Fakultät Maschinenwesen, Institut für Mechatronischen Maschinenbau
Professur für Werkzeugmaschinenentwicklung und adaptive Steuerungen

MASTER'S THESIS

Multi-task learning for holistic data-driven models of engineering systems

submitted in partial fulfillment of the requirements for the degree "Master of Science"

Anurag Trivedi

born on	13.06.1991
in	Unnao, U.P, India
submission date	27.01.2023
1st reviewer	Prof. Dr-Ing.Steffen Ihlenfeldt
2nd reviewer	Dr.-Ing.Hajo Wiemer

TECHNISCHE UNIVERSITÄT DRESDEN
Faculty of Mechanical Science and Engineering Institute of Mechatronic Engineering
Chair of Machine Tools Development and Additive Controls

Master Thesis Nr.: 16

for Trivedi, Anurag
Study number: 4821519
Course of studies: Computational Modelling and Simulation
Field of study: Computational Life Science

Topic: Multi-task learning for holistic data-driven models of engineering systems

Multi-task learning (MTL) has already been a successful learning strategy in many deep learning applications. A wide variety of modelling approaches have been developed. Still, they have found little application in the engineering sciences so far. Whereas data-driven models in engineering often represent only a small part of the behaviour of a system, i.e. represent only one parameter.
The aim of this thesis is to analyse the potential of MTL for application to tabular data from engineering, allowing data-driven models to represent a more holistic view of the system behaviour. To this aim, approaches in the literature to create synthetic datasets with a defined task relationship and MTL methods are to be reviewed first. Then, these methods are to be implemented, adapted and compared with a single-task learning baseline. Firstly on synthetic data to create different scenarios of task relationships and secondly on real-world datasets for the validation. The investigated real-world datasets are from the domain of process and material engineering.

Main tasks of this thesis:

- Literature review on the generation of synthetic tabular data for MTL, characteristics of the task relationship and methods of MTL for tabular data
- Implementation of methods for synthetic data generation to examine MTL approaches with different scenarios regarding the task relation
- Adjustment and further development of the methods of MTL from the literature
- Comparison of MTL with STL-Baseline for different scenarios on synthetic data
- Validation of the methods on real-world datasets
- Establishing evaluation criteria for data sets on the applicability of MTL

Supervisor:	Dr.-Ing. Hajo Wiemer, Dipl. Ing. Felix Conrad
Issued on:	05.08.2022
To be submitted on:	06.01.2023

The diploma examination regulations of the Faculty of Mechanical Engineering as well as the instructions for the preparation of student research projects and diploma theses issued by the Chair of Machine Tool Development and Adaptive Control Systems must be observed.

Prof. Dr.-Ing. S. Ihlenfeldt
Supervising university lecturer

Statement of Authorship

I, Anurag Trivedi hereby declare that this Master's Thesis on the topic

Multi-task learning for holistic data-driven models of engineering systems

was written completely independently by myself and did not use any other sources and aids other than those cited in the text. The project is handed over to the Fakultät Maschinenwesen, Institut für Mechatronischen Maschinenbau at TU Dresden.

Dresden, 27 . January 2023

Anurag Trivedi

Abstract

Humans are resourceful; they can analyze and solve tasks from all kinds of domains. By drawing on prior experiences, we are able to adapt quickly to any new task and accumulate information, allowing knowledge to be transferred across different scenarios. Multi-task learning is an approach that builds on this idea: constructing a common framework with shared components, improves generalization abilities while also being more economical in terms of memory, inference speed, and data requirements than individual learning. This study will take a comprehensive look at multi-task learning from varied angles.

Our first aim is to provide an overview of single-task learning and multi-task learning. We discuss the effectiveness of relative task-relatedness as well as some modern approaches to multi-task network design in a supervised and efficient manner. we then demonstrate how task-relatedness and task-specific features can be incorporated into a multi-task network design.

We conducted further introspection into multi-task learning's benefits with experiments using synthetic data to analyze how generalization behaves according to the complexity of tasks, such as the number of features, relatedness between tasks and data points, and training with an increasing number of learning tasks. To confirm our hypothesis, we also sampled real data from material and process engineering for our task-relatedness learning framework that automatically generates knowledge for better generalization. The results provide an optimistic outlook for building machine-generated wisdom and revealing fresh perspectives on deep learning's generalization capacities. Finally, part of our investigation examined whether multi-task learning approaches always prove superior then single-task learning.

Contents

Nomenclature

Symbols E Notations

D dataset

X dataset inputs

Y dataset outputs

x_n single data point

y_n single data label

$x_n(i, j)$ element at row i column j in data point x_n

$\hat{y}_n$ predicted model output on input x_n

ϵ

a random variable

λ task weighting

τ

temperature

$f_\omega(\cdot)$ a function f with parameters ω

$N(\cdot)$ Gaussian (normal) distribution

$\|\cdot\|$ norm-1 distance

$\|\cdot\|_2$ norm-2 distance (Euclidean norm)

$[\cdot;\cdot]$ concatenation

1 Introduction

Artificial intelligence (AI) has become a mainstay in the field of engineering. In recent years, many of AI's weaknesses have been overcome and the benefits of AI have increased enormously in many application areas. Also, multi-task learning was viewed as a cumbersome process. However, with the advancement in the era of artificial intelligence, multi-task learning is extensively studied and is applied to the problem of decision-making, since making decisions in the real world often involves multiple complex factors and criteria.

When a system is faced with a new scenario, it may try to learn from the previous experience. In this way, the system can deal with different scenarios with different tasks in parallel. Machine learning techniques can be used to identify the most important features of a complex system and then represent those features in a way that humans can understand and learn simultaneously. Multi-task learning models have been extensively studied during the last decade because of their importance in various online as offline engineering scenarios for diagnosis and self/multi-modal automatic parameter tuning. It is shown that multi-task learning with good initialization can outperform from single-task learning in various tasks, e.g. image classification, and regression problems.

1.1 Motivation

Engineers are often required to make predictions about the future state of their systems. These predictions can be made from data, models and other information. They need to understand what these predictions mean for the system, and decide how much trust should be put in them. Machine learning is playing an increasingly important role in the world of engineering, powering the automation and optimization of a wide range of systems. In order to power these systems, machine learning algorithms are trained on datasets that often contain both data about the system's outputs as well as input variables. This has traditionally been a limiting factor for engineers, who would need to either manually preprocess their dataset or use to separate models—for different task related outputs. A new concept called "multi-task learning" has emerged to help engineers overcome this challenge by letting them build models with multi-task outputs simultaneously.

It is essential to utilize all available means in order to satisfy the demand for high-quality

products efficiently in order to satisfy the complexity, dynamics, and sometimes even chaotic nature of manufacturing systems. one such example is the design of aircraft wings. In the past, engineers would focus on designing wings that were optimized for either lift or drag, but not both. However, by using a data-driven model that takes into account both lift and drag, it is possible to find a more optimal design. This type of multi-task learning can also be applied to other areas of engineering, such as the design of electric motors or the control of robotic systems. By taking into account multiple objectives simultaneously, machine learning models can help engineers find solutions that are more efficient and effective than those found using traditional methods.

2 Fundamentals

2.1 Machine Learning

Machine learning is a subfield of artificial intelligence that involves the use of algorithms and statistical models to enable a system to improve its performance on a specific task through experience. This means that a machine learning system can learn from data, identify patterns in the data, and make predictions or decisions based on those patterns.

The fundamentals of machine learning include concepts such as supervised and unsupervised learning, regression and classification, overfitting and regularization, and bias and variance. These concepts provide the building blocks for developing machine learning models and algorithms that can learn from data and make accurate predictions or decisions. In addition to these concepts, the fundamentals of machine learning also involve techniques such as feature engineering, data preprocessing, model selection, and evaluation. These techniques are used to prepare the data for machine learning, select the appropriate machine learning model, and evaluate its performance. Overall, the fundamentals of machine learning are essential for understanding and working with machine learning algorithms and models. They provide the foundation for developing effective and accurate machine-learning systems that can learn from data and make intelligent decisions.

I would like to restrict the scope of this work to supervised machine learning a bit more. In this thesis, it is defined, to assume that the algorithm gets provided some training set a total of N_{train} samples are included in each X_{train} example. The label is provided in the set Y_{train} With some objective. The algorithm can measure the gap between its prediction and the actual value. The outcome of this procedure is a prediction model. There is also a set of X_{test} tests that are never given to the algorithm or the thing to remember about machine learning is that it's a model for learning. It only exists to compare the predictions of the model. So we created a model with the actual results of Y_{test} and also rated the algorithm prediction improvement.

2.2 Deep Learning

As a subset of machine learning, deep learning uses artificial neural networks. Neural networks are modeled after the brain and consist of many interconnected processing nodes, or neurons. As humans, deep learning algorithms learn by example. Deep neural networks are algorithms used to automatically extract features from data that can be used to predict future activity or plan actions more effectively. Deep learning is an umbrella term for a variety of techniques in analytical computing. Artificial intelligence and cognitive science are deep learning's roots, with the goal of simulating human intelligence in computers. Like machine learning, deep learning can be used for both supervised and unsupervised tasks. However, deep learning is unique in its ability to learn from data that is unstructured or unlabeled. Recent advances in hardware and software have allowed deep learning to be applied to pattern recognition, image classification, natural language processing, and time series prediction.

One of the key benefits of deep learning is that it does not require extensive feature engineering. A lot of deep learning systems can learn from raw data and scale as more data is fed into the system. Deep learning is powered by neural networks, which are groups of interconnected nodes or cells. Each node corresponds to a unit of computational action, and the connections between nodes correspond to the flow of data. Nodes can learn complex patterns from data by adjusting the weights assigned to each connection.

There are three major types of neural networks: supervised learning algorithms, unsupervised learning algorithms, and reinforcement learning algorithms. Supervised learning is where the network is given a set of training data, and it learns to map input data to output labels. Unsupervised learning is where the network is given only input data, and it must learn to find patterns and relationships in the data. Reinforcement learning is where the network is given a set of rules or objectives, and it must learn how to best achieve those objectives.

2.3 Supervised learning

Supervised learning requires a dataset that has both features (the data that will be used to make predictions) and labels (the correct answers). The training data is fed into the learning

algorithm, which creates a model that can be used to make predictions on new, unlabelled data.This type of learning is divided into the categories of classification and regression. In order for the predictions to be accurate, it is important to have a large and representative dataset. The advantage of supervised learning is that it can be used to solve complex problems with high accuracy. However, it does require labeled data, which can sometimes be difficult or expensive to obtain.

- **Classification:** Classification algorithms are a type of algorithm that accurately assigns and classifies test data into certain categories (discontinuous values). It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Some common classification algorithms are linear classification, support vector machines, decision trees, k-nearest neighbors, and random forest.
- **Regression** Regression is a technique used to predict a continuous target variable based on one or more predictor variables. The goal of regression is to find the best-fit line or curve that describes the relationship between the predictor variables and the target variable. Regression analysis is a powerful tool that can be used to understand which factors influence the target variable and how much they influence it. Additionally, a regression can be used to make predictions about future values of the target variable based on new values of the predictor variables.

  There are many different types of regressions, but some of the most popular are linear regressions, logistic regressions, and polynomial regressions. Each type of regression has its own strengths and weaknesses, so it's important to choose the right one for your data and your goals.

2.4 Optimization Technique in supervised learning

Optimization is one of the core components of machine learning[77]. The basic concept of machine learning is to define the objective and optimize for the problem. In most machine learning models, the objective function parameters are learned from the given data by building an optimization model. As the era of immense data becomes more prevalent, numerical optimization algorithms have a dramatic influence on the popularity and applica-

tion of machine learning algorithms. Researchers have emphasized the need for developers to use machine learning models in their research. In order to promote the development of these models, a variety of successful optimization techniques were proposed Forward pass, which improves the performance and efficiency of machine learning methods, has been improved by these advances.

In supervised learning, the optimization method's goal is to find the optimal mapping function to minimize the loss of training samples.

$$min_\theta \frac{1}{N} \sum_{i=1}^{N} = L(y^i, f(x^i, \theta)), \tag{2.1}$$

where θ are the parameter of a mapping function and N is the number of training samples. X^i is the feature vector of the ith sample, y^i is the corresponding output in each case (label), and L is our usual loss function.

The easiest way of solving regression problems is by using the square of Euclidean distance as the loss function and minimizing errors in the data. The generalization performance of this form of empirical loss is not necessarily good. However, a more common solution to these kinds of problems is structured risk minimization, which can be done with an SVM. Regularization items are usually added to relieve overfitting for models where the objective function is nonconvex. in terms of L_2 norm,

$$min_\theta \frac{1}{N} \sum_{i=1}^{N} = L(y^i, f(x^i, \theta)) + \lambda \parallel \theta \parallel_2^2, \tag{2.2}$$

where λ is the parameter, which can be determined through cross-validation.

2.5 Optimizaton method

When it comes to optimization methods in machine learning, there are three main types: first order, higher order and derivative free. First order methods are the most common and include gradient descent, which is a method that calculates the slope of the cost function at a given point and then moves in the direction that minimizes the cost. Higher order methods are less common but can be more effective in certain situations. They include Newton's

Method, which uses second derivatives to find the minimum of a function, and conjugate gradient, which uses a combination of first and second derivatives. Derivative free methods are used when it's difficult or impossible to calculate derivatives, such as when dealing with discrete data or black-box functions. One popular derivative free method is evolutionary algorithms, which mimic natural selection to find optimal solutions. Optimization is a widely researched topic in both industry and academia. The following section shows the benefits of using different optimisation algorithms for machine learning and deep learning models.

2.5.1 A. First Order Method

In this section, we highlight some of the most commonly used first-order methods such as gradient descent as a means to solve problems.

- **Gradient Descent:** The gradient descent method is one of the earliest optimization methods. In it, variables update iteratively in the opposite direction of gradients of an objective function. This process is only executed as long as a learning rate η determines what step size each iteration needs to take in order to gradually converge to the optimal value. Gradient descent has a few disadvantages including that it makes it hard to find the global minimum and its reliance on sensitive randomization parameters which can limit the utility of its optimizations.

  First, the mathematical expression of gradient descent is given. For a linear regression model, it is supposed that $f_\theta(x)$ is the function to be learned, $L(\theta)$ is the loss function, and θ is the parameter to be optimized. The goal is to minimize the loss function by using

$$L(\theta) = \frac{1}{2D}\sum_{i=1}^{D}(y^i - f_\theta(x^i))^2, \tag{2.3}$$

$$f_\theta(x) = \sum_{j=1}^{M}\theta_j x_j \tag{2.4}$$

  where D is the number of training samples, M is the number of input features, x^i is an independent variable with $x^i = (x_1^i, ..., x_N^i)$ for $i = 1, ..., D$ and y^i is the target output. The gradient descent alternates the following two steps until it converges:

- Calculate the gradient corresponding to each θ_j by calculating Derivative of $L(\theta)$.

$$\frac{\partial L(\theta)}{\partial(\theta)} = -\frac{1}{D}\sum_{i=1}^{D}(y^i - f_\theta(x^i))^2 x_j, \tag{2.5}$$

- To minimize the risk function, each θ_j value needs to be updated in the negative gradient direction.

$$\theta_j' = \theta_j + \eta\frac{1}{D}\sum_{i=1}^{D}(y^i - f_\theta(x^i))^2 x_j^i \tag{2.6}$$

The gradient descent method is simple to implement, which makes it a global optimizer when the objective function is convex. However, it can be slower if the variable is closer to the optimal solution, and careful iterations need to be performed.

- **Stochastic Gradient Descent:** Stochastic gradient descent (SGD) is a popular optimization technique for training machine learning models. The idea of stochastic gradient descent is using one sample randomly to update the gradient per iteration, instead of directly calculating the exact value of the gradient[77]. This makes SGD much faster than other methods, such as gradient descent, which require waiting until all the data has been processed before making an update. For this reason, SGD is particularly suitable for large-scale data. However, this also means that SGD is more susceptible to noise and can sometimes lead to sub-optimal solutions. Despite these drawbacks, SGD is widely used in practice due to its computational simplicity and effectiveness on large-scale datasets.

  - The loss function (2.3) can be written as follows:

$$L(\theta) = \frac{1}{D}\sum_{i=1}^{D}(y^i - f_\theta(x^i))^2 = \frac{1}{D}\sum_{i=1}^{D} cost(\theta, (x^i, y^i)). \tag{2.7}$$

  - $L^*(\theta)$ is the loss function . if we randomly selected i in SGD

$$L^*(\theta) = cost(\theta, (x^i, y^i)) = \frac{1}{2}(y^i - f_\theta(x^i))^2 \tag{2.8}$$

  - One of the features of the SGD update is that it only uses a random sample i

rather than using all samples in each iteration.

$$\theta_j' = \theta_j + \eta(y^i - f_\theta(x^i))x^i \tag{2.9}$$

Since SGD uses a single input sample for each iteration, which makes computation complexity for each iteration $O(M)$, where M is the number of features (factors). The update rate for SGD updates are much faster than batch gradient descent when D (the number of samples) is high.

- **Nesterov Accelerated Gradient Descent:** Though SGD has been popular, it does have a learning process that can be time-consuming. There are three possible ways to adjust the learning rate - slowing, ramping up, and speeding it up. These could result in an improved speed of convergence, as well as prevent us from being trapped at a local minimum during the search. A lot of research has gone into improving SGD, but there is still room for improvement. For example, the momentum idea was proposed to be applied in SGD [70]. The momentum method can speed up the convergence of a gradient when dealing with high curvature or noisy gradients. The concept of momentum is derived from the mechanics of physics, which simulates the inertia of objects. The speed is set as the average exponential decay of the negative gradient.

  Nesterov momentum, also known as Nestorov accelerated gradient descent, is a first-order optimization technique for convex functions that can speed up convergence when the learning rate is large. The idea behind Nesterov momentum is to use a momentum term to smooth out the updates, which can help the algorithm escape from local minima. The momentum term also has the effect of increasing the learning rate when the gradient is small and decreasing it when the gradient is large. This can help the algorithm converge more quickly to a global minimum.

  - The classical momentum velocity vector is defined by Bengio as:

$$v_t = \mu_{t-1}v_{t-1} - \epsilon_{t-1}\nabla f(\theta_{t-}) \tag{2.10}$$

  - Sutskever defines classical momentum in the same way but without any t sub-

scripts on the velocity vector or the momentum coefficient.

$$\theta_{t+1} = \theta_t + \mu_t v_t - \epsilon_t \nabla f(\theta_t) \tag{2.11}$$

with velocity

$$v_{t+1} = \mu_t v_t - \epsilon_t \nabla f(\theta_t) \tag{2.12}$$

- Further on, the following definition of the term "momentum" will be used: It is a continuous quantity equal to the product of mass and velocity, and which is responsible for the motion or change in motion of objects. For example, it keeps us from falling back to Earth when jumping in the air. But here, that's not the full update so we can't use v_t. Later on, when we have the full update, I'll write it as m_t instead. That way there won't be any confusion.

$$m_{t+1} = \mu_t m_t - \epsilon_t \nabla f(\theta_t) \tag{2.13}$$

Remember: $v_{t+1} = m_{t+1}$ only applies in classical momentum.

- **Adaptive Learning Rate Method:** The manually regulated learning rate is one of the most important aspects of the SGD method for deep learning. This is a tricky issue when it comes to choosing an appropriate value for the LR. Some adaptive methods are designed to adjust the LR automatically, which is free from parameter adjustment, converges very quickly, and often provides good results. These methods are widely used in deep neural networks that solve optimization problems.

  AdaGrad is an update to the standard grad learning rate model. AdaGrad adjusts the learning rate based on a previous iteration's gradient and generalizes the SGD to use different learning rates in different parts of the loss surface. The update formulae are:

$$\begin{aligned} g_t &= \frac{\partial L(\theta_t)}{\partial \theta} \\ V_t &= \sqrt{(\sum_{i=1}^{t} (g_i)^2 + \epsilon} \\ \theta_{t+1} &= \theta_t - \eta \frac{g_t}{V_t} \end{aligned} \tag{2.14}$$

  AdaGrad and gradient descent are algorithms that both attempt to find the best pa-

rameters for a linear learner. AdaGrad has two main advantages over gradient descent: first, it can eliminate the need for training by just figuring out what to change, and second, its learning rate is improved. Users typically increase the default learning rate from 0.01 for η. AdaGrad was improved to AdaDelta and RMSProp in order to resolve the issue of the learning rate eventually reaching zero.

2.5.2 B. Higher-Order Method

Second-order optimization methods can be used to address the problem of highly-nonlinear and ill-conditioned objective functions. This is accomplished by introducing curvature information.

This section presents higher-order methods to handle large-sclae data by using stochastic techniques. Although the convergence of the algorithm can be guaranteed, the computational process is costly and thus rarely used for solving large machine-learning problems. The presented high-order algorithms enable low-cost methods to process large-scale data in a variety of ways.

- **1) Conjugate Gradient Method:** The conjugate gradient approach is a fascinating optimization method, which has been proven to be one of the most effective methods for solving large-scale linear systems of equations. It can also be used for solving nonlinear optimization problems [65]. Conjugate gradient optimization is an intermediate algorithm, which can only utilize the first-order information for some problems but ensures the convergence speed like high-order methods.

  The first method for nonlinear optimization came about in the 1960s, called the conjugate gradient. This method is sometimes used to solve systems of equations. In 1964, it was expanded to handle nonlinear optimization as well. For years, many different algorithms based on the new method have been proposed and research has shown that they are more efficient than the steepest descent. The details of these algorithms are described below.

  Let us consider a linear system,

$$A\theta = b \tag{2.15}$$

$$min_{\theta}F(\theta) = \frac{1}{2}\theta^{T}A\theta - b\theta + c \tag{2.16}$$

In equation (2.15) above, A is an $n \times n$ symmetric positive definite matrix, b is a vector, and it is to be solved for . It can be considered an optimization problem that minimizes the quadratic function (2.16) with a positive definite objective coefficient. The two equations shown above have the same unique, satisfying solution. The conjugate gradients can be considered a method for solving optimization problems.

We will solve $A\theta = b$ for θ iteratively. Define residual $r(\theta) := \nabla f(\theta) = A\theta - b$. Then at iterate θ_k, we have

$$r_k = A\theta_k - b \tag{2.17}$$

The one-dimensional minimizer of f along θ_k is given explicitly as follows.

$$\eta_k = -\frac{\nabla f(\theta_k)^T d_k}{d_k^T A d_k} = -\frac{r_k^T d_k}{d_k^T A d_k} \tag{2.18}$$

The conjugate gradient method uses the so-called "direction set" $d_0, d_1, \ldots d_n$ to find a stationary point in A. The direction set is chosen such that its elements are conjugates with respect to A, which means

$$d_k^T A d_k = 0 \forall i \neq j. \tag{2.19}$$

The first step is to choose the new direction d_k by taking the steepest descent direction of f and adding it to the previous direction d_k. That's why we write:

$$d_k = r_k + \beta_k d_{k-1} \tag{2.20}$$

Assuming that d_{k-1} and d_k are conjugate with respect to A and β_k is to be determined by a requirement that they be, we pre-multiply the last equation by

$$d_k^T A \tag{2.21}$$

imposing the condition (2.19) follows

$$\beta k = \frac{r_k^T A d_{k-1}}{d_{k-1}^T A d_{k-1}} \tag{2.22}$$

If get to know η_k and d_k, the next iterate can be calculated as usual using

$$\theta_{k+1} = \theta_k + \eta_k d_k \tag{2.23}$$

The CG method has a grace property. It can generate a new vector d_k only with the previous input vector d_{k-1}. There's no need to know all the previous vectors from d_0 up to d_{k-2}.

- **2) Quasi-Newton Methods:** The convergence rate for gradient descent is slow. To speed up this process, second-order information can be used, like Newton's method. Newton's method uses both the first-order derivative and the second-order derivative to approximate the objective function with a quadratic function. It solves the minimal optimization of the quadratic function until it converges.

$$\theta_{t+1} = \theta_t - \frac{f'(\theta_t)}{f"(\theta_t)}, \tag{2.24}$$

In equation (2.24) Newton's iteration formulas are demonstrated for one-dimensional optimization. In the equation, f is the objective function. More generally, the high-dimensional Newton's iteration formula can be written as,

$$\theta_{t+1} = \theta_t - \frac{\nabla f(\theta_t)}{\nabla^2 f(\theta_t)}, t \geq 0, \tag{2.25}$$

above $\nabla^2 f$ is a matrix of the Hessian of f. Basically, if the step size factor is introduced, the iteration formula appears as

$$\begin{aligned} d_t &= -\frac{\nabla f(\theta_t)}{\nabla^2 f(\theta_t)}, \\ \theta_{k+1} &= \theta_k + \eta_t d_t \end{aligned} \tag{2.26}$$

where d_t is Newton's direction, η_t is the step size.This method can be called damping Newton's method [37]. Newton's method can be called a damping Newton's method and is defined by the following equation. Geometrically speaking, Newton's method is to fit the current position surface with a quadratic surface, while the gradient descent method is to fit it with a plane.

Newton's method is iterative. the algorithm that requires the computation of the inverse Hessian. matrix of the objective function at each step, which. makes storage

and computation very expensive. To overcome this, an approximate algorithm was considered which is called the quasi-Newton method. This method uses a positive matrix to approximate the inverse of the Hessian, thus simplifying the complexity of the operation.

2.5.3 C. Derivative-Free Optimization

Sometimes, find the optimal point for optimization problems in practical applications where the derivative of the objective function does not always exist or is too complicated to calculate. The solution for finding this optimal point without the use of a gradient is called derivative-free optimization (DFOP), which is a well-known method for mathematical optimization.

There are two methods of derivative-free optimization. One type is to use heuristic algorithms. These methods work empirically and choose techniques that have worked well in the past, rather than deriving a solution systematically. There are many different types of heuristic optimization techniques, including classical simulated annealing arithmetic, genetic algorithms, ant colony algorithms, and particle swarm optimization. Generally, these heuristic methods yield an approximate global optimal value and theoretical support is weak. These techniques will not be discussed further in this thesis. The other option is to set an appropriate function based on the samples of the objective function. This type of method usually restricts the search space by adding constraints before sampling. The coordinate descent method is a typical generative-free algorithm [**bertsekas1999nonlinear**] and it can also be easily extended to become an optimization algorithm for machine learning problems. This thesis only discusses the coordinate descent method.

The coordinate descent is a derivative-free optimization method for multi-variable functions. The algorithm's idea is that one can perform sequential searches along each axis to obtain updated values for each dimension. This method is a good choice for some problems where the loss function is non-differentiable.

The vanilla approach is to use linear search steps in the orthogonal space to find the optimum of a function. For our objective function, $L(\Theta)$, when Θ^t is already known, we solve

the j^{th} dimension of Θ^{t+1} by [23]

$$\theta_j^{t+1} = argmin_{\theta_j \in R} L(\theta_1^{t+1}, ...\theta_{j-1}^{t+1}, \theta_j, \theta_{j+1}^t, ..., \theta_D^t). \tag{2.27}$$

As a result, the following inequality is guaranteed: $L(\Theta^{t+1}) \leq L(\Theta^t) \leq L(\Theta^0)$. As it converges, which is similar to the descent gradient method.

The coordinate descent method is similar to the gradient descent, except that it uses a different coordinate system and does not need to calculate the gradient of the objective function. The algorithm still has limitations when performing on the non-smooth objective function, which may fall into a non-stationary point. An appropriate coordinate system can be used to accelerate the convergence of the two algorithms.

2.6 Cross-validation

Quantifying the statistical relationships between variables within a set of data is a process known as validation. After training, the error estimate for the model should be evaluated. This evaluation is accomplished by comparing the expected responses and original responses using a numerical difference called residuals. Generally, this is done by using data used to train the model - but this only gives us an idea of how well the model performs with data we've seen before. To evaluate how well the model will generalize to an independent dataset, a technique called validation can be used. There are many cross-validation approaches used in machine and deep learning modeling a selection is explained in the following,

The holdout method for cross-validation is the simplest one. The data set is split into two separate sets called the training set and the testing set. The functions approximator fits a function using the data from only the training set. Then when it's asked to predict the output values of a new data point, it has never seen these outputs before, so it needs to predict them by "guessing" based on what it saw in the training set. These guesses are used to track its predictions of these values and eventually give us an average error rate which we use to evaluate how accurate our model is. One advantage of this method over residual methods (less accurate) is that it takes no more time to compute, but there's still a high degree of variance in how accurate this model will be evaluated because splitting up

this function can affect how well we see predictions referring to individual data points that were used for training and testing sets.

- **K-Fold Cross Validation:**

  K-fold cross-validation is a method for estimating the performance of a machine-learning algorithm on unseen data. In this 10-fold cross-validation process, the data is divided into ten subsets. The first holdout set is used as the test set and the other nine subsets form the training sets. The error estimation will be averaged over all ten trials to give us an idea of how well our algorithm performs. In K-fold cross-validation, every observation gets one strata in each role, so measurement bias is reduced and both variance and covariance are controlled.

- **Stratified K-Fold Cross Validation:**

  It is possible that there may be an imbalance in the response variables which occurs in datasets with a large number of high-value houses. Alternately, classification problems can have many more negative samples than positive samples. This variation of the K Fold may help to alter the distribution so that each fold contains a more even distribution of target classes, or an approximate equal mean prediction value, depending on what type of problem it is.

- **Leave-one-out cross validation:**

  The leave-one-out cross validation (LOO-CV) is a popular cross validation strategy. The idea behind LOO-CV is to use a single data point as the validation set and all other data points as the training set. This process is repeated for each data point, resulting in N models being fit, where N is the number of data points. The advantage of LOO-CV is that it maximizes the amount of data used to train each model and also provides an estimate of how well the model would generalize to new data. The downside of LOO-CV is that it can be computationally expensive, especially when working with large datasets.

Cross-validation is an important technique for assessing the accuracy of a model, especially when it is about avoiding overfitting and finding the parameters that lead to the lowest error rates.

3 State of the Art

3.1 Single Task Learning

In the past decade, machine learning and deep learning have been widely used in various fields such as computer vision, natural language processing, and recommended systems. However, when it comes to single-task learning, there is already a comprehensive survey of papers presented that systematically summarizes the state-of-the-art methods. In this section, we provide an overview of the current state-of-the-art methods for single-task learning from both machine learning and deep learning perspectives. We first cover the basic concepts and terminologies of single-task learning. Then, we review the key methods for tackling the regression problem, including feature engineering, model selection, and regularization. Finally, we discuss the recent advances in deep learning for single-task learning. This survey provides an overview of the current state-of-the-art methods for single-task learning and is expected to be helpful for readers who are new to this field.

3.2 Terminology of Regression Analysis

In Single Task Learning (STL), a regression problem is when trying to predict a continuous value. For example, it might be trying to predict the price of a house based on its size, age, and location. A linear regression algorithm would find the line of best fit for all the data points and then use that line to make predictions.

Regression problems are one of the most common types of machine learning problems and they're also one of the easiest to understand. In this article, we'll go over some of the basic concepts and terminology surrounding regression problems so that anyone can jump-start their own machine-learning projects.

First, let's talk about what we're trying to predict - the target variable. The target variable is the value that we're trying to predict and it can be anything from a simple yes/no classification to a more complex real number. In our house price example, the target variable would be the price of the house.

Next, we have our features or explanatory variables. These are the different pieces of

information that we're using to make our predictions. In our house price example, size, age, and location would be features that we use to try and predict the price.

Finally, we have our training data. This is a dataset that we use to train our machine-learning algorithms on. It contains both feature values and target values so that the algorithm can learn how to map the features to targets.

3.3 Machine Learning for Regression

Regression models have become more and more popular. They're used in many different fields, including financial forecasting and prediction, cost estimation, trend analysis, marketing, time series estimation, drug response modeling, and more. There are many types of commonly known regression algorithms: linear regression, polynomial regression, lasso regression, and ridge regression. These are only briefly explained here.

3.3.1 Simple and multiple linear regression:

This is one of the most simple regression techniques, as well as one of the most popular machine-learning modeling techniques. The dependent variable can be continuous or discrete, and the independent variable(s) are continuous or discrete. Both approaches create a linear form of the regression line. Linear regression creates a relationship between the dependent variable (Y) and one or more independent variables (X) (also known as a regression line) using the best fit straight line[69]. written by the following equations:

$$y = a + bx + \epsilon \tag{3.1}$$

$$y = a + b_1x_1 + b_2x_2 + ... + b_nx_n + \epsilon \tag{3.2}$$

where a is the intercept (predicted y-axis value when x = 0), b is the slope of the line (predicted change in y for every unit change in x), and e is the error term (difference between predicted and actual values of y). This equation can be used to predict the value of the target variable based on an inputted predictor variable. Multiple linear regression defined in Equ (3.2) lets us use two or more independent variables to predict the response variable. In comparison to simple linear regression, more than one independent variable can be

investigated. This is defined in Equation(3.1).

3.3.2 Polynomial regression:

Polynomial regression is a type of regression analysis in which the relationship between x and variable y is not linear but polynomial, poly(degree n^{th} in x). The equation for polynomial regression is derived from the linear regression equation.

$$y = b_0 + b_1 x_1 + b_2 x^2 + +b_3 x^3 ..... + b_n x^n + \epsilon \tag{3.3}$$

Here n^{th} degree of polynomial regression is used when we don't have linear data, with input variables $b_0, b_1, ...b_n$ and output y. We use the regression coefficients to get our predicted/target output, y. In simpler terms, we can say that if our data isn't distributed linearly (it's a n^{th} degree polynomial) then we'll use degrees of polynomial regression to predict what our target output should be.

3.3.3 LASSO and ridge regression:

LASSO and Ridge regression are purely model-based learning techniques that can be applied in the presence of a large number of features, thanks to the capability to prevent over-fitting by reducing the model complexity. LASSO regression uses the $L1$ regularization technique, which is based on shrinking: it penalizes any excessive value of the magnitude of coefficients ($L1$ penalty). In this way, LASSO appears to find coefficients that bring them as close as possible to absolute zero. Lasso regression points at the subset of predictors that minimizes prediction error for a quantitative response variable. On the other hand, ridge regression uses L2 regularization [68]. Ridge regression ($L2$ penalty) forces the weights to be small but never sets the coefficient value to zero. This is different from LASSO regression, which attempts to eliminate less important features. Overall, the data set will have fewer non-zero coefficients with LASSO than ridge regression, which works better in data sets that are correlated because of multicollinearity.

3.3.4 Dimensionality Reduction and Feature Learning

In Single task machine learning, high-dimensional data processing is a challenging task for researchers and app developers. To simplify this process, there are two main techniques: dimensionality reduction and feature selection. Dimensionality reduction is an unsupervised learning technique that's important because it leads to more human-interpretable models, lower computational costs, and model simplification. Both the process of feature selection and feature extraction can be used to manipulate data in order to produce dimensionality reduction. The primary distinction between the selection and extraction of features is that the "feature selection" keeps a subset of the original features [75], while "feature extraction" creates brand new ones[73].

- **Feature selection:** The process of selecting features (also known as variables or predictors) is the process of choosing a subset of unique features to use in data modeling and building machine learning algorithms. This will usually entail removing less important or irrelevant features and allow for faster training. By simplifying and generalizing the model as well as increasing its accuracy, the right and optimal subset of selected features in a problem domain can minimize the over-fitting problem [75]. Feature selection is an important part of creating a well-performing machine learning model. Various tests, including the Chi-Squared Test, Analysis of Variance (ANOVA) Test, and Pearson's correlation coefficient can be used to select features.
- **Feature extraction:** Machine learning-based models and systems often use feature extraction techniques to better comprehend the data and improve predictive accuracy. They also reduce the computational burden, which significantly speeds up training time. When there are a lot of features in a dataset, we can reduce the number by creating new ones using existing data and then discarding the original ones. The best thing about this is that most of the information found in the first set can be summarized using these new reduced sets. For instance, principal components analysis (PCA) is often used as a dimensionality-reduction technique to extract a lower-dimensional space creating new brand components from the existing features in a dataset [73].

Many algorithms have been proposed to reduce data dimensions in the machine learning and data science literature ([69], [83]. some of the popular methods we summarized here which used widely in various application areas.

- **Variance threshold:** A simple basic approach to feature selection is the variance threshold [68]. This excludes all features of low variance, like any that don't have a big enough range to be helpful. It also eliminates all the zero-variance characteristics by default, like those with the same value in every sample. This feature selection algorithm unsupervised learning only looks at the (X) features and doesn't require (y) outputs and can be used for many areas of analysis.
- **Pearson correlation:** Pearson's correlation method is another way to measure how a feature relates to the response variable and can be used for feature selection [74]. It is also used to find the association between two variables in a dataset. The resulting value is $[-1,1]$, where -1means perfect negative correlation, $+1$ means perfect positive correlation, and 0 means there is no linear correlation. If two random variables represent X and Y, then the correlation coefficient between X and Y is defined as [69]

$$r(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \tilde{X})(Y_i - \tilde{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \tilde{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \tilde{Y})^2}} \tag{3.4}$$

- **ANOVA:** Analysis of variance (ANOVA) tests the equality of mean values between two or more groups that are significantly different from one another. In order for it to work, the ANOVA assumes a linear relationship between the variables and the target, as well as normal distribution for all variables. In fact, F tests are utilized in order to statistically test the equality of means. For features that don't affect the result, use 'ANOVA F values' [68] for each level of each factor to perform a validity test.
- **Principal component analysis (PCA):** Principal component analysis (PCA) is used in the field of data science and is a powerful method for analyzing correlative variables without having to make any assumptions about their inter-relationships. PCA takes correlated variables and transforms them into uncorrelated variables known as principal components. Figure 1 shows an example of the effect of PCA on features in a 3D space. Figure 1a displays the original features, while Figure 1b displays the created principal components, PC1 and PC2, onto a 2D plane. Thus, PCA can be used as a feature extraction technique that reduces the dimensionality of the datasets and builds an effective machine learning model [73]. PCA works by identifying the components of your data with the highest eigenvalues. These are then projected into a new subspace of equal or fewer dimensions.

3.4 Deep Learning for Regression

Deep learning for regression can be used for a variety of tasks, such as predicting the price of a stock based on historical data or estimating the age of a person based on their photo. Generally speaking, deep learning networks are well-suited for problems where there is an abundance of data and where traditional methods (e.g., linear regression) struggle.

There are many different types of deep learning networks, each with its own strengths and weaknesses. The most popular types are general-purpose networks such as ANNs (artificial neural networks) and Convolutional Neural Networks (CNNs), which are often used for image processing and vision problems. More specialized networks include Recurrent Neural Networks (RNNs), which are good at modeling time series data, and Generative Adversarial Networks (GANs), which can generate new data samples from scratch.

The choice of network architecture will depend on the specific task at hand. However, in general, deeper networks (with more layers) tend to perform better than shallower ones. This is because deeper networks can learn more complex relationships between the input data and the output variable. Deep learning has allowed incredible advances in text and image datasets, but its superiority on tabular data (like training information) is still in question. in this section, we have done some review of deep learning on the tabular data architecture and also discussed views about the superiority of the neural network method over the tree-based model. This section aims to understand a series of challenges that need to be addressed in single-task deep learning problems: 1) the lack of informative features causes an overfitting problem, 2) the lack of orthogonal planes prevents the network from adjusting to different orientations of the data, and 3) the difficulty involved with learning irregular functions. To help push research in this area,

3.4.1 Artificial neural networks(ANN)

An artificial neural network (ANN) is a computational model based on the workings of the biological nervous system. These models are used to simulate the learning and decision-making processes of humans. Neural networks are composed of interconnected units, or nodes, that exchange information with each other. ANNs are capable of learning to recognize patterns of input, which means they can be used for tasks such as image recognition

or voice recognition. They can also be used to make predictions, based on data they have been trained on. For example, an ANN might be able to predict the likelihood of someone defaulting on a loan, based on their financial history.

The nodes in an ANN are connected together with weights. These weights determine how much influence each node has on the output of the network. The bias is another important parameter in an ANN; it provides more flexibility in the network model. Bias can be used to shift the result of the activation function in a positive or negative direction. The non-linear function is what allows the network to learn complex relationships between inputs and outputs.

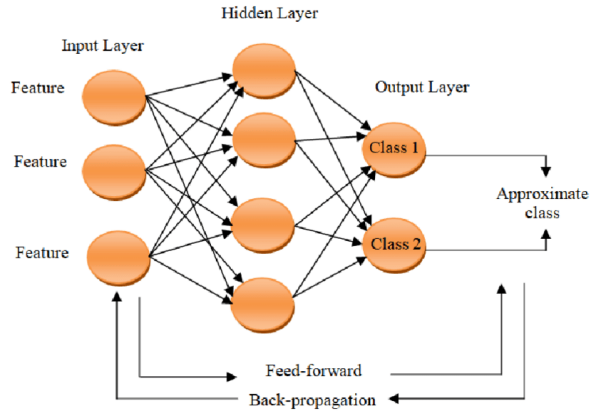

Figure 3.1: Artificial neural networks(ANN)[31]

Neurons take in inputs and then apply a non-linear activation function to calculate their weight. The result is the output, which is then represented as a vector dot product.

$$z = f(x.w) = f(\sum_{i=1}^{n} x_i w_i) x \in d_{1\times n}, w \in d_{1\times n}, z \in d_{1\times 1} \tag{3.5}$$

Equation (3,5) left out the bias term to make it easier to understand. Bias is a multiplier input to all the nodes and always has the value of 1. It allows you to shift the result of the activation function either way. It also helps the model train when all of the input features are 0. For completion's sake, the above equation looks like this with bias included.

$$z = f(b + x.w) = f(b + \sum_{i=1}^{n} x_i w_i) b \in d_{1\times 1} \tag{3.6}$$

In common training approaches for neural networks, the stochastic gradient descent algorithm is typically used (Bottou, 2012; LeCun, Bottou, Orr, Muller, 1998a) or a variant of it. The SGD algorithm comes with one major advantage: it's an optimization algorithm. When adjusting an individual variable θ in the function f parameterized by θ and minimizing the loss of f with respect to the training examples, SGD is one of the most common algorithms. It works as follows:

Algorithm 1 [15] Online Stochastic Gradient Descent Training

1: **Input**: Function $f(\mathbf{x};\theta)$ parameterized with parameters θ.

2: **Input**: Training set of inputs $\mathbf{x_1},\ldots,\mathbf{x_n}$ and outputs $\mathbf{y_1},\ldots,\mathbf{y_n}$.

3: **Input**: Loss function L.

4: **while** stopping criteria not met **do**

5: Sample a training example $\mathbf{x_i},\mathbf{y_i}$

6: Compute the loss $L\left(f\left(\mathbf{x_i};\theta\right),\mathbf{y_i}\right)$

7: $\hat{\mathbf{g}} \leftarrow$ gradients of $L\left(f\left(\mathbf{x_i};\theta\right),\mathbf{y_i}\right)$ w.r.t θ

8: $\theta \leftarrow \theta + \eta_k \hat{\mathbf{g}}$

9: return θ

The goal of the algorithm is to find the parameters θ which minimize the total loss function $\sum_{i=1}^{n} L\left(f\left(\mathbf{x_i};\theta\right),\mathbf{y_i}\right)$ over the training set. To do this, it repeatedly samples a training example and computes the gradient of the error on that example with respect to its parameters θ (line 7). The input and desired output are assumed to be fixed, and the loss is treated as a function of these parameters . The parameters θ are then updated in the direction of this gradient, scaled by a learning rate η_k (line 8).

3.4.2 Convolutional Neural Network (CNN)

This network structure was first proposed by Fukushima in 1988 [29]. As a result of the limitations of computation hardware, it was not widely used. A gradient-based learning algorithm was applied to CNNs in the 1990s by LeCun et al. [50] For the handwritten digit

classification problem. In the early 2000s, neural network methods for computer vision won remarkable recognition and research interest increased. One of these is the convolutional neural network (CNN), which proved to be more efficient than the deep neural networks (DNN) in most computer vision tasks. CNNs are easier to optimize than DNNs, especially for 2D and 3D image processing. Additionally, they can efficiently identify abstractions of 2D features. Furthermore, max-pooling layers significantly reduce variation in shape by absorbing images with identical or similar shapes - a precise function compared with DNNs. Most importantly, CNNs work on gradient-based learning algorithms and possess parameters that are not based on diminishing gradients. This is for both efficiency and because those algorithms train networks on various image tasks directly until they hit an error criterion.

In addition to classification, CNNs are employed to address regression-related issues. In this scenario, a fully connected regression layer with linear or sigmoid activations is frequently used in place of the softmax layer. Such systems are referred recognized as "vanilla deep regression." Numerous deep models were presented in line with this methodology, yielding cutting-edge solutions to traditional vision regression issues like head position estimation [54], human pose estimation [79], [9], or facial landmark recognition [78]. Vanilla deep regression is often used as a building block in cascaded approaches, where such regression methods are iteratively refined. we review only a detailed discussion of the deep learning model used for tabular data.

- **TabNet** Tabnet [6] is one of the first transformer-based models designed to process tabular data. Like a decision tree, TabNet comprises multiple subnetworks that are processed in a sequential hierarchy. According to [6] train TabNet, each subnetwork receives the current data batch as input. TabNet aggregates the outputs of all subnetworks to obtain the final prediction. The authors claim that TabNet can save valuable resources by focusing on the most important features at each decision step after applying a sparse feature mask [63]. TabNet is one of the few deep neural networks that offers different levels of interpretability by design. Each decision step tends to focus on a particular subdomain of the learning problem (i.e., one particular subset of features) This behaviour is similar to convolutional neural networks. TabNet can be used in a two-stage self-supervised learning procedure, which improves the overall predictive quality.

- **ARM-Net** ARM-Net [17] is an adaptive neural network that has been tailored to work with tabular data. The key idea of the ARM-net framework is modeling feature interactions by combining features. Adaptive, dynamic interaction weights result from the input features being transformed into exponential space and calculated according to their interaction order and weights. A novel sparse attention mechanism has been developed to generate the interaction weights given input data dynamically, and users can model cross interactions of arbitrary orders with filtered noisy features.
- **Res-Net** ResNet [38] is a popular architecture developed for ultra-deep networks that try to avoid the vanishing gradient problem. ResNet consists of many different numbers of layers; 34, 50,101, 152 or even 1202. The popular ResNet50 has 49 convolution layers and only 1 fully connected layer at the end with the total number of weights and MACs being 25.5M and 3.9M respectively.

  Figure 3.2 introduces the basic ResNet architecture. It is a traditional feedforward network with residual connections. The outputs of residual layers depend on operations that have been performed on output ($f(x_{l-1})$) from the previous layer, x_{l-1} (e.g., convolution with different sizes of filters, followed by Batch Normalization plus an activation function, such as ReLU). The final residual unit output after performing various operations, x_l can be defined with this equation:

$$x_l = f(x_{l-1}) + x_{l-1} \tag{3.7}$$

  As a result of the variable architecture of residual networks [38], the operations within the residual blocks can vary. Zagoruvko et al. [87]proposed a wider version of the residual network.

3.4.3 Recurrent neural network (RNN)

The recurrent neural network (RNN) is a type of neural network that analyzes sequential, time series, and continuous data, including text, speech, videos, and other continuous data. In contrast to artificial neural networks and convolutional neural networks, RNNs do not only feed-forward. They evaluate portions of an input and compare them to portions before and after it. As a result, the recurrent neural network uses weighted memory and

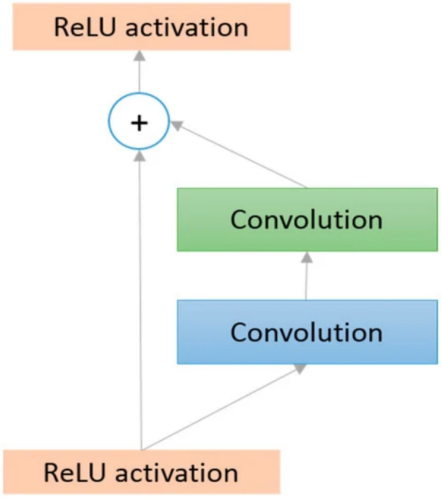

Figure 3.2: Basic diagram of the Residual block[3]

feedback loops to accomplish this.

In RNNs, we obtain an output after each of our inputs is evaluated on a single layer. In a feedback loop, a recurrent neural network evaluates the sequential features of the input and returns the output to the evaluation step. Backpropagation Through Time (BPTT) creates a way for the network to refer back to its previous features in the context of the current feature. The main problem with RNN approaches is that there exists the vanishing gradient problem. Several solutions have been proposed to solve this problem in recent decades. One of the better models was introduced by Felix A. et al. in 2000 named Long Short-Term Memory (LSTM). Hochreiter et al [41]. solve the problem for the first time.

- **Long Short-Term Memory (LSTM)** An LSTM's key concept is the cell state, represented by the horizontal line at the top of Figure 3.3. Gates remove or add information to the cell state: An input gate (i_t), forget gate (f_t), and output gate (o_t) can be defined as:

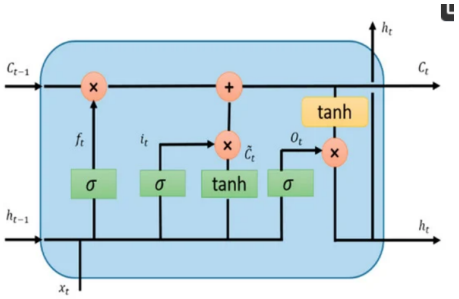

Figure 3.3: Long Short-Term Memory (LSTM)[3]

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}], x_t] + b_f), \\
i_t &= \sigma(W_i \cdot [h_{t-1}], x_t] + b_i), \\
\tilde{C}_t &= tanh(W_c \cdot [h_{c-1}], x_t] + b_C), \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t, \\
O_t &= \sigma(W_o \cdot [h_{t-1}], x_t] + b_o), \\
h_t &= O_t * tanh(C_t).
\end{aligned}
\tag{3.8}
$$

Many papers use LSTM models for temporal information processing. These models have a slight variance in some cases. In 2000 [30], Gers and Schimidhuber proposed a slightly modified version of the network with peephole connections. In this model, peepholes are included with almost all gates.

- **Gated Recurrent Unit (GRU)** Recurrent neural networks (RNNs) with long short-term memory (LSTMs) have been popular in the community of developers who work with recurrent networks. Until recently, GRUs were not popular among these developers. Now, however, they are becoming more popular; the main reason for this is that the cost of computation and the simplicity (which is shown in Figure 3.4) of GRUs is appealing to those who work with RNNs.
  In contrast to LSTMs, GRUs combine input and forget gates into a single update gate and merge cell state and hidden state together. They also contain some other changes that simplify their design when compared to RNNs. Even with these changes, GRU

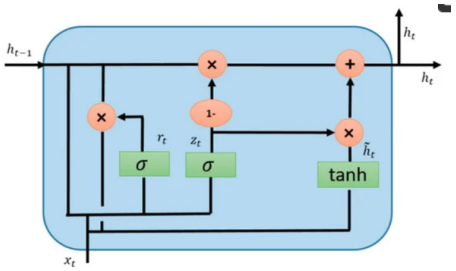

Figure 3.4: Gated Recurrent Unit (GRU)[3]

mathematically can be expressed using equations like this:

$$\begin{aligned} z_t &= \sigma(W_z \cdot [h_{t-1}], x_t]), \\ r_t &= \sigma(W_r \cdot [h_{t-1}], x_t]), \\ \tilde{h_t} &= tanh(W \cdot [r_t * [h_{c-1}], x_t]), \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h_t} \end{aligned} \tag{3.9}$$

Given the different empirical studies, there is no clear winner. However, the GRU only requires a few parameters, which makes it faster than the LSTM model. On the other hand, LSTM can provide better performance when you have enough data and computational power. There is also a variant called Deep LSTM. Another variant that's a bit different is called A Clockwork RNN[48]. In addition to empirically evaluating some of these new approaches, such as LSTM by Greff et al. in 2015, and concluding they were all about the same [34]. Another evaluation was conducted over thousands of RNN architectures, including LSTMs, GRUs, and so on, which found some that performed better than LSTMs for certain tasks [46].

3.4.4 Generative Adversarial Networks (GAN)

The concept of generative models in machine learning started a long time before they were used to create data models built from conditional probability density functions. In general, this kind of model is considered a probabilistic model with a joint probability distribution over observations and targets (labels). However, we didn't see much success for this generative model. Recently, deep learning-based generative models have become popular and shown great success in different applications.

Deep learning is a data-driven technique that performs better as input data becomes more plentiful. That's why deep learning has been used to generate similar samples from huge, un-labeled datasets with generative models. Computer vision has different tasks, like segmentation and classification. These can require large amounts of labeled datasets, which is an issue deep learning solves by generating new similar samples.

Generative Adversarial Networks are a recent invention by Goodfellow in 2014 that has seemed to create waves in the computer science world. As an unsupervised deep-learning process, two neural networks compete against each other in a zero-sum game. In the case of image processing problems, the generator starts with Gaussian noise to generate images and the discriminator determines how good these generated images are. This process continues until outputs from the generator become close enough to actual input samples. As shown in Figure 3.5, Discriminator (D) and Generator (G) are two players playing the min-max game with the function of $V(D, G)$, which can be expressed as follows in the paper ([53], [60]).

$$min_G max_D V(D,G) = E_{x-p_{data}(x)}[log(D(x))] + E_{z-p_{data}(z)}[log(1 - D(G(z)))]. \tag{3.10}$$

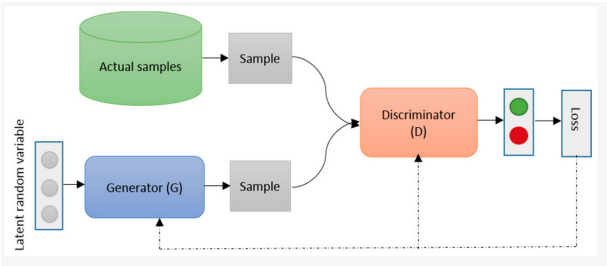

Figure 3.5: Generative Adversarial Networks[3]

While the above equation (3.10) initially provides a good gradient for learning G, it stops producing strong gradients very early into the training process. To get around this, train G to maximize the $log(G)$ objective function instead of minimizing $log(1 - D(G))$. While this means that parameters in G need to be carefully tuned, this should be easier than solving the optimization problem for training with D for every new dataset. However, there are some limitations and theoretical problems with optimizing D:

- There is no heuristic cost function (pixel-wise approximate means square errors (MSEs))[3]

- Training unstable (sometimes that can be because of producing nonsensical outputs) [3]

GANs have been used for various tasks such as image generation, text-to-image synthesis, style transfer, and more recently, tabular data generation. For example, GANs can be used to generate new samples of data that are similar to those in a training dataset (but not identical), which can be useful for fraud detection or other anomaly detection tasks. Tabular data GANs can also be used to fill in missing values in data sets or to generate synthetic datasets for use in machine learning models. Overall, GANs are a powerful tool for generative modeling and have many potential applications.

CNN, RNN, and GAN are popular approaches to single-task learning that can be used for a variety of tasks, including sequential and time series modeling. They are also popular approaches to multi-task learning, as they can be used to learn multiple tasks simultaneously. CNN and RNN are popular choices for both single-task and multi-task learning, as well as their benefits and drawbacks.

3.5 Multi-task learning

Machine Learning (ML) is a type of artificial intelligence that gives our models the ability to generate knowledge with experience. Generally, ML deals with optimizing for a particular metric. For example, we might want to identify bad web pages and flag them as "spam." In order to do this, ML might train a model and then tweak it until it achieves acceptable results. However, by focusing on only the task at hand, we ignore clues that might improve our performance on the metric we care about. To avoid this problem, Multi-Task Learning (MTL) can be used which shares task representations between related tasks. This approach allows ML algorithms to generalize better on our original goal.

Multi-task learning has been used successfully to improve machine learning algorithms in nearly every application, such as improving natural language processing [22], speech recognition [25], computer vision [32], and even drug discovery [71]. MTL can be defined as the optimization of more than one loss function - so anytime improving more than one metric for the same task, there can be benefits gained from MTL.It's helpful to reflect on what is done explicitly in MTL and draw insights from it to better understand this technique's

well-known applications.

As part of machine learning, multi-task learning allows algorithms to learn simultaneously from multiple tasks. This can be done using a variety of different methods, including supervised, unsupervised, semi-supervised, active, and reinforcement learning. Each method has its own advantages and disadvantages, so it is important to choose the right method for the particular application..

Supervised multi-task learning is typically used when there is a large amount of labeled data available for each task. This allows the algorithm to learn from the errors made on previous tasks and improve its performance on future tasks. Unsupervised multi-task learning is more suitable when there is only a small amount of labeled data available. In this case, the algorithm must learn from the raw data itself without any guidance from labels.

Semi-supervised multi-task learning combines both supervised and unsupervised learning methods. This approach can be useful when there are some labeled data available but not enough to train a supervised model. Active multi-task learning focuses on selecting which tasks to learn from at each iteration. This can be helpful when there are many tasks but only a limited amount of time or resources available.

Reinforcement multi-task learning is typically used in settings where an agent interacts with an environment to learn how to complete a task. In this case, the agent receives feedback after each interaction with the environment and uses this feedback to adapt its behavior and improve its performance on future tasks. In the following section, multi-task supervised learning is explained in more detail.

3.5.1 Multi-task supervised learning

In Multi-task Supervised Learning (MTSL) each supervised task independently models the function that maps data instances to labels. Mathematically, for each task $T_i(i = 1, ..., m)$, there is a training dataset $D_i = (x_j^i, y_j^i)_{j=1}^n$ where x_j^i is an instance in d-dimensional space and y_j^i is the corresponding label. Depending on the type of label, tasks can be either regression or classification. When y_j^i is a real scalar, it's regression while discrete values like -1 and 1 point towards classification. For the m tasks in the training set, MTSL aims to learn $f_i(x)_{i=1}^m$ as good approximations to y_j^i such that $f_i(x_j^i)$ is a good approximation to y_j^i for all i and j.

The way in which task-relatedness is assumed in the particular application influences the design of the MTSL models. Three existing kinds of these models can be distinguished accordingly: feature-based, parameter-based, and instance-based MTSL models. Feature-based assumes features of all tasks are alike or similar, which can be either a subset or transformation of the original features. Parameter-based models encode relatedness through regularization or prior on parameters. Instance-based techniques require the usage of data from all tasks to construct task-specific learners with weighted instances. In the following section, exemplary works in each category are presented.

- **Feature-based MTSL** In this category, multi-task models can be divided into three approaches: feature transformation, feature selection, and deep learning. The first approach turns the original features into a linear or nonlinear shared feature representation. The second assumes that a subset of the original features will suffice. Lastly, in the deep-learning approach, a deep neural network is used to learn a shared representation encoded in the hidden layers for multiple tasks.
  - **Feature transformation approach** This approach has different ways of sharing feature representations between multiple tasks. One way is to use a multi-layer feedforward neural network [18], which consists of an input layer, a hidden layer, and an output layer. The hidden layer contains multiple nonlinear activation units and receives the transformed output of the input layer as the input where the transformation depends on the weights connecting the input and hidden layers. Another way is to use the multi-task feature learning(MTFL) method and multi-task sparse coding method [5], [4], which is formulated under the regularization framework by first transforming data instances as $\hat{x}_j^i = U^T x_j^i$ and then learning a linear function as $f_i(x_j^i) = (a^i)^T \hat{x}_j^i + b_i$. Both of these methods are based on the regularization framework and aim to learn a linear transformation U. However, there are several differences between the two methods - for example, in the multi-task feature learning(MTFL) method, U is supposed to be orthogonal, while in the multi-task sparse coding method, U is overcomplete.
  - **Feature selection approach** In [90], a probabilistic interpretation of multi-task feature selection methods based on $l_{p,1}$ regularization is proposed, which revealed that the regularizer was akin to a prior with $w_{ji} \sim GN(0, \rho_j, p)$. This prior was then extended to the matrix-variate generalized normal prior for the pur-

pose of learning relations among tasks and detecting outliers. Subsequently, horseshoe prior was used for MTL in [39] and [40], with the difference lies in how [39] generalized it to learn feature covariance while [40] applied it as a basic prior in order to distinguish outlier tasks differently.

  - **Deep Learning approach** The deep-learning approach to modeling data involves using neural networks with a large number of hidden layers. The cross-stitch network proposed in [63] is a deep-learning model that combines the hidden feature representations of two tasks to construct more powerful hidden feature representations.

- **Parameter-based MTSL** Parameter-based MTSL uses model parameters to establish connections between different tasks. We classify these into five groups, namely the low-rank, task-clustering, task-relation learning, dirty and multi-level approaches. Low-rank modeling assumes that related tasks share similar parameter values. Task-clustering divides tasks into clusters where they are assumed to share identical or near similar parameters. In contrast, task-relation learning directly derives the pairwise connections between tasks from the data. The dirty method decomposes the parameter matrix into two components which are regularised using sparse techniques. Lastly, the multi-level approach goes further by splitting the matrix into several components to capture complex relations among all involved tasks - an overview of each of these will be discussed in more detail later on.

  - **Low-rank approach** The low-rank subspace assumption is adopted for similar tasks, where the parameter w^i is modeled as $w^i = u^i + \theta^T v^i$ with $\theta \in^{h \times d}$ and $h < d$. To eradicate redundancy, the parameters of this model are trained by optimizing the training loss on all tasks, under the constraint that θ is orthonormal (so that $\theta\theta^T = I$). In [20], a squared Frobenius regularization was added to W to yield an extended version of this model, which can then be relaxed into a convex objective function.

  - **Task-clustering approach** The task-clustering approach sorts tasks into clumps based on the similarity of their model parameters. The first algorithm proposed to do this decoupled the clustering process from the model-learning one, which could be suboptimal. A multi-task Bayesian neural network outlined in [7] used a Gaussian mixture model to allocate tasks according to their parameters (e.g.,

weights linking the hidden and output layers). Additionally, [85] suggested utilizing a Dirichlet process for task clustering based on w^i model parameters.

- **Task-relation learning approach** The task relations are either determined by model assumptions [28], [67] or by a priori information. A multi-task Gaussian process is proposed in [14] to define a prior on f_j^i, the functional value corresponding to x_j^i, as $f \sim N(0, \Sigma)$. The entry in Σ corresponding to the covariance between f_j^i and f_q^p is defined as $\sigma(f_j^i, f_q^p) = \omega_{ip} k(x_j^i, x_q^p)$, where $k(',')$ defines a kernel function and ω_{ip} is the covariance between tasks T_i and T_p. Based on the Gaussian likelihood on labels given f, the marginal likelihood is used to learn Ω, the task covariance to reflect the task relatedness, with its $(i,p)^{th}$ entry as ω_{ip}. A multi-task generalized process is proposed in [89] by placing an inverse-Wishart prior on Ω in order to utilize Bayesian averaging to achieve better performance.

- **Instance-based MTSL** Among the works in this category, [12] proposes the multitask distribution matching method as a representative one. The method estimates the probability that each instance is from its own task and that it is from a mixture of them all. Using softmax functions to determine ratios, this method then determines instance weights and then learns model parameters based on the weighted instances of each task.

Overy all multi-task supervised learning(MTSL) Feature-based MTSL is more suitable for applications where the original feature representation is not very informative or discriminative. However, it can be affected by outlier tasks that are unrelated to other tasks. Parameter-based MTSL is more robust to outlier tasks and can learn more accurate model parameters. Instance-based MTSL is currently being explored and seems similar to the other two types.

3.5.2 Aspect of Multi-task learning

Multi-task learning is an exciting aspect of machine learning that enables a model to learn multiple tasks simultaneously. This can be achieved by sharing task-related information between the tasks, or by using a model with increased capacity that can learn multiple tasks. Multi-task learning can improve the performance of a model on all tasks by leveraging the relationships between the tasks. It also allows for more efficient use of data, as multiple

tasks can be learned from the same data set. The optimization strategy and evaluation metrics used for multi-task learning can vary depending on the nature of the tasks and the data. However, it is important to ensure that the model is capable of generalizing to new data and that all tasks are given equal importance in the learning process. To Design an efficient model infrastructure of multi-task some aspects should be considered.

- **Task Relatedness** Obviously, one cannot expect that information gained from the learning of a set of tasks will be relevant to the learning of another task that has nothing in common with the already learned tasks. By leveraging multiple shared information sources, MTL has the potential to improve the process of completing multiple tasks. Its theoretical advantages, however, come with significant practical challenges. Too often, domain experts are needed to properly understand the data and configure a suitable shared representation - a form of knowledge that is usually expensive and not always available. Furthermore, existing MTL approaches require task relatedness to be established beforehand based on prior knowledge; something that would normally come intuitively to humans but requires an expert level of understanding for MTL models.
- **Learning Capacity** Efficacy of multi-task learning can also be affected by the model's learning capability, as well as the number of parameters it uses. As well as being able to learn more complex relationships between tasks, a model with a larger capacity may be more susceptible to overfitting. In contrast, a model with a smaller capacity may be less effective at learning complex relationships but may be more resistant to overfitting.
- **Dataset size** The benefits of multi-task learning include more robust shared feature representations and improved performance on all tasks. However, the size of the data set can have a significant impact on the ability of the model to learn these shared representations. Small data sets are limited in their ability to learn shared representations. This is because they do not have enough data points to accurately capture the variance in the data. As a result, the models trained on small data sets are less likely to generalize well to new data. Large data sets, on the other hand, are more likely to contain redundant information. This redundancy can lead to overfitting, which can ultimately degrade performance. However, large data sets also have the advantage of being able to learn more complex feature representations.

The trade-off between data set size and model performance is an important one to consider when designing multi-task learning models. In general, it is best to choose larger data sets, as they will provide more information for the model to learn from. In engineering, we are so often faced with the problem of not having enough data that we would always prefer larger data sets. Computational resources are now so highly available that they are rarely a limit in engineering data science, and if they are, the data sets can still be reduced. But having a big data set in the beginning is always an advantage.

- **Optimization Strategy** In recent years, there has been a growing interest in the impact of optimization strategies on the performance of multi-task learning models. A variety of approaches have been proposed, ranging from task-specific objective functions [18] in aggregated loss functions to partitioning methods.

  Most MTL works seek to optimize an objective function that comprises all the task-specific individual objectives. A number of approaches use dynamic loss weighting to regulate the impact of diverse tasks and capture the learning behavior of each task. Some solutions model it as a multi-objective optimization process, eventually reaching a Pareto optimal point, where any further improvement in one task will be harmful to another. Other techniques involve changing the task-specific gradients when gradients disagree before combining them – with this idea that they can have destructive effects on some tasks. Whilst these methods may provide better convergence in certain areas of the loss landscape, their guarantees are local and become insignificant when it comes to complex, nonconvex landscapes. This is why favor partitioning strategies here; as they do not limit learning potential and avoid greedy improvements. Partitioning parameters in order to allow different tasks to share parameters without interfering with each other. In order to reduce task interference. Some of these methods involve training a binary partitioning of the parameters (either with the Gumbel Softmax trick [44], [57] or by making the partitioning randomly evolve), while others involve using task-specific Squeeze and Excitation blocks [59].

3.5.3 Approaches in multi-task learning

Multi-Task Learning is a new type of learning model in machine learning, which improves the generalization performance of all related tasks by leveraging valuable information from various sources. This strategy can be advantageous when the tasks have a shared underlying structure, as it enables the model to leverage what has been learned from one task for another, thus improving the overall performance of all tasks. There are diverse ways in which multi-task learning can be applied, each with its own pros and cons. One popular method is utilizing a joint representation, where a single set of parameters is learned for all tasks. An alternative approach deploys an isolated representation for each task, necessitating a distinct set of parameters for every task. Finally, a hierarchical representation may also be employed in order to share parameters across various levels of abstraction. To encourage the model to learn related tasks, regularization techniques can also be used. If the model has large differences in the parameters learned for different tasks, a regularization term can be added to the objective function that penalizes it. In addition, there are also techniques such as Gradient Reversal Layer and Adversarial Multi-task learning, which use adversarial techniques to learn the shared representations between the tasks. There are many more techniques that have been proposed in the literature for multitasking learning, and these are just a few examples. For algorithmic modeling, introduce the definition of MTL approaches.

- **Hard Parameter Sharing** Hard parameter sharing is the go-to approach for MTL in neural networks, stretching back to [19]. It involves sharing the hidden layers across all tasks, with separate task-specific output layers. This reduces the likelihood of overfitting as Baxter [8] demonstrated that the chance of over-fitting the shared parameters is in order N (N being the number of tasks) lower than for task-specific parameters. Intuitively this makes sense - by simultaneously learning more tasks, a model must find a general representation suited to them, therefore decreasing the risk of over-fitting on any smaller original task.
- **Soft Parameter Sharing** In opposition to hard parameter sharing, soft parameter sharing allows each task to have its own model with distinct parameters. This has the effect of acting as a weight regularizer, as demonstrated by Duong et al. in [27]. As Yang et al. further attest in [86], this approach also works with factorized repre-

sentations (e.g. Tensor-Train [66] and Tucker [81]), whilst maintaining independence between tasks and allowing some influence between them due to the similarity in their parameter space.

- **Multi-head model** A common approach to multi-task learning is to use a shared representation for all tasks, with each task having its own output layer. Multi-head models are an extension of this approach, where each task has its own output layer and the hidden layers are shared.

  Multi-head models have several advantages over traditional multi-task learning models. First, they allow for task-specific representations in the hidden layers, which can lead to improved performance on some tasks. Second, they make it possible to learn tasks in parallel, which can speed up training. Finally, they can provide more interpretability than traditional multi-task models, as each task has its own output layer and so the effect of each task on the hidden layers can be more easily analyzed.

  Despite these advantages, multi-head models are not without their challenges. One challenge is that they can be more difficult to train than traditional multi-task models, as the gradients for each task must be calculated separately and then combined. Another challenge is that they can require more memory than traditional multi-task models, as each task requires its own output layer. Despite these challenges, multi-head models offer a promising approach for improved performance in multi-task learning.

- **Hierarchical MTL** Hierarchical training is an approach that trains a series of nested neural networks, where each subsequent network learns features that are more specific to its task than the previous networks. This approach is effective for complex problems that can be decomposed into a hierarchy of sub-problems, but it may be computationally expensive.

- **Transfer Learning** Transfer learning is an approach where a model trained on one task is reused to solve another related task. This can be done by either fine-tuning the model with new data or by using the model as a pre-trained feature extractor. Transfer learning can be used when there is not enough data for training a model from scratch or when the target task is different from the source task but has similar input data.

3.5.4 Optimization for multi-task learning

MTL architecture design acts as the modern generalization of hard parameter sharing, while MTL optimization is an expanded version of soft parameter sharing. Soft parameter sharing acts as a regularization technique that punishes the deviation of model parameters between corresponds of different, yet related tasks. Although there are several strategies to regularize model parameters via MTL optimization, researchers continue to develop new methods in this area. By viewing the issue of negative transfer from an optimization perspective, approaches other than various parameter-sharing systems arise.

By grouping MTL optimization methods into six categories, this review seeks to provide a better understanding of various research directions. Namely: loss weighting, regularization, gradient modulation, task scheduling, multi-objective optimization and knowledge distillation. Although the boundaries between these categories may overlap at times, this categorization is helpful in conceptualizing the diverse approaches taken towards MTL optimization.

- **Loss Weighting** A common tactic for handling multi-task optimization is to strike a balance between the loss functions for each of the tasks. When models are involved in more than one task during training, it is necessary to combine the distinct loss functions into a single aggregated one that is optimized. A significant factor in determining how to construct this aggregated loss function is what weights should be assigned to each individual loss. Several methods have been developed to calculate these weights and are discussed in this [33].

- **Regularization** Regularization is an indispensable part of multi-task learning, particularly soft parameter sharing. In this technique, the L2 distance between the parameters of task models is incorporated into the training objective to motivate consistency between various tasks. Soft parameter sharing is both easy to work with and has been used considerably for MTL methods.

  [27] proposed a method for training a dependency parser using soft parameter sharing across multiple languages. This approach was shown to outperform hard parameter sharing in the small data setting. [86] proposed a variant of soft parameter sharing, which encourages the learning of parameter vectors across tasks that are similar, by measuring similarity with the trace norm of a matrix instead of the L2 distance.

The authors interpret the resulting trace norms after training as a measure of sharing strength between corresponding layers in different task models and found that the sharing strength decreases with layer depth. Multilinear Relationship Networks (MRNs) [55] are a type of MTL model that regularizes by imposing a tensor normal distribution as a prior over the parameters in task-specific layers of multi-task models.

Deep Asymmetric Multitask Feature Learning (Deep-AMTFL) [52] is a method of regularizing deep multi-task neural networks by introducing an autoencoder term to the objective function. This auxiliary task involves reconstructing the features from the second to last layer of a network from the network output so that each of the task predictions is used to construct the features for all other tasks, a task that was proposed by Asymmetric Multi-Task Learning [51].

- **Task Scheduling** Task scheduling is the process of deciding which task or tasks to train on at each training step. Most MTL models use simple task schedulers, like uniform task sampling [27] or proportional task sampling [72]. However, it's a well-known fact that optimized task scheduling can significantly improve model performance [10]
- **Gradient Modulation** MTL is challenged by negative transfer, which occurs when joint training of tasks causes damage instead of support. From an optimization point-of-view, it is characterized by conflicting task gradients. When the gradients for two tasks are opposed to each other, following the gradient for one task will decrease performance in the other. Neither task can benefit from being trained together as they perform worse than if they were trained separately. To address this issue, explicit gradient modulation has been proposed as a possible solution. This involves altering training gradients either through adversarial techniques or by changing them when conflicts occur.
- **Knowledge Distillation** Originally presented as a method to condense an ensemble of non-neural models into one [16] knowledge distillation has seen many uses outside its initial scope. In particular, many studies in multi-task learning (MTL) have utilized knowledge distillation to pass on the expertise from multiple single-task "teacher" networks to one single "student" network. Interestingly, under certain conditions, the performance of the student model can even be superior to those of the teacher networks, making it an appealing technique with both memory and performance ben-

efits

- **Multi-Objective Optimization** Multi-objective optimization consists of optimizing several objective functions simultaneously. However, no globally optimal solution exists in this type of problem since it is not possible to improve the performance for any one objective without worsening the performance of another. In such cases, the best feasible options are referred to as Pareto optimal solutions and they make up what is known as the Pareto frontier - a set of solutions that represent the best trade-off between objectives. [26] research paper discussing how multi-task learning can be improved by using multiple-gradient-based methods to optimize multiple objectives. The paper shows that these methods can actually be formulated as methods to compute adaptive loss weights, similar to existing methods.

3.6 Multi-Output Regression

Multi-output regression attempts to establish predictions based on multiple real-valued output/target variables. That is why also known as a multi-target[1], [47] problem. When the outputs are binary, it is known as multi-label classification [58], [88], [80]; for discrete values (not just binary), it is referred to as multi-dimensional classification [13]. Existing methods for multi-output regression fall into two categories: problem transformation approaches (a.k.a. local methods) which treat the multi-output problem as a set of single-output issues that are solved independently using individual regression algorithms, and algorithm adaptation strategies (a.k.a global or big-bang methods) that alter a specific single-output system (such as decision trees and support vector machines) to deal with multi-output datasets right off the bat. Multi-output regression is distinct from multi-task procedures in that tasks may have different training sets and/or varied descriptive features, whereas target variables usually have shared data and/or descriptive elements.

3.6.1 Mathematical Definition

Suppose training data set, denoted as D, consisting of N instances each having m descriptive or predictive variables $x^{(l)} = (x_1^{(l)}, ..., x_j^{(l)}, ..., x_m^{(l)})$ and d target variables $y^{(l)} = (y_1^{(l)}, ..., y_i^{(l)}, ..., y_d^{(l)})$. The goal is to generate a multi-target regression model from the given

data, which requires finding an appropriate function F that assigns an output vector y to the corresponding input vector x.

$$\begin{aligned} F : \Omega_{X_1} \times ... \times \Omega_{X_m} &\longrightarrow \Omega_{Y_1} \times ... \times \Omega_{Y_d} \\ X = (x_1, ..., x_m) &\longrightarrow Y = (y_1, ..., y_d) \end{aligned} \tag{3.11}$$

In above equation (3.11) Ω_{X_j} and Ω_{y_i} are the sample spaces for each predictive variable X_j. Similarly, for each of the variables $i \in \{1, ..., d\}$, and each of the target variables Y_i. Note in this case, all target variables are considered continuous. The model will then be applied simultaneously to predict the values of $\{\hat{y}^{(N+1)}, ..., \hat{y}^{(N)}\}$, and all target variables of the new incoming unlabeled instances $\{x^{(N+1)}, ..., x^{(N)}\}$.

In this thesis, our problem also falls under problem transformation approaches. According to the review in the previous section, there are already several machine-learning and deep-learning methods that are very well established. But our thesis aims are different than using only a single task-learning method. In a further chapter, our aim is to present how problem transformation approaches can also be solved by multi-task learning.

4 Research Questions

4.1 Objective

While traditional engineering systems have been designed to focus on single tasks, recent advances in machine learning have enabled the development of holistic data-driven models that can take into account multiple tasks simultaneously. This approach can be particularly useful in resolving challenging engineering problems that require a multi-disciplinary approach.

In engineering, a holistic data-driven model of an engineering system is a system that takes in as input holistic measurements of phenomena such as mechanical properties, material properties, electric properties and chemical properties. The outputs of the model include predictions about the system's behavior. This thesis aims to introduce the concept of multi-task learning, which is still a nascent technique in machine learning that has become an important topic of discussion among academic researcher, machine learning engineer and data scientists.

The aim of this thesis is first to present the challenges of regression problems in task learning. Single-task learning is when a machine learning algorithm focuses on only one task. This can be difficult because the algorithm has to learn all the features and patterns that are relevant to that one target task. Multi-task learning is when an algorithm learns multiple tasks simultaneously. This can be beneficial because it allows the algorithm to learn features and patterns that are relevant to multiple output target tasks, which can make the overall learning process more efficient. Therefore, the main objectives of this thesis is to determine whether or not multi-task learning is a better way to solve engineering system problems.

4.2 Task Definition

In this thesis, the different steps towards multi-task learning for engineering system to develop holistic models are presented. The following questions are answered over the course of this chapter. (1) To understand the behaviour of the task relationship for the generation

of synthetic data, (2) implementation of models representing the learning strategies multi-task and single-task learning, (3) comparison of multi-task learning with single-task learning as baseline for different scenarios, (4) validation of hypothesis on real-world data-sets in the areas of process and material engineering.

4.3 Research Problem

When it comes to machine learning, one of the key research questions is how to best utilize data patterns in order to improve learning. This is especially important when it comes to deep learning, which often requires large amounts of data in order to train a model. As machine learning and deep learning algorithms become more sophisticated, the need for high-quality training data becomes more critical. One way to meet this demand is to generate synthetic data. In Further section of this thesis following research questions are answered:

- How to generate synthetic data without using a generative model for multi-output regression problems?
- Importance of task relationship and feature size for model performance.
- How the learning ability of different model approaches work in different scenarios.
- Understanding pattern of synthetic data for further analysis in real data-set

5 Methods

In order to answer the research questions and fulfill the main objective of this study, one single-task model and three multitask models were constructed. As an effort to boost performance, hyperparameter tuning processes and target task correlation calculation Pearson method were set during the data generation process. The performance metrics used are described in section 5.2, the synthetic datasets in 5.4, and the real datasets in 5.5.

5.1 Data Pipeline

In order to ensure that the model's performance was evaluated and not the preprocessing method, a preprocessing pipeline was designed and kept identical for each evaluation. Training and validation data tables were separated before each evaluation. Similarly, the layer type evaluation and the final architecture evaluation used the same data, except for the evaluation of performance based on target task co-relation described in section 5.6.

This research aimed to study the effects of task correlation on model performance. It hypothesized that using task-relatedness would provide a larger knowledge base and data augmentation, as well as act as a regularizing agent to help the model generalize better. However, there was no clear consensus on when, why, and how task co-relations could be used effectively. Thus, they were applied to each model at various sections in order to assess if they improved the performance or not.

The Multi-Task Learning architecture has many possible implementations, but it is still unclear when, how and why they are effective. To explore this, four architectures were tested. The first was a Single-Task learning decision-tree-based ensemble Machine Learning algorithm - which shared no tasks - used as the benchmark for comparison to the other Multi-Task models. The second was a multi-gate mixture of experts that applies a gating system to pick out one of a few experts for every task. The third model learned separate models for each task while sharing certain parameters between them; it's called a track specifics layer model. Finally, the fourth model was simply a feed-forward word layer that accepted concatenated vectors representing all tasks as its input.

In today's world, a model can be designed in a variety of ways, and the structures chosen

will also have a significant impact on its performance in the end. This is also an important complication when researching Machine Learning as there is often too little time to explore every possibility. In order to deal with this, we chose the structures in a simple and logical manner, based on a proposed architecture from the research phase. more details of model MTL and STL architectures are presented respectively in sections 5.6 and 5.7.

5.2 Metrics

The objective of the regression is to find a model that minimizes the prediction error of all data points. Evaluation of the model's accuracy is essential for any machine learning model. To evaluate the model's performance in regression analysis, Mean Squared Errors, Mean Absolute Errors, Root Mean Squared Errors, and R-Squareds or Coefficients of Determination are used.

- **Mean absolute errors(MAE)** The absolute error (AE), which is the absolute value of the difference between the predicted and expected value for a given observation, is the metric used to compare the outcomes from various network models. By merely averaging the metric across all outputs, it may be expanded to evaluate regression models with multiple outputs.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y_i}|, \tag{5.1}$$

- **Mean squared errors(MSE)** Mean squared error is a statistical measure of how close a predicted value is to the actual value. It's used in both machine learning and deep learning, and accuracy is one of its most important aspects. MSE is calculated by taking the average of the squares of the difference between the predicted and actual values. The smaller the MSE, the closer the predicted values are to the actual values.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2, \tag{5.2}$$

- **R-squareds** The coefficient of determination, also known as R-squared, is a measure of the accuracy of a machine learning or deep learning model. It is used to determine

how well the model predicts future data points and is a key metric in assessing the performance of a machine learning or deep learning algorithm. The higher the R-squared value, the more accurate the model is.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y_i})^2}{\sum_{i=1}^{N}(y_i - \tilde{y_i})^2}, \tag{5.3}$$

5.3 why synthetic data / Social cause about synthetic data

Synthetic data has been gaining popularity in recent years as a way to train machine learning models. There are several reasons for this: first, it can be generated very quickly and cheaply. Second, it can be generated in large quantities. Third, it can be generated to match specific requirements, such as being representative of a certain population. Finally, it can be used to test hypotheses about how machine learning models will perform on real data.

Synthetic data is generated by applying a data generation method to create new, realistic data that conforms to the same statistical properties as the original dataset. This involves using mathematical models or algorithms to mimic the real-world processes that generated the original data.

One of the most exciting things about synthetic data is that it opens up new possibilities for testing hypotheses about machine learning models. For example, we might want to know how a model will perform on data that is very different from the training data. Or we might want to know whether a model is biased against certain groups of people. With synthetic data, we can generate data that is specifically designed to test these hypotheses. There are many advantages to using synthetic data. First, it can be used to train machine learning and deep learning models without the need for expensive and time-consuming data collection efforts. Second, it can be used to test hypotheses about how these models will perform in the real world. Finally, it can be used to fill in gaps in real-world datasets (for example, if some data is missing).

However, there are also some challenges associated with synthetic data. First, it can be difficult to ensure that the generated data is realistic enough to be useful for training and testing purposes. Second, there is a risk of overfitting if the model is not properly validated on real-world data. Finally, synthetic data may not capture all of the nuances and complex-

ities of the real world, which could limit its usefulness. Overall, synthetic data has a lot of potentials but there are also some challenges that need to be addressed. With the right approach, however, these challenges can be overcome and synthetic data can become a powerful tool in the world of data.

5.4 Synthetic data preparation

There are many reasons why not to use a generative model to generate synthetic data for this thesis. One reason is that they use original data to generate synthetic data. This can be a problem because the original data may not be representative of the population, and so the synthetic data may not be either. Additionally, if there are errors in the original data, these will be carried over into the synthetic data. Another reason is that it can be difficult to control what features are generated and whether target tasks have a correlation or not, so that kind of synthetic data may not contain all of the information that is needed. Finally, generative models can be computationally expensive, so it may not be feasible to use them for large data sets. In this thesis, our goal is to understand the multi-output/target task relatedness impact of the multi-task learning model capability.

An earlier research study [27, 63] uncovered that the natural task relatedness in the data had a major influence on how multi-task learning models have performed. It is hard to straightforwardly evaluate how task relatedness impacts these models since we cannot adjust the relatedness between tasks and measure the effect in real-world applications. Accordingly, we use synthetic data where it is simple to gauge and modify the relatedness, to construct an empirical inquiry into this connection.

5.4.1 Creation of the synthetic data

We create two regression tasks in the approach of Kang et al [45]. and use the Pearson correlation of the labels of these two tasks as the quantitative task association indicator. We set the regression model as a combination of sinusoidal functions as used in [76] because we focus on DNN models rather than the linear functions used in [45]. The synthetic data is produced specifically as follows.

1) We create two orthogonal unit vectors $u_1, u_2 \in R^d$ from the input feature dimension d.

$$u_1^T u_2 = 0, \|u_1\|_2 = 1, \|u_2\|_2 = 1, \tag{5.4}$$

2) Create two weight vectors w_1, w_2 that are such that given a scale constant c and a correlation score of $-1 \leq p \leq 1$ respectively.

$$w_1 = cu_1, w_2 = c(pu_1 + \sqrt{(1-p^2)}u_2), \tag{5.5}$$

3) Sampling at random from $N(0,1)$ each of the elements of an input data point $x \in R^d$.

4) Create two labels, y_1, and y_2, for the subsequent two regression tasks:

$$y_1 = w_1^T x + \sum_{i=1}^{m} sin(\alpha_i w_1^T x + \beta_i) + \epsilon_1 \tag{5.6}$$

$$y_1 = w_2^T x + \sum_{i=1}^{m} sin(\alpha_i w_2^T x + \beta_i) + \epsilon_2 \tag{5.7}$$

5) where the parameters given to regulate the shape of the sinusoidal functions are $\alpha_i, \beta_i = 1, 2, ..., m$ and $\epsilon_1, \epsilon_2 \tilde{} N(0, 0\,01)$.

6) Repeat processes (5.6) and (5.7) until enough data are generated.

- **Non-sparse Tabular data** The method (5.5.1) is used to generate non-sparse tabular artificial data for regression tasks. First, two orthogonal unit vectors u_1 and u_2 in the feature dimension d are created. Then two weight vectors w_1 and w_2 are created from a scale constant c and a correlation score of 0 to 1. The elements of an input data point x are then sampled at random from $N(0, 1)$. After that, two labels y_1 and y_2 are created for the two regression tasks using sinusoidal functions with parameters α_i, β_i, and ϵ_1, ϵ_2. Finally, this process is repeated until enough data has been generated. this process generates non-sparse tabular data.
- **Sparse Tabular data** The method (5.5.1) is also used to generate sparse tabular artificial data for regression tasks. but for generating sparse tabular data then we have made some changes to generate sparse tabular data. For Generating sparse tabular data chose any random feature column and then set a threshold value of 0.5. if a

feature data point is greater than 0.5 the new data value is 1 or if less than 0.5 then the new data point value is 0.

5.5 Real world Data

The six different types of data sets are often used in thesis projects. Real data sets can come from different domains of engineering, such as material and process engineering. This data can be used to validate hypotheses and help to support the conclusions of the thesis. There are many different sources of real data sets, such as databases, journals, and online repositories. Using a variety of data sets is important to ensure that the results of the thesis are accurate and reliable. More details for these data sets are described below.

5.5.1 Description of the datasets

- **Huang2021 [42]**

  Several studies found that machine learning models are better than traditional methods at predicting the mechanical properties of carbon nanotube-reinforced cement composites. The length of the nanotubes and the curing temperature were found to be the most important factors affecting the compressive and flexural strength, respectively. To develop an MTL and STL model, a comprehensive dataset must be assembled to cover various parameters influencing the mechanical performance of CNT-reinforced cementitious composites. This study has accumulated 114 experimental samples, whose features are shown in Fig 5.1 [42]. the compiled dataset was split into training and testing at a ratio of 0.2.

Input variables				
Type of cement	1	2	–	I_1, discontinuous value (1 and 2)
Water-to-cement ratio	0.200	0.500	0.367	I_2, continuous value
Content of carbon nanotubes	0.000	0.008	0.002	I_3, continuous value
External diameter (nm)	4.000	250.250	19.974	I_4, continuous value
Length (μm)	1.000	250.250	21.588	I_5, continuous value
Functionalization method	1	4	–	I_6, discontinuous value (1, 2, 3 and 4)
Curing days	3.000	28.000	20.623	I_7, continuous value
Curing temperature (°C)	20.000	30.000	23.193	I_8, continuous value
Dispersion method	1	5	–	I_9, discontinuous value (1, 2, 3, 4, and 5)
Output variables				
CS (MPa)	27.300	154.400	73.132	O_1, continuous value
FS (MPa)	4.000	16.900	10.255	O_2, continuous value

Figure 5.1: Feature descriptions of training and testing data collected from the literature [42]

- **Guo 2019[36]**

This steel production data was collected by the Shanghai Meishan Iron and Steel Corporation Ltd. of Bao Steel Group. From the original 65,288 samples, 2151 were excluded due to incomplete or incorrect information, and 63,137 were chosen for study. The 27 features listed in Table 5.1 served as the process parameters and chemical compositions of the samples, which contained three properties: yield strength (YS), tensile strength (TS), and elongation (EL).

Table 5.1: Features and their numbers of steel production data [36]

Number	Feature	Number	Feature
1	Furnace temperature	15	Titanium content (Ti)
2	Exist temperature	16	Boron content (B)
3	Annealing temperature	17	Tin content (Sn)
4	Thickness	18	Arsenic content (As)
5	Width	19	Zirconium content (Zr)
6	Sulfur content (S)	20	Calcium content (Ca)
7	Copper content (Cu)	21	Lead content (Pb)
8	Nickel content (Ni)	22	Ceq (Carbon Equivalent #1)
9	Chromium content (Cr)	23	Pcm (Carbon Equivalent #2)
10	Molybdenum content (Mo)	24	Antimony content (Sb)
11	Vanadium content (V)	25	Nitrogen content (N)
12	Niobium content (Nb)	26	Oxygen content (O)
13	Total Aluminum content (Al)	27	Tungsten content (W)
14	Acid soluble Aluminum content		

- **Hu 2021 [62]** Aluminium (Al) alloys are invaluable to the aerospace and automotive industries due to their low density, large specific strength, outstanding corrosion resistance, and formability, as well as affordable manufacturing and maintenance costs. In this work, both commercially available engineering wrought Al alloys and those with modified compositions were collected to build a database of 930 samples used for multi-task learning analysis. This database encompasses features (such as chemical composition and manufacturing process) as well as targeted properties (e.g. YTS, UTS and ELONG). The data is pre-processed in order to ensure efficient integration of features during model training.

	Variable (Unit)	Min.	Max.	Average	Std.
Features	Si (wt pct)	0	12.25	0.381	0.761
	Fe (wt pct)	0	1.6	0.343	0.305
	Cu (wt pct)	0	6.3	1.587	2.013
	Mn (wt pct)	0	2.06	0.324	0.380
	Mg (wt pct)	0	5.8	1.247	1.361
	Cr (wt pct)	0	0.5	0.068	0.092
	Ni (wt pct)	0	2	0.041	0.225
	Zn (wt pct)	0	8.69	0.682	1.707
	Ga (wt pct)	0	0.03	0.001	0.003
	V (wt pct)	0	0.158	0.006	0.022
	Ti (wt pct)	0	0.2	0.048	0.070
	Zr (wt pct)	0	0.4	0.029	0.059
	Bi (wt pct)	0	0.55	0.011	0.068
	Pb (wt pct)	0	0.55	0.011	0.068
	Li (wt pct)	0	4	0.171	0.625
	B (wt pct)	0	0.06	0.001	0.006
	Sc (wt pct)	0	1.44	0.050	0.172
	Be (wt pct)	0	0.1	0.000	0.007
	Solution/annealing T (°C) (SA Temp)	25	649	327.129	230.421
	Ageing T (°C) (A Temp)	25	290	89.121	84.413
	Ageing Time (h) (A Time)	0	1440	18.973	105.008
Targeted Properties	YTS (MPa)	9	684	246.562	139.443
	UTS(MPa)	34	732	311.470	151.844
	Elongation (pct EI)	0.5	50	12.583	8.078

Figure 5.2: Numerical Features and Targeted Properties in the Database with Minimum, Maximum, Average and Standard Deviation Values [62]

- **UCI-CBM [24]**

  The article discusses the use of machine learning methods for condition-based maintenance of gas turbines used for vessel propulsion. The authors test the performance and advantages of using machine learning methods in modeling the degradation of the propulsion plant over time. Experiments conducted on data generated from a sophisticated simulator of a gas turbine show that the proposed machine-learning approaches are effective in this maritime application. This data set contains 11, 934 samples for the feature set given in table 5.2. our aim is to apply approaches MTL and STL to predict target the prediction of the GT compressor decay and GT turbine decay.

Table 5.2: Features and their numbers of steel production data[24]

Variable	Symbol	Units
Lever position	lp	$(\cdot)$
Ship speed	v	knot
Gas turbine shaft torque	G T_T	kN m
Gas turbine rate of revolutions	G T_n	r / min
Gas generator rate of revolutions	G G_n	r / min
Starboard propeller torque	T_s	kN
Port propeller torque	T_p	kN
HP turbine exit temperature	T_{48}	°C
GT compressor inlet air temperature	T_1	°C
GT compressor outlet air temperature	T_2	°C
HP turbine exit pressure	P_{48}	bar
GT compressor inlet air pressure	P_1	bar
GT compressor outlet air pressure	P_2	bar
GT exhaust gas pressure	P_{exh}	bar
Turbine injection control	TIC°	%
Fuel flow	m_f	kg / s

- **Xiong-2014 [84]**

  As part of rapid manufacturing, this study applies a neural network and a second-order regression analysis to predict the bead geometry during robotic gas metal arc welding. Moreover, since the neural network model has a great capacity to approximate any nonlinear process, it performs better than the second-order regression model in terms of predicting the bead width and height. The data set contains four

independent input process feature feed rate (F), welding speed (S), arc voltage (V), and nozzle-to-plate distance (D) and target responses were bead width (W) and bead height (H). In this data set 31 samples for training 12 sample used for testing.

Parameters	Symbol	Factor levels				
		Level 1	Level 2	Level 3	Level 4	Level 5
Wire feed rate (m/min)	F	2.8	3.6	4.4	5.2	6.0
Welding speed (cm/min)	S	15.0	22.5	30.0	37.5	45.0
Arc voltage (V)	V	16.0	17.5	19.0	20.5	22.0
Nozzle-to-plate distance (mm)	D	6.0	9.0	12.0	15.0	18.0

Figure 5.3: Numerical Features of Xiong datasets [84]

- **Yin-2014[82]**

  Based on previous micro bond tests, this work presents machine learning-assisted models for determining the interfacial properties. It is demonstrated that the proposed model predicts interfacial shear strength and maximum force effectively through a comparison between pullout test results and prediction results. There is a reliable relationship between influencing attributes and interfacial properties.

  Fiber pullout tests and simulations are reliable, quantitative techniques for evaluating the F_{max} and $IFSS$ of fiber-reinforced composites. This study presents a comprehensive dataset consisting of 922 fiber pullout test results from different sources and the literature, including 818 results from experiments that incorporated features described in Table Figure 5.4 and an additional 104 from simulations. The input data includes 11 independent features that address fiber properties, sample preparation environment, and test conditions; the output yields two parameters - F_{max} and $IFSS$ - which provide insight into the interfacial properties. Results reveal that all variables are highly independent and suitable for training the model. To obtain an unbiased distribution of data, all samples were split into training (experimental data), validation (experimental data), and testing (simulation data) groups. The training and validation

datasets were used to train and validate the model respectively; the testing dataset was utilized to evaluate its generalization ability on unseen data points.

Variables	Min	Max	Mean	Description
Input variables				
Type of fiber	1	10	–	I_1, discontinuous value (1–10)
Fiber diameter (μm)	5	300	47.14	I_2, continuous value
Embedded length (μm)	24.5	2018.0	411.0	I_3, continuous value
Young's modulus of fiber (GPa)	3.39	294	160.27	I_4, continuous value
Poisson's ratio of fiber	0.17	0.39	0.26	I_5, continuous value
Type of matrix	1	10	–	I_6, discontinuous value (1–10)
Young's modulus of matrix (GPa)	1.2	3.96	2.90	I_7, continuous value
Poisson's ratio of matrix	0.31	0.37	0.34	I_8, continuous value
Loading rate (m/s)	0.0017	6	0.109	I_9, continuous value
Preparation temperature (°C)	20	370	137.0	I_{10}, continuous value
Test temperature (°C)	−196	120	17.45	I_{11}, continuous value
Output variables				
F_{max} (N)	0.0048	1.9019	0.3022	O_1, continuous value
IFSS (MPa)	29.73	137.48	36.81	O_2, continuous value

Figure 5.4: Feature representations of 818 training and validation samples accessed from experimental results [82]

5.5.2 Balancing data

A data set is a collection of data, usually organized in tabular form. Each row of the table represents a single data point, and the columns represent the attributes or features of that data point. A well-balanced data set is one that has an equal number of data points for each attribute. This is important because it helps to ensure that the machine learning algorithm you train on the data will be able to generalize well to new data.

There are several ways to balance a data set. This work uses two methods. One common method is to split the data into a training set and a validation set, making sure that each set has an equal number of data points for each attribute. Another common method is to use a sampling technique such as stratified k-fold(for Single task learning) and random permutation (for multi-task learning) sampling to make sure that the resulting sample has the right distribution of classes.

5.5.3 Data Preprocessing

Pre-processing of data sets is a critical step in the development of neural networks. Neural networks are designed to learn from data, and the quality of the data sets that are used to train them can have a significant impact on their performance. Poor-quality data sets can lead to neural networks that do not generalize well to new data, while high-quality data

sets can lead to neural networks that are much more accurate.

One of the most important aspects of pre-processing is ensuring that the data sets are correctly formatted for training. This includes ensuring that the inputs and outputs are properly scaled, that the data is shuffled correctly, and that any missing values are handled in a way that does not introduce bias. Another important aspect is feature selection, which is the process of choosing which inputs to use in training the network. This can be tricky, as using too many features can lead to over-fitting while using too few can limit the network's ability to learn.

Choosing the right pre-processing techniques can make a big difference in how well a neural network performs. It is therefore important to carefully consider all aspects of pre-processing when developing neural networks.In order to feed our model three-step data pre-processing pipeline was used which consisted of cleaning, missing values, and resolving skewness. They are each described in more detail in this Sections

To ensure that the models performance is tested and not the pre-processing method, this process was kept exactly the same for all evaluated models

- **Cleaning** Data cleaning involves identifying and correcting inaccuracies and inconsistencies in data. It's a critical part of any data analysis pipeline, as doing so can guarantee that your results are accurate and dependable. Data cleaners must contend with missing values frequently when working with large datasets - errors in the collection process or certain values simply not being available could be causing it. To prevent any skewing of results, you must take care to handle these values judiciously. Outliers also present an issue as these data points lie greatly from other data they may be caused by errors or represent atypical events. By taking care of these issues before beginning any analysis, you can make sure that your findings are both precise and dependable.
- **Handling Missing Labels** A Missing Value is a value in a data set that is not present. It is usually represented by a blank space, or a zero. Missing values can be caused by many factors, such as data entry error, incorrect coding, or simply missing data. In most cases, missing values are not a big deal and can be easily fixed. However, in some cases they can cause problems with your data analysis.

  For example, let's say you have a tubular data set with three columns: height, weight,

and age. Let's say the first row of data is missing the weight column. This would create a problem when trying to calculate the BMI (body mass index) because BMI requires both height and weight. In this case, you would either have to delete the entire row of data, or find another way to calculate BMI (such as using only height).

In general, it's best to avoid deleting data unless absolutely necessary. If possible, try to impute the missing values instead. Imputing is when you replace the missing values with estimated values. There are many ways to estimate missing values, but one common method is to use the mean of the non-missing values. So in our example above, you would take the mean weight of all the other rows in the dataset and use that as the estimated value for the missing weight. Of course, there are other methods for dealing with missing values and each situation is unique. In this thesis, we use those two methods for Handel missing value in real data sets.

- **Resolving skewness** Skewness is a statistical measure of the asymmetry of a data set around its mean. It can be either positive or negative, but is most often negative. That is, the data are more spread out on the left side of the mean than on the right side. This is common in data sets that are heavily left-tailed (i.e., have a long left tail).

  A data set can be described as being skewed if it has a non-normal distribution. Skewness can be caused by many factors, such as extreme values, outliers, or a lack of symmetry. The skewness of a data set is usually measured by its moment coefficient of skewness, which is a scaled version of the third moment about the mean.

  The skewness of a data set can have important implications for its analysis and interpretation. For instance, skewed data can lead to problems with estimation and hypothesis testing. Therefore, it is often necessary to transform skewed data before performing statistical analyses on it.

  In this thesis, we use two popular methods for reducing the skewness of real datasets: Box-Cox and Yeo-Johnson. Both of these methods involve transforming the data so that they are more symmetrical. The Box-Cox method is typically used for data that are positively skewed, while the Yeo-Johnson method is used for data that are both positively and negatively skewed.

- **Normalization** Normalization is a process in which data is converted into a form that can be read and processed by a machine. This process is important for two reasons:

first, it allows machines to understand and interpret data more effectively; second, it helps to prevent errors and ensure consistency in the data. Normalization is often used in conjunction with other processes, such as feature selection and feature engineering, to create more effective models. Generally, the mathematical expression of the normalization formula called "min-max scaler" is used here:

$$\bar{x}' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{5.8}$$

The above formula is used when the feature requires more or less uniform distribution across a fixed range. For example, a particular range of features is 270 to 1170. Then according to formula subtract 270 from every value of the feature and the result is divided by 1000. that process re-scale value between range [0, 1].

To ensure that the model's performance is tested and not the pre-processing method, this process was kept exactly the same for all evaluated models.

5.6 MTL Modeling Techniques

This study investigated three different methods of sharing information between tasks using different styles of sharing tasks in order to achieve multi-task learning. This Section describes the method of constructing and designing three MTL models, each utilizing a unique approach to sharing tasks and learning abilities. One Single task learning model where no task sharing information with each other is also described.

5.6.1 Multi-gate Mixture-of-Experts

Specifically learning to model task interactions from data, Multi-gate Mixture-of-Experts (MMoE)[56] is a novel multi-task learning approach that we employ in this work. This approach is a modified version of the Mixture-of-Experts (MoE) [43] structure for multi-task learning by distributing the expert submodels across all tasks and using a gating network that has been trained to optimize each task. We initially use our method on a synthetic dataset where we may modify the task relatedness in order to validate it on data with varied levels of task relatedness.

5.6.1.1 Model Architecture

MMoE Architecture as shown in Figure 5.5.[56] has a group of bottom networks, each of which is called an expert. Each expert, according to the author, is a feed-forward network. then each task's gating network is introduced. The gating networks use the input features to generate softmax gates that assemble the experts with various weights, enabling different task to employ experts in various ways. The task-specific tower networks are then updated with the findings of the assembled specialists. By learning varied patterns of expert assembly, the gating networks for various tasks can better understand the relationships between tasks.

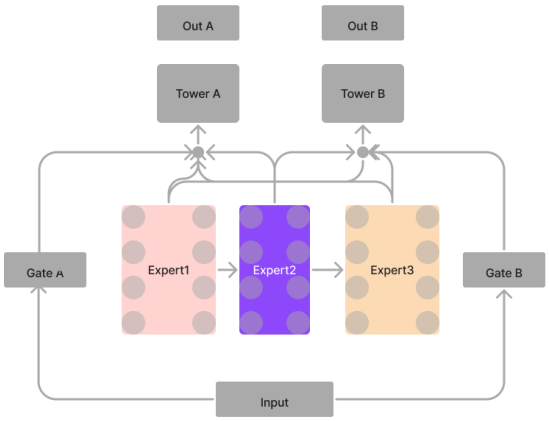

Figure 5.5: Multi-gate MoE model.[56]

5.6.2 Feed-Forward Neural Networks

The neural networks considered in this analysis have feed-forward neural network is a type of artificial neural network in which there is no cycle in the connections between the nodes. Since input is only processed in one direction, the feed forward model is the simplest type of neural network. Although the data may flow via several hidden nodes, it always proceeds forward and never backward.

5.6.2.1 Model Architecture

This model architecture was proposed by Umberto, Michelucci, Francesca Venturini [61]. This MTL network, shown in Figure 5.6, is made up of three common hidden layers with 80 neurons each, three branches that each have two additional task-specific layers to predict task A and task B separately, and one that doesn't have any further layers to predict both tasks simultaneously. Each task-specific hidden layer contains five neurons. The additional task-specific layers in this network are intriguing since they are predicted to increase the network's capacity for task prediction.

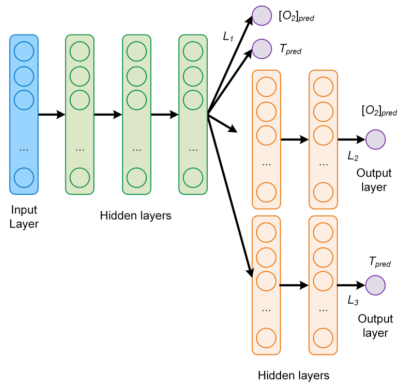

Figure 5.6: Architecture of the feed-forward MTL model.[61]

5.6.3 Neural networks $NN - MT2$[49]

This network also simple Feed Forward Neural Network.

5.6.3.1 Model Architecture

The NN-MT2 [49] model employs a concatenation-based MT architecture that only outputs the chosen polymer attribute after taking the fingerprint vector and a selector vector as inputs. 36 components make up the selector vector, with one component being 1 and the others being 0. Two dense layers, a parameterized ReLU activation function, and a dropout layer with a rate of 0.5. The two dense layers were reduced by the Hyperband technique to 224 and 160 neurons for the NN-MT2 model. To resize the output layer, a dense layer

with 1 neuron for the NN-MT2 model and 36 for the NN-MT2 model was added.

5.7 STL Modeling Techniques

Single-task learning is a modeling technique where only one task is learned at a time. This thesis investigates how well single-task learning models can be used to compare our multi-task learning approaches.

5.7.1 XGboost

A gradient boosting framework is used by the ensemble machine learning method XGBoost[21], which is decision-tree based. Artificial neural networks frequently outperform all other algorithms or frameworks in prediction issues involving unstructured data (pictures, text, etc.). However, decision tree-based algorithms are now regarded as best-in-class for small-to-medium structured/tabular data. In order to produce better results with fewer computer resources and in the shortest amount of time, it is the ideal mix of software and hardware optimization techniques. Both XGBoost and Gradient Boosting Machines (GBMs), ensemble tree approaches, use the gradient descent architecture to boost weak learners (CARTs in general). But XGBoost enhances the fundamental GBM architecture with system optimization and algorithmic improvements.

- **Parallelization:** The sequential tree construction method is approached by XGBoost using a parallelized implementation. This is made possible by the interchangeability of the two inner loops that calculate the features and the outer loop that counts the leaf nodes of a tree when creating base learners. This nesting of loops restricts parallelization because the outer loop cannot be initiated until the inner loop, which is the more computationally intensive of the two, has been finished. As a result, the order of the loops is adjusted utilizing initialization, a global scan of all instances, and sorting using parallel threads to save run time. By balancing any parallelization overheads in computation, this choice enhances algorithmic performance.
- **Tree Pruning:** Inside the GBM architecture, the tree-splitting halting criterion is greedy in nature and depends on the negative loss criterion at the split point. Instead of starting with the criterion first, XGBoost prunes trees backward using the '$max - depth$' op-

tion as specified. The "$depth - first$" method dramatically enhances computational performance.

- **Hardware Optimization:** The hardware resources will be used effectively by this algorithm. By allocating internal buffers in each thread to hold gradient statistics, cache awareness does this. Additional improvements, such as "out-of-core" computation, maximize disk space while managing large data frames that don't fit in memory.

5.7.2 Why XGBoost for Single task learning

XGBoost is the way to go if searching for the best machine learning algorithm for learning a single task. Here's why according to neural IPS journal [35]:

- Tree-based models perform much better in cases where the data does not have a smooth decision boundary but has an overly smooth solution. This is because tree-based models are more flexible when it comes to modeling non-linear relationships.
- Tree-based models are generally resilient to the impact of uninformative features in the dataset, while networks are sensitive to such information. This is because tree-based models can ignore irrelevant features, while networks will try to learn from them and potentially overfit the data.
- Tabular data contains lots of unformed features and networks are very sensitive to them. Tree-based models, on the other hand, are not rotational invariant and access each feature of the data set for training and test data in a different way from their tree-based counterparts. This means that they can learn from tabular data without being impacted by unformed features.

In general, tree-based models outperform networks for single-task learning due to their flexibility, robustness, and ability to learn from tabular data. So if you're looking for the best machine-learning algorithm for your single-task learning problem, XGBoost is a great choice!

6 Results and Discussion

Throughout this chapter, we will examine the results obtained, the data used, and the experimentation process used to address the research questions given. In section 6.1, we will discuss the setup of the hyperparameter experiment, while in section 6.2, we will discuss the results:

6.1 Hyper Parameter Tuning

Hyperparameter tuning is a process of optimizing the performance of a machine-learning model by fine-tuning the hyperparameters. The goal of this process is to find the set of hyperparameters that results in the best performance for the model on the given dataset.

There are many different methods for hyperparameter tuning, but one promising approach is to use the optuna library [2]. This library provides a TSPsampler which is Bayesian method to perform hyperparameter optimization. Using the TSPsampler, we can define a search space for our hyperparameters and then use Optuna's sampling methods to explore this space and find the optimal set of values for our model.

This approach has several advantages over other methods, including its ability to handle complex search spaces, its flexibility, and its efficiency. Additionally, Bayesian optimization [64] has been shown to outperform other popular hyperparameter optimization methods in terms of both accuracy and computational cost.

Overall, using Optuna for hyperparameter tuning is a promising approach that can lead to improved model performance.

6.1.1 Single-Task Learning

In this thesis, we aim to use the Xgboost library with the optuna library for single task learning to do hyperparameter tuning. The optuna library is a tool that can help us automate the process of hyperparameter tuning by using a technique called Bayesian optimization. Bayesian optimization is a method of search that is based on probabilistic models instead of heuristics like grid search or random search. This makes it more efficient at finding the

best values for hyperparameters. We will use the TSPsampler to sample different values for each hyperparameter and then use these samples to train our model. By doing this, we hope to improve the performance of our model for more accuracy.

Table 6.1: XGBoost Parameter Space of the Hyper Parameter Tuning Process

Hyper Parameter	**Alternatives**
max depth	3, 5
estimators	0, 500
learning rate	0.05, 0.5
subsample	0.4, 0.9
colsample bytree	0.4, 0.9
min child weight	0, 5
reg alpha	0, 6
reg lambda	0, 2

6.1.2 Multi-Task Learning

In this thesis, we will use the pytorch+ optuna library to hyperparameter tune all three of our multi-task learning models. For each model, we will use a different set of parameters, including the learning rate, number of neurons, dropout rate, and activation function. AS all model have different kind of architecture. We will also use different combinations of these parameters to try to improve model performance. To do this, we will use the TSPsampler which can Bayesian method to search for the best parameter values. This should help us to find the best performing model and achieve more accurate results. more details of search space of parameter presented in tables.

Table 6.2: MMOE Model Parameter Space of the Hyper Parameter Tuning Process

Hyper Parameter	**Alternatives**
Dropout	0.3, 0.4, 0.7
num experts	2, 4, 6
Activation Function	sigmoid, relu, leakyrelu
hidden neu expert	0, 40
num neurons expert	1, 90
num neurons expert2	5, 20
hidden tower neu	0, 10
optimizer	'Adam', 'RMSprop', 'SGD'
Learning Rate	1e-5, 1e-1

Table 6.3: Task Specific layer model Parameter Space of the Hyper Parameter Tuning Process

Hyper Parameter	**Alternatives**
Dropout	0.3, 0.4, 0.7
hidden layer size	10, 80
Activation Function	sigmoid, relu, leakyrelu
task layesr1	1, 10
task layesr2	1, 10
optimizer	'Adam', 'RMSprop', 'SGD'
Learning Rate	1e-5, 1e-1

Table 6.4: MTL NNMT-2 Parameter Space of the Hyper Parameter Tuning Process

Hyper Parameter	**Alternatives**
Dropout	0.3, 0.4, 0.7
Activation Function	sigmoid, relu, leakyrelu
hidden layer1	1, 224
hidden layer2	1, 160
hidden layer3	1, 36
optimizer	'Adam', 'RMSprop', 'SGD'
Learning Rate	1e-5, 1e-1

6.2 Result Analysis

The results of the analysis presented in this thesis are very exciting. The synthetic data generation method presented in section 5.5.1 allows for the differentiation of data sets with different sizes and feature sizes. This makes it possible to analyze the impact of task correlation on different models' learning abilities. The results show that the task correlation has a significant impact on the learning ability of different models, and that the higher the task correlation, the better the learning ability of the model.

6.2.1 Synthetic data Analysis

In this analysis, we generate synthetic data sets with different sizes of features (5, 10, 20, 50, 100) and data points (100, 1000, 5000, 10000, 20000). We set the output task correlation to, 0.3, 0.5, and 0.9 and do an analysis of how different models learn from these data sets. The performance of single-task learning and multi-task learning model with a different set of hyperparameters as mentioned in 6.1.1 and 6.1.2 used for the trained model and MSE value is calculated to compare 'learning ability' and presented in table 6.4.

We performed a differently set of analyses on synthetics data like feature size, data point, and target task correlation. All MTL model training with 200 or more than 200 epochs to perform this analysis. impact of task correlation can be easily identified by the shown figure 6.1, 6.2, and 6.3 of all three models. then also plot compares the models with a learning ability with task correlation.

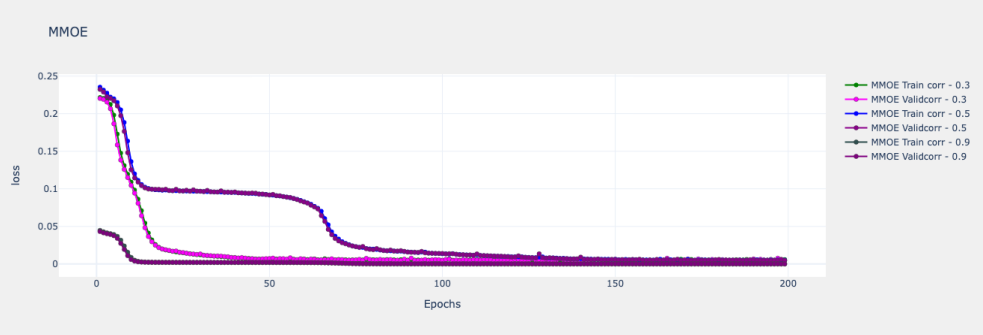

Figure 6.1: MMOE task correlation plot with 50 features and 10000 data points

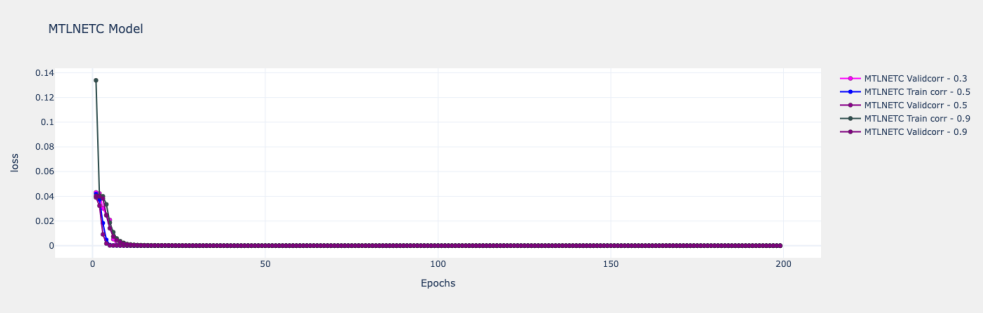

Figure 6.2: MTLNETC task correlation plot with 50 features and 10000 data points

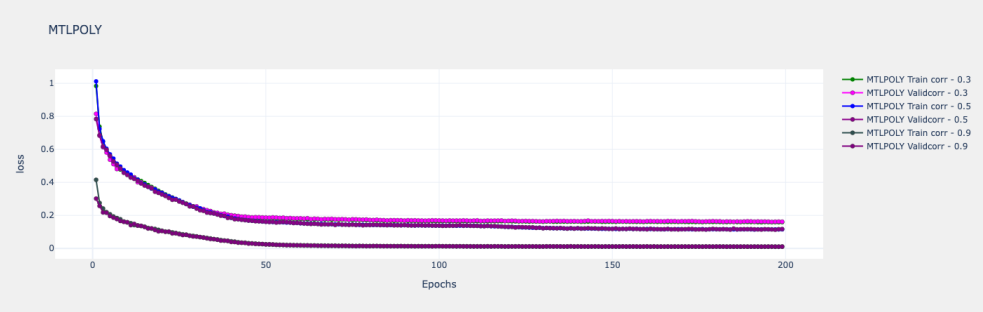

Figure 6.3: MTLpoly task correlation plot with 50 features and 10000 data points

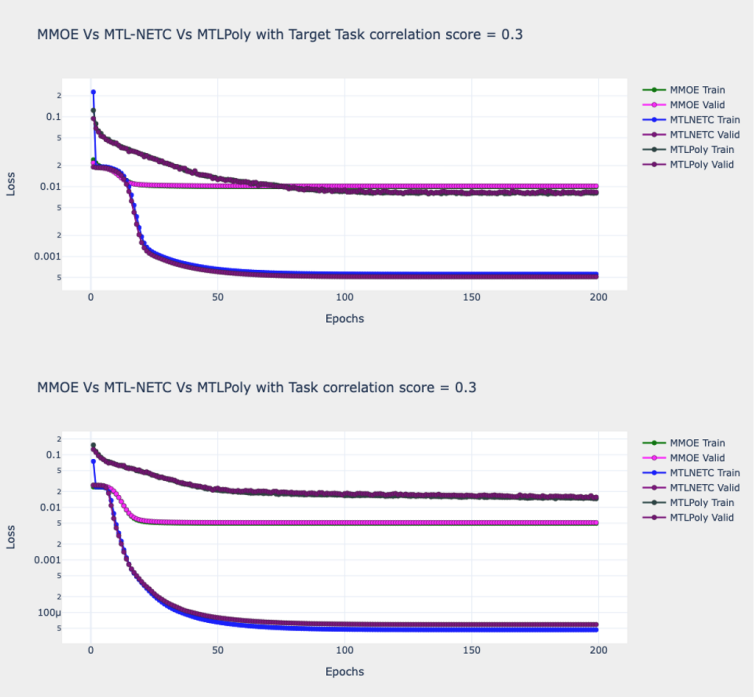

Figure 6.4: All Three Model with 50 features and 10000 data points and a task correlation of 0.3 1) Above Task A and 2) Below Task B

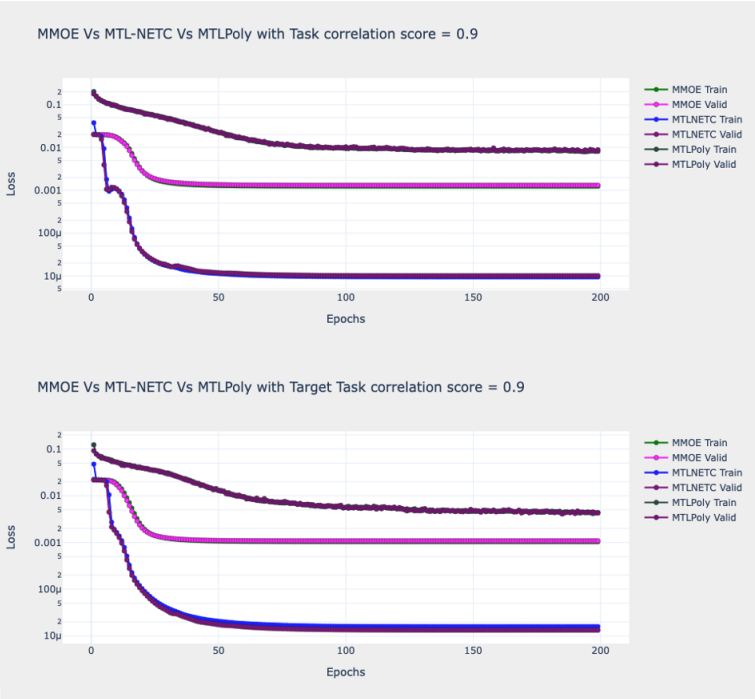

Figure 6.5: All Three Model with 50 features and 10000 data points and a task correlation of 0.9 1) Above Task A and 2) Below Task B

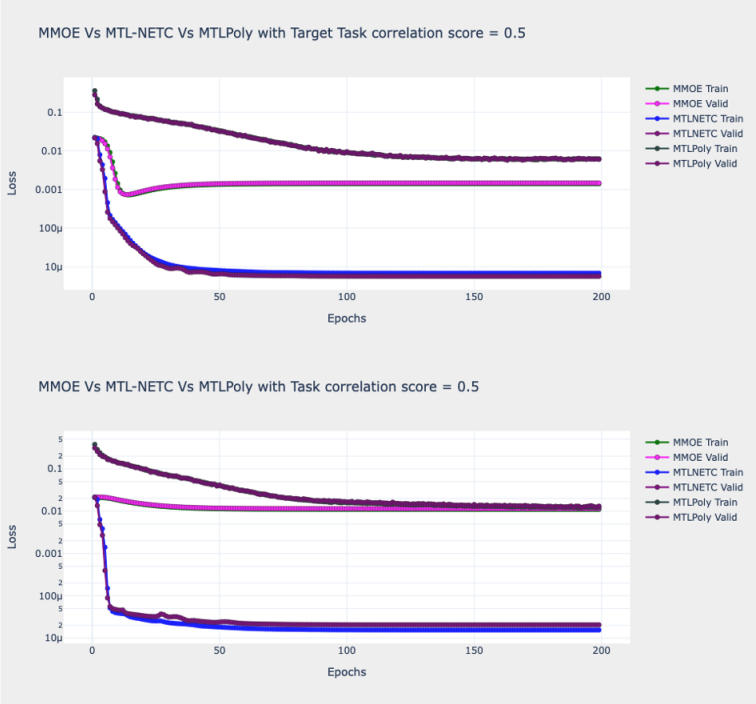

Figure 6.6: All Three Model with 50 features and 10000 data points and a task correlation of 0.5 1) Above Task A and 2) Below Task B

Result Analysis With Sparse Data With Correlation Value = 0.8

	MMOE Model					MTLNET -C					MTLPoly					XGBoost				
Dimension	5	10	20	50	100	5	10	20	50	100	5	10	20	50	100	5	10	20	50	100
Datapoint																				
100	0.003, 0.001	0.002, 0.002	0.008, 0.009			0.008, 0.012	0.002, 0.017	0.002, 0.002			0.256, 0.221	0.224, 0.267				0.0017, 0.0036	0.0032, 0.0046	0.0094, 0.0162		
1000	0.005, 0.006	0.004, 0.003	0.006, 0.003			0.005, 0.012	0.010, 0.002	0.002, 0.002			0.181, 0.176	0.164, 0.185	0.220, 0.124			0.0003, 0.0002	0.0023, 0.0013	0.0029, 0.0056	0.0055, 0.0078	
5000	0.006, 0.004	0.005, 0.006	0.0008, 0.0007	0.180, 0.213		0.001, 0.003	0.002, 0.002	0.002, 0.001			0.236, 0.104	0.240 0.232	0.286, 0.205	0.181, 0.213		0.0001, 0.0001	0.0003, 0.0005	0.0010, 0.0019	0.0041, 0.0037	0.0047, 0.0036
10000	0.004, 0.002	0.0008, 0.0008	0.0008, 0.0007	0.063, 0.261	0.257, 0.001	0.001, 0.002	0.001, 0.001	0.002, 0.001	0.004, 0.003	0.210, 0.221	0.232, 0.246	0.241, 0.147	0.244, 0.257	0.081, 0.264	0.0001, 0.0001	0.0006, 0.0005	0.0007, 0.0008	0.0023, 0.0018	0.0039, 0.0039	
20000	0.005, 0.002	0.0007, 0.0006	0.0006, 0.0007	0.0005, 0.0006	0.0002, 0.0003	0.005, 0.002	0.0007, 0.0006	0.0006, 0.0007	0.0005, 0.0006	0.210, 0.221	0.240 0.232	0.241, 0.147	0.286, 0.205	0.286, 0.205	0.0001, 0.0001	0.0001, 0.0003	0.0006, 0.0005	0.0018, 0.0015	0.0025, 0.0026	

Result Analysis With Sparse Data With Correlation Value = 0.5

	MMOE Model					MTLNET -C					MTLPoly					XGBoost				
Dimension	5	10	20	50	100	5	10	20	50	100	5	10	20	50	100	5	10	20	50	100
Datapoint																				
100	0.004, 0.002	0.002, 0.017				0.009, 0.011					0.234, 0.253	0.224, 0.234				0.0033, 0.0012	0.0056, 0.0095	0.0269, 0.0147		
1000	0.003, 0.005	0.012, 0.008	0.008, 0.005			0.004, 0.008	0.005, 0.011				0.172, 0.175	0.193, 0.153	0.180, 0.167			0.0001, 0.0001	0.0017, 0.0017	0.0024, 0.0024	0.0097, 0.0060	
5000	0.106, 0.002	0.005, 0.008	0.002, 0.003	0.180, 0.205		0.008, 0.002	0.002, 0.001	0.002, 0.002			0.261, 0.129	0.086, 0.305	0.220, 0.121	0.181 0.206		0.0001, 0.0001	0.0005, 0.0004	0.0018, 0.0018	0.0037, 0.0033	0.0043, 0.0057
10000	0.005, 0.004	0.0008, 0.0008	0.0003, 0.0006	0.0003, 0.0006		0.001, 0.002	0.002, 0.002	0.0007, 0.0008	0.0010, 0.0009	0.164, 0.187	0.232, 0.246	0.241, 0.153	0.286, 0.275	0.206, 0.205	0.0001, 0.0001	0.0004, 0.0003	0.0010, 0.0011	0.0022, 0.0024	0.0034, 0.0030	
20000	0.005, 0.004		0.0006, 0.0007	0.0005, 0.0005																

Result Analysis With Sparse Data With Correlation Value = 0.3

	MMOE Model					MTLNET -C					MTLPoly					XGBoost				
Dimension	5	10	20	50	100	5	10	20	50	100	5	10	20	50	100	5	10	20	50	100
Datapoint																				
100	0.003, 0.008	0.002, 0.006				0.008, 0.008					0.265, 0.280	0.301, 0.280				0.0084, 0.0041	0.0042, 0.0043	0.0291, 0.0211		
1000	0.001, 0.052	0.004, 0.006	0.011, 0.008			0.005, 0.007	0.010, 0.003				0.170, 0.173	0.163, 0.182	0.180, 0.179			0.0004, 0.0003	0.0008, 0.0015	0.0042, 0.0037	0.0079, 0.0045	
5000	0.005, 0.003	0.004, 0.001	0.004, 0.053	0.180, 0.201		0.009, 0.003	0.001, 0.001	0.002, 0.001			0.165, 0.174	0.081, 0.290	0.243, 0.150	0.181, 0.201	0.181, 0.201	0.0001, 0.0001	0.0006, 0.0006	0.0007, 0.0017	0.0035, 0.0031	0.0053, 0.0042
10000	0.0007, 0.0005	0.0002, 0.0003	0.0002, 0.0003	0.0007, 0.0008	0.0001, 0.0004	0.0005, 0.0002	0.0007, 0.0002	0.0007, 0.0002	0.0005, 0.0007	0.138, 0.177	0.171, 0.262	0.165, 0.174	0.165, 0.174	0.163, 0.182	0.0001, 0.0001	0.0004, 0.0003	0.0008, 0.0003	0.0013, 0.0006	0.0020, 0.0018	0.0046, 0.0041
20000	0.0004, 0.0003	0.0007, 0.0005	0.0002, 0.0003	0.0007, 0.0008	0.0002, 0.0008	0.004, 0.053	0.0005, 0.0002	0.0008, 0.0002	0.0005, 0.0007	0.165, 0.174	0.165, 0.174	0.165, 0.174	0.181, 0.201	0.171, 0.262	0.0001, 0.0001	0.0002, 0.0002	0.0004, 0.0008	0.0004, 0.0012	0.0014, 0.0012	0.0031, 0.0022

Figure 6.7: All Three Model Vs XGboost on Synthetic Sparse Data

		Result Analysis Without Sparse Data with Correlation Value = 0.8																													
		MMOE Model					MTLNET -C					MTLPoly					XGBoost													
Dimension		5	10	20	50	100	10	20	50	100	5	10	20	50	100	5	10	20	50	100	5	10	20	50	100						
Datapoint																															
	100	0.003, 0.001	0.002, 0.002				0.002, 0.002				0.181, 0.280	0.301, 0.280				0.0133, 0.0058	0.0234, 0.0180	0.0329, 0.0193													
	1000	0.005, 0.006	0.004, 0.003	0.006, 0.003			0.004, 0.003	0.006, 0.003			0.162, 0.173	0.163, 0.182	0.180, 0.179			0.0009, 0.0005	0.0024, 0.0047	0.0048, 0.0058	0.0114, 0.0090												
	5000	0.006, 0.004	0.005, 0.006	0.067, 0.008	0.180, 0.213		0.005, 0.006	0.067, 0.008	0.180, 0.213		0.165, 0.174	0.081, 0.290	0.243, 0.150	0.181, 0.201	0.181, 0.201	0.0005, 0.0003	0.0010, 0.0008	0.0016, 0.0025	0.0046, 0.0056	0.0085, 0.0080											
	10000	0.004, 0.002	0.0008, 0.0008	0.0008, 0.136	0.063, 0.261	0.257, 0.001	0.0008, 0.0008	0.0008, 0.136	0.063, 0.261	0.257, 0.001	0.138, 0.164	0.171, 0.202	0.165, 0.174	0.163, 0.182		0.0001, 0.0002	0.0011, 0.0007	0.0018, 0.0019	0.0047, 0.0038	0.0062, 0.0062											
	20000	0.0004, 0.0003	0.0006, 0.0005	0.0001, 0.0001	0.0002, 0.0003	0.0002, 0.0008	0.004, 0.053	0.0005, 0.0002	0.0002, 0.0001	0.0005, 0.0007						0.0001, 0.0001	0.0007, 0.0006	0.0010, 0.0012	0.0034, 0.0036	0.0046, 0.0045											
		Result Analysis Without Sparse Data with Correlation Value = 0.5																													
		MMOE Model					MTLNET -C					MTLPoly					XGBoost													
Dimension		5	10	20	50	100	10	20	50	100	5	10	20	50	100	5	10	20	50	100	5	10	20	50	100						
Datapoint																															
	100	0.003, 0.001	0.002, 0.002				0.002, 0.002				0.255, 0.280	0.301, 0.280				0.0049, 0.0030	0.0119, 0.0168	0.0296, 0.0237													
	1000	0.005, 0.006	0.004, 0.003	0.0007, 0.0003			0.004, 0.003	0.006, 0.003			0.170, 0.173	0.163, 0.182	0.180, 0.179			0.0004, 0.0010	0.0043, 0.0030	0.0041, 0.0054	0.0143, 0.0110												
	5000	0.006, 0.004	0.005, 0.006	0.0005, 0.0006	0.0004, 0.0003		0.005, 0.006	0.067, 0.008	0.180, 0.213		0.165, 0.174	0.081, 0.290	0.243, 0.150	0.181, 0.201	0.181, 0.201	0.0004, 0.0004	0.0010, 0.0013	0.0024, 0.0030	0.0041, 0.0053	0.0083, 0.0087											
	10000	0.004, 0.002	0.0008, 0.0008	0.0004, 0.0003	0.0007, 0.0003	0.0010, 0.0009	0.0008, 0.0008	0.0008, 0.136	0.063, 0.261	0.257, 0.001	0.138, 0.177	0.171, 0.262	0.165, 0.174	0.163, 0.182		0.0003, 0.0002	0.0004, 0.0007	0.0018, 0.0025	0.0038, 0.0045	0.0065, 0.0073											
	20000	0.0001, 0.0001	0.0002, 0.0002	0.0004, 0.0003	0.0001, 0.0003	0.0004, 0.0003										0.0001, 0.0001	0.0007, 0.0004	0.0012, 0.0010	0.0028, 0.0032	0.0057, 0.0054											
		Result Analysis Without Sparse Data with Correlation Value = 0.3																													
		MMOE Model					MTLNET -C					MTLPoly					XGBoost													
Dimension		5	10	20	50	100	10	20	50	100	5	10	20	50	100	5	10	20	50	100	5	10	20	50	100						
Datapoint																															
	100	0.003, 0.001	0.002, 0.002				0.002, 0.002				0.245, 0.280	0.301, 0.280				0.0038, 0.0064	0.0050, 0.0117	0.0564, 0.0123													
	1000	0.005, 0.006	0.004, 0.003	0.006, 0.003			0.004, 0.003	0.006, 0.003			0.170, 0.171	0.163, 0.182	0.180, 0.179			0.0015, 0.0027	0.0044, 0.0030	0.0070, 0.0055	0.0152, 0.0111												
	5000	0.006, 0.004	0.005, 0.006	0.067, 0.008	0.180, 0.213		0.005, 0.006	0.067, 0.008	0.180, 0.213		0.164, 0.174	0.082, 0.290	0.243, 0.159	0.181, 0.205	0.181, 0.201	0.0007, 0.0004	0.0007, 0.0014	0.0027, 0.0019	0.0056, 0.0055	0.0088, 0.0083											
	10000	0.004, 0.002	0.0008, 0.0008	0.0008, 0.136	0.063, 0.261	0.0021, 0.0008	0.0008, 0.136	0.0008, 0.136	0.063, 0.261	0.257, 0.001	0.138, 0.177	0.171, 0.262	0.165, 0.174	0.165, 0.174	0.163, 0.182	0.0001, 0.0002	0.0007, 0.0006	0.0018, 0.0023	0.0043, 0.0039	0.0055, 0.0053											
	20000	0.004, 0.002	0.0008, 0.0008	0.0008, 0.0010	0.063, 0.261	0.0008, 0.0010	0.004, 0.002	0.0008, 0.0008	0.0008, 0.0010	0.063, 0.261	0.165, 0.174	0.165, 0.174	0.165, 0.174	0.181, 0.201	0.171, 0.262	0.0002, 0.0001	0.0006, 0.0005	0.0012, 0.0009	0.0024, 0.0024	0.0057, 0.0044											

Figure 6.8: All Three Model Vs XGboost on Synthetic NonSparse Data

- Task correlation creates an impact on model learning ability. A higher correlation leads to a significantly better performance than with a lower correlation of the multi-output objective.

- MMOE and MTLNETC have better learning ability than the simple feed-forward network model. More complex architecture has better generalization of data than a simple one.
- Data points and the feature size are very important in the MTL model. according to the analysis present in Figure 6.7 and 6.8, fewer data with more feature has over-fitting and more data point with fewer feature size have an under-fitting issue in the model.

6.2.2 Real data Analysis

Table 6.5: Result of Guo-2019 data sets on 500 epochs

	Target label	MSE Train	MSE Valid	MSE Test
XGBOOST	Yield Strength	0.0008	0.0008	0.0010
	Tensile Strength	0.0001	0.0001	0.0001
	Elongation	0.0017	0.0017	0.0018
MMOE	Yield Strength	0.0023	0.0024	0.0026
	Tensile Strength	0.0002	0.0001	0.0001
	Elongation	0.0034	0.0039	0.0039
MTLNETC	Yield Strength	0.0013	0.0009	0.0018
	Tensile Strength	0.0006	0.0004	0.0009
	Elongation	0.0013	0.0004	0.0009
MTLPoly	Yield Strength	0.0925	0.0908	0.0903
	Tensile Strength	0.0931	0.0924	0.0924
	Elongation	0.0097	0.0924	0.0095

Findings of four distinct machine learning models utilizing 500 epochs on three target output —yield strength, tensile strength, and elongation—are shown in the table above. The models being compared are MTL model MTLPoly, MTLNETC, MMOE, and one STL XGboost Algorithm. The training and validation sets' mean squared errors (MSE) for each model and dataset are displayed in table 6.5.

The XGBOOST method, which has the lowest MSE throughout the train, validation, and test sets, performs the best on all three target labels, as shown in the table. Compared to XGBOOST, the MSE of the MMOE method is marginally higher on the train, validation, and test sets, but MTLNETC and MTLPoly have a markedly higher MSE on the train, validation, and test sets.

XGBoost and MTLPoly, on the other hand, underperformed the other models. With 0.0017 on the Yield Strength dataset, 0.007 on the Tensile Strength dataset, and 0.0033 on the Elongation dataset, XGBoost had the highest MSE on the validation sets. With an MSE of 0.0909 on the Yield Strength dataset, 0.0918 on the Tensile Strength dataset, and 0.0918 on the Elongation dataset, MTLPoly likewise did poorly.

Although the MSE on the train sets is generally low for all methods MTL and STL, it is greater for the validation and test sets, suggesting that the models may have overfitted the train set. It may be because the models don't use enough regularization or because the dataset isn't very large.

The MTLPoly method performs poorly on this data set since it has high MSE on all three target labels. This can be the result of bad feature selection, flawed model design, or insufficient data. One might experiment with a more complicated model architecture, a larger dataset, or feature selection techniques to boost performance.

In conclusion, the Guo-2019 data set exhibits the highest performance for the XGBOOST method and the worst performance for the MTLPoly technique. It is crucial to take into account methods to avoid overfitting and guarantee excellent generalization on fresh data in order to enhance performance.

Table 6.6: Result of Hu2021 data sets on 1000 epochs

Algorithm	Target label	MSE Train	MSE Valid	MSE Test
XGBOOST	UTS	0.0014	0.0031	0.0030
	YTS	0.0026	0.0071	0.0070
	Elongation	0.0050	0.0122	0.0108
MMOE	UTS	0.0009	0.0057	0.0051
	YTS	0.0009	0.0086	0.0091
	Elongation	0.0007	0.0145	0.0089
MTLNETC	UTS	0.0029	0.0068	0.0078
	YTS	0.0036	0.0117	0.0117
	Elongation	0.0085	0.0117	0.0117
MTLPoly	UTS	0.0141	0.0152	0.0182
	YTS	0.0148	0.0165	0.0194
	Elongation	0.0409	0.0165	0.0379

The results of four distinct machine learning models on three datasets—UTS (Ultimate Tensile Strength), YTS (Yield Tensile Strength), and Elongation—across 1000 epochs are shown in the following table. The models being compared are MTLPoly, XGBoost, MMOE, and MTLNETC. The training , validation and test sets' mean squared errors (MSE) for each model and dataset are displayed in table 6.6.

From table 6.6 MSE values of 0.0014 and 0.0026 on the train set and 0.0030 and 0.0070 on the test set, respectively, the XGBoost algorithm performs well on the UTS and YTS target labels. With an MSE of 0.0108 on the test set, it performs really poorly on the Elongation target label.

The MMOE algorithm consistently outperforms all three target labels, with MSE values on

the train set ranging from 0.0007 to 0.0057 and on the test set from 0.0051 to 0.0091.Similar to the XGBoost method, the MTLNETC algorithm performs well on the test set, with somewhat higher MSE values for the UTS and YTS labels, but a substantially higher MSE value of 0.0117 on the Elongation label. The MSE values for the MTLPoly algorithm range from 0.0141 to 0.0412 on the train set and from 0.0182 to 0.0379 on the test set, indicating poor performance across all target labels.

It is essential to remember that MSE values on the test set are often higher than those on the train set. This suggests that overfitting, which occurs when a model performs well on training data but does not generalize well to new data, is likely to take place. Another possibility is that the model is underfitting, or that it is too simple to detect the underlying patterns in the data. Therefore, to avoid overfitting and underfitting and to make sure that the model generalizes effectively to new data, it is necessary to apply techniques like cross-validation.

Table 6.7: Result of Huang2021 data sets on 1000 epochs

Algorithm	Target label	MSE Train	MSE Valid	MSE Test
XGBOOST	Compressive strength (CS)	0.0007	0.0018	0.0021
	Flexural strength (FS)	0.0033	0.0072	0.0029
MMOE	Compressive strength (CS)	0.0007	0.0050	0.0057
	Flexural strength (FS)	0.0016	0.0093	0.0024
MTLNETC	Compressive strength (CS)	0.0007	0.0077	0.0034
	Flexural strength (FS)	0.0039	0.0060	0.0076
MTLPoly	Compressive strength (CS)	0.0141	0.0182	0.0111
	Flexural strength (FS)	0.0183	0.0186	0.0124

The compressive strength (CS) and flexural strength (FS) of four distinct models (XGBOOST, MMOE, MTLNETC, and MTLPoly), as well as their mean squared errors (MSEs), are shown in the table using three separate data sets: train, validation, and test.

With the lowest MSE on the validation and test sets for both CS and FS, it appears that the XGBOOST model performs the best overall on both tasks. On the CS task, the MMOE model also does well, but less well on the FS task. With larger MSE values on the validation and test sets, the MTLNETC and MTLPoly models both perform worse than the other two models on both tasks.

It is important to note that while all of the models have very low MSE values on the train set, their MSE values are rather high on the validation and test sets, which raises the possibility of overfitting. Overfitting happens when a model is very complicated, fits the training data exceptionally well, but struggles to generalize to new, untried data. Monitoring the model's performance on a validation set and using methods like regularization are crucial for reducing overfitting.

It's also important to note that the validation and test sets for the MTLNETC and MTLPoly models have rather high MSE values, which may indicate that they underfit the data. When a model is too simple to recognize the underlying patterns in the data, underfitting occurs. In this situation, using more intricate models or making the present models more powerful may be beneficial.

In general, it's crucial to remember that a model's success on a test set does not always translate to how well it performs on fresh, untested data. When selecting and fine-tuning a model, it's crucial to assess its performance using a test set and take both overfitting and underfitting into account.

Table 6.8: Result of UCL-CBM data sets on 1000 epochs

Algorithm	Target label	MSE Train	MSE Valid	MSE Test
XGBOOST	Compresssor-dsc	0.0063	0.0063	0.0064
	Turbine-dsc	0.0095	0.0109	0.0111
MMOE	Compresssor-dsc	0.0005	0.0007	0.0007
	Turbine-dsc	0.0005	0.0006	0.0007
MTLNETC	Compresssor-dsc	0.0001	0.0001	0.0001
	Turbine-dsc	0.0004	0.0009	0.0009
MTLPoly	Compresssor-dsc	0.0446	0.0448	0.0432
	Turbine-dsc	0.0448	0.0144	0.0434

The evaluations of various machine learning models applied to the UCL-CBM data sets are shown in the table 6.8 above. For the two tasks of Compressor-dsc and Turbine-dsc, the training and validation performance of the models XGBOOST, MMOE, MTLNETC, and MTLPoly is quantified in terms of mean squared error (MSE).

The MTLNETC model had the lowest MSE values on all three sets for both outputs, which indicates from the findings that it performed the best on this data set. Despite having somewhat higher MSE values than MTLNETC, the MMOE model also performed admirably. The XGBoost and MTLPoly models, on the other hand, did not perform as well, with noticeably higher MSE values on all three sets.

It is significant to observe that all models had reasonably low MSE values for the training sets, indicating that all models were successful in accurately fitting the data. The validation and test sets' MSE values, on the other hand, are higher, suggesting that the models might not be generalizing effectively to fresh data. This may be the result of overfitting, in which the models learnt the noise in the training data rather than the underlying patterns. The use of methods like regularization or early halting can help solve this widespread issue in machine learning. Additionally, the MTLPoly model did very poorly; this might be because

it wasn't the best fit model or because there wasn't appropriate hyperparameter tuning.

Seeing as that it obtained the lowest MSE values among all three sets for both outputs, the MTLNETC model appears to be the most effective one on the UCL-CBM data set. Further research should be done since it's possible that the models are overfitting to the training data and not generalizing effectively to fresh data.

Table 6.9: Result of Xiong 2014 data sets on 1000 epochs

Algorithm	Target label	MSE Train	MSE Valid	MSE Test
XGBOOST	W-output(mm)	0.0008	0.0467	0.0052
	H-output(mm)	0.0007	0.0425	0.0026
MMOE	W-output(mm)	0.0008	0.0019	0.0021
	H-output(mm)	0.0007	0.0114	0.0006
MTLNETC	W-output(mm)	0.0002	0.0045	0.0015
	H-output(mm)	0.0003	0.0122	0.0013
MTLPoly	W-output(mm)	0.0101	0.0168	0.0019
	H-output(mm)	0.0101	0.0138	0.0008

Results from four different machine learning models on the Xiong 2014 data set are shown in table 6.9 above. Each model was trained for 1000 epochs. The models are MTLPoly, XGBoost, MMOE, and MTLNETC. The data set contains two output variables, W-output (mm) and H-output (mm), and the mean squared error (MSE) for the train, validation, and test sets is used to assess the performance.

All of the models fit the training set's data well, as seen by the findings, which demonstrate that they all have low MSEs. The performance of the models varies, nevertheless, depending on the validation and test sets. For both W-output and H-output, the MMOE model has the lowest MSE on the validation and test sets, demonstrating that it generalizes well to new data. However, for both W-output and H-output, the XGBoost model has the highest MSE on the validation and test sets, suggesting that it may be overfitting the training data.

In comparison to the MMOE model, the MTLNETC and MTLPoly models perform similarly, with a little higher MSE on the validation and test sets.

The MMOE model, which exhibits high generalization to new data while avoiding overfitting, seems to be the best performer among the four models in conclusion. It's important to keep in mind, though, that the results might potentially be influenced by the sample size and data dispersion. It's feasible that different outcomes may be obtained with a bigger sample size and a more varied set of data.

Table 6.10: Result of Yin 20214 data sets on 1000 epochs

Algorithm	Target label	MSE Train	MSE Valid	MSE Test
XGBOOST	Fmax(N)-output	0.0005	0.0014	0.0014
	IFSS(Mpa)-output	0.0005	0.0012	0.0007
MMOE	Fmax(N)-output	0.0010	0.0010	0.0013
	IFSS(Mpa)-output	0.0012	0.0019	0.0013
MTLNETC	Fmax(N)-output	0.0007	0.0008	0.0013
	IFSS(Mpa)-output	0.0007	0.0007	0.0007
MTLPoly	Fmax(N)-output	0.0700	0.0673	0.0581
	IFSS(Mpa)-output	0.0696	0.0658	0.0558

The performance of four distinct models—XGBOOST, MMOE, MTLNETC, and MTLPoly—on the Yin 2014 data sets is compared in table 6.10 above. The mean squared error (MSE) on the train, validation and test sets is used to evaluate the models.

The Fmax(N)-output and IFSS(Mpa)-output tasks showed the lowest MSE for the XGBOOST model, which was the model that performed the best overall. The MMOE model performed similarly to the XGBOOST model on the test set, but had a little higher MSE on the train and validation sets. Similar MSE on the train, validation, and test sets indicated that the MTLNETC model likewise performed well.

The MTLPoly model, on the other hand, had considerably higher MSE on all three sets, showing that it did not do as well as the other models on this task.

It's vital to remember that in certain situations, both overfitting and underfitting can happen. When a model performs badly on untrained data but too well on training data, this is known as overfitting. This may be brought either by the model having too many parameters or by their not being sufficient data to train the model. Underfitting, on the other hand, happens when a model is not trained sufficiently on the training data and hence performs poorly on unobserved data.

A successful model should, in general, be able to generalize well, which means that it can perform well with previously unexplored data. The model that performs well on both the training set and the test set is the best one.

According to the findings in the table, the XGBOOST model appears to have high generalization ability because it has the lowest MSE on both the train set and the test set, making it a suitable model for the Yin 2014 data sets. Due to its identical MSE on the train and test sets, the MMOE model offers high generalization capabilities as well. Since the MTLNETC model's MSE on the train set and test set are comparable, it also has strong generalization.

7 Conclusion

7.1 Summary

In the conclusion, it is clear that the synthetic data sets outperform the real-world data sets in terms of both label quality and distribution. This is likely because the synthetic data sets are generated from a known underlying process, which results in perfect labels and a better distribution. The performance of multi-task learning versus single-task learning is also an important factor to consider. In general, multi-task learning outperforms single-task learning because it can learn from multiple tasks simultaneously. According to the analysis, one thing to consider in the multi-output problem. First, if multi-task output shares the same feature set then it is worth applying multi-task learning approaches even if the multi-target output has a positive and negative correlation. Second multi-task learning performance is better with small data in compression to single-task learning according to results of Xiong 2014 table 6.9.

The benefits of multi-task learning have been demonstrated on both synthetic and real-world data sets. In general, the more complex the model design, the better the performance of the multi-task learning model. However, some real-world data sets(e.g results of data sets Yin 2014(Table 6.10) and Guo 2019(Table 6.5)) have been found to be better suited to a single-task model than a more complex Multi-task model.

Multi-task learning and single-task learning were evaluated on several benchmark datasets, showing that they outperform the state of the art in all/most. cases. Furthermore, it was shown that these methods are particularly effective when only a small amount of training data is available. This makes them well suited for real-world applications where data is often scarce.

7.2 Future Work

The current state-of-the-art in machine learning is to build models that are specific to a single task. However, there is a growing body of work on multi-task learning, which seeks to build models that can learn multiple tasks simultaneously. This is an important direction

of research, as it has the potential to improve the performance of machine learning models by leveraging knowledge from related tasks.

In this thesis, we focus on the development of multi-task learning models for tabular data. We use three different approaches for building such models. According to the review, very few multi-task learning approaches design tabular data sets. The future aim is to build and explore more multi-task learning architectures for tabular data sets. And also possible to explore two different approaches. The first is the data quality approach used in standard deep learning models, with data augmentation and transfer learning. Our second approach uses a more advanced regularization technique, which we refer to as "stochastic depth".

References

[1] Timo Aho et al. “Multi-target regression with rule ensembles.” In: *Journal of Machine Learning Research* 13.8 (2012).

[2] Takuya Akiba et al. “Optuna: A next-generation hyperparameter optimization framework”. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[3] Md Zahangir Alom et al. “A state-of-the-art survey on deep learning theory and architectures”. In: *Electronics* 8.3 (2019), p. 292.

[4] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. “Convex multi-task feature learning”. In: *Machine learning* 73.3 (2008), pp. 243–272.

[5] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. “Multi-task feature learning”. In: *Advances in neural information processing systems* 19 (2006).

[6] Sercan Ö Arik and Tomas Pfister. “Tabnet: Attentive interpretable tabular learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 8. 2021, pp. 6679–6687.

[7] BJ Bakker and TM Heskes. “Task clustering and gating for bayesian multitask learning”. In: (2003).

[8] Jonathan Baxter. “A Bayesian/information theoretic model of learning to learn via multiple task sampling”. In: *Machine learning* 28.1 (1997), pp. 7–39.

[9] Vasileios Belagiannis et al. “Robust optimization for deep regression”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2830–2838.

[10] Yoshua Bengio et al. “Curriculum learning”. In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48.

[11] Dimitri P Bertsekas, W Hager, and O Mangasarian. “Nonlinear programming. athena scientific belmont”. In: *Massachusets, USA* (1999).

[12] Steffen Bickel et al. “Multi-task learning for HIV therapy screening”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 56–63.

[13] Concha Bielza, Guangdi Li, and Pedro Larranaga. “Multi-dimensional classification with Bayesian networks”. In: *International Journal of Approximate Reasoning* 52.6 (2011), pp. 705–727.

[14] Edwin V Bonilla, Kian Chai, and Christopher Williams. “Multi-task Gaussian process prediction”. In: *Advances in neural information processing systems* 20 (2007).

[15] Léon Bottou. “Stochastic gradient descent tricks”. In: *Neural Networks: Tricks of the Trade: Second Edition* (2012), pp. 421–436.

[16] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. “Model compression”. In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006, pp. 535–541.

[17] Shaofeng Cai et al. “Arm-net: Adaptive relation modeling network for structured data”. In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 207–220.

[18] Rich Caruana. “Multitask learning”. In: *Machine learning* 28.1 (1997), pp. 41–75.

[19] Rich Caruana and Joseph O’Sullivan. “Multitask Pattern Recognition for Vision-Based Autonomous Robots”. In: *International Conference on Artificial Neural Networks*. Springer. 1998, pp. 1115–1120.

[20] Jianhui Chen et al. “A convex formulation for learning shared structures from multiple tasks”. In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 137–144.

[21] Tianqi Chen and Carlos Guestrin. “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[22] Ronan Collobert and Jason Weston. “A unified architecture for natural language processing: Deep neural networks with multitask learning”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 160–167.

[23] Andrew R Conn, Katya Scheinberg, and Luis N Vicente. *Introduction to derivative-free optimization*. SIAM, 2009.

[24] Andrea Coraddu et al. “Machine learning approaches for improving condition-based maintenance of naval propulsion plants”. In: *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment* 230.1 (2016), pp. 136–153.

[25] Li Deng, Geoffrey Hinton, and Brian Kingsbury. “New types of deep neural network learning for speech recognition and related applications: An overview”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 8599–8603.

[26] Jean-Antoine Désidéri. “Multiple-gradient descent algorithm (MGDA) for multiobjective optimization”. In: *Comptes Rendus Mathematique* 350.5-6 (2012), pp. 313–318.

[27] Long Duong et al. “Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser”. In: *Proceedings of the 53rd annual meeting of the Association for Computational Linguistics and the 7th international joint conference on natural language processing (volume 2: short papers)*. 2015, pp. 845–850.

[28] Theodoros Evgeniou and Massimiliano Pontil. “Regularized multi–task learning”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 109–117.

[29] Kunihiko Fukushima. “Neocognitron: A hierarchical neural network capable of visual pattern recognition”. In: *Neural networks* 1.2 (1988), pp. 119–130.

[30] Felix A Gers and Jürgen Schmidhuber. “Recurrent nets that time and count”. In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. Vol. 3. IEEE. 2000, pp. 189–194.

[31] Rania M Ghoniem. “A novel bio-inspired deep learning approach for liver cancer diagnosis”. In: *Information* 11.2 (2020), p. 80.

[32] Ross Girshick. “Fast r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[33] Ting Gong et al. “A comparison of loss weighting strategies for multi task learning in deep neural networks”. In: *IEEE Access* 7 (2019), pp. 141627–141632.

[34] Klaus Greff et al. “LSTM: A search space odyssey”. In: *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2222–2232.

[35] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. “Why do tree-based models still outperform deep learning on typical tabular data?” In: *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2022.

[36] Shun Guo et al. “A predicting model for properties of steel using the industrial big data based on machine learning”. In: *Computational Materials Science* 160 (2019), pp. 95–104.

[37] Patrick T Harker and Jong-Shi Pang. “A damped-Newton method for the linear complementarity problem”. In: *Lectures in Applied Mathematics* 26 (1990), pp. 265–284.

[38] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[39] Daniel Hernández-Lobato and José Miguel Hernández-Lobato. “Learning feature selection dependencies in multi-task learning”. In: *Advances in Neural Information Processing Systems* 26 (2013).

[40] Daniel Hernández-Lobato, José Miguel Hernández-Lobato, and Zoubin Ghahramani. “A probabilistic model for dirty multi-task feature selection”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1073–1082.

[41] Sepp Hochreiter et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 2001.

[42] JS Huang, JX Liew, and KM Liew. “Data-driven machine learning approach for exploring and assessing mechanical properties of carbon nanotube-reinforced cement composites”. In: *Composite Structures* 267 (2021), p. 113917.

[43] Robert A Jacobs et al. “Adaptive mixtures of local experts”. In: *Neural computation* 3.1 (1991), pp. 79–87.

[44] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax”. In: *arXiv preprint arXiv:1611.01144* (2016).

[45] Zhuoliang Kang, Kristen Grauman, and Fei Sha. “Learning with whom to share in multi-task feature learning”. In: *ICML*. 2011.

[46] Andrej Karpathy and Li Fei-Fei. “Deep visual-semantic alignments for generating image descriptions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3128–3137.

[47] Dragi Kocev et al. “Using single-and multi-target regression trees and ensembles to model a compound index of vegetation condition”. In: *Ecological Modelling* 220.8 (2009), pp. 1159–1168.

[48] Jan Koutnik et al. “A clockwork rnn. arXiv”. In: *arXiv preprint arXiv:1402.3511* (2014).

[49] Christopher Kuenneth et al. “Polymer informatics with multi-task learning”. In: *Patterns* 2.4 (2021), p. 100238.

[50] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[51] Giwoong Lee, Eunho Yang, and Sung Hwang. “Asymmetric multi-task learning based on task relatedness and loss”. In: *International conference on machine learning*. PMLR. 2016, pp. 230–238.

[52] Hae Beom Lee, Eunho Yang, and Sung Ju Hwang. “Deep asymmetric multi-task feature learning”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2956–2964.

[53] Huan Liang, Wenlong Fu, and Fengji Yi. “A survey of recent advances in transfer learning”. In: *2019 IEEE 19th international conference on communication technology (ICCT)*. IEEE. 2019, pp. 1516–1523.

[54] Xiabing Liu et al. “3D head pose estimation with convolutional neural network trained on synthetic images”. In: *2016 IEEE international conference on image processing (ICIP)*. IEEE. 2016, pp. 1289–1293.

[55] Mingsheng Long et al. “Learning multiple tasks with multilinear relationship networks”. In: *Advances in neural information processing systems* 30 (2017).

[56] Jiaqi Ma et al. “Modeling task relationships in multi-task learning with multi-gate mixture-of-experts”. In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 1930–1939.

[57] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. “The concrete distribution: A continuous relaxation of discrete random variables”. In: *arXiv preprint arXiv:1611.00712* (2016).

[58] Gjorgji Madjarov et al. “An extensive experimental comparison of methods for multi-label learning”. In: *Pattern recognition* 45.9 (2012), pp. 3084–3104.

[59] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. “Attentive single-tasking of multiple tasks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 1851–1860.

[60] Xudong Mao et al. “Least squares generative adversarial networks”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2794–2802.

[61] Umberto Michelucci and Francesca Venturini. “Multi-task learning for multi-dimensional regression: Application to luminescence sensing”. In: *Applied Sciences* 9.22 (2019), p. 4748.

[62] Hu Mingwei et al. “Prediction of Mechanical Properties of Wrought Aluminium Alloys Using Feature Engineering Assisted Machine Learning Approach”. In: *Metallurgical and Materials Transactions* 52.7 (2021), pp. 2873–2884.

[63] Ishan Misra et al. “Cross-stitch networks for multi-task learning”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3994–4003.

[64] Vu Nguyen. “Bayesian optimization for accelerating hyper-parameter tuning”. In: *2019 IEEE second international conference on artificial intelligence and knowledge engineering (AIKE)*. IEEE. 2019, pp. 302–305.

[65] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[66] Ivan V Oseledets. “Tensor-train decomposition”. In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317.

[67] Shibin Parameswaran and Kilian Q Weinberger. “Large margin multi-task metric learning”. In: *Advances in neural information processing systems* 23 (2010).

[68] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[69] Paulo R Peloia and Luiz HA Rodrigues. “Identification of commercial blocks of outstanding performance of sugarcane using data mining”. In: *Engenharia Agrıcola* 36 (2016), pp. 895–901.

[70] Michael JD Powell. “A method for nonlinear constraints in minimization problems”. In: *Optimization* (1969), pp. 283–298.

[71] Bharath Ramsundar et al. “Massively multitask networks for drug discovery”. In: *arXiv preprint arXiv:1502.02072* (2015).

[72] Victor Sanh, Thomas Wolf, and Sebastian Ruder. “A hierarchical multi-task approach for learning embeddings from semantic tasks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 6949–6956.

[73] Iqbal H Sarker, Yoosef B Abushark, and Asif Irshad Khan. “Contextpca: Predicting context-aware smartphone apps usage based on machine learning techniques”. In: *Symmetry* 12.4 (2020), p. 499.

[74] Iqbal H Sarker et al. “Context pre-modeling: an empirical analysis for classification based user-centric context-aware predictive modeling”. In: *Journal of Big Data* 7.1 (2020), pp. 1–23.

[75] Iqbal H Sarker et al. “Intrudtree: a machine learning based cyber security intrusion detection model”. In: *Symmetry* 12.5 (2020), p. 754.

[76] Shengyang Sun, Changyou Chen, and Lawrence Carin. “Learning structured weight uncertainty in bayesian neural networks”. In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1283–1292.

[77] Shiliang Sun et al. “A survey of optimization methods from a machine learning perspective”. In: *IEEE transactions on cybernetics* 50.8 (2019), pp. 3668–3681.

[78] Yi Sun, Xiaogang Wang, and Xiaoou Tang. “Deep convolutional network cascade for facial point detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 3476–3483.

[79] Alexander Toshev and Christian Szegedy. “Deeppose: Human pose estimation via deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1653–1660.

[80] Grigorios Tsoumakas and Ioannis Katakis. “Multi-label classification: An overview”. In: *International Journal of Data Warehousing and Mining (IJDWM)* 3.3 (2007), pp. 1–13.

[81] Ledyard R Tucker. “Some mathematical notes on three-mode factor analysis”. In: *Psychometrika* 31.3 (1966), pp. 279–311.

[82] Luis Enrique Vivanco-Benavides et al. “Machine learning and materials informatics approaches in the analysis of physical properties of carbon nanotubes: A review”. In: *Computational Materials Science* 201 (2022), p. 110939.

[83] Ian H Witten and Eibe Frank. “Data mining: practical machine learning tools and techniques with Java implementations”. In: *Acm Sigmod Record* 31.1 (2002), pp. 76–77.

[84] Jun Xiong et al. “Bead geometry prediction for robotic GMAW-based rapid manufacturing through a neural network and a second-order regression analysis”. In: *Journal of Intelligent Manufacturing* 25 (2014), pp. 157–163.

[85] Ya Xue et al. “Multi-Task Learning for Classification with Dirichlet Process Priors.” In: *Journal of Machine Learning Research* 8.1 (2007).

[86] Yongxin Yang and Timothy M Hospedales. “Trace norm regularised deep multi-task learning”. In: *arXiv preprint arXiv:1606.04038* (2016).

[87] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).

[88] Min-Ling Zhang and Zhi-Hua Zhou. “A review on multi-label learning algorithms”. In: *IEEE transactions on knowledge and data engineering* 26.8 (2013), pp. 1819–1837.

[89] Yu Zhang and Dit–Yan Yeung. “Multi-task learning using generalized t process”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 964–971.

[90] Yu Zhang, Dit-Yan Yeung, and Qian Xu. “Probabilistic multi-task feature selection”. In: *Advances in neural information processing systems* 23 (2010).

Acknowledgements

I'd like to thank my supervisor Dipl.-Ing. Felix Conrad for his expertise, guidance, and patience during the course of this project. His tips have had a huge impact on the successful completion of this master's thesis and will surely continue to influence my future works in this field, as I try to implement learned concepts like critical analysis into my work. I would also like to thank Prof. Dr. -Ing Steffen Ihlenfeldt and Dr.-Ing. Hajo Wiemer for making me aware of the interesting topic and providing me with the opportunity to conduct this thesis at the Fakultät Maschinenwesen, Institut für Mechatronischen Maschinenbau at TU Dresden.