Deep Learning

Assigment-1

1.Early Stopping

A very different way to regularize iterative learning algorithms such as Gradient Descent is to stop training as soon as the validation error reaches a minimum. This is called early stopping. Figure 4-20 shows a complex model (in this case a high-degree Polynomial Regression model) being trained using Batch Gradient Descent. As the epochs go by, the algorithm learns and its prediction error (RMSE) on the training set naturally goes down, and so does its prediction error on the validation set. However,after a while the validation error stops decreasing and actually starts to go back up.This indicates that the model has started to overfit the training data. With early stop-ping you just stop training as soon as the validation error reaches the minimum. It is such a simple and efficient regularization technique that Geoffrey Hinton called it a “beautiful free lunch.”

Fig:Early stopping regularization

With Stochastic and Mini-batch Gradient Descent, the curves are not so smooth, and it may be hard to know whether you have reached the minimum or not. One solution is to stop only after the validation error has been above the minimum for some time (when you are confident that the model will not do any better), then roll back the model parameters to the point where the validation error was at a minimum.

Here is a basic implementation of early stopping:

```
from sklearn.base import clone

# prepare the data

poly_scaler = Pipeline([

  ("poly_features", PolynomialFeatures(degree=90, include_bias=False)),

  ("std_scaler", StandardScaler())

  ])

X_train_poly_scaled = poly_scaler.fit_transform(X_train)

X_val_poly_scaled = poly_scaler.transform(X_val)

sgd_reg = SGDRegressor(max_iter=1, tol=-np.infty, warm_start=True,

  penalty=None, learning_rate="constant", eta0=0.0005)

minimum_val_error = float("inf")

best_epoch = None

best_model = None

for epoch in range(1000):

sgd_reg.fit(X_train_poly_scaled, y_train) # continues where it left off

  y_val_predict = sgd_reg.predict(X_val_poly_scaled)

  val_error = mean_squared_error(y_val, y_val_predict)

  if val_error < minimum_val_error:

  minimum_val_error = val_error

  best_epoch = epoch

  best_model = clone(sgd_reg)
```

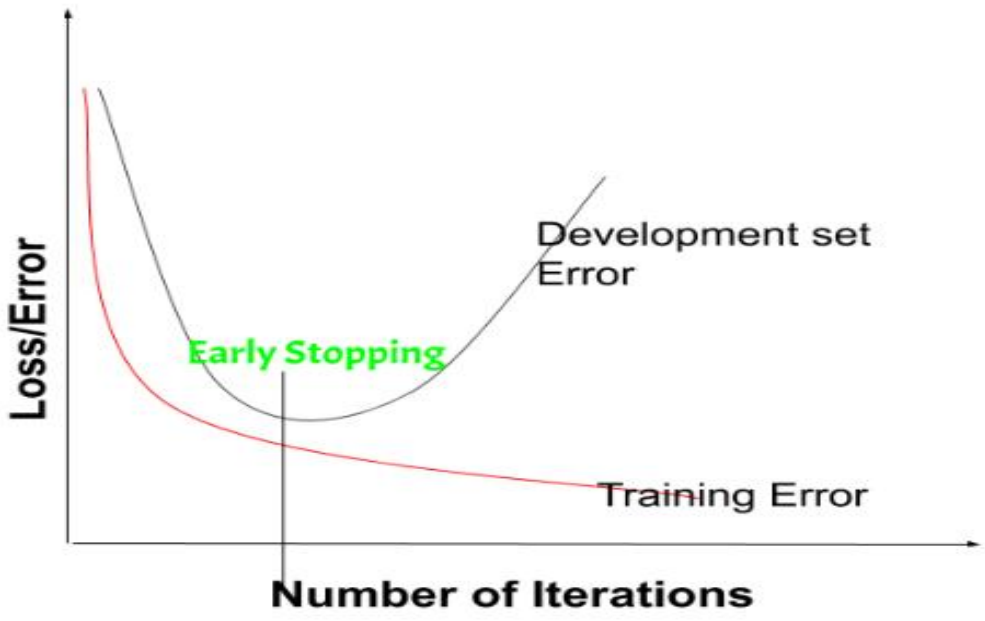
Note that with warm_start=True, when the fit() method is called, it just continues

training where it left off instead of restarting from scratch.

What is Early Stopping?

Early Stopping, we stop training the model when the performance on the validation set is getting worse- increasing loss decreasing accuracy, or poorer scores of the scoring metric.By plotting the error on the training dataset and the validation dataset together, both the errors decrease with a number of iterations until the point where the model starts to overfit. After this point, the training error still decreases but the validation error increases.

So, even if training is continued after this point, early stopping essentially returns the set of parameters that were used at this point and so is equivalent to stopping training at that point. So, the final parameters returned will enable the model to have low variance and better generalization. The model at the time the training is stopped will have a better generalization performance than the model with the least training error.



Early stopping can be thought of as implicit regularization, contrary to regularization via weight decay. This method is also efficient since it requires less amount of training data, which is not always available. Due to this fact, early stopping requires lesser time for training compared to other regularization methods. Repeating the early stopping process many times may result in the model overfitting the validation dataset, just as similar as overfitting occurs in the case of training data.

The number of iterations(i.e. epoch) taken to train the model can be considered a hyperparameter. Then the model has to find an optimum value for this hyperparameter (by hyperparameter tuning) for the best performance of the learning model.

Benefits of Early Stopping:

Helps in reducing overfitting

It improves generalisation

It requires less amount of training data

Takes less time compared to other regularisation models

It is simple to implement

Limitations of Early Stopping:

If the model stops too early, there might be risk of underfitting

It may not be beneficial for all types of models

If validation set is not chosen properly, it may not lead to the most optimal stopping

2.Feature Engineering in Deep Learning

Feature engineering is a crucial step in the machine learning pipeline, including deep learning. While deep learning models can automatically learn complex representations from raw data, effective feature engineering can still significantly improve model performance, reduce training time, and enhance interpretability. Here are some considerations for feature engineering in the context of deep learning:

1.Normalization and Standardization:

Scale numerical features to a similar range. Standardization (subtracting the mean and dividing by the standard deviation) or normalization (scaling to a specific range, e.g., [0, 1]) helps prevent certain features from dominating the learning process.

2.**Handling Categorical Variables:**

Convert categorical variables into a suitable format for neural networks. This can involve one-hot encoding, label encoding, or using embeddings for high-cardinality categorical features.

3.Missing Data Handling:

Deal with missing data appropriately. Imputation techniques, such as filling missing values with mean, median, or using advanced methods like K-nearest neighbors imputation, can be employed.

4.Feature Scaling:

Scaling features can be crucial, especially for algorithms sensitive to the scale of input features. This ensures that the optimization process during training is more stable.

5.Embedding:

For categorical variables, especially those with high cardinality, consider using embedding. Embedding can be learned from the data and can provide a more compact representation, often capturing semantic relationships.

6.Dimensional Reduction:

Use dimensional reduction techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce the number of features while preserving essential information.

7.Feature Interaction:

Create new features that capture interactions between existing features. This can involve combining or transforming features to expose non-linear relationships that the model might struggle to learn on its own.

8.Data Augmentation:

For image data, data augmentation techniques (such as rotation, flipping, zooming) can be used to artificially increase the size of the training dataset and improve the model's ability to generalize.

9.Text Data Processing:

For natural language processing tasks, tokenize and preprocess text data, remove stop words, and consider techniques like TF-IDF or word embeddings (e.g., Word2Vec, GloVe) to represent words.

What is Feature Engineering?

Feature Engineering is the process of creating new features or transforming existing features to improve the performance of a machine-learning model. It involves selecting relevant information from raw data and transforming it into a format that can be easily understood by a model. The goal is to improve model accuracy by providing more meaningful and relevant information.

Feature engineering is the process of transforming raw data into features that are suitable for machine learning models. In other words, it is the process of selecting, extracting, and transforming the most relevant features from the available data to build more accurate and efficient machine learning models.

The success of machine learning models heavily depends on the quality of the features used to train them. Feature engineering involves a set of techniques that enable us to create new features by combining or transforming the existing ones. These techniques help to highlight the most important patterns and relationships in the data, which in turn helps the machine learning model to learn from the data more effectively.

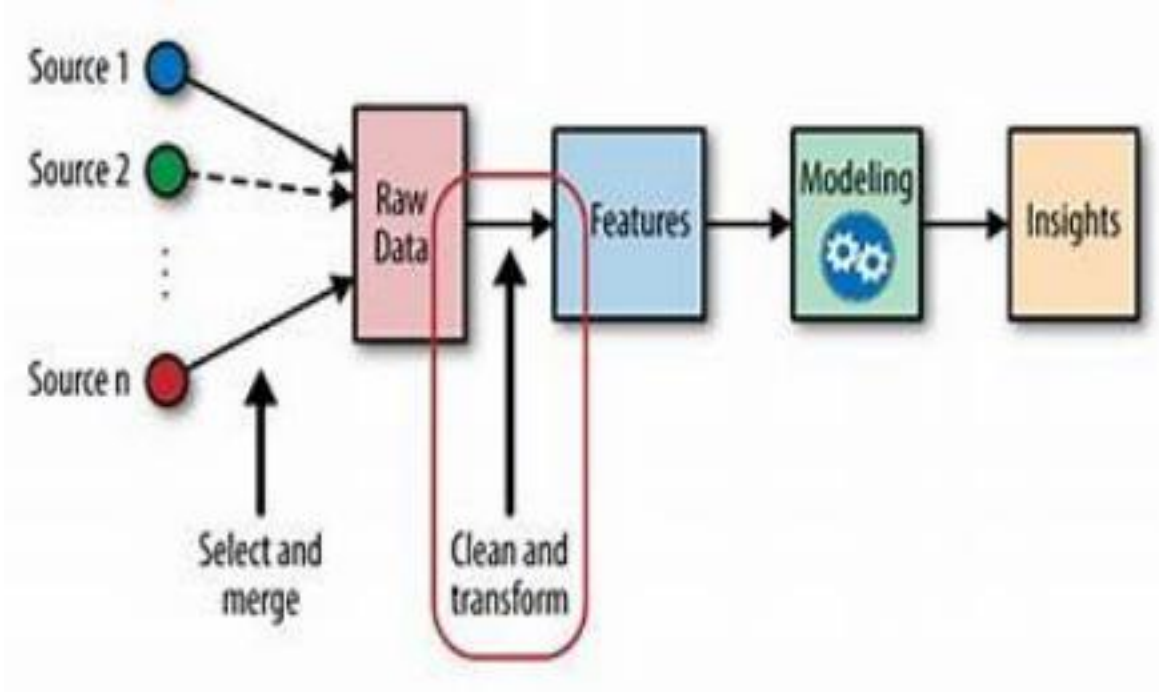


Fig:Feature Engineering

Need for Feature Engineering in Deep Learning

We engineer features for various reasons, and some of the main reasons include:

Improve User Experience:

The primary reason we engineer features is to enhance the user experience of a product or service. By adding new features, we can make the product more intuitive, efficient, and user-friendly, which can increase user satisfaction and engagement.

Competitive Advantage:

Another reason we engineer features is to gain a competitive advantage in the marketplace. By offering unique and innovative features, we can differentiate our product from competitors and attract more customers.

Meet Customer Needs:

We engineer features to meet the evolving needs of customers. By analyzing user feedback, market trends, and customer behavior, we can identify areas where new features could enhance the product's value and meet customer needs.

Increase Revenue:

Features can also be engineered to generate more revenue. For example, a new feature that streamlines the checkout process can increase sales, or a feature that provides additional functionality could lead to more upsells or cross-sells.

Future-Proofing:

Engineering features can also be done to future-proof a product or service. By anticipating future trends and potential customer needs, we can develop features that ensure the product remains relevant and useful in the long term.

Processes Involved in Feature Engineering

Feature engineering in Machine learning consists of mainly 5 processes: Feature Creation, Feature Transformation, Feature Extraction, Feature Selection, and Feature Scaling. It is an iterative process that requires experimentation and testing to find the best combination of features for a given problem. The success of a deep learning model largely depends on the quality of the features used in the model.

1. Feature Creation

Feature Creation is the process of generating new features based on domain knowledge or by observing patterns in the data. It is a form of feature engineering that can significantly improve the performance of a machine-learning model.

Types of Feature Creation:

Domain-Specific: Creating new features based on domain knowledge, such as creating features based on business rules or industry standards.

Data-Driven: Creating new features by observing patterns in the data, such as calculating aggregations or creating interaction features.

Synthetic: Generating new features by combining existing features or synthesizing new data points.

2. Feature Transformation

Feature Transformation is the process of transforming the features into a more suitable representation for the machine learning model. This is done to ensure that the model can effectively learn from the data.

Types of Feature Transformation:

Normalization: Rescaling the features to have a similar range, such as between 0 and 1, to prevent some features from dominating others.

Scaling: Scaling is a technique used to transform numerical variables to have a similar scale, so that they can be compared more easily. Rescaling the features to have a similar scale, such as having a standard deviation of 1, to make sure the model considers all features equally.

Encoding: Transforming categorical features into a numerical representation. Examples are one-hot encoding and label encoding.

Transformation: Transforming the features using mathematical operations to change the distribution or scale of the features. Examples are logarithmic, square root, and reciprocal transformations.

3. Feature Extraction

Feature Extraction is the process of creating new features from existing ones to provide more relevant information to the machine learning model. This is done by transforming, combining, or aggregating existing features.

Types of Feature Extraction:

Dimensionality Reduction: Reducing the number of features by transforming the data into a lower-dimensional space while retaining important information. Examples are PCA and t-SNE.

Feature Combination: Combining two or more existing features to create a new one. For example, the interaction between two features.

Feature Aggregation: Aggregating features to create a new one. For example, calculating the mean, sum, or count of a set of features.

Feature Transformation: Transforming existing features into a new representation. For example, log transformation of a feature with a skewed distribution.

4. Feature Selection

Feature Selection is the process of selecting a subset of relevant features from the dataset to be used in a machine-learning model. It is an important step in the feature engineering process as it can have a significant impact on the model's performance.

Types of Feature Selection:

Filter Method: Based on the statistical measure of the relationship between the feature and the target variable. Features with a high correlation are selected.

Wrapper Method: Based on the evaluation of the feature subset using a specific machine learning algorithm. The feature subset that results in the best performance is selected.

Embedded Method: Based on the feature selection as part of the training process of the machine learning algorithm.