Machine Learning (CE 40477)

Fall 2024

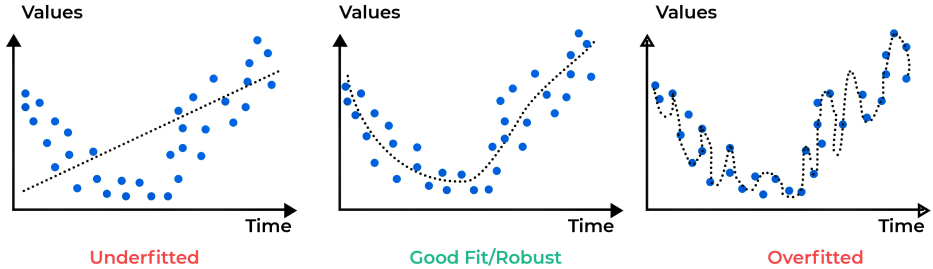
Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

December 1, 2025

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.



Understanding the Bias-Variance Tradeoff

- **Motivation:**
  - Balancing bias and variance is essential for training neural networks that **generalize** well to **unseen data**.
  - In neural networks, model complexity can be increased with more layers or neurons, impacting both bias and variance.
- **Core Concepts:**
  - **Bias:** Systematic error due to overly simplified assumptions.
    - High-bias networks (e.g., shallow networks) may **underfit**, failing to capture complex patterns.
  - **Variance:** Sensitivity to small changes in training data.
    - High-variance networks (e.g., deep networks) may **overfit**, capturing noise in the data.

Visualizing the Bias-Variance Tradeoff

- **Tradeoff in Neural Networks:** Increasing layers and parameters can reduce bias but increase variance.
- **Finding the Balance: Regularization** methods help control this tradeoff by **adjusting model complexity**.

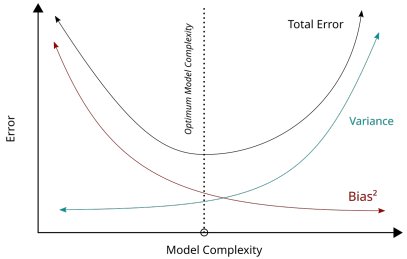


Figure 1: Adapted from Wikipedia

Inductive Bias in Neural Networks

- **Definition:**
  - Inductive bias refers to **assumptions** embedded in the model's design to help it generalize to new, unseen data.
- **Why it Matters:**
  - Classical ML methods rely on explicit assumptions:
    - **Linear models**: assume linear relationships between features and outputs.
    - **k-NN**: assumes nearby points in feature space share similar labels.
    - **Decision trees**: assume data can be separated via axis-aligned splits.
  - Without inductive bias, models would require **enormous data** to generalize effectively.
  - Model choices (e.g., linearity, smoothness, locality) embody inductive biases in classical ML.
- **Classical Example:**
  - *Occam's Razor*: Favoring simpler hypotheses (e.g., smaller trees, fewer parameters) is an inductive bias that often aids generalization.

Visualizing Inductive Bias

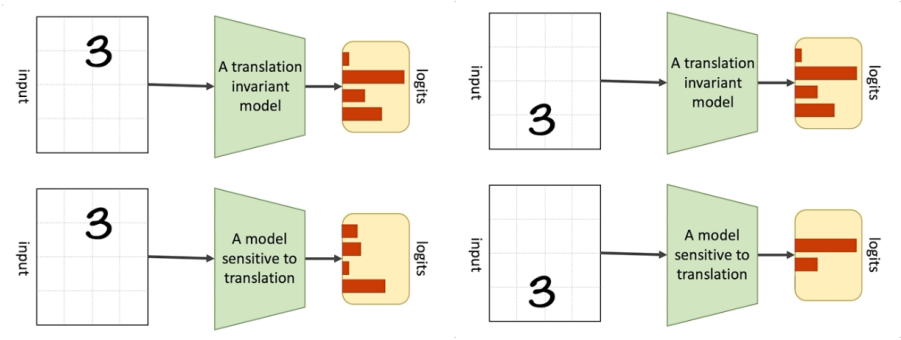


Figure 2: Adapted from Distilling Inductive Biases

Visualizing Inductive Bias

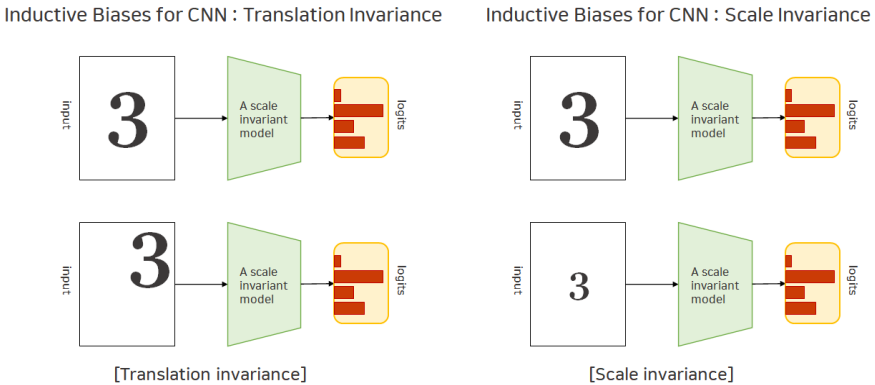


Figure 3: Adapted from Transferring Inductive Bias Through Knowledge Distillation

Visualizing Inductive Bias

A drawing of how inductive biases can affect models' preferences to converge to different local minima. The inductive biases are shown by colored regions (green and yellow) which indicates regions that models prefer to explore.

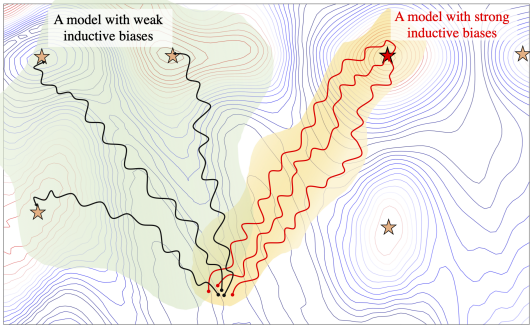


Figure 4: Adapted from Distilling Inductive Biases

Types of Inductive Bias in Neural Networks

- **Convolutional Neural Networks (CNNs):** Designed with **locality and translation-invariance bias**.
  Example: CNNs excel in image classification because nearby pixels often share related features.
- **Recurrent Neural Networks (RNNs):** Designed with **temporal continuity** bias, making them suitable for sequential data.
  Example: RNNs work well for time series and natural language processing, where current information depends on previous data points.
- **Transformer Models:** Use attention mechanisms to capture **dependencies across positions** in data, embodying a flexible bias suitable for diverse tasks.
  Example: Transformers are highly effective in language modeling and tasks requiring understanding of long-range dependencies.

Adaptive Inductive Bias in Neural Networks

- **Bias Shifting During Training:**
  - Neural networks can adapt inductive bias as they learn, **dynamically adjusting** feature extraction or attention patterns.
- **Implications in Transfer Learning:**
  - Pre-trained models on large datasets (e.g, ImageNet) carry learned inductive biases that can be **fine-tuned for specific tasks**.
- **Example in Practice (Classical ML):**
  - **Kernel SVMs:** Switching from a linear kernel to an RBF kernel changes the **inductive bias** to favor more complex, nonlinear relationships.

Adaptive Inductive Bias in Neural Networks (Future Topics)

- **Neural Networks:**
  - During fine-tuning, neural networks modify their learned representations, shifting from **generic features** to **task-specific patterns**.
  - Earlier layers often stay stable (retaining broad biases), while deeper layers adapt more strongly.
- **Convolutional Neural Networks (CNNs):**
  - Pre-trained CNNs adjust their **spatial feature detectors** to new domains (e.g., from natural images to medical imaging).
- **Large Language Models (LLMs):**
  - Fine-tuning shifts the model's **attention bias** toward patterns in the target dataset (e.g., legal text, medical notes, or code).
  - Instruction tuning further changes the bias toward producing **helpful, structured outputs**.

Regularization in Neural Networks

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

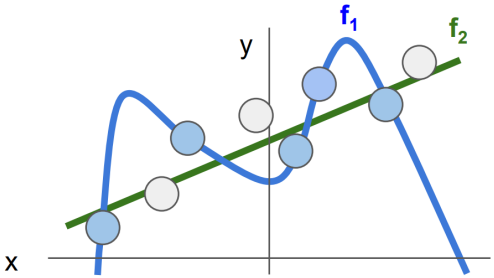


Figure 5: Regularization prevents overfitting by avoiding fitting noise in the data.

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

- **Lasso Regression (L1 Regularization):** Ideal for feature selection by setting some coefficients to zero.

Key Regularization Techniques

- **Lasso Regression (L1 Regularization):** Ideal for feature selection by setting some coefficients to zero.
  - **Example:** Predicting house prices with many features. Lasso simplifies the model by focusing on key predictors (e.g., location, size).

Key Regularization Techniques

- **Lasso Regression (L1 Regularization):** Ideal for feature selection by setting some coefficients to zero.
  - **Example:** Predicting house prices with many features. Lasso simplifies the model by focusing on key predictors (e.g., location, size).
  - **Application:** Useful in NLP, where it identifies significant words by reducing others to zero.

Key Regularization Techniques

- **Lasso Regression (L1 Regularization):** Ideal for feature selection by setting some coefficients to zero.
  - **Example:** Predicting house prices with many features. Lasso simplifies the model by focusing on key predictors (e.g., location, size).
  - **Application:** Useful in NLP, where it identifies significant words by reducing others to zero.
- **Ridge Regression (L2 Regularization):** Suitable when all features contribute meaningfully, minimizing overfitting without eliminating features.

Key Regularization Techniques

- **Lasso Regression (L1 Regularization):** Ideal for feature selection by setting some coefficients to zero.
  - **Example:** Predicting house prices with many features. Lasso simplifies the model by focusing on key predictors (e.g., location, size).
  - **Application:** Useful in NLP, where it identifies significant words by reducing others to zero.
- **Ridge Regression (L2 Regularization):** Suitable when all features contribute meaningfully, minimizing overfitting without eliminating features.
  - **Example:** Medical research using gene expression data, where every feature holds predictive power.

Key Regularization Techniques

- **Lasso Regression (L1 Regularization):** Ideal for feature selection by setting some coefficients to zero.
  - **Example:** Predicting house prices with many features. Lasso simplifies the model by focusing on key predictors (e.g., location, size).
  - **Application:** Useful in NLP, where it identifies significant words by reducing others to zero.
- **Ridge Regression (L2 Regularization):** Suitable when all features contribute meaningfully, minimizing overfitting without eliminating features.
  - **Example:** Medical research using gene expression data, where every feature holds predictive power.
  - **Application:** Ideal for multicollinear data, like predicting product sales with multiple correlated predictors.

Key Regularization Techniques

- **Lasso Regression (L1 Regularization):** Ideal for feature selection by setting some coefficients to zero.
  - **Example:** Predicting house prices with many features. Lasso simplifies the model by focusing on key predictors (e.g., location, size).
  - **Application:** Useful in NLP, where it identifies significant words by reducing others to zero.
- **Ridge Regression (L2 Regularization):** Suitable when all features contribute meaningfully, minimizing overfitting without eliminating features.
  - **Example:** Medical research using gene expression data, where every feature holds predictive power.
  - **Application:** Ideal for multicollinear data, like predicting product sales with multiple correlated predictors.
- **Elastic Net (L1 + L2):** Balances feature selection and generalization, effective with high feature correlation.

Key Regularization Techniques

- **Lasso Regression (L1 Regularization):** Ideal for feature selection by setting some coefficients to zero.
  - **Example:** Predicting house prices with many features. Lasso simplifies the model by focusing on key predictors (e.g., location, size).
  - **Application:** Useful in NLP, where it identifies significant words by reducing others to zero.
- **Ridge Regression (L2 Regularization):** Suitable when all features contribute meaningfully, minimizing overfitting without eliminating features.
  - **Example:** Medical research using gene expression data, where every feature holds predictive power.
  - **Application:** Ideal for multicollinear data, like predicting product sales with multiple correlated predictors.
- **Elastic Net (L1 + L2):** Balances feature selection and generalization, effective with high feature correlation.
  - **Example:** Financial models where predictors are correlated. Elastic Net combines Lasso's selection with Ridge's handling of multicollinearity.

Key Regularization Techniques

Characteristic	Ridge Regression	Lasso Regression
Penalty Type	L2 (squared coefficient magnitudes)	L1 (absolute coefficient magnitudes)
Coefficient Shrinkage	Shrinks coefficients without setting to zero	Can shrink coefficients exactly to zero
Feature Selection	Retains all features	Selects features by setting some to zero
Solution Path	Coefficients generally non-zero	Many coefficients can be zero
Model Complexity	Includes all features, higher complexity	Simplifies model by excluding features

Table 1: Comparison of Ridge and Lasso Regression Techniques

Key Regularization Techniques

Impact on Prediction	Good for multicollinear data	Reduces model complexity, aiding high-dimensional data
Interpretability	Less interpretable; all features retained	More interpretable; excludes irrelevant features
Best for	When all features are relevant	When identifying key features among many
Bias-Variance Tradeoff	Adds bias, reduces variance	Similar to Ridge, but with higher bias from exclusion
Computation	Faster without feature selection	Slightly slower due to feature selection

Table 2: Comparison of Ridge and Lasso Regression Techniques

Key Regularization Techniques

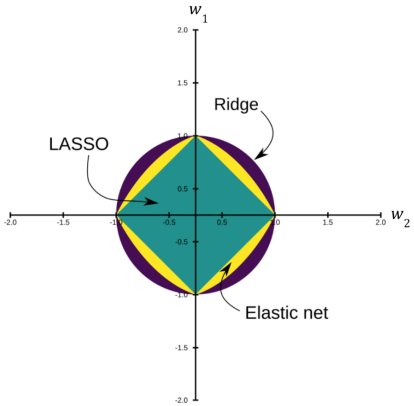
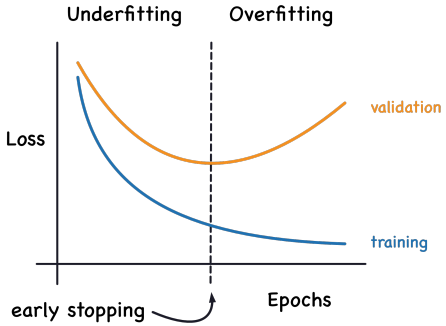


Figure 6: Lasso, Ridge, and Elastic net Comparision

Key Regularization Techniques

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

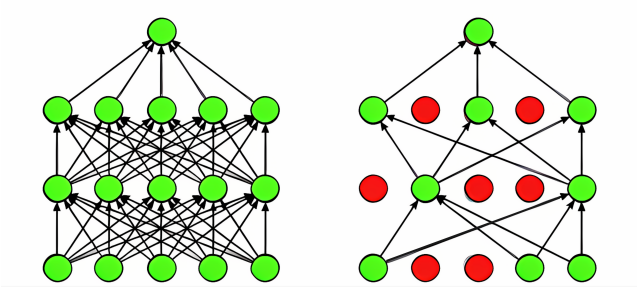


Background and Motivation for Dropout

- **Ensemble Learning**
  - Combines **multiple models** to increase model capacity and reduce overfitting.
  - However, training multiple large models can be computationally **expensive**.
- **Bayesian Regularization**
  - Assumes a **prior distribution** on parameters, balancing data fit with model complexity.
  - Prevents overfitting by **limiting model reliance** on specific parameter values.
- **Inspiration from Nature**
  - In evolution, genes are mixed randomly in each generation, creating robust organisms by avoiding dependency on specific gene combinations.
  - Similarly, dropout **reduces neural dependencies**, encouraging robust, generalizable features.

Concept of Dropout

- Dropout is a regularization technique that reduces overfitting in neural networks.
  - **Randomly deactivates** neurons in each layer during training.
  - Effectively trains **multiple "thinned" sub-networks**, then averages them at test time.
- This makes neurons more robust, as they cannot rely on specific other neurons to perform well.



(a) Standard Neural Network (b) After Applying Dropout

Mechanics of Dropout: Training vs. Inference

- **During Training**
  - A fraction of neurons is randomly set to zero in each layer (dropout rate typically 0.5 for hidden layers).
  - Only **active neurons** contribute to forward pass and backpropagation.
- **During Inference**
  - All neurons are active.
  - To balance the increase in neuron count, **scale the weights** by the dropout probability.
  - Weight scaling compensates for the increase in neuron count, maintaining consistent output expectations.

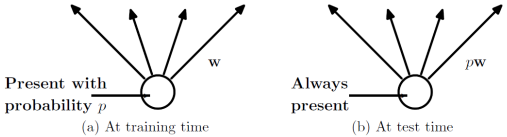


Figure 7: Training vs. Inference in Dropout adapted from Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Why Dropout Prevents Overfitting

- **Breaks Co-Adaptations**
  - Forces neurons to **operate independently**, creating more generalizable feature representations.
- **Acts as Model Averaging**
  - Randomly sampling neuron subsets is akin to training an ensemble of networks.
  - Inference combines these sub-networks, improving model performance on unseen data.

Real-World Impact of Dropout

- **Applications in Image Recognition**
  - Enhanced performance on MNIST, CIFAR-10, and ImageNet, helping models generalize to varied visual inputs.
- **Speech Recognition and Text Classification**
  - Reduced error rates on TIMIT speech dataset and improved document classification on Reuters.
- **Computational Biology**
  - Used for gene splicing prediction, dropout helps extract robust features from small genetic datasets.

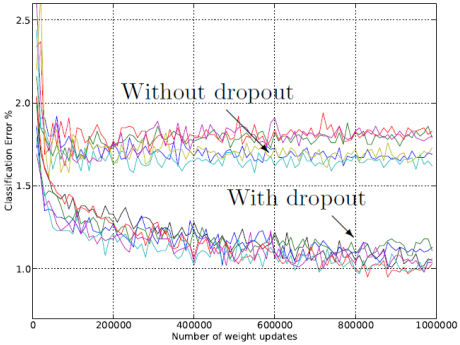


Figure 8: Effect of dropout for different architectures: adapted from Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Dropout Algorithm

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

where f is any activation function, for example, $f(x) = 1/\left(1 + \exp(-x)\right)$.

With dropout, the feed-forward operation becomes

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

Figure 9: Dropout Algorithm in Feedforward Networks: adapted from Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Dropout Best Practices

- **Dropout Rate:** 0.5 for hidden layers is a common starting point, but the rate should be tuned for each model.
- **Where to Apply:** Often applied in fully connected layers; for convolutional layers, spatial dropout is preferred.
- **Avoiding Underfitting:** Avoid using too high of a dropout rate, as this may cause excessive sparsity and underfitting.

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.

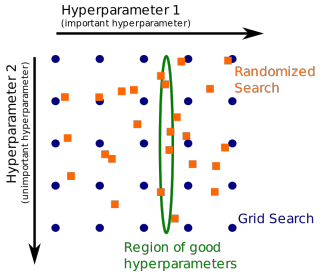


Figure 10: Grid Search vs. Random Search in Hyperparameter Tuning.

Hyperparameter Tuning Best Practices

- **Start Simple:** Begin with tuning a smaller subset of key hyperparameters (e.g., learning rate, batch size).
- **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.
- **Automate:** Leverage frameworks like `scikit-learn` to automate and parallelize the tuning process.
- **Track Experiments:** Use tools like `Wandb` to systematically track hyperparameters, training metrics, and results.
- **Use Learning Curves:** Plot learning curves to monitor the model's performance over time and detect overfitting or underfitting early.
- **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.

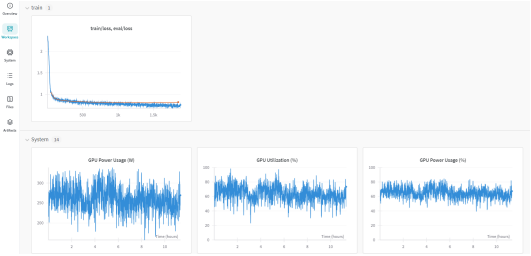


Figure 11: Training metrics visualization in `wandb`.

[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.

Contributions

These slides are authored by:

- Faezeh Sarlakifar
- Sogand Salehi
- Sina Daneshgar

Any Questions?