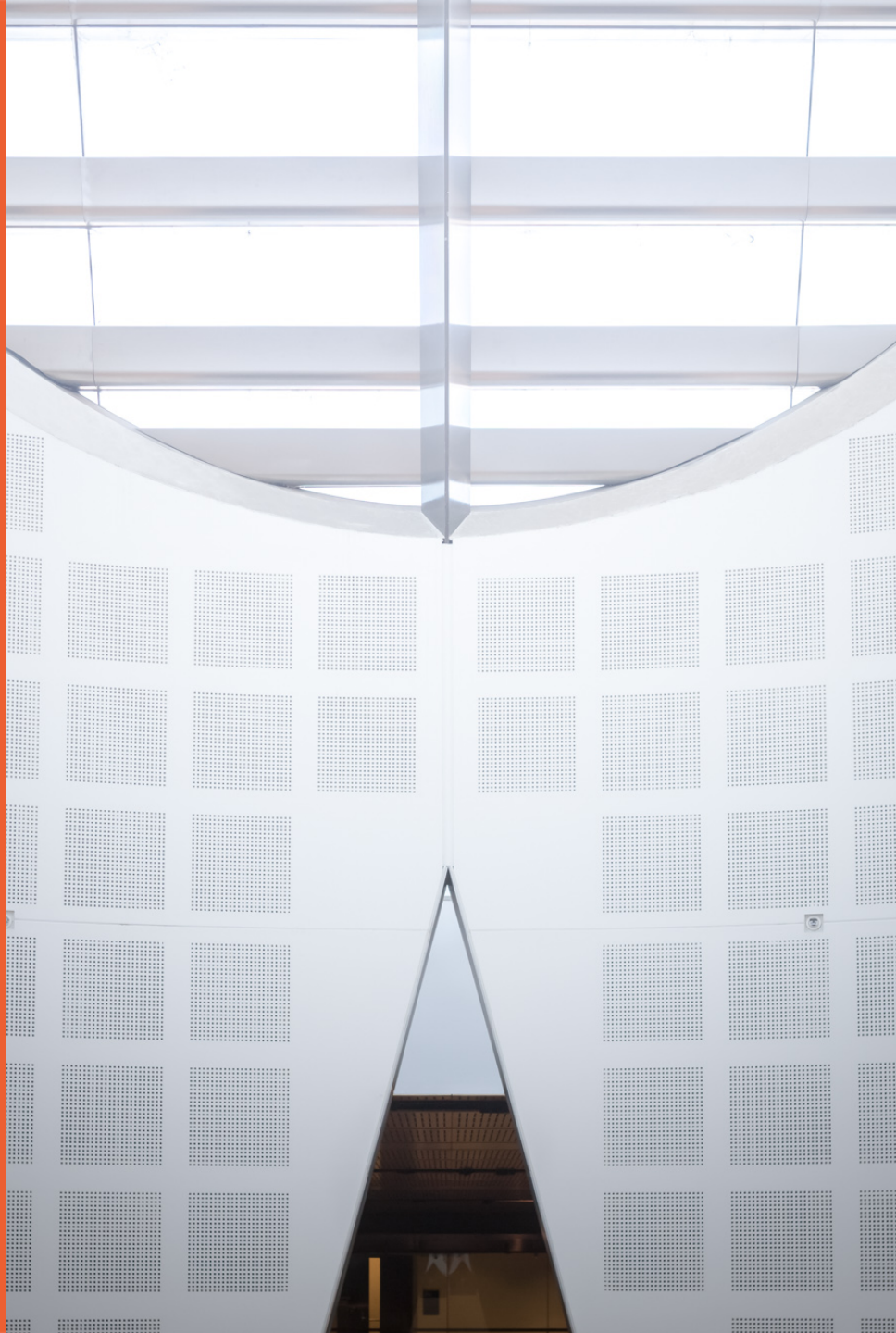
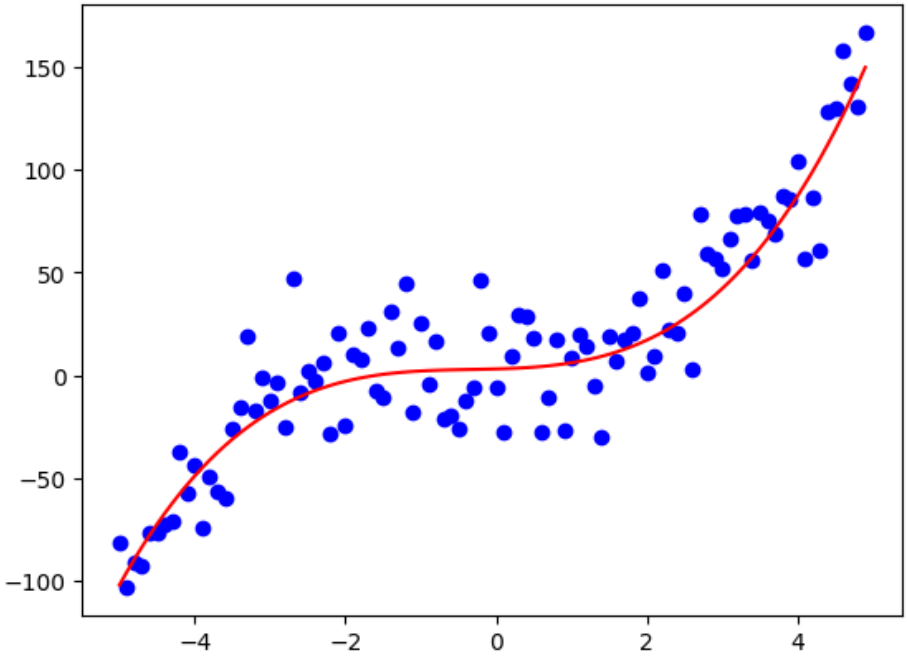
Week 8 Drug Discovery Tutorial



THE UNIVERSITY OF SYDNEY

Our Objective



Regression in is a fundamental statistical technique used for predicting numerical values based on input data. It involves building a model that can capture the relationship between independent variables (inputs) and a continuous target variable (outputs), enabling us to make accurate predictions or estimate values within a specific range.

Packages

1. PyTorch

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs. Most machine learning and deep learning workflows involve working with *data*, *creating models*, *optimizing model parameters*, and *saving the trained models*. PyTorch provides useful components to construct such workflows.

2. PyTorch Geometric

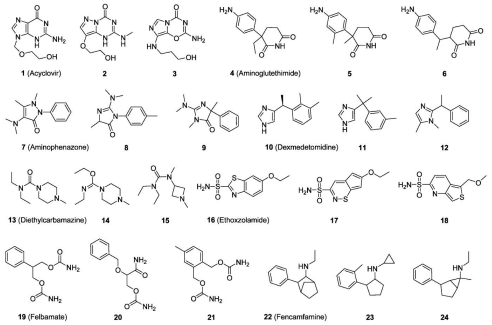
PyTorch Geometric is a library built upon PyTorch to easily write and train *Graph Neural Networks* (GNNs) for a wide range of applications related to structured data.

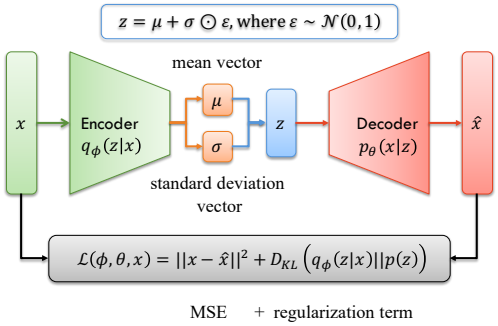
Pipeline

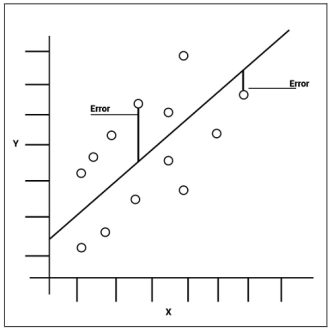
Load and Pre-process Data

Construct Graph Neural Networks

Optimization and Checkpoints







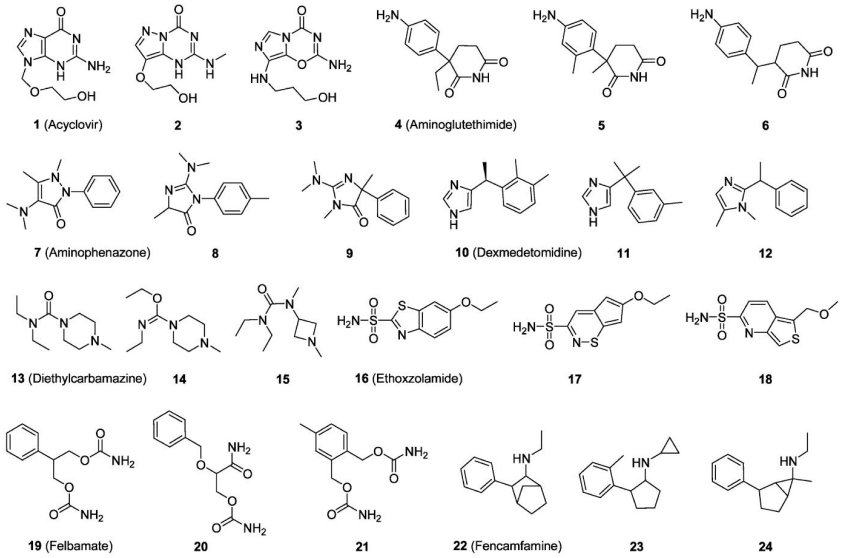
1. Load and Pre-process Data

Load and Pre-process Data

- torch_geometric.datasets.QM9
- torch_geometric.loader.DataLoader
- torch_geometric.transforms.Compose
- torch_geometric.transforms.Distance
- torch_geometric.transforms.NormalizeFeatures

torch_geometric.datasets.QM9

Return the QM9, consisting of about 130,000 molecules and 19 regression targets. Each molecule includes complete spatial information for the single low energy conformation of the atoms in the molecule.



Parameters:

- **root:** Root directory where the dataset should be saved. It should be a string.
- **transform:** A function or transform that takes in a data object and returns a transformed version. The data object will be transformed before every access.

torch_geometric.transforms

- torch_geometric.transforms.Compose
  - Composes several transforms together.
  - Parameters:
    - **transforms:** A list of transforms to compose.

- torch_geometric.transforms.Distance
  - Saves the Euclidean distance of linked nodes in its edge attributes.
  - Parameters:
    - **norm:** If set to True, the output will be normalized to the interval [0, 1]. We set to False and do not normalize the distance.

- torch_geometric.transforms.NormalizeFeatures
  - Row-normalizes the attributes given in “attrs” to sum-up to one.

torch_geometric.loader.DataLoader

A data loader which merges data objects from a dataset to a mini-batch.

Parameters:

- **dataset:** The dataset from which to load the data.
- **batch_size:** How many samples per batch to load.
- **shuffle:** If set to True, the data will be reshuffled at every epoch. We set to True for training and False for validation.

2. Construct Graph Neural Networks

Construct Graph Neural Networks

- torch.nn.Module
- torch.nn.functional.dropout
- torch.nn.functional.relu

- torch_geometric.nn.GCNConv
- torch_geometric.nn.VGAE
- torch_geometric.nn.InnerProductDecoder
- torch_geometric.nn.Set2Set

torch.nn.Module

- The base class for all neural network modules (e.g., GCNConv, VGAE etc.). Our models also subclass it. Modules can also contain other Modules, allowing to nest them in a tree structure.
- We need to define an "__init__" method to describe the components of the module and a "forward" method to describe the calculation of the module.

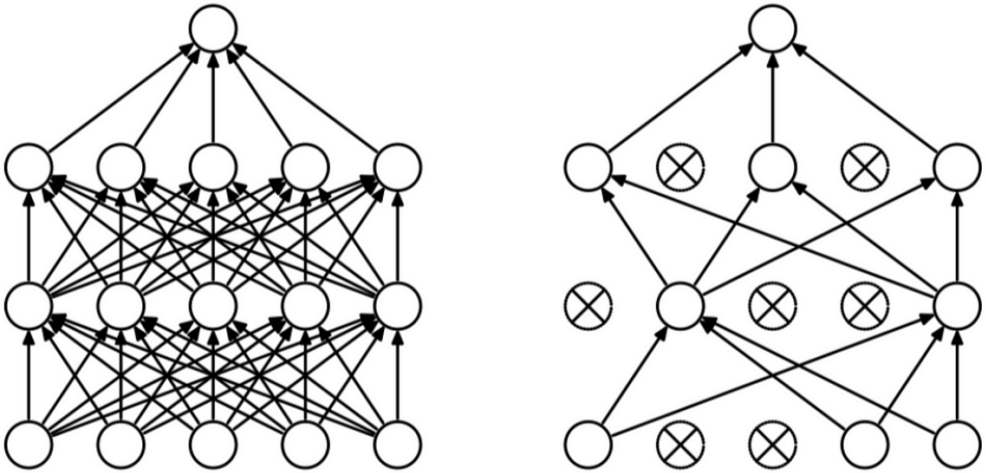
- Example:

```
class MyModel(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super(MyModel, self).__init__()
        self.conv = GCNConv(in_channels, out_channels)

    def forward(self, x, edge_index):
        return self.conv(x, edge_index)
```

torch.nn.functional

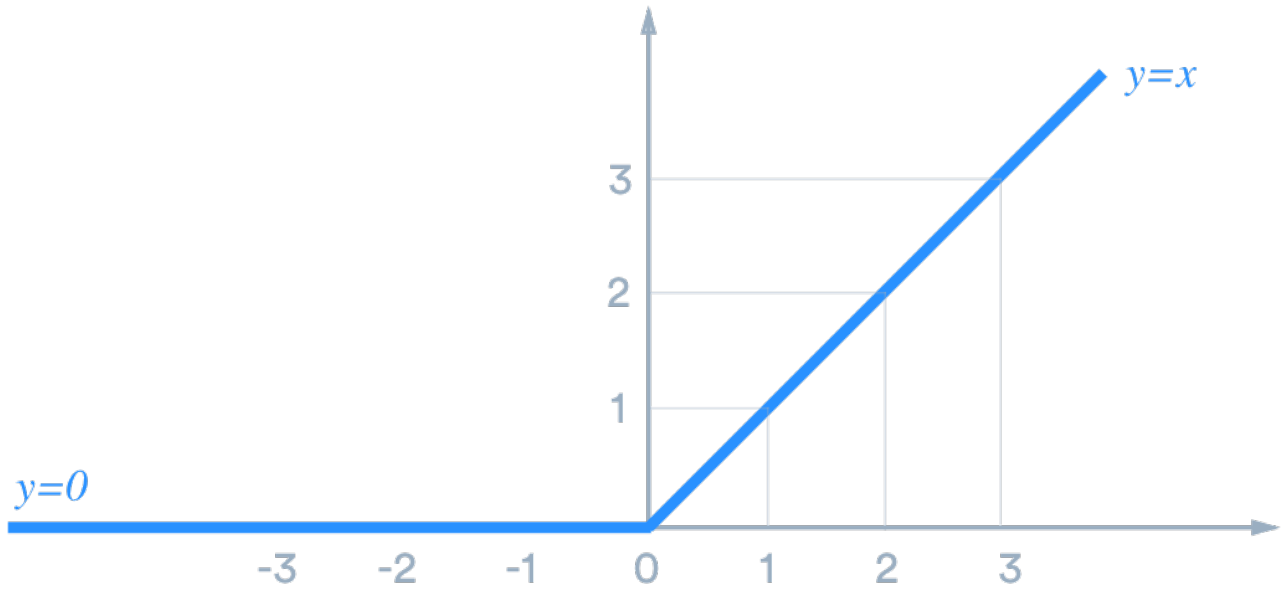
- torch.nn.functional.dropout
  - During training, randomly zeroes some of the elements of the input tensor with probability p using samples from a Bernoulli distribution.
  - Parameters:
    - **p:** Probability of an element to be zeroed. We use 0.5.
    - **training:** Apply dropout if is True. We set to True for training and False for validation.



(a) Standard Neural Net (b) After applying dropout.

torch.nn.functional

- torch.nn.functional.relu
  - The ReLU activation function, which applies the rectified linear unit function element-wise.
  - Parameters:
    - **input:** The input to the ReLU function.

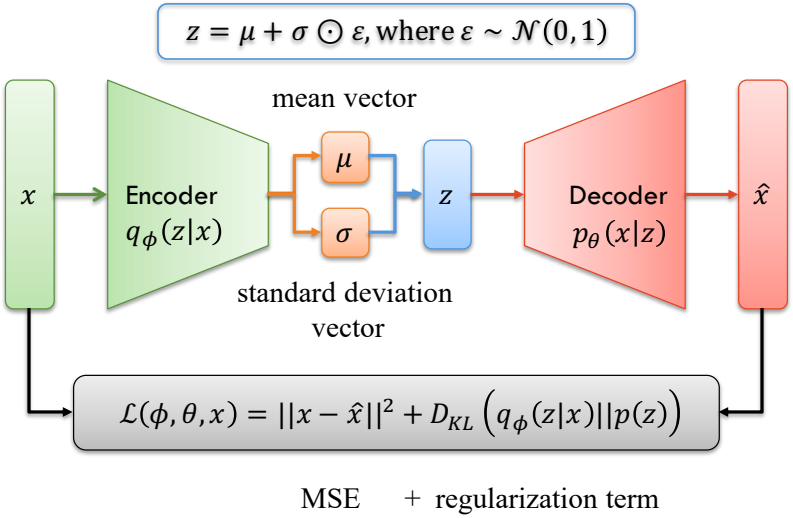


torch_geometric.nn.VGAE

The Variational Graph Auto-Encoder model from the "Variational Graph Auto-Encoders" paper.

Parameters:

- **encoder:** The encoder module to compute μ and σ^2. We use GCNs as the encoder.
- **decoder:** The decoder module for reconstruction. We use the inner product decoder.

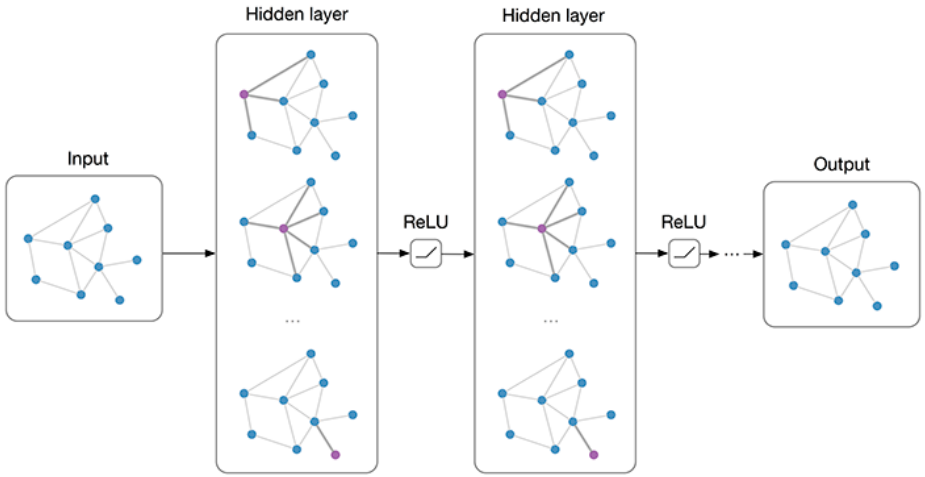


Encoder: torch_geometric.nn.GCNConv

The graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper.

Parameters:

- **in_channels:** Size of each input sample, or -1 to derive the size from the first input(s) to the forward method.
- **out_channels:** Size of each output sample.



Decoder: torch_geometric.nn.InnerProductDecoder

The inner product decoder from the “Variational Graph Auto-Encoders” paper.

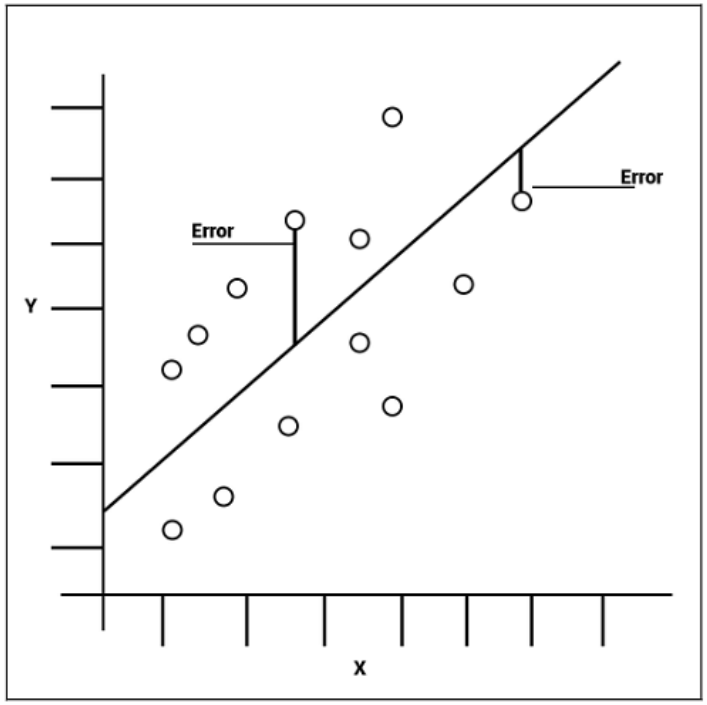
Parameters: No parameters required.

3. Optimization and Checkpoints

Optimization and Checkpoints

- torch.nn.functional.mse_loss
- torch.optim.Adam
- torch.optim.lr_scheduler.CosineAnnealingLR
- torch.save
- torch.load

torch.nn.functional.mse_loss



Measures the element-wise mean squared error. This is our loss function for the regression task.

The mean squared error error loss function is defined as:

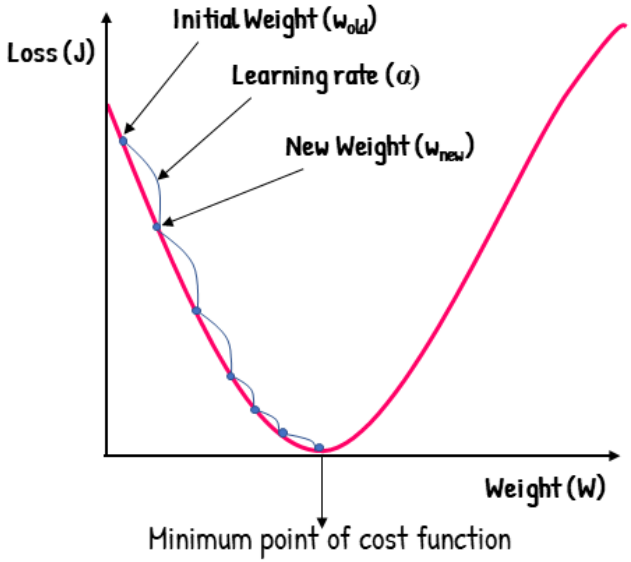
$$\text{MSE}(\tilde{y}_i, y_i) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 .$$

Parameters:

- **input:** The output of our model ($\tilde{y}_i$).
- **target:** The ground-truth value to be learned (y_i).

torch.optim.Adam

The Adam algorithm to optimize neural networks.



Parameters:

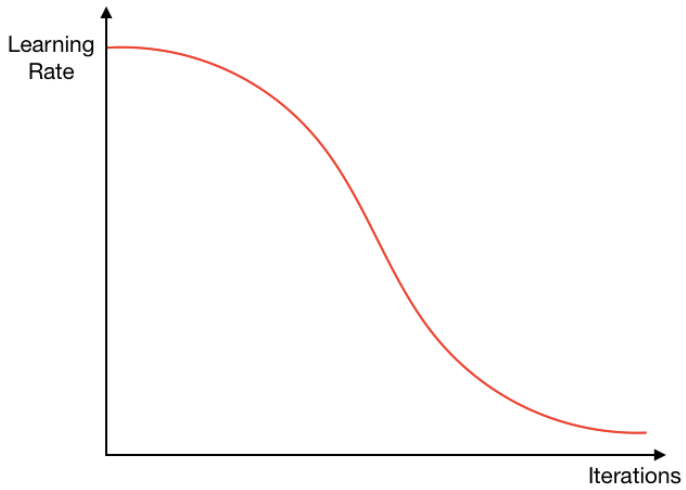
- **params:** An iterable object of parameters to optimize or dicts defining parameter groups.
- **lr:** The learning rate.
- **betas:** Coefficients used for computing running averages of gradient and its square. We use the default value: (0.9, 0.999).
- **eps:** Term added to the denominator to improve numerical stability. We use the default default value: 1e-8.

Example:

```
optimizer = torch.optim.Adam(
    params=model.parameters(), lr=0.001,
    betas=(0.9, 0.999), eps=1e-8,
)
```

torch.optim.lr_scheduler.CosineAnnealingLR

Dynamically adjust the learning rate of optimizer at each epoch using a cosine annealing schedule.



Parameters:

- **optimizer:** The optimizer to be adjusted.
- **T_max:** The maximum number of epochs.

Example:

```
scheduler = CosineAnnealingLR(optimizer=optimizer, T_max=100)

for epoch in range(100):
    ### other code ###
    scheduler.step()
    ### other code ###
```

torch.save and torch.load

- torch.save
  - Saves an object to a disk file.
  - Parameters:
    - **obj:** the object to be saved.
    - **f:** a string containing a file name.
  - Example:

```
torch.save(model.state_dict(), 'model-weights.pt')
```

- torch.load
  - Loads an object saved with torch.save from a file.
  - Parameters:
    - **f:** a string containing a file name.
  - Example:

```
model.load_state_dict(torch.load('model-weights.pt'))
```

Thank you.

THE UNIVERSITY OF
SYDNEY