

# Training and Generalization

- Overfitting, Underfitting, and Goldilocks Fitting
- Training, Validation, and Testing Data Sets
- Model Order, Model Capacity, Generalization Loss

# Training and Generalization

- Goal:

- Learn the “true relationship” from training data pairs  $(x_k, y_k) |_{k=0}^{K-1}$ .

$$x = f_{\theta}(y) + \text{error}$$

- What we learn needs to *generalize* beyond the training data.

- Key parameters:

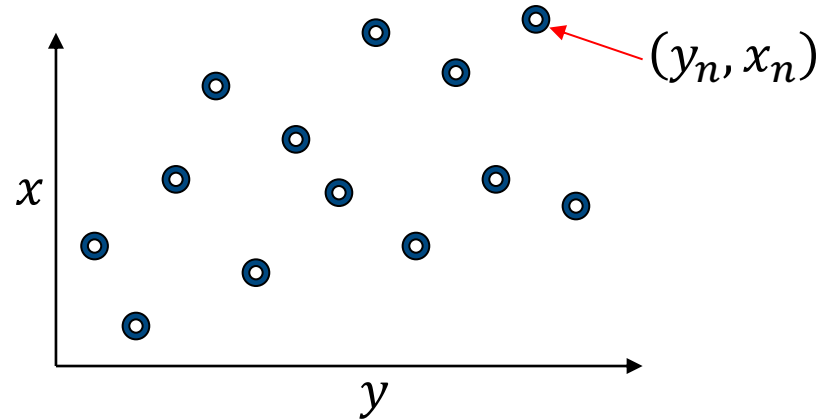
- $P = \underline{\text{Model Order}} = \text{number of parameters} = \text{Dimension of } \theta \in \mathfrak{R}^P$
- $N_x \times K = \# \text{ training points} = (\text{Dimension of } x) \times (\# \text{ of training pairs})$

- Key issues

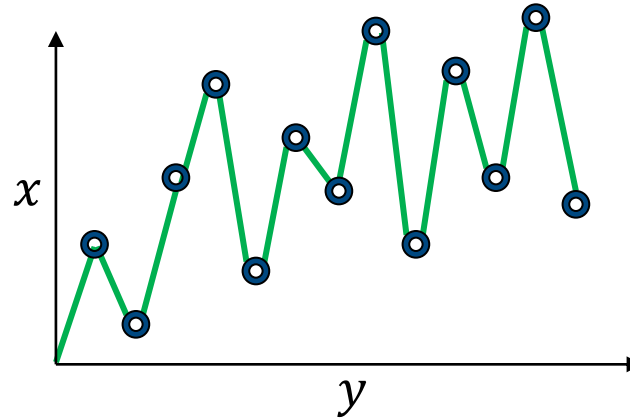
- If  $P \gg N_x \times K$ : Model order is too high and there is a tendency to over fit.
- If  $P \ll N_x \times K$ : Model order is too low, and there is a tendency to under fit.

# Overfitting

- Training data



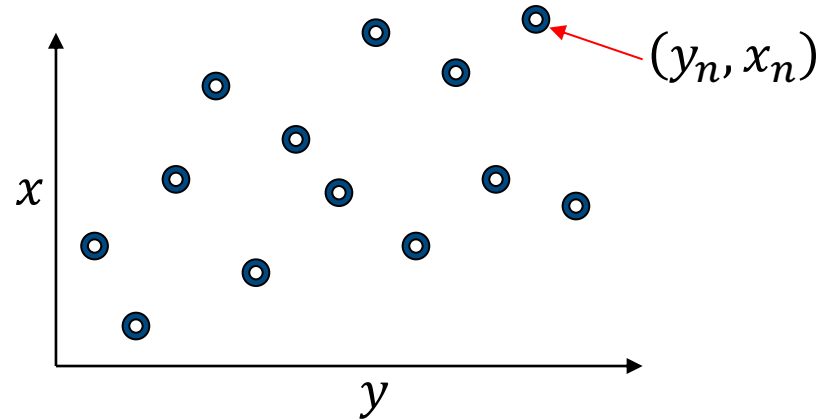
- Overfitting



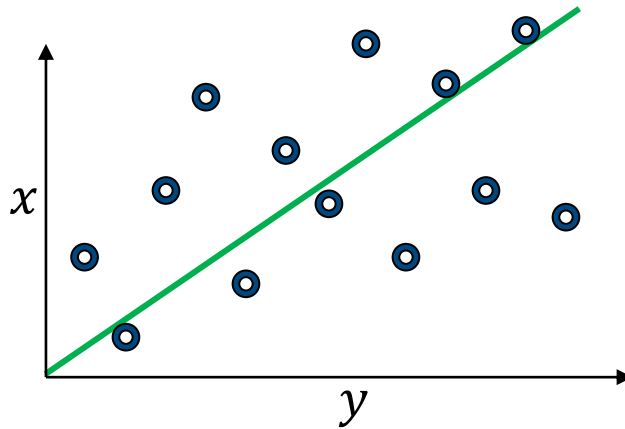
- Model order too high
- Doesn't generalize well

# Underfitting

- Training data



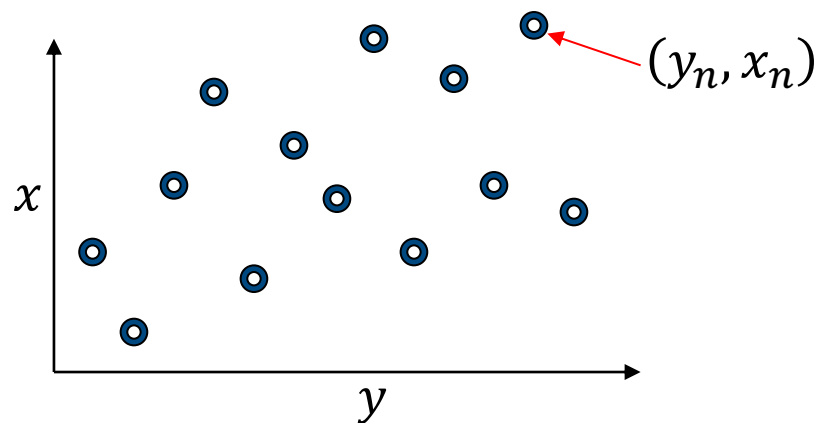
- Underfitting



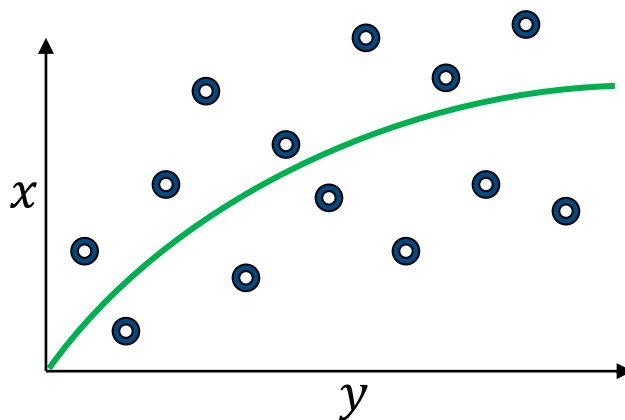
- Model order too low
- Doesn't generalize well

# Goldilocks Fitting

- Training data



- Best fitting



- Model order “just right”
- Best generalization

# Partitioning of Labeled Data

- Let  $(x_k, y_k)$  for  $k \in S = \{0, \dots, K - 1\}$  be the full set data.
  - $y_k$  is the input data.
  - $x_k$  is the label or “ground truth” data.
- Typically, we randomly partition\* the data into three subsets:
  - $S_T$  is the training data
  - $S_V$  is the validation data
  - $S_E$  is the testing (evaluation) data

\* Note that “partition” means  $S = S_T \cup S_V \cup S_E$  and  $\emptyset = S_T \cap S_V = S_T \cap S_E = S_V \cap S_E$
- For each partition, we define a loss function:

$$L_T(\theta) = \frac{1}{|S_T|} \sum_{k \in S_T} \|x_k - f_\theta(y_k)\|^2$$

$$L_V(\theta) = \frac{1}{|S_V|} \sum_{k \in S_V} \|x_k - f_\theta(y_k)\|^2$$

$$L_E(\theta) = \frac{1}{|S_E|} \sum_{k \in S_E} \|x_k - f_\theta(y_k)\|^2$$

# Roles of Data

- Training data:

- Only data used to train model

$$\begin{aligned}\theta^* &= \arg \min_{\theta} \{L_T(\theta)\} \\ &= \arg \min_{\theta} \left\{ \frac{1}{K} \sum_{k \in S_T} \|x_k - f_{\theta}(y_k)\|^2 \right\}\end{aligned}$$

- Validation data:

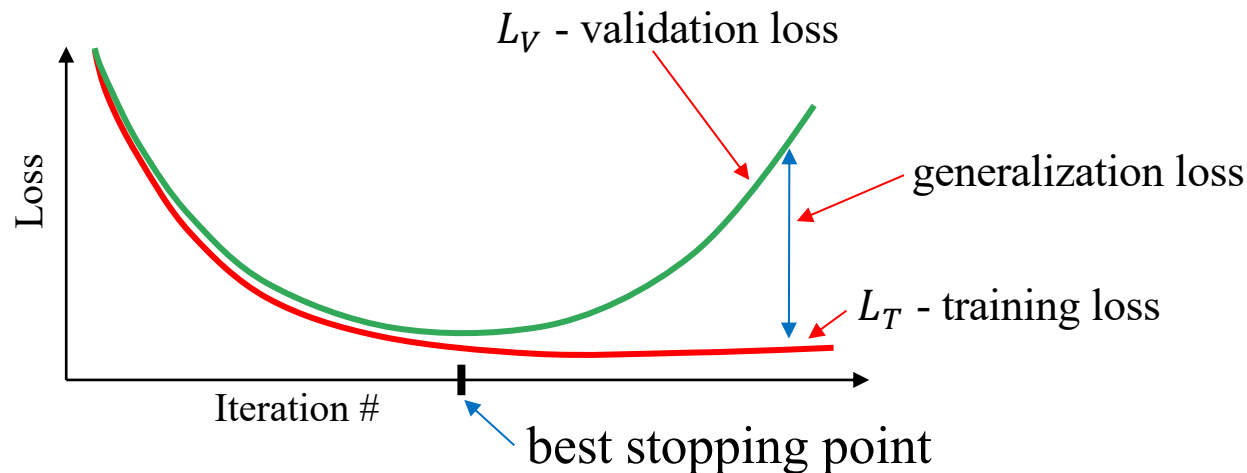
- Used to compare models of different order.

- Testing data

- Used for final evaluation of model performance.

# Loss Function Convergence

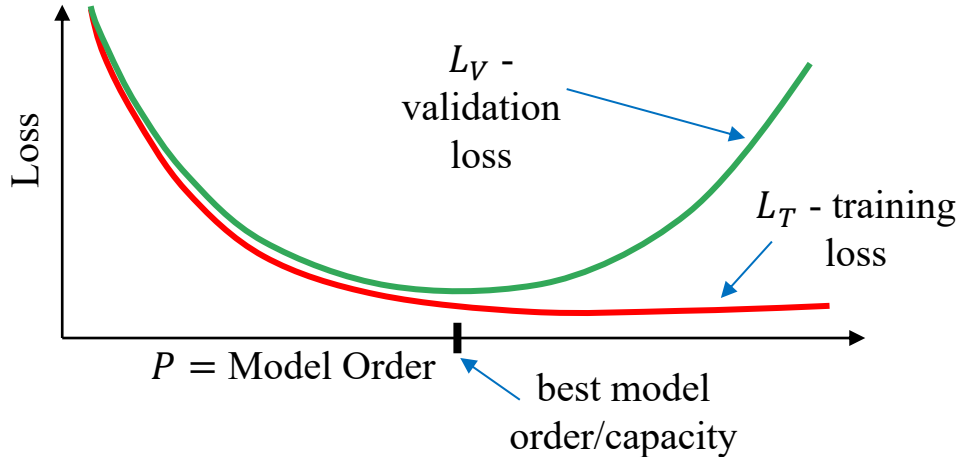
- Loss vs. iterations of gradient-based optimization



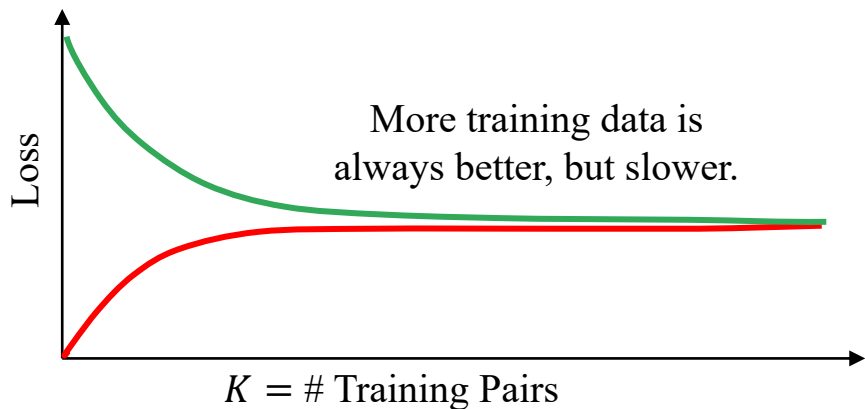
- Notice:
  - As training continues, the model is overfit to the data
  - Best to stop training when  $L_V$  is at a minimum
  - Model order is too high, but early termination of training can help fix problem

# Loss vs. Model Order vs. # Training Pairs

- Loss vs. model order

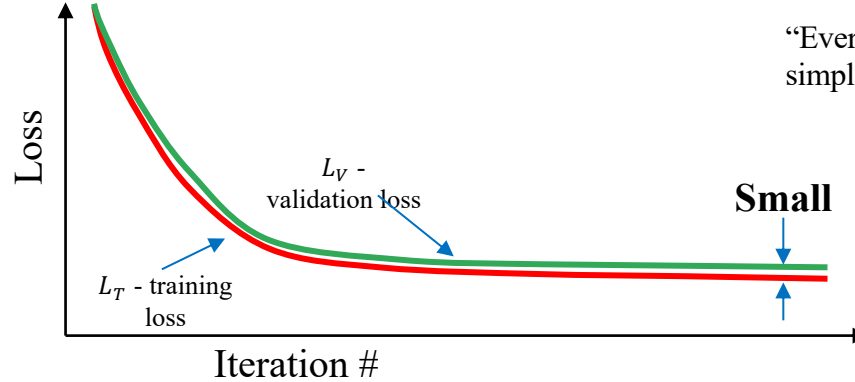


- Loss vs. # of training pairs



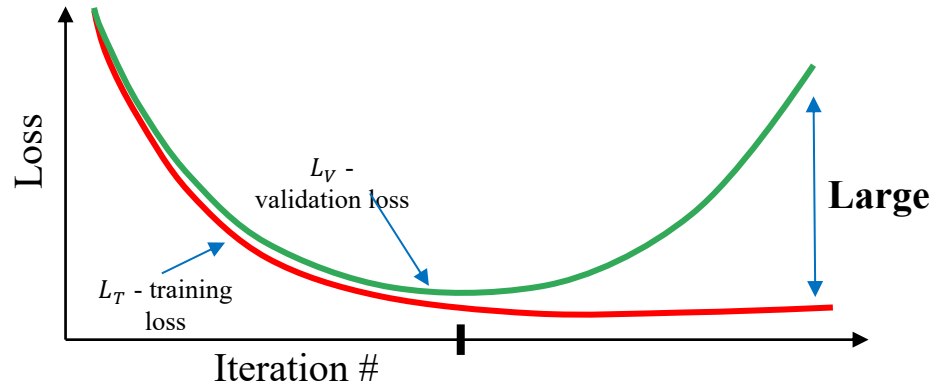
# What are $L_T$ and $L_V$ telling you?

- Model order/model capacity may be too low...



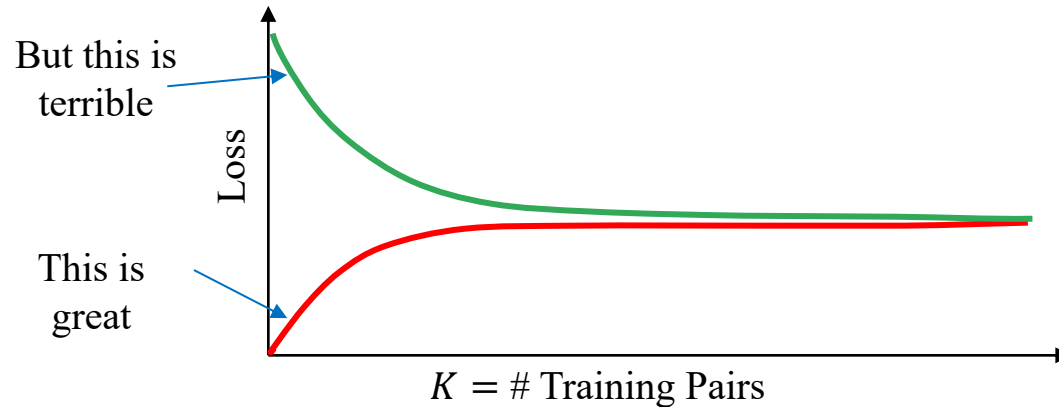
“Everything should be made as simple as possible, but no simpler,”  
-Inspired by Albert Einstein

- Model order/model capacity may be too high...



# Never Test on Training Data!

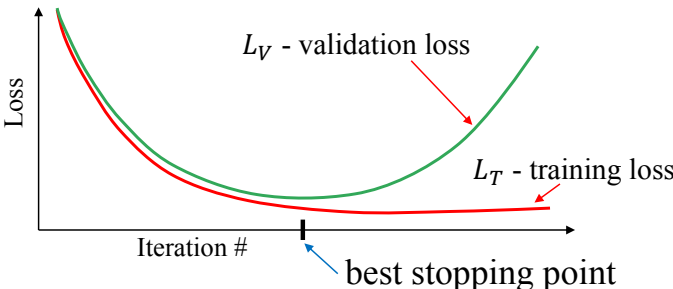
- Never report training loss,  $L_T$ , as your ML system accuracy!



- This is like doing a homework problem after you have seen the solution.
- The network has “memorized” the answers.
- Don't ever report validation loss,  $L_V$ , as your ML system accuracy.
  - This is also biased by the fact that your tuned model order parameters.
- Only report testing loss,  $L_E$ , as your ML system accuracy.
  - This data is sequestered to ensure it is an unbiased estimate of loss.

# Solutions to Parameter Overfitting

## 1. Early termination



## 2. Regularization

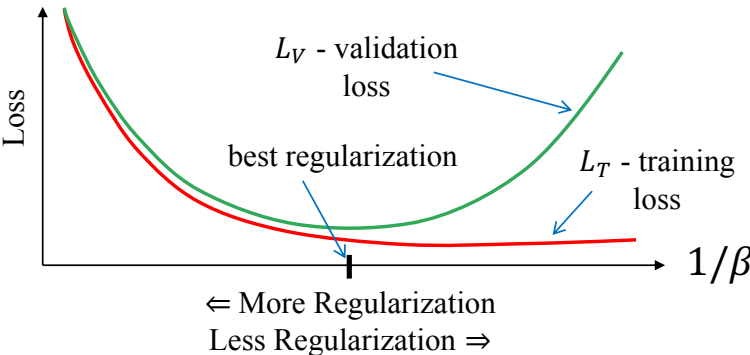
- $L_2$  and  $L_1$  weight regularization
- Loss function is modified to be

$$\tilde{L}(\theta) = L(\theta) + \beta S(\theta)$$

- $\beta$  larger  $\Rightarrow$  less overfitting

$L_2$  norm -  $S(\theta) = \|\theta\|^2$

$L_1$  norm -  $S(\theta) = \|\theta\|_1$



## 3. Dropout Method: Next slide

# Regularization and Dropout

- Weight Regularization and Initialization
- Dropout Methods

# Regularized Maximum Likelihood

- Regularize ML estimate:

$$\hat{\theta} = \arg \min_{\theta} \{-\log p_{\theta}(x, y) + \beta S(\theta)\}$$

where  $S(\theta)$  is a “regularizing” function, and  $\beta$  is the regularization weight.

Typical choices are

$$S(\theta) = -\log p(\theta) \quad \leftarrow \text{MAP estimate}$$

$$S(\theta) = \|\theta\|^2 \quad \leftarrow \begin{array}{l} \text{Like a Gaussian Prior} \\ \text{Reduces amplitude of weights} \end{array}$$

$$S(\theta) = \|\theta\|_1 \quad \leftarrow \begin{array}{l} \text{Like a Laplacian Prior} \\ \text{Encourages weights to go to zero} \end{array}$$

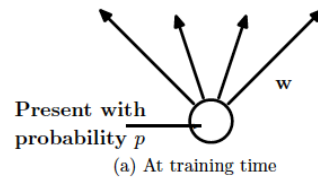
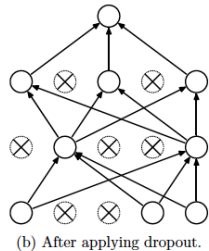
- Modified Loss function

$$\tilde{L}(\theta) = L(\theta) + \beta S(\theta)$$

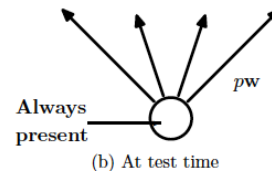
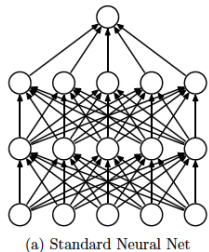
- Can be interpreted as MAP estimate with  $p(\theta) = \frac{1}{z} \exp\left\{-\frac{\beta}{2} S(\theta)\right\}$
- Introduces bias into the estimate of  $\theta$
- Reduces overfitting
- Use regularization if training error  $\gg$  validation error

# The Dropout Method\*

- Drop nodes with probability  $1 - p \approx 0.2$
- Retain nodes with probability  $p \approx 0.8$
- Scale all node outputs by  $p$ :
  - To compute loss for validation and test
  - During inference



During Training: Done independently for each batch



During Validation, Testing, and Inference

\*[Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", vol. 15, no. 56, pp. 1929–1958, 2014.](#)

# Dropout: Training Algorithm

```
For each batch{
  For each layer  $l$  {
     $r^{(l)} \leftarrow \text{Bernoulli}(p, \text{shape}(y^{(l)}))$ 
  }
  For  $n = 0$  to  $K_b - 1$  {
    For each layer  $l$  {
       $\tilde{y}^{(l)} \leftarrow r^{(l)} .* y^{(l)}$ 
       $z^{(l+1)} \leftarrow w^{(l+1)} * \tilde{y}^{(l)}$ 
       $y^{(l+1)} \leftarrow f(z^{(l+1)})$ 
    }
  }
}
```

*Dropout*

*Training*

## ■ Dropouts are:

- Independent for each internal node in the network
- A single set of Bernoulli weights are computed for each batch.

# Dropout: Validation and Testing

```
For each batch{
  For  $n = 0$  to  $K_b - 1$  {
    For each layer  $l$  {
       $\tilde{y}^{(l)} \leftarrow p * y^{(l)}$ 
       $z^{(l+1)} \leftarrow w^{(l+1)} * \tilde{y}^{(l)}$ 
       $y^{(l+1)} \leftarrow f(z^{(l+1)})$ 
    }
  }
}
```

*scaling*

*Training*

- Scale output to account for increased number of nodes

# Dropout: Stochastic Generator

- Dropouts can be used to generate stochastic outputs for generators described later in class.
  - Leave dropouts on during inference
  - Output of DNN is then a random vector

$$\boxed{X} = f_{\theta}(\boxed{y})$$

*Random Vector* →      ← *Deterministic Input*