

Lecture 1: Foundations of Deep Learning

Aobo Li
Halicioğlu Data Science Institute
Department of Physics
UC San Diego

National Nuclear Physics Summer School, 07/2026

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

UC San Diego
PHYSICS

Course Objectives

1. Understand Deep Learning from 0 background

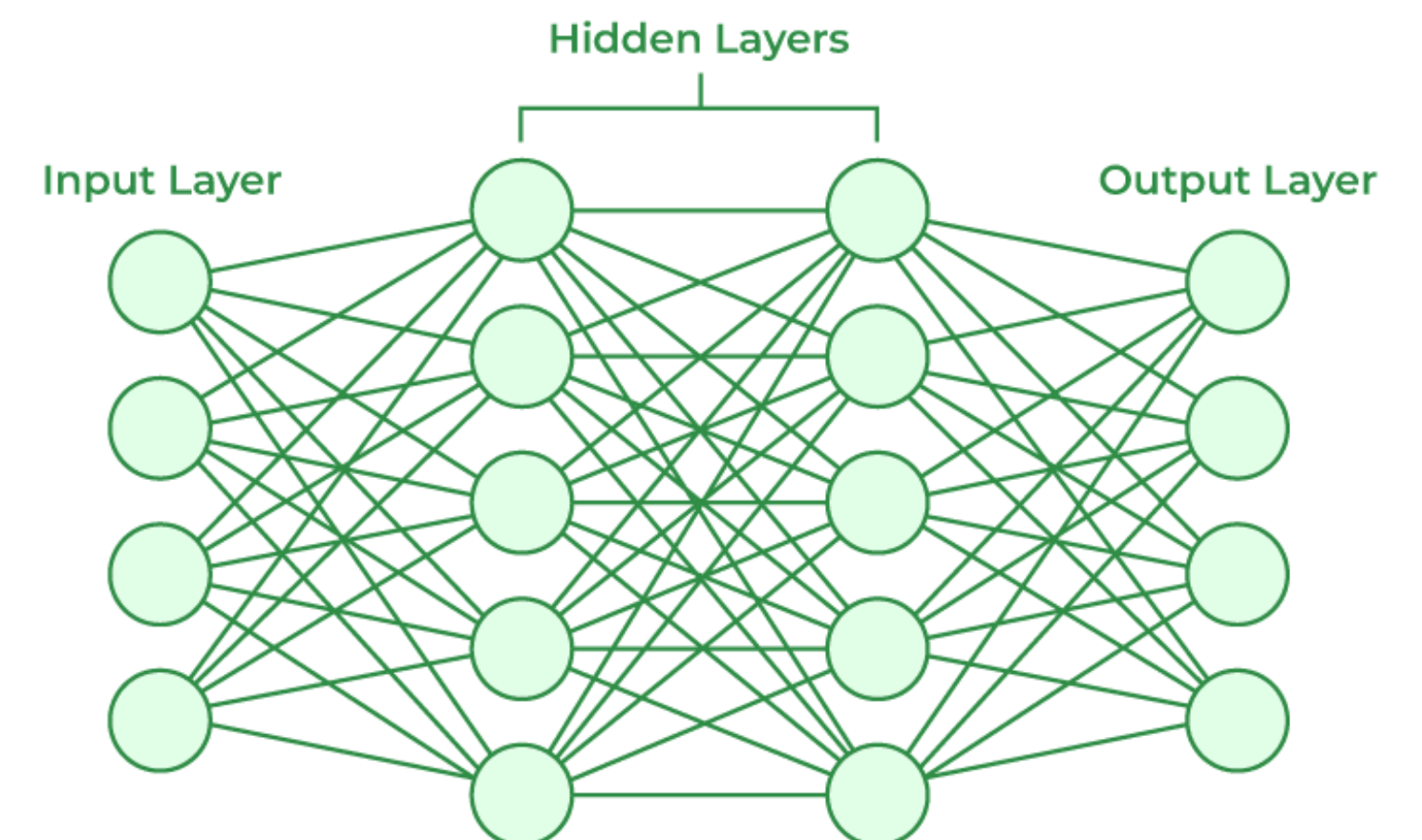
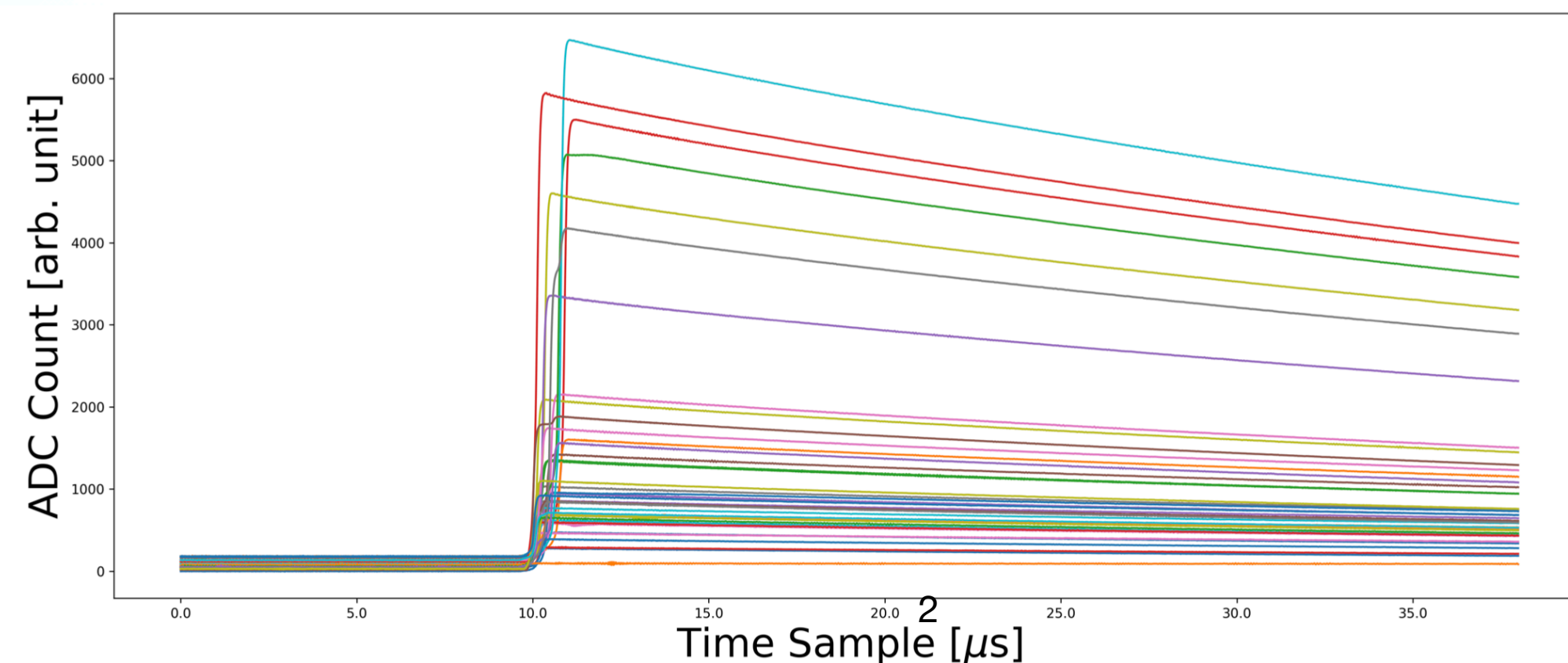
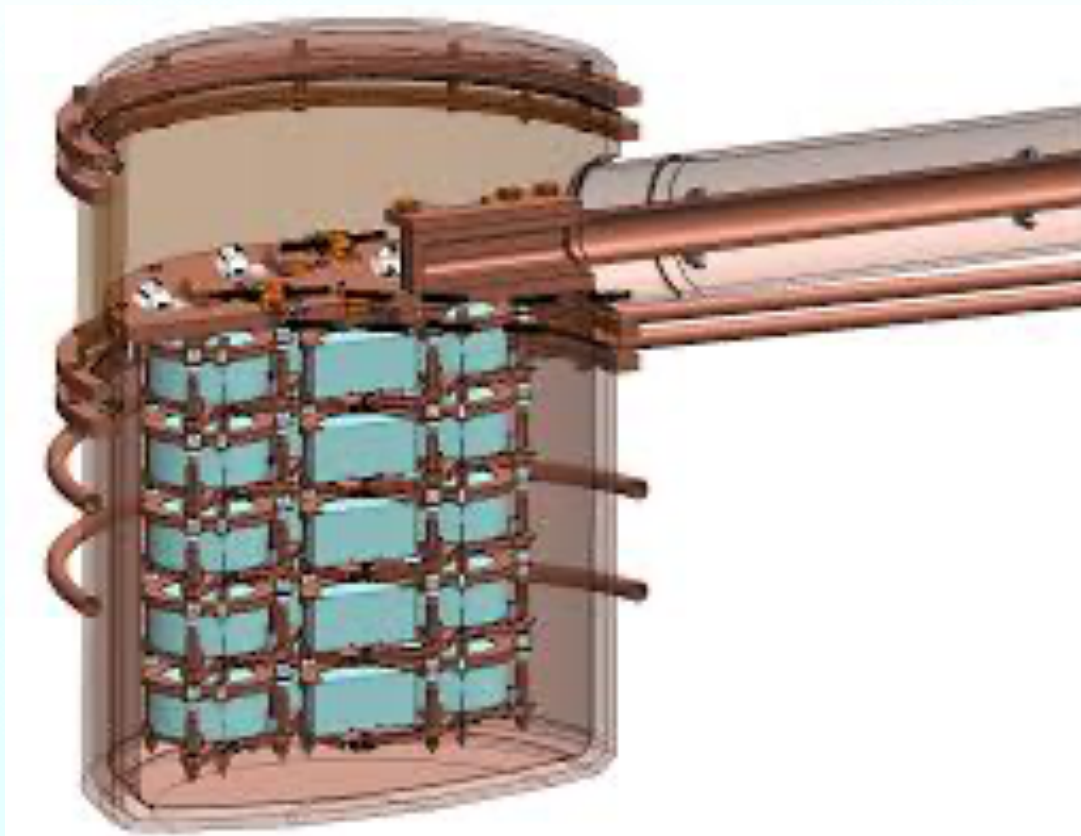
- Assuming some background in Python programming

2. Playing with real detector data

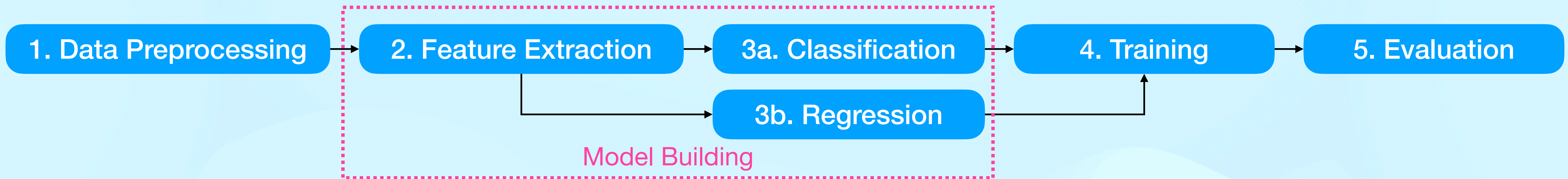
- Open time series data from the MAJORANA DEMONSTRATOR experiment

3. Fast Prototyping and Validation of Complex Deep Learning Models

- Next lecture: Large Language Model, AI Agent, and Vibe Coding



Course Outline



Theory

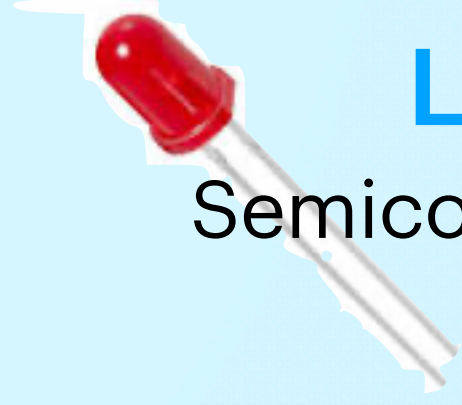
Important equations/theoretical derivations you should understand

Concept

Fundamental concepts that appears everywhere in AI/ML paper and textbook

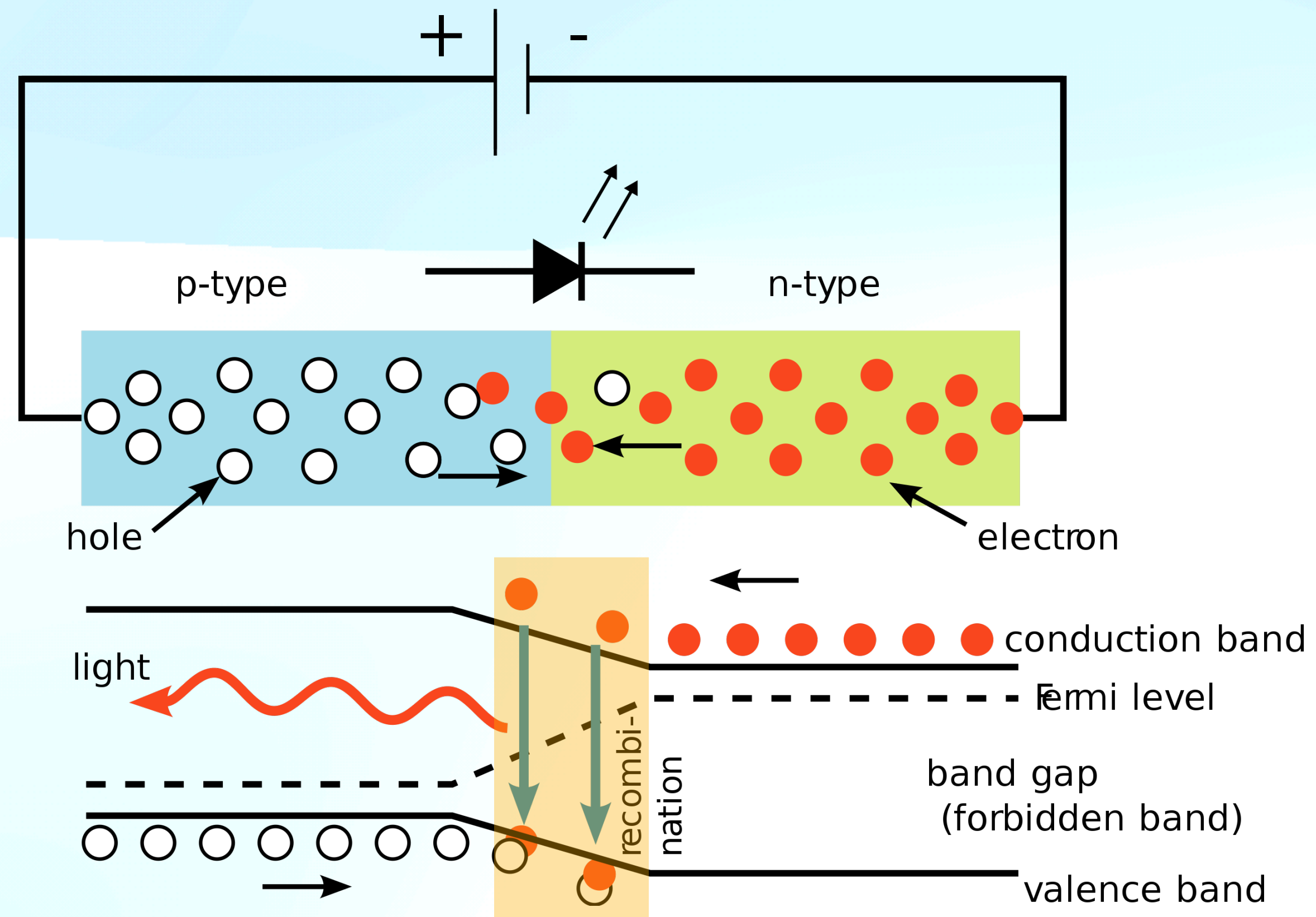
Code

The actual PyTorch implementation, useful for building your own models



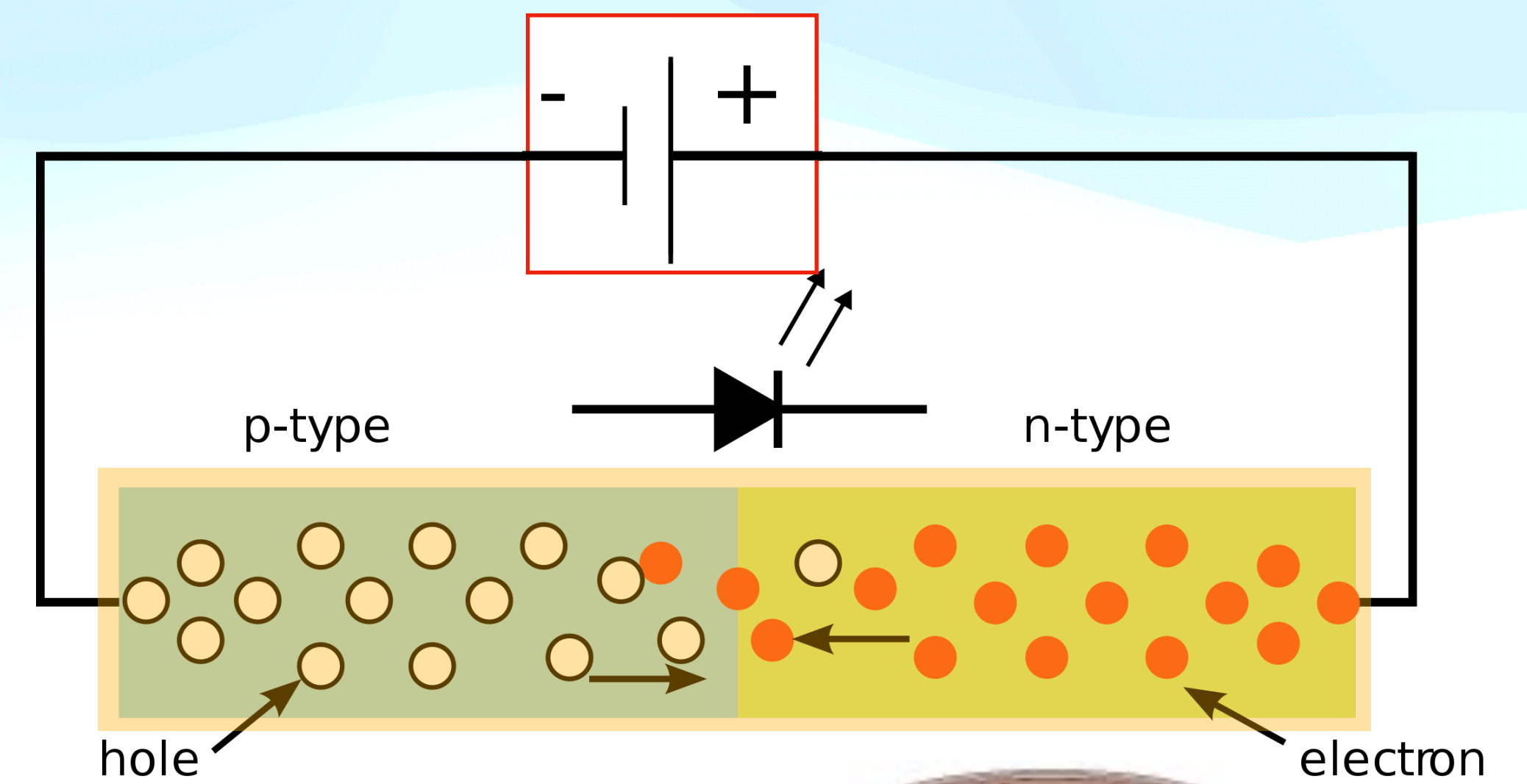
Light Emitting Diode (LED)

Semiconductor device with a **depletion region**

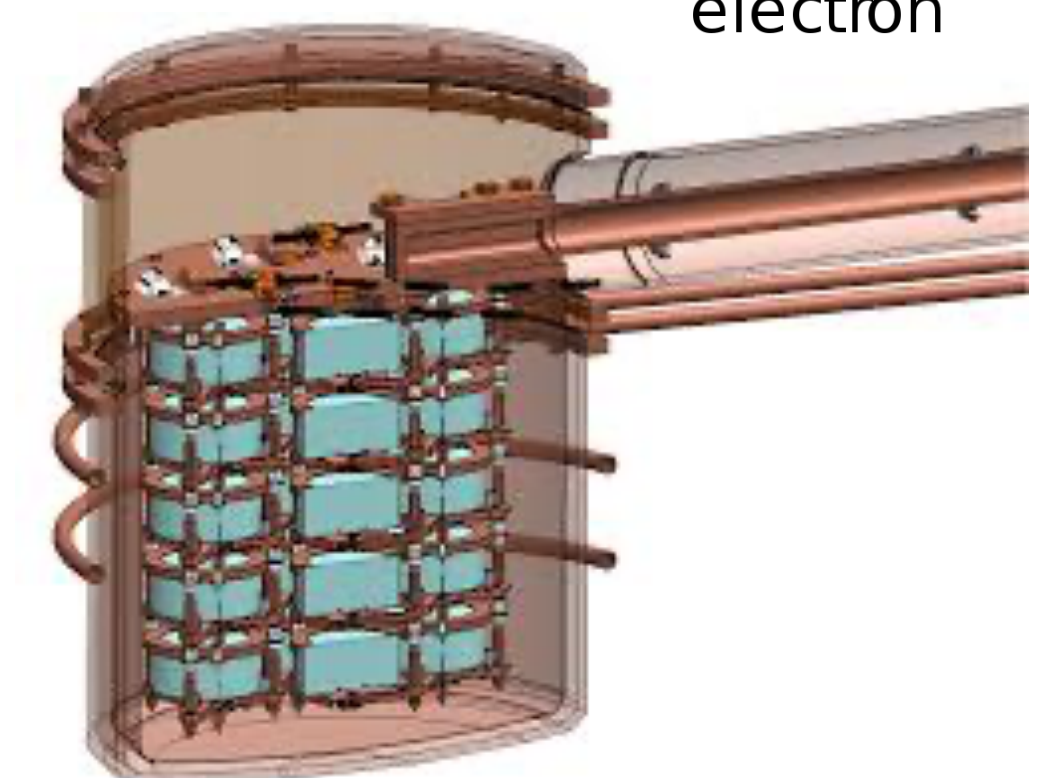


High Purity Ge Detector (HPGe)

⁷⁶Ge is a double-beta decay isotope
Reverse Bias: increase the size of **depletion region**

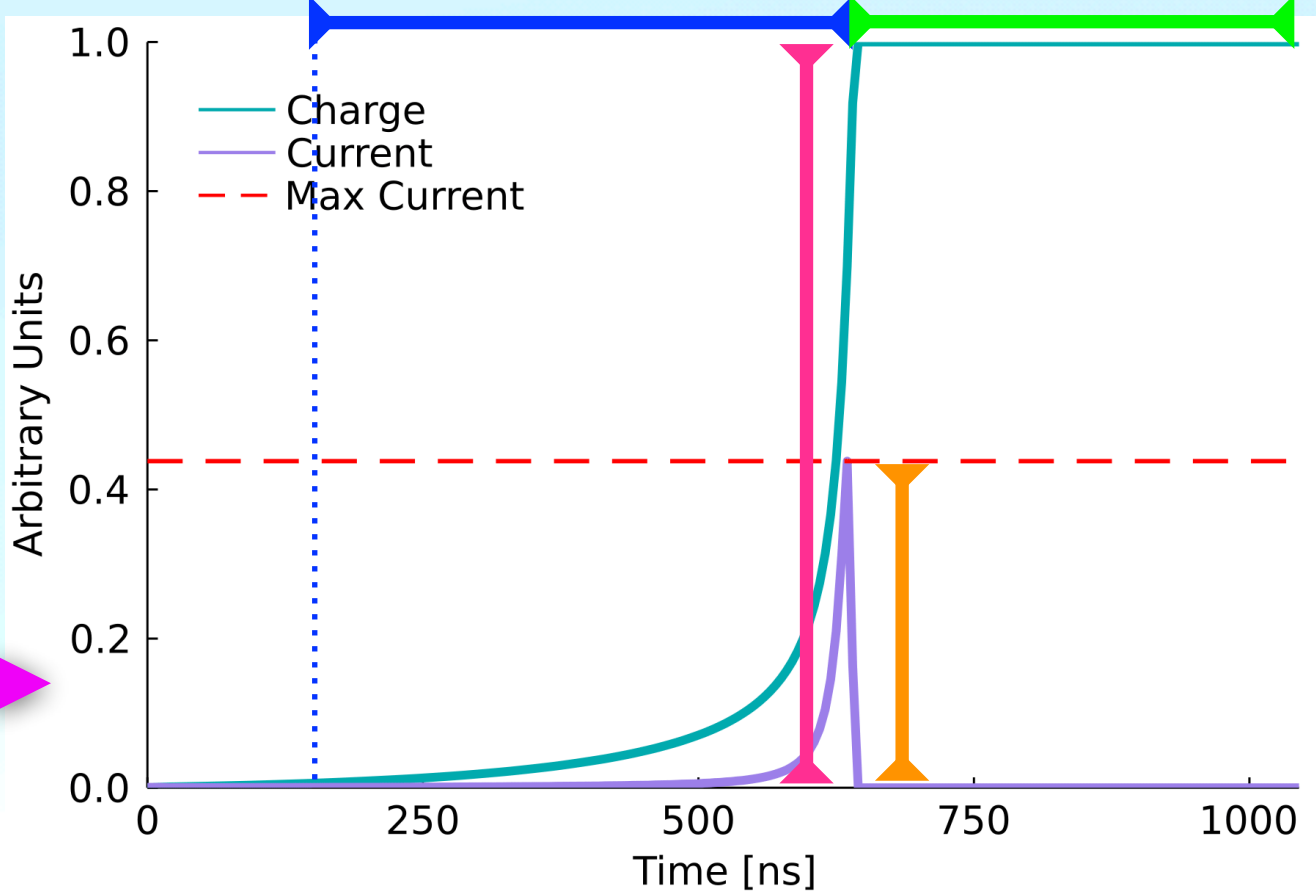
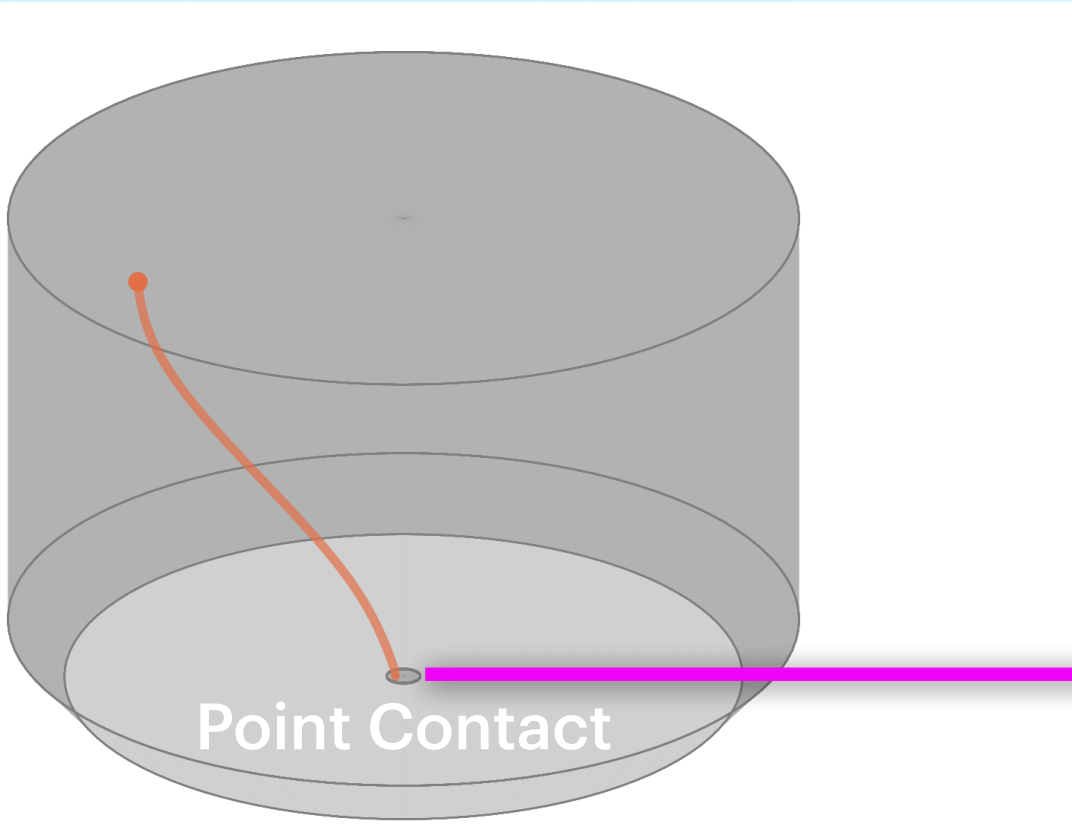
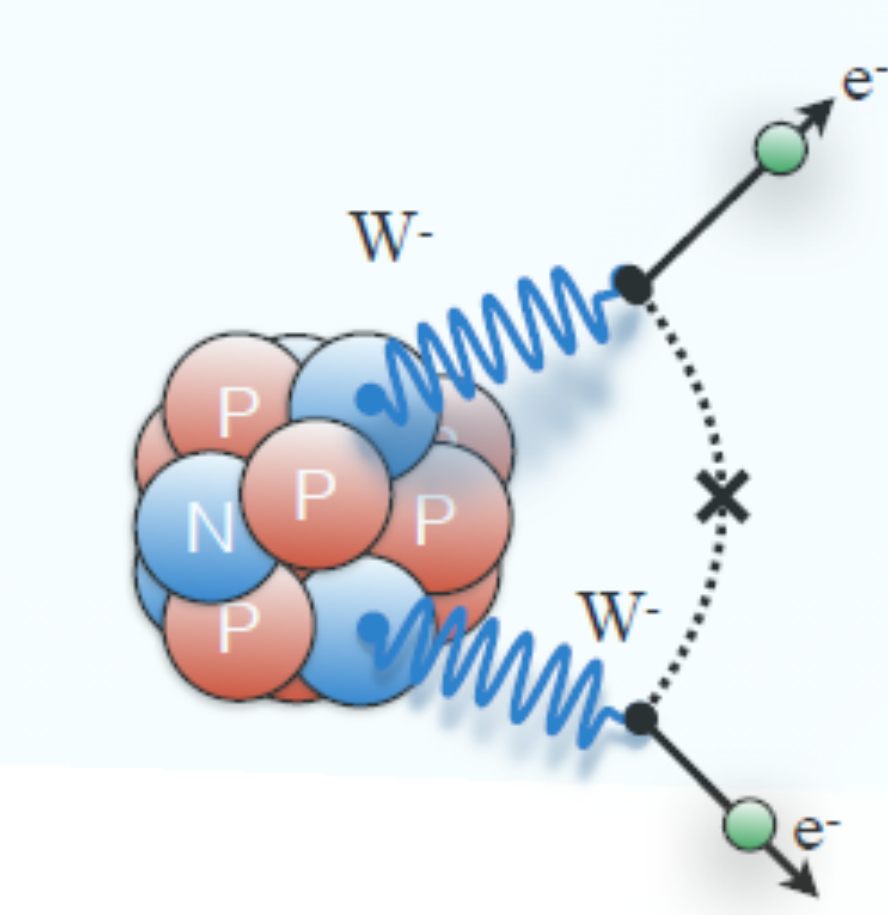


Detector Array





MAJORANA DEMONSTRATOR: HPGe Detector Array Experiment for $0\nu\beta\beta$ Search

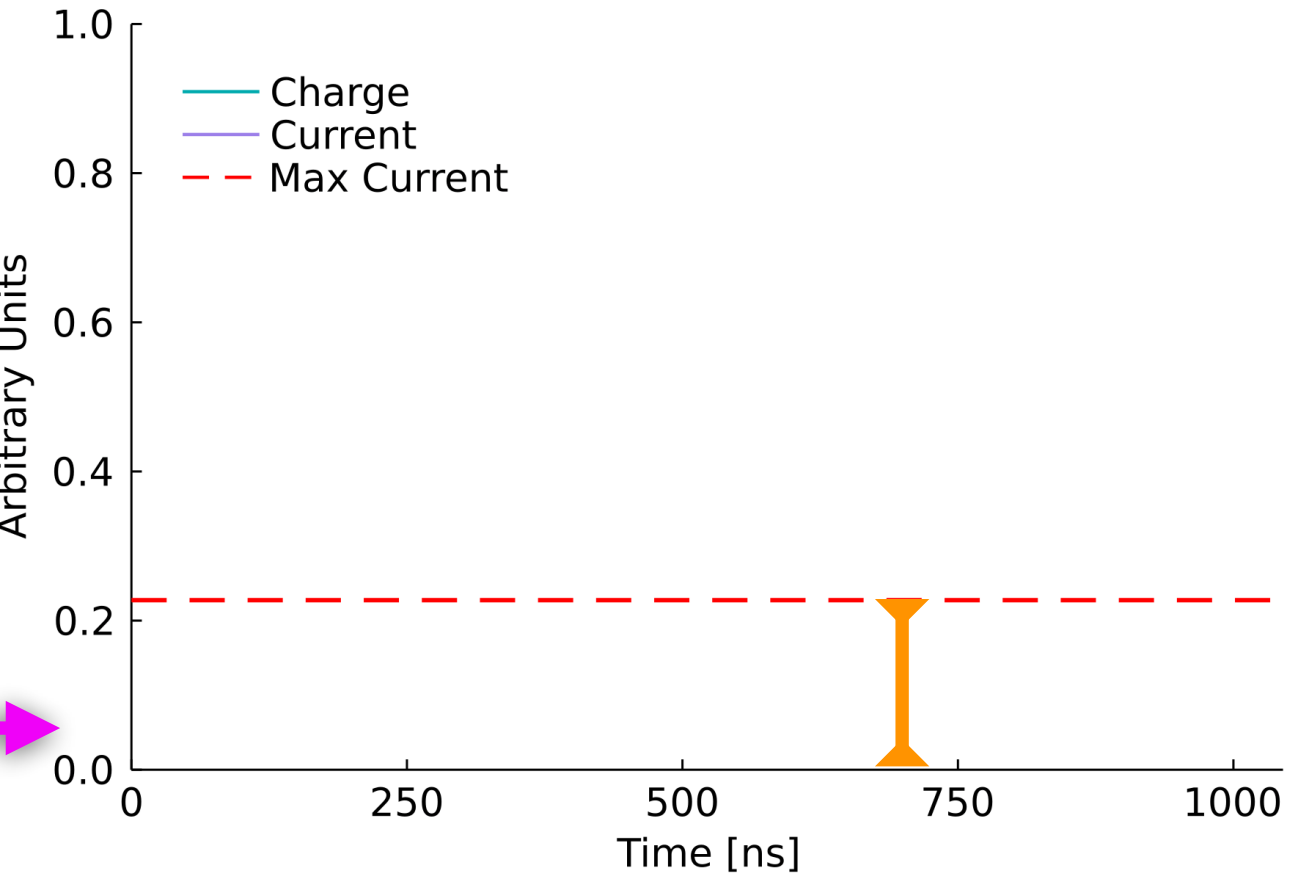
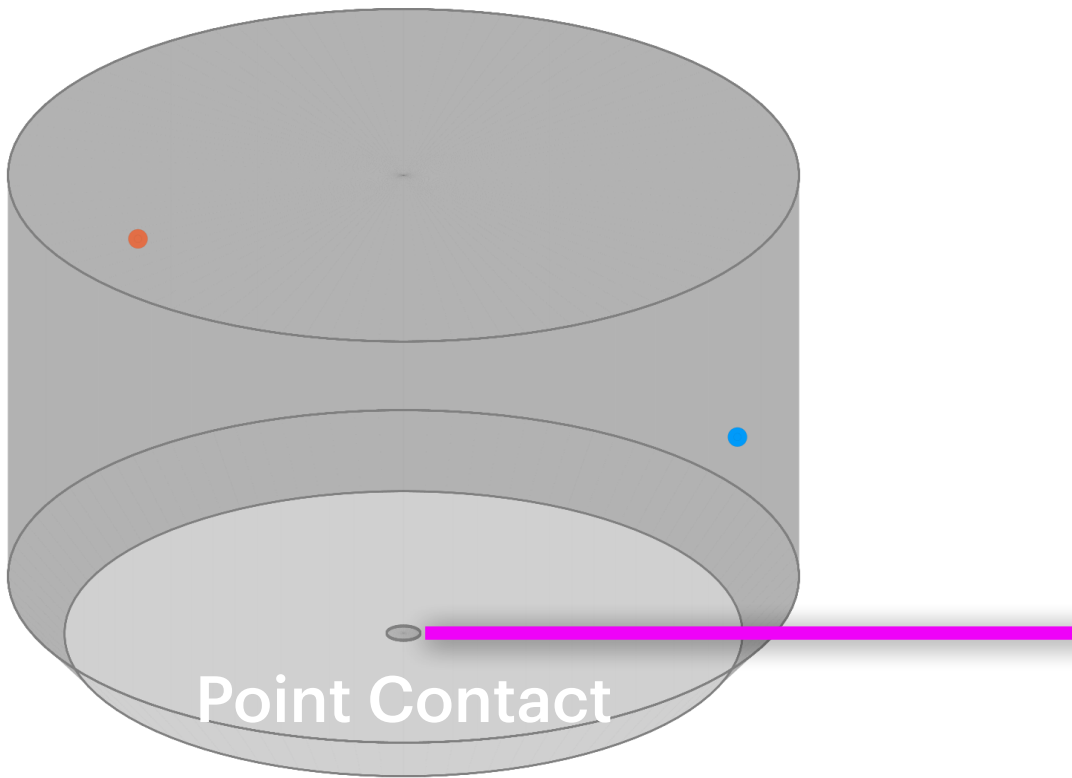
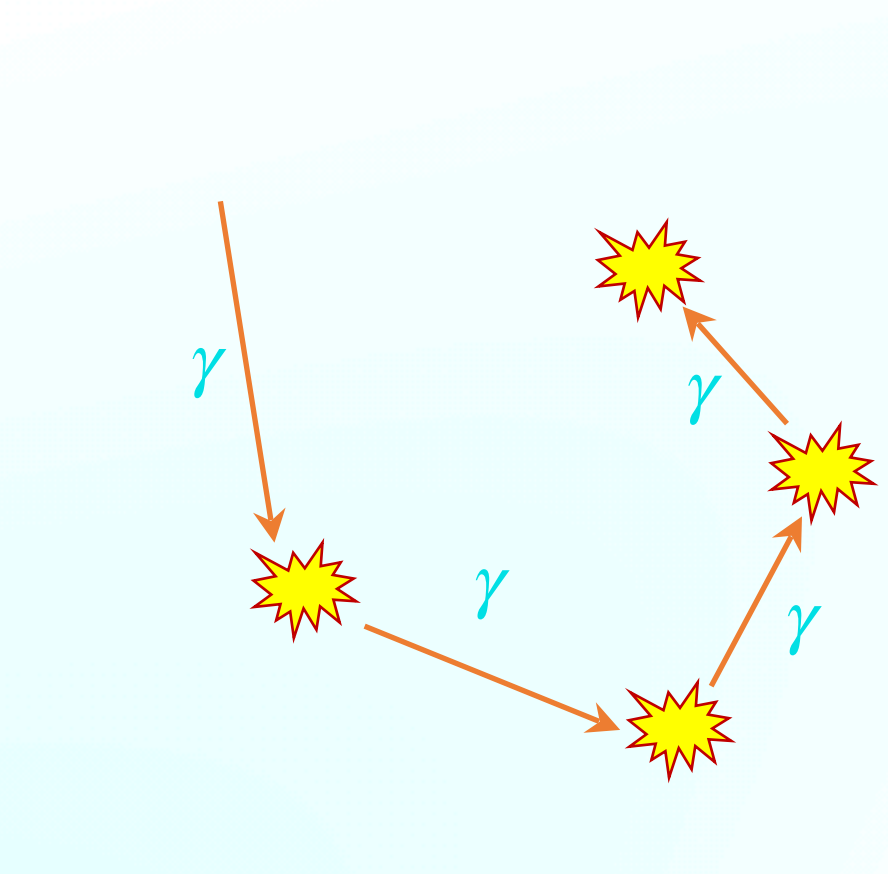


Single-Site Waveform

Tail Slope

For surface background rejection

Energy



5 Multi-Site Waveform

AvsE

For multi-site background rejection

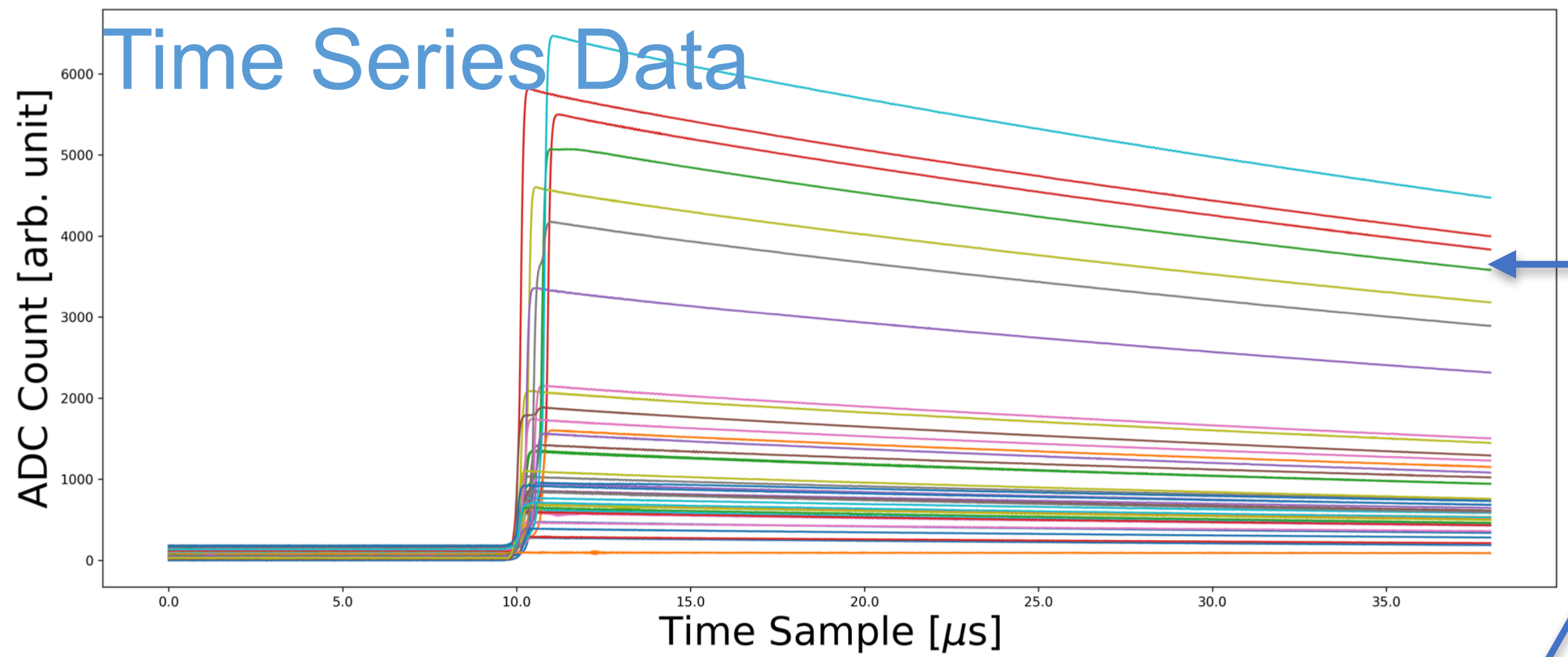
Drift Time

Reflect the location of incident particle

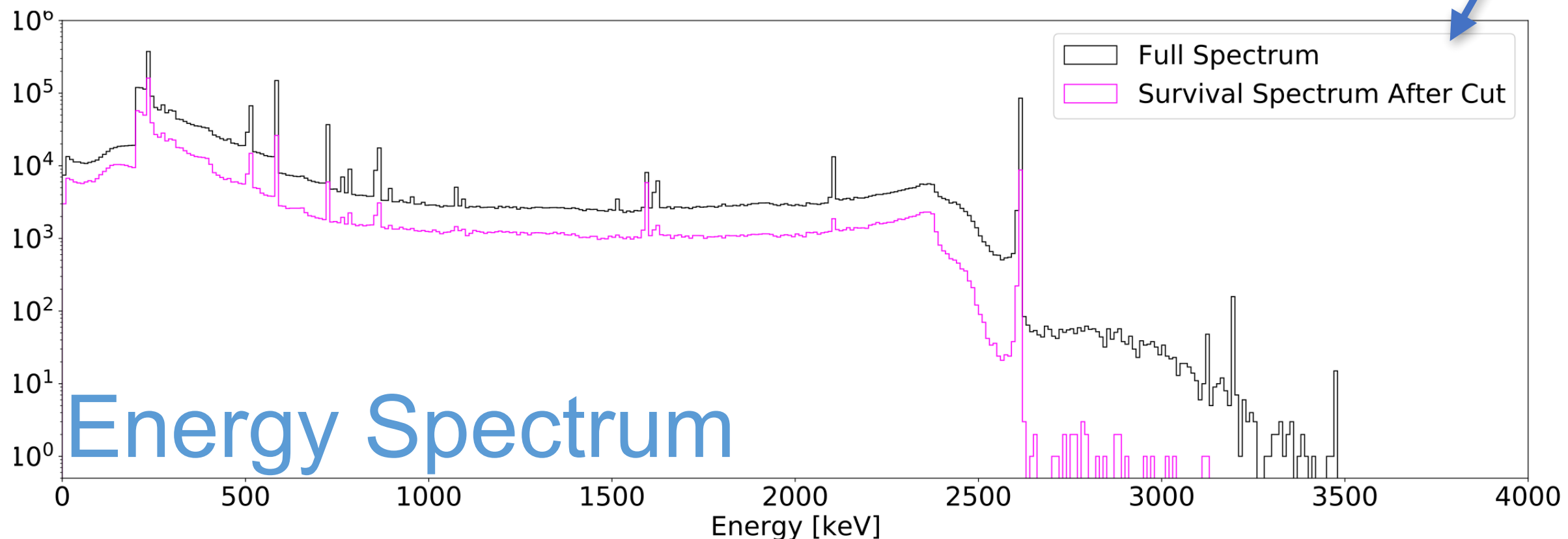


Dataset: <https://zenodo.org/records/8257027>
Document: <https://arxiv.org/abs/2308.10856>

This is real data from a state-of-the-art physics experiment, feel free to play with it to do your own machine learning project!



Field	Description	Data Type
raw_waveform	Detector Waveform	array(size=(3800,) dtype=float)
energy_label	Analysis Label	float
psd_label_low_avse	Analysis Label	binary
psd_label_high_avse	Analysis Label	binary
psd_label_dcr	Analysis Label	binary
psd_label_lq	Analysis Label	binary
tp0	Analysis Parameter	integer
detector	Metadata	integer
run_number	Metadata	integer
id	Metadata	integer



More on MJD: <https://www.energy.gov/science/np/articles/majorana-demonstrator-gives-its-final-answer-about-rare-nuclear-decay>



All files are stored in .hdf5 format

More readings on HDF5: https://web.mit.edu/fwtools_v3.1.0/www/H5.intro.html

```

MJD_Test_2.hdf5
├── detector (65000,)
├── energy_label (65000,)
├── id (65000,)
├── psd_label_dcr (65000,)
├── psd_label_high_avse (65000,)
├── psd_label_low_avse (65000,)
├── psd_label_lq (65000,)
├── raw_waveform (65000, 3800)
├── run_number (65000,)
└── tp0 (65000,)
  
```

Read file with h5py.File command

Code

```

with h5py.File(file_path, 'r') as file:
    energy = np.array(file["energy_label"])
    #only selecting higher energy event to make the task easier
    selection_flgae = energy>self.energy_threshold
    randind = np.arange(selection_flgae.sum())
    np.random.shuffle(randind)
    data = np.array(file["raw_waveform"])[selection_flgae][randind][:,500:1500]
    label = np.array(file["psd_label_low_avse"])[selection_flgae][randind]
    energy = np.array(file["energy_label"])[selection_flgae][randind]
  
```

Read "energy" field with indexing

Read "waveform" field with indexing

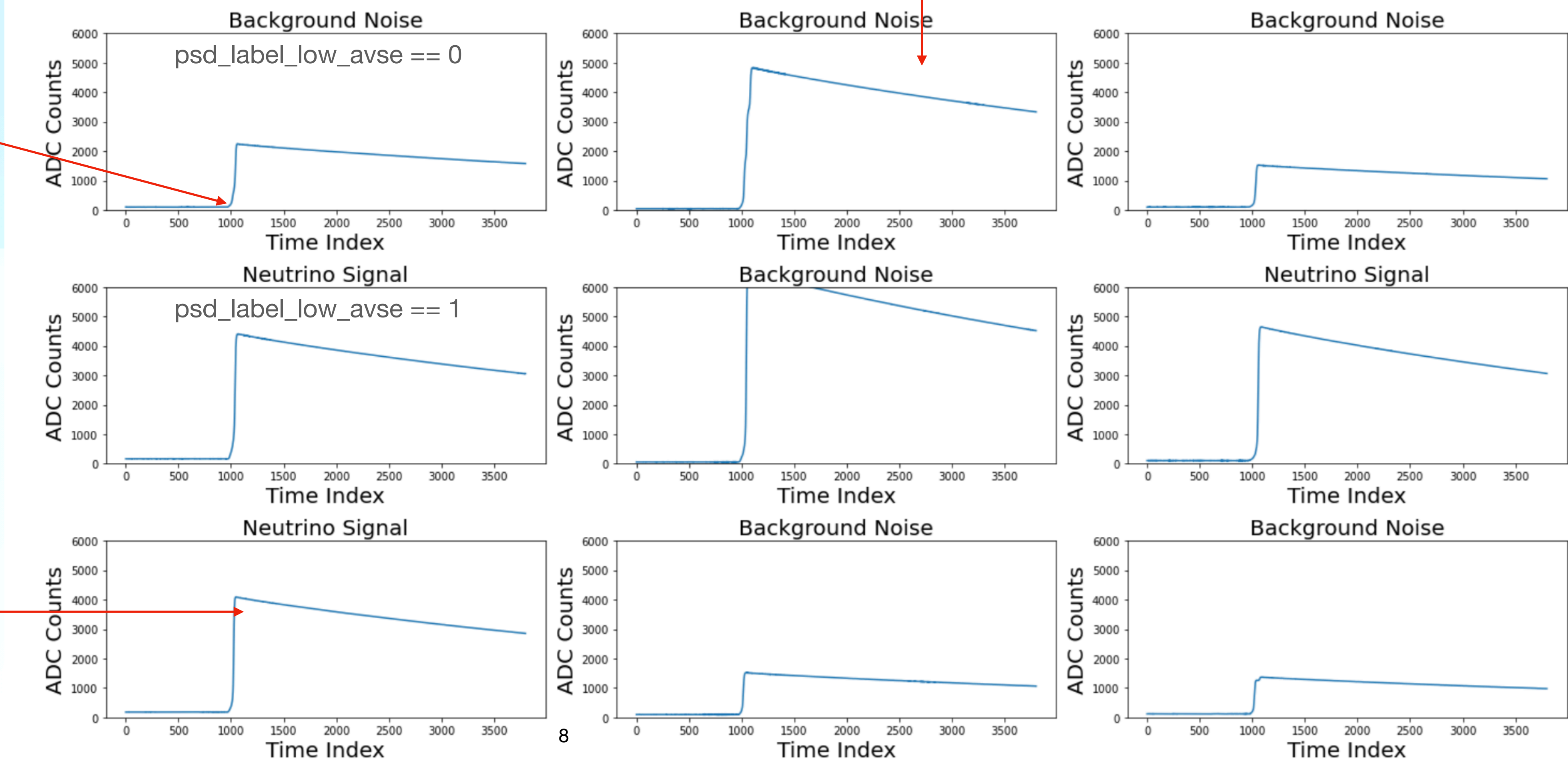
.....



Take a look at raw waveforms:

Alarm: tail region is too long, but it does not contain physics

Alarm: Waveforms may start rising at different positions



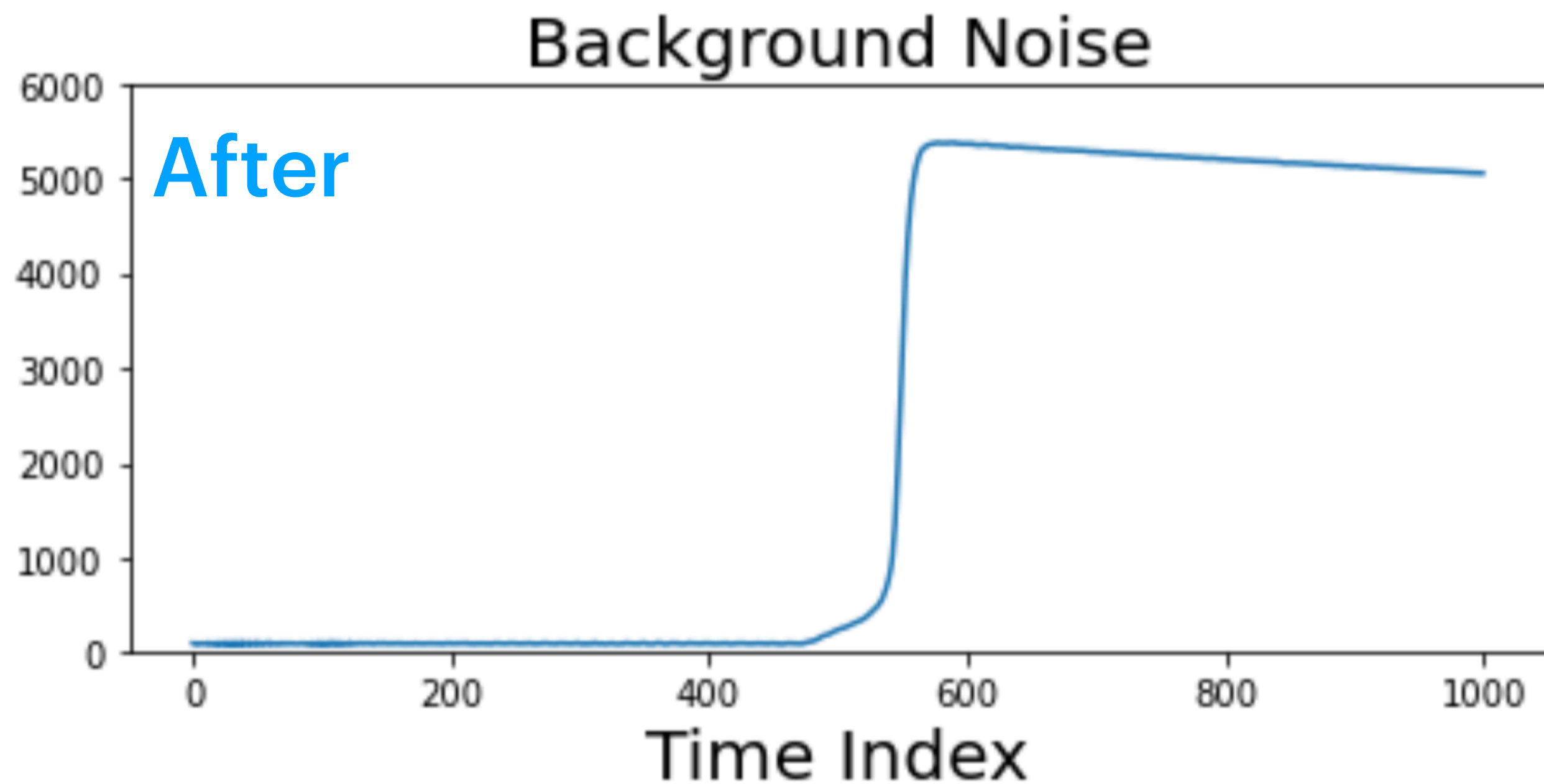
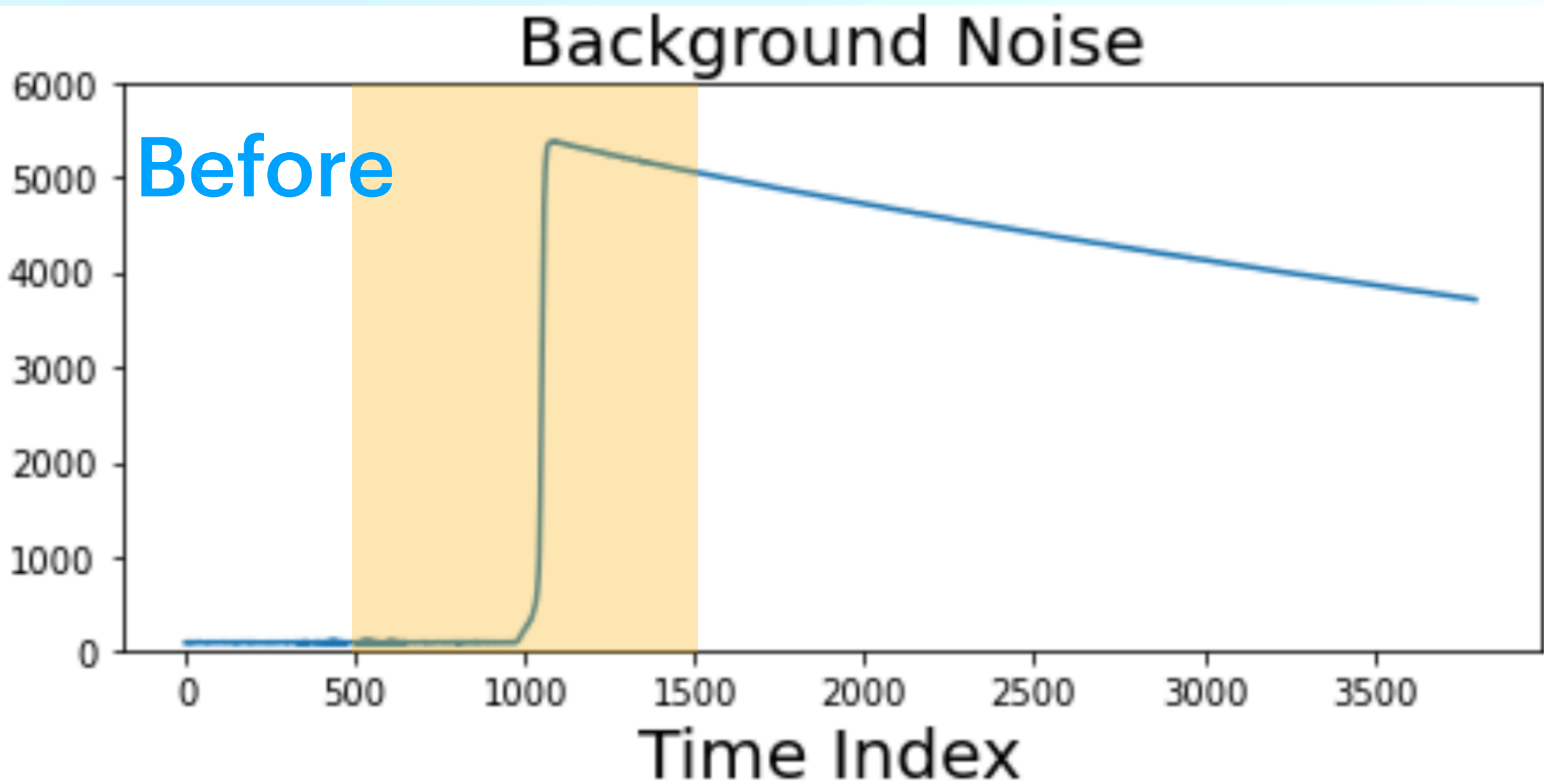
Alarm: Waveforms have different height due to energy difference.



Concept

Data Preprocessing: Transform our data in certain way so that unwanted features does not affect model training

Windowing: zoom in to the rising edge of the waveform that contains most of the physics





Concept

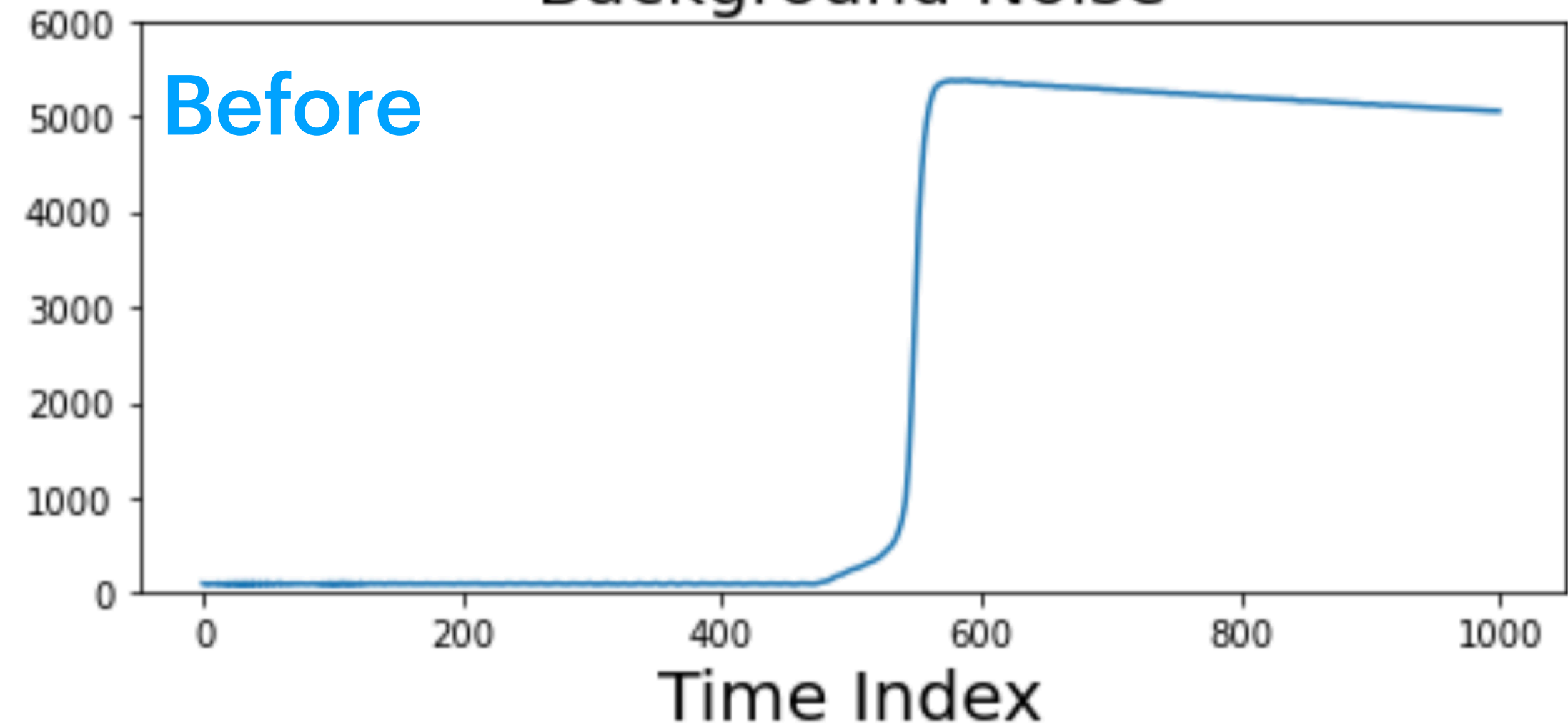
Data Preprocessing: Transform our data in certain way so that unwanted features does not affect model training

Normalizing: make sure all waveforms are of the same height (remove energy dependency)

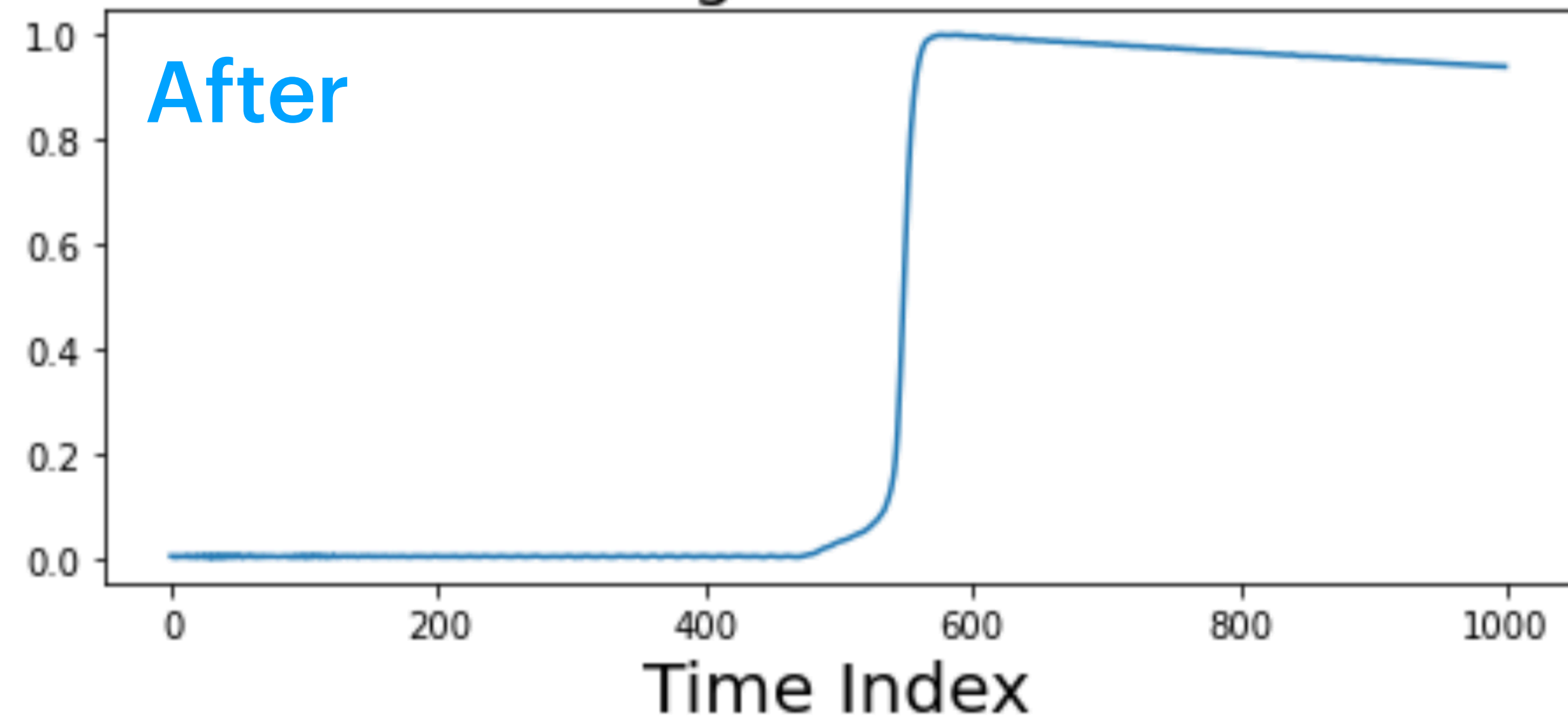
Code

```
wf_norm = (wf - wf.min()) / (wf.max() - wf.min()) # wf is a 1-d numpy array
```

Background Noise



Background Noise





Concept

Data Preprocessing: Transform our data in certain way so that unwanted features does not affect model training

Concept

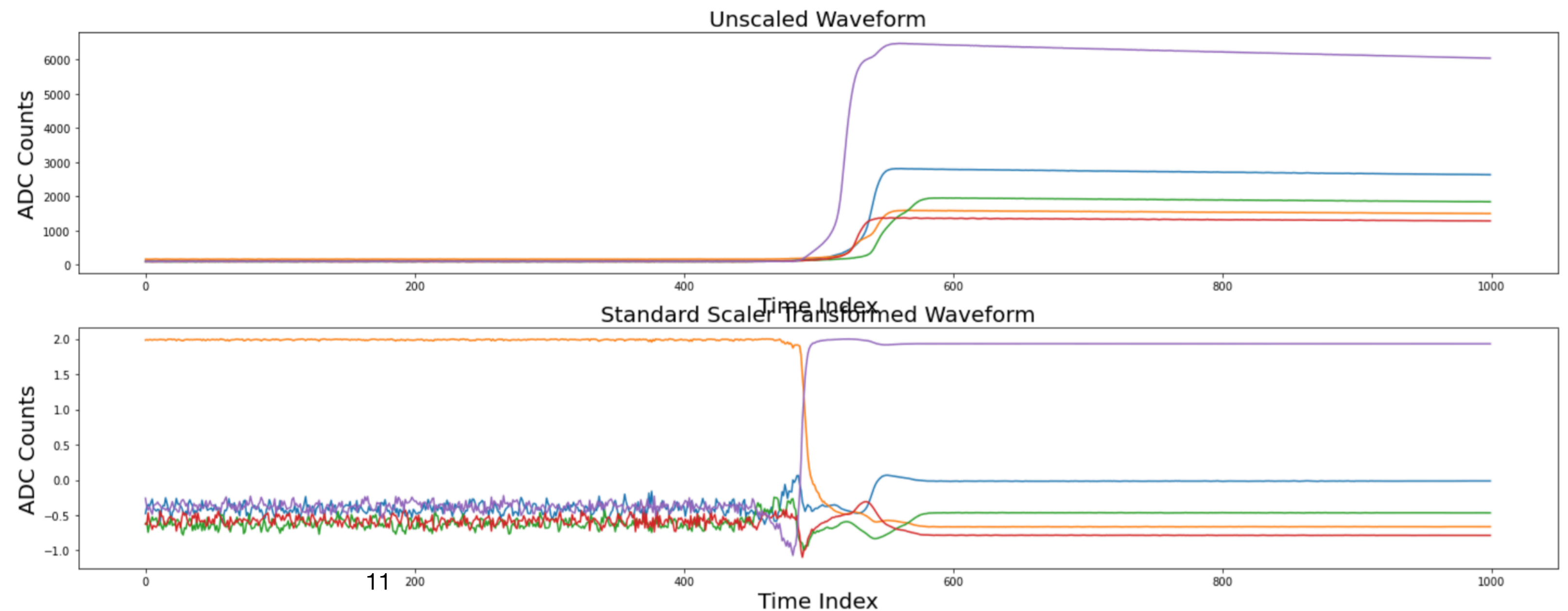
Standard Scaler: transform each dimension of data so that its mean = 0 and standard dev. = 1

Code

```

# sklearn package contains Standard Scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Has to be applied over multiple wfs
wf_norm = scaler.fit_transform(wf_arr)
  
```





Code

Putting everything into a PyTorch Dataset class

```

class NeutrinoDataset(torch.utils.data.Dataset):
    def __init__(self, plot=True):
        """
        REQUIRED function
        This function initializes the dataset object
        """
        self.waveform: (65000, 1000)
        self.energy_label: (65000,)
        self.psd_label_low_avse: (65000,)

    def __len__(self):
        """
        REQUIRED function
        This function returns the size of overall dataset
        """
        return 65000

    def __getitem__(self, idx):
        """
        REQUIRED function
        This function extract a single waveform from the dataset at the given index idx
        """
        return self.waveform[idx], self.psd_label_low_avse[idx], self.energy_label[idx]

    def plot_data(self):
        """
        OPTIONAL functions
        """

```

- **Only called once** when initializing the object
- Read information from the .hdf5 file into array(s)
 - apply dataset-level pre-processing like StandardScaler

- Return length of the dataset
- Called whenever we run len(dataset_object)

- Return **one data point** at input index **idx**
- Datapoint-level preprocessing can go here (i.e. Normalizing)
- Will be **called very frequently** during network training and validation
 - Don't put slow operations here!

- Define as many helper functions as you like



Code

Used the pre-defined class on last slides to construct training and testing data loader

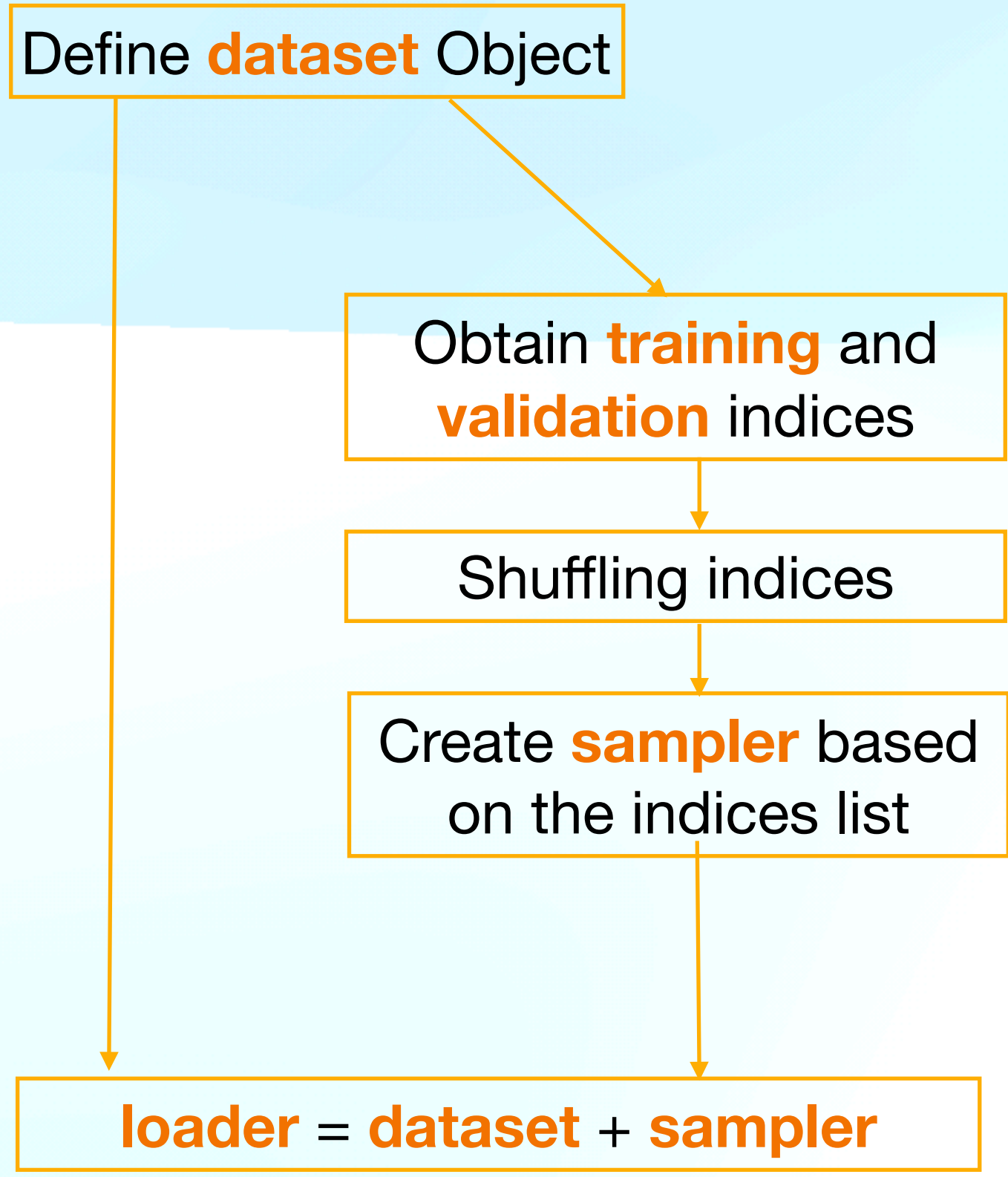
```

dataset = NeutrinoDataset(plot=False)
'''
Get the indices of train dataset and test dataset correspondingly,
indices [0:train_test_split] is the training dataset,
indices [train_test_split, len(dataset)] is the test dataset.
'''
train_test_split = int(0.7*len(dataset))
train_indices, val_indices = list(range(train_test_split)), list(range(train_test_split, len(dataset)))

#Shuffle the two indices list
np.random.shuffle(train_indices)
np.random.shuffle(val_indices)

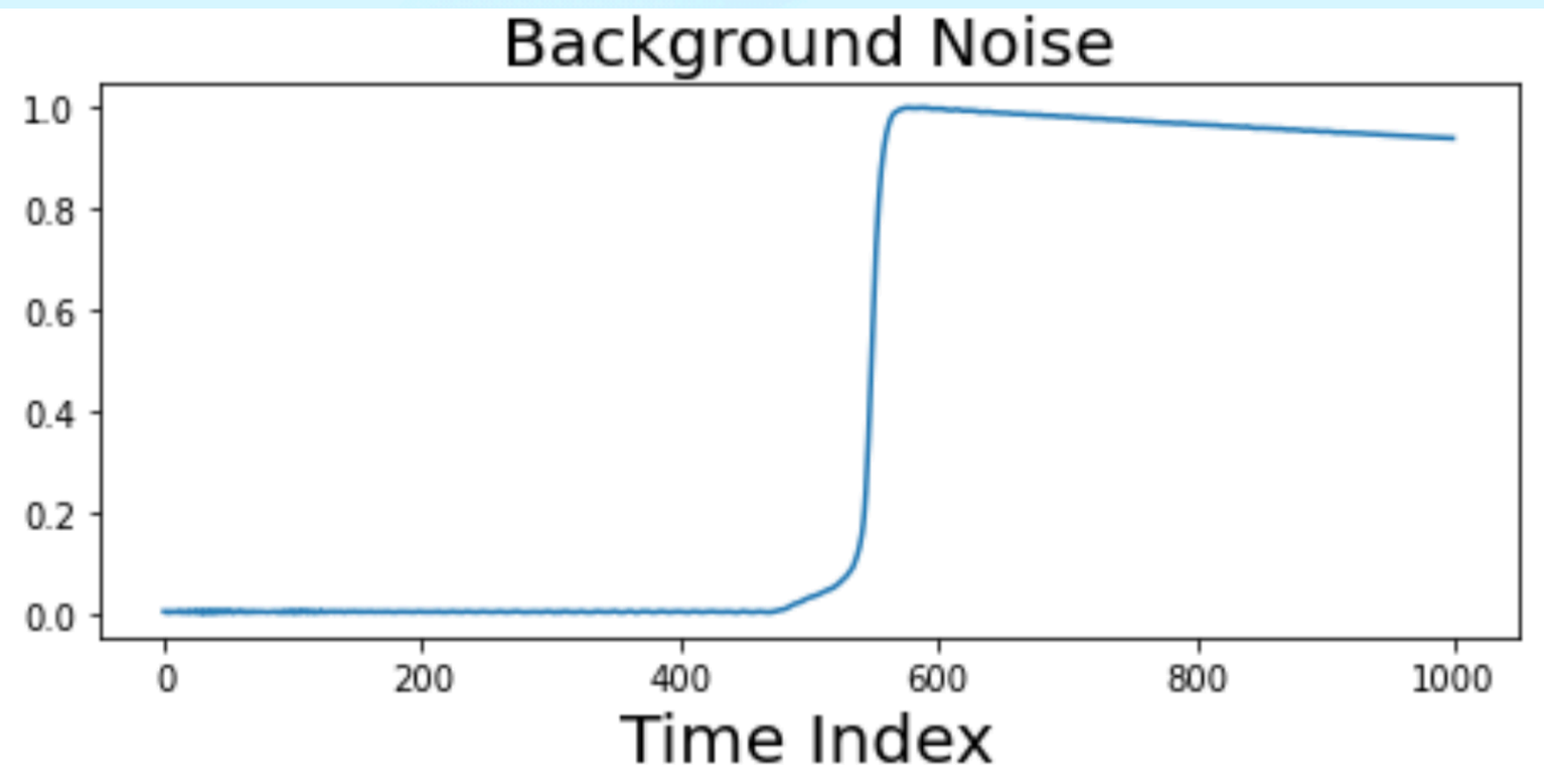
# Define two subset random sampler to sample events according to the training indices
train_sampler = SubsetRandomSampler(train_indices)
valid_sampler = SubsetRandomSampler(val_indices)
'''
Finally, define the loader by passing in the dataset, batch size and corresponding sampler
Note that the number of data in each sub-dataset might not be divisible by the batch size,
so drop_last=True drops the last batch with all the residual events.
'''
train_loader = data_utils.DataLoader(dataset, batch_size=BATCH_SIZE, sampler=train_sampler, drop_last=True)
test_loader = data_utils.DataLoader(dataset, batch_size=BATCH_SIZE, sampler=valid_sampler, drop_last=True)
  
```

Batch: group n waveforms together into a 2D array (1000,) → (BATCH_SIZE, 1000)

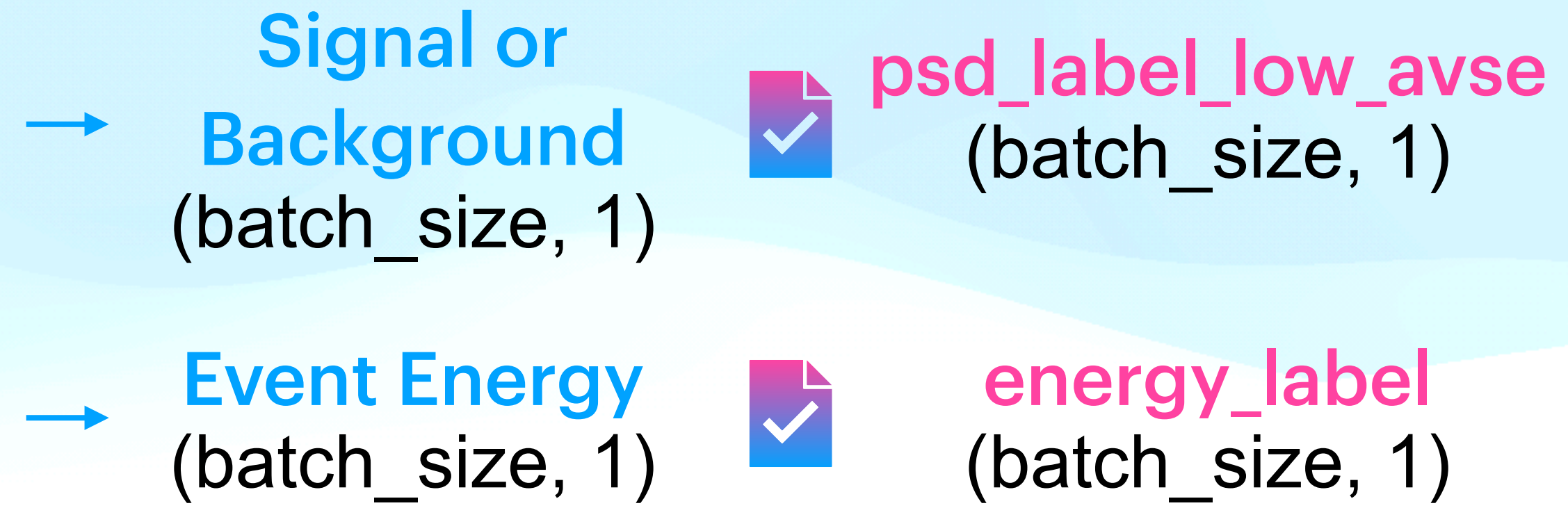
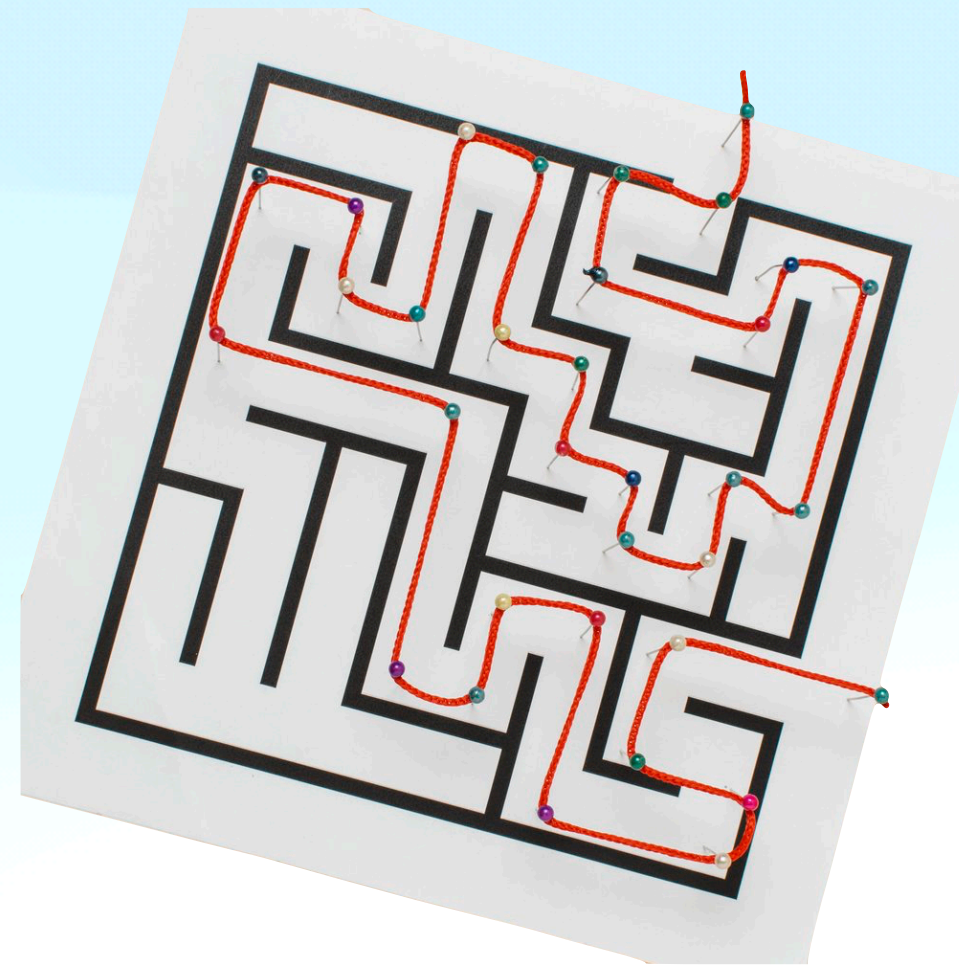




Time Series Data
(BATCH_SIZE, 1000)

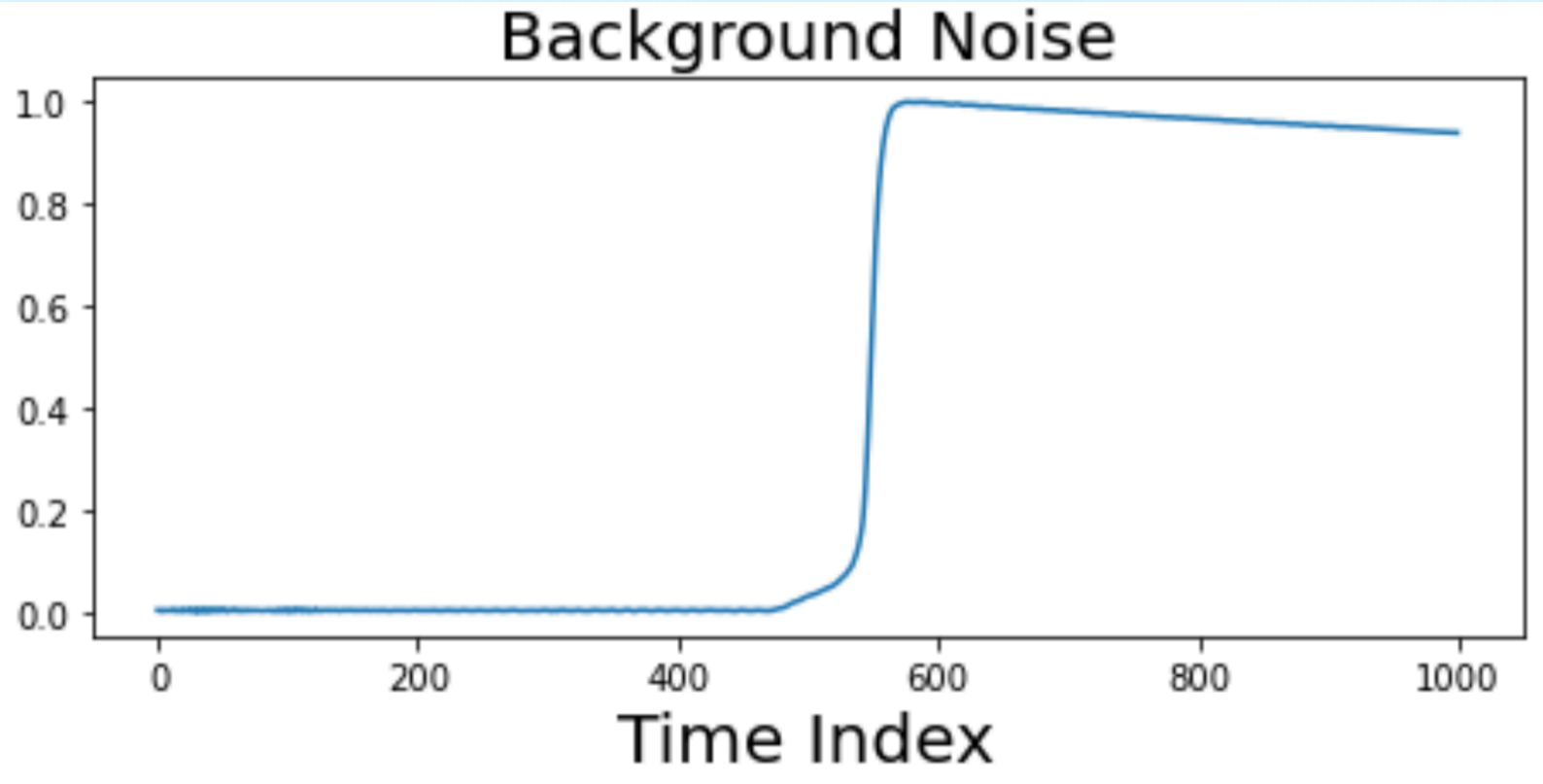


Model
Map input to output

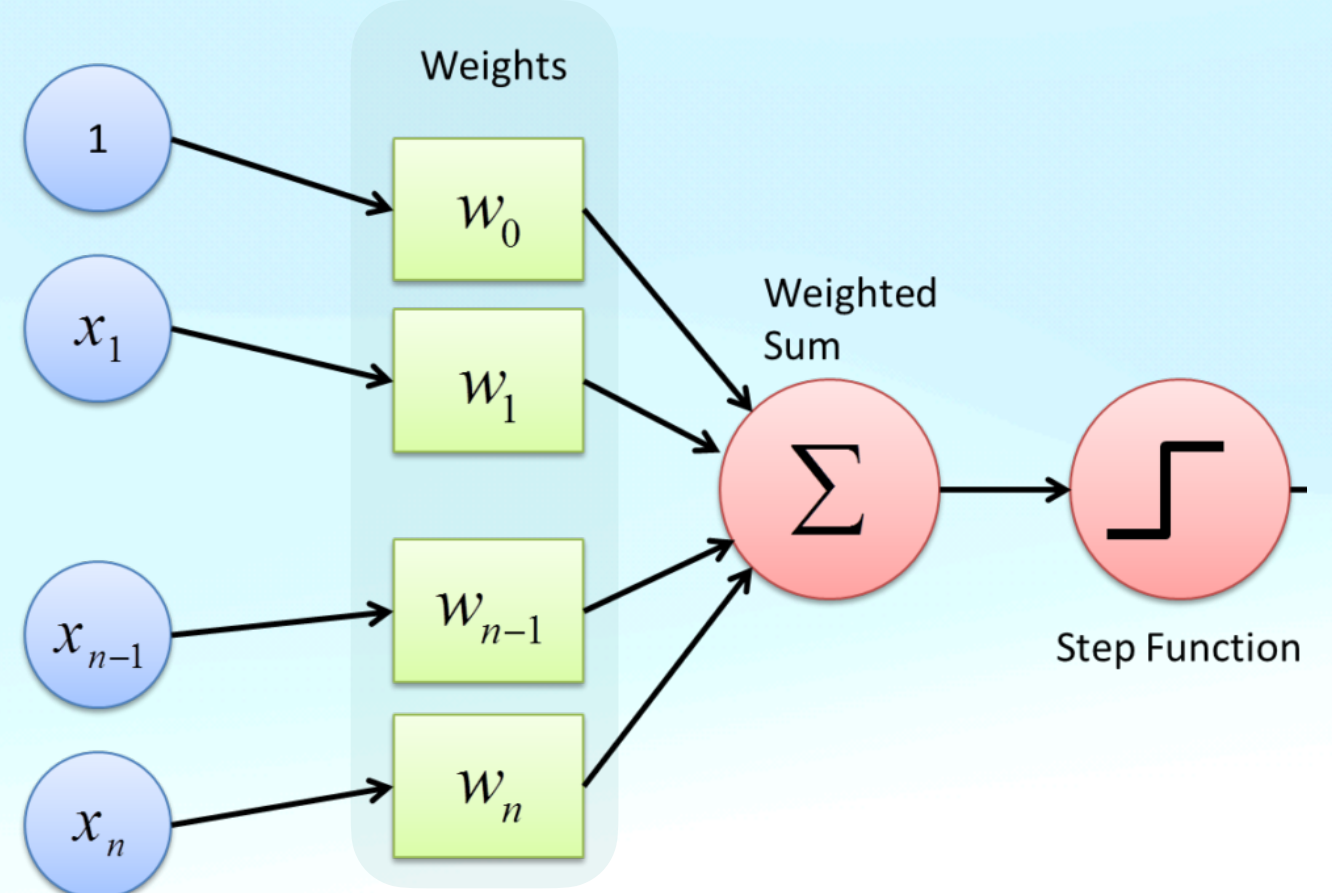




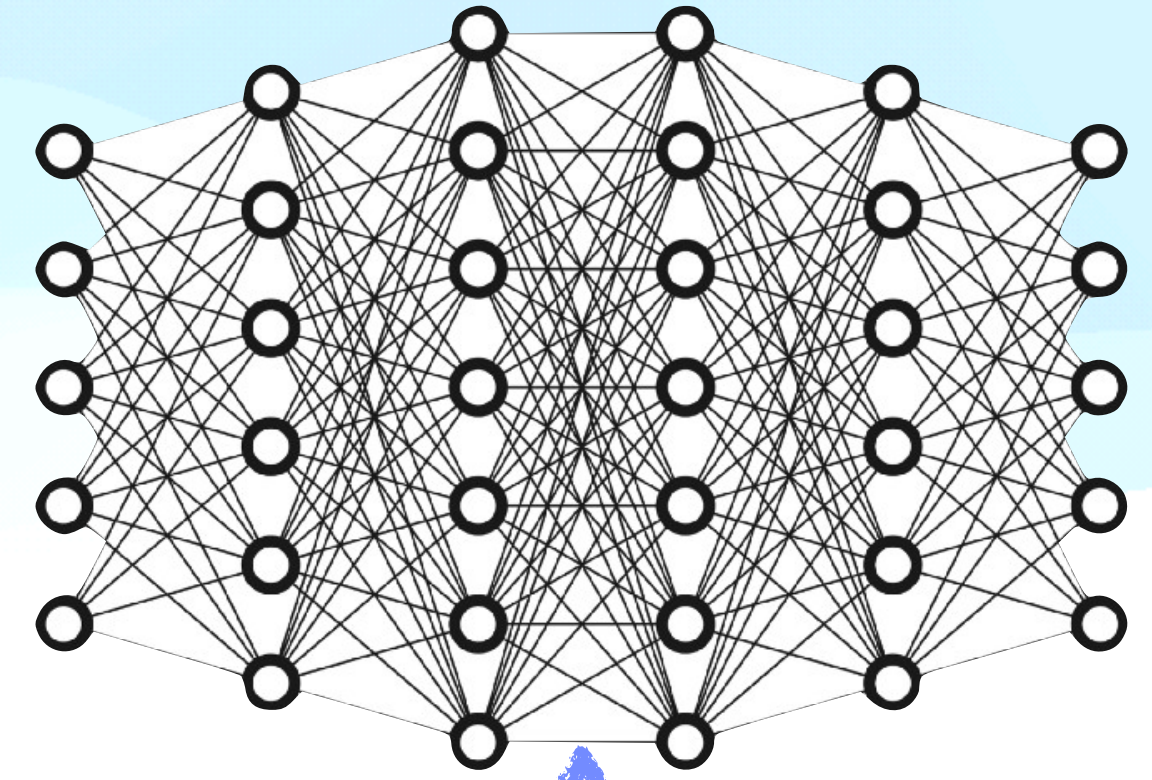
Time Series Data
(batch_size, 1000)



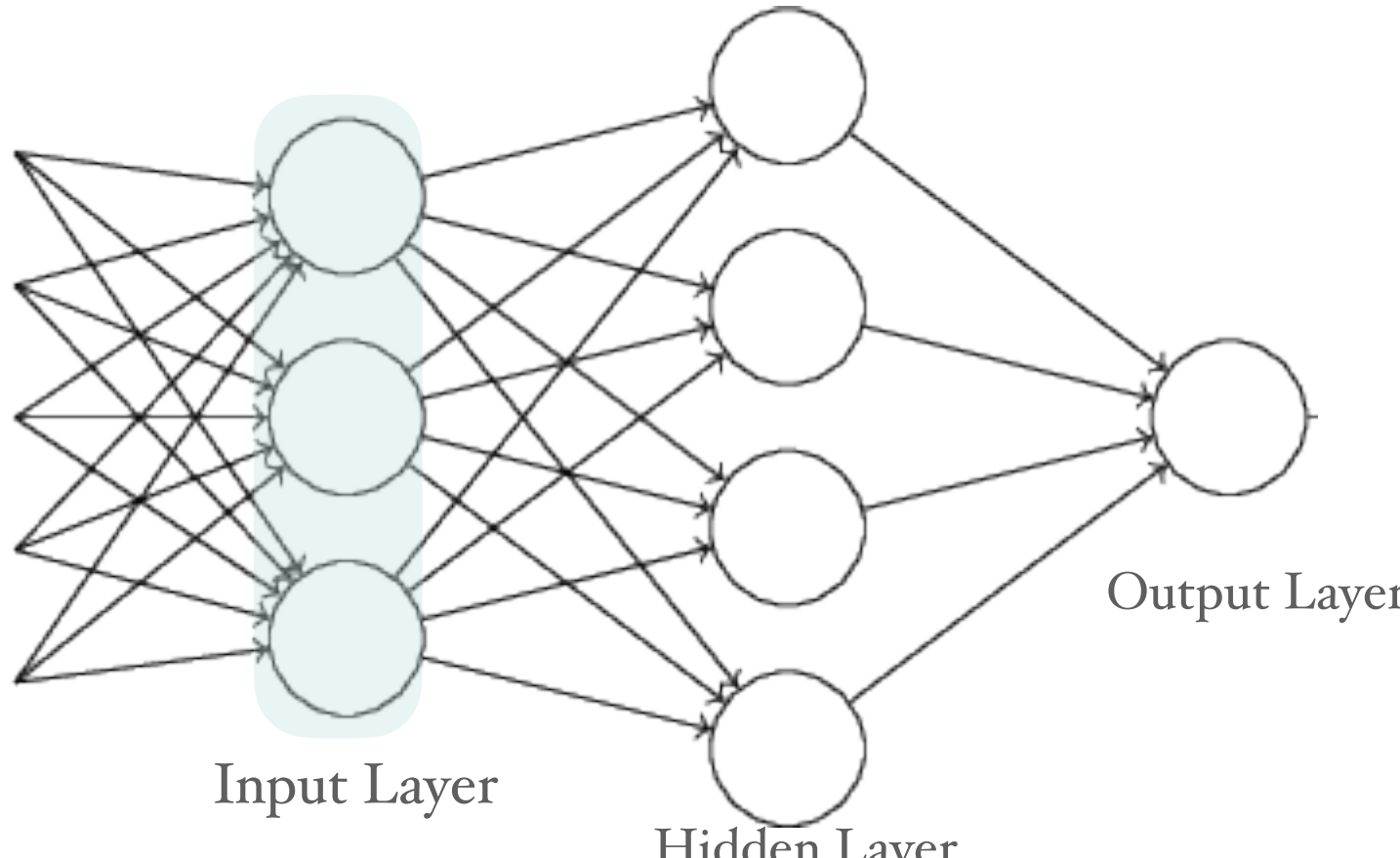
Perceptron Learning
(Linear Classifier)



Deep Neural Networks
Adding more layers!



Layers
Many parallel neurons



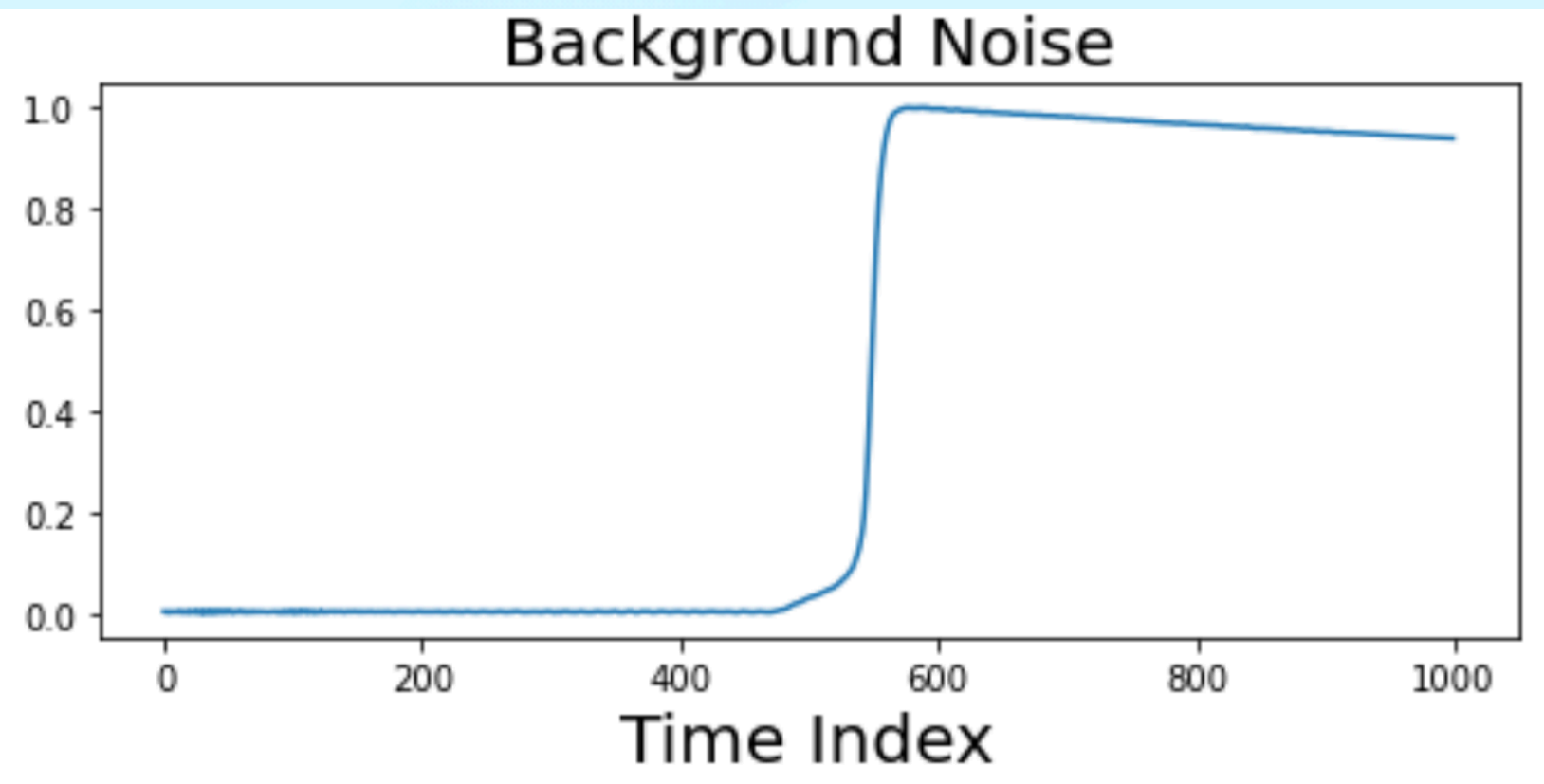
Concept

Elements of NN Layers:

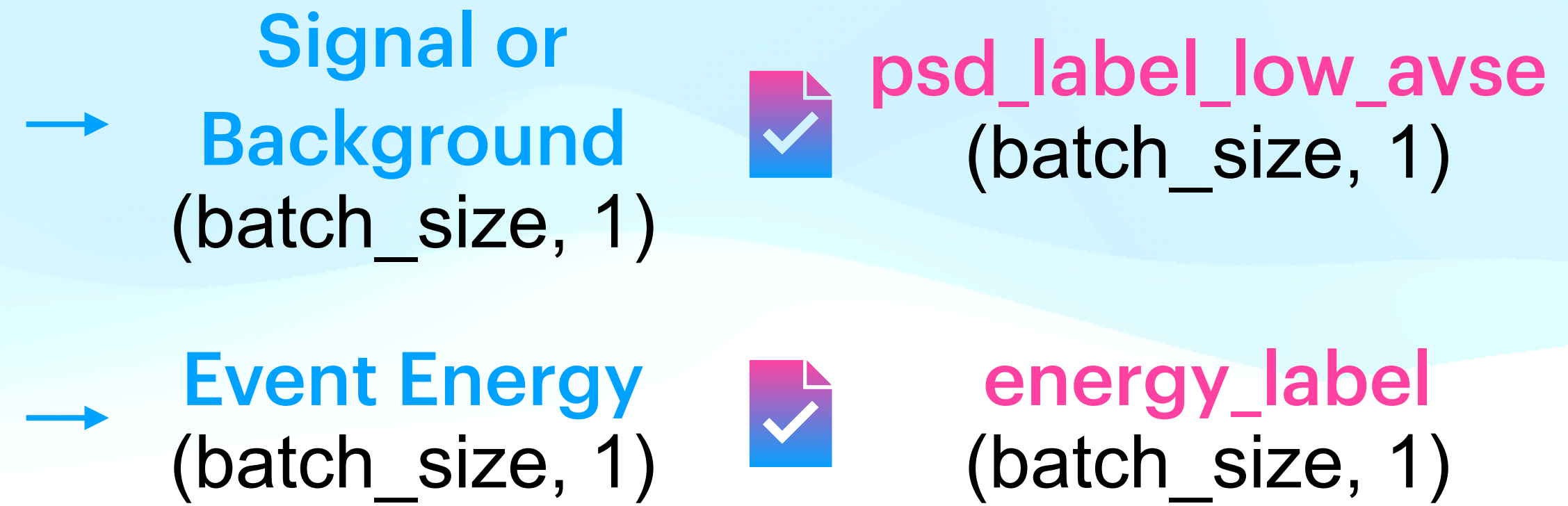
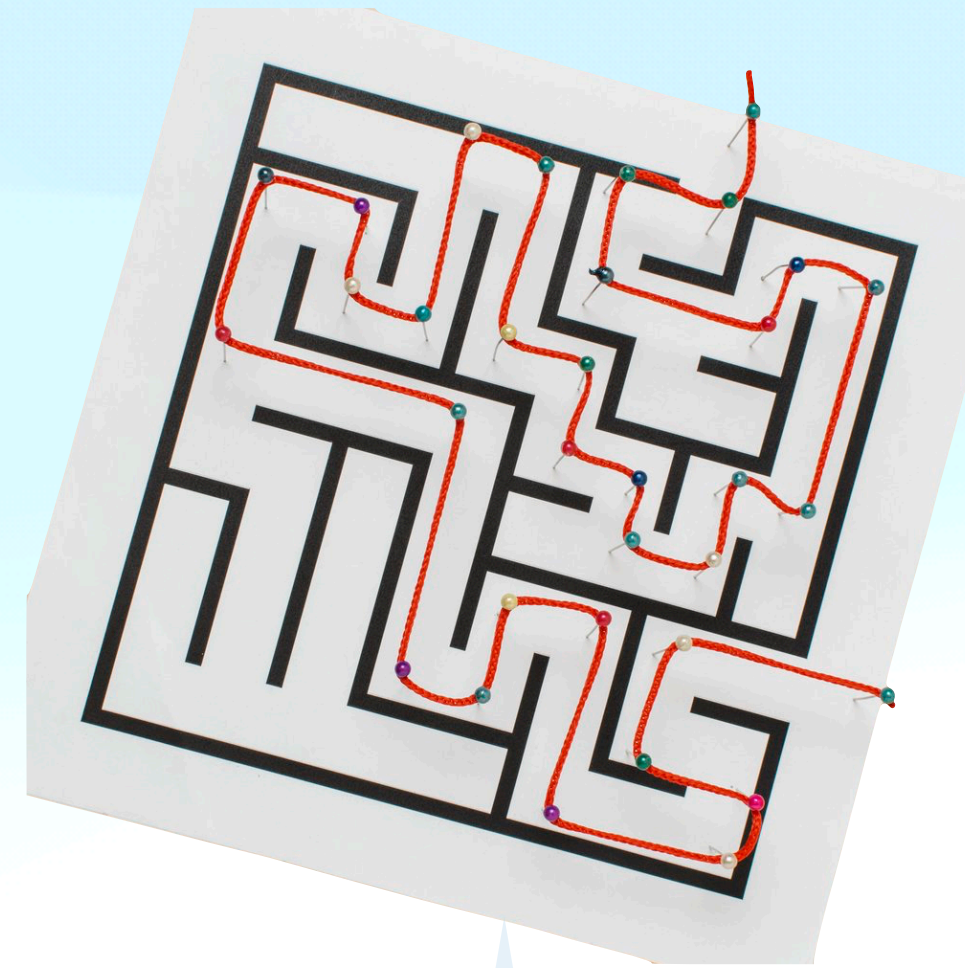
- Input dimension
- Output dimension
- Activation function



Time Series Data (BATCH_SIZE, 1000)



Model Map input to output



🧠 Concept

Feature Extractor Network

Take raw data as the input and output a low-dimensional vector

$$\begin{aligned}
 & \text{NNLayer}(1000, 512) \rightarrow \sigma \\
 & \rightarrow \text{NNLayer}(512, 256) \rightarrow \sigma \\
 & \rightarrow \text{NNLayer}(256, 32) \rightarrow \sigma \\
 & \rightarrow \text{NNLayer}(32, 1) \rightarrow \sigma
 \end{aligned}$$



- **Only called once** when initializing the neural network object
- Define network structures

- Linear Layer:**
- Defined with input & output dim

- Forward Pass:**
- Feed data through the neural network to obtain desired outputs
 - (BS, 1000,) → (BS, 1)
 - **Called multiple times** during training and validation

Code

```

class FCNet(nn.Module):
    def __init__(self):
        super(FCNet, self).__init__()

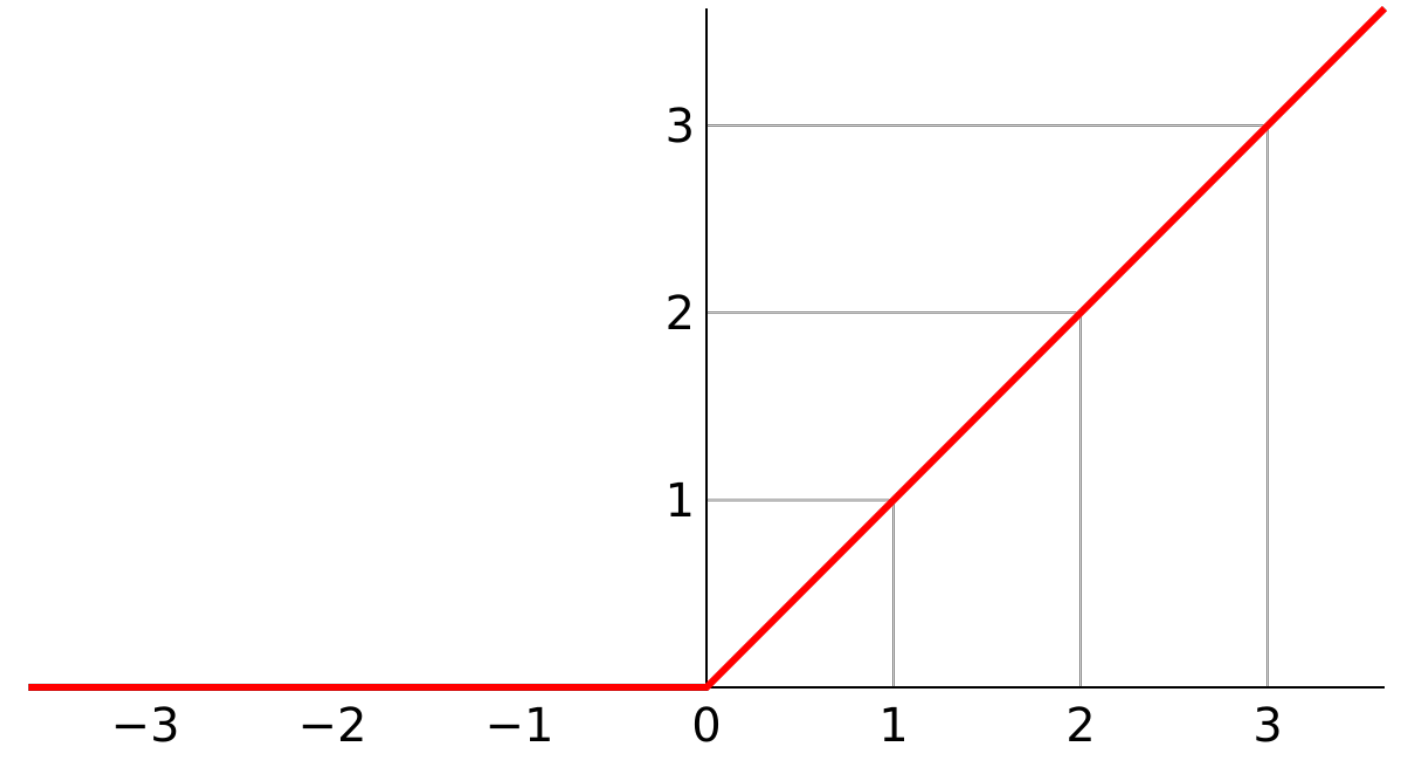
        self.feature_extractor = nn.Sequential(
            torch.nn.Linear(1000, 512),
            torch.nn.ReLU(),
            torch.nn.Linear(512, 256),
            torch.nn.ReLU(),
            torch.nn.Linear(256, 32),
            torch.nn.ReLU(),
        )

        self.task_layer = torch.nn.Linear(32, 1)

    def forward(self, x):
        """
        The forward operation of each training step
        """
        x = self.feature_extractor(x)
        x = self.task_layer(x)

        return x
  
```

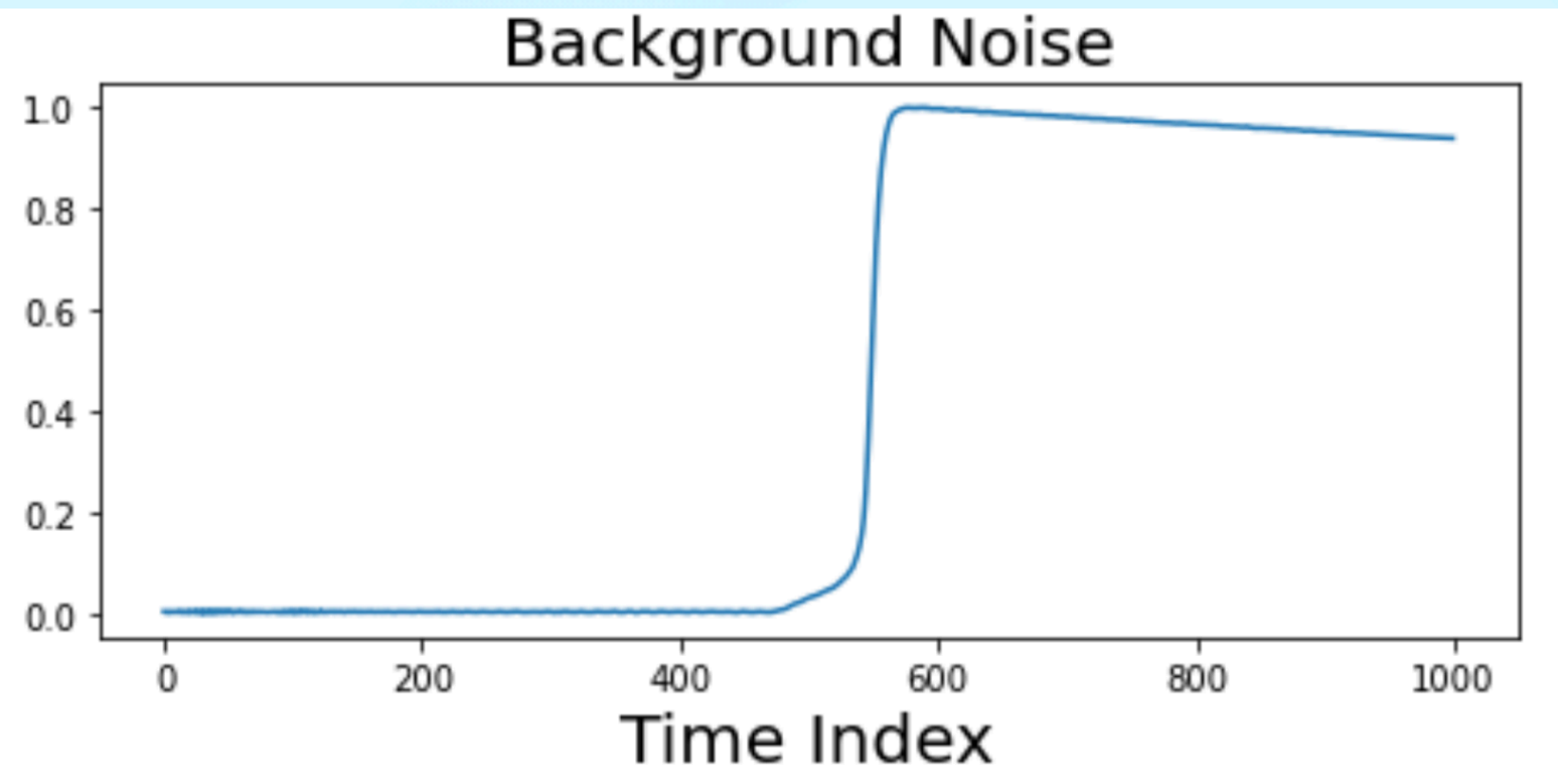
- Activation Function:**
- Adding non-linearity to NN
 - ReLU is most commonly used



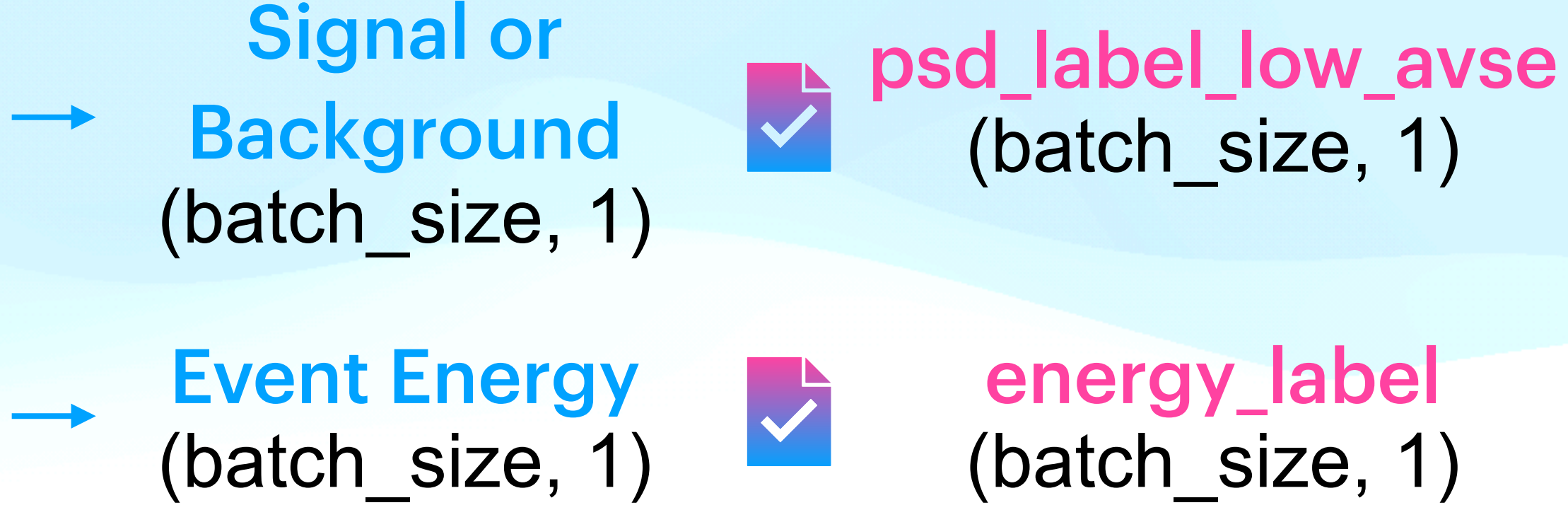
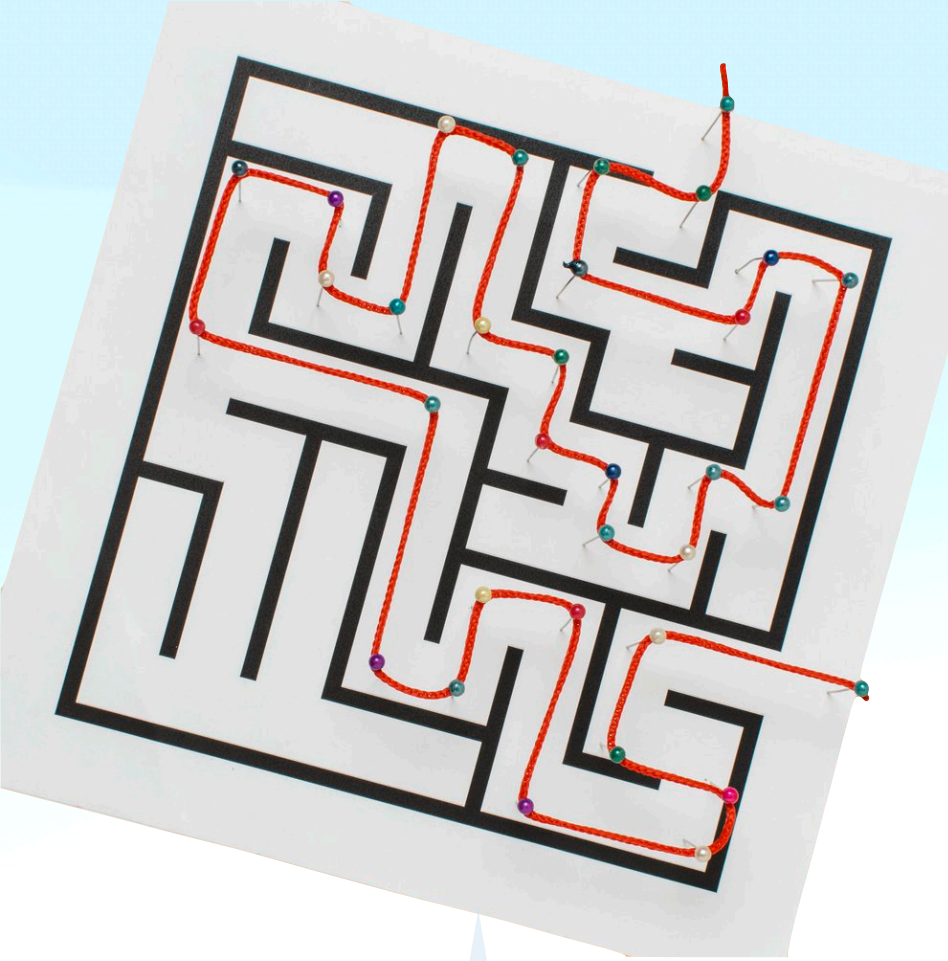
Backward Pass will be discussed in Sec. 4!



Time Series Data (BATCH_SIZE, 1000)



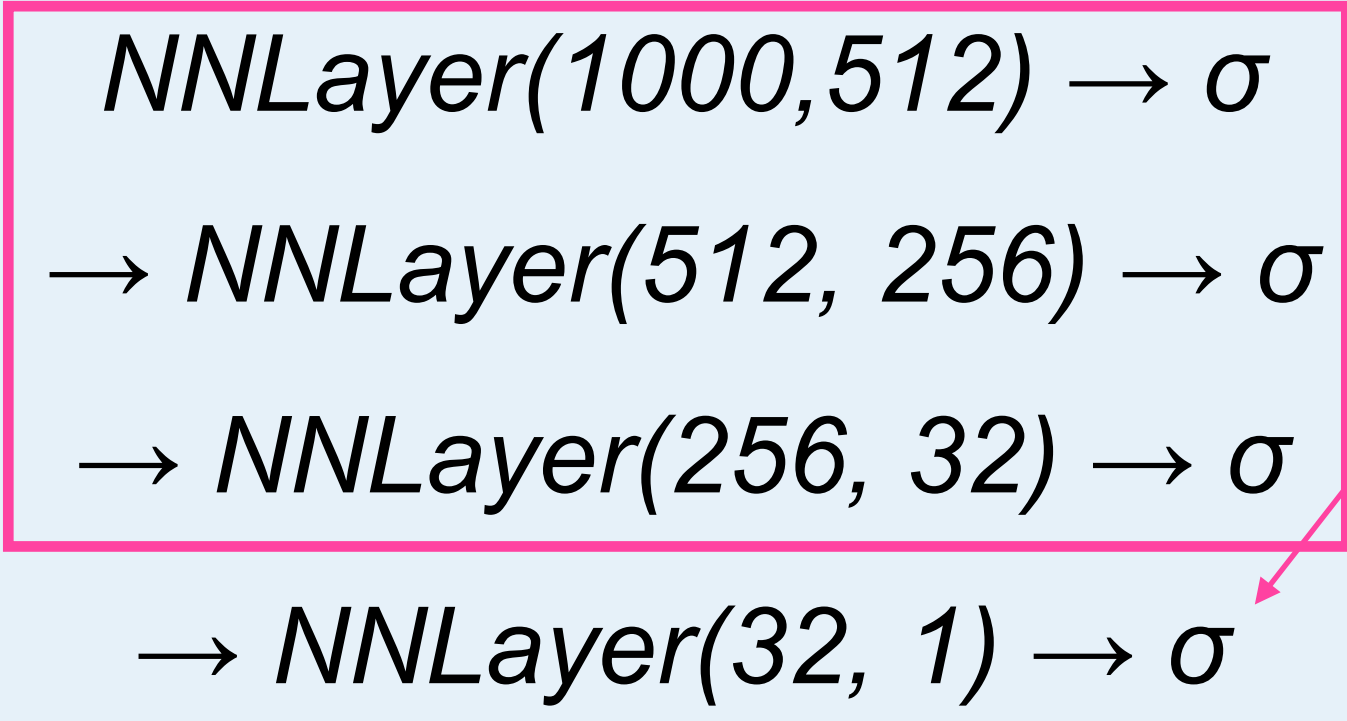
Model Map input to output



Concept

Feature Extractor Network

Take raw data as the input and output a low-dimensional vector



Concept

Task Module: Task Layer + Loss Function

- **Task Layer:** Take low-dimensional vector as input and produce a value/vector as **output**
- **Loss Function:** produce a quantitative measure evaluating how well your algorithm models your dataset
- Minimizing the loss function means that the **model output** reproduces the **label** better

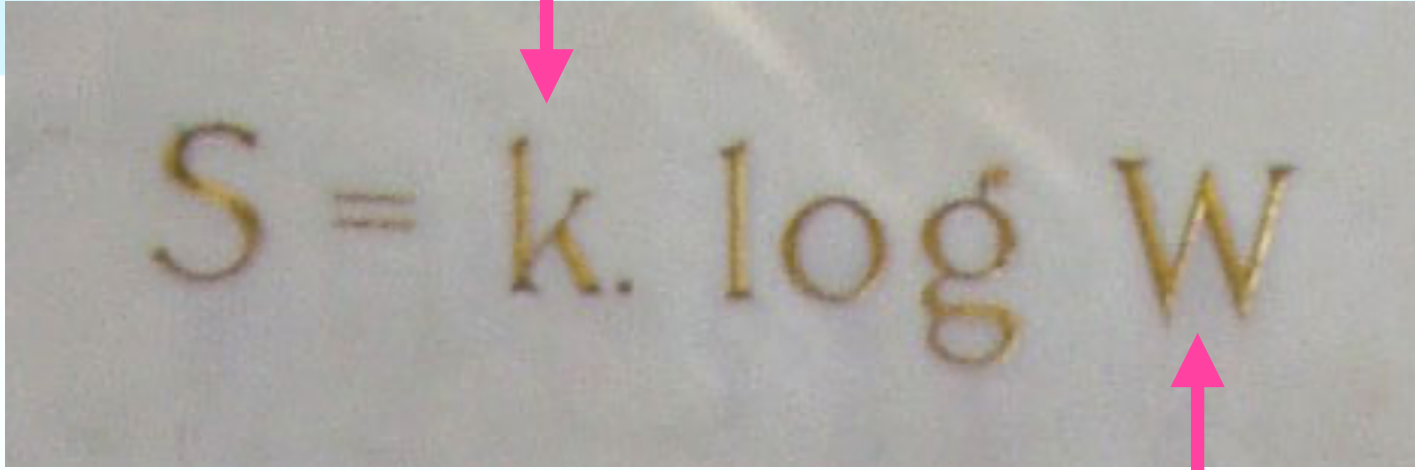


BOLTZMANN ENTROPY


INFORMATION ENTROPY

Boltzmann's Constant

Probability of state *i*

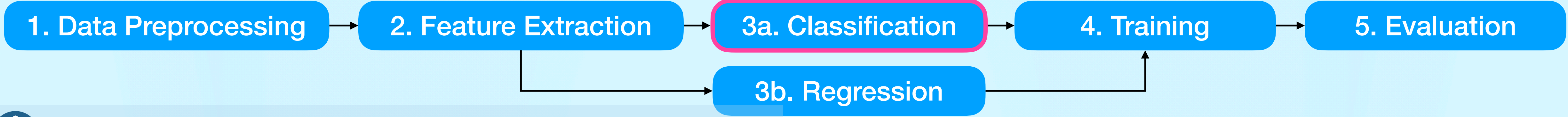


The number of real microstates corresponding to the gas's macrostate

 **Theory**

$$H = \sum_i p_i \log \frac{1}{p_i} = - \sum_i p_i \log p_i$$

Optimal coding length
 The number of states in state *i* assuming all states have equal probability



Theory

Binary entropy: $H(x) = p \log \frac{1}{p} - (1 - p) \log \frac{1}{1 - p}$

Suppose we are tossing a **fair coin**:

- That is, the probability of head (H) and tail (T) are equal

- $H(X) = \frac{1}{2} \log \frac{1}{1/2} + \frac{1}{2} \log \frac{1}{1/2} = \log 2 \approx 0.6931$

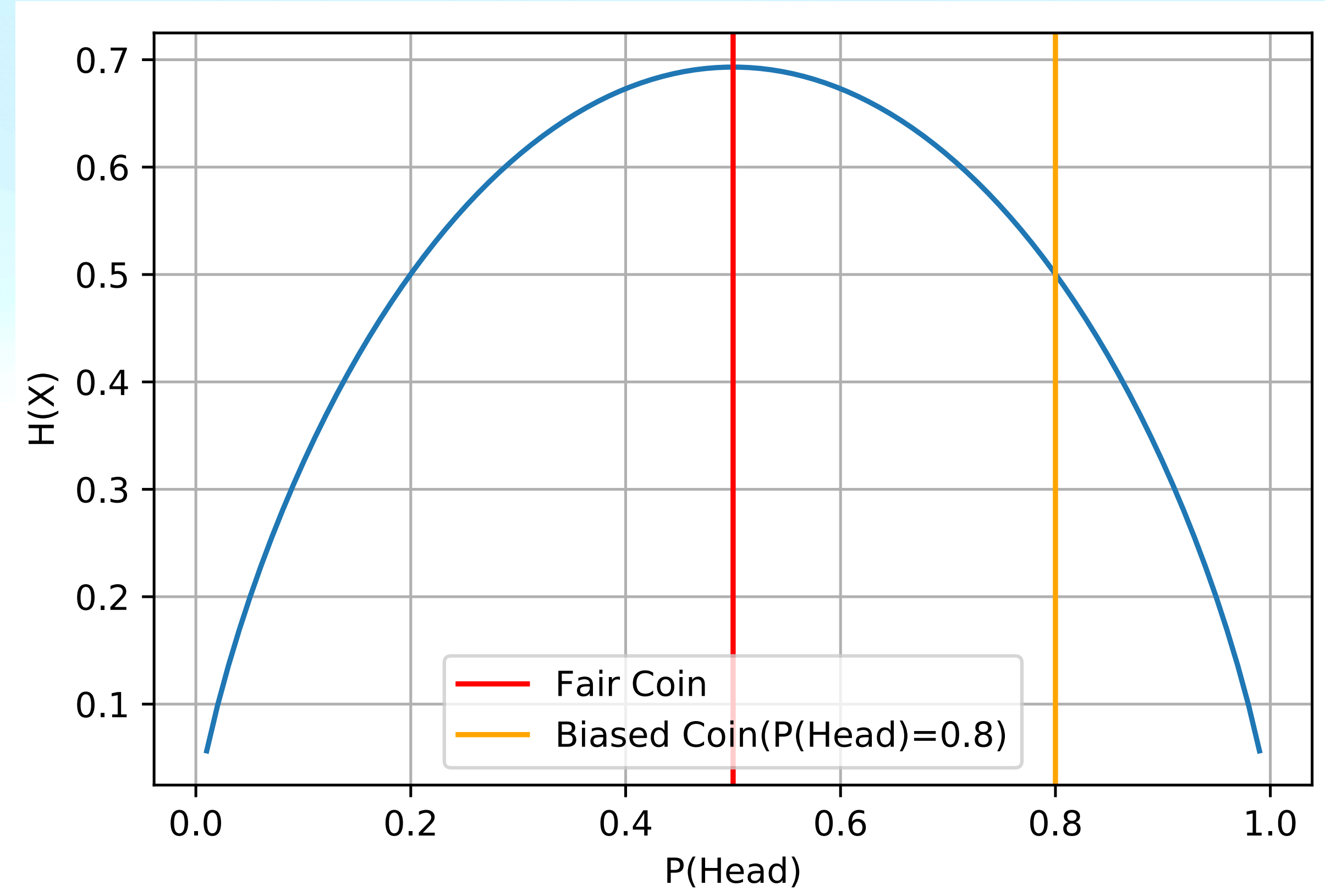
Suppose we have an **unfair coin** where $P(\text{Head}) = 0.8$:

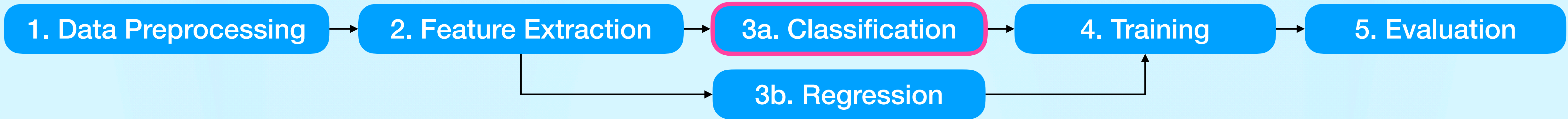
- $H(X) = 0.8 \log \frac{1}{0.8} + 0.2 \log \frac{1}{0.2} \approx 0.5$

Concept

Information entropy is the **average amount of surprise** we measure

- For a fair coin, it is hard to predict the outcome of next toss, so the average surprisal is **high** and thus **high entropy**
- For a unfair coin, the next outcome is likely to be a head, so the average surprisal is **low** and thus **low entropy**





Theory

Cross Entropy $H(p, q)$ is calculated over two distributions $p(X)$ and $q(X)$:

$$H(p, q) = \sum p(X) \log \frac{1}{q(X)}$$

- Suppose we have a classification task to separate neutrino signals from backgrounds
- The **ground truth distribution** $p(X)$ follows the distribution of labels
- On the other hand, we build up a neural network, which analyze every single waveform to produce an **output distribution** $q(X)$

Concept

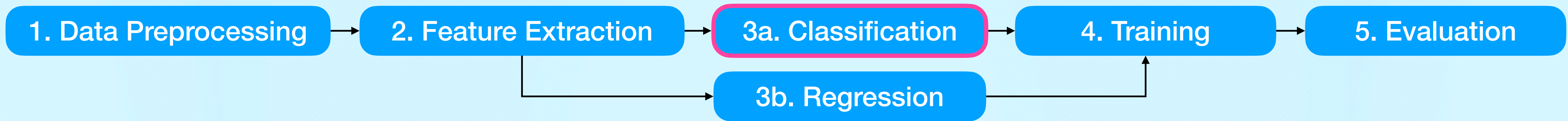
Binary Cross Entropy: label is either 1 or 0

$$H = - [p \cdot \log(q) + (1 - p) \cdot \log(1 - q)]$$

Concept

Cross Entropy: label contains n classes

$$H = - \sum \vec{p} \cdot \log(\vec{q}) \quad \vec{p} = \{1, 0, 0 \dots 0\}$$



Information Entropy measures the average amount of surprise



Cross Entropy measure the average amount of surprise from **network output distribution** $q(X)$, given that the **ground truth distribution** follows $p(X)$

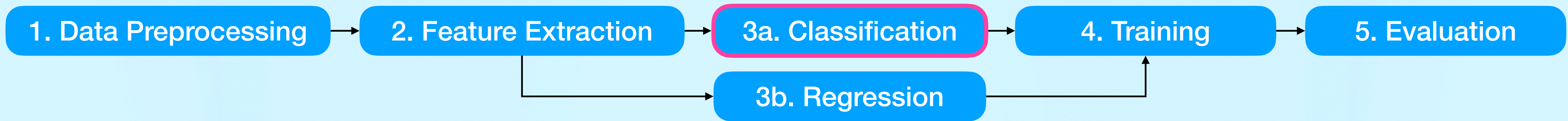


Concept

Cross Entropy Loss: Minimize Cross Entropy is identical to minimizing the surprise of NN output with respect to the ground truth label

- When $p(\text{Waveform}) = \text{Neutrino}$, $q(\text{Waveform}) \rightarrow \text{Neutrino}$
- When $p(\text{Waveform}) = \text{Background}$, $q(\text{Waveform}) \rightarrow \text{Background}$

Lower Cross Entropy loss means better separation between neutrino signal and background



NNLayer

σ

L

Concept

Binary Classification 1

Task Layer: `NNLayer(32, 1)` → One float point No. between $[-inf, inf]$

- After training, it can be considered as a “**classification score**”:
- Higher score means the answer is more likely “yes”
- Lower score means the answer is more likely “no”
- A **threshold** is need to distinguish “yes” from “no”

σ : `torch.sigmoid(x)`

- Sigmoid function $\sigma(x) = 1/(1 + e^x)$ maps input from $[-inf, inf]$ to $[0,1]$

Loss Function: `torch.nn.BCELoss()`

- Input: has to be a number between $[0,1]$
- Target: has to be either 0 or 1, cannot be other number

Concept

Binary Classification 2

Task Layer: `NNLayer(32, 1)` → One float point No. between $[-inf, inf]$

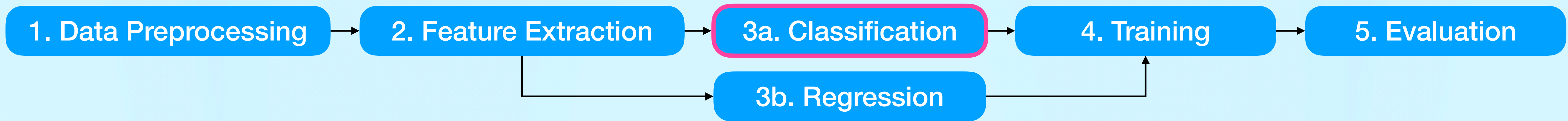
After training, it can be considered as a “**classification score**”:

- Higher score means the answer is more likely “yes”
- Lower score means the answer is more likely “no”
- A **threshold** is need to distinguish “yes” from “no”

σ : None

Loss Function: `torch.nn.BCEWithLogitsLoss()`

- Input: any range between $[-inf, inf]$
- Target: has to be either 0 or 1, cannot be other number



NNLayer

σ

L

Concept

Binary Classification 3

Task Layer: `NNLayer(32, 2)` → two float point No. between $[-inf, inf]$

- After training, it can be considered as a “**classification decision**”:
 - Assign meaning to the two numbers (1st - yes, 2nd - no)
 - The larger number of the two represents the one we select
 - No **threshold** needed

σ : None

Loss Function: `torch.nn.CrossEntropyLoss()`

- Input: array of size 2 with two number between $[-inf, inf]$
- Target: the **array indices** of correct choice

Concept

Multiclass Classification

Task Layer: `NNLayer(32, n)` → n float point No. between $[-inf, inf]$

- After training, it can be considered as a “**classification decision**”
 - Assign meaning to the each numbers (1st - C1, 2nd - C2, 3rd - C3,)
 - The largest number represents the one we select
 - No **threshold** needed

σ : None

Loss Function: `torch.nn.CrossEntropyLoss()`

- Input: array of size n with two number between $[-inf, inf]$
- Target: the **array indices** of correct choice



NNLayer

σ

L

Concept

Regression 1

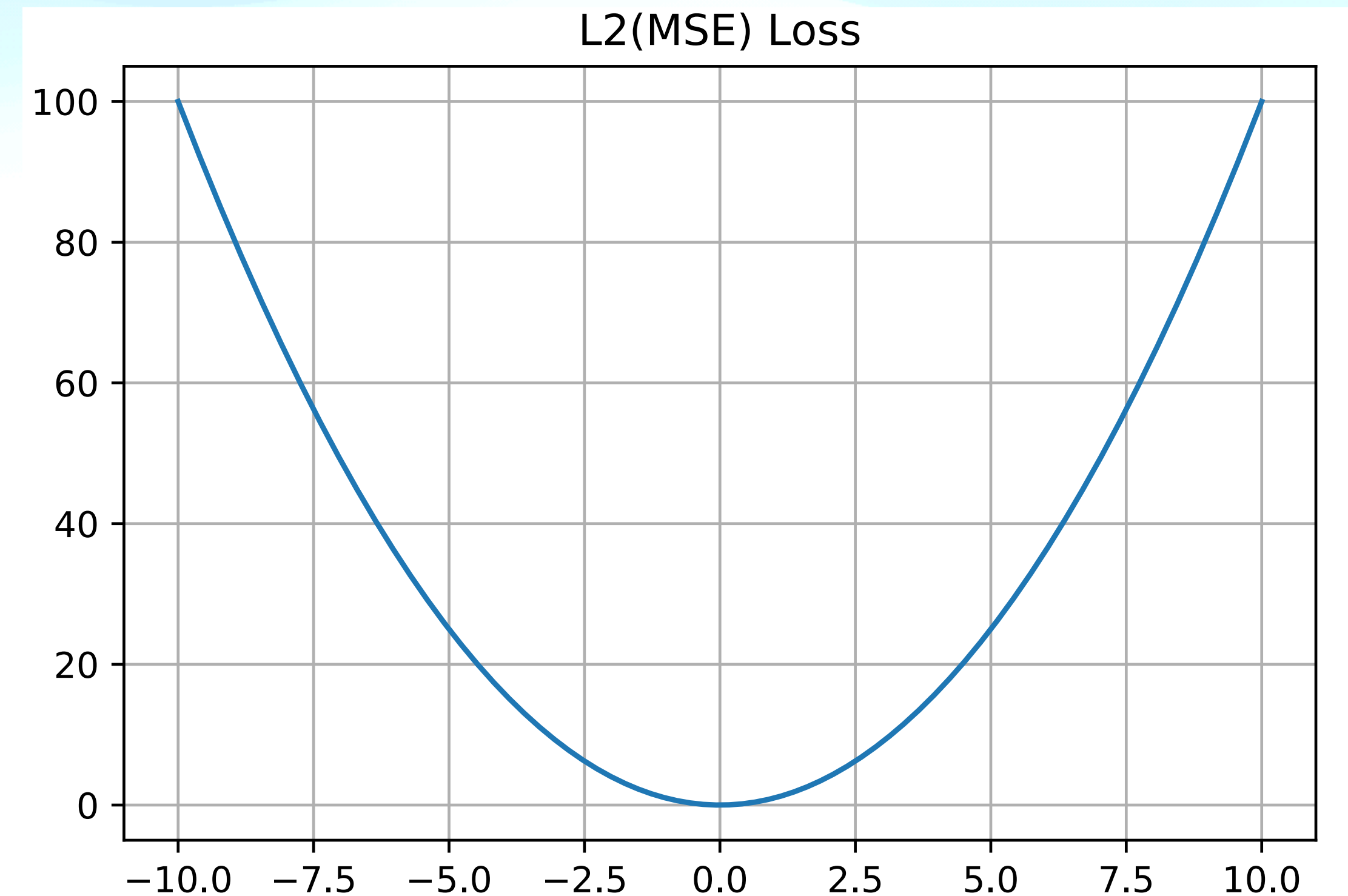
Task Layer: `NNLayer(32, 1)` → One float point No. between $[-inf, inf]$

σ :

- None if you want to fit a physics quantity like energy
- `torch.sigmoid(x)` if you want to model a percentage like efficiency

Loss Function: `torch.nn.MSELoss()`

- $L = (TL_Output - Energy)^2$ for energy reconstruction
- Most commonly used, everywhere differentiable
- Gradient explosion with extreme values





NNLayer

σ

L

Concept

Regression 2

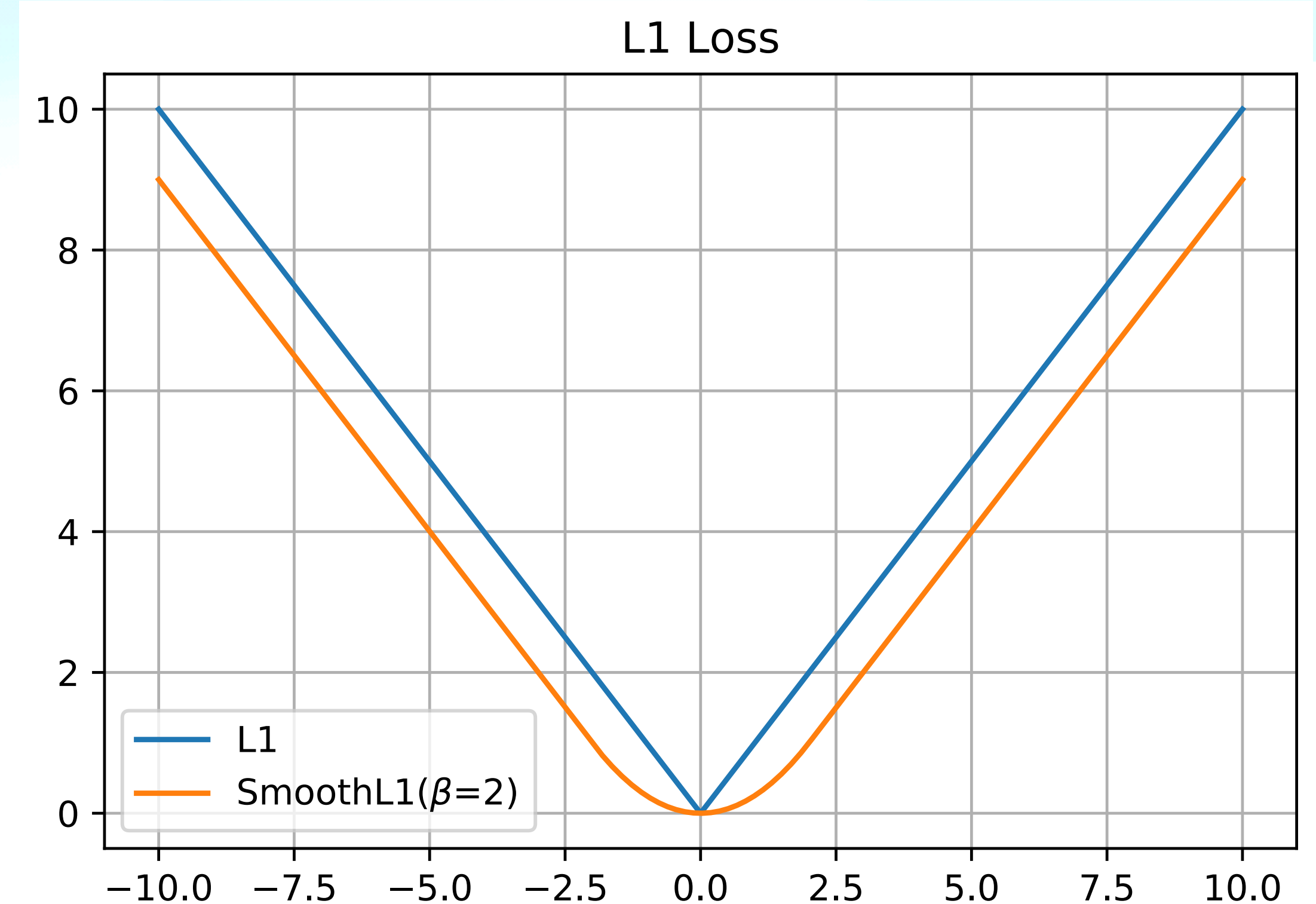
Task Layer: `NNLayer(32, 1)` → One float point No. between $[-inf, inf]$

σ :

- None if you want to model a physics quantity like energy
- `torch.sigmoid(x)` if you want to model a percentage like efficiency

Loss Function: `torch.nn.L1Loss()`

- $L = |TL_Output - Energy|$ for energy reconstruction
- Generally more Robust, but not Not everywhere differentiable
- Sometimes replaced with `torch.nn.SmoothL1Loss()`





Code

```

class FCNet(nn.Module):
    def __init__(self):
        super(FCNet, self).__init__()

        self.feature_extractor = #Already Discussed

        self.task_layer = torch.nn.Linear(32, 1)

    def forward(self, x):
        """
        The forward operation of each training step of the neural
        """
        x = self.feature_extractor(x)
        x = self.task_layer(x)
        #x = torch.sigmoid(x)

        return x

regressor = FCNet() # Change here to try different networks
criterion = torch.nn.L1Loss()

for i, (waveform, label, energy) in tqdm(enumerate(train_loader)):
    #Pull out 1 event from the dataset
    outputs = regressor(waveform)
    loss = criterion(outputs, energy)
  
```

Task Layer: output one float point number

σ : activation function, None for energy regression task since energy could be any values

Initialize model as an regressor object

Calculate regression loss between regressor output and true energy

Replace **Task Layer**, σ , and **Loss Function** using the previously-provided concept templates to achieve different tasks!

Initialize **Loss Function** as an object

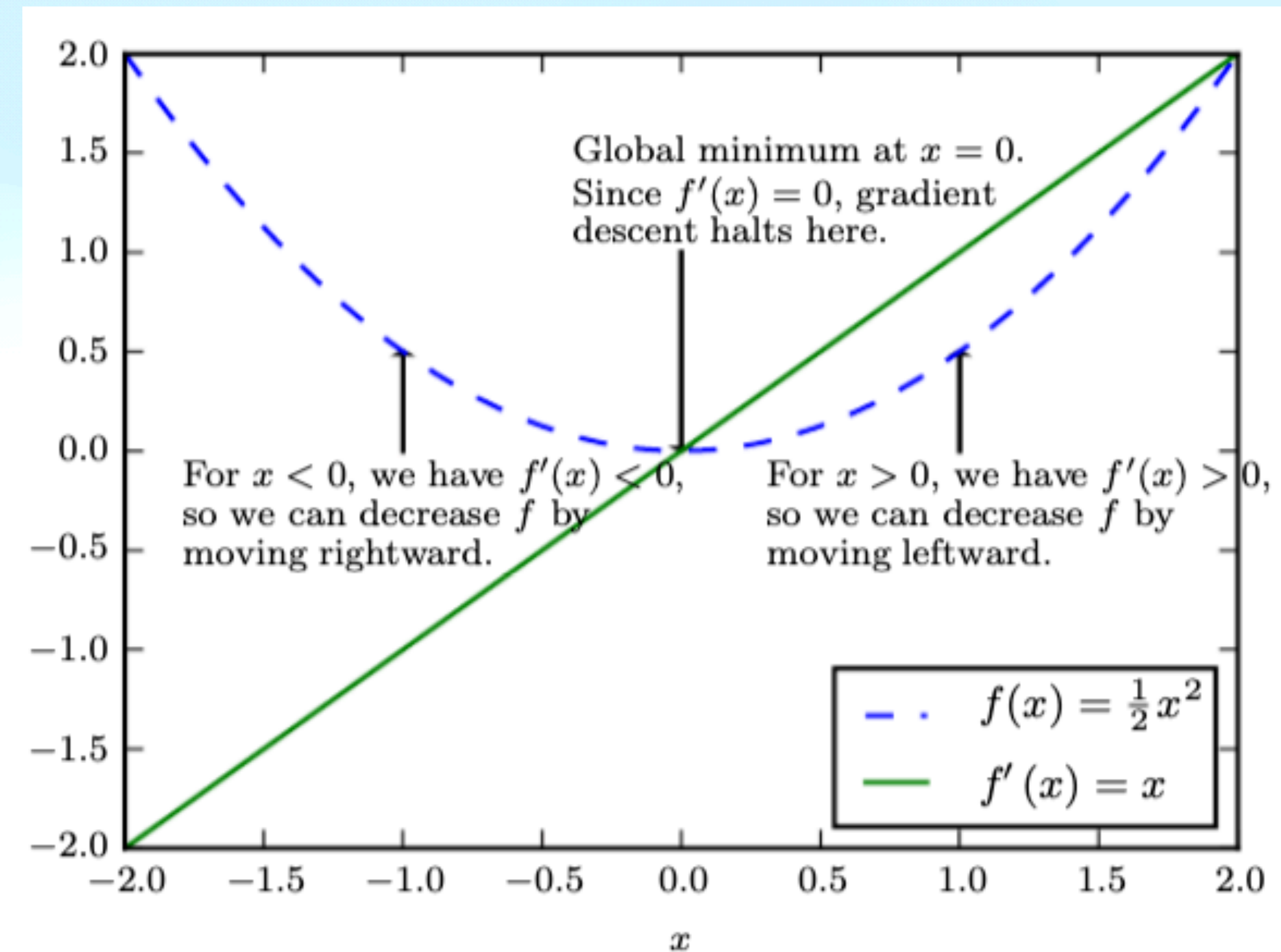
Feed waveform through the regressor object for output



🧠 Concept

- Training of neural network is a **gradient descent** optimization process:
 - To find $\operatorname{argmax}_x f(x)$, we calculate the gradient $\nabla_x f(x)$
 - Moving x in the opposite direction of gradient: $x' = x - \gamma \nabla_x f(x)$
 - γ is the **learning rate** of gradient descent
- Mainstream machine learning optimizers [Adam](#), [AdaGrad](#), [RMSprop](#) are first order optimization algorithms that only utilize gradient
- There are second order optimization algorithms which utilizes the Hessian matrix to avoid poorly conditioned probability space

$$f(x^{(0)} - \gamma g) = f(x^{(0)}) - \gamma g^T g + \frac{1}{2} \gamma^2 g^T H g$$

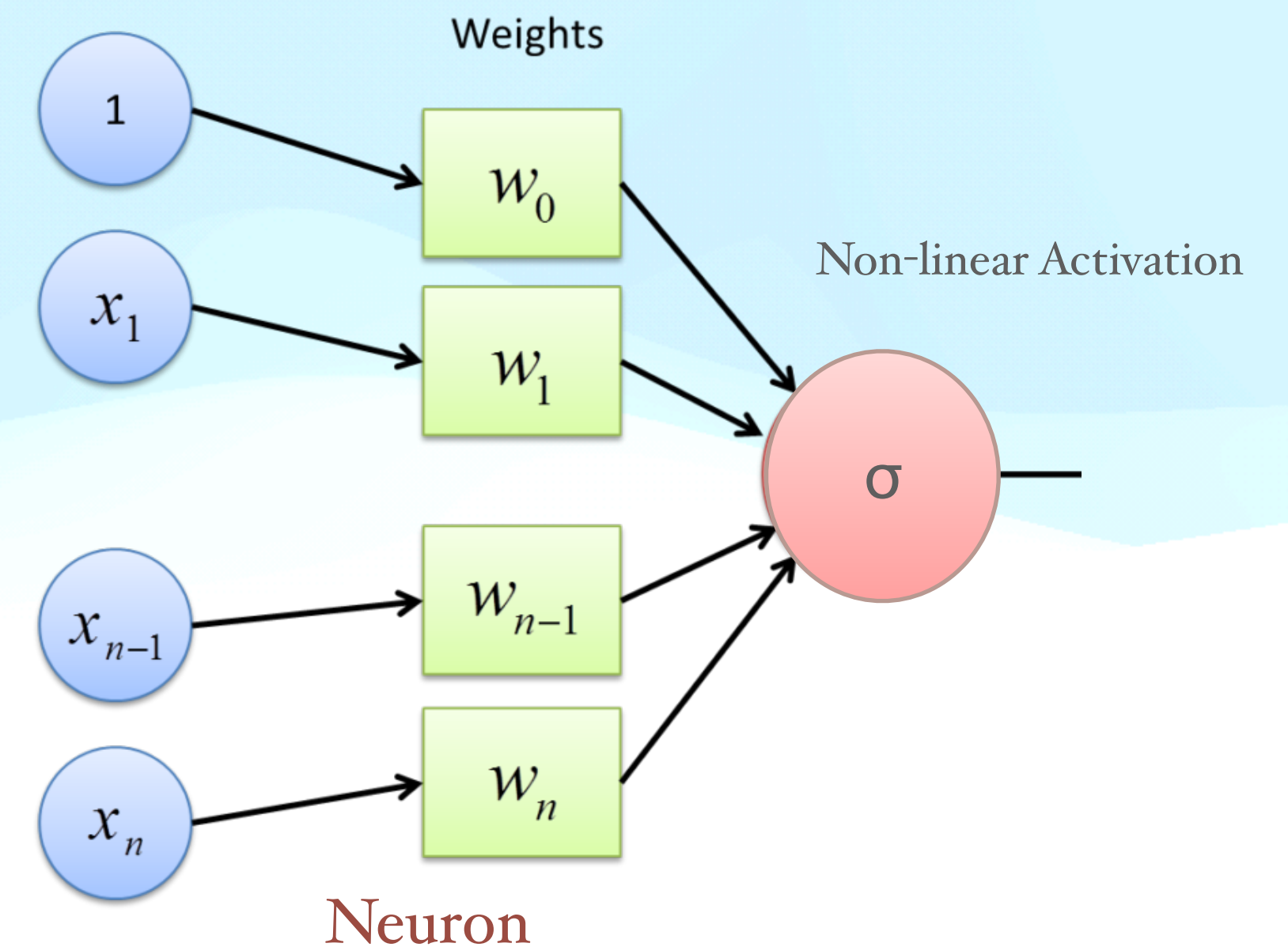




Theory

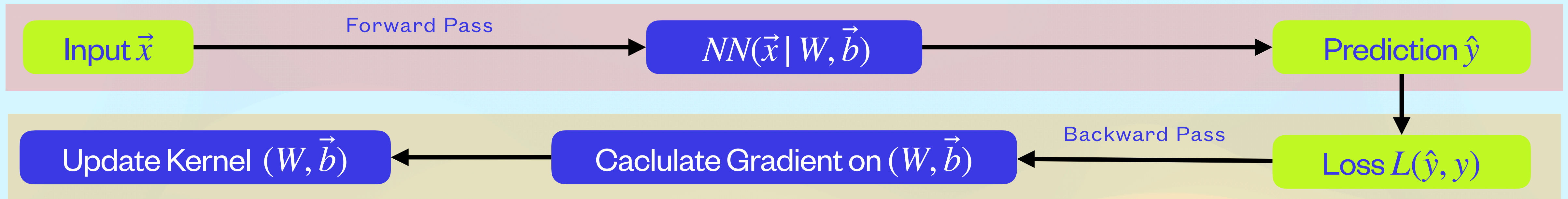
Expression of Neural Network:

- One Layer: $\hat{y} = \sigma(W_1 \vec{x} + \vec{b}_1)$
- Two Layer: $\hat{y} = \sigma_2(W_2 \sigma(W_1 \vec{x} + \vec{b}_1) + \vec{b}_2)$
- (W^i, \vec{b}_i) are the **weight matrices** of the i th layer, their values change during the training
 - also defined as the **kernel** of neural network



Concept Stochastic Gradient Descent (SGD):

- **Stochastic:** breaks the training data into **batches**, each batch is randomly sampled from the training dataset
- Size of batch is an important hyperparameter of NN model
- (α_m^T, β_k^T) is updated in a step-wise manner by sequentially feeding each batch into the model
- This will reproduce the effect of training using all data simultaneously
- SGD is possible because gradient is an expectation



Forward Pass:

- produce \hat{y} from input \vec{x} and kernel (W, \vec{b})

Concept

Backward Pass:

- Calculate loss $L(\hat{y}, y)$
- Calculate the gradient on kernel using back propagation
- Update kernel by gradient descent



Treating $NN(\vec{x} | W, \vec{b})$ as a multilayer complex function, we can use **chain rule** to calculate its derivative:

$$y = f_L(\dots f_2(f_1(x)))$$



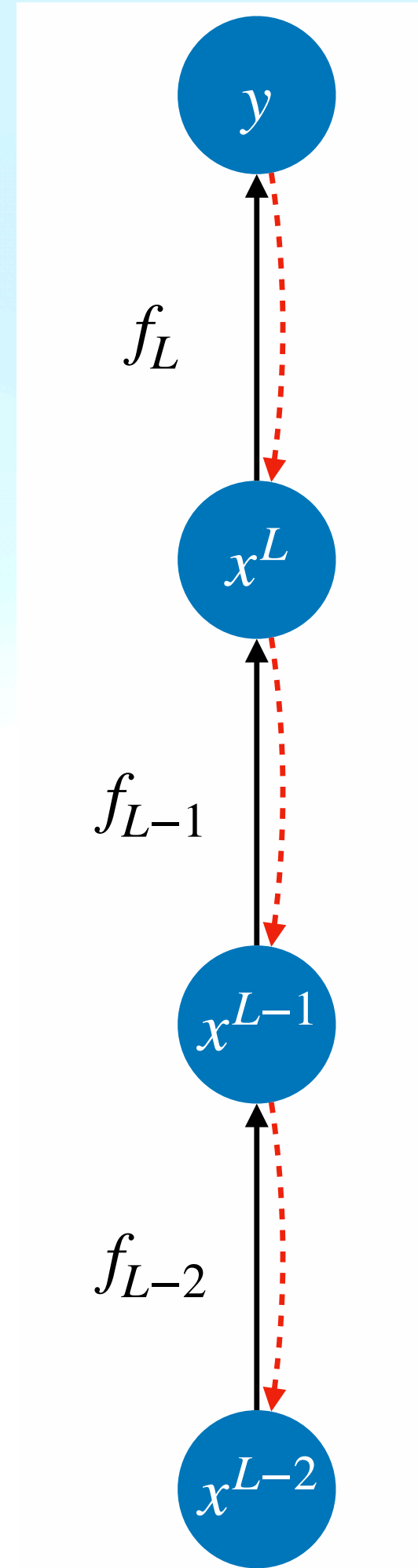
$$y = f_L(x_L)$$

$$x_L = f_{L-1}(x_{L-1})$$

⋮

$$x_1 = f_1(x_1)$$

We want to compute $\frac{\partial y}{\partial x_l}$ for all $l \in \{1, \dots, L\}$



$$\frac{\partial y}{\partial x_L} = \frac{\partial y}{\partial x_{L-1}} \frac{\partial x_L}{\partial x_{L-1}}$$

$$\frac{\partial y}{\partial x_{L-1}} = \frac{\partial y}{\partial x_{L-2}} \frac{\partial x_{L-1}}{\partial x_{L-2}}$$

$$\frac{\partial y}{\partial x_{L-2}} = \frac{\partial y}{\partial x_{L-3}} \frac{\partial x_{L-2}}{\partial x_{L-3}}$$

At each step we can reuse the computation of the previous step!



🧠 Concept

A **computation graph** is a directed graph where on each node we have an operation.

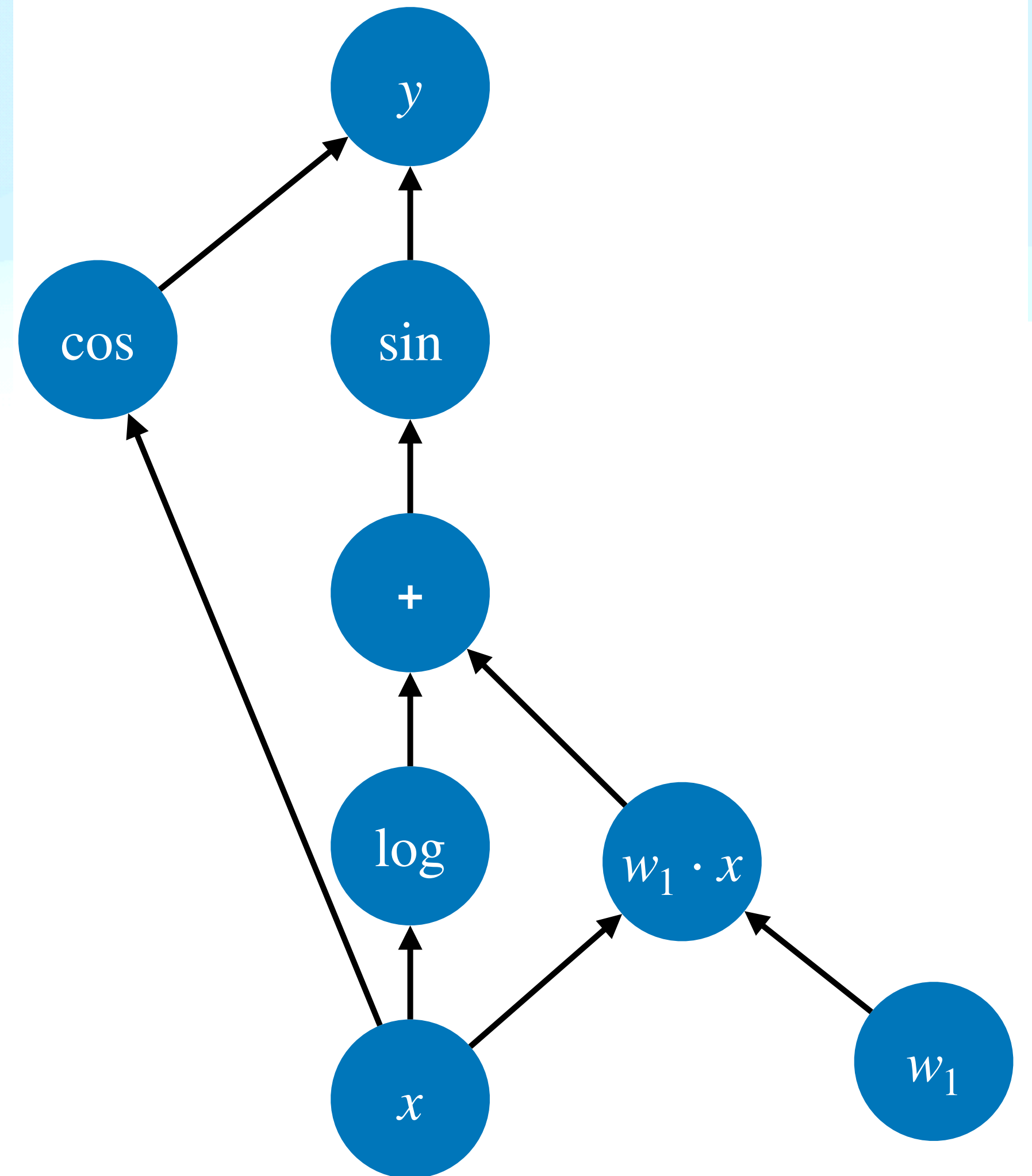
$$y = \sin(w_1x + \log(x)) + \cos(x)$$

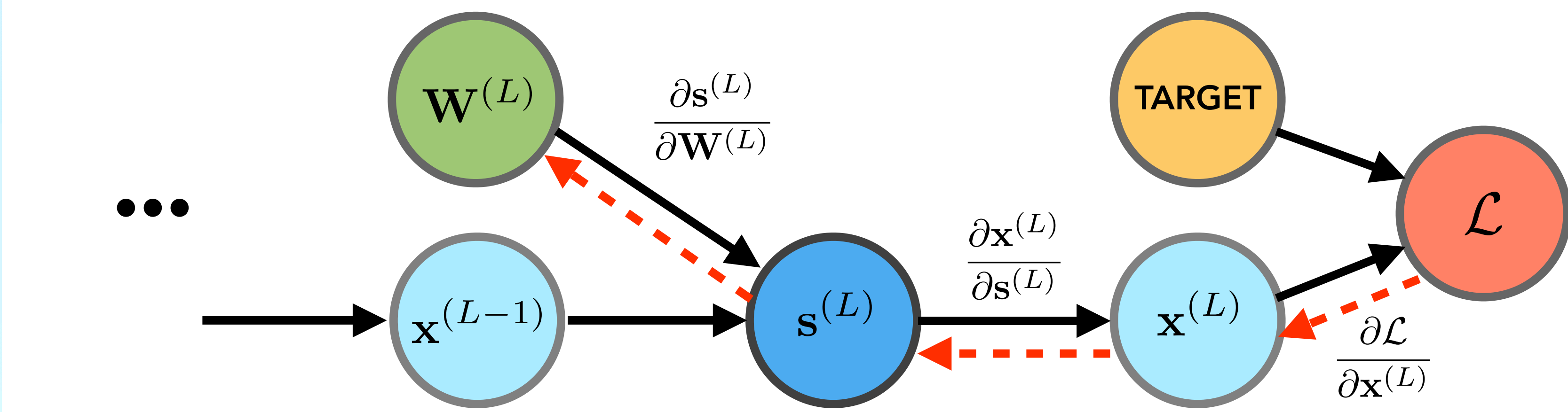
Modern deep neural networks are also computational graph! This means can calculate gradient along the graph.

most deep learning software libraries automatically handle this for you



just build the computational graph and define the loss





Theory

Given value of the loss, using **backpropagation equation** to calculate gradient with respect to each weight kernel parameters

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \frac{\partial \mathbf{s}^{(L)}}{\partial \mathbf{W}^{(L)}}$$

depends on the form of the loss

derivative of the non-linearity

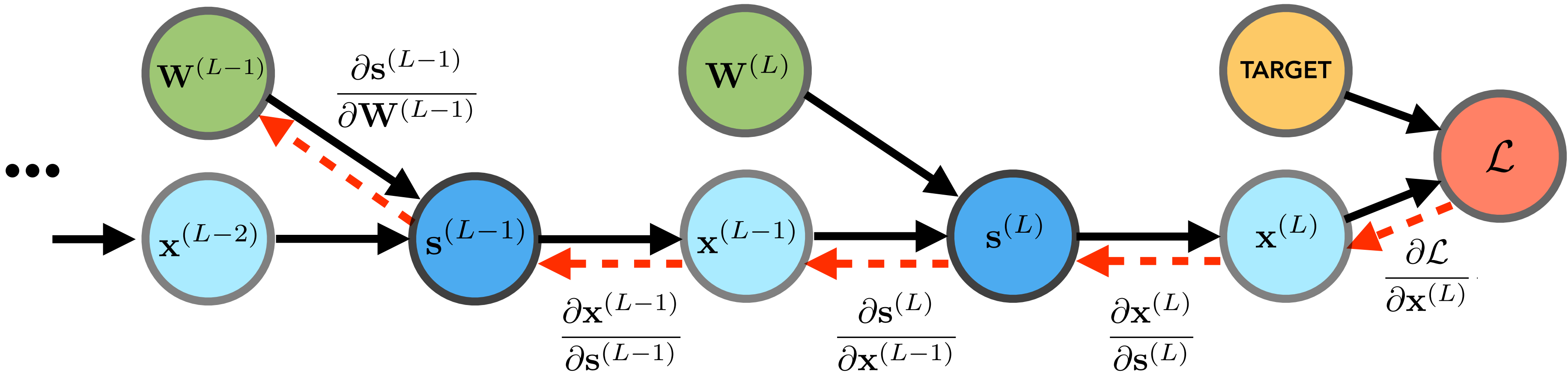
$$\frac{\partial}{\partial \mathbf{W}^{(L)}} (\mathbf{W}^{(L)\top} \mathbf{x}^{(L-1)}) = \mathbf{x}^{(L-1)\top}$$

note $\nabla_{\mathbf{W}^{(L)}} \mathcal{L} \equiv \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}}$ is notational convention



now let's go back one more layer...

again we'll draw the dependency graph:



$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L-1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \frac{\partial \mathbf{s}^{(L)}}{\partial \mathbf{x}^{(L-1)}} \frac{\partial \mathbf{x}^{(L-1)}}{\partial \mathbf{s}^{(L-1)}} \frac{\partial \mathbf{s}^{(L-1)}}{\partial \mathbf{W}^{(L-1)}}$$



Set up variables:

- **Model:** classifier
- **Loss Function:** criterion
- **Optimizer:** can use SGD, but we recommend torch.optim.Adam()

Epoch: number of iterations to train through the same dataset
train_loader: iterate through each batches within the dataset

Backpropagation: this single line of code does everything we discussed from sides 28-35!

This line reset the gradient to 0 at the end of current batch, so that we are ready to train the next batch of data

Code

```

classifier, criterion, optimizer = set_up_classifier()
train_loader, test_loader = get_dataloader()

loss_values = []
accuracy_values = []
y_true = []
y_pred = []

for epoch in range(NUM_EPOCHS):
    for i, (waveform, labels, energies) in tqdm(enumerate(train_loader)):
        classifier.train() # This line set the neural network to train mode

        waveform = waveform.to(DEVICE).float()
        labels = labels.to(DEVICE).view(-1,1).float()

        #Train the RNN classifier
        outputs = classifier(waveform).view(-1,1)

        # Calculate loss
        loss = criterion(outputs, labels)

        # Back-propagate loss to update gradient
        loss.backward()

        # Perform gradient descent to update parameters
        optimizer.step()

        # reset gradient to 0 on all parameters
        optimizer.zero_grad()

    print('\rEpoch [{0}/{1}], Iter [{2}/{3}] Loss: {4:.4f}'.format(
        epoch+1, NUM_EPOCHS, i+1, len(train_loader),
        loss.item(), end=""), end="")
    loss_values.append(loss.item())
  
```

Forward pass: feed data through the neural network model and calculate loss value (already discussed)

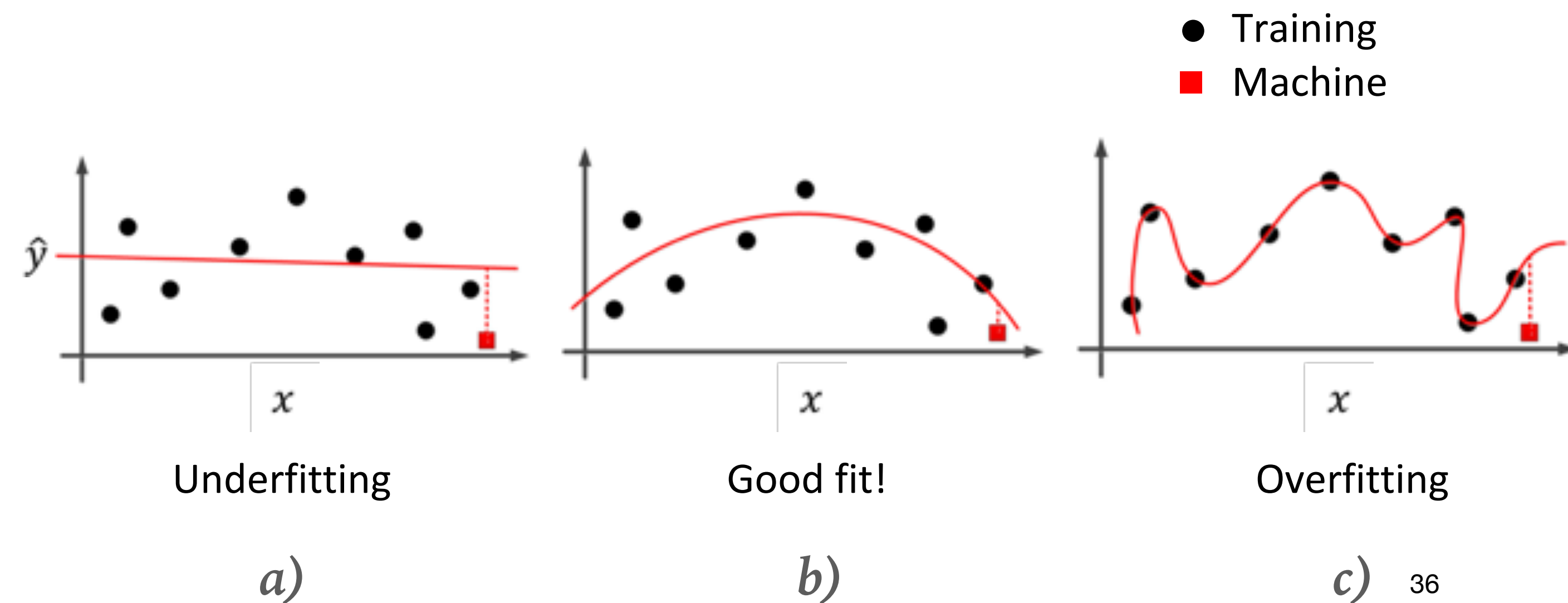
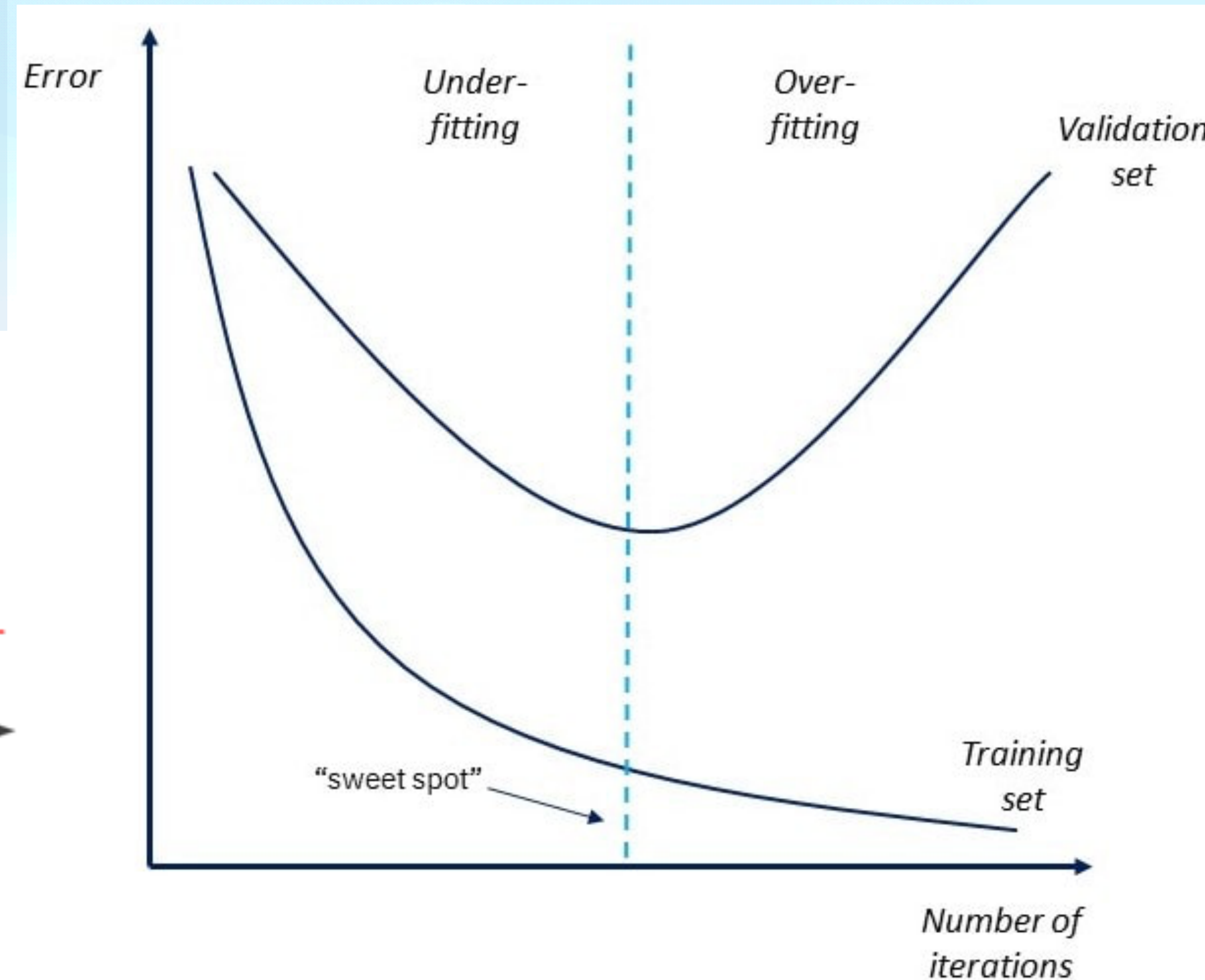
This line updates model parameters according to calculated gradients



🧠 Concept

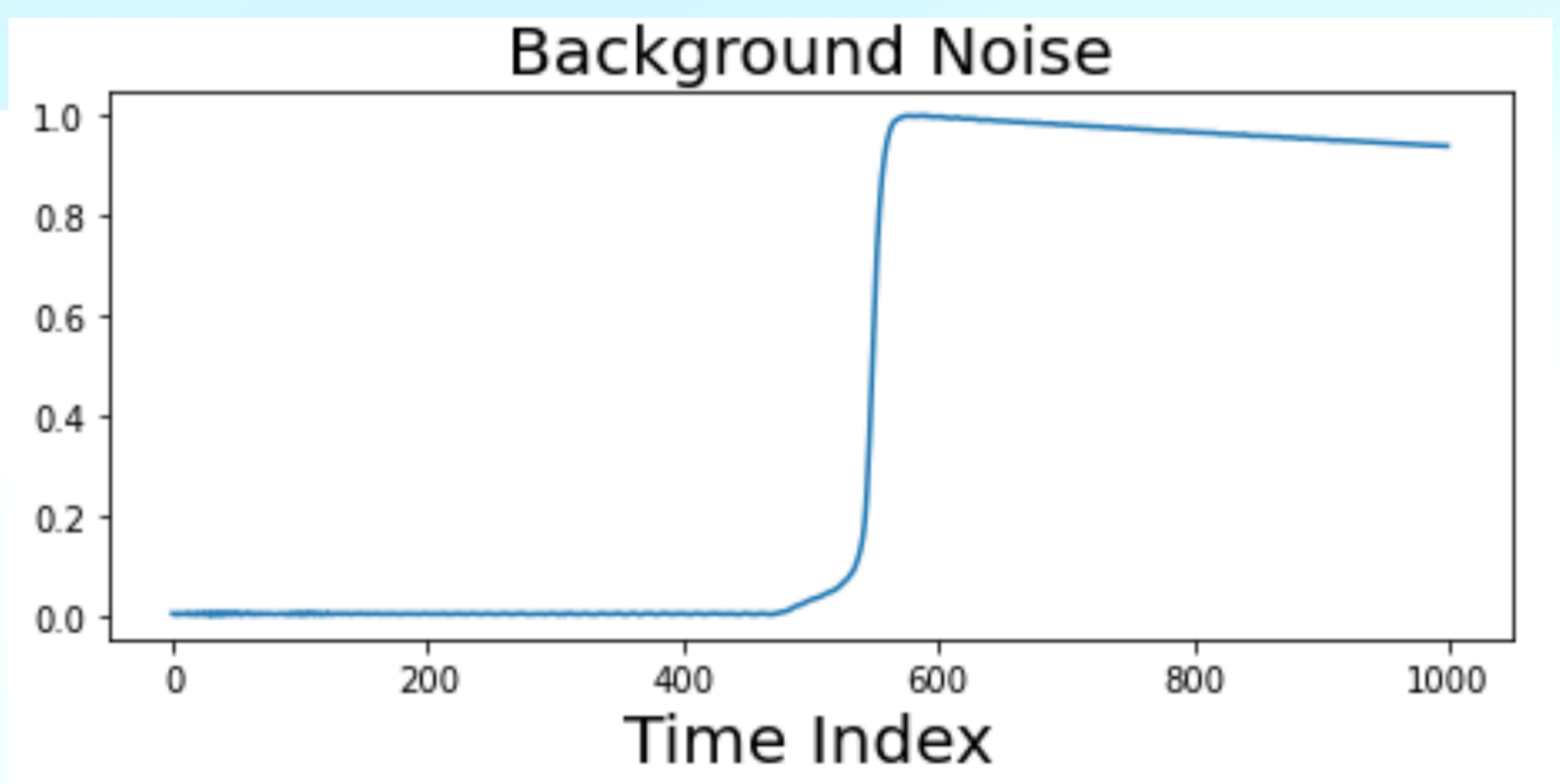
Underfitting: occurs when model cannot adequately capture the underlying structure of the data

Overfitting: model that corresponds too closely/exactly to the training data and fail to generalize to validation data





Time Series Data (BATCH_SIZE, 1000)



Concept

Feature Extractor Network

Take raw data as the input and output a low-dimensional vector

$NNLayer(1000, 512) \rightarrow \sigma$
 $\rightarrow NNLayer(512, 256) \rightarrow \sigma$
 $\rightarrow NNLayer(256, 32) \rightarrow \sigma$

Concept

Binary Classification 1 Task Module

Task Layer: $NNLayer(32, 1) \rightarrow$ One float point No.

- After training, it can be considered as a **“classification score”**:
 - Higher score means the answer is more likely “yes”
 - Lower score means the answer is more likely “no”
 - A **threshold** is need to distinguish “yes” from “no”

σ : $torch.sigmoid(x)$

Loss Function: $torch.nn.BCELoss()$



Setup: Classifying Majorana Demonstrator Waveforms

- Signal/positive: single-site waveform
- Backgrounds/negative: multi-site waveform
- λ : Classification score produced by training the NN
- Cutting Threshold: set at to at 0.5 to define our yes/no answer:
 - $\lambda > 0.5$: event is a signal
 - $\lambda \leq 0.5$: event is a background

	psd_label_avse: True Signal	psd_label_avse: True Background
Classified As Signal	True Positive (TP)	False Positive (FP)
Classified as bkg	False Negative (FN)	True Negative (TN)

Concept

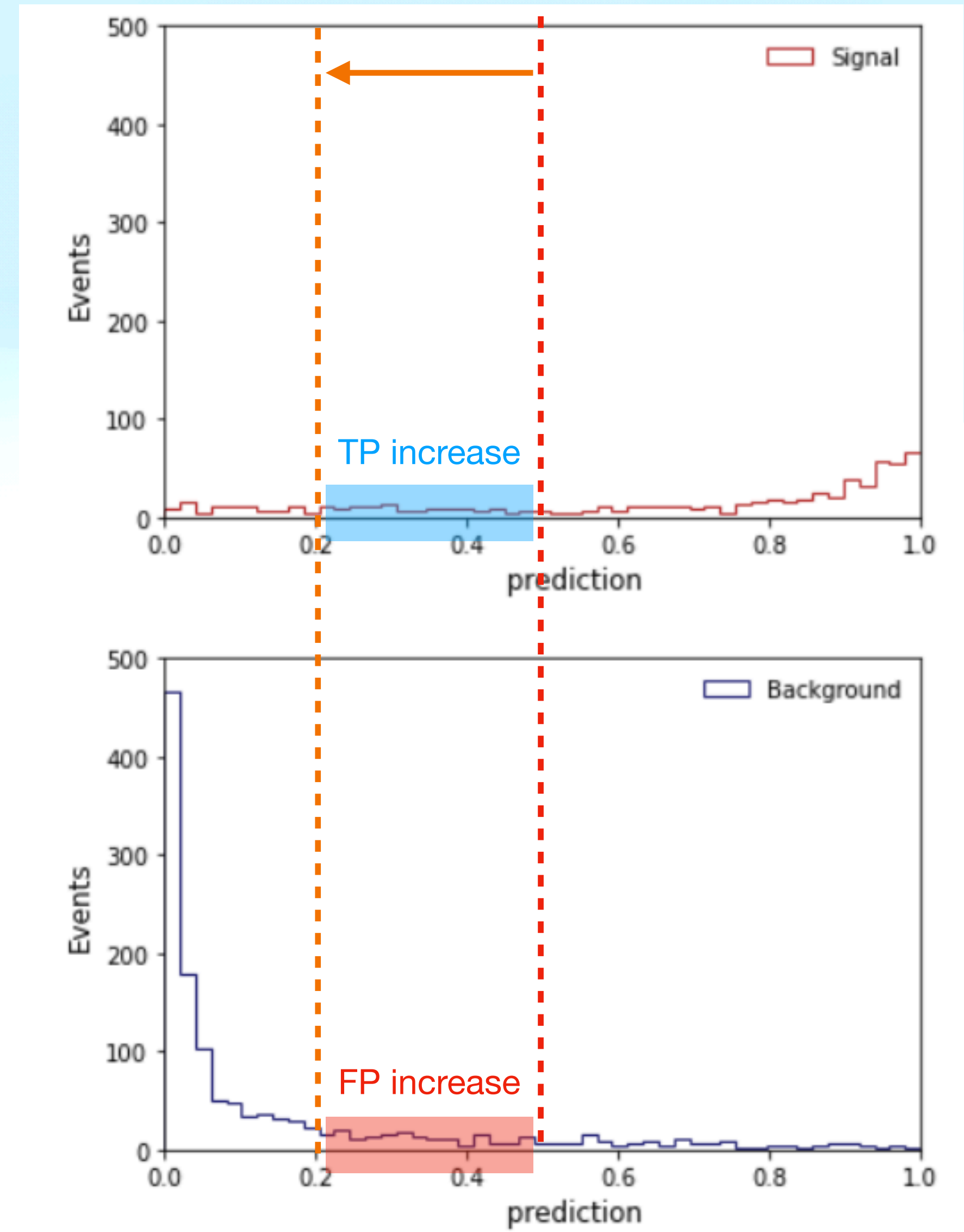
True Positive Rate (TPR) = $\frac{TP}{TP + FN}$ [TP+FN is the total number of signal in the dataset]

False Positive Rate (FPR) = $\frac{FP}{FP + TN}$ [FP+TN is the total number of backgrounds in the dataset]



Setup: Classifying Majorana Demonstrator Waveforms

- Signal/positive: single-site waveform
- Backgrounds/negative: multi-site waveform
- λ : Classification score produced by training the NN
- Cutting Threshold: set at to at 0.5 to define our yes/no answer:
 - $\lambda > 0.5$: event is a signal
 - $\lambda \leq 0.5$: event is a background
- Model did not change at all
- simply changing the cutting threshold will result in different TPR & FPR
- We need a threshold-independent metric to compare model performance!





Concept
Receiver Operating Characteristic (ROC) Curve: Evaluate our model at all thresholds possible

Cutting at -0.1:

- TPR = 100%
- FPR = 100%

Cutting at 0.2:

- TPR = 87%
- FPR = 24%

Cutting at 0.5:

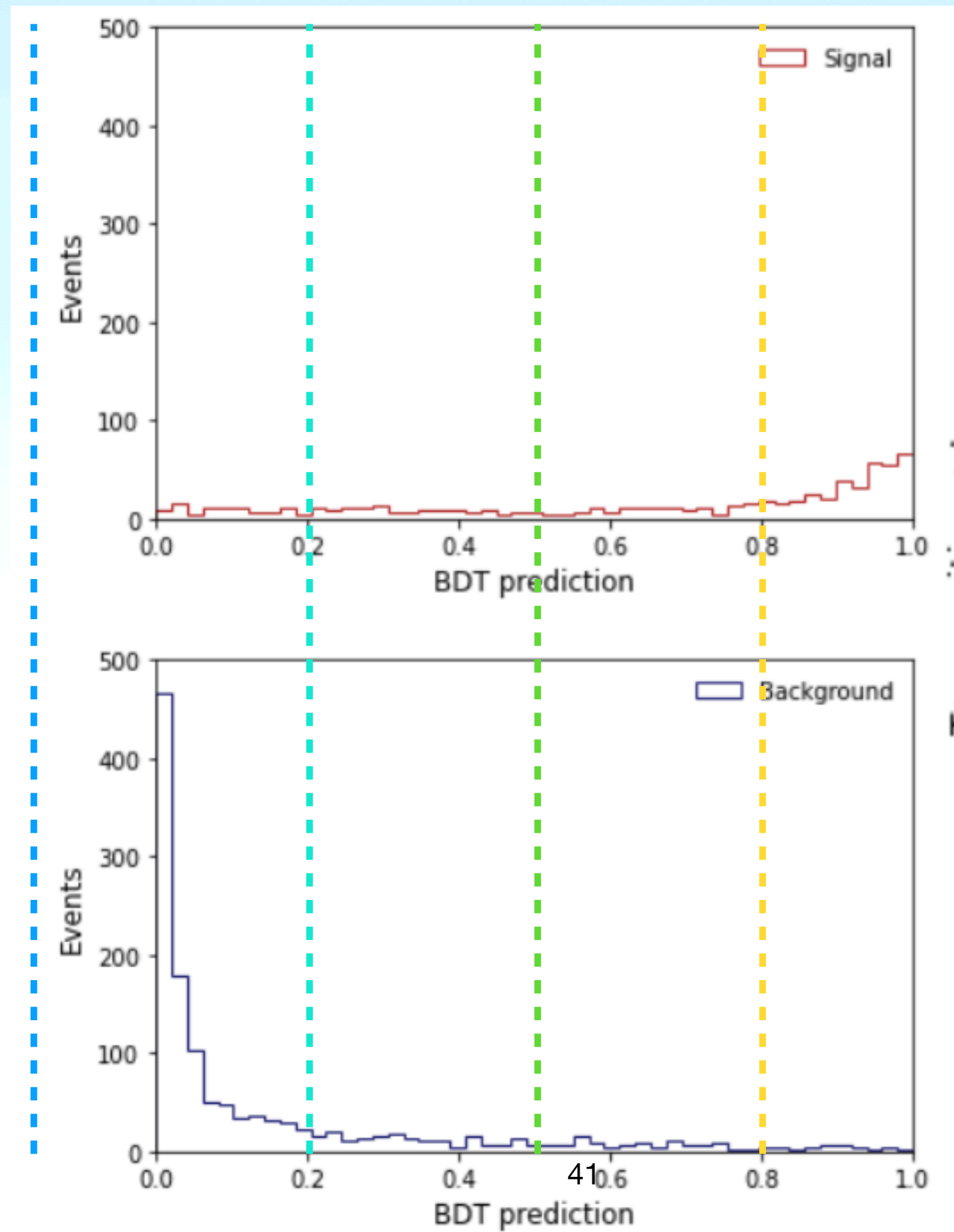
- TPR = 70%
- FPR = 10%

Cutting at 0.8:

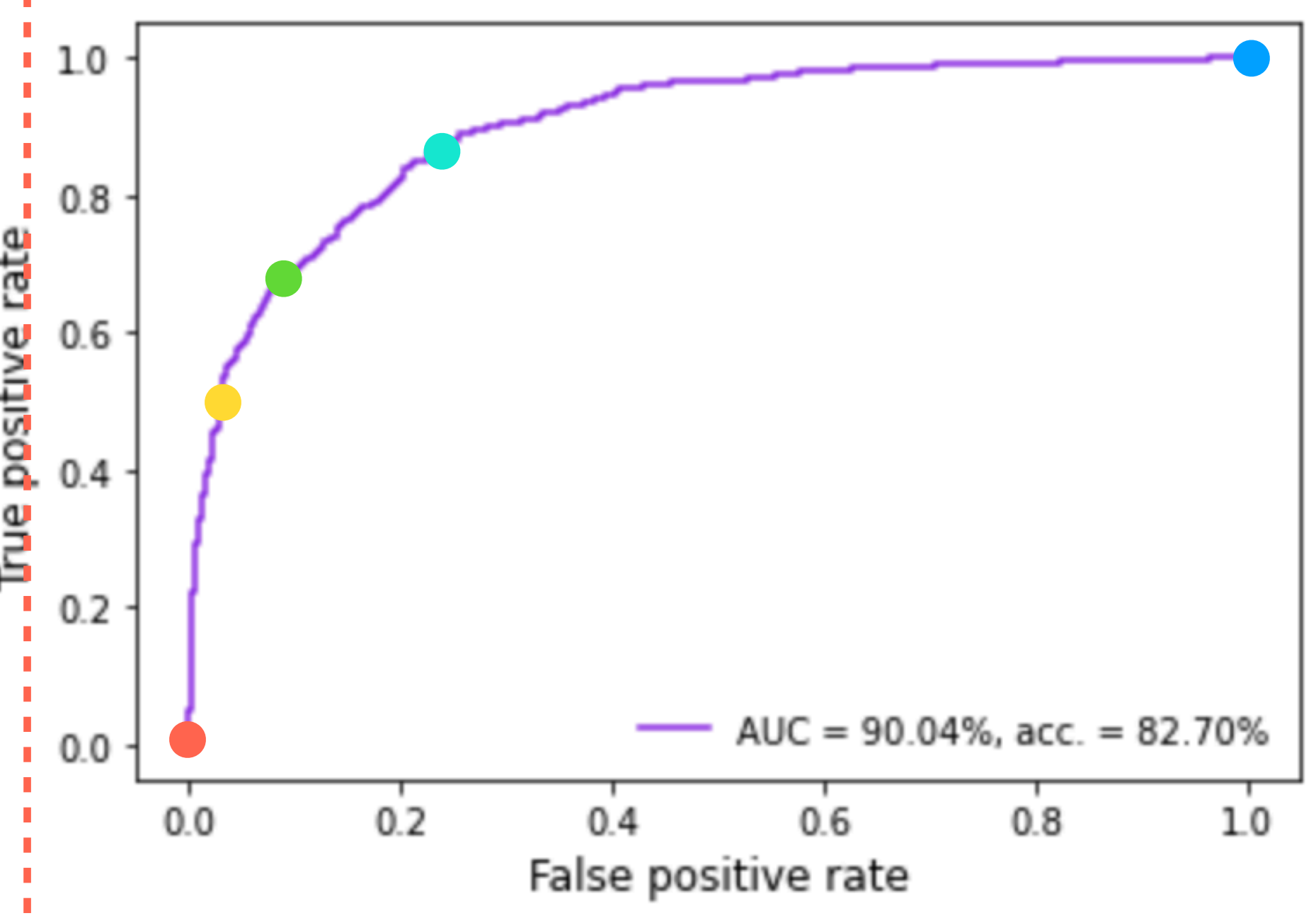
- TPR = 50%
- FPR = 3%

Cutting at 1.1:

- TPR = 0%
- FPR = 0%



All ROC curves start at (1,1)



All ROC curves ends at (0,0)



Perfect ROC Curve

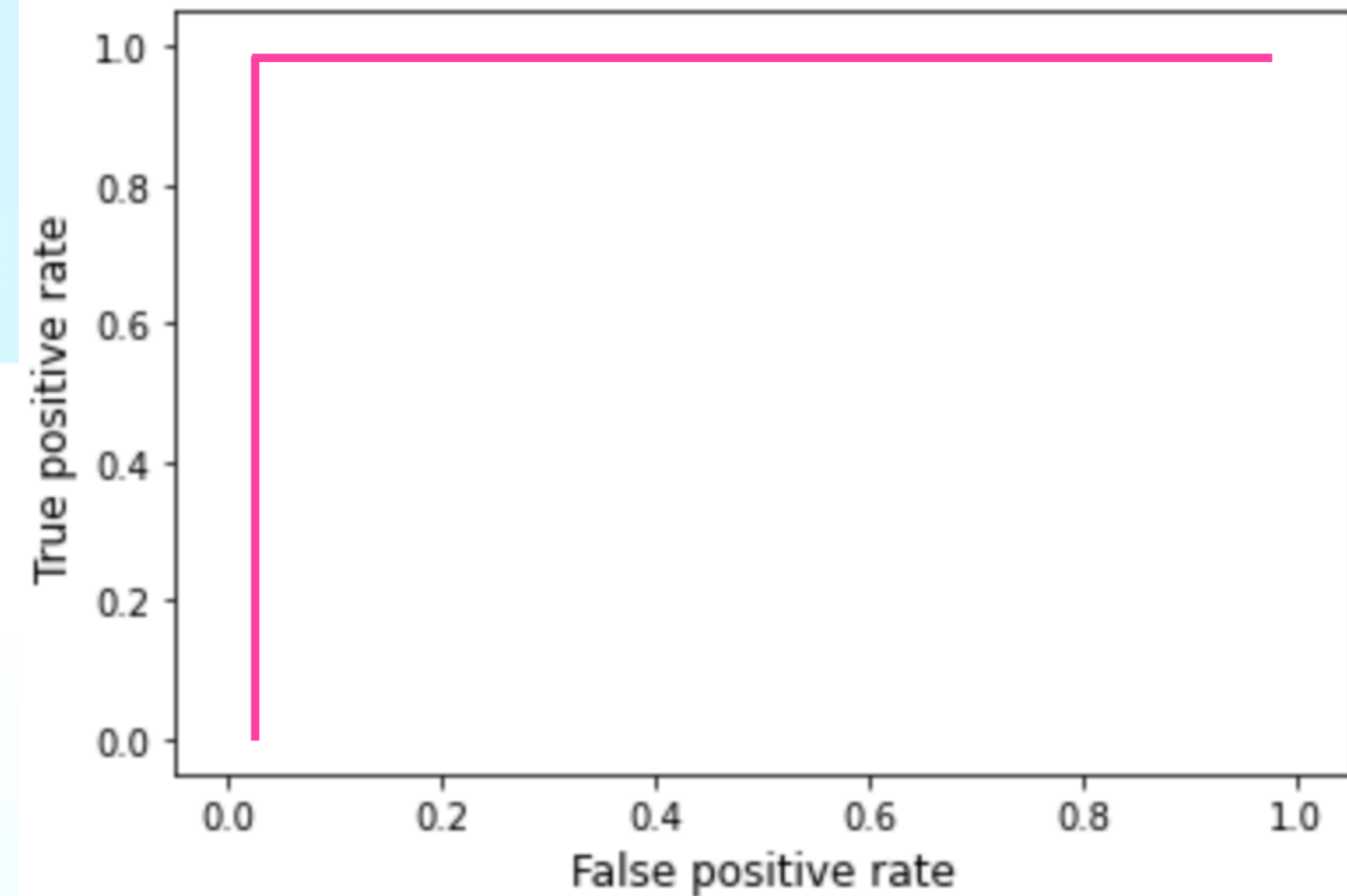
- TPR = 100% at FPR = 0%
- Area under curve = 1
- Every event is classified correctly

Worst ROC Curve

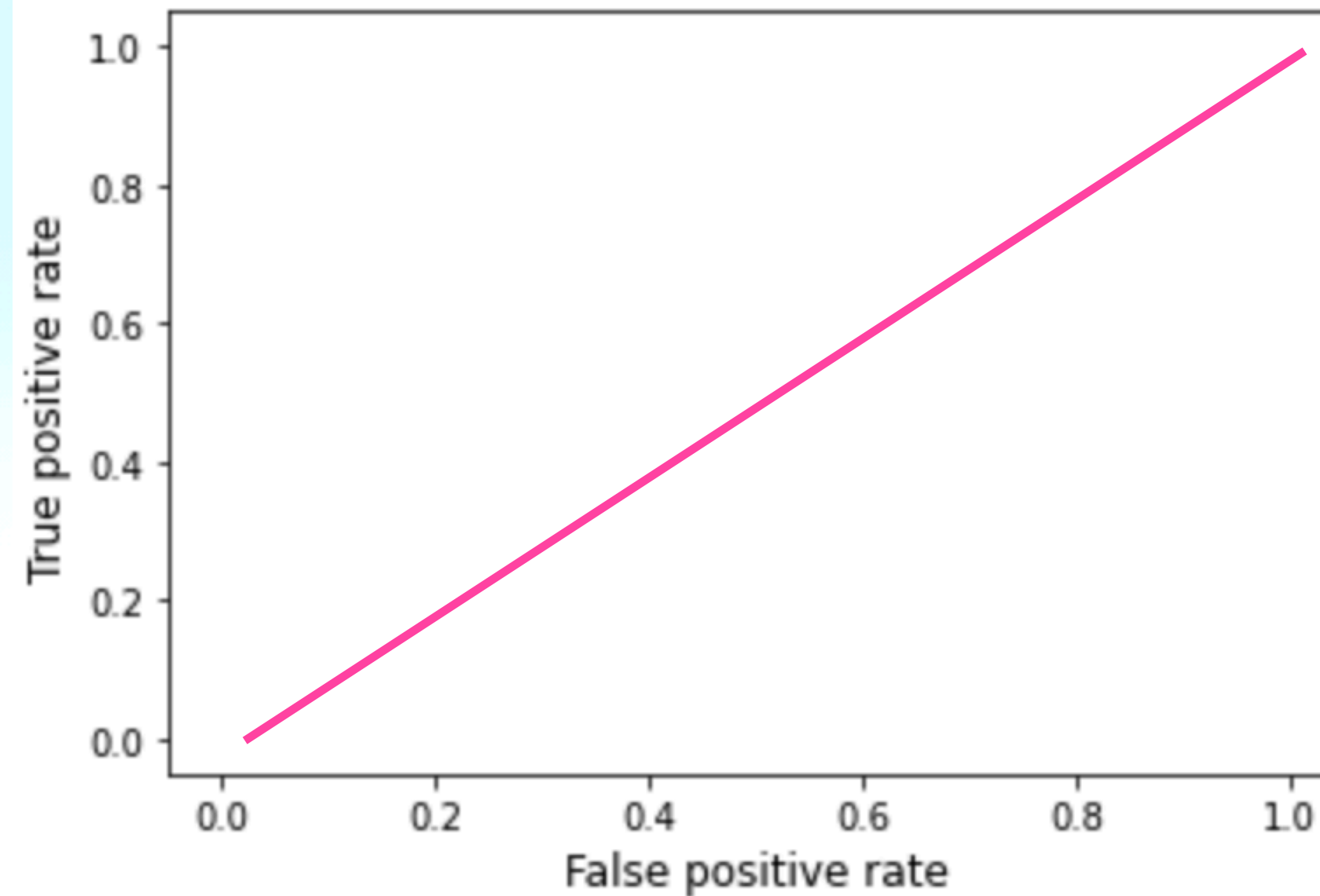
- TPR == FPR
- Area under curve = 0.5
- Classify by tossing a coin

Anti-perfect ROC Curve

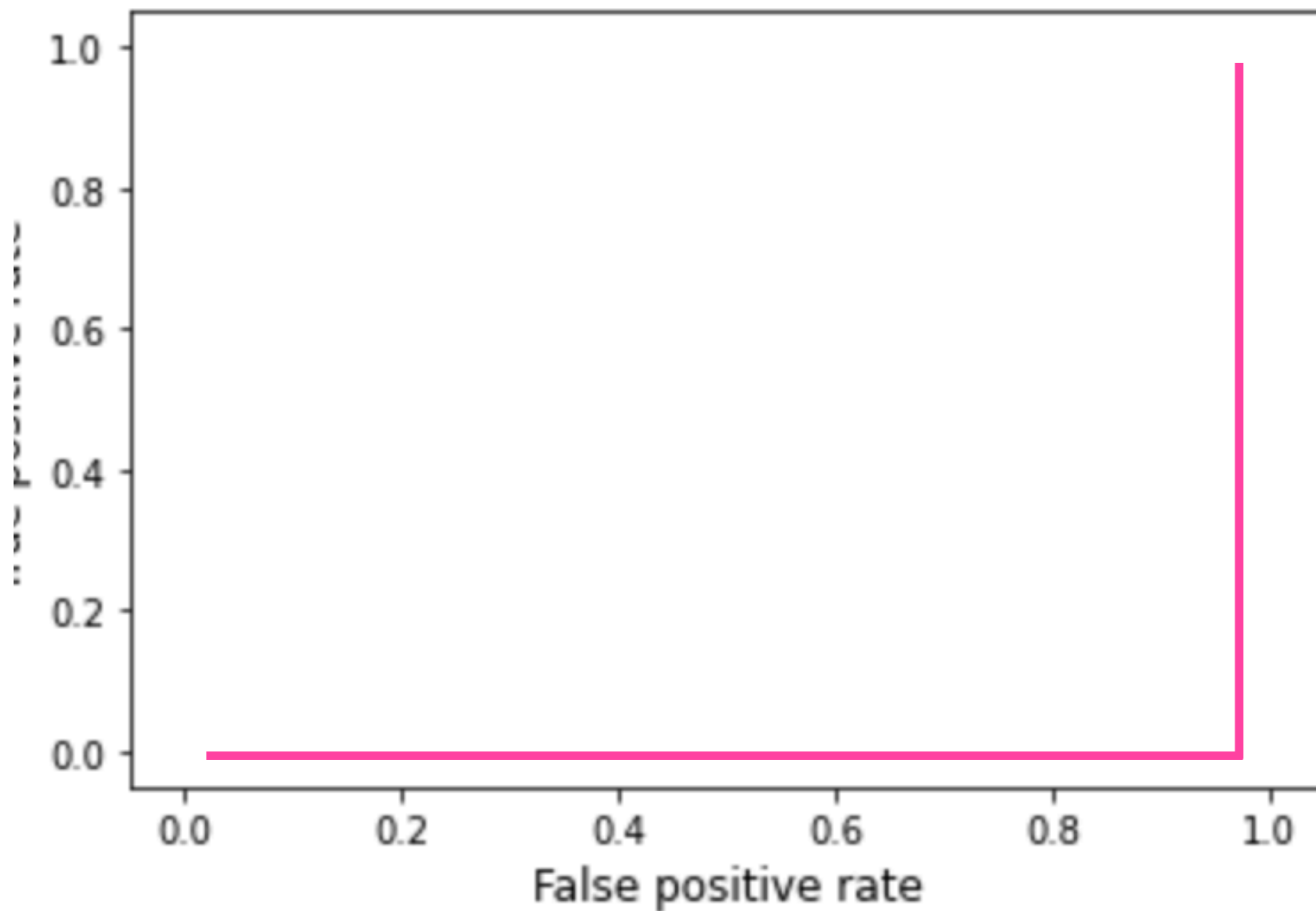
- TPR = 0% at FPR = 100%
- Area under curve = 0
- Every event is classified wrong



$$\forall \lambda : \lambda_{sig} > \lambda_{bkg}$$



$$\lambda \sim \text{Unif}(0,1)$$



$$\forall \lambda : \lambda_{sig} < \lambda_{bkg}$$

🧠 Concept

The **Area Under the Curve (AUC)** of ROC curve is equal to the probability that a model will rank a randomly chosen signal higher than a randomly chosen background

1. Majorana Demonstrator Dataset and HPGe Detector
2. PyTorch Dataset Class
3. **Data pre-processing**: transform your data to remove unwanted informations
4. Wrap dataset object to create a **data loader**

1. **Task Module: Task Layer + Loss Function**
2. **Cross Entropy**
3. Binary Classification 1-3
4. Multiclass Classification

1. **Stochastic Gradient Descent**
2. **Backward Pass**: Computational graph and **Backpropagation**
3. PyTorch implementation: one line



1. **Neural Network**: linear layer and activation function
2. How to create a simple NN in PyTorch
3. **Forward pass**

MSE Loss, L1 Loss, and Smooth L1 Loss

1. **Overfitting** vs. Underfitting
2. Evaluate binary classification: **ROC Analysis**
3. Evaluate regression: reconstruction algorithm

Some Useful Links

Course Material

All lecture materials: [Link](#)

Oνββ AI Summer School [Link](#)

Overview

Timetable

Contribution List

My Conference

My Contributions

Registration

Participant List

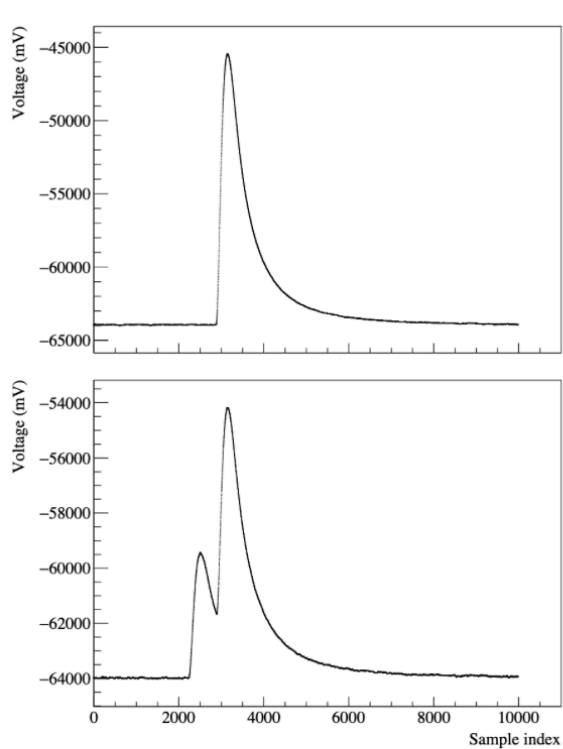
Datasets

Contact

✉ ai-summer@ucsd.edu

Datasets

Physics Datasets



CUORE
This dataset contains 10,000 triggered events from the CUORE (Cryogenic Underground Observatory for Rare Events) experiment. Each event is a 10-s waveform, beginning with a 3-s pretrigger baseline. CUORE digitizes their detector readout at 1 kHz, thus each event contains 10,000 samples. Two classes of CUORE detector pulses are included in this dataset: single-pulse events and pile-up events; the latter contains two or more pulses. This dataset is evenly split between the two classes. Only 90% of the total dataset is labeled for training purposes, while the remaining 10% is unlabeled and reserved for testing.

- **Input Data:** 1D array with 10000 samples (Waveform), eventId
- **Labels:** 1 Binary Classification label (clean single-pulse event: 0, pile-up pulse event: 1)
- **Format:** HDF5

Note: The waveforms stored in the HDF5 files are raw and have not been normalized. The provided normalization parameters allow normalization to be applied during data loading.

[Download Dataset \(Zenodo\)](#)
[Read Description under cuore directory on NERSC](#)

FIG. 1. Representative CUORE detector pulses. **Top:** A clean single-pulse event illustrating the characteristic fast rise (~50 ms) and slow exponential decay (~1 s) of the CUORE cryogenic calorimeters. **Bottom:** A two-pulse pile-up event in which the triggered pulse begins while an earlier pulse is still decaying, producing a distorted double-peaked waveform.

09:00 → 09:10	Opening: Welcome Convener: Aobo Li Ovbb_AI_summer.pdf
09:10 → 10:30	Lecture 1: Intro to Datasets Speaker: Zepeng Li (University of Hawaii Manoa) AI_SummerSchoolL...
10:30 → 10:40	Break
10:40 → 12:00	Lecture 2: Intro to Machine Learning Speaker: Aobo Li Intro_to_ML.pdf
12:00 → 16:00	Hands-On Session 1: Create Your First AI Model (Lunch Provided) AISummerSchool_V...
16:00 → 17:00	Panel Discussion: Turning your physics task into AI model Panel_Discussion.pdf video1895644319...
SUNDAY, 21 JUNE	
09:00 → 10:30	Lecture 3: Advanced ML Models Speaker: Vivek Sharma (U. Pittsburgh) ovbb_ML_school_le...
10:30 → 10:40	Break
10:40 → 12:10	Lecture 4: LLM, AI Agent, and Vibe Coding Speaker: Yue Ma (UC San Diego) Neutrino2026_AI_su...
12:10 → 16:00	Hands-On 2: Upgrading Your ML Model with Vibe Coding (Lunch Provided) aisummerschool_v4... vibe_coding_compe...
16:00 → 17:00	Closing: Final Presentation, Award and Closing Remarks



My Website:

<https://aobol.github.io/AoboLi/>

My Email Addresses:

aol002@ucsd.edu