Biogen

Deep Learning Best Practices

Author List: Jake Gagnon, Hangyu (Cedric) Liu, Lira Pi, Lukasz Kniola, Himanshu Pandya, Jeff Thompson

 [1]

[1] Cover image created with DALL·E 3 by OpenAI

Table of Contents

Introduction/Scope

Deep Learning methods have been quite popular in recent years in areas such as Computer Vision, Natural Language Processing, and Generative AI. In this white paper, we hope to guide deep learning practitioners in industry best practices and introduce the core concepts and architectures of deep learning. The manuscript is organized as follows: we begin by providing recommendations for exploratory data analysis, data pre-processing, and data governance. After these preliminaries, we review data splitting/cross validation techniques to reduce overfitting of the data and describe common neural network architectures. We then discuss common techniques for model evaluation and data augmentation. Since pretrained models may not have acceptable performance for your specific domain of interest, we discuss various techniques to improve model performance such as transfer learning or fine-tuning. Lastly, we discuss other deep learning pipeline considerations such as model interpretation, model reproducibility, and deployment of models into production.

Chapter 1: Exploratory Data Analysis

Assess Data Bias

There are multiple types of biases which can occur during the data gathering process. Selection bias occurs when the sample is not randomly selected from the population but influenced by external factors such as convenience, availability, and preference. A non-response bias occurs when the sample is randomly selected, but some of the participants do not respond or drop out of the study, creating a critical difference between the respondents and the non-respondents. A third type of bias is measurement (technical) bias. This occurs when the data is collected or recorded in a way that introduces errors or inaccuracies, such as using faulty instruments, ambiguous questions, or subjective judgments. Another type of bias that should be avoided is survivorship bias. This type of bias occurs when the sample only includes the survivors or the winners of a process, ignoring those who failed or dropped out. Lastly, we have confirmation bias which occurs when the analyst interprets or presents the data in a way that confirms their pre-existing beliefs or expectations, rather than being objective and open-minded. If any of the above biases are expected to be present in your data, correction of the biases is highly recommended prior to model training.

Assess Representativeness

Check that your gathered data is representative of your desired target population. This includes checks such as demographic tables to validate that your gathered data represents the demographic composition (gender, race, ethnicity) of your target population.

Assess Data Balance

It’s very crucial and of paramount importance to have balanced data during data analysis and machine learning. Balanced data means you have the same number of samples/individuals for each group in a classification dataset. Using imbalanced data when training a classifier can cause a bias in favor of the majority class and can lead to incorrect predictions, so having a balanced dataset is highly recommended. See “Imbalanced data strategies” section below for potential solutions.

Assess FAIR Principles

FAIR data are data which meet principles of findability, accessibility, interoperability, and reusability (Wilkonson et al. 2016). The FAIR principles emphasize machine-actionability (i.e., the capacity of computational systems to find, access, interoperate, and reuse data with none or minimal human intervention) because humans increasingly rely on computational support to process/analyze due to the increase in volume, complexity, and the computational time to analyze data. The FAIR Data Principles are a set of guiding principles proposed by a consortium of scientists and organizations to support the reusability of digital assets. computational time to analyze data. The FAIR Data Principles are a set of guiding principles proposed by a consortium of scientists and organizations to support the reusability of digital assets.

Assess Batch Effects

Another important aspect of data exploration is the assessment of batch effects. “Batch” effects include the effects of sample run, time of day, day of the week, experimenter, reagent batch, location, etc. To assess your data for batch effects, visual displays such as PCA, density plots, boxplots, RLE plots, or heatmaps can be constructed and interpreted. For example, in PCA, a dataset with a batch effect will show a clear separation of batches in PCA space. On the other hand, if the batches are well-mixed in PCA space then the batch effect is minimal. If a significant batch effect is detected, then the analyst should consider batch effect correction methods such as: 1) regressing out a batch covariate in a linear regression or linear mixed model, 2) surrogate variables analysis (sva) which is a common method in genomics, 3) ComBat, 4) batch mean centering, or 5) SVD based approaches.
(See https://evayiwenwang.github.io/Managing_batch_effects/index.html#study-description for more details)

References

1. Wilkinson, M., Dumontier, M., Aalbersberg, I. *et al.* The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* **3**, 160018 (2016).

Chapter 2: Data preprocessing

Database Normalization

Database normalization is the process of reorganizing data within a database so that users can utilize it for further queries and analysis. It is one of the popular processes for developing clean data, which includes eliminating redundant and unstructured data and making the data appear similar across all records and fields. The main objective of database normalization is to eliminate redundant data, minimize data modification errors, and simplify the query process for better data query/access.

Ultimately, normalization goes beyond simply standardizing data, and can even improve your workflow, increase security, and lessen costs. When data is normalized, the entire database is more logically structured, leading to an ease of understanding and use. This uniformity ensures that teams can collaborate more effectively, as everyone is working with the same consistent information. Also, by eliminating redundant data, the size of files are reduced, which speeds up both analysis and data processing. This efficiency is a boon for cost-saving, as it means quicker access to data and reduced requirements for extensive storage and processing capacities. Moreover, a well-organized database is inherently more secure, as data is precisely located and uniformly maintained, adding an extra layer of protection against security breaches. Overall, data normalization isn't just a cleanup task; it's an essential step towards making databases more functional, secure, and cost-effective.

Statistical Normalization

Data normalization, or rescaling of the data to a specified range, is an essential step to biologically interpret your data. For example, in genomics, the total RNA content can vary from cell to cell. Consequently, to enable fair comparisons across cells, we need to adjust (normalize) each cell's gene expression with respect to its total RNA content. For a second example, to enable fair sample to sample comparisons in qPCR gene expression measurements, a reference gene (control) is utilized to normalize the gene expressions of each sample. In general statistics, normalization is typically a min-max re-scale of values between 0 and 1.

Data Scaling

Data scaling is the process of transforming the values of the features of a dataset until they are within a specific range, i.e. 0 to 1 or -1 to 1. One common technique of standardization is z-scaling (i.e. centering and scaling) to a standard normal distribution of mean 0 and standard deviation 1. This scaling is to ensure that no single feature dominates and can help improve the performance of a ML algorithm. As another example, in lasso regression, scaling allows us to fairly interpret the effects of each feature and aids in feature selection. However, some ML methods are not sensitive to the magnitude of data (e.g. tree-based algorithms) so standardization is not needed for them.

Data Cleaning

Data cleaning is important because it ensures data quality. This can prevent errors, improve decision-making, and increase productivity. Data cleaning techniques are essential to getting accurate results when you analyze data for various purposes.

There are many methods that can be used to improve data quality. The following are some common and recommended approaches: removing duplicates, removing irrelevant data, converting data types, and handling outliers. It is important to remove duplicates to maintain data quality. Duplicates not only represent a data storage problem but can also affect the accuracy and processing time of the model. Removing duplicates sometimes involves merging several partial records together, while other times it involves deletion of the duplicate records.

Another important method is to remove irrelevant data. Similarly, to duplicate records, irrelevant data increases the storage demands for your data, will increase processing costs, and potentially decrease output accuracy.

Data type conversion can be needed when the source of the data differs from how it is stored. When this method is used properly, it ensures all data will be in an easily storable format, can be easily processed, and maintains the data with minimal or no loss of information.

Lastly, handling outliers is an important method to maintaining data quality. Depending on the robustness of your model, outliers can significantly bias the model results. Outliers often indicate data source issues such as data errors, a lack of a representative sample, or an exceptional data point. They can also indicate an issue with the current usage of the data. For example, it is common to apply a logarithmic scale to financial data; however, if no transformation was applied, the data quality issue could show up as an outlier.

Missingness/Imputation

The data missingness proportion should be assessed for each feature, for each subject, as well as the response variable. Additionally, ML practitioners should determine the missing data mechanism. If data missingness is observed, it can be addressed using a variety of different techniques. The following techniques are some recommendations for data imputation of missing data:

Mean/median

Imputation using the mean or median value is a straightforward approach that replaces all occurrences of missing values (NA) for a feature within a dataset by using the mean (if the variable has a Gaussian distribution) or the median (if the variable has a skewed distribution).

Multivariate Imputation by Chained Equations (MICE)

MICE performs imputation by creating several sets of data, each with imputed values based on specified prediction models for that feature. The imputation sets are chained, meaning that each subsequent dataset uses the prediction model from the previous one to obtain new values for each feature that contains missingness. The missing values are then updated across the full dataset based on the new fit in each iteration. There is a small amount of randomness introduced in the iterated predictions to attempt to avoid bias during the chaining procedure. The MICE procedure ends when the variable predictions have converged, and the set of MICE-generated data can then be pooled, including the imputed values. The R package mice can be used to perform MICE imputation and has documentation available on its usage. For more details, the package publication and example R codes can be found from https://www.jstatsoft.org/article/view/v045i03. There are also several Python implementations of MICE, such as scikit-learn IterativeImputer (see https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html for vignettes and example codes). The MICE approach can be used together with other techniques to optimize imputed values (for an example, see Random Forest imputation).

K-nearest neighbor (KNN)

KNN imputation employs the k-nearest neighbor algorithm in multi-dimensional space to impute missing values for a feature. Using the dataset present, the KNN approach will select the closest neighboring points using an arbitrary distance metric, which is typically the Euclidian distance. For a datapoint with missingness in a feature, KNN imputation will then assign an imputed value calculated from its neighbors for that feature. The assigned imputed value for that datapoint can be calculated using any arbitrary metric such as the mean, distance-weighted mean, or the mode. The optimization of the number of neighbors (k value) is important to avoid overfitting (low k) or noise (high k), and often requires tuning the value of k using cross-validation or other k selection strategies. KNN imputation is non-parametric and can be sensitive to outliers in the data and will tend not to provide reliable imputation values in generally sparse datasets or in datasets with regional sparsity. KNN imputation using the average value of the nearest k neighbors to impute missing features can be performed using several different R packages, such as caret, which has documentation on how to perform this method, utilizing caret::preProcess(..., method = “knnImpute”,...) function. For more details, see the documentation of CRAN webpage, https://cran.r-project.org/web/packages/caret/index.html. For the Python implementation, there are again several options such as scikit-learn function KNNImputer (find more details from https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html#sklearn.impute.KNNImputer), which can be used to perform KNN imputation.

Random Forest (RF)

Random Forest imputation uses a random sampling approach to create a set of decision trees to create the RF from existing data. Imputation is performed for the missing data within this RF by averaging the values for that feature. RF imputation does not require a specific parametric model, nor does it require an assumption of a normal distribution; however, this approach can still be sensitive to highly skewed distributions. There are R packages that can perform random forest imputation, such as missForest and

CALIBERrfimpute, which have documentation available for their usage. The missForest package is a popular choice for performing RF imputation using RF alone (for more details, see https://cran.r-project.org/web/packages/missForest/missForest.pdf). On the other hand, CALIBERrfimpute uses MICE-imputed datasets in concert with RF to optimize imputed values (for example codes, visit https://cran.r-project.org/web/packages/CALIBERrfimpute/CALIBERrfimpute.pdf). In Python, there are several implementations that can be used for RF imputation, such as the missingpy MissForest function (find example codes from https://pypi.org/project/missingpy/), and miceforest represents a Python package that can use MICE in concert with RF for imputation. See https://pypi.org/project/miceforest/ for more details on the package.

Choosing an imputation strategy

Deciding on which imputation strategy to employ will be highly dependent on a user's needs. Factors to consider include the data size format and sparsity, the computational resources available, and the missing data mechanism. A table is displayed below which may help in the decision-making process.

Factor	Random Forests	KNN	Median	MICE
Data Features				
- Size	Large (millions of rows)	Large (millions of rows)	Small to Medium (10-100k rows)	Large (millions of rows)
- Type	Mixed (categorical and numerical)	Mixed (categorical and numerical)	Numerical	Mixed (categorical and numerical)
- Sparsity	Less sensitive, can learn from similar patterns but accuracy may decrease	Sensitive, performance worsens significantly with higher sparsity, may overfit	Not recommended (assumes underlying distribution)	Moderately sensitive, requires careful parameter tuning and model selection
- Outliers	Moderately robust, internal trees can downweight outliers in decision making	Sensitive, distance-based approach can be misled by outliers, consider outlier detection beforehand.	Not recommended (can amplify outliers)	Moderately robust, multiple imputations can capture some outlying values, consider robust model selection within MICE.
- Linear relationships	Moderately preserves	Generally preserves for nearby neighbors	Not recommended	Moderately preserves through multiple imputations
- Non-linear relationships	Less effective, limited ability to capture complex interactions	Not effective, distance-based approach struggles with non-linearity	Not recommended	Can be more effective with appropriate model choices within MICE
Missing Value Type				
- Missing Completely At Random (MCAR)	Performs well in general	Performs well in general	Not recommended	Performs well in general

- Missing At Random (MAR)	Can perform well, may exhibit slight bias	Can perform well, consider advanced distance metrics	Not recommended	Moderately robust, careful model selection needed
- Missing Not At Random (MNAR)	Not recommended, may amplify biases	Not recommended, can be misled by missingness patterns	Not recommended	Limited effectiveness, requires additional techniques to address underlying mechanism
Model requirement	No (internal trees)	No (distance-based)	No	Yes
Ease of use	Medium	Medium	Easy	Complex
Implementation language	R (MissForest is robust and well-regarded), Python (miceforest is efficient)	Python or R (both have efficient implementations)	Python or R (both have simple implementations)	R (more mature packages and expertise)
Computational resources	High	Medium	Low	High
Scalability	Highly scalable due to parallelization options	Moderately scalable, efficient implementations available	Not recommended for large datasets	Highly scalable with parallelization and efficient mode choices

Table 1: Choosing an imputation strategy

References

1. https://medium.com/@Cambridge_Spark/tutorial-introduction-to-missing-data-imputation-4912b51c34eb.
2. Li, J., Guo, S., Ma, R. et al. Comparison of the effects of imputation methods for missing data in predictive modelling of cohort study datasets. BMC Med Res Methodol 24, 41 (2024). https://doi.org/10.1186/s12874-024-02173-x
3. Hong, S., Lynn, H.S. Accuracy of random-forest-based imputation of missing data in the presence of non-normality, non-linearity, and interaction. BMC Med Res Methodol 20, 199 (2020). https://doi.org/10.1186/s12874-020-01080-1
4. https://www.analyticsvidhya.com/blog/2022/05/handling-missing-values-with-random-forest/
5. https://towardsdatascience.com/missforest-the-best-missing-data-imputation-algorithm-4d01182aed3
6. https://cran.r-project.org/web/packages/finalfit/vignettes/missing.html

Considerations for Image Preprocessing

Prior to training a deep learning model on images, pre-processing of images is required. The exact pre-processing steps needed will depend on your problem domain. As a first step, inspect the input size of your neural network. If your image size doesn't match your input size, then your training data needs to be cropped or resized to match your input requirements. You should also inspect the training data for any illumination problems. For example, if the illumination is not constant across the image, then illumination correction will be needed. Additionally, you may want to consider conversion of your images to grayscale. In some computer vision tasks, color information may not be needed so a conversion to grayscale will make your deep learning model more computationally efficient. Other pre-processing steps such as whitening, gamma correction, histogram equalization, or removal of redundant images may be needed depending on your problem domain.

Another consideration in image processing is the stability and robustness of training the neural network. To improve stability, it is common to normalize your images with techniques such as mean image subtraction, per-channel normalization, or per-channel mean subtraction. As for robust model training, data augmentation is a common method to increase training data size and model robustness [See Data Augmentation in CV subsection below]. This includes operations such as flipping, rotating, adding noise, changing contrast, or varying exposure/brightness.

Additionally, model performance will be improved with high quality training images. The images should be in-focus and under a variety of real-life situations (images under different lighting conditions, different angles, and different distances). You also need to collect sufficient images for each image category.

References

1. https://medium.com/@saba99/preprocessing-techniques-182c8ac186f6
2. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8562352/
3. https://encord.com/blog/data-refinement-guide-computer-vision/
4. https://www.quora.com/What-kind-of-image-pre-processing-should-be-done-before-feeding-it-to-a-Convolutional-Neural-Network
5. https://medium.com/@davidfriml/how-to-prepare-images-for-a-training-dataset-f6889433249b

Considerations for Text Preprocessing

In the field of deep learning, preparing text data starts with collecting it from various sources, such as clinical operations, external regulatory agencies, publications, etc. This involves handling different file formats like CSV and JSON. The crucial step of cleaning the data involves removing unnecessary elements like HTML tags and standardizing the text, for instance, by converting it to lowercase. This ensures uniformity and prepares the data for further processing.

The next phase is text normalization. Here, the text is broken down into smaller units, known as 'tokens', and common but less meaningful words (such as 'and', 'the') are removed as they are known as 'stopwords'. Techniques like stemming and lemmatization are applied to simplify words to their base

forms. For large datasets, breaking the data into smaller chunks or using parallel processing can enhance efficiency. The dataset can also be augmented through methods like synonym replacement, which adds variability and depth to the text.

The text transformation is central to preparing data for deep learning models. Here, methods like Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF) convert text into numerical features, while sophisticated word embeddings and/or sentence embeddings such as Word2Vec, GloVe, BERT, and Universal Sentence Encoder capture the semantic intricacies of language. Special elements like numbers, dates, and emojis are handled with specific strategies to retain their contextual significance.

Finally, it's essential to ensure the quality of the pre-processed text. This involves thorough checks for consistency and verifying the accuracy of each pre-processing step. Such diligence is key to maintaining the integrity of the data, which in turn, significantly impacts the performance of the deep learning models.

Chapter 3: Data Governance

Data governance refers to a collection of policies and standards overseeing how data is gathered, stored, processed, shared or reused, and disposed of. It governs who can access what kinds of data and what the expectations and limitations are. Good data governance improves decision-making, innovation, regulatory compliance, and efficient data usage. Proper knowledge of the data available – both its location and contents, consistent policies aligned with business goals, and privacy by design are all signs of data governance being done well.

Data access

Refers to the ability to retrieve and utilize data stored in a database or other storage system. It encompasses a range of activities, including data retrieval, management, analysis, and protection (i.e retrieving data from the Biostats drive, RWE NaSuni, etc). Depending on the infrastructure, access can be granted "in-place" – where the data is not moved or copied, but rather access to a specific asset is granted to a specific user for a specific period. Alternatively, a copy of the data asset can be provided using secure file transfers, S3 bucket transfers, etc. Data Use Agreements may be required to govern the use of such data assets outside of the original storage system.

Data storage

Data storage is the retention of information via technology that is explicitly designed to store that data and make it as accessible as possible (i.e, storing data under the Biostats drive, RWE NaSuni, HPC etc).

Data Privacy/Anonymization/De-identification

Data Privacy encompasses processes and standards aimed at protecting the privacy and identity of individuals represented in data records in line with relevant regulations.

Data de-identification is the process of removing or transforming personally identifiable information (PII) in a data set to facilitate the reuse of data in scenarios for which the data was not originally collected.

Data anonymization combines data de-identification methods with statistical assessment (ideally, quantitative in nature) which ensures the right level of data transformation is used – enough to reduce residual reidentification risk to an acceptable level, but no more than necessary, so that maximum data utility can be retained.

It needs to be stressed that reducing the risk to 0 is neither possible nor desired as it would inevitably lead to total loss of data utility. Rather, the goal is to reduce the risk of reidentification below a threshold deemed acceptable. There are precedents for risk thresholds that should be observed in the context of clinical data which guard against targeted and inadvertent reidentification.

There are several types of identifiers for which different de-identification strategies can be utilized. Direct identifiers are details which refer to individuals directly. A tested approach to deal with them is pseudonymization, which is a process of substituting identifiable information with a placeholder. A common application of this method is hashing. Each ID is hashed consistently across the records. Hashes should not be reversable to the original value. Another approach is data masking which is the process of

de-identifying where the records are not only changed to no longer point to an individual, but the overall structure of the data is also maintained. A common example of this method is using XXX-XX-XXXX to denote an SSN. Pseudonymization does not affect the data's utility if links within the data set are maintained. Moreover, direct identifiers other than subject ID rarely have scientific value and can be dropped.

Quasi-identifiers are a collection of data that in themselves are not identifying but when combined or used alongside other information can make records identifiable. Examples of such information include birth date, race, sex, and geographical location. An effective strategy to deal with quasi-identifiers is to generalize them. The birth date can be replaced with age or age group, while geographical details can be elevated to country, region, continent, etc. It should be noted that generalization affects data utility. The broader the criteria to generalize quasi-identifiers the less granular those datapoints become. Residual risk of re-identification can be assessed to ensure that the right combination of rules has been applied to reduce the risk to an acceptable level (data anonymization). One approach is the K-anonymity test, which focuses on ensuring that no record would correspond to a single individual but to at least K individuals.

Another type of identifier is dates. Where possible, these can be replaced with relative study days, or offset by a random number, consistent within a record, to retain relationships between events.

Free entry and verbatim text values can contain identifiable details and rule-based de-identification, even in combination with regular expressions, may be impractical. Where coded values are available, those should be retained, while the verbatim values should be dropped.

De-identifying the data, and anonymization in particular, leads to some key benefits. First, it limits your risk exposure and protects individuals' identity and privacy. Second, since the data is no longer considered to be identifiable or personal, it makes sharing and reusing data possible without requiring additional consent from data subjects. It may also address privacy limitations and regulations, e.g., it may not be required to report breaches or data leaks and it certainly minimizes the impact of any data leaks. Lastly, anonymization facilitates data reuse and makes it easier to share internally and with third parties through data use agreements or secure data licensing.

Chapter 4: Data Splitting Strategies and Imbalanced Data

Train/Test Split

It is recommended to split your dataset into training data and test data for deep learning projects. A common splitting strategy is 80% of the data for training and 20% of the data for testing. Data splitting helps minimize overfitting of your data and helps estimate model performance on unseen data in an unbiased way. The training set is used for training the model and determining NN weights, whereas the test set is useful for determining model performance.

Train/Validation/Test Splits

Typically, neural networks have various hyperparameters that require tuning. For this scenario, it is recommended to divide your dataset into three partitions: training, validation, and test. The training and test set have a similar purpose as above, while the validation set can be used to tune NN hyperparameters and perform feature selection.

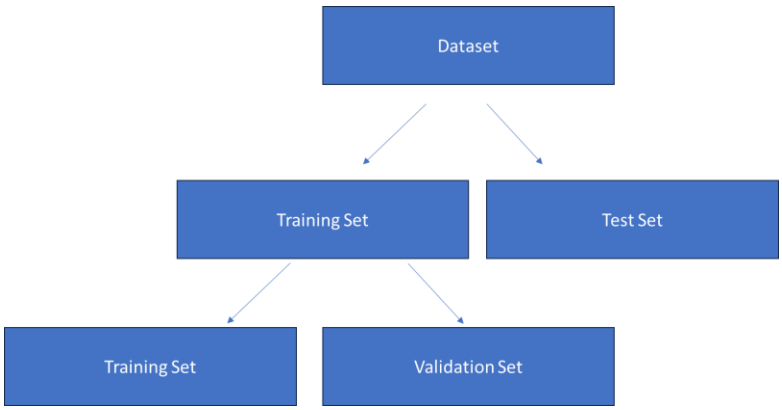


Figure 1: Data splitting
This figure shows a data set split into training, validation, and test sets.

Data Splits and Imputation

Note that data splitting should occur prior to imputation. The imputation model should be trained on the training data to avoid data leakage. Then, the fitted imputation model is applied to the validation and test sets via re-imputation. See https://mlr.mlr-org.com/articles/tutorial/impute.html for more information.

Imbalanced data strategies

For imbalanced data (i.e. one class has many more examples than other class, say 10:1), it is recommended to rebalance the data (1:1) for the training set to create an unbiased training sample. Rebalancing can be accomplished through subsampling, ROSE, SMOTE, propensity score matching, or by improving the data

gathering strategy. For the test and validation sets, the imbalance ratio should match the target population of interest. If the observed imbalance ratio in the validation and test sets do not match the target population ratios, then resampling can be performed.

Cross-validation

Another approach for model evaluation is cross-validation. While this approach is common for traditional machine learning, it is not used often for neural networks due to its computational cost. We begin by splitting our dataset into a training part and testing part. In 5-fold CV applied to the training set, we split the training set into 5 equal parts.

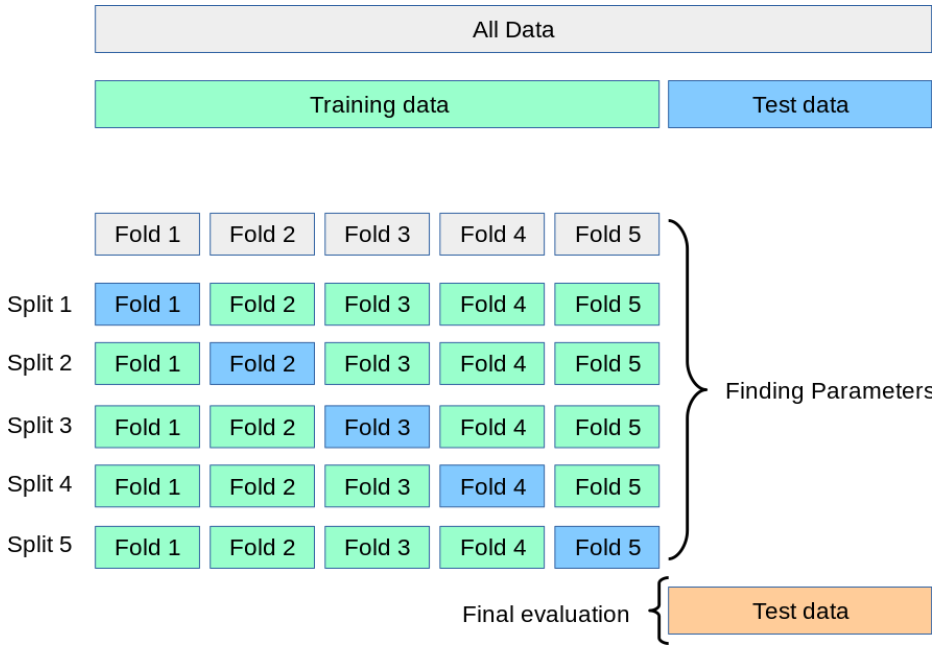


Figure 2: Example of 5-fold cross validation from https://scikit-learn.org/stable/modules/cross_validation.html

In the first iteration the deep NN is trained on folds 2-5 and assessed on fold 1 (validation set). In the second iteration, the deep NN is trained on folds 1,3,4,5 and assessed on fold 2. The process continues until the 5th iteration. Averaging the performance across the blue folds gives our final performance estimate. By varying the hyperparameters of the NN (ie learning rate, drop-out rate, batch size, etc), we can determine the optimal hyperparameter set that maximizes our average performance estimate. With the final set of hyperparameters, we fit the model to the training set and assess its performance on the test data.

Chapter 5: Neural Network Selection

Feedforward network

A feedforward network is the simplest deep learning architecture. It consists of a series of input neurons (which are fed a single data point or image), a series of hidden layers, and output neurons. These layers are connected with a dense set of connections each with an associated weight. A simple FF network looks like:

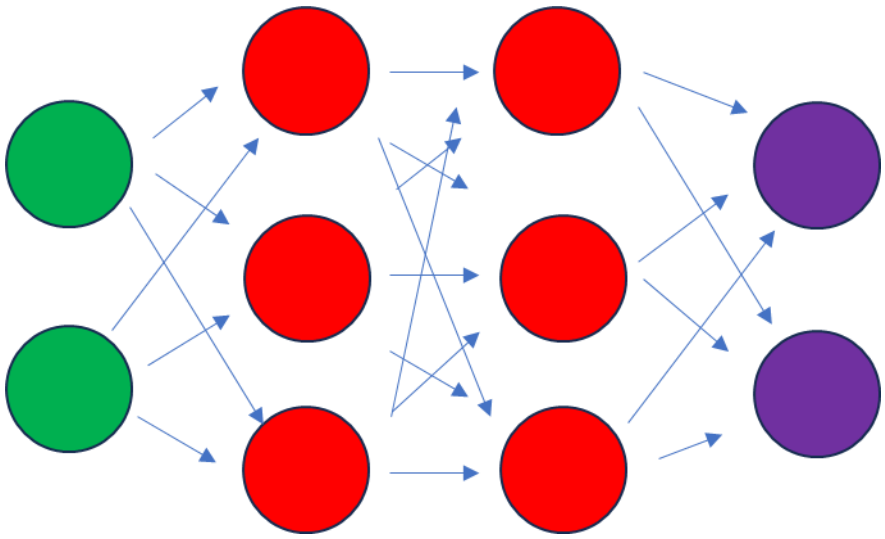


Figure 3: A feedforward network
Green nodes denote input nodes, red nodes are hidden layers, and purple nodes are output nodes.

The output states help make a prediction for a regression problem or output a category for a classification problem. Each neuron contains a series of incoming connections, a bias term, and one output. The final output of an individual neuron is a weighted sum of inputs plus bias then activated with an activation function:

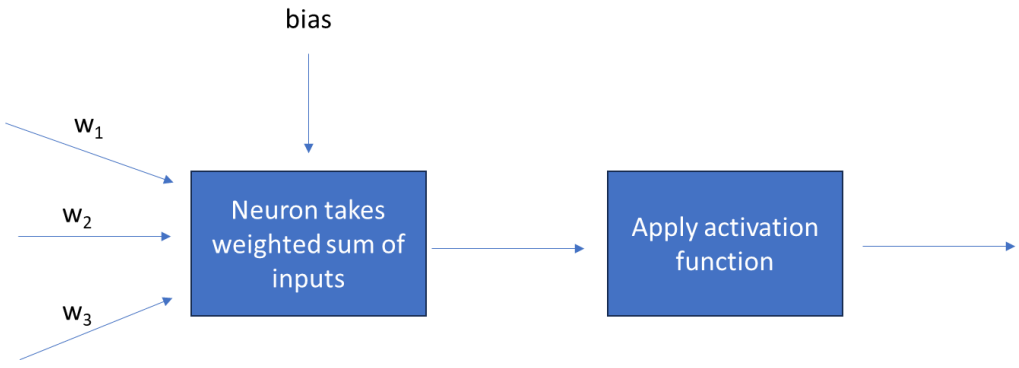


Figure 4: a schematic of a neuron

Convolution Neural Network (CNN)

A convolutional neural network is a common architecture for image analysis which usually consists of a series of convolution and pooling operations. One famous example is the LeNet architecture:

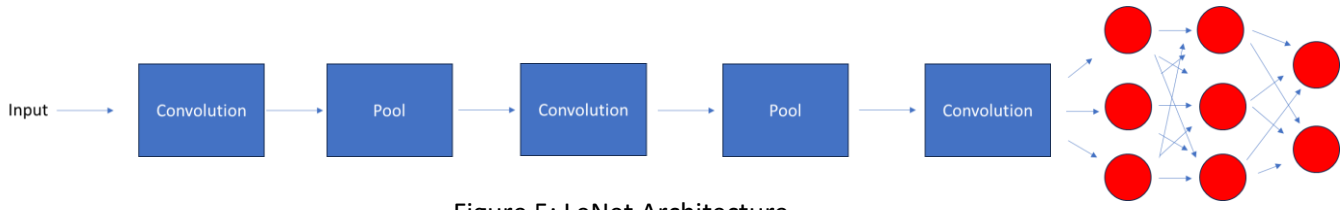


Figure 5: LeNet Architecture

The convolutional layers help extract key features from the image by applying a series of image filters, whereas the pooling layers help to down sample the feature maps and investigate the image at different levels of resolution. See example code at: https://medium.com/@siddheshb008/lenet-5-architecture-explained-3b559cb2d52b.

The applications of CNNs are widespread in image analysis (object detection, face detection, emotion detection, medical imaging, and image captioning) and text analysis (machine translation, next word prediction for smartphones, and document classification).

One challenge of CNN networks is interpretability of the neural network layers. To address this challenge, various methods such saliency maps, grad-cam, and LIME can highlight regions of the image that the network focused on during its classification of image. For more information about deep learning interpretability methods, see the chapter on **Model Interpretability**.

Recurrent neural networks (RNNs)

A RNN is a deep learning architecture that is often used to model sequence data. Common applications include speech recognition, natural language processing, time series analysis, video tagging, OCR, and others. Its architecture is shown below:

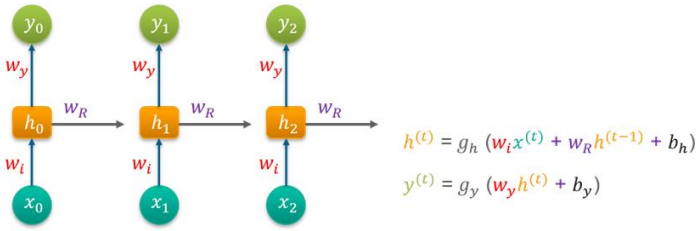


Figure 6: RNN architecture
Image reference: https://www.edureka.co/blog/recurrent-neural-networks/#z2

where h is a hidden state, x are inputs, y are outputs, w are weights, b is bias, and g is a nonlinear activation function. RNNs are named recurrent neural networks due to their recurrence relation in h. Hidden state, $h^{(t)}$, relates to a linear combination of inputs, previous hidden state, and bias which is then activated by

the activation function, g. The output state y is the activation function applied to a linear combination of the hidden state and a bias term.

In recent years, RNNs are not a popular architecture due to two major limitations. First, RNNs can suffer from the vanishing gradient problem which can lead to slow training, and secondly, RNNs can have exploding gradients leading to unstable training. Alternative architectures such as LSTMs and GRUs overcome these limitations. For a code example of an RNN, please see: https://github.com/microsoft/AI-For-Beginners/blob/main/lessons/5-NLP/16-RNN/RNNPyTorch.ipynb

Long short-term memory network (LSTM)A LSTM network is an improvement compared to RNNs since it has less problems with vanishing gradients and can learn long range dependencies. However, LSTMs are limited to a 100-word context for text mining applications and are still slow to train. Another disadvantage is that LSTM networks are not parallelizable so they can't take advantage of GPU acceleration. Despite these disadvantages, LSTMs have been applied in text mining (sentiment analysis, language modelling, text generation), audio analysis (speech recognition, wake word detection), and video classification.

The architecture of a LSTM consists of a forget gate, a learn gate, a remember gate, and a use gate. The forget gate allows the network to decide which information to forget, whereas the learn gate helps the network decide which information from the inputs and short-term memory to learn. The remember gate decides how to update long term memory and lastly the use gate updates short-term memory based on long term memory, short term memory, and the current input.

For a code example of a LSTM applied to time series forecasting, please visit: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/.

Transformers

A recent popular neural network architecture is the transformer architecture. It has some advantages compared to other networks, namely that the input tokens can be parallelized so that gpu acceleration can be applied. A disadvantage of this network architecture is a much more complex architecture compared to previous networks. Despite its complexity, transformers have been applied to many areas of text mining including sentiment analysis, question and answering, machine translation, text summarization, and text classification.

Specifically, the transformer architecture consists of an encoder subnetwork on the left and a decoder subnetwork on the right. The encoder subnetwork finds an encoded vector representation of input words, while the decoder subnetwork can learn representations of the relationship between the input and outputs. Some major architectural innovations compared to the previous architectures include attention (the network pays attention to certain words and their context), masked attention (which allows for masked learning), and positional encoding (which encodes the word position in a sentence).

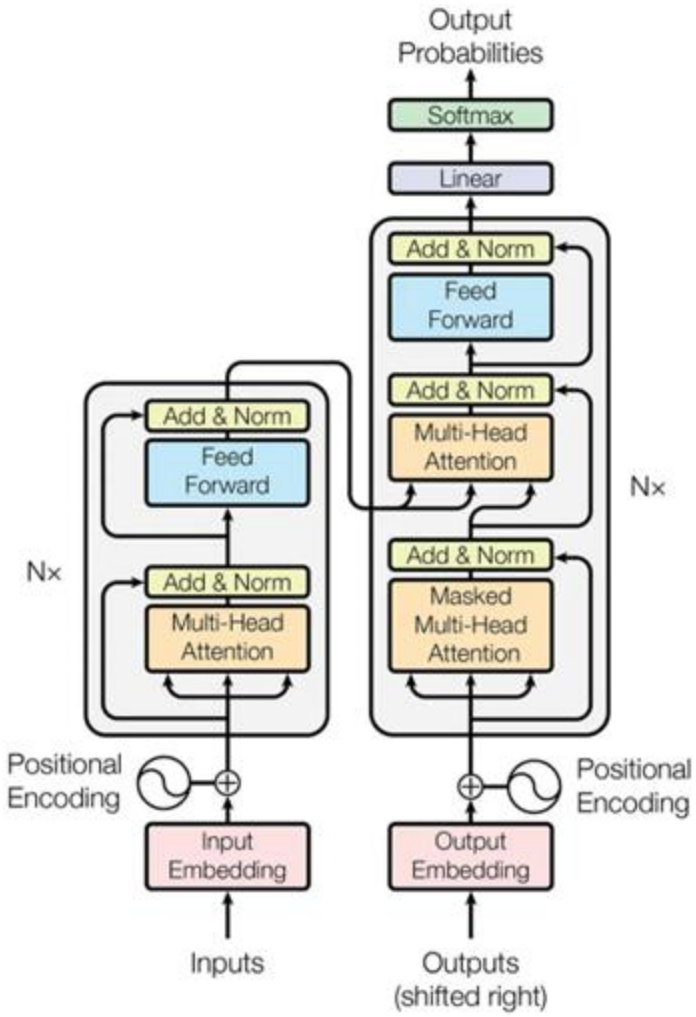


Figure 7: Transformer architecture
Image reference: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

The Text Mining Center of Excellence at Biogen has implemented transformer models for various internal applications in regulatory, quality, and other R&D domains. By leveraging state-of-the-art language models and machine learning techniques, the team has developed solutions to enhance text analysis, risk assessment, pattern recognition, and document processing. These initiatives aim to improve efficiency, streamline operations, and provide valuable insights across different business areas. While showing promising results, the projects need further validation for potential wider deployment. For code examples of the transformer architecture, please see: https://huggingface.co/docs/transformers/v4.41.3/en/model_doc/roberta#transformers.RobertaForQuestionAnswering

The Autoencoder

The autoencoder architecture is commonly used in applications that require compression or dimension reduction. Other applications include image processing (image generation, image denoising, and image compression), time series data generation, and anomaly detection. Here we will focus on the simplest autoencoder architecture, but other variants such as Deep CNN autoencoder and Denoising autoencoder exist.

The simplest model of an autoencoder architecture is displayed below. The inputs are compressed into a lower dimensional representation through the bottleneck layer. This process is called encoding. From the hidden layer to the output layer performs decoding (or transforming the latent representation back to the original dimensionality). The objective in training this model is for the reconstruction image (or dataset) at the output layer to be as close as possible to the input image (or dataset). For code examples, please visit: https://www.tensorflow.org/tutorials/generative/autoencoder

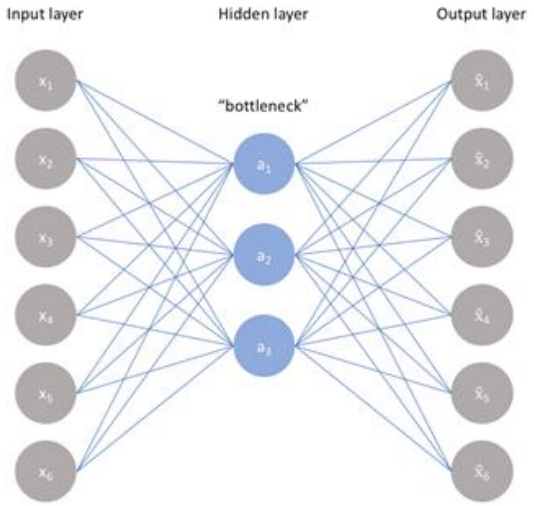


Figure 8: autoencoder architecture
Image reference: https://www.v7labs.com/blog/autoencoders-guide

Generative Adversarial Network architecture (GANs)

GANs have been recently quite popular due to their ability to generate new images from a text prompt (ie deepfakes) and their ability to age faces, perform photo up-sampling, perform photo inpainting, and perform style conversion (such as transforming a sketch to a photo or transforming a photo to a painting). GANs are based on the idea of an adversarial game, where the discriminator tries to determine which images are real versus fake and the generator tries to improve its ability of generating fake images to fool the discriminator. After much training, the generator becomes so good at generating fakes that the discriminator only has a 50% chance of detecting the fakes.

Early literature used the DCGAN model (Deep Convolutional Generative Adversarial Networks) with CNNs for both the generator and the discriminator. The generator has as input a random input and will generate a fake image. The discriminator is a binary classification model to distinguish real versus fake images. The detailed architecture of GANs is shown below:

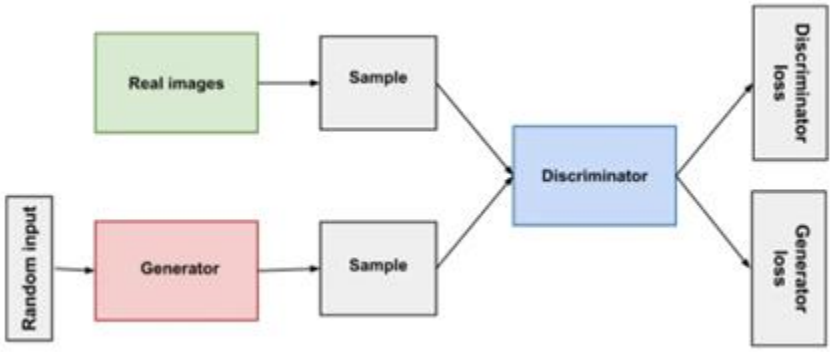


Figure 9: GAN architecture from https://developers.google.com/machine-learning/gan/images/gan_diagram.svg

The generator and discriminator are updated as follows:

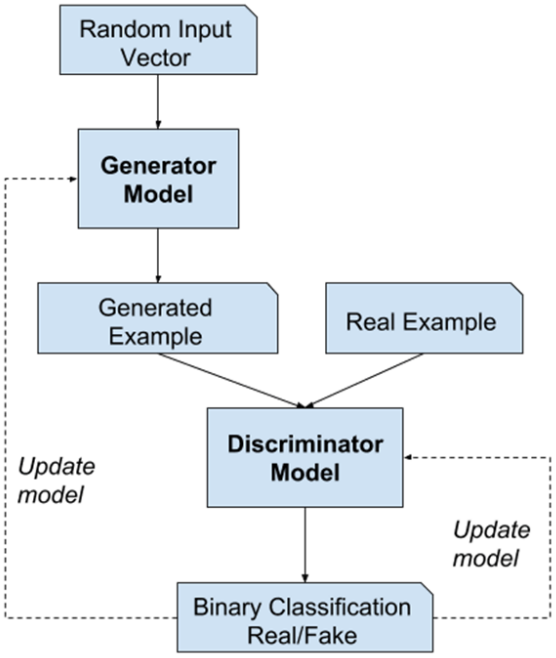


Figure 10: GAN update process flow, image from:
https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

In detail, if the generator fools the discriminator, the generator network is rewarded and the discriminator network is penalized. In contrast, if the discriminator successfully detects the fake, the discriminator network is rewarded and the generator network is penalized. For code examples, please visit: https://www.tensorflow.org/tutorials/generative/dcgan

Deep Learning Use Cases and their Associated Architectures

To aid users in selecting a proper neural network architecture, we have listed below common use-cases and their associated neural network architectures.

Common architectures for NLP use-cases:

Sentiment analysis	LSTM, transformer
Text summarization	RNN, transformers
Machine Translation	RNN, transformers
Text generation	LSTM, transformers
QA system	Transformers
Text classification	Transformers, LSTM, GRU

Table 2: NLP use-cases

Computer vision applications:

Object detection	CNN
Image classification	CNN
OCR	RNN
Video classification	LSTM
Image denoising	autoencoder
Image compression	autoencoder
Image generation (text to image)	GANs
Image upsampling	GANs

Table 3: Computer vision applications

Audio applications:

Speech recognition	LSTM, transformers
Video classification	LSTM

Table 4: Audio applications

Translational biology applications:

Protein Folding	Alphafold, Deep residual network
Antigen–antibody affinity prediction	ConvNeXt network

Table 5: Translational biology applications

Statistics applications:

Time series prediction	RNN, LSTM
Dimension reduction	autoencoder
Anomaly detection	autoencoder

Table 6: Statistics applications

Chapter 6: Parameter Tuning

Parameter tuning is an essential step in building effective deep learning models. Choosing the right hyperparameters can significantly improve the model's performance. In this section, we will introduce some common parameter tuning approaches in deep learning, along with some Python/R code examples.

Before we go into details, we would like to first provide a decision graph to help illustrate under which scenarios you should use different type of hyperparameter tuning approaches:

Scenario	Grid Search	Random Search	Bayesian Optimization
Small search space	Recommended	Optional	Optional
Large search space	Not recommended	Recommended	Recommended
Limited time budget	Not recommended	Recommended	Recommended
Limited computational resources	Not recommended	Recommended	Recommended
Searching for global optima	Optional	Optional	Recommended
High-dimensional hyperparameters	Not recommended	Recommended	Recommended
Continuous hyperparameters	Not recommended	Optional	Recommended
Noisy objective function	Not recommended	Optional	Recommended
Parallelization	Optional	Recommended	Recommended
History of prior evaluations	Not applicable	Not applicable	Recommended
Adaptive exploration-exploitation trade-off	Not applicable	Not applicable	Recommended

Table 7: Comparison of tuning approaches

Grid Search is suitable when you have a limited number of hyper-parameters and their performance characteristics are known. It is feasible to try all combinations and find the best configuration. It's the most exhaustive but can be computationally expensive when the search space is large.

Random Search is suitable when you have a large search space and it's difficult to determine the range or relationships between hyperparameters. It provides a more efficient search compared to Grid Search by randomly sampling configurations, but it doesn't guarantee finding the optimal solution.

Bayesian Optimization is suitable when you have a large search space and it's difficult to determine the range or relationships between hyperparameters. It builds a probabilistic model of the objective function and intelligently selects configurations to evaluate. It can efficiently find optimal configurations with fewer iterations compared to random or grid search.

Brute Force Grid Search

Grid search is a simple and widely used approach for hyperparameter tuning. It exhaustively searches through a specified parameter space by trying all possible combinations of the given hyperparameters. Scikit-learn provides the GridSearchCV class for performing grid search with cross-validation. See more details here: https://towardsdatascience.com/grid-search-in-python-from-scratch-hyperparameter-tuning-3cca8443727b

Random Search

Random search is a more efficient alternative to grid search. Instead of exhaustively searching through all possible combinations, random search samples a random subset of the hyperparameter space. Scikit-learn provides the RandomizedSearchCV class for performing random search with cross-validation. See more details here: https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/

Bayesian Search

Bayesian search is a probabilistic approach that models the unknown objective function (model performance) as a Gaussian process. It iteratively updates the beliefs about the objective function using Bayesian inference, and then selects the next point to sample based on acquisition functions. Scikit-optimize provides the BayesSearchCV class for performing Bayesian search with cross-validation. See more details here: https://scikit-optimize.github.io/stable/modules/generated/skopt.BayesSearchCV.html

The mlrMBO package in R also provides a framework for parameter tuning using Bayesian optimization. See more details here: https://cran.r-project.org/web/packages/mlrMBO/index.html

Parameter tuning for deep learning models with Optuna in Python

Optuna (for details, see https://optuna.org/) is a popular hyperparameter optimization library (implemented in PyTorch) to tune hyperparameters of a deep learning model. Optuna uses a more advanced approach compared to grid search or random search for hyperparameter optimization. It employs a Tree-structured Parzen Estimator (TPE), a Bayesian optimization method, to efficiently explore the hyperparameter search space. TPE builds a probabilistic model to estimate the distribution of hyperparameters that yield better results and guides the search towards promising areas of the search space. This approach is more efficient than grid search or random search, as it uses the information from previous trials to decide the next set of hyperparameters to evaluate.

Parameter tuning for deep learning models with GenSA in R

One popular optimization technique used for hyperparameter tuning is simulated annealing (SA). SA is inspired by the annealing process in metallurgy and mimics the cooling process of a material to reach a low-energy state. It has been successfully applied to various optimization problems, including parameter tuning in deep learning. In this section, we introduce the Generalized Simulated Annealing (GenSA) method, which is an extension of the SA algorithm (For details, see https://journal.r-

project.org/archive/2013/RJ-2013-002/RJ-2013-002.pdf). GenSA is a derivative-free optimization algorithm that uses a modified version of the Metropolis-Hastings algorithm to explore the parameter space. It introduces several enhancements to the original SA algorithm, such as dynamic rescaling, parallel tempering, and adaptive cooling schedules, to improve its convergence and exploration capabilities.

Comparison of Search Methods and Implementation Tools

	PROS	CONS
Brute force grid search	• Can find the optimal hyper-parameters with high precision	• Computationally expensive and time-consuming, especially for large parameter spaces
Random search	• Less computationally expensive and faster than grid search • Can achieve similar performance with fewer iterations	• Less precise than grid search
Bayesian search (See more details here: https://arxiv.org/pdf/1012.2599)	• Can find optimal hyper-parameters more efficiently than grid and random search • Incorporates prior knowledge and uncertainty about the objective function	• More complex than grid and random search
Optuna in Python	• Efficient optimization using Bayesian optimization with TPE • Easy integration with popular machine learning and deep learning frameworks • Supports pruning, parallelization, and visualization • Flexible in defining search spaces, objective functions, and optimization algorithms	• More complex than simpler methods like grid and random search • Less suitable for small search spaces • Computationally expensive for large search spaces and complex models
GenSA in R	• Robustness: GenSA is robust in finding global optima in complex search spaces • Exploration-exploitation balance: it efficiently explores the parameter space while focusing on promising regions • Derivative-free optimization: it works without requiring gradient information, making it applicable to a wide range of problems	• Computationally expensive: the GenSA algorithm can be computationally expensive, especially for large-scale optimization problems or when the objective function evaluation is time-consuming • Sensitivity to initial values: the performance of GenSA can be sensitive to the initial parameter values. If the initial values are far from the optimal solution, it may take

		longer for the algorithm to converge, or it may converge to suboptimal solutions. • Limited scalability: while GenSA performs well for many optimization problems, it may face challenges when applied to highly complex or high-dimensional search spaces

Table 8: comparison of search methods

Chapter 7: Model Evaluation

Model evaluation is a critical step in the machine learning and deep learning process. It allows developers to measure the performance of their models, identify issues like overfitting and underfitting, and make improvements to achieve better results. This section introduces the best practices for conducting model evaluation, including various evaluation metrics, loss functions, the detection of overfitting and underfitting, and possible solutions. We will also discuss the role of subject matter experts in the validation process and the risks associated with subjectivity.

Model Evaluation Metrics

There are several metrics to evaluate the performance of a model, including loss, accuracy, F1 score, precision, and recall. These metrics help us understand different aspects of a model's performance. Before we go into details, we would like to first provide a decision graph to help illustrate under which scenarios you should use the different types of model evaluation metrics:

Scenario	Regression Metrics	Binary Classification Metrics	Multiclass Classification Metrics
General performance evaluation	Mean Squared Error (MSE)	Accuracy	Accuracy
Emphasizing larger errors	Mean Squared Error (MSE)	F1-score	Macro-averaged F1-score
Penalizing overestimation	Quantile Loss (https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/)	Precision	Macro-averaged Precision
Penalizing underestimation	Root Mean Squared Logarithmic Error (RMSLE, see details: https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/)	Recall	Macro-averaged Recall
Interpretability	R-squared (R^2)	Area Under the ROC Curve (AUC-ROC)	One-vs-One ROC AUC
Imbalanced data	Not applicable	F1-score, Precision-Recall Curve (PRC)	Weighted F1-score, per-class metrics
Probability calibration	Not applicable	Brier score (see details: https://en.wikipedia.org/wiki/Brier_score), Log-loss	Log-loss, Brier score (multiclass extension)

Table 9: Comparison of Regression, Binary Classification, and multi-class metrics

For **binary class classification**, when you choose the proper performance metrics, there are multiple considerations:

Consideration	Yes	No
Is the dataset balanced (roughly equal number of samples in each class)?	Accuracy, Sensitivity, Specificity, and ROC-AUC	Precision-Recall Curve, Average Precision, or Balanced Accuracy
Are the classes mutually exclusive (samples belong to only one class)?	Accuracy, Precision, Recall, F1-Score, and Confusion Matrix	Hamming Loss, Exact Match Ratio, or Micro/Macro-Averaged metrics
Are there specific concerns regarding false positives/negatives or class imbalance?	For false positives/negatives: Examine Precision, Recall, and F1-Score. For class imbalance: Consider using Weighted or Balanced Accuracy	Accuracy as the primary metric and consider additional metrics based on the specific problem context.

Table 10: Binary classification considerations

For multi-class classification, a similar strategy applies, but with the necessity to binarize the output. Metrics for each individual class label can be calculated, and specifically, the Micro and Macro F1 scores offer different approaches to compute the F1-score in multiclass classification problems, which is the harmonic mean of precision and recall. Macro F1 calculates the F1-score independently for each class and then averages them, making it suitable when all classes are of equal importance. Micro F1, in contrast, aggregates the true positives, false positives, and false negatives across all classes before computing the F1-score, making it more appropriate for imbalanced class distributions and for assessing overall performance.

See the details of the most commonly used performance metrics for the evaluation of classification models as follows:

- Accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

- Precision (Positive Predictive Value)

$$Precision = \frac{TP}{TP + FP}$$

where TP is the number of true positives and FP is the number of false positives.

- Recall (Sensitivity or True Positive Rate)

$$Recall = \frac{TP}{TP + FN}$$

where TP is the number of true positives and FN is the number of false negatives.

- Specificity (True Negative Rate)

$$Specificity = \frac{TN}{TN + FP}$$

where TN is the number of true negatives and FP is the number of false positives.

- F1 Score

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN}$$

where Precision is the precision value and Recall is the recall value.

- Negative Predictive Value (NPV)

$$NPV = \frac{TN}{TN + FN}$$

where TN is the number of true negatives and FN is the number of false negatives.

Loss Functions

A loss function in deep learning is a measure of the model's performance by quantifying the difference between predicted and true values. It guides the model's learning process by minimizing the error and updating its parameters. On the other hand, performance metrics evaluate the model's quality and effectiveness, providing additional insights such as accuracy, precision, recall, F1 score, and AUC-ROC. While loss functions are optimized during training, performance metrics are computed on separate evaluation datasets to assess the model's generalization and to compare different models.

In this section, we will discuss some decision rules for **choosing Loss Functions** for deep learning models.

Classification Deep Learning Models

	Balanced scenario (approximately equal number of samples in each class)	Imbalanced scenario (significantly unequal number of samples in each class)
Binary class	1. Binary Cross-Entropy Loss (Log Loss): Suitable for problems with a binary output and when the model outputs class probabilities	1. Weighted Binary Cross-Entropy Loss: Assign higher weights to the minority class to address the class imbalance problem. 2. Focal Loss: Introduce a modulating factor to downweight the easy examples and focus on hard examples,

		helping to alleviate the impact of class imbalance
Multi-class	1. Categorical Cross-Entropy Loss: Suitable when the model outputs class probabilities and the classes are mutually exclusive	1. Class Weighted Categorical Cross-Entropy Loss: Assign higher weights to the minority classes to address the class imbalance problem. 2. Focal Loss: Introduce a modulating factor to downweight the easy examples and focus on hard examples, helping to alleviate the impact of class imbalance

Table 11: Classification scenarios and their loss functions

If the problem is **multi-label classification** (instances can belong to multiple classes simultaneously), consider using Binary Cross-Entropy Loss: Treat each class as a separate binary classification problem and use binary cross-entropy individually for each class.

Regression Deep Learning Models

Decision Rules for Choosing Loss Function for Regression Models:

Mean Squared Error (MSE)	1. MSE is a common loss function for regression problems and is suitable for most scenarios. 2. Use MSE as a default choice for regression unless there are specific requirements.
Mean Absolute Error (MAE)	1. MAE calculates the average absolute difference between the predicted and actual values. 2. MAE is less sensitive to outliers compared to MSE, making it suitable when outliers are present or when the absolute error is more important than the squared error.
Huber Loss (See more details here: https://en.wikipedia.org/wiki/Huber_loss)	1. Huber loss combines properties of MSE and MAE. 2. It is less sensitive to outliers like MAE while having quadratic behavior for small errors like MSE. 3. When the dataset contains outliers and you want a balance between the robustness of MAE and the smoothness of MSE.
Log-Cosh Loss (See more details here: https://arxiv.org/pdf/2208.04564)	1. Another option that combines properties of MSE and MAE. 2. Similar characteristics to Huber loss but with a smoother transition. 3. Can be suitable when the dataset contains outliers and you want a smooth loss function.

Custom Loss functions	1. Depending on the specific requirements of your regression problem, you can create custom loss functions tailored to the task. 2. Can incorporate domain-specific knowledge or address specific challenges in the data.

Table 12: Regression Loss Functions

As we discussed above, loss is a measure of the difference between the predicted values and the actual values. The goal is to minimize the loss function during training. Different types of loss functions are more suitable for different types of problems. Here, we will discuss four common loss functions: Mean Squared Error (MSE), Mean Absolute Error (MAE), Cross-Entropy Loss, and Log Loss.

Mean Squared Error (MSE) is expressed as $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$,where y_i is the true label, $\hat{y}_i$ is the predicted value, for i = 1 to n (number of samples). MSE computes the average squared difference between predicted and actual values, with a smaller value indicating a more accurate model.

Mean Absolute Error (MAE) is given by $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}|$, it uses the same parameters as MSE, but instead calculates the average absolute difference. MAE is particularly effective in regression tasks, being less sensitive to outliers compared to MSE.

Cross-Entropy Loss is vital for classification tasks, and its formula is $CE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(p_{ij})$, where y_{ij} is the indicator function (1 if sample i belongs to class j and 0 otherwise), p_{ij} is the predicted probability of sample i belonging to class j, n is the number of samples, and m is the number of classes. Cross-Entropy Loss is widely used for classification tasks. It measures the performance of a classification model by calculating the difference between the predicted probabilities and the actual class labels. For binary classification, the loss function is called Binary Cross-Entropy Loss, and for multi-class classification, it is called Categorical Cross-Entropy Loss.

Log Loss, or Binary Cross-Entropy Loss, is tailored for binary classification tasks,
Log Loss $= -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$, where y_i is the true label (0 or 1), p_i is the predicted probability of the positive class (between 0 and 1), and n is the number of samples. Log Loss, or Binary Cross-Entropy Loss, is used for binary classification tasks. It calculates the loss between the predicted probabilities and the actual binary labels.

Detection of Overfitting and Underfitting

In the context of deep learning, neural network models often face the challenges of overfitting and underfitting during training. Overfitting occurs when a model becomes overly complex and starts memorizing the training data rather than learning to understand the underlying patterns or relationships. This leads to excellent performance on the training data but poor generalization to new, unseen data. The telltale signs of overfitting include a model performing markedly better on training data than on validation or test data, exhibiting low training loss but high validation loss, and capturing noise or random fluctuations in the training data. To detect overfitting, one can plot learning curves to observe the divergence between training and validation loss, or one can evaluate the model on unseen data to assess its generalization capability.

Conversely, underfitting is when a neural network model is too simplistic, failing to grasp the complexities in the data, and showing subpar performance on both training and validation data. Characteristics of underfitting are poor performance across both training and validation datasets, high losses in both cases, and an inability to discern patterns in the data. Detecting underfitting involves similar methods as overfitting: analyzing learning curves where both training and validation losses remain high and show minimal improvement, and scrutinizing performance metrics like accuracy. Consistently low performance on training data is a red flag for underfitting, suggesting the need for a different architectural approach.

Solutions for Overfitting and Underfitting

In deep learning, solving overfitting involves several strategies. Increasing the training data by obtaining more labeled data can expand the training set, allowing for better generalization and reducing overfitting. Data augmentation techniques such as random rotations, translations, or flips can artificially enlarge and vary the training data, aiding the model in learning more robust features. Regularization is key, with techniques such as L1 and L2 regularization, which penalize large weights in the loss function and reduce dependency on specific features. Dropout regularization involves randomly deactivating neurons during training, encouraging diverse representations, while early stopping halts training when validation loss increases, finding the optimal generalization point. Reducing model complexity by simplifying architecture, like fewer layers or neurons, also helps in preventing overfitting.

Conversely, solving underfitting in deep learning might require increasing model complexity. This could involve adding more layers or neurons or employing non-linear activations to capture complex data patterns. Incorporating domain knowledge through feature engineering provides more meaningful data representations, enhancing model performance. Experimenting with different architectures, like convolutional neural networks (CNNs) for image data or recurrent neural networks (RNNs) for sequential data, can offer better fits. Extending training time allows the model to learn from data more comprehensively, especially when signs of underfitting are present. Hyperparameter tuning, including adjustments to learning rate, batch size, optimizer choice, or activation functions, is crucial for fine-tuning performance.

Addressing overfitting and underfitting is iterative, often requiring a combination of techniques and adjustments for optimal model performance. Continuously monitoring and modifying the model based on its performance with unseen data is essential for effective mitigation of these issues in deep learning models.

Validation by Subject Matter Experts

Leveraging Subject Matter Experts (SMEs) to evaluate Deep Learning and AI models in the pharmaceutical industry is essential to ensure the validity, accuracy, and relevance of the models to the specific domain. SMEs bring invaluable domain knowledge that can significantly enhance the development and evaluation of AI models tailored to the unique challenges and requirements of the pharma industry.

To effectively utilize SMEs, their input should be sought at different stages of model development, including data curation, feature selection, model training, and validation. During the data curation stage, SMEs can help identify the most relevant and high-quality data sources, ensuring that the model is trained on representative and diverse data. This is particularly important in the pharma industry, where data quality and integrity are critical for the success of AI-driven drug discovery and clinical trials.

SMEs can guide the selection of appropriate features and variables to include in the model, identifying key relationships in the data that might not be evident to data scientists. This ensures that the AI models are tailored to capture the intricacies of the pharmaceutical domain, such as complex molecular interactions, patient populations, and regulatory requirements.

During the model training and validation stages, SMEs can provide insights into the most appropriate performance metrics and evaluation criteria, ensuring that the model's predictions align with the underlying scientific principles and industry expectations. They can help in interpreting the model outputs, identifying potential areas of improvement, and validating the results against established benchmarks.

Advantages of involving SMEs in the evaluation process include increased confidence in the model's accuracy and relevance, improved model explainability, and better alignment with industry regulations and standards. Involving SMEs can also foster a sense of ownership and trust among stakeholders, leading to better acceptance and adoption of AI-driven solutions in the pharma industry.

However, there are also potential cons and pitfalls. SMEs may have limited availability, which could lead to delays in the model development and evaluation process. They may also not be well-versed in AI and deep learning concepts, which could lead to communication challenges and slower progress. Additionally, relying heavily on expert input might introduce biases and limit the potential for novel insights or discoveries that AI models can offer.

To overcome these challenges, organizations should invest in educating SMEs about AI and deep learning concepts and foster a collaborative environment between data scientists and domain experts. This can be achieved through cross-functional teams, joint workshops, and regular knowledge-sharing sessions. It is also crucial to ensure a balanced approach to model evaluation, considering both expert input and data-driven insights to maximize the potential benefits of AI in the pharmaceutical industry.

Lastly, organizations should be aware of the potential risks associated with over-reliance on SMEs and strive to maintain a healthy balance between human expertise and data-driven AI insights. By combining the strengths of both SMEs and AI models, the pharmaceutical industry can unlock the full potential of AI-driven innovation, leading to faster drug discovery, improved clinical trial outcomes, and ultimately, better patient care.

Chapter 8: Data augmentation

Data augmentation is a technique widely employed in machine learning to artificially increase the diversity of training data. It involves creating variations of the original data by applying a set of transformations. This section explores data augmentation methods in the context of Natural Language Processing (NLP) and Computer Vision (CV).

The purpose of this section is to provide a comprehensive overview of common data augmentation methods in NLP and CV, highlighting their pros and cons. By understanding these techniques, practitioners can make informed decisions about which augmentation strategies to apply in their specific applications.

Data Augmentation in NLP

In the realm of Natural Language Processing (NLP), data augmentation can be achieved through various methods. Synonym replacement is one such method, where words in a sentence are replaced with their synonyms, maintaining the original meaning. Mask language modeling, another technique, involves inserting random words into a sentence, increasing its length and complexity. For instance, in the sentence "The quick brown fox jumps over the lazy dog," replacing a word with a mask might result in "The quick brown [MASK] jumps over the lazy dog," and a language model could then predict the missing word. Random deletion and text shuffling are also employed, with the former removing random words to create a concise version and the latter rearranging words to form new structures. Back translation is another strategy, involving translation between languages and back to the original to create variations. See some code examples here: https://github.com/makcedward/nlpaug

These NLP augmentation methods come with pros and cons. The advantages include enhanced model robustness and generalization, reduced risk of overfitting, improved performance in low-resource languages or domains, and better handling of out-of-vocabulary words. However, they also have downsides, such as the risk of introducing grammatical errors or nonsensical text, potential unsuitability for specific tasks like translation or summarization, and increased computational costs due to larger datasets. Below is a table to summarize Pros and Cons for NLP data augmentation:

Pros	Cons
• Enhanced model robustness and generalization. • Reduced risk of overfitting. • Improved performance on low-resource languages or domains. • Better handling of out-of-vocabulary words.	• Risk of introducing grammatical errors or nonsensical text. • May not be suitable for all NLP tasks (e.g., translation or summarization). • Increased computational cost due to increased dataset size. • Medical/Legal domains may have special considerations to retain meaning during augmentation

Table 13: Pros and Cons for NLP data augmentation

Data Augmentation in CV

Similarly, in Computer Vision (CV), data augmentation encompasses techniques like image rotation, which involves rotating an image by a random angle to aid object recognition from various perspectives. Image flipping, either horizontally or vertically, creates mirror images, enhancing dataset diversity. Zooming and cropping allow models to focus on specific image regions, while color jitter introduces random color variations to bolster model robustness against lighting and color changes. Gaussian noise and adding random pixel-level noise simulates real-world imperfections. See some code examples here: https://github.com/AISangam/Image-Augmentation-Using-OpenCV-and-Python

The pros and cons of CV data augmentation parallel those in NLP to some extent. The benefits include improved model generalization, enhanced performance in object recognition and classification tasks, and increased robustness against variations in lighting, orientation, and noise. However, the drawbacks involve the risk of generating unrealistic images, heightened computational costs due to larger training datasets, and limited effectiveness for specific image types, such as medical images. Below is a table of the Pros and Cons for CV data augmentation:

Pros	Cons
• Improved model generalization. • Enhanced performance on object recognition and classification tasks. • Increased robustness to variations in lighting, orientation, and noise.	• Risk of generating unrealistic images. • Increased computational cost due to larger training datasets. • Limited effectiveness for certain types of images (e.g., medical images).

Table 14: Pros and Cons for CV data augmentation

Data Augmentation Best Practices

In the area of data augmentation, several best practices and considerations emerge. First and foremost, the selection of the most appropriate augmentation techniques should be driven by the unique characteristics of the dataset, the specific task at hand, and domain knowledge. It is advisable to experiment with various augmentation methods to determine which ones yield the most significant improvements, but please keep in mind that some augmentations may not be appropriate based on which domain you are studying. For example, in the medical domain, using the wrong synonym could lead to misinterpretation and potentially dangerous outcomes. Additionally, fine-tuning augmentation hyperparameters, such as rotation angles, noise levels, or the extent of text modifications, can significantly impact the augmentation process's effectiveness. When implementing data augmentation, a rigorous evaluation and validation process should be in place to ensure that the augmented datasets maintain data quality and do not introduce biases that could affect model performance. Moreover, ethical considerations must be at the forefront, particularly in sensitive domains like healthcare, to ensure that augmented data does not compromise privacy or fairness, adhering to responsible and ethical AI practices.

Data augmentation is a crucial technique for improving model generalization and robustness in NLP and CV. This white paper has provided an overview of common augmentation methods, their pros and cons, and their applications in both domains. Understanding and applying data augmentation effectively can lead to more reliable and accurate deep learning models in various applications.

Chapter 9: Improving Model Performance of Pretrained Models

Transfer learning

Transfer learning is the idea of utilizing a pretrained model in one domain and transferring that knowledge to a related domain. Since we expect low level features will be beneficial for both domains, transfer learning is typically achieved by utilizing the first layers of the pre-trained model (representing low level features of an images) in the new transfer learning model. However, if the two domains are not related or low-level features are not in common for the two domains, transfer learning won't be that useful.

Transfer learning has some key strengths compared to training the whole model. First, samples required for training in the related domain are typically much less than the samples used for training the pre-trained model. Additionally, the training time needed for transfer learning is typically less than training time needed for the whole model. Lastly, the model performance on the related domain can improve after applying transfer learning.

Transfer learning architecture

In detail, we illustrate the architecture of transfer learning below:

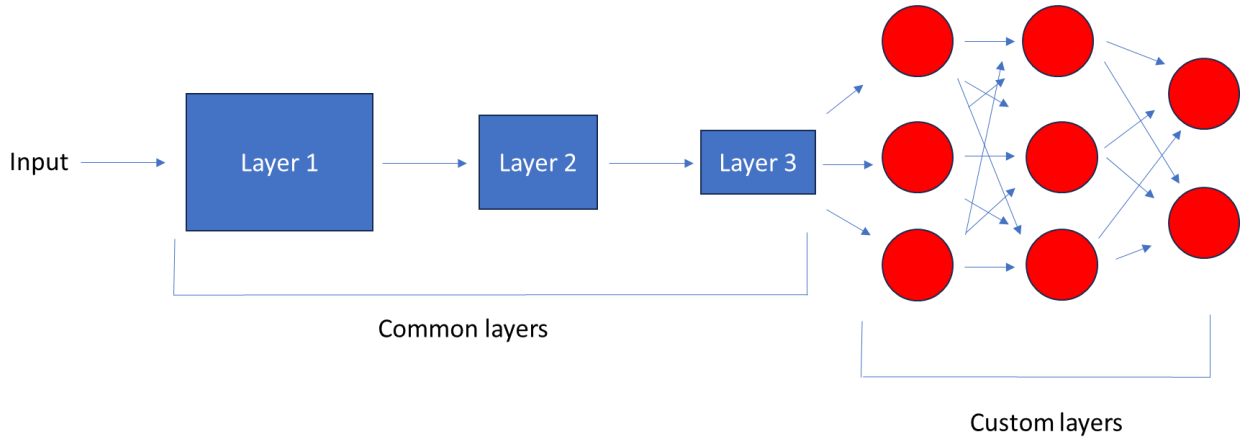


Figure 11: Transfer learning architecture

In our transfer learning model (custom model), the first layers are frozen, and we utilize the weights/biases from the pretrained model. Since we expect the high-level features to differ across domains, the FC layers (red) are not frozen and are retrained with random weights initialization. The output layer may need to be replaced if the number of classes is different in the target domain. This architecture is applicable if the domains are related and the number of training examples from the target domain is small.

Fine tuning

In the scenario where the number of training examples in the target domain is large and the domains are similar, we can utilize the fine-tuning approach. This architecture begins similarly as above: red FC layers are not frozen and are initialized with random weights whereas the output layer is modified for the number of the classes in the target domain. Next, the common inner layers are unfrozen and initialized with weights from the pre-trained model. The entire custom model is trained on examples from the target domain. We called this methodology, fine tuning, since the common layers are initialized with pre-trained weights are their weights are just fine tuned on the target domain. We expect the model performance to improve in the target domain after fine tuning and the training time should be reduced compared to the scenario of retraining the entire model with random initialization.

LORA

LORA, or low rank adaptation of large language models, allows for efficient fine tuning of large language models. Typically, these models can have 100s of billions of parameters making them computationally expensive to fine tune. However, it has been observed that weight matrices of a LLM can be represented by a low rank representation. With LORA, we can represent the update of the weight matrices of the attention layers of an LLM as: delta W = B*A where B*A is a low rank decomposition of delta W. During training, only A and B are learned whereas the original weights are frozen. Since the rank decomposition matrices have much fewer parameters than W, training of LLMs is possible with much less VRAM and computational resources. Another advantage of LORA is efficient task switching since you only need to provide new LORA weights per task. However, LORA has some disadvantages as well: 1) the low rank adaptation procedure may lose information, and 2) the performance of LORA can be impacted by the selection of the LORA rank.

References

1. https://www.ml6.eu/blogpost/low-rank-adaptation-a-technical-deep-dive
2. https://arxiv.org/abs/2106.09685
3. https://huggingface.co/docs/diffusers/main/en/training/lora
4. https://www.entrypointai.com/blog/lora-fine-tuning/

Adapter transfer learning

Standard fine tuning of a pretrained neural network model is inefficient when your goal is to fine tune multiple new downstream tasks requiring new models for each task and tuning of 100% of the model weights for each task. Houlsby et al propose the idea of adapters, neural network modules with a small number of weights, which are fine tuned for each new task, while pre-existing neural network weights are frozen. This results in a dramatic reduction in the number of trainable parameters per task and a high degree of parameter sharing across task models. Even though adapters have two orders of magnitude fewer parameters than full fine tuning, Houlsby et al illustrated similar performance between adapters and full fine tuning.

Specifically for the transformer architecture, adapter modules are added to the transformer layers (see below left). These adapter modules consist of a bottleneck layer, an up-projecting layer, and a skip-connection (see below right). The adapter has a single hyperparameter, namely the number of units in

the bottleneck layer. With this architecture, adapters modules, layer norm layers, and the final classification layer are tuned per task while all other layers are frozen. Adapter learning has some advantages compared to previous learning approaches: 1) compared to the continual learning method, adapter tuning will not forget previous tasks during retraining (aka catastrophic forgetting) and tasks will not interact; and 2) compared to multi-task learning, adapter learning does not require simultaneous access to datasets for all tasks.

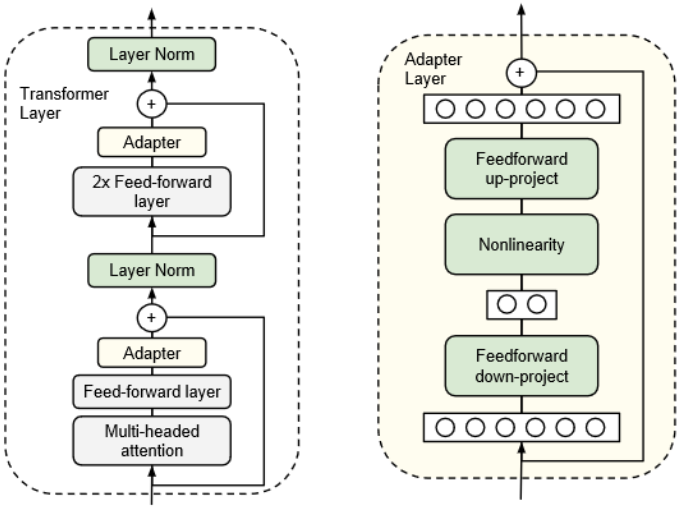


Figure 12: Adapter Learning Architecture from https://arxiv.org/pdf/1902.00751.pdf

References

1. Houlsby, Giurgiu, et al. “Parameter Efficient Transfer Learning for NLP”. https://arxiv.org/pdf/1902.00751.pdf

Reinforcement learning

Reinforcement learning is a machine learning framework where an AI agent explores an environment with a set of possible actions based on its current state and tries to maximize a sum of rewards. This framework has been applied to a diverse set of applications including robot navigation, factory process control, games, and operations research. To gain practice with reinforcement learning, an ML practitioner can use the test environments of the openAI gym (https://github.com/Farama-Foundation/Gymnasium).

More specifically, we denote the finite space of possible actions in the environment as A and the finite set of possible states in the environment as S. However, only part of the environmental state is actually observable by the agent at a given time. For example, in reinforcement learning applied to poker, the environment state is all of the players’ hands and community cards, whereas the observable state is just the agent’s hand and community cards (https://ai.stackexchange.com/questions/38977/when-is-it-necessary-to-explicitly-define-both-the-state-and-observation-space-i). Based on an agent’s current observations, the agent will take an action based on its policy $\pi(a_t|s_t)$ resulting in a reward and new

observations. During reinforcement learning, the policy is updated to try to maximize the sum of rewards, $\sum_{t=0}^{T} \gamma^t r_t$, where gamma is the discounting rate. Alternatively, the sum of rewards can be represented recursively using the Bellman equation:

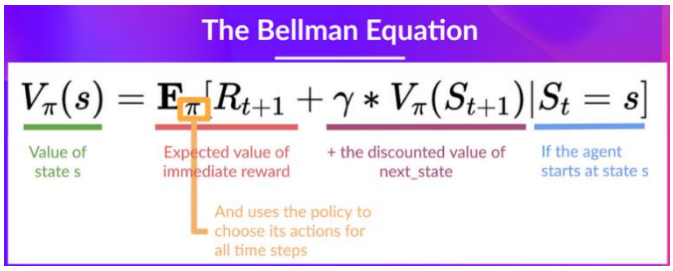


Figure 13: Bellman Equation from: https://huggingface.co/learn/deep-rl-course/unit2/bellman-equation

This equation says that the value of states is equal to an immediate reward plus a discounted value of the next state while the agent follows policy pi to choose its actions.

Some specific methods for achieving reinforcement learning are Q learning, deep Q learning, and E3. The Q function or the state-action value function attempts to model the expected return or the discounted sum of rewards. The Q function can be represented as either a table of values (the Q table) for each combination of (s,a) or can be approximated by a neural network, aka deep Q-learning.

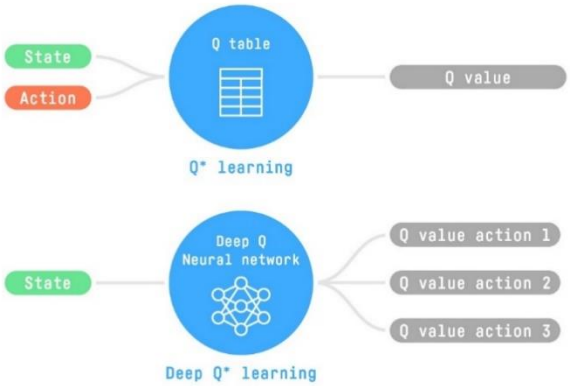


Figure 14: Q learning and Deep Q learning from: https://huggingface.co/learn/deep-rl-course/unit3/from-q-to-dqn

Additionally, we want to balance exploration of the environment versus exploitation (performing an action that maximizes Q). This balance can be achieved by taking the optimal action with probability, 1 – epsilon, and taking a random action (exploration) with probably epsilon. Over time, epsilon is reduced so the algorithm focuses on the optimal action. To achieve learning with Q tables, the Q function is updated after an agent takes an action using the Q update rule:

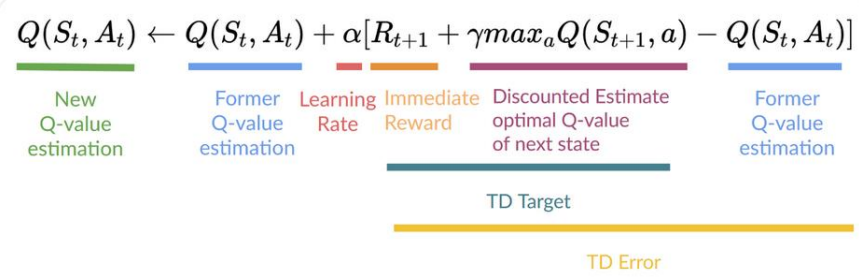


Figure 15: Q update equation from: https://huggingface.co/learn/deep-rl-course/unit2/q-learning

whereas in deep Q learning, gradient descent minimizes the q-loss:

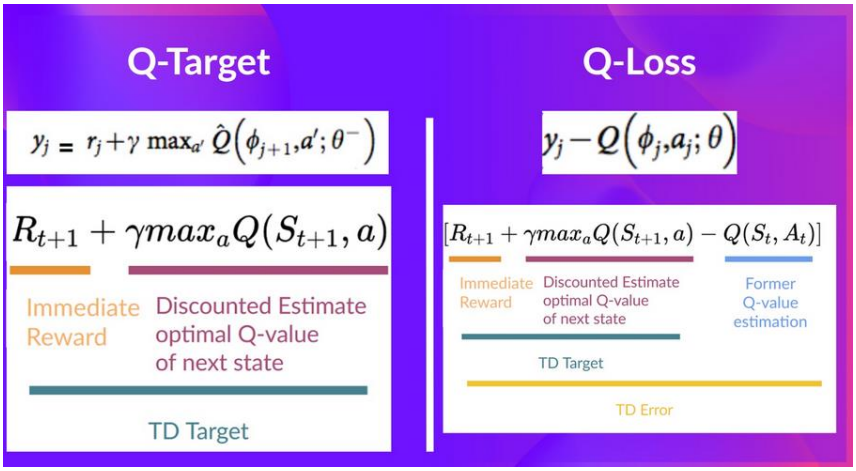


Figure 16: Q-Target and Q-loss from: https://huggingface.co/learn/deep-rl-course/unit3/deep-q-algorithm

References

1. https://web.eecs.umich.edu/~baveja/NIPS05RLTutorial/NIPS05RLMainTutorial.pdf
2. https://www.tensorflow.org/agents/tutorials/0_intro_rl
3. https://huggingface.co/learn/deep-rl-course/

Other Learning Methods

Auxiliary task learning

Some other recent learning methods in the literature include auxiliary task learning, prefix fine tuning, prompt fine tuning, and the IPT method (intrinsic prompt tuning). Briefly, auxiliary task learning consists of learning to accomplish auxiliary tasks with the main objective of having good performance on one or more primary tasks. For example, in computer vision, a primary task could be "Identifying facial landmarks on face pictures" with an auxiliary task of "Estimating head pose and predicting facial attributes" (from ref below). Formally, in auxiliary task learning, we use the loss gradient as: $G = \nabla\mathcal{L}_{primary} + \lambda G_{auxiliary}$ and $\nabla\mathcal{L}_{primary}$ is the gradient of the loss for the primary task with the auxiliary loss gradient as:

Method	Adjusted auxiliary loss gradient $G_{\text{auxiliary}}$
Unweighted cosine	$\begin{cases} \nabla\mathcal{L}_{\text{auxiliary}} & \text{if } \cos(\nabla\mathcal{L}_{\text{primary}}, \nabla\mathcal{L}_{\text{auxiliary}}) \geq 0 \\ 0 & \text{if } \cos(\nabla\mathcal{L}_{\text{primary}}, \nabla\mathcal{L}_{\text{auxiliary}}) < 0 \end{cases}$
Weighted cosine	$\max(0, \cos(\nabla\mathcal{L}_{\text{primary}}, \nabla\mathcal{L}_{\text{auxiliary}})). \nabla\mathcal{L}_{\text{auxiliary}}$
Projection	$\nabla\mathcal{L}_{\text{auxiliary}} - \min(0, \nabla\mathcal{L}_{\text{auxiliary}}. \frac{\nabla\mathcal{L}_{\text{primary}}}{\|\|\nabla\mathcal{L}_{\text{primary}}\|\|}). \frac{\nabla\mathcal{L}_{\text{primary}}}{\|\|\nabla\mathcal{L}_{\text{primary}}\|\|}$

Table 15: Loss Gradient equation from below reference

References

1. https://vivien000.github.io/blog/journal/learning-though-auxiliary_tasks.html#auxiliary-tasks-in-machine-learning

Prefix finetuning

Prefix finetuning is an alternative strategy to fine tuning that more efficiently tunes multiple tasks. In standard fine tuning, a copy of all model parameters is needed for each task:

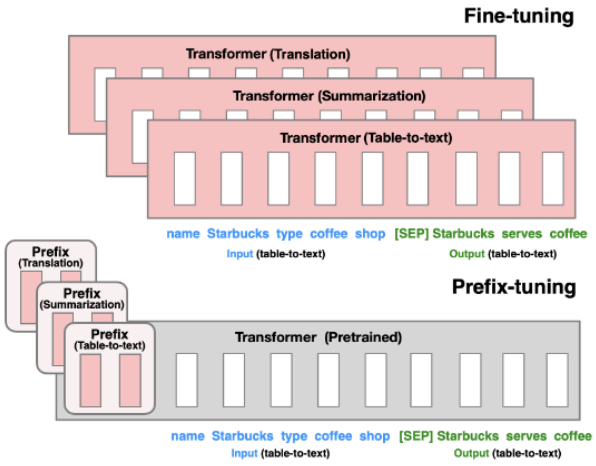


Figure 17: Schematic of prefix tuning vs fine tuning (https://arxiv.org/pdf/2101.00190.pdf)

In contrast, prefix tuning keeps the neural network model parameters frozen and only requires tuning of a small prefix ("sequence of continuous task-specific vectors", see ref below) for each downstream task. Literature experiments illustrate prefix tuning is effective for table to text generation while only requiring 0.1% of the total parameters for tuning. Additionally, prefix tuning performs well in low-data settings and extrapolation settings. However, in the summarization setting, prefix tuning achieved lower performance when compared to fine tuning.

References

1. https://arxiv.org/pdf/2101.00190.pdf

Prompt tuning and intrinsic prompt tuning

Prompt tuning is an alternative strategy to standard prompt engineering (hard prompts) where a person hand crafts a prompt to adapt a large language model to a specialized task. In contrast, in prompt tuning also called soft prompting, the AI tunes an embedding vector for each downstream task. Unfortunately, soft prompts are not interpretable in contrast to the interpretability of hard prompts.

Intrinsic prompt tuning (or IPT) is a variant of soft prompting for multi-task learning. IPT has two key stages: 1) identifying a low-dimensional intrinsic task subspace for multiple tasks, and 2) adapting the neural network to new tasks by tuning a small number of parameters in the identified subspace.

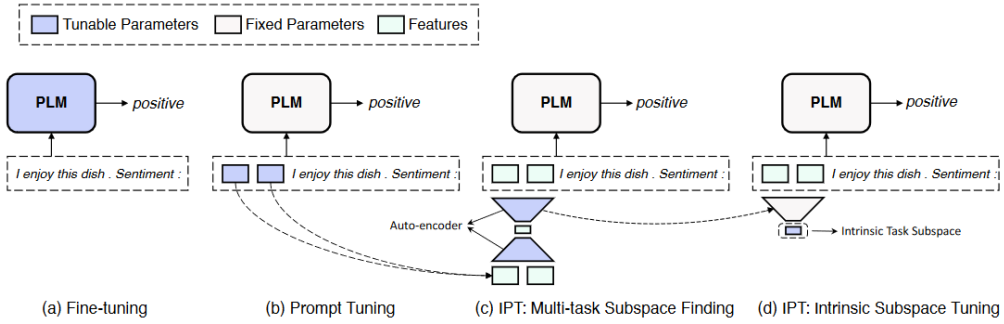


Figure 18: Comparison of fine tuning, prompt tuning, and IPT from: https://arxiv.org/pdf/2110.07867.pdf

Literature experiments show that the intrinsic task subspace identified across 100 tasks which is then tuned within this subspace achieves 90% of the performance of full prompt tuning (relative performance) when evaluated on unseen data. In addition, IPT achieves 83% relative performance when evaluated on unseen tasks while only using 250 tunable parameters.

References

1. https://research.ibm.com/blog/what-is-ai-prompt-tuning
2. https://arxiv.org/pdf/2110.07867.pdf

Semi-supervised Learning

Semi-supervised learning (SSL, see more details: https://arxiv.org/abs/2103.00550) is an approach that uses a combination of labeled and unlabeled data to improve the performance of pretrained models. The rationale behind SSL is that even though labeled data can be scarce and expensive to obtain, there is usually an abundance of unlabeled data available. By leveraging this unlabeled data, SSL can improve the generalization capabilities of the pretrained models.

One way to use semi-supervised learning with pretrained models is to fine-tune the model using a combination of labeled and unlabeled data. This can be achieved by employing several semi-supervised learning techniques, such as:

i. Self-training: In this approach, the pretrained model is initially fine-tuned using labeled data. The model is then used to predict labels for the unlabeled data. The most confident predictions are added to the labeled dataset and used for further fine-tuning of the model.
ii. Pseudo-labeling: Generate "pseudo-labels" for unlabeled data using the pretrained model's predictions and add the pseudo-labeled data to the training set. This expands the labeled dataset to improve the model.
iii. Consistency regularization: This technique aims to encourage the model to produce consistent predictions for different perturbations of the same input data. By enforcing this consistency, the model can learn to generalize better from the available labeled and unlabeled data.
iv. Mean Teacher: The Mean Teacher approach maintains two versions of the same model: a student and a teacher. The student model is trained on labeled data, while the teacher model is an exponential moving average of the student model. The consistency between the predictions of the student and teacher models is enforced, which helps in utilizing the information from the unlabeled data. (see details: https://github.com/CuriousAI/mean-teacher)
v. Adversarial Attacks: Using adversarial attacks on pretrained models with semi-supervision has shown substantial advances in classifying images. (see details: https://arxiv.org/abs/2308.04018)

Caution is essential to prevent the reinforcement of errors when utilizing unlabeled data. In summary, semi-supervised learning serves as an effective method for harnessing the potential of unlabeled datasets and enhancing the performance of deep learning models.

RLHF (Reinforcement learning from human feedback)

Reinforcement learning from human feedback (RLHF, see details: https://huggingface.co/blog/rlhf) is a technique that involves training a model using feedback from humans to improve its performance. In the context of pretrained models, RLHF can be used to fine-tune and adapt the model to specific tasks or domains by incorporating human expertise and preferences.

The RLHF process involves three core steps: “pretraining a language model (LM), gathering data and training a reward model, and fine-tuning the LM with reinforcement learning” (see above ref). The pretraining phase doesn't result in a perfect model; the model is expected to make mistakes and generate incorrect outputs. However, it provides a substantial starting point upon which RLHF can build, making the model more accurate, safe, and useful.

The process of using RLHF to improve the performance of pretrained models involves the following steps:

i. Collect human feedback: Obtain expert feedback on model predictions or behavior. This can be done by presenting model-generated outputs to experts and asking them to rate or rank the outputs based on their quality, relevance, or correctness.

ii. Create a reward model: Train a reward model using the collected human feedback. The reward model captures the relationship between the model's outputs and the feedback provided by the human expert.
iii. Fine-tune the pretrained model: Use the reward model as a guide to fine-tune the pretrained model using reinforcement learning techniques. This process involves updating the model's parameters to maximize the expected reward, which is estimated using the reward model.
iv. Iterate the process: The process of collecting human feedback, training the reward model, and fine-tuning the pretrained model can be iterated multiple times to continually improve the model's performance.

By incorporating human feedback and using reinforcement learning techniques, pretrained models can be fine-tuned to perform better in specific tasks or domains. This approach allows for the seamless integration of human expertise into the model, leading to improved performance and more reliable results. The common package in Python to implement RLHF is “TRL - Transformer Reinforcement Learning” (see details: https://huggingface.co/docs/trl/en/index).

Gradual unfreezing/Chain-thaw

The gradual unfreezing and chain-thaw approaches are alternatives to fine turning and transfer learning to improve model performance. Gradual unfreezing is discussed in Howard et al (https://arxiv.org/pdf/1801.06146) and Yosinski et al (https://arxiv.org/pdf/1411.1792). The idea here is to unfreeze the last layer and train for 1 epoch. Then unfreeze the next to last layer and train for 1 epoch the unfrozen layers. This process is continued until all layers are fine tuned. This approach has advantages to full fine tuning in which it minimizes catastrophic forgetting.

A related method is called chain-thaw as introduced in Felbo et al (https://arxiv.org/pdf/1708.00524). In contrast to gradual unfreezing, the chain-thaw approach trains a single layer at a time and trains each layer sequentially until convergence with all other layers frozen. In detail, chain-thaw trains any new layers with the rest of the layers frozen. Then layer 1 is trained with the remaining layers frozen; layer 2 is trained with remaining layers frozen, etc; and finally, the whole network is fine tuned. This procedure allows us to transfer knowledge (ie transfer learning) to a new domain while minimizing overfitting.

Prompt Engineering

Prompt engineering refers to the technique of carefully constructing the prompts or inputs fed into deep learning models to improve their performance. As deep learning models are trained on large datasets to recognize patterns, the way the input data is framed impacts how the model processes it.

With advanced large language models like GPT, LLaMa, etc., the prompt serves as the main mechanism for guiding the model - essentially the prompt defines the task for the model. As such, better prompts can enhance what the model pays attention to and improve the relevance of its response. For instance, consider the task of sentiment analysis. Instead of simply providing sentiment labeled sentences, the prompt could frame it as: “I will provide student survey responses. Please analyze if the response shows a positive or negative sentiment regarding the grading policy. Positive means students liked the setting of grading policy. Negative means students were unhappy about it. Please return ‘Irrelevant’ if the response is not related to the grading policy. Let’s analyze the following comment: I love this class, but it would be better if grading policy can be more transparent.”

Some best practices for prompt engineering include (see details: https://guides.library.georgetown.edu/ai/prompts):

- Using clear task-defining instructions - Prompts should clearly state the task the model is expected to perform rather than just providing input data. For example, "Please translate the following text from French to English" is more effective than just providing the text.
- Providing relevant context - Adding related contextual details allows the model to better interpret the intent and make connections.
- Structuring with examples - Providing the model with a few examples of desired inputs and outputs helps define the expected response format.
- Simplifying input data - Avoiding complex terminology or phrasing generally produces better responses from the model.

- Iteratively testing and tweaking - Start with a simple prompt and test model outputs, then tweak parts like the instructions, examples, or data simplification to steadily improve results.

Tracking model performance across prompt variations allows us to determine optimal prompts. Continual prompt tuning and A/B testing prompts on new model versions enables us to engineer increasingly effective prompts over time. The careful art of prompt engineering allows us to take advantage of the capabilities already within pretrained models instead of extensively retraining them, serving as a powerful tool for practitioners applying deep learning techniques.

Chapter 10: Model Interpretability

Perturbation-based methods

LIME

LIME (Local Interpretable Model-agnostic Explanations, Ribeiro et al. 2016) can interpret ANY trained black-box model using a local surrogate mechanism. The local surrogate model with interpretability constraint is expressed by:

$$explanation(x) = \underset{g \in G}{\operatorname{argmin}} L(f, g, \pi_x) + \Omega(g(K))$$

- g is the explanation model for instance x (e.g. linear regression model)
- L is a loss function to be minimized (e.g. mean squared error, MSE)
- f is the original model (e.g. an xgboost model)
- $\Omega(g(K))$ is the model complexity (e.g. fewer features = smaller K) such as LASSO and forward or backward selection
- π_x is the proximity measure to define the boundary of the neighborhood around instance x for the explanation
- K is the number of features

Figure 19 illustrates the original model (f)'s binary decision (dark blue vs. yellow) non-linear function. Given the instance being explained (bold red cross, **X**), a linear model of dashed line (g) is learned from f's prediction based on the perturbed instances in the vicinity of **X**. Be aware that g is trustful locally around **X**, not globally.

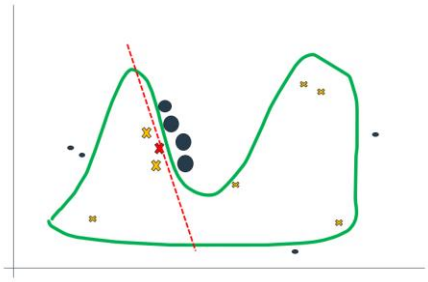


Figure 19: LIME schematic

LIME is a well-known interpretation method for classification models by identifying important features in the binary or multi-level class(es)-prediction. For example, when dichotomizing the following two sentences in such binary classes (1 for spam and 0 for normal comment), word(s) affecting the binary decision model may be of interest.

	CONTENT	CLASS
267	PSY is a good guy	0
173	For Christmas Song visit my channel! ;)	1

Table 16: Classification of each sentence (example from: https://christophm.github.io/interpretable-ml-book/lime.html)

To run the model-agnostic algorithm, multiple new texts are created by arbitrarily excluding words from the original sentence. For each new text, the prediction probability of class == 1 (prob) and the proximity of the new text to the original sentence (weight) are computed.

For	Christmas	Song	visit	my	channel!	;)	prob	weight
1	0	1	1	0	0	1	0.17	0.57
0	1	1	1	1	0	1	0.17	0.71
1	0	0	1	1	1	1	0.99	0.71
1	0	1	1	1	1	1	0.99	0.86
0	1	1	1	0	0	1	0.17	0.57

Table 17: Probability and weight table (Example from: https://christophm.github.io/interpretable-ml-book/lime.html)

Based on the probability and weight estimations, LIME evaluated local weights for each feature (word) as follows:

case	label_prob	feature	feature_weight
1	0.1701170	PSY	0.000000
1	0.1701170	guy	0.000000
1	0.1701170	good	0.000000
2	0.9939024	channel!	6.180747
2	0.9939024	;)	0.000000
2	0.9939024	visit	0.000000

Table 18: Feature weights (Example from: https://christophm.github.io/interpretable-ml-book/lime.html)

The word “channel!” resulted in a significant contribution with non-zero feature_weight == 6.18 in a high probability of spam (label_prob = 0.99).

In addition to the text mining on classification prediction, LIME has also an explanation capability of general prediction model trained by tabular data, consisting of numerical inputs and values. For example, the following figure shows LIME explanations with two features (numerical temp and categorical weather situation or season) from the trained model for two instances (a classification random forest with 100 trees): warmer temperature and good weather situation make a positive influence on the average number of bicycles rented above a trend line, which is a linear prediction from a multi-variable regression.

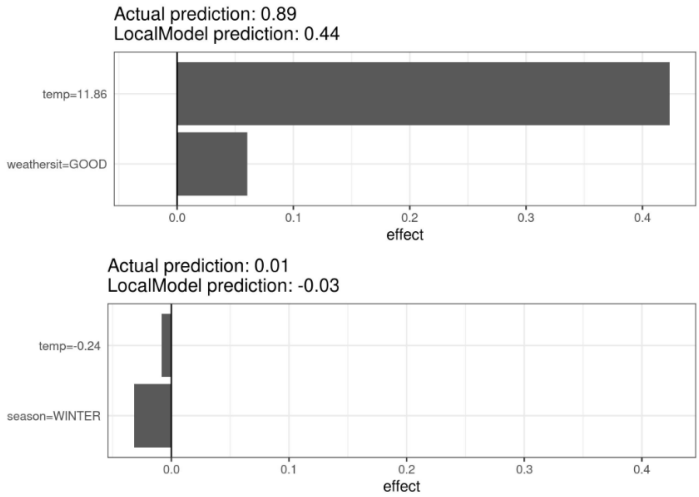


Figure 20: Effect of temperature and season (image from: https://christophm.github.io/interpretable-ml-book/lime.html)

- Image Data

  Let's see the example below: left showing the original image of a home-made bread, the middle presenting better prediction and explanation for "Bagel" with prediction probability of 77%, and the right displaying worse prediction and explanation for "Strawberry" with a prediction probability of 4% given two LIME explanations for "Bagel" and "Strawberry".

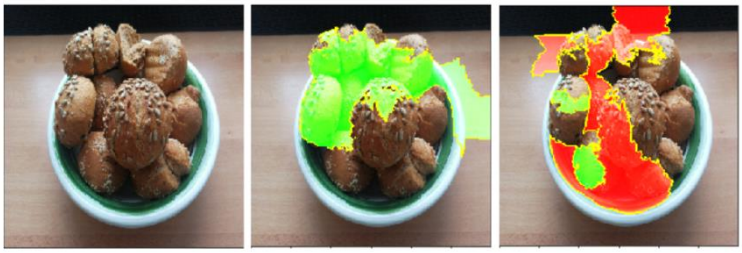


Figure 21: LIME explanations for bagel and strawberry (permission from author to use image from: https://christophm.github.io/interpretable-ml-book/lime.html)

SHAP

SHAP (SHapley Additive exPlanations, Lundberg et al. 2017) is the explanation model in SHAP is basically expressed as

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z_j'$$

- g is the explanation model.

- $z' \in \{0,1\}^M$ is the coalition vector containing simplified input features (e.g. for image data, if there are four aggregated super-pixels (i.e. M = 4) and a coalition {1,3} is simulated, the coalition vector should be {1,0,1,0} where 0 and 1 mean that the corresponding feature values are "present" and "absent", respectively).
- M is the maximum coalition size.
- $\phi_j \in \mathbb{R}$ is the feature attribution for a feature j, the Shapley values.

Examples:

- SHAP explanation force plots (explanations for individual predictions) using SHAP values to interpret the predicted cervical cancer probabilities of two women.

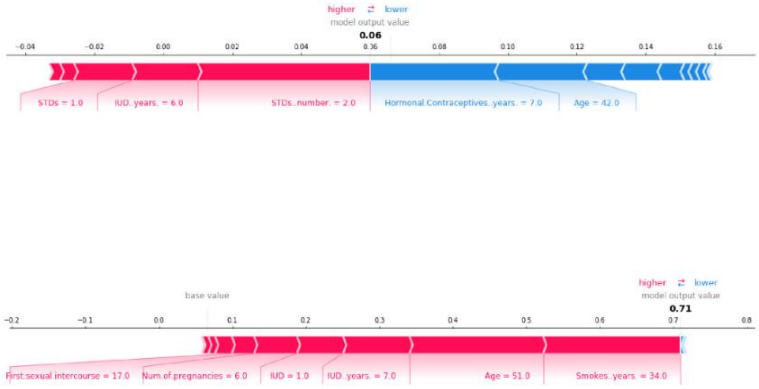


Figure 22: Example of SHAP force plots (image from: https://christophm.github.io/interpretable-ml-book/shap.html)

A risk for cervical cancer was predicted by training a random forest classifier (whether the biopsy result is "Healthy" or "Cancer") with 100 trees. Given the baseline (the predicted probability on average) of 0.066, the first woman (on top) returns a low predicted risk of 0.06 while the second woman (on bottom) shows high predicted risk of 0.71. For an example of the first woman, the low predicted risk was evaluated from the offset between increasing effects such as STDs (number of STD diagnoses) and IUD (number of years with an intrauterine device) and decreasing effects such as years of hormonal contraceptives and age.

- SHAP Feature Importance

Features with large absolute Shapley values are significant. Global explanation is driven by averaging the absolute Shapley values per feature across the matrix with each row per instance (i) and each column per feature (j).

$$I_j = \frac{1}{n}\sum_{i=1}^{n}\left|\phi_j^{(i)}\right|$$

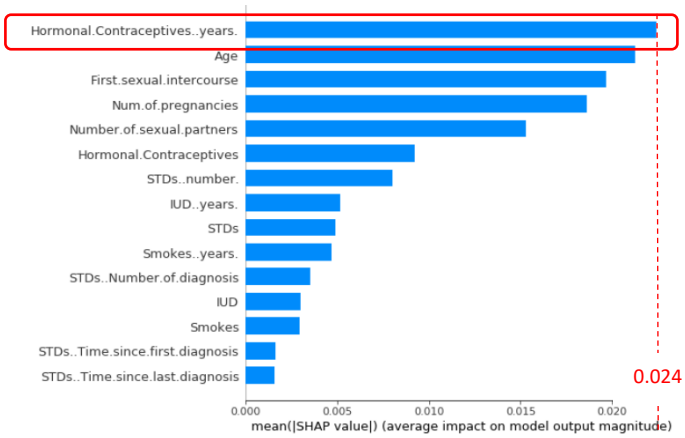


Figure 23: SHAP feature importances (image from: https://christophm.github.io/interpretable-ml-book/shap.html)

For instance, years with hormonal contraceptives is the most significant feature, adjusting the predicted probability of risk by 2.4% points on average.

- SHAP Summary Plot

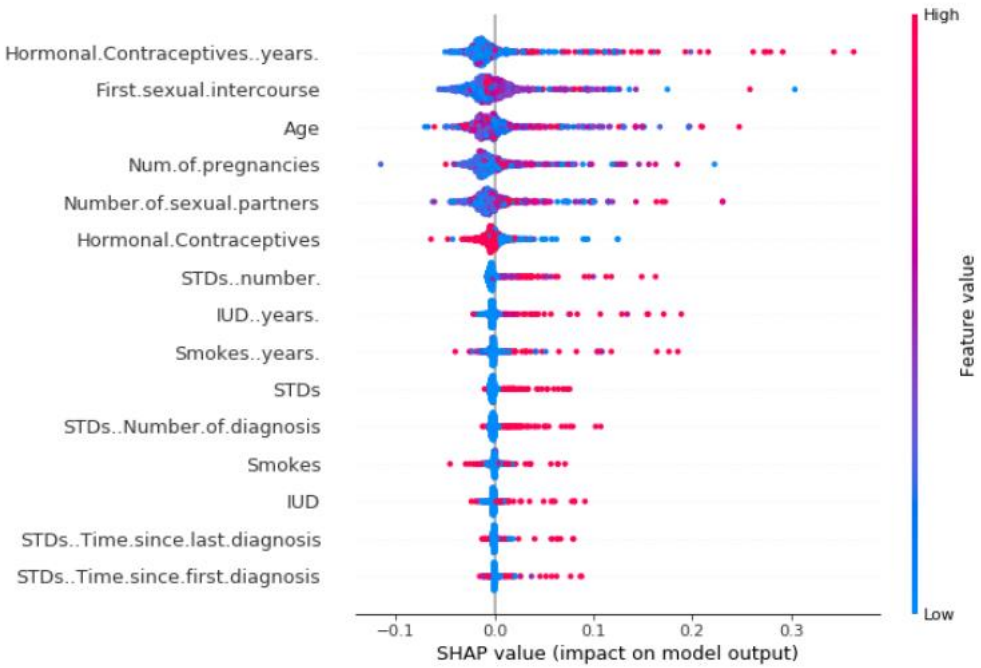


Figure 24: SHAP summary plot (image from: https://christophm.github.io/interpretable-ml-book/shap.html)

The above SHAP summary plot shows jittered points along with horizontal line where each point corresponds to Shapley value per instance per feature. For example, there is a relationship between the low predicted probability of risk and low number of years on hormonal contraceptives. Note that the color represents the feature value from low in blue to high in red.

Gradient-based methods

Saliency Maps (Simonyan et al. 2014)

For class c, $S_c(I)$ represents the predicted output of a neural network (e.g. CNN) for image I. The linear score model for the class c can be expressed by

$$S_c(I) = w_c^T I + b_c$$

- I is the input image in the form of a numeric vector.
- w_c is a weight vector.
- b_c is a bias of the model.

Interpretation is determined by the gradient of class score (S_c) of interest with respect to the input image (I) at specific pixel (I_0):

$$w_c = \frac{\partial S_c}{\partial I}|_{I=I_0}$$

Then, the gradients will be visualized, displaying the absolute values or highlighting negative and positive impacts.

In addition to the image interpretability, Saliency maps can be utilized for natural language explanations (NLEs) by discovering important features (tokens) and spans that are the most salient. A verbalized saliency map S_V (see: https://arxiv.org/pdf/2210.07222) is expressed by,

$$v\big(W, S = e(W, m)\big) = S_V$$

- v is any verbalizer (e.g. IG: Integrated Gradients), which discretizes attribution scores/saliency map (S) and produces a natural language representation, S_v.
- $W = (w_1, \dots, w_n)$ is a vector of source (input) tokens.
- e is a feature explanation method (e.g. a BERT model), which generates a saliency map.
- m is an underlying (explained) black-box model which predicts outcomes.

To make S_V concise for a human explanation, a coverage metric is defined by the proportion of coverage by the tokens involved in S_V out of the total attribution in S (see: https://arxiv.org/pdf/2210.07222),

$$Coverage(S_V) = \frac{\sum |v_i| \in S_V}{||S||}$$

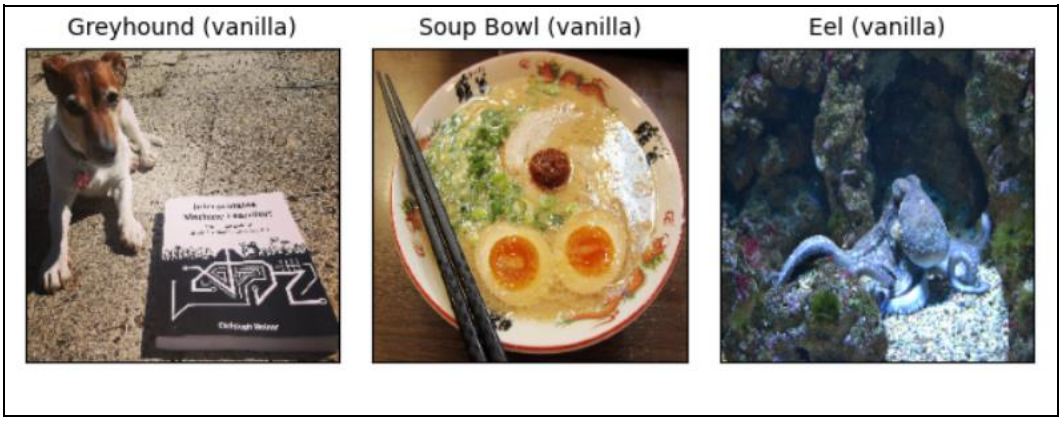
The example of Feldhus et al. (2022) (https://arxiv.org/pdf/2210.07222v1) illustrates the candidate generation and selection process with an instance from IMDb. This is an example when the sentiment analyzer (BERT model) wrongly predicted the negative movie review from IMDb as positive sentiment by focusing on the "wrong" tokens. The colors and darkness in each word indicate how important it was for the analyzer to predict "positive sentiment" by running instruction-based verbalizations SMV (Saliency Map Verbalizer) using GPT-3.5. Depending on the negative or positive saliency (importance) score for each word, the word was colored in blue or red. The darker color represents more departure from zero, hence more importance (see Table 1 of Feldhus et al. 2022 for more details).

Grad-CAM (Gradient-weighted Class Activation Mapping, Selvaraju 2019)

Grad-CAM is utilized to analyze which area is activated in the feature maps of the last convolutional layers by estimating importance weights. Grad-CAM decides the importance weights for each of the k feature maps to a class c of interest.

$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\delta y^c}{\delta A_{ij}^k}}_{\text{gradients via backprop}}$$

- y^c is the score of class c.
- A_{ij}^k is the activaction at pixel location (i, j) of the feature map k activation of a convolutional layer, A^k.
- α_k^c represents a partial linear transformation of the deep network downstream from A, and captures the "importance" of feature map k for a target class c.
- Z is the total number of dimensions encompassing $u \times v$ (i.e. $Z = \sum_i \sum_j 1$) when an input image is composed of $u \times v$ (width X height) pixels.
    - Examples



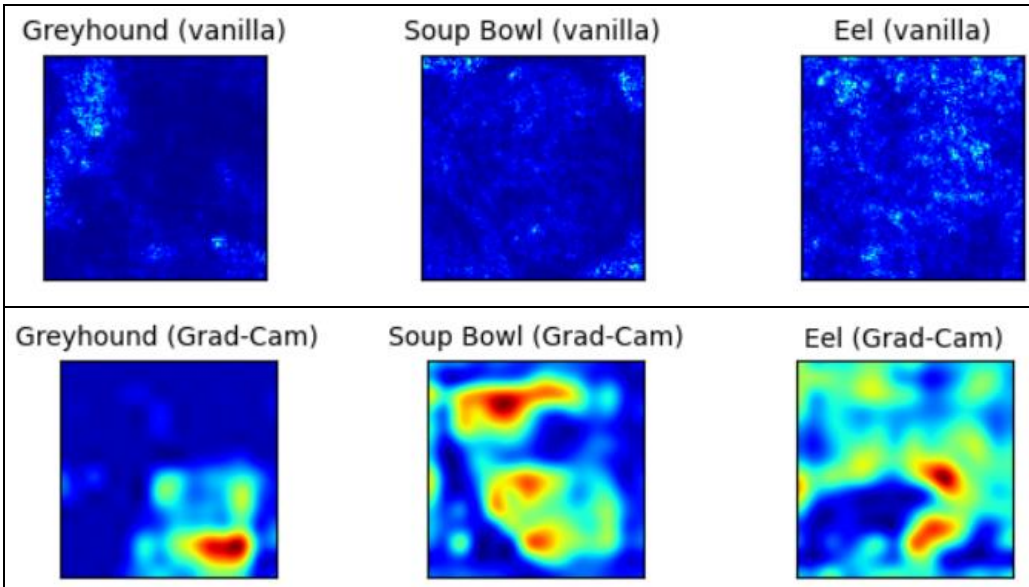


Figure 25: Comparison of vanilla gradients and Grad-Cam approaches (Image permission granted by author, image from https://christophm.github.io/interpretable-ml-book/pixel-attribution.html)

The first image was classified as “Greyhound” with a probability of 35%; the second image correctly fell into “Soup Bowl” with 50% probability; however, the third image was incorrectly classified as “Eel” with high probability of 70% although it was indeed octopus.

Saliency maps highlighted the dog area correctly but some area around the book oddly. Worse decision was made by Grad-CAM by highlighting only book area. For both soup bowl and octopus, Saliency maps fail to highlight important concepts. Grad-CAM highlighted around eggs and some upper part of bowl. But for octopus, the interpretation by Grad-CAM does not make sense.

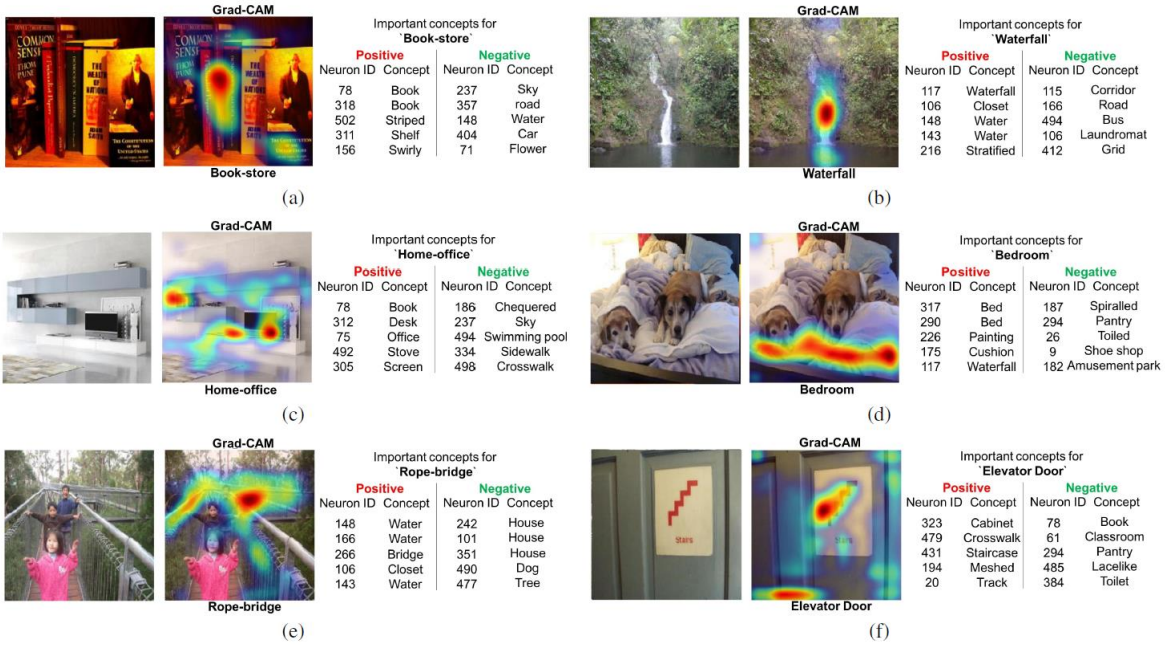


Figure 26: Grad-CAM explanations (permission granted from author)

Looking at the figures in (a)-(d), positively important neurons make sense intuitively while the negatively important neurons are not, explained by Grad-CAM. However, the figure in (e) shows mis-classification from the network and the figure in (f) demonstrates incorrect detection of doors.

Implementation

	Python	R packages
LIME	https://github.com/marcotcr/lime	**lime** R-package https://cran.r-project.org/web/packages/lime/lime.pdf **iml** R-package https://cran.r-project.org/web/packages/iml/iml.pdf
SHAP	https://github.com/slundberg/shap	**Shapper** R-package https://modeloriented.github.io/shapper/ **fastshap** R-package https://github.com/bgreenwell/fastshap
Saliency Maps	https://pypi.org/project/tf-keras-vis/	N/A
Grad-CAM	https://github.com/albermax/innvestigate https://github.com/marcoancona/DeepExplain	

Table 19: Software packages for LIME, SHAP, Saliency Maps, and Grad-CAM

Comparisons

	PROS	CONS
LIME	Regardless of machine learning model used for training, the **same** local and interpretable machine learning model can be used for explanation. LIME makes human-friendly and interpretable explanations, not like principal components as an example. LIME works for tabular data, text, and images. **Easy to implement** in Python and R	Hard to define optimal neighborhood when using LIME with tabular data (trying different kernel settings may be helpful until finding reasonable explanations) Instability of the explanations stemming from the sampling process Possibility of manipulation of explanations to hide bias
SHAP	Feature values are fairly evaluated from a solid theoretical foundation. **Fast implementation** for TreeSHAP, allowing to compute many Shapley values needed for the **global model interpretations**	KernelSHAP is super slow and ignores dependency among features. From TreeSHAP, unreasonable features can be selected as important from unlikely data points sampling. Possibility of manipulation of explanations to hide bias
Saliency Maps	Easy and fast to recognize the important regions (super-pixels) of the image. Compute faster than model-agnostic methods such as LIME and SHAP	Constant bias to all images can produce unreliable explanations for Saliency Maps. No evaluation method to assess their performance.
Grad-CAM		No R package published yet

Table 20: Comparison Table for LIME, SHAP, Saliency Maps, and Grad-CAM

Decision rule

If a user is much more comfortable with coding in R than Python, LIME and/or SHAP will be more feasible for the user. However, due to SHAP and LIME's computational inefficiency, Saliency Maps (or Grad-CAM) would be recommended to explain the model for NLP or computer vision in Python.

There are various evaluation methods for interpretable deep-learning in terms of three aspects: fidelity with respect to the original model, comprehensibility of the model, stability of interpretation (see details in https://www.mdpi.com/2078-2489/14/8/469). By examining the computational metrics, the best explainable method for NLP, Computer Vision, or tabular data can be selected among candidates.

References

1. Ribeiro et al. “Why should I trust you?” explaining the predictions of any classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016: 1135-1144.

2. Lundberg et al. A unified approach to interpreting model predictions. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

3. Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps.” arXiv preprint arXiv:1312.6034v2 (2014).

4. Selvaraju et al. Grad-CAM: Visual explanations from deep networks via gradient-based localization. 2019, arXiv.

5. https://christophm.github.io/interpretable-ml-book/interpretability.html

6. Feldhus, N., Hennig, L., Nasert, M.D., Ebert, C., Schwarzenberg, R., & Moller, S. (2022). Constructing Natural Language Explanations via Saliency Map Verbalization. ArXiv, abs/2210.07222.

Chapter 11: Model Reproducibility

Seed values

To create reproducible work (e.g. when validating trained-model using cross-validation), controlling seed values is crucial for others to generate the same outputs from original programmer's codes. The most general approach is to use np.random as below. The global random seed set by np.random.seed(number) affects all uses to the np.random.* module.

NumPy's Global Random Seed

```
np.random.seed(2023)
np.random.rand(5)
```

```
array([0.3219883 , 0.89042245, 0.58805226, 0.12659609, 0.14134122])
```

But, some imported packages or other scripts might reset the global random seed to another in larger projects. The unexpecting changes will affect outputs and hence make results unreproducible. To avoid the issue, the best practice is to create a random number generator.

Best Practice

```
import numpy as np
rng = np.random.default_rng(2023)
rng.random(5)
```

```
array([0.08805445, 0.22044004, 0.11317055, 0.44296683, 0.69733142])
```

Another benefit using the random number generator is transparency where we can see exactly what random number generator is used in each part of project by passing a random number generator to functions that need to be reproducible. For details, see: https://builtin.com/data-science/numpy-random-seed

The generated numbers from the best practice are different from the global random seed, as we can see. For such a case where a user wants to replicate the old global random seed results, try the following module,

A generator for old method

```
rng = np.random.RandomState(2023)
rng.rand(5)
```

```
array([0.3219883 , 0.89042245, 0.58805226, 0.12659609, 0.14134122])
```

In addition to "NumPy" package, visit the website (https://wandb.ai/sauravmaheshkar/RSNA-MICCAI/reports/How-to-Set-Random-Seeds-in-PyTorch-and-Tensorflow--VmlldzoxMDA2MDQy) for the

code snippets which can be used in the codebase to set up random seeds for all the involved libraries in a PyTorch or Tensorflow pipeline.

Non-deterministic algorithm: GPT

It has been well-known that later versions of GPT-4 and GPT-3.5-turbo are non-deterministic, even at temperature=0.0. In general, (Sparse) MoE (Mixture of Experts: combining multiple experts and a trainable gating network that chooses a subset of experts for each sequence example) approaches are regarded as the reason of the inherent non-determinism. Nevertheless, in practice, by summarizing multiple GPT outputs via GPT again, we anticipate the summary output to become less variable across different implementations.

Caching in Python

By freezing the state of data/outputs/results in cache, reproducibility of code is guaranteed. In practice, multiple caching methods with example Python code are introduced (**https://www.scaler.com/topics/python-cache**): using Python dictionary, using @lru_cache decorator, and using external caching services. The example displays utilization of the decorator @lru_cache (part of the functools module in Python), which is useful in the case of reproducibility and recursion making it faster.

The decorator stores the output of each function call (findNthFib), and when it encounters a recursive function call whose output is present in the cache, it does not compute the output of the recursive call again. It instead uses the output stored in the cache, making the computation time faster and outputs reproducible.

Python Markdown

Firstly, install Python-Markdown under your environment in the terminal by running the command line, **$pip install markdown**.

- Create Markdown string.
- Convert the markdown (markdown_string) to HTML (html) using the markdown method from the markdown package.
- Create a sample.html file and writing the HTML (html) to it.

For more detailed examples, see https://www.scaler.com/topics/python-cache to convert a Markdown file to HTML in Python.

R Quarto

Quarto is a multi-language, next generation version of R Markdown from RStudio, with many new features and capabilities. The latest release (download from https://posit.co/download/rstudio-desktop/) of RStudio is highly recommended when running Quarto within RStudio. See the reference (https://quarto.org/docs/computations/r.html#chunk-options) for more details.

Differences between R Markdown and R Quarto

Quarto **chunk options** are typically included in special comments at the top of code chunks rather than within the line that begins the chunk. Quarto uses this approach to both better accommodate longer options like "fig-cap" as well as to make it straightforward to edit chunk options within more structured editors that only have a difficult way to edit chunk metadata.

Quarto includes many more built-in **output formats** (and many more options for customizing each format). Quarto also has native features for special project types like websites, books, and blogs (rather than relying on external packages). In Quarto, "format" key rather than the "output" key is used as below.

R Quarto	R Markdown

title: "My Document"
format:
  html:
  toc: true
  number-sections: true
  css: styles.css
	title: "My Document"
	output:
	  html_document:
	  toc: true
	  number_sections: true
	  css: styles.css

Table 21: Comparison of R Quarto and R Markdown

References

1. https://builtin.com/data-science/numpy-random-seed
2. https://quarto.org/docs/computations/r.html
3. https://enjoymachinelearning.com/blog/is-gpt-3-deterministic/?expand_article=1
4. https://152334h.github.io/blog/non-determinism-in-gpt-4/
5. https://wandb.ai/sauravmaheshkar/RSNA-MICCAI/reports/How-to-Set-Random-Seeds-in-PyTorch-and-Tensorflow--VmlldzoxMDA2MDQy
6. https://realpython.com/lru-cache-python/#:~:text=One%20way%20to%20implement%20an,the%20least%20recently%20used%20entry.
7. https://www.scaler.com/topics/python-cache/

Chapter 12: Model Deployment

Deep learning (DL) has emerged as a powerful tool in pharmaceutical research and development. Deploying these models efficiently and reliably is crucial for maximizing their impact. The deployment process involves transitioning models from development environments like HPC to production-ready applications, such as APIs or dashboards. This section will discuss the usage of high-performance computing (HPC), Posit Connect, database systems (e.g., Oracle, Postgre, Snowflake), data storage solutions (e.g., AWS S3, HPC), and some potential challenges like GPU and memory limitations.

At Biogen R&D, our utilization of High-Performance Computing (HPC) environments is integral to providing the computational horsepower required to train sophisticated deep learning (DL) models. We have developed a set of best practices to ensure the efficacy of this process. It is suggested to establish a virtual environment specifically tailored to organizational needs, incorporating all requisite deep learning frameworks and GPU drivers, ensuring readiness for use.

For the acceleration of neural network training, it is recommended to employ GPU acceleration, taking advantage of GPUs within the HPC infrastructure. These GPUs are equipped with Tensor Cores optimized for tasks such as deep learning training and inference, enabling mixed-precision matrix multiplications at a faster rate than traditional CUDA cores.

It is suggested to manage large datasets and complex architectures through distributed training across multiple GPUs or nodes, coupled with the implementation of model checkpointing. This allows for the resumption of training from the last saved state in the event of interruptions. The recommendation extends to facilitating collaboration via version control systems like Git for effective code and model version management.

Transitioning DL models from development to production is recommended to follow a meticulous and detailed approach, including encapsulating models within Docker containers or creating shared virtual environments to ensure consistent behavior across environments. Robust pipelines for data preprocessing, model inference, and result post-processing are recommended. Additionally, incorporating error handling and logging mechanisms is suggested for troubleshooting, along with comprehensive testing through unit, regression, load testing, and user acceptance testing (UAT) to identify potential bottlenecks. It is crucial to document all test cases and validation steps clearly to maintain the integrity of the deployment process. This documentation should align with the standards of the software development lifecycle to ensure traceability, reproducibility, and compliance with regulatory requirements.

The deployment strategy is recommended to enhance model availability via APIs or dashboards for user-friendly access to predictions and insights, utilizing frameworks like Flask or FastAPI for APIs and tools like RShiny, Streamlit, Chainlit, or Dash for dashboards. Both APIs and Dashboards can be seamlessly deployed on Posit Connect server. Security measures such as token-based authentication, user access control, and encryption are recommended to protect sensitive data.

Additionally, deploying large language models (LLMs) like GPT-3.5, GPT-4, and open-source alternatives such as LLaMA2 and WizardLM presents unique strategic considerations and challenges. A critical concern is data privacy, as LLMs process vast amounts of potentially sensitive information, necessitating stringent measures to ensure compliance with data protection regulations. Inference speed is another

critical issue, as the real-time application of these models requires rapid response times not easily achieved with complex LLMs. For commercial models such as GPT-3.5 and GPT-4, deployment on cloud platforms like Azure provides a scalable and secure environment, benefiting from Microsoft's infrastructure and expertise in managing data privacy and computational efficiency. Open-source models, such as LLaMA and WizardLM, offer flexibility in deployment options, including Databricks and High-Performance Computing (HPC) environments, utilizing frameworks such as vLLM to optimize performance and manage resource allocation effectively. This approach allows for leveraging the specific advantages of each platform to address the challenges associated with deploying LLMs, including balancing inference speed with data privacy and computational resource requirements.

Database integration is a key component of the model deployment workflow, enabling effective data storage, retrieval, and management in a production environment. This includes connecting ML/DL pipeline to various databases for real-time data access and ensuring data consistency and integrity through well-defined schemas and appropriate data validation processes.

For unstructured data storage, cloud-based solutions like AWS S3 or HPC are recommended for scalable and cost-effective storage options. Data partitioning and indexing strategies are suggested to improve data retrieval speeds, with regular data archiving and backups to prevent data loss.

Addressing challenges such as GPU limits and memory constraints, particularly for large-scale model training, is recommended through strategies like multi-GPU training, distributed training across multiple nodes, and leveraging cloud-based GPU resources as needed.

Monitoring for scalability and performance issues is crucial as user numbers and requests grow. Implementing monitoring mechanisms to detect and address any degradation in model performance over time is recommended to maintain efficiency and overall performance of deployed models.

In conclusion, the deployment of deep learning models and large language models at Biogen incorporates a comprehensive set of best practices designed to optimize computational resources, ensure data privacy, and maintain high inference speeds. The strategic deployment of both commercial and open-source models across suitable platforms, from Azure to Databricks and HPC environments, underscores a commitment to innovation while addressing the inherent challenges of working with sophisticated AI models. These practices not only facilitate the seamless transition of models from development to production but also ensure that Biogen remains at the forefront of leveraging AI to drive advancements in the biotechnology sector.