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

- **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:

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

$L_V$ - validation loss

$L_T$ - training loss

best stopping point

iteration #
Loss vs. Model Order vs. # Training Pairs

- Loss vs. model order
  - $L_V$ - validation loss
  - $L_T$ - training loss
  - Best model order/capacity

- Loss vs. # of training pairs
  - More training data is always better, but slower.
What are $L_T$ and $L_V$ telling you?

- Model order/model capacity may be too low…

- 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 even 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.
Stochastic Gradient Descent

- Batches and Epochs
- Learning Rate and Momentum
- Weight initialization and regularization
- Dropout method
Batches, Epochs, and Stochastic Gradient Descent (SGD)

- Partition training set into randomized non-intersecting batches

\[ S = \{1, \cdots, K\} = \bigcup_{b=1}^{B} S_b \]

- For each batch you compute a separate gradient

\[ \nabla L(\theta; S_b) \leftarrow \text{Gradient for } b^{th} \text{ batch of training data} \]

One epoch

\[
\begin{aligned}
\text{Repeat until converged } & \{ \\
\text{Repeat } b = 1 \text{ to } B & \{ \\
& d \leftarrow -\nabla L(\theta; S_b) \\
& \theta \leftarrow \theta + \alpha d^t \\
\} \\
\} \\
\text{One batch}
\end{aligned}
\]

Stochastic Gradient Descent (SGD)
Why SGD works?
- The gradient for a small batch is much faster to compute and almost as good as the full gradient.
- If $K = 10,000$ and $K_b = |S_b| = 32$, then one iteration of SGD is approximately $\frac{10,000}{32} \approx 312$ times faster than GD.

Batch size
- Larger batches: less “noise” in gradient ⇒
  - *Worse*: slower updates; more exploration.
  - *Better*: better local convergence.
- Smaller batches: more “noise” in gradient ⇒
  - *Worse*: hunts around local minimum.
  - *Better*: faster updates; better exploration.

Patch size:
- Many algorithms train on image “patches”
- Apocryphal: Smaller patches speed training. Not true!!!!

Step size $\alpha$
- Too large ⇒ hunts around local minimum
- Too small ⇒ slow convergence
Momentum

- SGD with momentum
  - $\alpha$ is step size, and $\gamma$ is momentum typically with $\gamma = 0.9$

Repeat until converged {
  init $m \leftarrow 0$
  Repeat $b = 1$ to $B$ {
    $d \leftarrow -\nabla L(\theta; S_b)$
    $m \leftarrow \gamma m + \alpha d$
    $\theta \leftarrow \theta + m^t$
  }
}

Stochastic Gradient Descent (SGD) with momentum

- Intuition: If $d_n = \delta_n$ is an impulse, then $\theta_n \leftarrow \theta_0 + (\alpha \sum_{i=0}^{n} \gamma^i) d_0^t$

Asymptotic value $\frac{\alpha}{1-\gamma}$
Time constant $\gamma$
Nesterov Momentum*

- SGD with momentum
  - \( \alpha \) is step size, and \( \gamma \) is momentum typically with \( \gamma \approx 0.9 \)

Repeat until converged {
  init \( m \leftarrow 0 \)
  Repeat \( b = 1 \) to \( B \) {
    \[ d \leftarrow -\nabla L(\theta + \gamma v^t; S_b) \]
    \[ m \leftarrow \gamma m + \alpha d \]
    \[ \theta \leftarrow \theta + m^t \]
  }
}

Stochastic Gradient Descent (SGD) with momentum

- Intuition:
  - Even if \( d_n = 0 \), we have that \( \theta_{n+1} = \theta_n + \gamma m^t \) because of momentum
  - So compute the gradient at \( \theta_n + \gamma m^t \)

ADAM (Adaptive Moment Estimation)*

- SGD with ADAM optimization = Momentum + Preconditioning

Repeat until converged {
  init $m \leftarrow 0; v \leftarrow 0$
  init $t \leftarrow 0$
  Repeat $b = 1$ to $B$ {
    $t \leftarrow t + 1$
    $d \leftarrow -\nabla L(\theta; S_b)$
    $m \leftarrow \beta_1 m + (1 - \beta_1) d$
    $v \leftarrow \beta_2 v + (1 - \beta_2) d^2$
    $\hat{m} \leftarrow m/(1 - \beta_1^t)$
    $\hat{v} \leftarrow v/(1 - \beta_2^t)$
    $\theta \leftarrow \theta + \alpha (\sqrt{\hat{v}} - \epsilon)^{-1} \hat{m}_t$
  }
} 

**ADAM Optimization**

- Typical parameters: $\alpha = 0.001; \beta_1 = 0.9; \beta_2 = 0.999; \epsilon = 10^{-8}$

ADAM (Adaptive Moment Estimation)*

- SGD with ADAM optimization = Momentum + Preconditioning

Repeat until converged {  
  init $m \leftarrow 0$; $v \leftarrow 0$  
  init $t \leftarrow 0$  
  Repeat $b = 1$ to $B$ {  
    $t \leftarrow t + 1$  
    $d \leftarrow -\nabla L(\theta; S_b)$  
    $m \leftarrow \beta_1 m + (1 - \beta_1) d$  
    $v \leftarrow \beta_2 v + (1 - \beta_2) d^2$  
    $\hat{m} \leftarrow m / (1 - \beta_1^t)$  
    $\hat{v} \leftarrow v / (1 - \beta_2^t)$  
    $\theta \leftarrow \theta + \alpha (\sqrt{\hat{v}} - \epsilon)^{-1} \hat{m}_t$  
  }  
}

- Typical parameters: $\alpha = 0.001$; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 10^{-8}$

Parameter Overfitting

- 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

- Early termination

- Dropout*: Drop nodes with probably $p \approx 0.2$, then at test scale all outputs by $p$.