Practice 3: Model Optimization and HyperParameter Tuning

Outline

- Python Operators
  - Activation Functions
  - Normalization
  - Dropout
- Designing Training Procedures

PyTorch Operators and Layers

PyTorch Operators/Layers

- Activation Functions
- Normalization
- Dropout

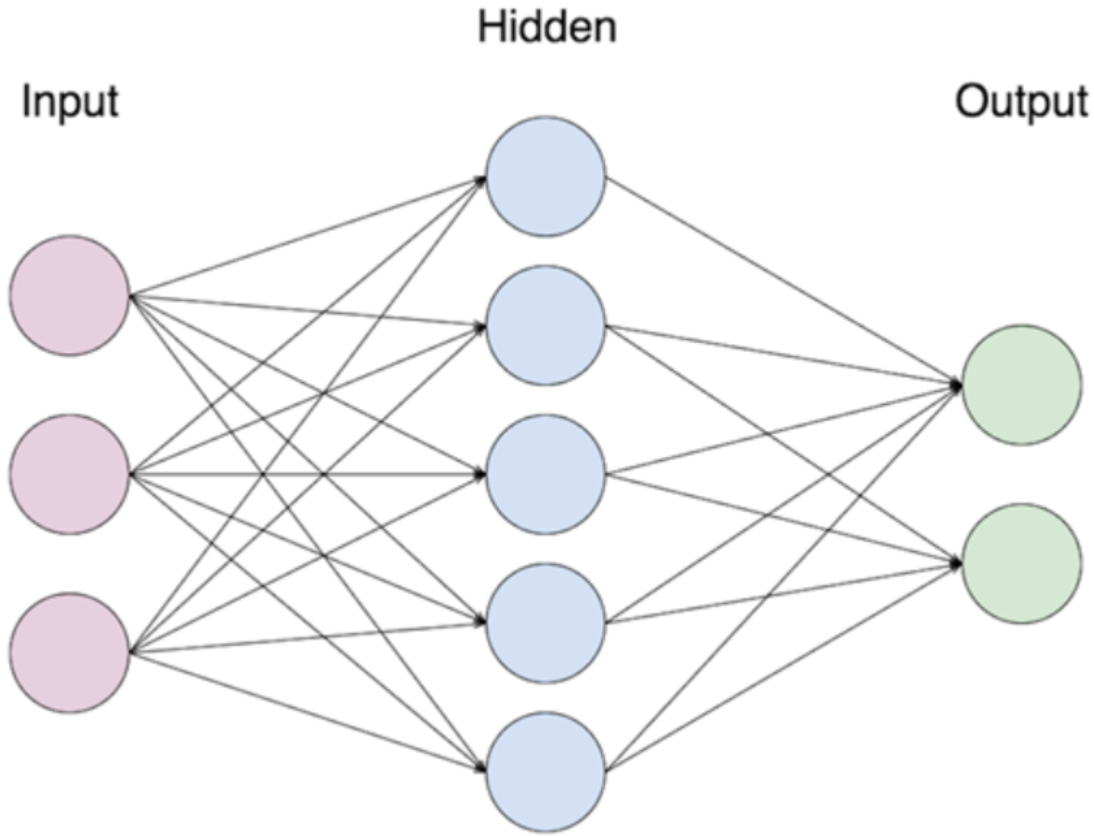


Image source: https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e

PyTorch Operators/Layers

- Activation Functions
- Normalization
- Dropout
- Initialization

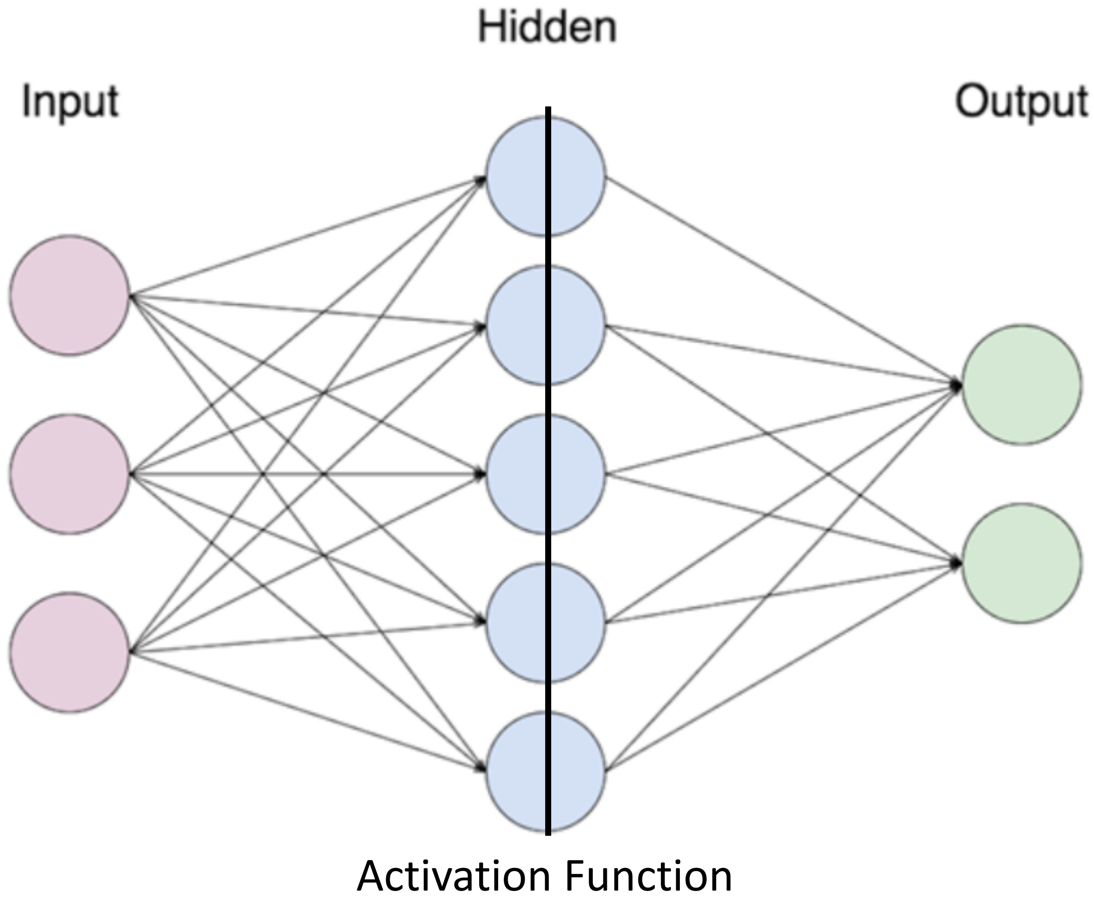
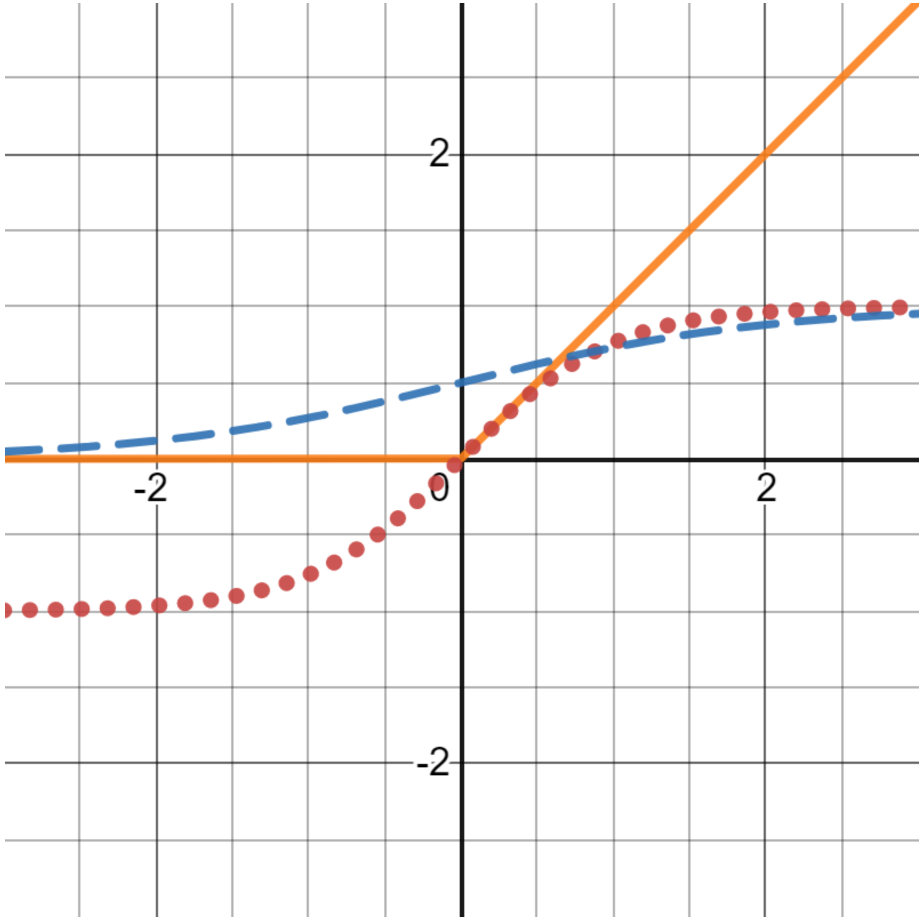


Image source: https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e

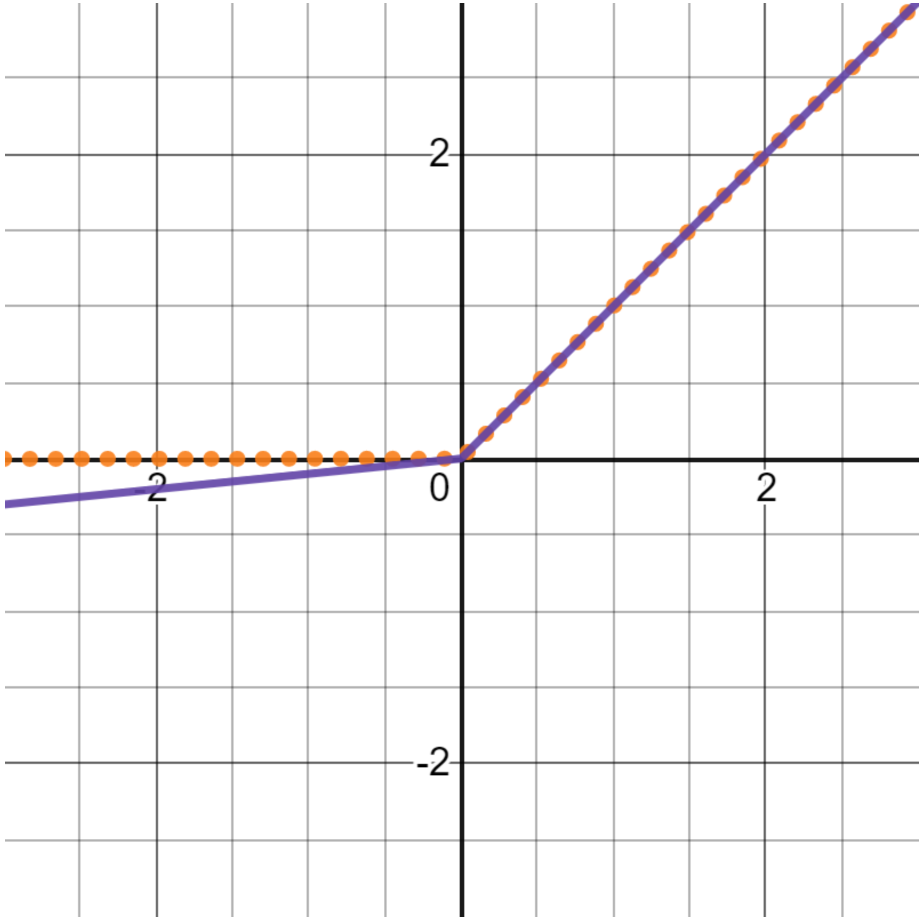
Activation Functions

- Non-linear functions performed by neurons
- ReLU - Rectified Linear Unit (nn.ReLU)
  - $y \geq 0$
- Tanh (nn.tanh)
  - -1<y<1
  - nn.Tanh
- Sigmoid (nn.Sigmoid)
  - 0<y<1



Activation Functions

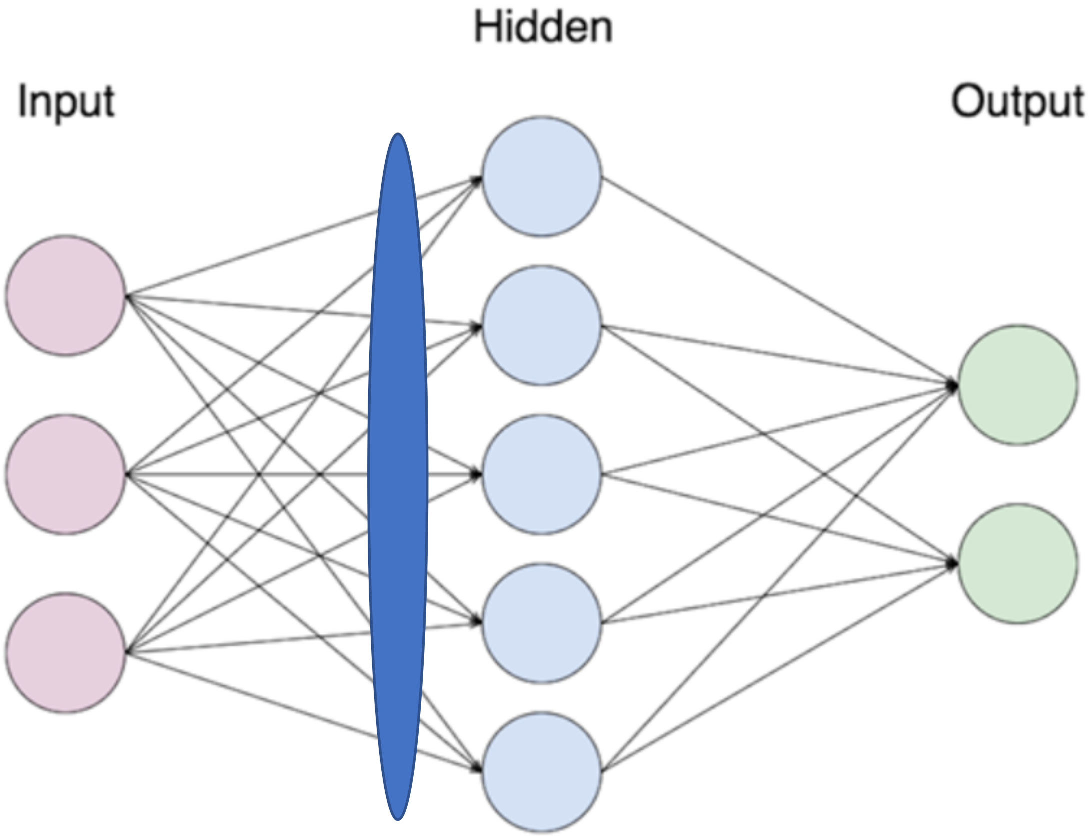
- Leaky ReLU
  - Similar to ReLU, but has non-zero values for negative x
  - Takes argument *negative_slope*, which determines the slope for x<0.
- For full list of activation functions, see: https://pytorch.org/docs/stable/nn.html



Leaky ReLU with negative slope = 0.1

Pytorch Operators/Layers

- Activation Functions
- Normalization
- Dropout



Normalization

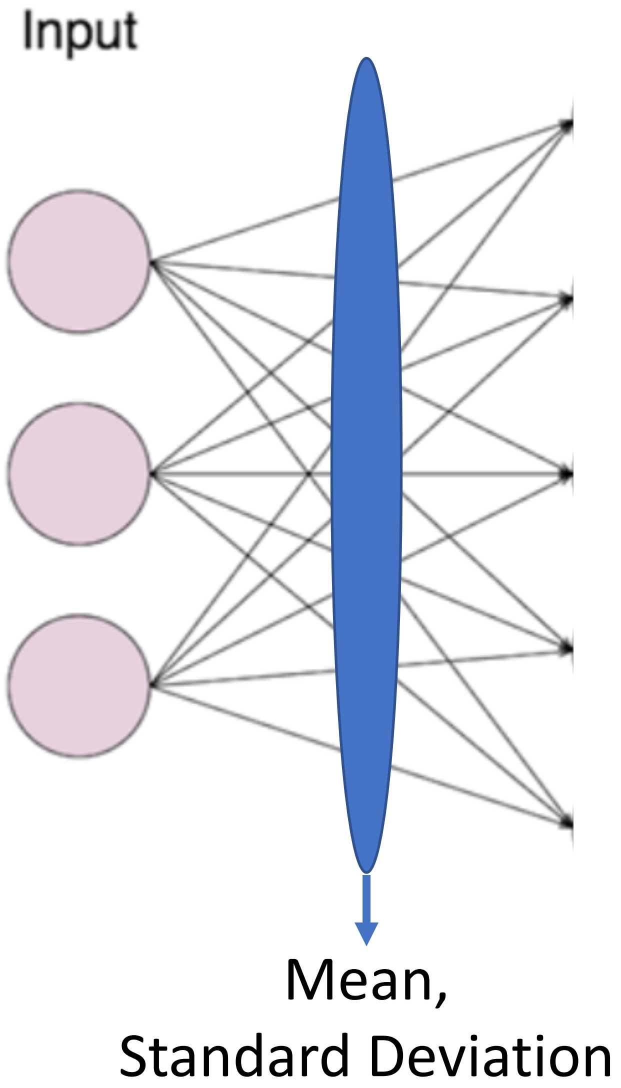
Image source: https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e

Input Normalization: Batch Normalization

- Normalizes input into each layer for each training mini-batch
- Addresses issue of shifting input distributions over training
- Inputs:
  - num_features: Number of features in the input vector
  - eps: numerical stability parameter

Syntax Example:

```
self.bn2 = torch.nn.BatchNorm1D(input_dim)
self.layer2 = torch.nn.Linear(input_dim,output_dim)
```



Input Normalization

Other normalization procedures include:

- Layer Norm: Transposes Batch Norm. Normalizes over all summed inputs to a layer
  - https://arxiv.org/abs/1607.06450
- Group Norm: Normalizes by grouped channels instead of batches
  - https://arxiv.org/abs/1803.08494

Pytorch Operators/Layers – Dropout Regularization

- Activation Functions
- Normalization
- Dropout

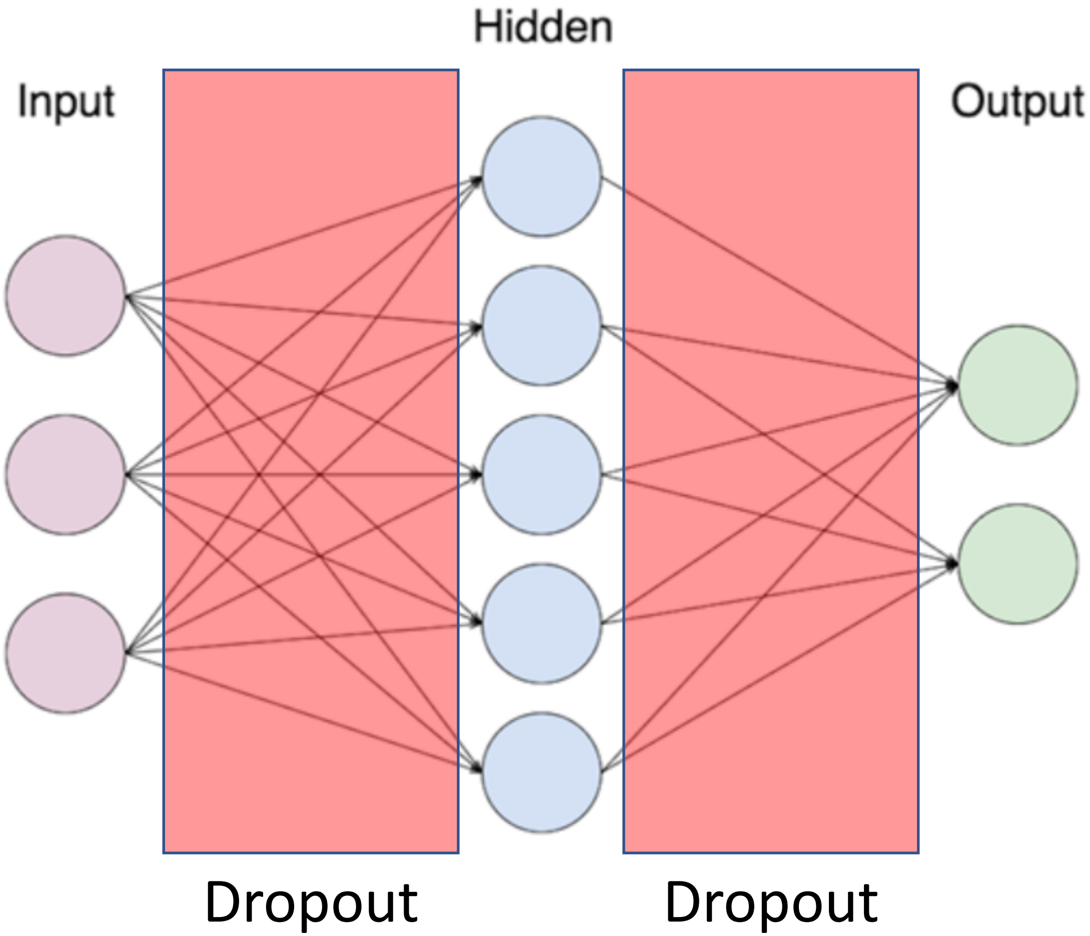


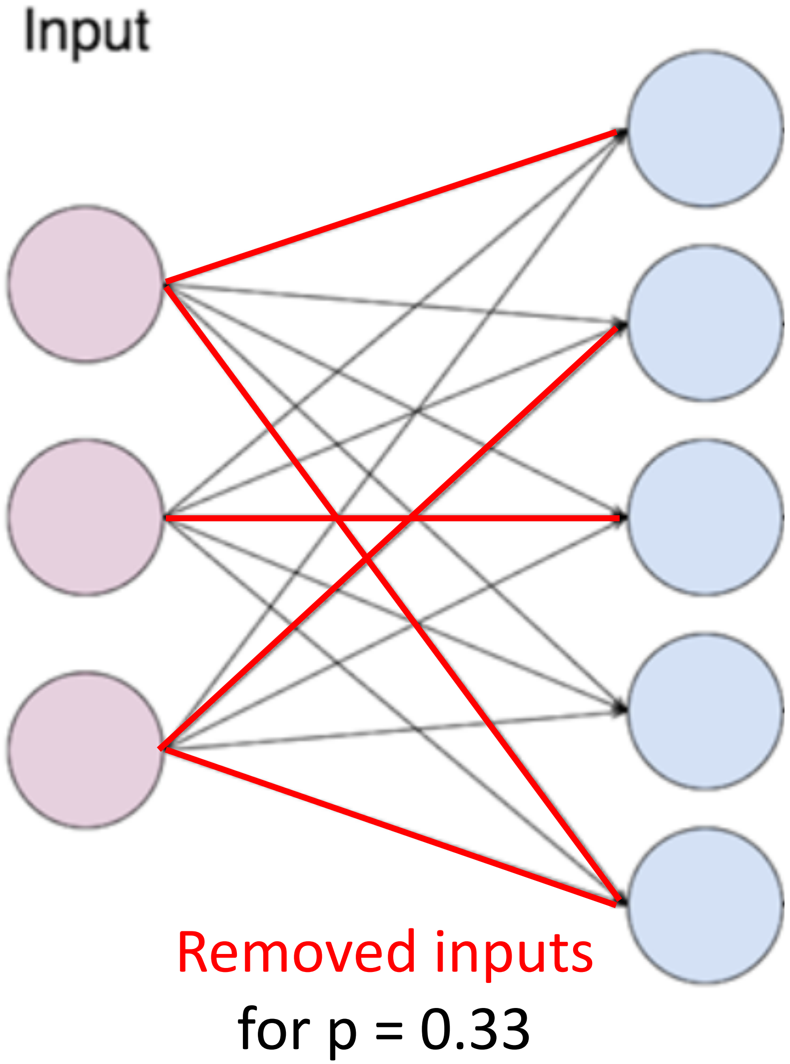
Image source: https://towardsdatascience.com/step-by-step-guide-to-building-your-own-neural-network-from-scratch-df64b1c5ab6e

Dropout

- Randomly zeroes some elements of input tensor with probability *p*
- Effective technique for regularization
- Outputs scaled by 1/1-p
- Treated as identity during evaluation

Syntax Example:

```
self.dp2 = torch.nn.Dropout(p=0.33)
self.layer2 = torch.nn.Linear(input_dim,output_dim)
```

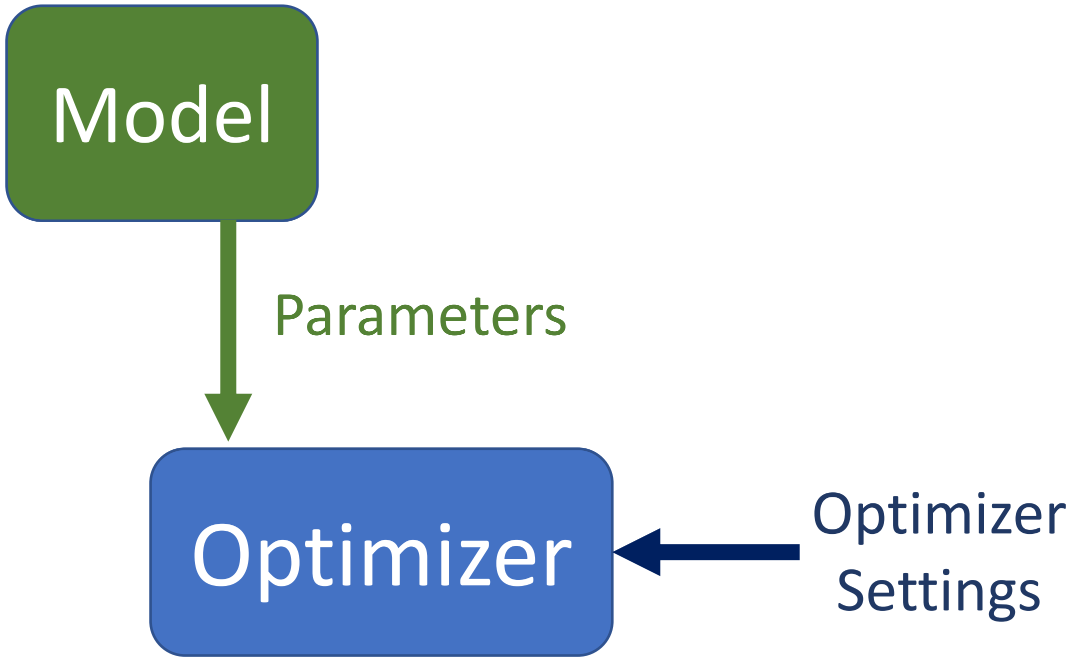


Designing Training Procedures

- Optimizer Setup
- Weight Initialization

Optimizer Initialization

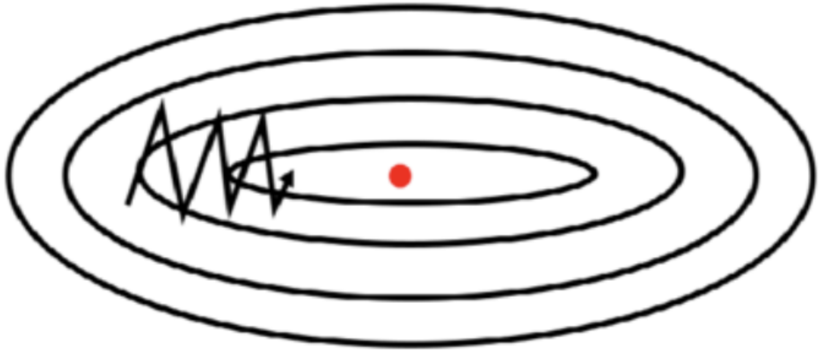
- Parameters:
  - Should be iterable containing parameters to optimize
  - E.g., model.parameters() or [var1, var2]
  - Parameters must be defined BEFORE the optimizer
- Optimizer Settings
  - Learning rate, weight decay, etc.



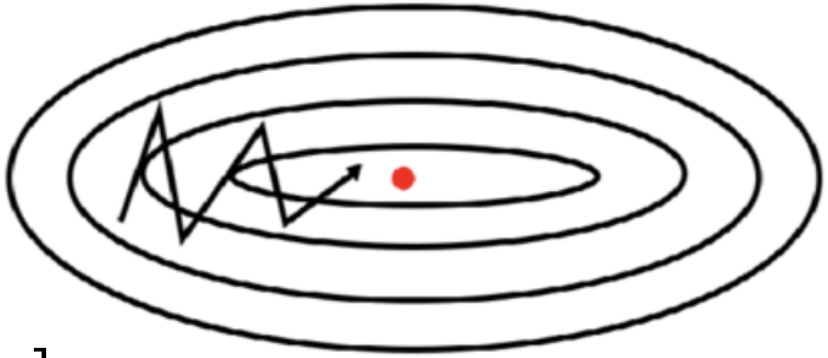
Optimizers

- Stochastic Gradient Descent (`torch.optim.SGD`)
  - params: Model parameters
  - lr: Learning rate (required)
  - momentum: momentum factor (default: 0)
  - weight_decay: (default: 0)

SGD without momentum



SGD with momentum



Example: `torch.optim.SGD(model.parameters(), lr = 0.001, momentum = 0.2, weight_decay = 0.1)`

Optimizers

- Adam (`torch.optim.Adam`)
  - params: Model parameters
  - lr: Learning rate (default: 0.001)
  - betas: coefficients (tuple) used for computing running averages of gradient and its square (default: (0.9, 0.999))
  - eps: term added to denominator to improve numerical stability (default: 1e-8)
  - weight_decay: (default: 0)

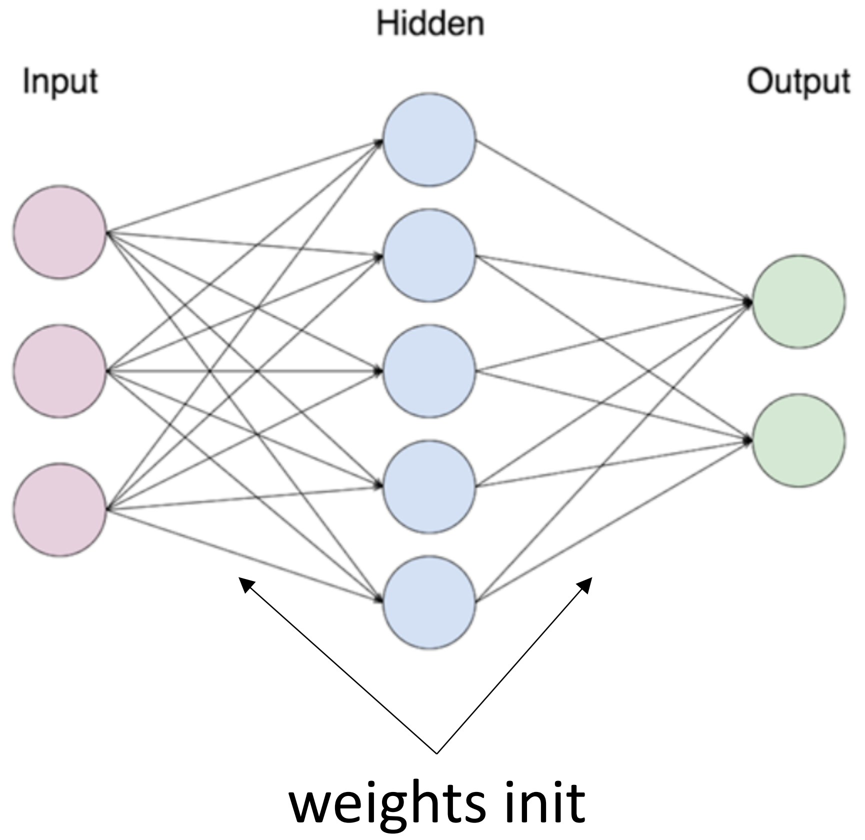
Example:

```
torch.optim.Adam(model.parameters(),
lr = 0.01, betas = (0.95, 0.998),
eps = 1e-7)
```

Other Common Optimizers

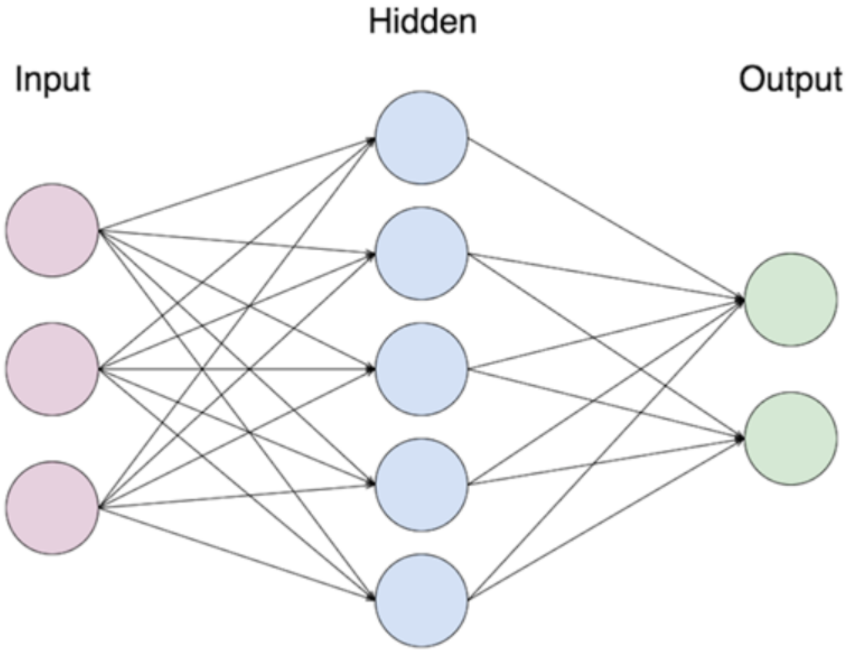
- AdaDelta (`torch.optim.Adadelta`)
  - Precursor to Adam which uses first-order estimates to adapt learning rate
- Adamax (`torch.optim.Adamax`)
  - Variant on Adam based on infinity norm
- RMSProp (`torch.optim.RMSprop`)
  - Take the square root of the gradient average before adding epsilon to normalization of LR

Pytorch Operators/Layers - Weight Initialization



Weight Initialization

- Initializing weights from various distributions plays a role in training. Some frequently used initialization procedures are:
  - Normal distribution
  - Xavier - compatible w. tanh
  - Kaiming (He) – compatible w. relu



weights init

Syntax Example (initialize weights of all FCN layers):

```
# define a function to init weights (here xavier_uniform)
def init_weights(m):
        if type(m) == nn.Linear:
                torch.nn.init.xavier_uniform(m.weight)

# initialize the weights once model is created
myFCNmodel = myFCN(params)
myFCNmodel.apply(init_ weights)
```