

## Plan of Action

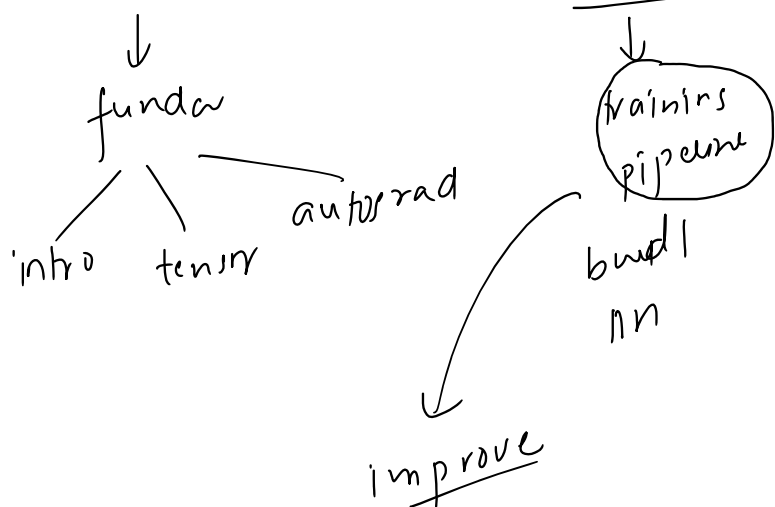
03 December 2024 19:05

1. Revision
2. Improvements
3. The nn module
4. The torch.optim module

build

④


③



# Improvements

03 December 2024

19:11

- 
1. Building the neural network using nn module
  2. Using built-in activation function
  3. Using built-in loss function
  4. Using built-in optimizer

# The nn module

03 December 2024 19:14

The `torch.nn` module in PyTorch is a core library that provides a wide array of classes and functions designed to help developers build neural networks efficiently and effectively. It abstracts the complexity of creating and training neural networks by offering pre-built layers, loss functions, activation functions, and other utilities, enabling you to focus on designing and experimenting with model architectures.

## Key Components of torch.nn:

### 1. Modules (Layers):

- nn.Module: The base class for all neural network modules. Your custom models and layers should subclass this class.
- Common Layers: Includes layers like `nn.Linear` (fully connected layer), `nn.Conv2d` (convolutional layer), `nn.LSTM` (recurrent layer), and many others.

### 2. Activation Functions:

- Functions like `nn.ReLU`, `nn.Sigmoid`, and `nn.Tanh` introduce non-linearities to the model, allowing it to learn complex patterns.

### 3. Loss Functions:

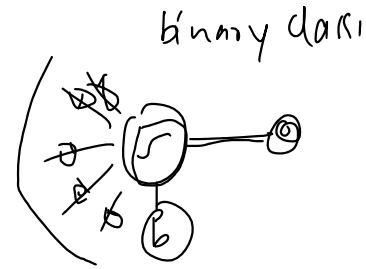
- Provides loss functions such as `nn.CrossEntropyLoss`, `nn.MSELoss`, and `nn.NLLLoss` to quantify the difference between the model's predictions and the actual targets.

### 4. Container Modules:

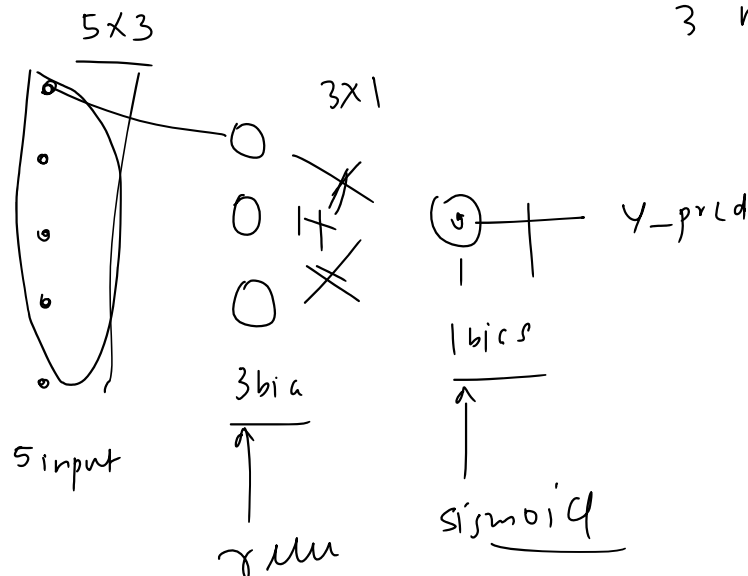
- `nn.Sequential`: A sequential container to stack layers in order.

### 5. Regularization and Dropout:

- Layers like `nn.Dropout` and `nn.BatchNorm2d` help prevent overfitting and improve the model's ability to generalize to new data.



$$z = wx + b$$



# Improved Code v1

03 December 2024

19:16

# The torch.optim module

03 December 2024 19:16

**torch.optim** is a module in PyTorch that provides a variety of optimization algorithms used to update the parameters of your model during training.

It includes common optimizers like Stochastic Gradient Descent (SGD), Adam, RMSprop, and more.

It handles weight updates efficiently, including additional features like learning rate scheduling and weight decay (regularization).

The `model.parameters()` method in PyTorch retrieves an iterator over all the trainable parameters (weights and biases) in a model. These parameters are instances of `torch.nn.Parameter` and include:

- **Weights:** The weight matrices of layers like `nn.Linear`, `nn.Conv2d`, etc.
- **Biases:** The bias terms of layers (if they exist).

The optimizer uses these parameters to compute gradients and update them during training.

model →  
↓  
model.parameters()

# Improved Code v2

03 December 2024

19:16