Category	Hyperparameter	**What is it**	**Why is it important**	**How to optimize**	**PyTorch code**
Optimization	**Learning rate**	The size of the steps taken during optimization	Controls how fast or slow a model learns. Too high might miss minima, too low might take too long.	Often found through trial and error, grid search, or algorithms like learning rate annealing/scheduling, or adaptive learning methods (like in Adam optimizer). **Not typically left at default, as it's critical for model performance.**	`optimizer = optim.Adam(model.parameters(), lr=0.01)`
	Momentum	The amount of "velocity" carried from previous gradients during optimization	Helps accelerate gradients vectors in the right directions, thus leading to faster converging	**Often kept at a default** (like 0.9) for optimizers like SGD with momentum.	`optimizer = optim.Adam(model.parameters(), lr=0.01, momentum=0.9)`
	Optimizer	The method used to update the model's weights based on the data's gradients	Different optimizers may converge faster and more reliably depending on the task.	Choice is based on the specific task, the size of the data, or **historical precedence**. Adam is very popular due to its performance across a wide range of tasks.	`optimizer = optim.Adam(model.parameters(), lr=0.01, momentum=0.9)`
Training configuration	**Number of epochs**	Number of times the entire training dataset is passed forward and backward through the neural network	Determines how long the model trains, which affects the learning detail and potential for overfitting.	Set based on when the model stops improving on a validation set, **often using techniques like early stopping rather than fixing a number upfront**.	`for epoch in range(num_epochs): # training loop...`
	Batch size	The number of training samples used in one iteration to update the model's weights	Larger batches provide more accurate gradient estimates but require more memory and often lead to poorer generalization, while smaller batches use less memory and can generalize better but may lead to less stable convergence.	It's often chosen based on memory limitations and **empirically** set by trying several different values.	`train_loader = DataLoader(dataset=dataset, batch_size=batch_size, shuffle=True)`
Network architecture	**Number of hidden layers and units**	The layers within the neural network that perform computations and hold weights	They capture features and complexities within the data. Too many can lead to overfitting.	Generally found through experimentation, **historical precedence**, or sometimes heuristics.	`self.fc1 = nn.Linear(4, 12)`
	Activation functions	Non-linear functions that decide if a neuron should be activated	They introduce non-linearity, helping NNs learn complex patterns that linear models cannot.	Often chosen based on **historical success** (e.g., ReLU for hidden layers, softmax for output layers in classification tasks). **Seldom tuned intensively** unless in research or highly specialized applications.	`x = torch.relu(self.fc1(x))`
Regularization	**Dropout rate**	The probability of temporarily removing a neuron from the network during training	Prevents over-reliance on any one neuron, promoting robustness and reducing overfitting.	Typically found via experimentation, as it can vary significantly based on the specific network architecture and problem. **It's not usually left at a default value.**	`self.dropout = nn.Dropout(p=0.5)`
	L1/L2 regularization	Additional term in the loss function that penalizes certain parameter values to prevent the coefficients from overfitting	Adds penalties on weight size, encouraging simpler models that are less likely to overfit.	The strength of regularization (lambda) is often determined through cross-validation and grid search. It's problem-specific and thus **not left at default.**	`optimizer = optim.Adam(model.parameters(), lr=0.01, momentum=0.9, weight_decay=1e-5) #L2`