation
- Magnetic measurements on spare quadrupole showed field variation compatible with beam observations

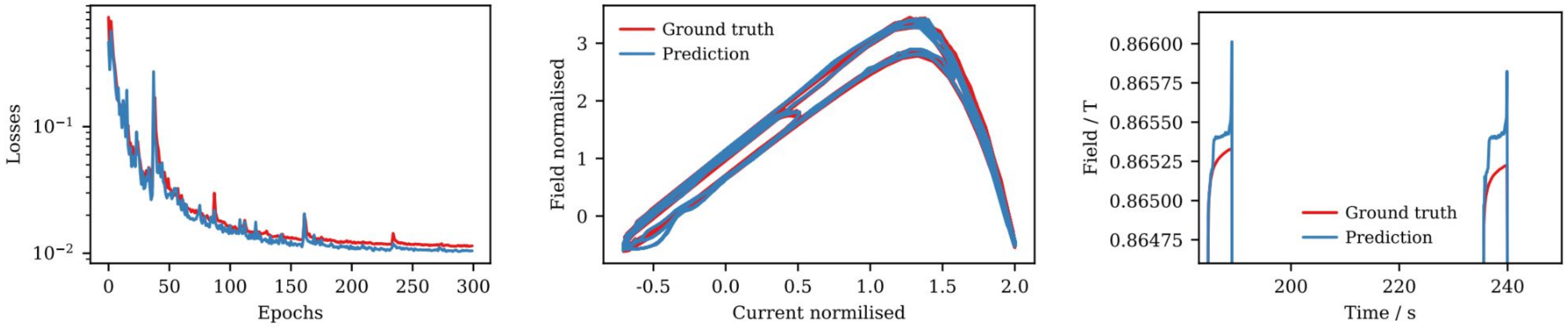




Hysteresis prediction for slow extraction

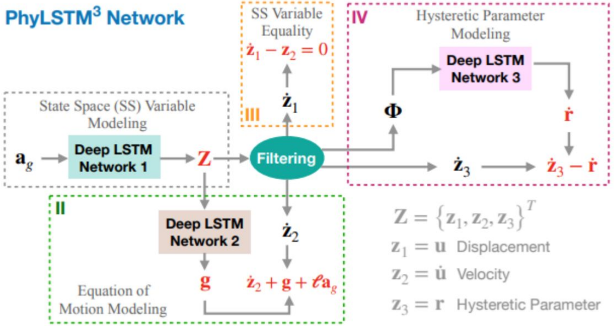
➔ Hysteresis on the main SPS quadrupoles responsible for extracted beam quality degradation [9]
  - Beam based measurements highlighted tune variation
  - Magnetic measurements on spare quadrupole showed field variation compatible with beam observations

➔ Classic model possible but complicated, simple NN not enough! **We need more information!**



Hysteresis modelling

- Hysteresis is rather common in physics and many other fields (chemistry, biology, economics...)
- Modelling is rather challenging: main models Preisach and Bouc-Wen
- In [11], PINN applied to hysteresis modelling of behaviour of structures under seismic excitation
  - This was our inspiration => very similar problem but different system
- Here is the model used in [11]:



PINN for SPS quadrupole hysteresis

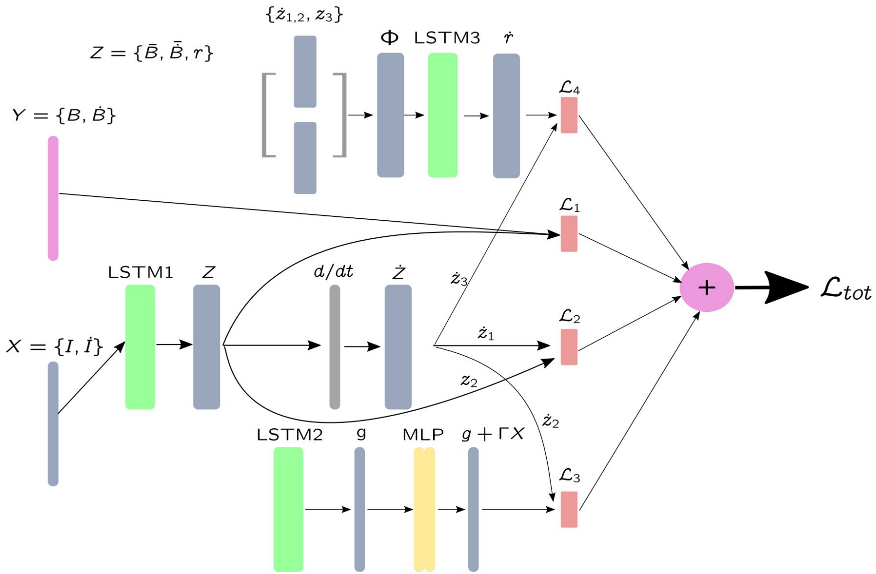
- A generic hysteretic model can be written as [11]:

$$a\ddot{y}(t) + b(y, \dot{y}) + r(y, \dot{y}, y(\tau)) = \Gamma x(t) \qquad \ddot{y} + g = \Gamma x$$

- Using input x = {I, dI/dt} and output y = {B, dB/dt}, we wrote our model and loss:

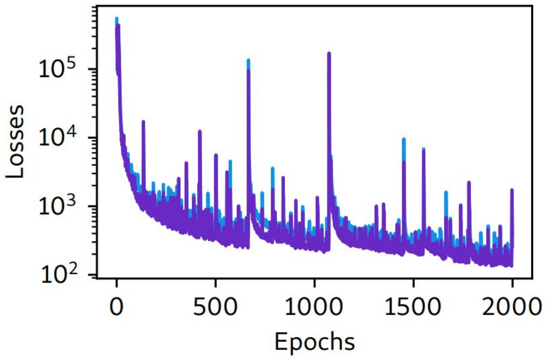
$$\mathcal{L}_1 = MSE(z_1(\theta_1) - y_1) + MSE(z_2(\theta_1) - y_2)$$
$$\mathcal{L}_2 = MSE(\dot{z}_1(\theta_1) - z_2(\theta_1))$$
$$\mathcal{L}_3 = MSE(\dot{z}_2(\theta_1) + MLP(g(\theta_1, \theta_2), x_1))$$
$$\mathcal{L}_4 = MSE(\dot{r}(\theta_1, \theta_3) - \dot{z}_3(\theta_1)); \dot{r} = f(\Phi); \Phi = \{\Delta z_2, r\}$$

$$\mathcal{L}_{tot} = \alpha\mathcal{L}_1 + \beta\mathcal{L}_2 + \gamma\mathcal{L}_3 + \eta\mathcal{L}_4$$



PINN for SPS quadrupole hysteresis

- After many attempts, we managed to train successfully one PINN for hysteresis prediction
  - Not fully optimised yet
  - Not enough data to make a proper general model for SPS quadrupoles
  - Hyperparameters not tuned yet

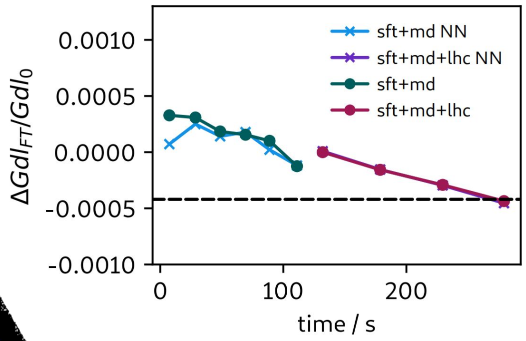


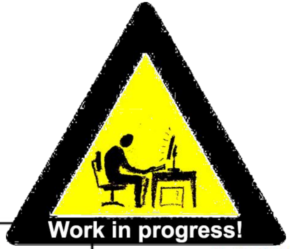
PhyLSTM^3

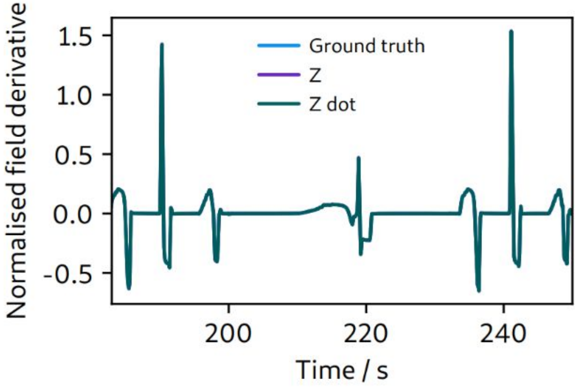
```
(relu): LeakyReLU(negative-slope=0.01)
(lstm0): LSTM(1, 350, num-layers=3, batch-first=True, dropout=0.2)
(fc0): Linear(in-features=350, out-features=175, bias=True)
(fc01): Linear(in-features=175, out-features=3, bias=True)
(gradient): GradientTorch()
(lstm): LSTM(3, 350, num-layers=3, batch-first=True, dropout=0.2)
(fc1): Linear(in-features=350, out-features=175, bias=True)
(fc11): Linear(in-features=175, out-features=1, bias=True)
(lstm3): LSTM(2, 350, num-layers=3, batch-first=True, dropout=0.2)
(fc2): Linear(in-features=350, out-features=175, bias=True)
(fc21): Linear(in-features=175, out-features=1, bias=True)
(g-plus-x): Sequential(
(0): Linear(in-features=2, out-features=350, bias=True)
(1): ReLU()
(2): Linear(in-features=350, out-features=1, bias=True))
```

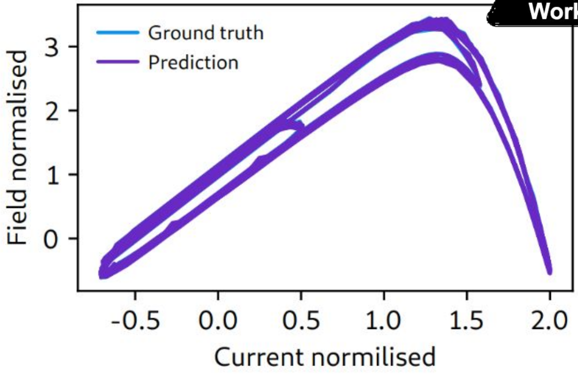
PINN for SPS quadrupole hysteresis

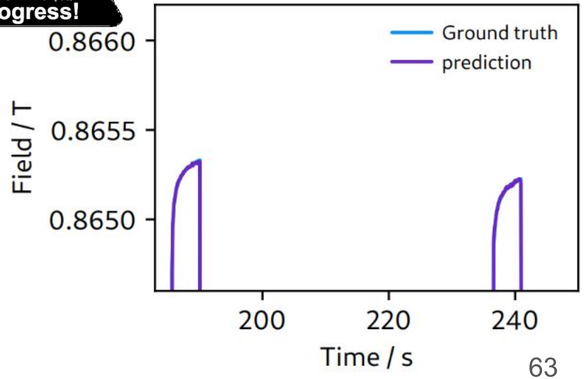
➔ After many attempts, we managed to train successfully one PINN for hysteresis prediction
- Not fully optimised yet
- Not enough data to make a proper general model for SPS quadrupoles
- Hyperparameters not tuned yet











PINN simple example

➔ Let's see a simple example we can quickly solve

➔ **Problem example:**

$$u_t - K u_{xx} = 0; 0 < x < L, t > 0$$

➔ With initial and boundary conditions (Dirichlet):

$$u(x, t = 0) = f(x) \equiv x(x^2 - 3Lx + 2L^2), 0 \leq x \leq L$$

$$u(x = 0, t) = u(x = L, t) = 0.0, t > 0$$

➔ **We can see in two ways:**

- Solve a IVP => PINN as PDE solver
- u(x, t=0) = f(x) are data (it could also be x(x=0, t)) => PINN with data

PINN simple example

- **Code example on indico:** here
- NN definition (see Andreas' slides and tutorial)
- PDE problem definition (with derivatives)
- Training loop

PINN simple example

- ➔ Code example on indico: here
- ➔ **NN definition** (see Andreas' slides and tutorial)
- ➔ PDE problem definition (with derivatives)
- ➔ Training loop

```
class ModelNN(nn.Module):
    def __init__(self, layers=4, neurons=5):
        super().__init__()
        self.nn_list = []
        for i in range(layers):
            self.nn_list.append(nn.Linear(neurons, neurons))
            self.nn_list.append(nn.Sigmoid())
        self.dnn = nn.Sequential(
            nn.Linear(2, neurons),
            nn.Sigmoid(),
            *self.nn_list,
            nn.Linear(neurons, 1),
        )

    def forward(self, x, t):
        u_hat = self.dnn(torch.cat([x, t], dim=-1))
        return u_hat
```

PINN simple example

- ➔ Code example on indico: [here]
- ➔ NN definition (see Andreas' slides and tutorial)
- ➔ **PDE problem definition** (with derivatives)
- ➔ Training loop

```
def diff(y, x, require_graph=True):
    ones = torch.ones_like(y)
    (der,) = torch.autograd.grad(
        y, x, create_graph=True, grad_outputs=ones, allow_unused=True
    )
    if require_graph:
        der.requires_grad_()
    return der

K = 0.3
L = 2

def pde(x, t, model):
    u_hat = model(x, t)
    u_x = diff(u_hat, x)
    u_xx = diff(u_x, x)
    u_t = diff(u_hat, t)
    return u_t - K * u_xx

def u_ic_f(x):
    return x * (x**2 - 3 * L * x + 2 * L**2)
```

PINN simple example

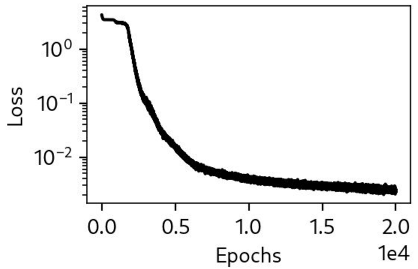
- ➔ Code example on indico: here
- ➔ NN definition (see Andreas' slides and tutorial)
- ➔ PDE problem definition (with derivatives)
- ➔ **Training loop**

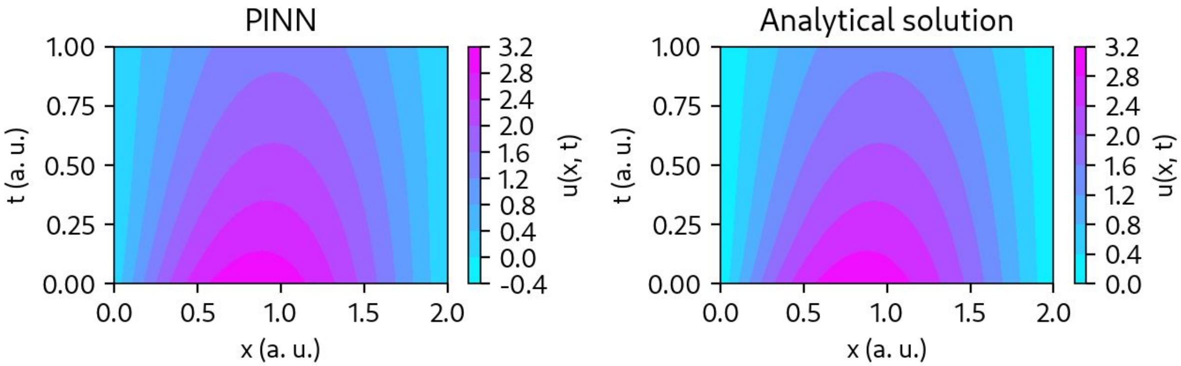
```
pde_target = torch.zeros((500, 1)).to(dev)

x_ic = torch.rand((500, 1)).to(dev) * L
t_ic = torch.zeros((500, 1)).to(dev)

u_ic = u_ic_f(x_ic)

x_bc = torch.zeros((500, 1)).to(dev) + L
x_bc_2 = torch.zeros((500, 1)).to(dev)
t_bc = torch.rand((500, 1)).to(dev)
u_bc = 0.0 * t_bc
u_bc_2 = 0.0 * t_bc

epochs = 20000
```

```
losses = []
progress_bar = trange(epochs, unit="epoch")
for epoch in progress_bar:
    optimiser.zero_grad()
    u_bc_hat = model_nn(x_bc, t_bc)
    l_bc = mse_loss(u_bc_hat, u_bc)

    u_bc_2_hat = model_nn(x_bc_2, t_bc)
    l_bc_2 = mse_loss(u_bc_2_hat, u_bc_2)

    u_ic_hat = model_nn(x_ic, t_ic)
    l_ic = mse_loss(u_ic_hat, u_ic)

    t = torch.rand((500, 1)).to(dev)
    t.requires_grad = True
    x = torch.rand((500, 1)).to(dev) * L
    x.requires_grad = True
    pde_hat = pde(x, t, model_nn)
    l_pde = mse_loss(pde_hat, pde_target)

    loss = l_ic + l_pde + l_bc + l_bc_2
    loss.backward()
    optimiser.step()
    losses.append(loss.item())
    progress_bar.set_postfix(loss=loss.item())
```

PINN simple example

- ➔ Results as compared with analytical solution
  - Indistinguishable!
- ➔ Caveats:
  - Training takes quite some time (well, not in this particular case!)
  - With data, need to balance properly the different loss function components
- ➔ Easily possible to extend to inhomogeneous cases





CERN Academic Lecture, Applications of computer vision and forecasting to the CERN accelerators, 5th May 2022

Summary

- ➔ CNNs can be used quite effectively in the accelerator complex
  - First results very promising
- ➔ LSTM-based models used for kicker heating predictions and hysteresis modelling
  - Physics loss fundamental for low data
- ➔ PINN introducing a new way to train NN
  - Include more information via problem definition and a priori knowledge
  - Great for “extrapolation”
  - Still quite a lot to explore, for example Maxwell equations solved with NN [12]
- ➔ What’s coming next?
  - Transformer (or attention) based NN are destroying the competition in NLP, time series forecasting, image classification... => we should look into this ASAP!

Thank you very much!