Deep Dive into Overfitting and Regularization

Swish Act f
exp linear unit

Farid Afzali, Ph.D., P.Eng.

Clarification: Steps for Model Training with 5 Mini-Batches

- Preprocessing and Initialization
  1. **Data Shuffling**: Shuffle the entire training dataset to ensure randomness.
  2. **Mini-Batch Creation**: Divide the shuffled training data into 5 mini-batches.
  3. **Initialize Parameters**: Initialize the model's weights and biases.
- Training Loop
  4. **For each Epoch, perform the following steps**:
  - **For each of the 5 Mini-Batches, perform these sub-steps**:
    - **Forward Pass**: Pass the mini-batch through the network to get the output.
    - **Compute Loss**: Calculate the loss function using the predicted output and true labels.
    - **Backward Pass**: Compute the gradients of the loss function with respect to the model parameters.
    - **Update Parameters**: Update the model's weights and biases using gradient descent or some variant thereof.
- Post-Training
  5. **Evaluation**: After training is complete, turn off any regularization techniques like dropout and evaluate the model on a validation set.
  6. **Hyperparameter Tuning**: If necessary, adjust hyperparameters and repeat the training process.
- Model Assessment
  7. **Final Evaluation**: Test the model on an independent test set to gauge its generalization capability.

Regularization

why need cv

test data don’t use in feature engineering, divide from very first

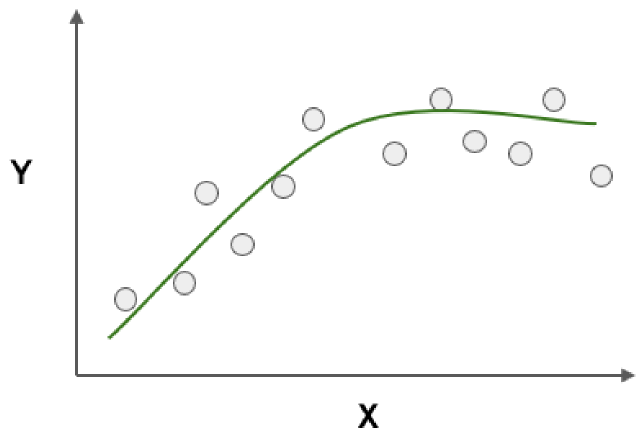
- Regularization is a powerful technique in machine learning that addresses common model issues by achieving the following objectives:

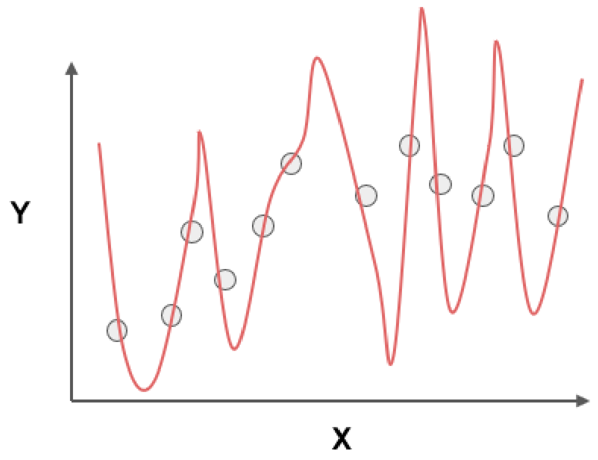
1. **Minimizing Model Complexity:** Regularization aims to control and reduce the complexity of a model. Complex models, especially those with many features or high polynomial degrees, have a greater capacity to fit noise in the data. Regularization techniques add constraints to the model to prevent it from becoming overly complex and overfitting the training data.
2. **Penalizing the Loss Function:** Regularization introduces additional terms in the loss function that penalize large coefficients or weights associated with features. These penalty terms encourage the model to use smaller coefficients, effectively discouraging it from relying too heavily on any single feature. This helps prevent overfitting and encourages a simpler model.
3. **Reducing Model Overfitting:** Regularization effectively reduces model overfitting by adding bias to the model in order to decrease variance. By adding constraints to the model's parameters, regularization forces the model to generalize better to unseen data. It strikes a balance between fitting the training data closely and ensuring that the model is not too sensitive to noise or outliers.

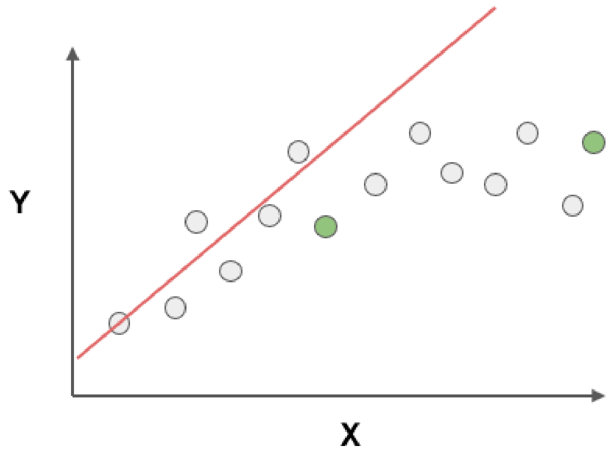
Bias Variance Trade off

- The trade-off can be summarized as follows:
- **High Bias, Low Variance**: A model that is too simplistic, with high bias and low variance, may not capture the underlying patterns in the data. It underfits the data.
- **Low Bias, High Variance**: A highly complex and flexible model, with low bias and high variance, can fit the training data well but may generalize poorly to new, unseen data because it captures noise.
- The goal in machine learning is to find the right balance between bias and variance. This balance depends on the specific problem, the complexity of the model, and the amount of training data available. Some techniques to manage the bias-variance trade-off include cross-validation, regularization, and choosing an appropriate model complexity.

Bias-Variance Trade off







How to figure out?

- Evaluating underfitting and overfitting in multi-dimensional datasets, especially when dealing with higher-dimensional feature spaces, can be challenging to visualize directly. However, you can use certain techniques to indirectly assess these issues and find the right balance of model complexity. Here are some common methods:

1. **Train-Test Split:** Split your dataset into training and testing subsets. Train your model on the training data and evaluate its performance on the testing data. By comparing training and testing errors, you can get an idea of whether your model is underfitting or overfitting.
    1. **Underfitting:** If both training and testing errors are high, it suggests the model is too simple and underfitting the data.
    2. **Overfitting:** If the training error is much lower than the testing error, it indicates the model is too complex and overfitting the training data.
2. **Cross-Validation:** Utilize k-fold cross-validation to assess model performance across multiple subsets of the data. Cross-validation provides a more robust estimate of model performance and can help identify issues related to underfitting or overfitting.
3. **Learning Curves:** Plot learning curves that show how training and testing errors change with the increase in training data size. Learning curves can reveal patterns of underfitting and overfitting.
    1. **Underfitting:** In the early stages, both training and testing errors are high, indicating a model is underfitting. As you provide more data, both errors may decrease, but the gap between them remains significant.
    2. **Overfitting:** In the early stages, training error is low, but testing error is high. As you provide more data, testing error may decrease, but the gap between training and testing errors remains significant.

Cross Validation

- Cross-validation is indeed a more advanced and robust technique for splitting data into training and testing sets compared to a simple one-time train-test split. It's an essential tool in machine learning for model evaluation, and it helps address some of the limitations of a single train-test split. Here are some key points about cross-validation:

1. **Mitigates Variability:** A single train-test split can be sensitive to the specific random sampling of data points. Cross-validation mitigates this variability by performing multiple splits, providing a more stable estimate of model performance.
2. **Better Utilization of Data:** Cross-validation allows you to make more efficient use of your dataset. Instead of setting aside a fixed portion for testing, it divides the data into multiple subsets for training and testing in a rotating manner. This ensures that all data points are used for both training and testing at some point.
3. **K-Fold Cross-Validation:** One of the most common cross-validation techniques is k-fold cross-validation. In k-fold cross-validation, the dataset is divided into k equally sized "folds." The model is trained and tested k times, each time using a different fold as the testing set and the remaining k-1 folds as the training set. The results are then averaged to obtain a more robust performance estimate.
4. **Stratified Cross-Validation:** In classification problems, stratified cross-validation ensures that each fold has a similar distribution of class labels as the original dataset. This is particularly useful when dealing with imbalanced datasets.
5. **Leave-One-Out Cross-Validation (LOOCV):** In LOOCV, each data point serves as the testing set once, while the rest of the data points are used for training. This is an extreme form of cross-validation and is computationally expensive but provides a very accurate estimate of model performance.
6. **Grid Search and Hyperparameter Tuning:** Cross-validation is often used in conjunction with grid search or random search for hyperparameter tuning. It helps identify the best hyperparameters by evaluating model performance across various parameter combinations.

Model Complexity

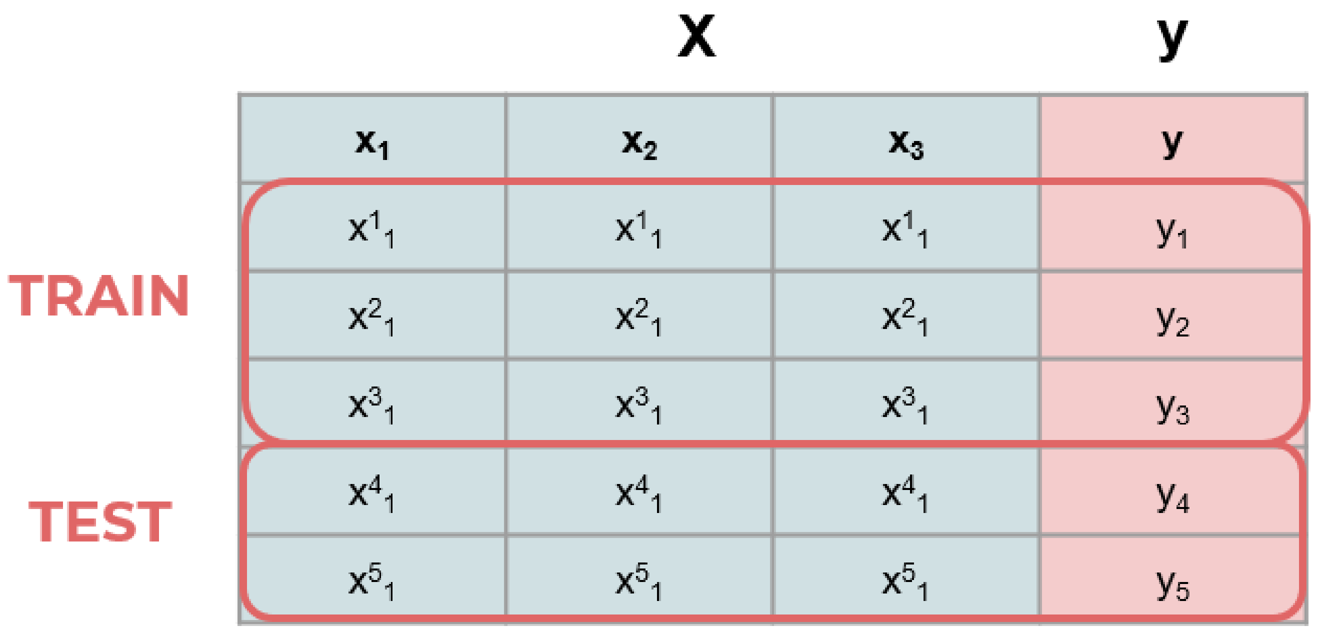
- Many machine learning/Deep Learning algorithms have their own hyperparameters that can significantly impact model complexity. These hyperparameters allow you to fine-tune the model's behavior and complexity to better fit the data. Here are a few examples:
    - ❑**Regularization Strength (e.g., Ridge and Lasso Regression)**
    - ❑**Depth of Decision Trees (e.g., Decision Trees and Random Forests)**
    - ❑**Number of Estimators (e.g., Random Forests and Gradient Boosting)**
    - ❑**Kernel Parameters (e.g., Support Vector Machines)**
    - ❑**Learning Rate (e.g., Gradient Boosting)**
    - ❑**Number of Neighbors (e.g., k-Nearest Neighbors)**
    - ❑**Hidden Units and Layers (e.g., Neural Networks)**

Cross Validation

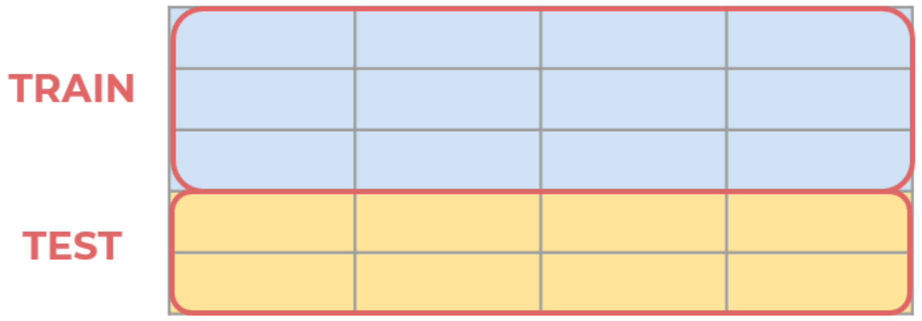
- Cross-validation is indeed a more advanced and robust technique for splitting data into training and testing sets compared to a simple one-time train-test split. It's an essential tool in machine learning for model evaluation, and it helps address some of the limitations of a single train-test split. Here are some key points about cross-validation:

1. **Mitigates Variability:** A single train-test split can be sensitive to the specific random sampling of data points. Cross-validation mitigates this variability by performing multiple splits, providing a more stable estimate of model performance.
2. **Better Utilization of Data:** Cross-validation allows you to make more efficient use of your dataset. Instead of setting aside a fixed portion for testing, it divides the data into multiple subsets for training and testing in a rotating manner. This ensures that all data points are used for both training and testing at some point.
3. **K-Fold Cross-Validation:** One of the most common cross-validation techniques is k-fold cross-validation. In k-fold cross-validation, the dataset is divided into k equally sized "folds." The model is trained and tested k times, each time using a different fold as the testing set and the remaining k-1 folds as the training set. The results are then averaged to obtain a more robust performance estimate.
4. **Stratified Cross-Validation:** In classification problems, stratified cross-validation ensures that each fold has a similar distribution of class labels as the original dataset. This is particularly useful when dealing with imbalanced datasets.
5. **Leave-One-Out Cross-Validation (LOOCV):** In LOOCV, each data point serves as the testing set once, while the rest of the data points are used for training. This is an extreme form of cross-validation and is computationally expensive but provides a very accurate estimate of model performance.
6. **Grid Search and Hyperparameter Tuning:** Cross-validation is often used in conjunction with grid search or random search for hyperparameter tuning. It helps identify the best hyperparameters by evaluating model performance across various parameter combinations.

Cross Validation

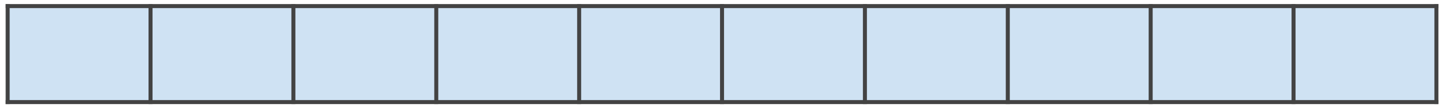


	X			**y**
	$\mathbf{x_1}$	$\mathbf{x_2}$	$\mathbf{x_3}$	$\mathbf{y}$
TRAIN	x^1_1	x^1_1	x^1_1	y_1
TRAIN	x^2_1	x^2_1	x^2_1	y_2
TRAIN	x^3_1	x^3_1	x^3_1	y_3
TEST	x^4_1	x^4_1	x^4_1	y_4
TEST	x^5_1	x^5_1	x^5_1	y_5

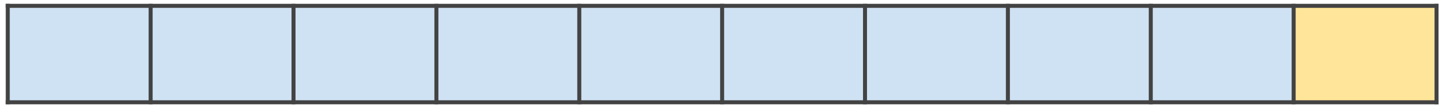


Cross Validation

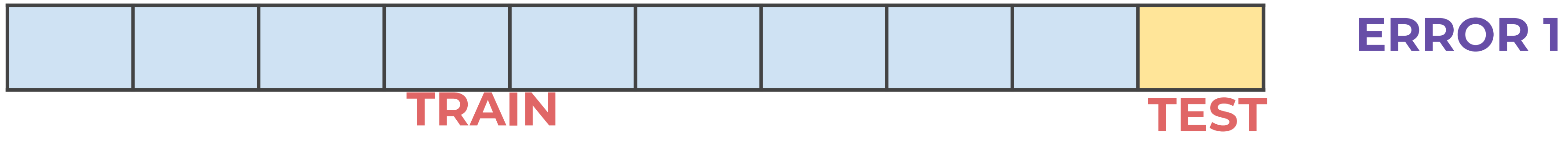
- Split data into K equal parts:



- 1/K left as test set

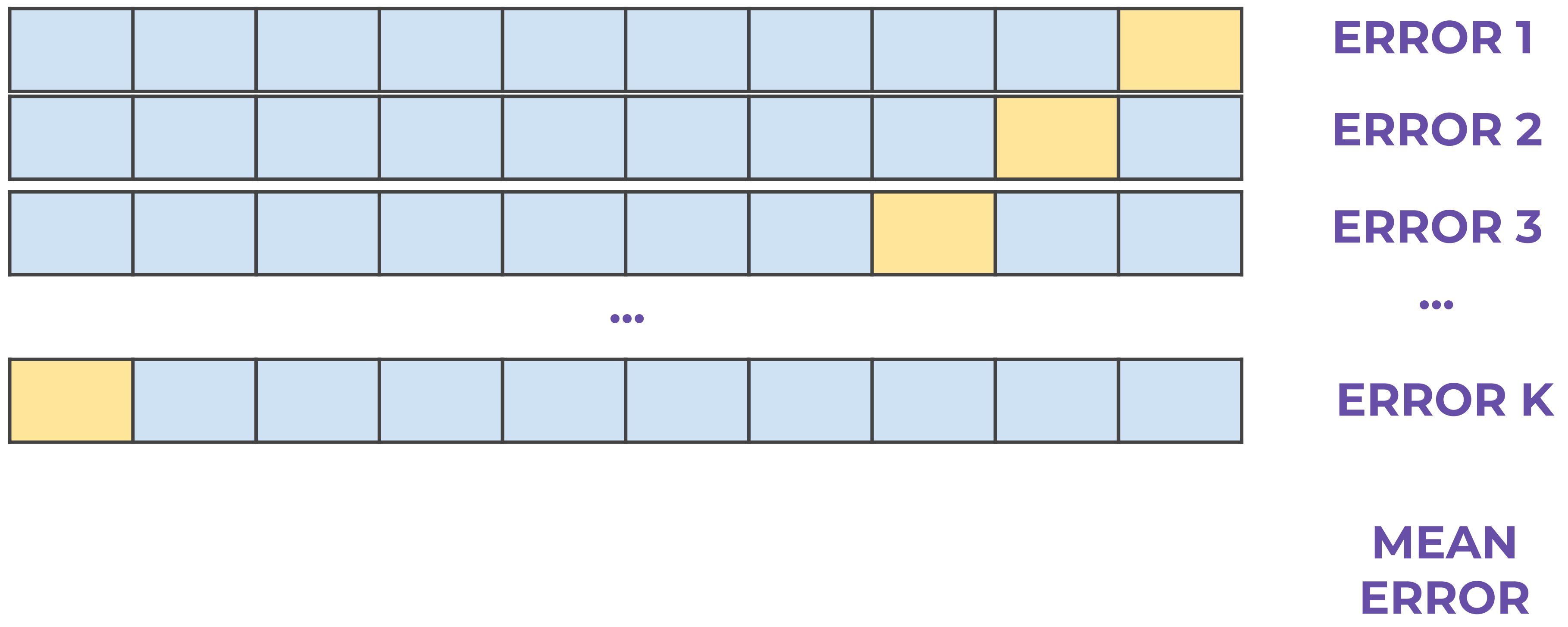


- Train model and get error metric for split:



Cross Validation

- Keep repeating for all possible splits



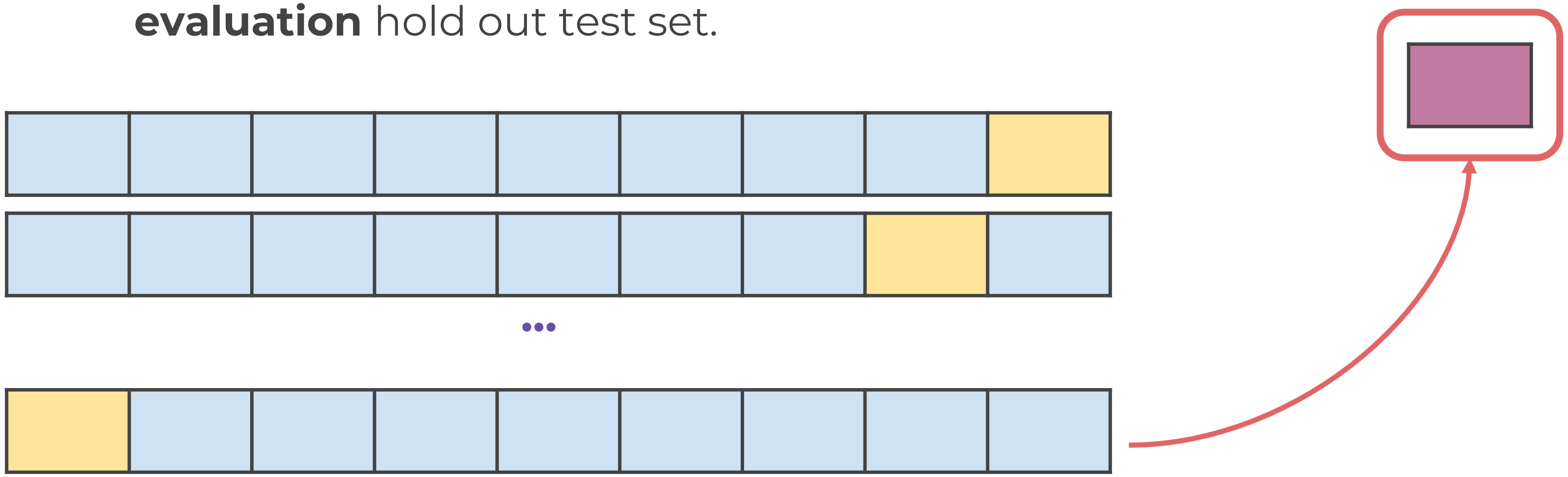
Cross Validation

- A hold-out test set, also known simply as a "test set," is a subset of your dataset that you reserve for the final evaluation of your machine learning model after training and hyperparameter tuning. The purpose of a hold-out test set is to provide an independent and unbiased measure of how well your model is expected to perform on unseen, real-world data. Here are the key characteristics and roles of a hold-out test set:

1. **Independence:** The hold-out test set should be completely independent of the data used for training and model validation. It should not be used at any point during model development, hyperparameter tuning, or cross-validation. This independence ensures that the test set provides an unbiased assessment of your model's generalization performance.
2. **Unseen Data:** The data in the test set should be entirely unseen by the model during training. This means that the model has never been exposed to this data in any way. It simulates how the model will perform when making predictions on new, previously unseen examples.
3. **Final Evaluation:** The hold-out test set is typically used for the final evaluation of your model's performance. After you have trained your model and optimized its hyperparameters using cross-validation, you assess how well it generalizes to the test set. This step gives you an estimate of how the model is likely to perform in a real-world scenario.

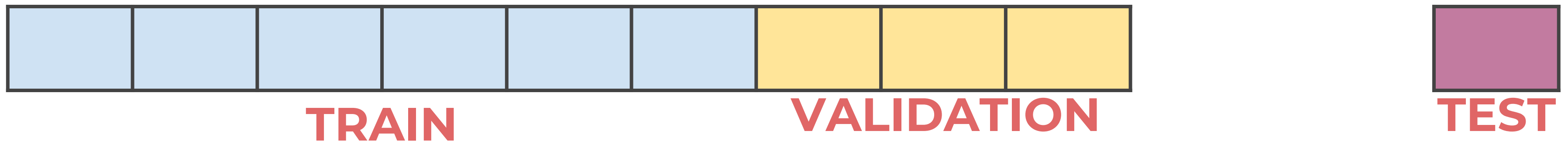
Cross Validation

- Remove a hold out test set
- Perform “classic” train test split:
- Train **and** tune on this data:
- **After** training and tuning perform **final evaluation** hold out test set.



Cross Validation

- Train | Validation | Test Split



- Allows us to get a true final performance metric to report.
- No editing model after this!

Cross Validation

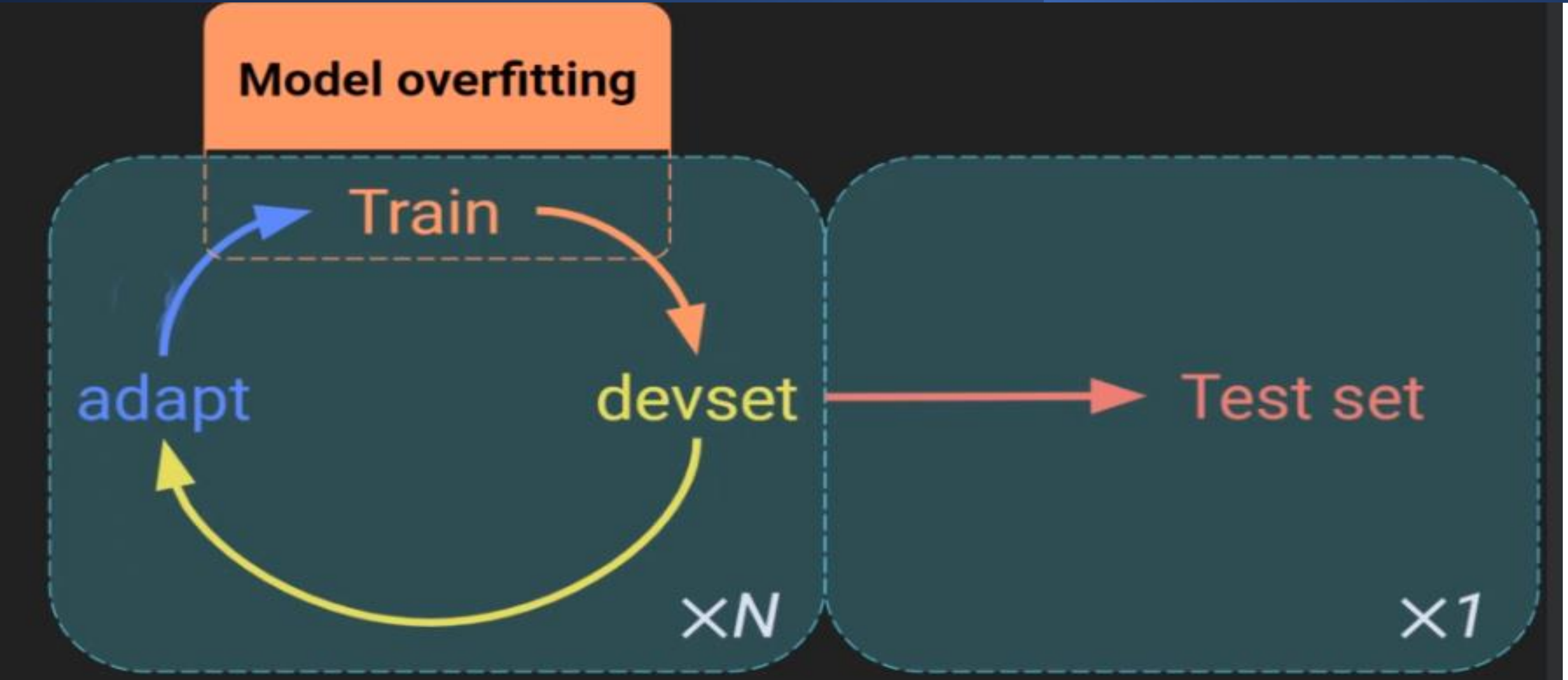
Cross-Validation Basics:

- ❑ Cross-validation is a technique used to assess and improve the performance of machine learning models.
- ❑ It involves dividing the dataset into training, development (or holdout), and test sets.
- ❑ The model is trained exclusively on the training set, avoiding the use of the development and test sets during training.

Cross-Validation Cycle:

- ❑ The training process typically involves multiple epochs where the model is fine-tuned iteratively.
- ❑ After each training phase, the model's performance is evaluated on the development set.
- ❑ Based on the results, adjustments can be made to the model's architecture, regularization, or hyperparameters.
- ❑ This cycle repeats until the model performs satisfactorily on both the training and development sets.
- ❑ The final evaluation is done on the test set to assess the model's generalization.

Cross Validation



Cross Validation

Overfitting and Cross-Validation:

- ❑ Overfitting, a perilous adversary in model development, transpires when a model becomes excessively specialized in fitting the training data, consequently limiting its proficiency in generalizing to previously unseen data.
- ❑ The essence of cross-validation lies in its ability to thwart overfitting by ensuring that the model is evaluated on independent datasets—the development and test sets.
- ❑ Central to cross-validation is the crucial assumption that the development and test sets are both independent and devoid of correlations with the training set.

Assumption Validity:

- ❑ The validity of this independence assumption is exemplified through pertinent examples.
- ❑ In scenarios where this assumption holds, overfitting is effectively curtailed, enabling models to generalize optimally.

Cross Validation

Valid Independence Assumption:

- ❑ Scenario: Training a model to classify pictures of pets.
- ❑ Training Set: Encompasses an array of pet pictures.
- ❑ Test Set: Comprises images captured by different individuals, entirely distinct from those featured in the training set.
- ❑ Validity: The independence assumption remains unscathed, as the test set comprises unrelated pet images, thus ensuring robust model evaluation.

Possible Independence Violation:

- ❑ Scenario: Developing a model to predict a person's age based on facial images.
- ❑ Training Set: Contains facial images alongside corresponding ages.
- ❑ Test Set: Includes images of siblings (brothers or sisters).
- ❑ Potential Violation: While the test set technically differs from the training data, a strong familial resemblance may lead to a correlation, potentially constraining the model's generalization.

Cross Validation

Another Independence Violation Example:

- ❑Scenario: Predicting home market values based on various property features.
- ❑Training Set: Constitutes randomly selected homes within a city.
- ❑Test Set: Incorporates houses located adjacent to those in the training set.
- ❑Violation: Although distinct homes, the proximity of houses within the same neighborhood may challenge the independence assumption, potentially impairing generalization.

Cross Validation

- **The Nuances of Overfitting:**
  - ❑It is imperative to recognize that overfitting is not inherently detrimental; its implications depend on the specific context and objectives of the model.
  - ❑In certain scenarios, overfitting to a particular dataset aligns with the model's intended purpose, emphasizing the need for a nuanced approach.

Generalization

- **1. The Significance of Generalization:**
  - ❑Generalization stands as one of the pivotal concepts in the development and application of deep learning models and machine learning more broadly.
  - ❑Its essence lies in the model's ability to perform effectively when exposed to new, unseen data.
  - ❑Models inherently possess generalization boundaries that determine the scope of their applicability.
- **2. Generalization Boundaries Defined:**
  - ❑Generalization boundaries refer to the limits within which a model is expected to function optimally.
  - ❑These boundaries are defined based on the specific population or dataset to which the model should apply.
  - ❑The following examples shed light on the concept of generalization boundaries.

Generalization-Examples

- **2.1 Example 1: Predicting Weight Based on Height and Calories:**
  - ❑Scenario: Modeling weight prediction based on height and calorie consumption.
  - ❑Generalization Boundaries: The model aims to generalize across adult humans of all sexes and nationalities.
  - ❑Exclusion: The model does not need to apply to children or non-human animals due to distinct physiological differences.
- **2.2 Example 2: City-Specific Model:**
  - ❑Scenario: Model development for predicting home market values in a specific city (City A).
  - ❑Generalization Boundaries: The model is focused solely on City A, catering to its unique dynamics and requirements.
  - ❑Exclusion: The model's generalization scope does not extend to other cities, as it aligns with the specific needs of City A.

Generalization

- **3. Generalization and Loss of Accuracy:**
  - ❑It is crucial to acknowledge that broader generalization often comes at the cost of reduced accuracy.
  - ❑Generalizing to diverse populations or datasets inherently introduces variability and noise, which can impact model accuracy.
  - ❑The challenge lies in constructing a model that maximizes performance within the defined generalization boundaries.
- **4. Setting Generalization Boundaries:**
  - ❑The careful delineation of generalization boundaries is a critical step in model development.
  - ❑These boundaries should align with the intended application and scope of the model.
  - ❑Ensuring representation of all relevant populations within the training, development, and test datasets is essential to effective generalization.

Scikit-learn library

- **Why Data Splitting Matters:**
  - ❑The rationale behind splitting data into training and testing sets is to strike a balance between model learning and assessment.
  - ❑While the desire is to maximize the training data, a sufficient amount must be retained for testing and validation.
  - ❑The concept of "enough" data depends on several factors, including dataset size, model architecture, and generalizability considerations.
  - ❑It is crucial to recognize that there is no universal rule for data partitioning; it often hinges on the specific problem at hand.
  - ❑ *For instance, consider the Iris dataset, where an 80% training and 20% test split yielded only 30 data points for testing. This demonstrates that small test sets may limit the variability in accuracy. Conversely, a massive dataset like ImageNet can accommodate a 90-10 split, resulting in substantial test sets, such as 140,000 images for evaluation.*

Scikit-learn library(PP#)

- **The Flexibility of Data Partitioning:**
  - ❑One key takeaway is that data partitioning is not an exact science.
  - ❑There is no universally correct way to divide data; instead, it hinges on the problem's specifics and dataset characteristics.
  - ❑Researchers must exercise flexibility and adapt partitioning strategies accordingly.
- **Running Systematic Experiments:**
  - ❑To comprehensively explore the impact of data partitioning proportions, the work activity presents an experiment involving repeated model training with varying proportions of training and test data.
  - ❑While the Iris dataset serves as the experimental platform, the approach is generalizable to other datasets and models.
  - ❑The experiment aims to uncover trends and insights regarding the relationship between data partitioning and model performance.

Data Loaders and Mini-Batch Training in Deep Learning

- **Data Splitting Techniques**

❖ Manual Data Splitting

  ❑ Data can be partitioned into training and test sets manually using libraries such as NumPy or scikit-learn. This approach provides control but may not be efficient for larger datasets.

❖ Introduction to Data Loaders

  ❑ For more complex tasks and larger datasets, data loaders come to the rescue. Data loaders are utilities provided by PyTorch that create iterable objects, making it easier to manage and feed data during training.

- **Mini-Batch Training**
- The Concept of Batches

  ❑ Mini-batches involve dividing the training set into smaller groups, each containing a subset of the data. This contrasts with training on individual samples or the entire dataset simultaneously.

- Benefits of Mini-Batch Training

  ❑ Training in mini-batches offers advantages, including reduced computation time and improved model accuracy. The following examples illustrate the impact of batch size on training.

Mini Batch Examples/Reasons

- **Example 1: Batch Size of 1**
  - ❑Training one sample at a time.
  - ❑Requires a large number of epochs for high accuracy (e.g., 900-1000 epochs).
  - ❑Computationally intensive.
- **Example 2: Batch Size of 30**
  - ❑Training in groups of 30 samples.
  - ❑Achieves similar high accuracy but converges faster (e.g., 150-200 epochs).
  - ❑Reduces computational load.

Clarification

- ✓ Mini-batch:
  - ❑ This technique is used during the training phase to update the model's weights. It is a compromise between stochastic gradient descent, which updates the weights using one sample at a time, and batch gradient descent, which uses the entire dataset for each weight update. The primary objective of mini-batch gradient descent is to optimize the training process.
- ✓ Cross-Validation:
  - ❑ This is a model evaluation technique, often used to assess a model's generalization performance. In k-fold cross-validation, for example, the data is divided into k subsets. The model is trained on k-1 of these subsets and tested on the remaining one, cycling through all k subsets for testing. The objective is to provide a more robust measure of model performance.

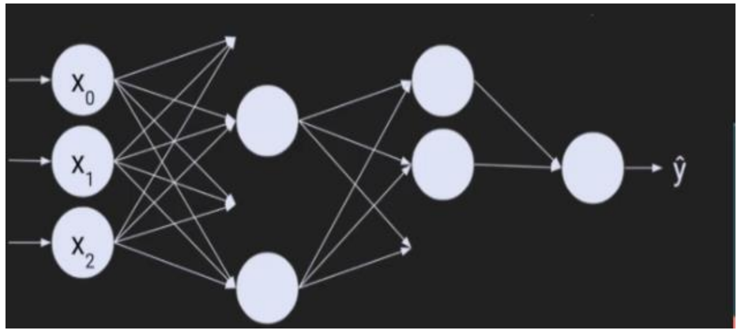
Regularization

- ❑Regularization is a critical aspect of developing complex and effective deep learning models, as it prevents overfitting and promotes better generalization.
- ❑Without regularization, deep learning models would be limited in their capacity to tackle intricate problems with complex architectures.
  - ✓Introduction Regularization is a fundamental technique in the field of deep learning that plays a pivotal role in building more robust and generalizable models.
  - ✓Regularization is indispensable for preventing models from memorizing training examples, which can lead to poor generalization.
  - ✓It also influences the nature of the representations within the model, making them either more sparse or distributed.

Regularization and its impacts on Deep Learning

- ***Preventing Overfitting:*** Regularization techniques penalize models for fitting training data too closely, ensuring they do not memorize specific examples but generalize to unseen data.
- ***Shaping Model Representations:*** Regularization methods can shape the nature of model representations, making them more sparse or distributed, depending on the technique employed.
- ***Training Efficiency:*** While regularization may increase or decrease training time, its effects depend on the chosen method, model complexity, and dataset size.
- ***Training Accuracy:*** In general, regularization tends to decrease training accuracy but enhances generalization and test accuracy. However, for larger and more complex models, it can increase training accuracy.

Regularization and its impacts on Deep Learning



Types of Regularization

✓ There are three primary families of regularization in deep learning:

❑ Node Regularization: Modifying the model architecture itself. The most common method is dropout, where nodes are randomly dropped during training.

❑ Loss Regularization: Adding a penalty term to the loss function. L1 and L2 regularization are examples that discourage weights from becoming excessively large.

❑ Data Regularization: Modifying or augmenting the dataset to increase its size. Data augmentation is commonly used, especially in image processing tasks.

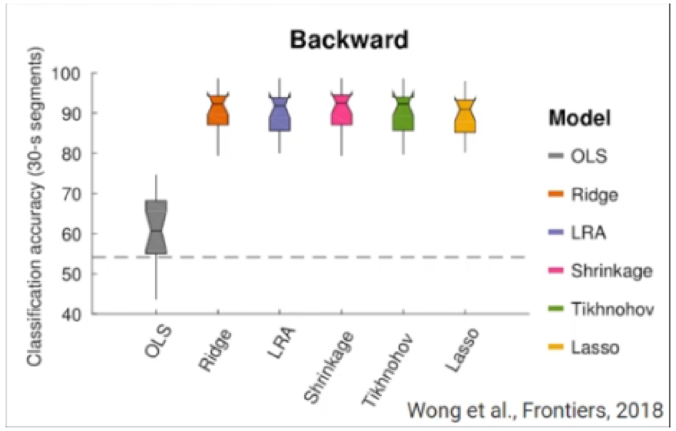
Selecting the best Regularization method

❖Selecting the appropriate regularization method for a specific problem can be challenging.

❖ Guidelines and empirical experiments can help guide the decision-making process.

❖Often, the best method depends on the model architecture and dataset characteristics.

❖ In some cases, different regularization techniques may yield comparable results.



A Comprehensive Guide to Mode Toggling for Deep Learning Models

- Deep learning models can operate in two distinct modes: training and evaluation.
- Switching between these modes is essential for managing gradient computations and applying regularization techniques.
- The primary reasons for toggling between training and evaluation modes are as follows:
  - ❑Gradient Computation: During training, gradients are computed through backpropagation to update model parameters. In evaluation, we do not want to compute gradients.
  - ❑Regularization: Specific regularization methods, such as dropout and batch normalization, are applied during training and must be deactivated during evaluation.

A Comprehensive Guide to Mode Toggling for Deep Learning Models

- There are two key functions in PyTorch for mode toggling:
  - ❑ **net.train():** This function puts the model into training mode, enabling gradient computations and regularization.
  - ❑ **net.eval():** Used to switch the model into evaluation mode, deactivating gradient computations and regularization.
- Dealing with Gradient Computation
- While **net.train()** and **net.eval()** control gradient computations, there is a third function, **torch.no_grad()**, that turns off gradient-related overhead computations.
- This function is especially useful for larger and more complex models.

Practical Implementation

- Training Loop:
  - ❑ Begin your training loop.
  - ❑ Start with **net.train()** to activate training mode.
  - ❑ Perform training, typically over mini-batches.
- Evaluation:
  - ❑ After training, switch to **net.eval()** to deactivate gradient computations and regularization.
  - ❑ Use **torch.no_grad()** within the evaluation block to further optimize computation time.
- Reverting to Training Mode:
  - ❑ When returning to the training loop or starting a new epoch, use **net.train()** to re-enable training mode.

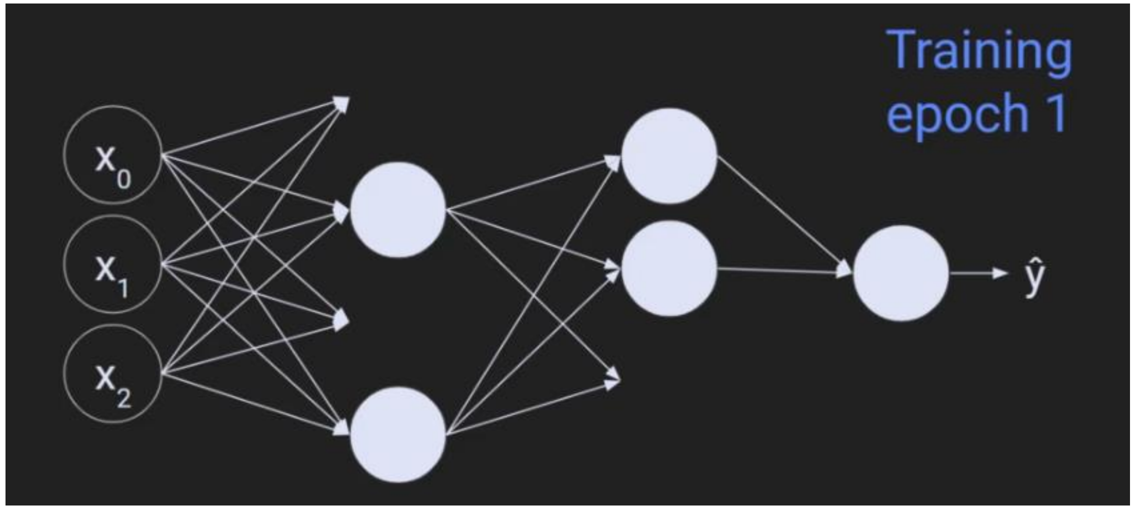
Dropout Regularization

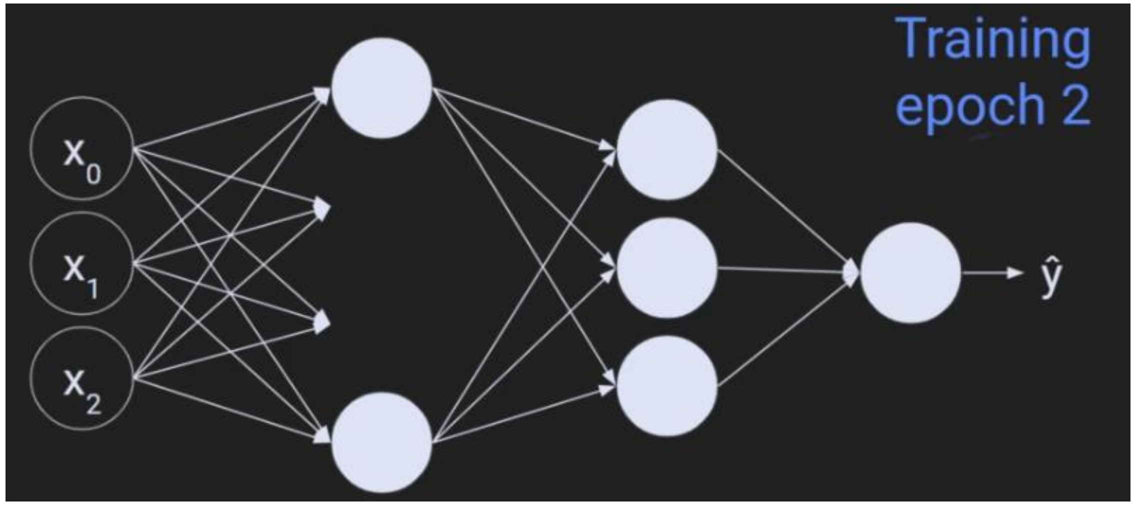
- Dropout regularization has proven to be a highly effective method for improving the robustness and generalization capabilities of deep learning models.
- Dropout regularization is a conceptually straightforward technique.
- During training, a portion of neural network units, chosen randomly with a specified probability, is temporarily switched off.
- This encourages the network to develop more robust, distributed representations.
- However, dropout introduces scaling issues, which require careful handling during testing.

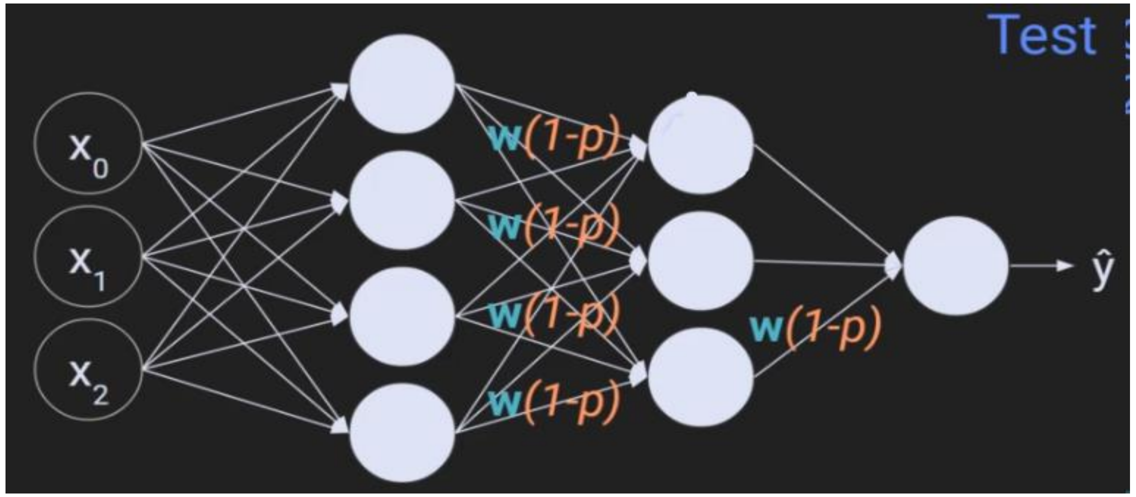
Scaling Methods for Dropout

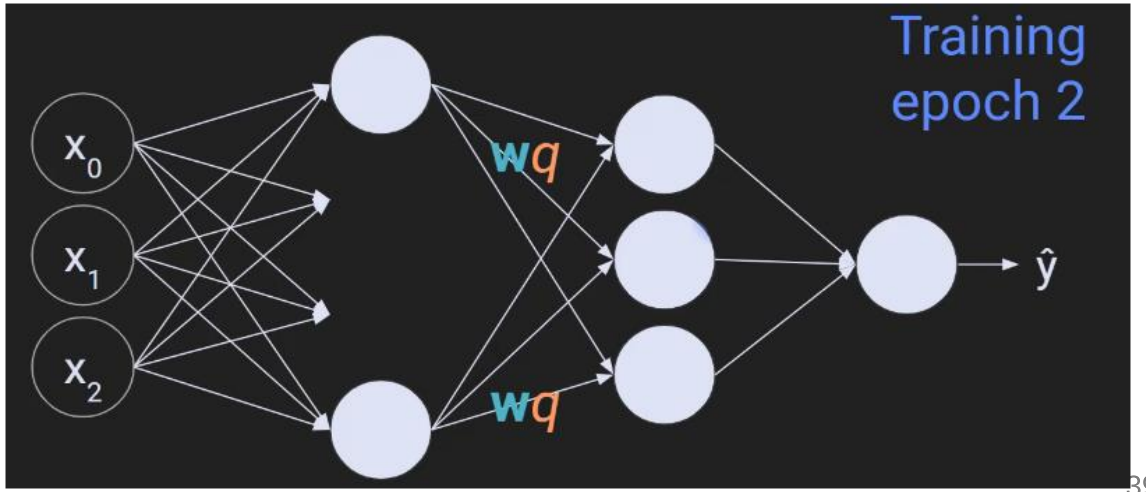
- Scaling Down during Testing
  - ❑During testing, weights are scaled down by a factor of (1 - p), where 'p' is the dropout probability. This ensures consistent input strength during training and testing.
- Scaling Up during Training
  - ❑During training, weights are scaled up by a factor of (1 / (1 - p)), effectively increasing their magnitudes. This compensates for dropout's reduction in input strength.

Scaling Methods for Dropout









Interpretations of Dropout

While dropout regularization is widely adopted in practice, its exact mechanism remains somewhat mysterious.

Some interpretations include:

- ❑Preventing single units from overfitting.
- ❑Encouraging more distributed representations.
- ❑Reducing reliance on individual units.
- ❑Enhancing model stability.

Regularization

- Three main types of Regularization:
  - L1 Regularization
    - LASSO Regression
  - L2 Regularization
    - Ridge Regression
  - Combining L1 and L2
    - Elastic Net

Regularization

- L1 regularization, also known as Lasso regularization, adds a penalty to the loss function that is equal to the absolute value of the magnitude of the coefficients. This penalty has several important effects on the model:
  - ❑ **Coefficient Shrinkage:** L1 regularization encourages the coefficients associated with less important features to be reduced toward zero. This means that the model becomes less reliant on certain features when making predictions, effectively shrinking the magnitude of these coefficients.
  - ❑ **Sparsity:** One of the distinctive features of L1 regularization is that it can yield sparse models. Sparse models have the property that some of the coefficients are exactly equal to zero, effectively eliminating those features from the model. This can be particularly useful in feature selection, where irrelevant or redundant features are automatically pruned from the model, leading to a more interpretable and efficient model.
  - ❑ **Feature Selection:** Because L1 regularization can lead to sparse models, it naturally performs feature selection by identifying and favoring the most important features while reducing the impact of less important ones. This is valuable in situations where you have a large number of features but only a subset of them are truly relevant.
- L1 regularization is a powerful technique that not only helps limit the size of coefficients and reduce overfitting but also has the added benefit of performing automatic feature selection by setting some coefficients to zero. It's particularly useful when dealing with high-dimensional datasets and can lead to simpler and more interpretable models.

L1 Regularization-Lasso

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|$$

L2 Regularization

- L2 regularization, which is also known as Ridge regularization. L2 regularization adds a penalty to the loss function that is equal to the square of the magnitude of the coefficients. This penalty has several important characteristics:

1. **Coefficient Shrinkage:** L2 regularization encourages all coefficients to be reduced in magnitude, but not necessarily to zero. It does this by adding a term to the loss function that depends on the sum of the squared coefficients. As a result, large coefficients are penalized more, and the model tends to prefer smaller coefficients.
2. **Uniform Shrinkage:** One key feature of L2 regularization is that it applies uniform shrinkage to all coefficients. In other words, it doesn't favor the elimination of specific coefficients. Instead, it reduces their impact proportionally. This can help prevent overfitting by discouraging the model from relying too heavily on any one feature.
3. **Continuous Variable Selection:** Unlike L1 regularization (Lasso), which can lead to sparse models with some coefficients exactly equal to zero, L2 regularization tends to produce models where all coefficients are non-zero but reduced in magnitude. This is often referred to as "continuous variable selection," as it retains all features while moderating their influence.
4. **Regularization Strength:** The strength of L2 regularization is controlled by a hyperparameter often denoted as α (or λ in some contexts). Increasing α increases the strength of regularization, which results in greater coefficient shrinkage.

- L2 regularization is a valuable tool for preventing overfitting and improving the generalization performance of machine learning models. It's commonly used in linear regression, logistic regression, and various other algorithms.

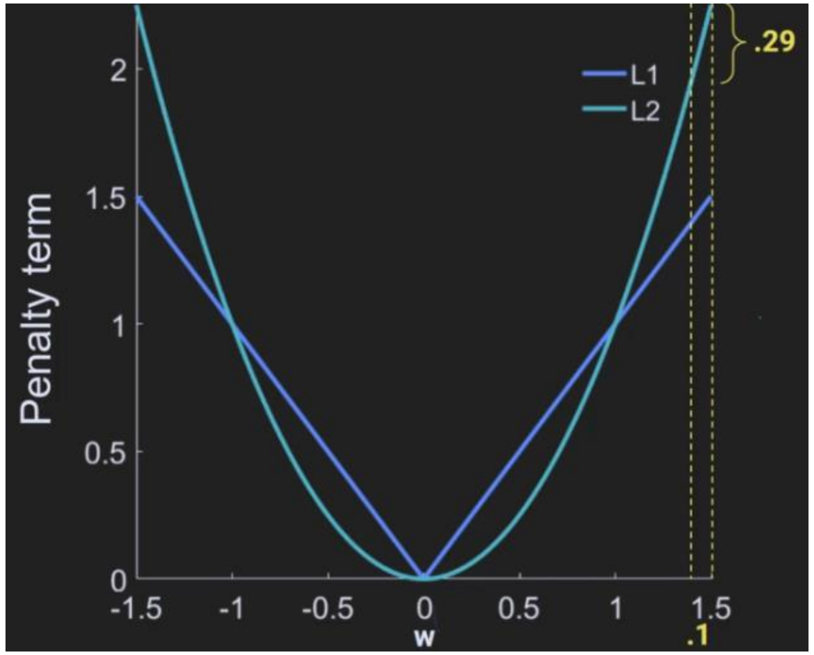
L2 Regularization

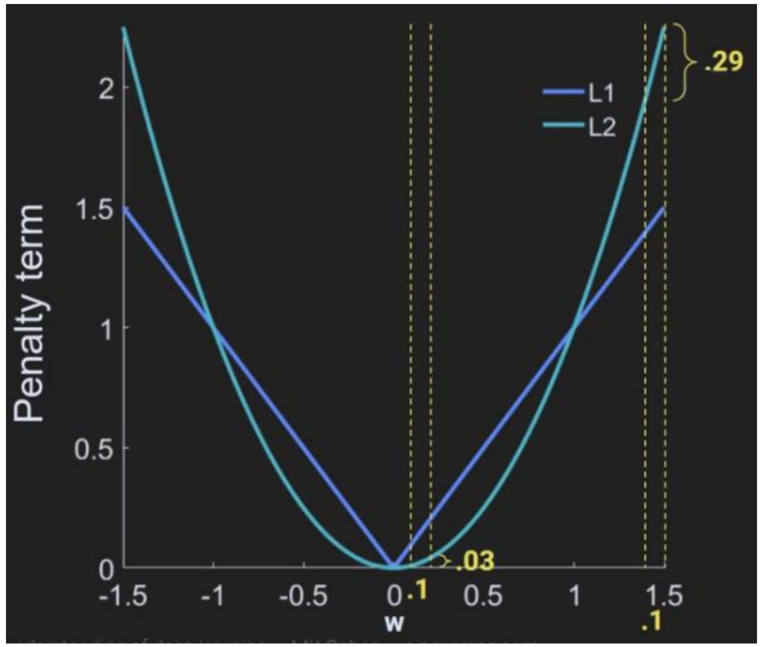
$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS} + \lambda\sum_{j=1}^{p}\beta_j^2$$

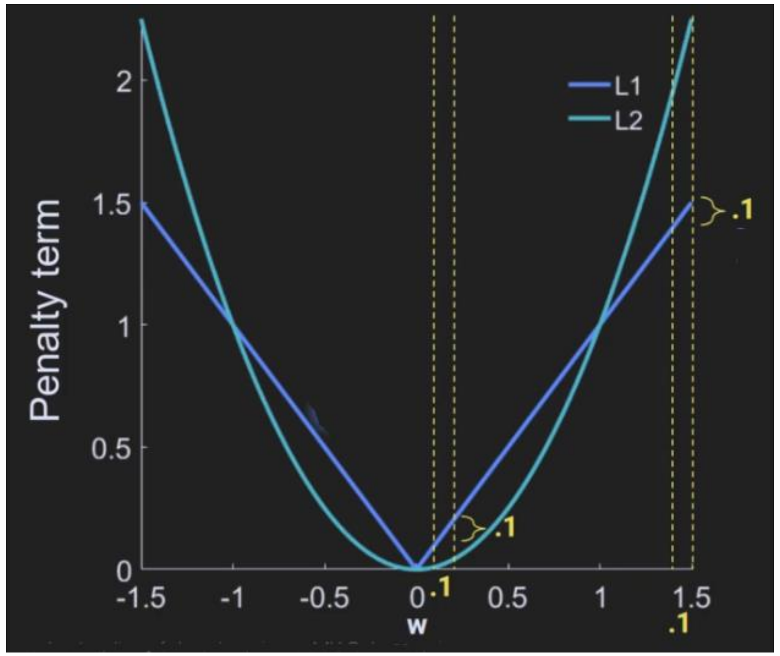
Elastic net Regularization

$$\frac{\sum_{i=1}^{n}(y_i - x_i^J \hat{\beta})^2}{2n} + \lambda \left(\frac{1-\alpha}{2} \sum_{j=1}^{m} \hat{\beta}_j^2 + \alpha \sum_{j=1}^{m} |\hat{\beta}_j| \right)$$

L2 & L1 Regularization







Mini-Batch Theoretical Framework

- ❑ Mini-batching operates on the premise of dividing the training data into smaller subsets, often referred to as batches or mini-batches.

- **Batch vs. Mini-Batch**:
  - ❑ To distinguish between a batch and a mini-batch, we consider the entire training dataset as a batch.
  - ❑ In contrast, a mini-batch represents a smaller, non-overlapping slice of the data. The size of the mini-batch is often chosen as powers of two (e.g., 16, 32, 64) for computational efficiency.
- **Computational Efficiency**:
  - ❑ Mini-batching leverages matrix multiplications, enhancing computational efficiency by exploiting vectorization. This optimization reduces the training time by performing operations on multiple samples simultaneously.
- **Regularization**:
  - ❑ Mini-batching serves as a form of regularization. Averaging the loss over several samples smooths the learning process, reducing overfitting.
  - ❑ However, for some datasets and model architectures, setting the mini-batch size to one (true stochastic gradient descent) may accelerate training by providing faster convergence.

Advantages of Mini-Batching

1. **Efficient Computation**: Mini-batching enables matrix-based computations, optimizing computational efficiency while training deep learning models.
2. **Regularization**: Averaging losses over mini-batches helps prevent overfitting, improving model generalization.
3. **Balanced Learning**: Mini-batching offers a compromise between detailed feedback for each sample (batch size of one) and computational efficiency (batch size equal to the full dataset).

Mini-Batching Practical Implementation:

- The practical implementation of mini-batching in Python, especially with PyTorch, is a critical aspect of deep learning. While the theoretical foundation is essential, the following are key considerations in practice:
- **Batch Size Selection**: Choosing an appropriate mini-batch size depends on the dataset and model architecture. Common choices are powers of two, but it's not a strict rule.
- **Matrix Multiplication**: Utilizing libraries like PyTorch, which support efficient matrix multiplications, is essential for reaping the benefits of mini-batching.
- **Regularization**: Understanding that mini-batching inherently acts as a form of regularization helps in designing better models.

Feature Scaling

- Feature scaling is an essential preprocessing step in many machine learning workflows, and it offers several benefits to the modeling process. It involves transforming the features (variables) in your dataset to bring them into a similar scale or range. The main ideas behind feature scaling are as follows:

1. **Equalizes the Influence of Features:** Feature scaling ensures that all features contribute equally to the modeling process. When features have different scales or units, those with larger magnitudes can dominate the learning process and have a disproportionate influence on the model's decisions. Scaling mitigates this issue, allowing each feature to contribute more evenly.
2. **Improves Convergence:** In iterative optimization algorithms, such as gradient descent, feature scaling can help the algorithm converge faster and more reliably. When features have vastly different scales, the optimization process can take longer to find the optimal solution or may become unstable. Scaling helps the optimization process by providing a more balanced landscape.
3. **Enhances Interpretability:** Scaling makes it easier to interpret the importance of coefficients or feature contributions in models like linear regression. With scaled features, the magnitude of the coefficients directly reflects their impact on the target variable, and it's easier to compare their importance.
4. **Facilitates Distance-Based Metrics:** Machine learning models that rely on distance metrics, such as k-Nearest Neighbors (KNN), are particularly sensitive to feature scaling. Without scaling, features with larger ranges or magnitudes can disproportionately affect the distance calculations, potentially leading to suboptimal results. Scaling ensures that all features have a similar influence on distance-based computations.

Feature Scaling Benefits

- ❖ **Great Increases in Performance:** Feature scaling can indeed lead to significant improvements in model performance. By ensuring that features are on a similar scale and reducing the influence of outliers, scaling can help models converge faster and make more accurate predictions.

When to do feature scaling

- ❖ **Necessary for Some Models:** Feature scaling is essential for certain machine learning models, particularly those that rely on distance metrics like K-Nearest Neighbors (KNN) and clustering algorithms like K-Means. These models can perform poorly without feature scaling, as they are sensitive to the magnitudes of features.

- ❖ **Virtually No "Real" Downside:** In general, there are very few downsides to scaling features. While some might argue that feature scaling introduces a slight loss of interpretability (since the scaled features are no longer in their original units), this is a minor concern compared to the benefits gained in terms of model performance and reliability.

Feature Scaling

- Two main ways to scale features:
  - *Standardization*

    change distribution

    $$X_{changed} = \frac{X - \mu}{\sigma}$$

    - Rescales data to have a mean (**μ**) of 0 and standard deviation (**σ**) of 1.
  - *Normalization*

    $$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

    - Rescales all data values to be between 0-1.

Feature Scaling Scikit-Learn

1. **.fit() Method:** In this step, the method calculates and computes the necessary statistics or parameters from the data that will be used for the transformation. For feature scaling, this typically involves calculating statistics like the minimum (Xmin), maximum (Xmax), mean, and standard deviation of the features.

2. **.transform() Method:** After the necessary statistics are calculated during the fitting step, the .transform() method applies the scaling or transformation to the data. It uses the computed statistics to transform the data according to a specific scaling method (e.g., min-max scaling, standardization) and returns the transformed version of the data.

- This separation of fitting and transforming is a common pattern in many preprocessing techniques and allows you to reuse the same transformation parameters on different datasets or apply the same transformation consistently to training and testing datasets. This approach ensures that the same scaling is applied to both training and testing data, maintaining data consistency.

Feature Scaling Some Important Notes

- ✓ **Fitting to Training Data:** When you perform any preprocessing step, such as calculating statistics for feature scaling, it's essential to fit the transformation only to the training data.
  - ❑ This means that you compute the necessary statistics (e.g., mean, standard deviation, minimum, maximum) using the training dataset alone.
  - ❑ The reason for this is to prevent any knowledge or information about the test data from leaking into the training process.
- ✓ **Avoiding Data Leakage:** Using the full dataset (both training and testing) for fitting can lead to data leakage.
  - ❑ Data leakage occurs when information from the test set inadvertently influences the modeling process.
  - ❑ This can result in overly optimistic model performance estimates during training, leading to poor generalization when the model encounters unseen data.
- ✓ **Maintaining Independence:** The test set should represent completely independent and unseen data that the model has not been exposed to during training.
  - ❑ By fitting preprocessing steps only to the training data, you ensure that the model doesn't rely on any information specific to the test set, which preserves the integrity of your model evaluation.

Feature Scaling-Target Variable??

- In most cases, it is not necessary or advised to scale the label (target variable) in regression problems. Scaling the label can introduce several complications and is generally not a common practice for the following reasons:

1. **Altering the Definition of the Target:** Scaling the label changes the meaning of the target variable. For example, if you scale a house price label, it will no longer represent the actual price of houses but rather a scaled version of it. This makes interpretation and communication of results more challenging.

2. **Predicting a Different Distribution:** When you scale the target, you're effectively transforming it to a different distribution. This can lead to confusion when comparing model predictions to the original, unscaled target values.

3. **Loss of Interpretability:** Scaling the target makes it more difficult to interpret the model's coefficients. In linear regression, for instance, the coefficients represent the change in the target variable for a one-unit change in the corresponding feature. Scaling the target would mean that the coefficients no longer have their original interpretation.

4. **Evaluation Metrics:** Common evaluation metrics for regression, such as Mean Absolute Error (MAE) and Mean Squared Error (MSE), are designed to work with unscaled targets. If you scale the target, you would need to use different metrics that account for the scaling to assess model performance accurately.

Ridge Regression

- Ridge Regression is a regularization technique that works by helping reduce the potential for overfitting to the training data.
- It does this by adding in a penalty term to the error that is based on the squared value of the coefficients.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_p x_p$$

$$\mathrm{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Ridge Regression

$$\begin{aligned}\mathrm{RSS} &= \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \\ &= \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} - \cdots - \hat{\beta}_p x_{ip})^2\end{aligned}$$

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2$$

Ridge Regression

- Ridge Regression adds a **shrinkage penalty**:

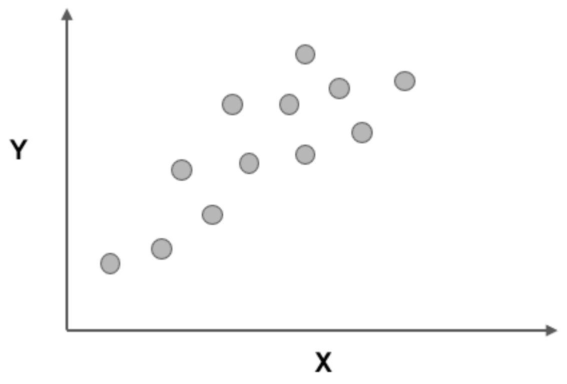
$$\text{Error} \quad \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 \boxed{+\lambda\sum_{j=1}^{p}\beta_j^2}$$

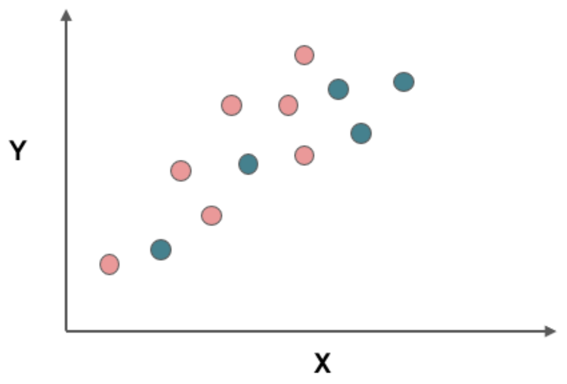
- Ridge Regression seeks to minimize this entire error term **RSS + Penalty**.

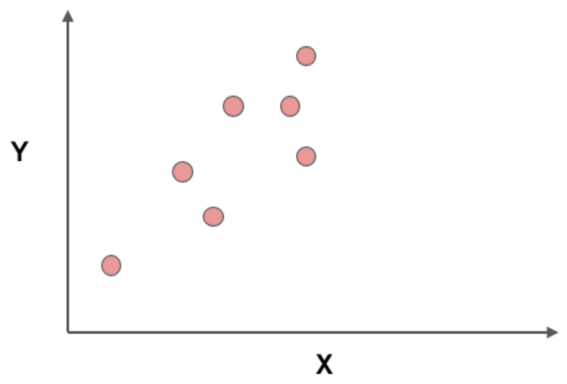
- **Shrinkage penalty** has a **tunable lambda parameter!**

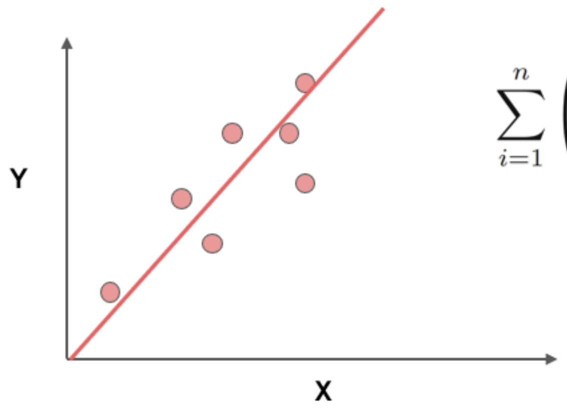
Ridge Regression

$\hat{y} = \beta_1 x + \beta_0$

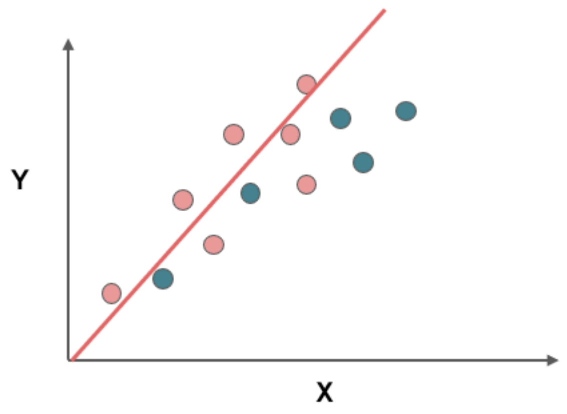


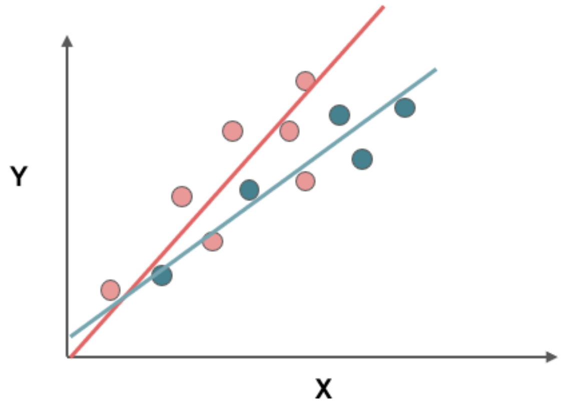




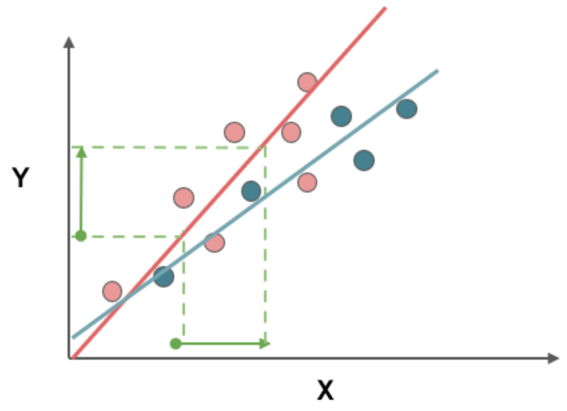


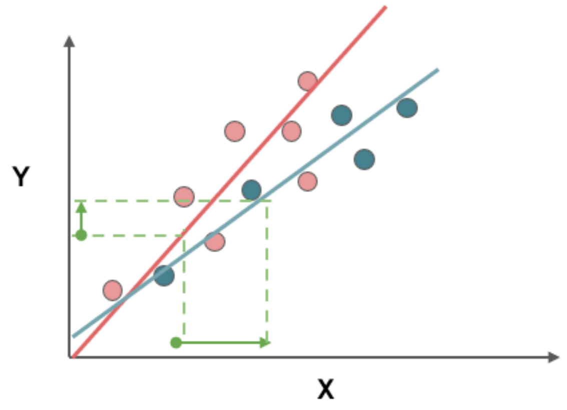
$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2$$





Ridge Regression





Lasso Regression

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \text{RSS} + \boxed{\lambda\sum_{j=1}^{p}|\beta_j|}$$

- LASSO can force some of the coefficient estimates to be exactly equal to zero when the tuning parameter λ is sufficiently large.
- Similar to subset selection, the LASSO performs variable selection.
- Models generated from the LASSO are generally much easier to interpret.

Regularization

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \le s \tag{6.8}$$

and

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \le s, \tag{6.9}$$

Hyperparameters (meta parameters) and model parameters

- **Understanding the Distinction**:
- **Model Parameters**:
  - ❑These encompass the weights and biases within a deep learning model. Model parameters are learned during training and adapt to the data.
- **Meta-Parameters**:
  - ❑Unlike model parameters, meta-parameters are predefined characteristics of the model. They are not learned from data but are specified by the practitioner. Meta-parameters play a critical role in shaping the model's behavior.

Hyperparameters in Deep Learning Exploring Meta-Parameters

1. **Model Architecture**: Defines the structure of the neural network, including the number of hidden layers and nodes per layer.
2. **Cross-Validation Size**: Determines the size of data splits for cross-validation, influencing model evaluation and generalization.
3. **Optimization Functions**: The choice of optimization algorithm significantly affects training speed and convergence.
4. **Weight Normalization and Initializations**: How weights are initialized and whether weight normalization techniques are applied can impact training dynamics.
5. **Data Normalization**: Techniques for preprocessing input data, such as mean-centering and scaling, play a role in model stability.

The Challenge of Meta-Parameters

- ❑The abundance of meta-parameters introduces a unique challenge—exploring the vast meta-parameter space is impractical.
- ❑With countless combinations and settings, it's impossible to exhaustively search for the optimal configuration.

➢**However, all hope is not lost.**

- ❑Practical strategies include running parametric experiments to explore key meta-parameters.
- ❑Additionally, building intuition through experience, learning from successes, and staying informed through educational resources contribute to making informed meta-parameter decisions.

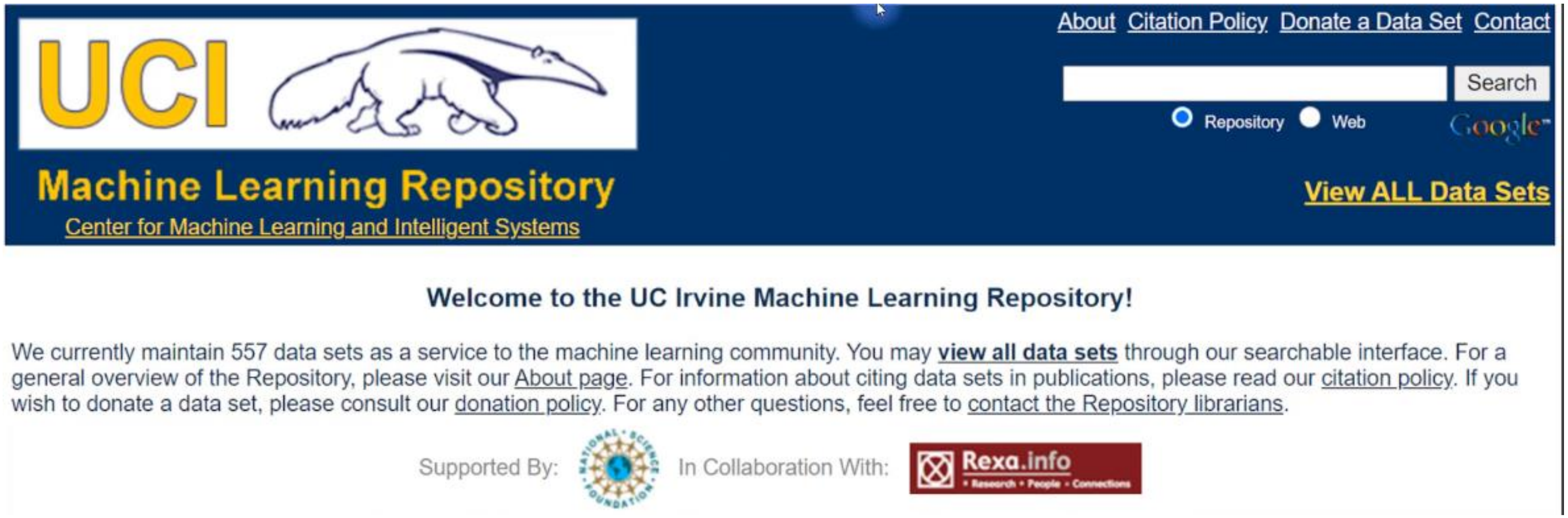
The Challenge of Meta-Parameters

- ❑The abundance of meta-parameters introduces a unique challenge—exploring the vast meta-parameter space is impractical.
- ❑With countless combinations and settings, it's impossible to exhaustively search for the optimal configuration.

➢**However, all hope is not lost.**

- ❑Practical strategies include running parametric experiments to explore key meta-parameters.
- ❑Additionally, building intuition through experience, learning from successes, and staying informed through educational resources contribute to making informed meta-parameter decisions.

Wine Quality Dataset



Welcome to the UC Irvine Machine Learning Repository!

We currently maintain 557 data sets as a service to the machine learning community. You may **view all data sets** through our searchable interface. For a general overview of the Repository, please visit our About page. For information about citing data sets in publications, please read our citation policy. If you wish to donate a data set, please consult our donation policy. For any other questions, feel free to contact the Repository librarians.