Machine Learning (CE 40477)

Fall 2024

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

October 8, 2024

Reviewing the Problem of Overfitting

- **Recall from previous lectures:** Overfitting vs Underfitting
- **Overfitting:** Learning the underlying patterns and noise in the training data.
  - High accuracy on training data but poor generalization on unseen data.
- **Underfitting:** The model is too simple to capture the patterns in the data.
  - Poor performance on both training and test data.

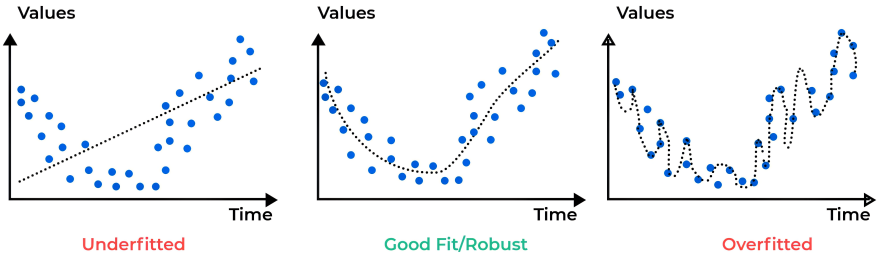

Figure 1: **Source**

Regularization in Neural Networks

- **Goal:** Prevent overfitting.
- **Key Idea:** Favor simpler models to avoid fitting noise in the data.

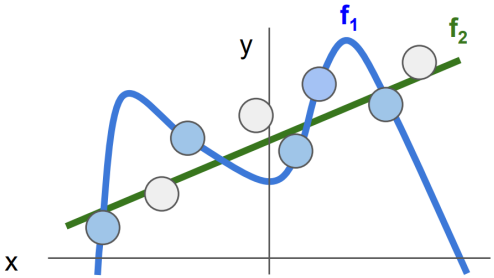

Figure 2: Regularization prevents overfitting by avoiding fitting noise in the data [1].

Regularization in Neural Networks

$$J_\lambda(w) = \underbrace{J(w)}_{\text{Data loss}} + \underbrace{\lambda R(w)}_{\text{Regularization}} \tag{1}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Key Regularization Techniques

- L1 (Lasso): Sparse weights, feature selection

$$\text{L1 Regularization} = \lambda \sum_{j=1}^{m} |w_j| \tag{2}$$

Key Regularization Techniques

- L1 (Lasso): Sparse weights, feature selection

$$\text{L1 Regularization} = \lambda \sum_{j=1}^{m} |w_j| \tag{2}$$

- L2 (Ridge): Adds a penalty for large weights to the loss function

$$\text{L2 Regularization} = \lambda \sum_{j=1}^{m} w_j^2 = \lambda \mathbf{W}^T \mathbf{W} \tag{3}$$

Key Regularization Techniques

- L1 (Lasso): Sparse weights, feature selection

$$\text{L1 Regularization} = \lambda \sum_{j=1}^{m} |w_j| \tag{2}$$

- L2 (Ridge): Adds a penalty for large weights to the loss function

$$\text{L2 Regularization} = \lambda \sum_{j=1}^{m} w_j^2 = \lambda \mathbf{W}^T \mathbf{W} \tag{3}$$

- Elastic Net (L1 + L2): Combines both L1 and L2 penalties, where β controls the balance between L1 and L2 regularization.

$$\text{Elastic net Regularization} = \lambda \sum_{j=1}^{m} (\beta |w_j| + \frac{1-\beta}{2} w_j^2) \tag{4}$$

Key Regularization Techniques

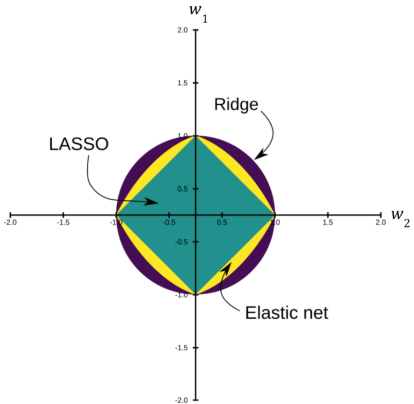

Figure 3: Lasso, Ridge, and Elastic net Comparision. **Source**

Key Regularization Techniques

- Early Stopping
  - Stops training when the validation loss stops improving.
  - Prevents overfitting by avoiding excessive training beyond the optimal point.

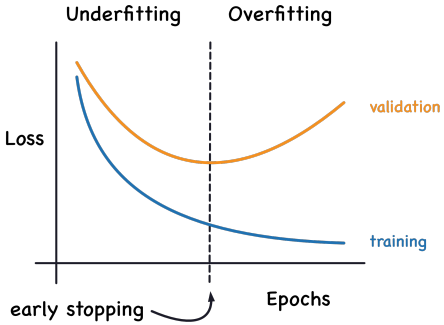

Figure 4: **Source**

Key Regularization Techniques

- Dropout
  - Randomly deactivates some of the neurons **during training**.
  - Reduces neuron dependency and co-adaptation, thereby helping to prevent overfitting.

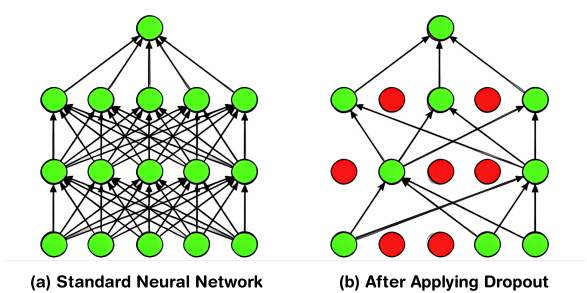

Figure 5: Source

Mechanism of Dropout

- **During Training**
  - A fraction (e.g., 20%-50%) of neurons in each layer is randomly set to zero during each forward pass.
  - This fraction is called the `dropout rate`.
  - The `dropout rate` is a value between 0 and 1. For example, if 30% of neurons are deactivated, the `dropout rate` equals 0.3.
  - Neurons are dropped out independently, and their weights are not updated during backpropagation.
- **During Inference**
  - All neurons are active.
  - But the final weights will be larger than expected!

Mechanism of Dropout

- **Challenge**
  - During training, the network learns with a subset of neurons, leading to different activation patterns.
  - During inference, when all neurons are active, the network's learned weights might cause activations to differ from training due to the sudden increase in neuron count.

Mechanism of Dropout

- **Challenge**
  - During training, the network learns with a subset of neurons, leading to different activation patterns.
  - During inference, when all neurons are active, the network's learned weights might cause activations to differ from training due to the sudden increase in neuron count.
- **Solution**
  - The activations of all neurons during inference will be multiplied by `dropout rate`. This adjustment maintains the expected sum of activations, allowing the model to make predictions comparable to those during training.

Why is Dropout a Good Regularization Idea?

- **Prevents Overfitting**
  - Dropout forces the network to learn more robust features by preventing over-reliance on specific neurons.
  - Randomly deactivating neurons during training reduces the risk of overfitting the training data.
- **Acts as an Ensemble**
  - Dropout effectively trains an ensemble of different sub-networks by sampling different neuron combinations in each iteration.
  - During inference, these sub-networks are combined, resulting in a more generalized model that performs better on unseen data.

Dropout Best Practices

- **Dropout Rate:** Typically 0.5 for hidden layers, but it should be tuned based on the network architecture and problem complexity.
- **Where to Apply:** Commonly used in fully connected layers. For convolutional layers (discussed in the next chapter), spatial dropout is often preferred.
- **Use Sparingly:** Excessive dropout can cause underfitting or overly sparse networks.

Introduction to Hyperparameter Tuning

- **Definition:** Hyperparameters are settings external to the model that control the training process and model capacity. They cannot be learned from the data.
- **Examples:** Learning rate, batch size, number of layers, number of neurons per layer, dropout rate, etc.
- **Impact:** Choosing the right hyperparameters can significantly enhance model performance, while poor choices may cause underfitting or overfitting.

Why Hyperparameter Tuning Matters

- **Optimization Goal:** Identify the hyperparameters that maximize model performance on validation data.
- **Challenges:**
  - Large, high-dimensional search space.
  - High computational cost due to repeated training and evaluation.
  - Noisy objective function – performance can vary due to randomness (e.g., weight initialization).
- **Trade-offs:** Balancing computation time with model performance.

Approaches to Hyperparameter Tuning

- **Grid Search:** An exhaustive search over a predefined set of hyperparameter combinations.
- **Random Search:** Randomly samples hyperparameters, sometimes finding good configurations more efficiently than grid search.

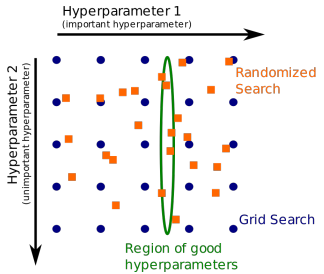

Figure 6: Grid Search vs. Random Search in Hyperparameter Tuning. **Source**

Hyperparameter Tuning Best Practices

1. **Start Simple:** Start by tuning a smaller subset of key hyperparameters (e.g., learning rate, batch size).
2. **Use Cross-Validation:** Instead of a single validation set, use techniques like k-fold cross-validation to ensure performance is not dependent on a specific train-validation split.
3. **Automate:** Leverage frameworks like `scikit-learn` to automate and parallelize the tuning process.

Hyperparameter Tuning Best Practices

4. **Track Experiments:** Use tools like `Wandb` to systematically track hyperparameters, training metrics, and results.
5. **Use Learning Curves:** Plot learning curves to monitor the model's performance over time and detect overfitting or underfitting early.
6. **Budget Time and Resources:** Set limits on tuning (e.g., time, number of iterations) to avoid excessive computational costs.

Monitoring Training Process

- Track key metrics: Loss, accuracy, validation loss.
- Use `wandb` for real-time monitoring and logging.
- Wandb Panels: visualizations to explore your logged data, the relationships between hyperparameters and output metrics, and dataset examples.
- Detect overfitting early by analyzing trends in training and validation metrics.
- Visualize progress using learning curves to assess model performance over time.

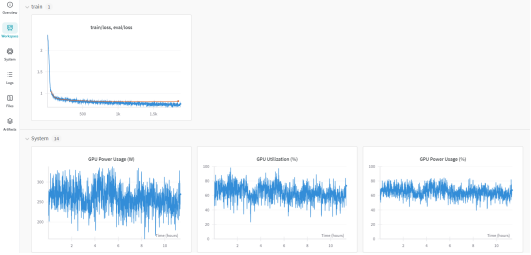

Figure 7: Training metrics visualization in `wandb`.

[1] F.-F. Li and Z. Durante, *CS231n Lectures*.
Stanford University, June 2024.
Updated June 3, 2024.

[2] M. Soleymani Baghshah, “Machine learning.” Lecture slides.

[3] A. Sharifi Zarchi, “Machine learning 2022.” Lecture slides.
Only Introduction-to-NN.pdf is referenced.

Any Questions?