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

- **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 \(k \in S = \{0, \cdots, K - 1\}\) 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:

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

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

  
  
  \[
  L_E(\theta) = \frac{1}{|S_E|} \sum_{k \in S_E} \|y_k - f_\theta(x_k)\|^2
  \]
Roles of Data

- **Training data:**
  - Only data used to train model

\[ \theta^* = \arg \min_{\theta} \{ L_T(\theta) \} \]

\[ = \arg \min_{\theta} \left\{ \frac{1}{K} \sum_{k \in S_T} \| y_k - f_\theta(x_k) \|^2 \right\} \]

- **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

  \[ \mathcal{L}_V \] - validation loss

  \[ \mathcal{L}_T \] - training loss

  \[ P = \text{Model Order} \]

  best model order/capacity

- Loss vs. # of training pairs

  More training data is always better, but slower.

\[ K = \# \text{Training Pairs} \]
What are $L_T$ and $L_V$ telling you?

- Model order/model capacity may be too low…

![Graph showing training and validation loss for small model capacity.]

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

- Model order/model capacity may be too high…

![Graph showing training and validation loss for large model capacity.]
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
  - \( S(\theta) = -\log p(\theta) \)  
    - MAP estimate
  - \( S(\theta) = \|\theta\|^2 \)  
    - Like a Gaussian Prior
    - Reduces amplitude of weights
  - \( S(\theta) = \|\theta\|_1 \)  
    - Like a Laplacian Prior
    - Encourages weights to go to zero

- 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

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 $l$ {
      $\tilde{y}^{(l)} \leftarrow r^{(l)} \cdot y^{(l)}$
      $z^{(l+1)} \leftarrow w^{(l+1)} \cdot \tilde{y}^{(l)}$
      $y^{(l+1)} \leftarrow f(z^{(l+1)})$
    }
  }
}

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

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

- 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

\[ X = f_\theta(y) \]