Unit 3

Optimization and Regularization

Optimization

- Deep learning relies on optimization methods.
- Training a complicated deep learning model, on the other hand, might take hours, days, or even weeks. The training efficiency of the model is directly in influenced by the optimization algorithm's performance.
- Understanding the fundamentals of different optimization algorithms and the function of their hyperparameters, on the other hand, will allow us to modify hyperparameters in a targeted manner to improve deep learning model performance.
- Although optimization may help deep learning by lowering the loss function, the aims of optimization and deep learning are fundamentally different. The former is more focused on minimizing an objective, whereas the latter is more concerned with finding a good model given a finite quantity of data.
- Training error and generalization error, for example, vary in that the optimization algorithm's objective function is usually a loss function based on the training dataset, and the purpose of optimization is to minimize training error.

Gradient Descent

- Consider that you are holding a ball that is lying on the rim of a bowl. When you lose the ball, it travels in the steepest direction until it reaches the bowl's bottom. A gradient directs the ball in the steepest way possible to the local minimum, which is the bowl's bottom.
- Gradient descent works with a set of coeffcients, calculates their cost, and looks for a cost value that is lower than the current one. It shifts to a lesser weight and updates the values of the coffcients. The procedure continues until the local minimum is found. A local minimum is a point beyond which it is impossible to go any farther.
- For the most part, gradient descent is the best option. It does, however, have significant drawbacks. Calculating the gradients is time-consuming when the data is large. For convex functions, gradient descent works well, but it doesn't know how far to travel down the gradient for nonconvex functions. Gradient descent works well for convex functions

Stochastic Gradient Descent Deep Learning Optimizer

- On large datasets, gradient descent may not be the best solution. We use stochastic gradient descent to solve the problem.
- The word stochastic refers to the algorithm's underlying unpredictability. Instead of using the entire dataset for each iteration, we use a random selection of data batches in stochastic gradient descent.
- As a result, we only sample a small portion of the dataset. The first step in this technique is to choose the starting parameters and learning rate. Then, in each iteration, mix the data at random to get an estimated minimum. When compared to the gradient descent approach, the path taken by the algorithm is full of noise since we are not using the entire dataset but only chunks of it for each iteration

- As a result, SGD requires more iterations to attain the local minimum. The overall computing time increases as the number of iterations increases. However, even when the number of iterations is increased, the computation cost remains lower than that of the gradient descent optimizer. As a result, if the data is large and the processing time is a consideration, stochastic gradient descent should be favored over batch gradient descent.

Mini-batch Stochastic Gradient Descent

- Mini batch SGD straddles the two preceding concepts, incorporating the best of both worlds. It takes training samples at random from the entire dataset (the so-called mini batch) and computes gradients just from these.
- By sampling only a fraction of the data, it aims to approach Batch Gradient Descent. We require fewer rounds because we're utilizing a chunk of data rather than the entire dataset.
- As a result, the mini-batch gradient descent technique outperforms both stochastic and batch gradient descent algorithms. This approach is more efficient and reliable than previous gradient descent variations.

- Because the method employs batching, all of the training data does not need to be placed into memory, making the process more efficient.
- In addition, the cost function in mini-batch gradient descent is noisier than that in batch gradient descent but smoother than that in stochastic gradient descent.
- Mini-batch gradient descent is therefore excellent and delivers a nice mix of speed and precision. Mini-batch SGD is the most often utilized version in practice since it is both computationally inexpensive and produces more stable convergence.

Adagrad(Adaptive Gradient Descent) Optimizer

- Adagrad keeps a running total of the squares of the gradient in each dimension, and we adjust the learning rate depending on that total in each update. As a result, each parameter has a variable learning rate (or an adaptive learning rate).
- Furthermore, when we use the root of the squared gradients, we only consider the magnitude of the gradients, not the sign. We can observe that the learning rate is reduced when the gradient changes rapidly.
- The learning rate will be higher when the gradient changes slowly. Due to the monotonic growth of the running squared sum, one of Adagrad's major flaws is that the learning rate decreases with time.

RMS prop (Root Mean Square) Optimizer

- Among deep learning acionados, the RMS prop is a popular optimizer. This might be due to the fact that it hasn't been published but is nonetheless well-known in the community.
- RMS prop is a natural extension of RPPROP's work. The problem of fluctuating gradients is solved by RPPROP. The issue with the gradients is that some were modest while others may be rather large.
- As a result, establishing a single learning rate may not be the ideal option. RPPROP adjusts the step size for each weight based on the sign of the gradient.
- The two gradients are initially compared for signs in this technique.

Adam Deep Learning Optimizer

- To update network weights during training, this optimization approach is a further development of stochastic gradient descent.
- Unlike SGD, Adam optimizer modifies the learning rate for each network weight independently, rather than keeping a single learning rate for the entire training. The Adam optimizers inherit both Adagrad and RMS prop algorithm characteristics.
- Instead of using the first moment (mean) like in RMS Prop, Adam employs the second moment of the gradients to modify learning rates. We take the second instance of the gradients to imply the uncentered variance (we don't remove the mean).

Saddle point problem in neural network

- The term **“saddle point”** in the context of machine learning refers to a specific point in the optimization landscape of a cost function where the gradient is zero, but the point is neither a minimum nor a maximum.
- Instead, it’s a point where the surface of the cost function resembles a saddle, with some dimensions curving upward and others downward.
- **Key Characteristics of a Saddle Point:**

1. Zero Gradient:
2. Neither Minimum nor Maximum
3. Flat in Some Dimensions, Steep in Others
4. Challenge for Optimization Algorithms

- **Why Saddle Points are a Challenge:**

1. Gradient Misleading
2. Slow Convergence
3. Dimensionality Matters
4. Saddle Points vs. Minima

- **Dealing with Saddle Points:**

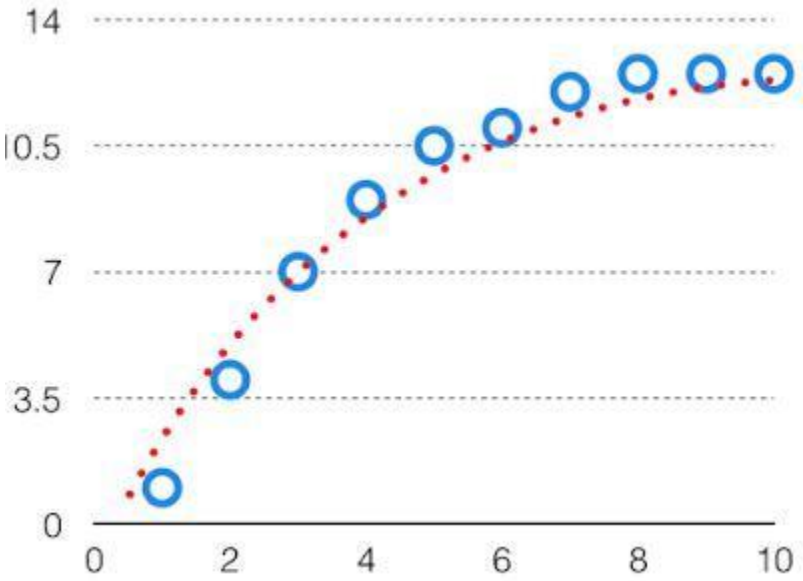
1. Higher-Order Optimization
2. Random Initialization
3. Noise Injection
4. Gradient Clipping
5. Use of Momentum

Saddle points present challenges in the optimization of cost functions in machine learning. While not as problematic as local minima, they can still hinder the convergence of optimization algorithms. Researchers and practitioners employ various strategies, such as higher-order optimization, random initialization, and noise injection, to mitigate the impact of saddle points and improve the efficiency of optimization processes in high-dimensional spaces.

Bias Variance Tradeoff

- If the algorithm is too simple (hypothesis with linear equation) then it may be on high bias and low variance condition and thus is error-prone. If algorithms fit too complex (hypothesis with high degree equation) then it may be on high variance and low bias.
- In the latter condition, the new entries will not perform well. Well, there is something between both of these conditions, known as a Trade-off or Bias Variance Trade-off.
- This tradeoff in complexity is why there is a tradeoff between bias and variance. An algorithm can’t be more complex and less complex at the same time.

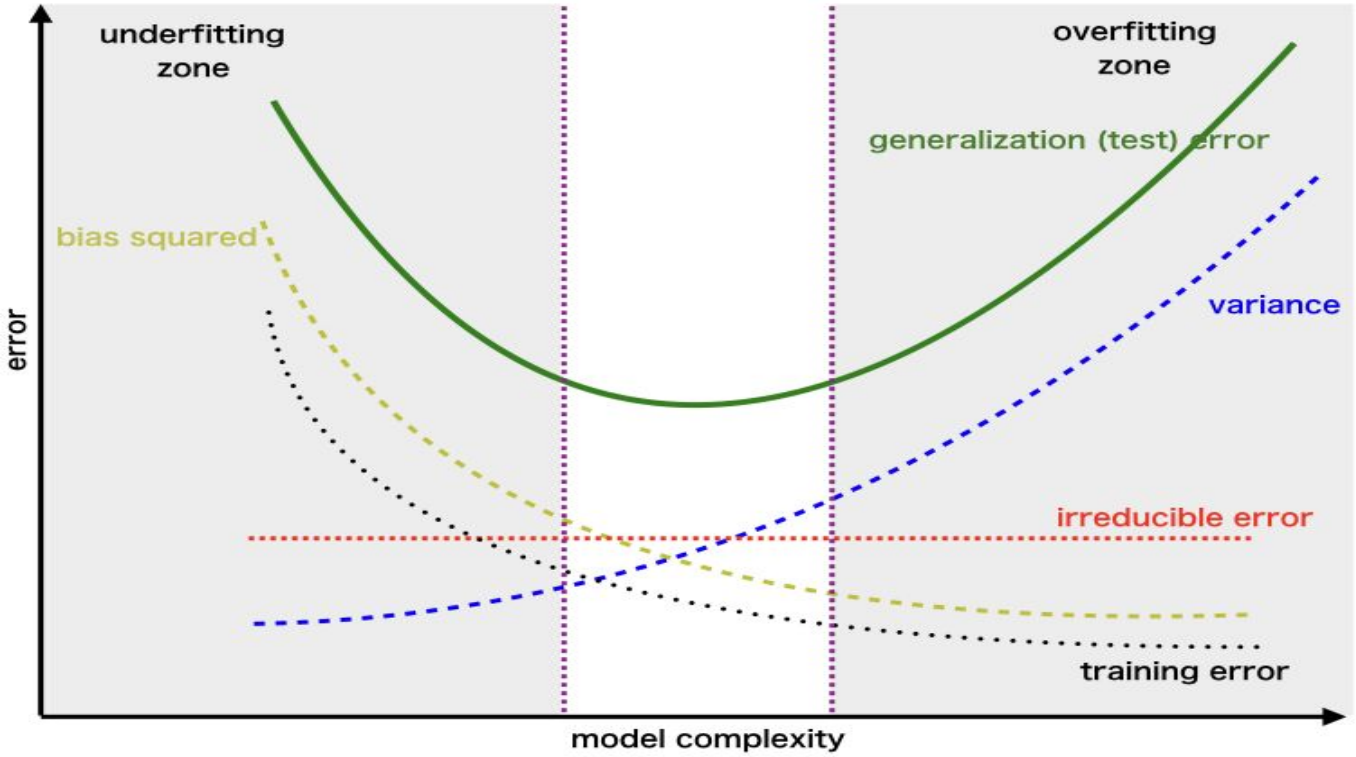
- For the graph, the perfect tradeoff will be like this.



- We try to optimize the value of the total error for the model by using the Bias-Variance Tradeoff.

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

- The best fit will be given by the hypothesis on the tradeoff point. The error to complexity graph to show trade-off is given as –



- This is referred to as the best point chosen for the training of the algorithm which gives low error in training as well as testing data.

Regularization

- Regularization may be defined as any modification or change in the learning algorithm that helps reduce its error over a test dataset, commonly known as generalization error but not on the supplied or training dataset.
- In learning algorithms, there are many variants of regularization techniques, each of which tries to cater to different challenges. These can be listed down straightforwardly based on the kind of challenge the technique is trying to deal with:

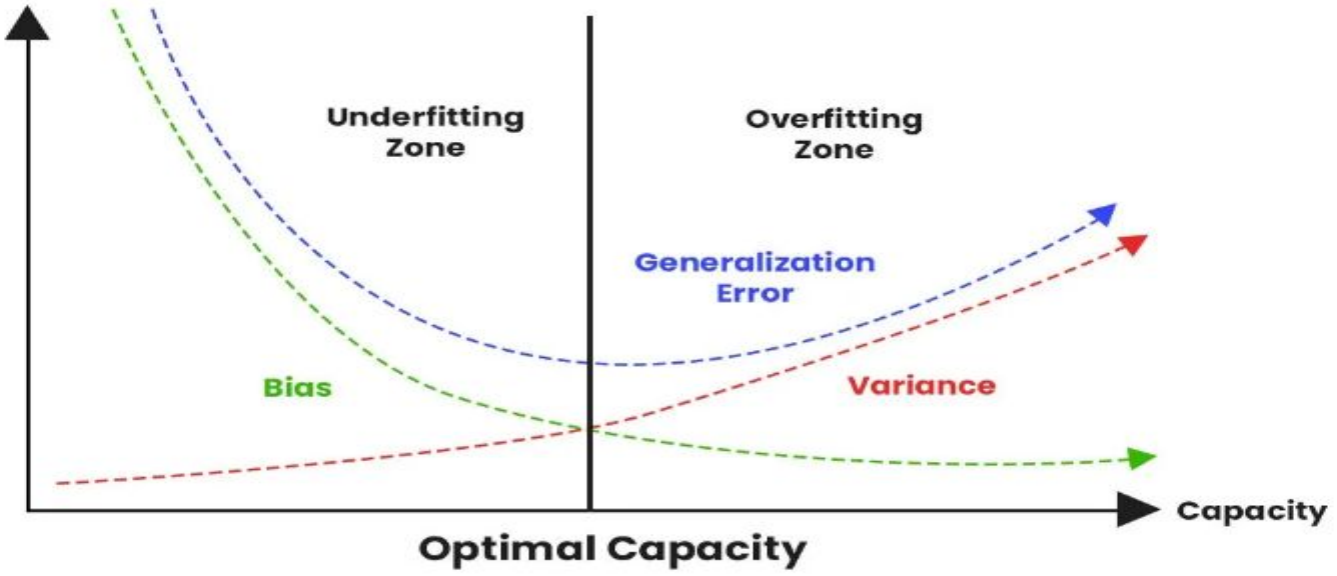
1. Some try to put extra constraints on the learning of an ML model, like adding restrictions on the range/type of parameter values.
2. Some add more terms in the objective or cost function, like a soft constraint on the parameter values. More often than not, a careful selection of the right constraints and penalties in the cost function contributes to a massive boost in the model's performance, specifically on the test dataset.

3. These extra terms can also be encoded based on some prior information that closely relates to the dataset or the problem statement.
4. One of the most commonly used regularization techniques is creating ensemble models, which take into account the collective decision of multiple models, each trained with different samples of data.

   The main aim of regularization is to reduce the over-complexity of the machine learning models and help the model learn a simpler function to promote generalization.

Regularization in Deep Learning:

- In the context of deep learning models, most regularization strategies revolve around regularizing estimators. So now the question arises what does regularizing an estimator means?
- Bias vs variance tradeoff graph here sheds a bit more light on the nuances of this topic and demarcation:



Bias vs Variance tradeoff graph

- Regularization of an estimator works by trading increased bias for reduced variance. ***An effective regularize will be the one that makes the best trade between bias and variance, and the end-product of the tradeoff should be a significant reduction in variance at minimum expense to bias.*** In simpler terms, this would mean low variance without immensely increasing the bias value.
- We consider two scenarios:

1. The true data-generating process/function: F1, which created the dataset
2. Creating a generating process/function: F2 that mimics F1 but also explores other possible generating scenarios/functions

- The work of regularization techniques is to help take our model from F2 to F1 without overly complicating F2. Deep learning algorithms are mostly used in more complicated domains like images, audio, text sequences or simulating complex decision making tasks.
- **The True data-generation process: F1 is almost impossible to be correctly mapped, hence with regularization, we aim to bring our model with F2 function as close as possible to the original F1 function.**

L2 Parameter Regularization:

- The Regression model that uses L2 regularization is called Ridge Regression.

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

$$\min_\theta J(\theta)$$

- Regul[illegible] The regularization parameter (lambda) penalizes all the parameters except intercept so that the model generalizes the data and won't overfit. Ridge regression adds **"squared magnitude of the coefficient"** as penalty term to the loss function. Here the box part in the above image represents the L2 regularization element/term.

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

- Lambda is hyperparameter

- If lambda is zero, then it is equivalent to OLS.
- ***Ordinary Least Square or OLS, is a stats model which also helps us in identifying more significant features that can have a heavy influence on the output.***
- But if lambda is very large, then it will add too much weight, and it will lead to under-fitting. Important points to be considered about L2 can be listed below:

1. **Ridge regularization forces the weights to be small but does not make them zero and does not give the sparse solution**.
2. **Ridge is not robust to outliers** as square terms blow up the error differences of the outliers, and the regularization term tries to fix it by penalizing the weights.
3. Ridge regression performs better when all the input features influence the output, and all with **weights are of roughly equal size**.
4. **L2 regularization can learn complex data patterns**

Early Stopping

- Early stopping is a regularization technique that aims to find the optimal point at which to halt the training process of a deep learning model.
- This technique uses a hold-out validation set and a performance metric, such as loss, to monitor the model’s progress.
- During training, the model's performance is evaluated on both the training set and a separate validation set.
- When the model's performance on the validation set stops improving and starts to degrade, early stopping triggers the halt of training.
- Typically, the training loss decreases continuously, while the validation loss decreases initially but starts to increase once overfitting begins.
- Early stopping aims to stop training at the inflection point where the validation loss is lowest.

Here's how early stopping works:

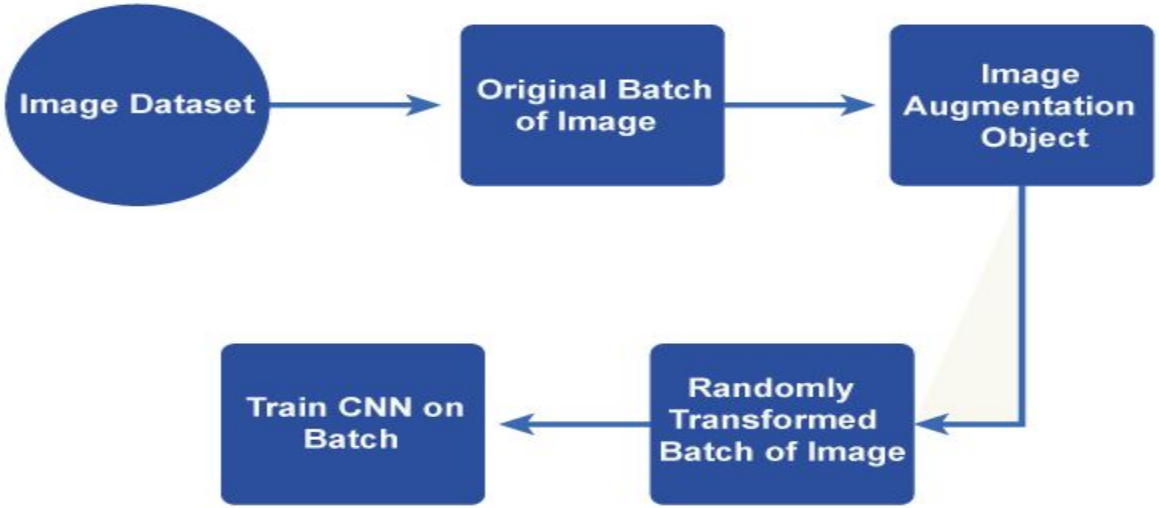
- Split the available data into three sets: training, validation, and test sets.
- Train the model on the training set for a certain number of epochs.
- Make sure the best model is saved at each epoch, early stopping often involves checkpointing.
- At each epoch, evaluate the model's performance on the validation set using a chosen metric (e.g., loss or accuracy).
- If the model's performance on the validation set starts to worsen for a specified number of consecutive epochs (known as the patience), stop the training process.
- Select the model checkpoint that achieved the best performance on the validation set as the final model.

Benefits of Early Stopping

- **Regularization:** Early stopping acts as a regularization technique by preventing the model from overfitting to the training data. It encourages the model to find a balance between fitting the training data and generalizing to unseen examples.
- **Computational Efficiency:** By stopping the training process early, we can save computational resources and time. Instead of training the model for an excessive number of epochs, early stopping allows us to find the optimal stopping point efficiently.
- **Automated Tuning:** Early stopping automates the process of determining the optimal number of training epochs. It eliminates the need for manual intervention and helps find the sweet spot for stopping the training.
- **Robustness:** Early stopping makes the model more robust to variations in the training data and hyperparameters. It reduces the model's sensitivity to noise and outliers in the training set.

Data Augmentation

- In machine learning, data augmentation is a common method for manipulating existing data to artificially increase the size of a training dataset. In an attempt to enhance the efficiency and flexibility of machine learning models, data augmentation looks for the boost in the variety and volatility of the training data.
- Data augmentation can be especially beneficial when the original set of data is small as it enables the system to learn from a larger and more varied group of samples.



- By applying arbitrary changes to the information, the expanded dataset can catch various varieties of the first examples, like various perspectives, scales, revolutions, interpretations, and mishappenings. As a result, the model can better adapt to unknown data and become more resilient to such variations.
- Techniques for data augmentation can be used with a variety of data kinds, including time series, text, photos, and audio.
- Here are a few frequently used methods of data augmentation for image data:

1. Images can be rotated at different angles and flipped horizontally or vertically to create alternative points of view.
2. **Random cropping and padding:** By applying random cropping or padding to the photos, various scales, and translations can be simulated.
3. **Scaling and zooming:** The model can manage various item sizes and resolutions by rescaling the photos to different sizes or zooming in and out.
4. **Shearing and perspective transform:** Changing an image's shape or perspective can imitate various viewing angles while also introducing deformations.
5. **Color jittering:** By adjusting the color characteristics of the images, including their brightness, contrast, saturation, and hue, the model can be made to be more resilient to variations in illumination.
6. **Gaussian noise:** By introducing random Gaussian noise to the images, the model's resistance to noisy inputs can be strengthened.

Types of Data Augmentations

1. Real Data Augmentation

- The process of modifying real-world data samples to enhance the base of training for artificial intelligence models is referred to as "real data augmentation." Real data augmentation, as compared to synthetic data augmentation produces new samples based on existing data and also modifies the original data in a way that accurately depicts fluctuations and disturbances that occur in the real world.

2. Synthetic Data Augmentation

- In machine learning, synthetic data augmentation creates additional artificial data samples based on current data to increase the training set. It is a method for broadening the variety and volume of data accessible for model training. When a dataset is scarce or more variations are required to boost a model's performance, synthetic data augmentation can be especially helpful

Parameter Sharing and Typing

- We usually apply limitations or penalties to parameters in relation to a fixed region or point. $\mathbf{L^2}$ regularisation (or weight decay) penalises model parameters that deviate from a fixed value of zero, for example.
- However, we may occasionally require alternative means of expressing our prior knowledge of appropriate model parameter values. We may not know exactly what values the parameters should take, but we do know that there should be some dependencies between the model parameters based on our knowledge of the domain and model architecture.
- We frequently want to communicate the dependency that various parameters should be near to one another.

Parameter Typing

- Two models are doing the same classification task (with the same set of classes), but their input distributions are somewhat different.
- We have model **A** has the parametersw(A) $\boldsymbol{w}(A)$
- Another model **B** has the parameters w(B) $\boldsymbol{w}(B)$

y^(A)=f(w(A),x) $y\hat{}(A)=f(\boldsymbol{w}(A),\boldsymbol{x})$

and

y^(B)=g(w(B),x) $y\hat{}(B)=g(\boldsymbol{w}(B),\boldsymbol{x})$

- are the two models that transfer the input to two different but related outputs.
- Assume the tasks are comparable enough (possibly with similar input and output distributions) that the model parameters should be near to each other: $\forall$ i,wi(A) $\forall\, i,wi(A)$ should be close to wi(B) $wi(B)$. We can take advantage of this data by regularising it. We can apply a parameter norm penalty of the following form: Ω (w(A),w(B))= // w(A)−w(B) // 22 $\Omega(\boldsymbol{w}(A),\boldsymbol{w}(B))=\boldsymbol{w}(A)-\boldsymbol{w}(B)22$. We utilised an $\mathbf{L}^2$ penalty here, but there are other options.

Parameter Sharing

- The parameters of one model, trained as a classifier in a supervised paradigm, were regularised to be close to the parameters of another model, trained in an unsupervised paradigm, using this method (to capture the distribution of the observed input data). Many of the parameters in the classifier model might be linked with similar parameters in the unsupervised model thanks to the designs. While a parameter norm penalty is one technique to require sets of parameters to be equal, constraints are a more prevalent way to regularise parameters to be close to one another. Because we view the numerous models or model components as sharing a unique set of parameters, this form of regularisation is commonly referred to as parameter sharing. The fact that only a subset of the parameters (the unique set) needs to be retained in memory is a significant advantage of parameter sharing over regularising the parameters to be close (through a norm penalty). This can result in a large reduction in the memory footprint of certain models, such as the convolutional neural network.

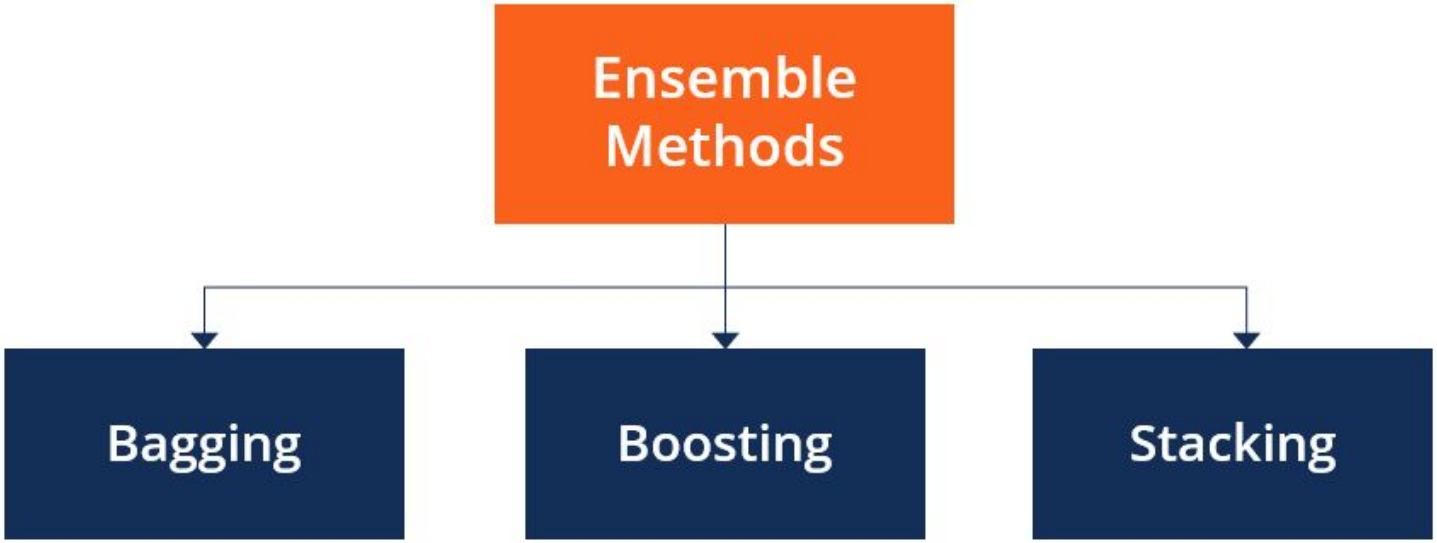
Injecting noice at input

- Injecting noise at the input is a common technique used in machine learning to improve the robustness and generalization of models. By adding noise to the input data, the model is forced to learn the underlying patterns in the data rather than simply memorizing specific instances. This helps to prevent overfitting and improves the model's ability to generalize to new, unseen data. By exposing the model to different types of noise, it can learn to recognize patterns in the input data that are more invariant to variations in the data.
- There are different types of noise that can be injected at the input of a deep learning model. Each type of noise has its advantages and disadvantages, and the choice of which type to use depends on the specific problem and the characteristics of the input data.

1. Gaussian noise
2. Dropout
3. Salt-and-pepper noise
4. Random rotation, scaling or translation

Ensemble Methods

- Ensemble methods are techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model. The combined models increase the accuracy of the results significantly. This has boosted the popularity of ensemble methods in machine learning.



Categories of Ensemble Methods

- Ensemble methods fall into two broad categories, i.e., sequential ensemble techniques and parallel ensemble techniques. **Sequential ensemble techniques** generate base learners in a sequence, e.g., Adaptive Boosting (AdaBoost). The sequential generation of base learners promotes the dependence between the base learners. The performance of the model is then improved by assigning higher weights to previously misrepresented learners.
- In **parallel ensemble techniques**, base learners are generated in a parallel format, e.g., random forest. Parallel methods utilize the parallel generation of base learners to encourage independence between the base learners. The independence of base learners significantly reduces the error due to the application of averages.

Main Types of Ensemble Methods

1. **Bagging:** Bagging, the short form for bootstrap aggregating, is mainly applied in classification and regression. It increases the accuracy of models through decision trees, which reduces variance to a large extent. The reduction of variance increases accuracy, eliminating overfitting, which is a challenge to many predictive models.
2. **Boosting:** Boosting is an ensemble technique that learns from previous predictor mistakes to make better predictions in the future. The technique combines several weak base learners to form one strong learner, thus significantly improving the predictability of models. Boosting works by arranging weak learners in a sequence, such that weak learners learn from the next learner in the sequence to create better predictive models.
3. **Stacking:** Stacking, another ensemble method, is often referred to as stacked generalization. This technique works by allowing a training algorithm to ensemble several other similar learning algorithm predictions. Stacking has been successfully implemented in regression, density estimations, distance learning, and classifications. It can also be used to measure the error rate involved during bagging.

Greedy Layer Wise Pre-Training

- Artificial intelligence has undergone a revolution thanks to neural networks, which have made significant strides possible in a number of areas like speech recognition, computer vision, and natural language processing. Deep neural network training, however, may be difficult, particularly when working with big, complicated datasets. One method that tackles some of these issues is greedy layer-wise pre-training, which initializes deep neural network settings layer by layer.
- Greedy layer-wise pre-training is used to initialize the parameters of deep neural networks layer by layer, beginning with the first layer and working through each one that follows. A layer is trained as if it were a stand-alone model at each step, using input from the layer before it and output to go to the layer after it. Typically, developing usable representations of the input data is the training aim.

Processes of Greedy Layer-Wise Pre-Training

The process of greedy layer-wise pre-training can be staged as follows:

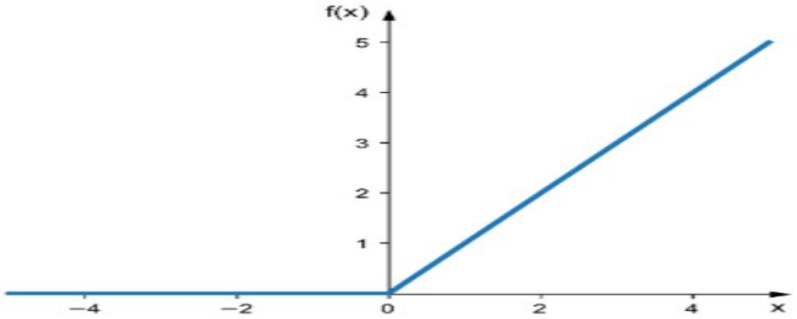
- **Initialization:** The neural network's first layer is trained on its own using autoencoders and other unsupervised learning strategies. Learning a collection of features that highlight important elements of the input data is the aim.
- **Extracting Feature:** The activations of the first layer are utilized as features to train the subsequent layer after it has been trained. Each layer learns to represent the traits discovered by the layer before it in a higher-level abstraction when this process is repeated repeatedly.
- **Fine-Tuning:** The network is adjusted as a whole using supervised learning methods once every layer has been pretrained in this way. To maximize performance on a particular job, this entails simultaneously modifying all of the network's parameters using a labeled dataset.

Better Activation Functions

- Activation functions are a crucial part of artificial neural networks (ANNs) used for deep learning. They determine the output of a neuron, which then propagates forward to the next layer of neurons. The activation function essentially decides whether a neuron should be "activated" or not based on the input it receives. There are several types of activation functions, each with their own advantages and disadvantages. The better activation functions that can be used in deep learning are:

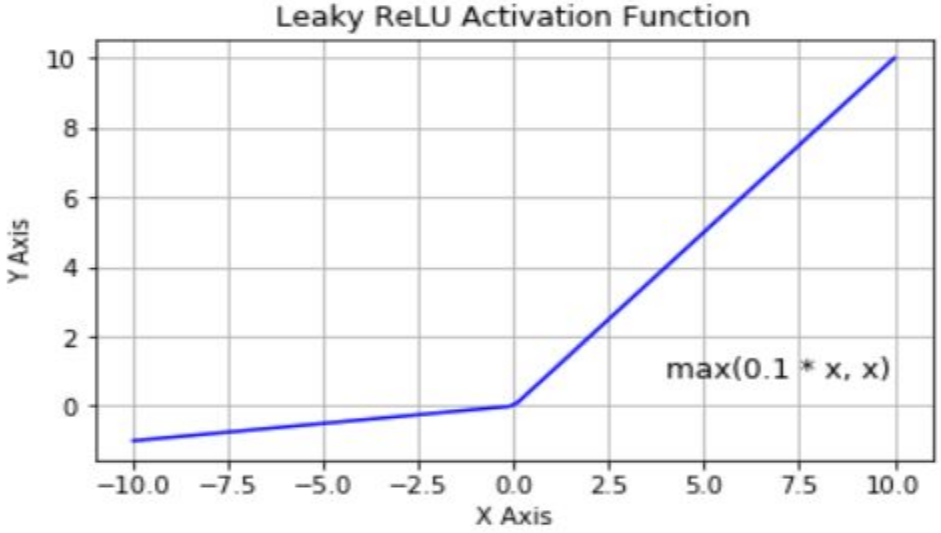
ReLU (Rectified Linear Unit) Activation Function

- ReLU is one of the most commonly used activation functions in deep learning. It is a simple function that returns the input if it is positive, and zero otherwise. ReLU is computationally efficient, and it is easy to implement. It has been shown to work well in deep neural networks, particularly for image classification tasks.



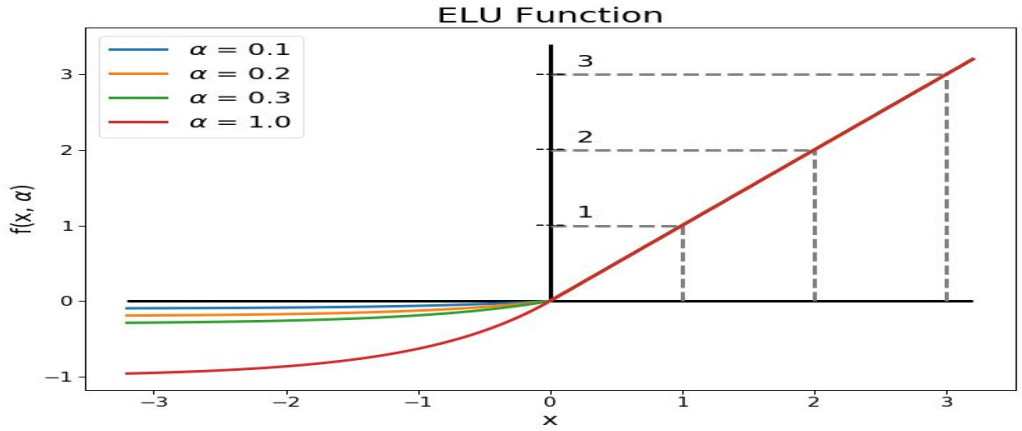
2. Leaky ReLU Activation Function

- Leaky ReLU is an extension of the ReLU activation function. It is similar to ReLU, but instead of returning zero for negative inputs, it returns a small negative value. This helps to avoid the "dying ReLU" problem, where some neurons can become permanently inactive during training.
- The Leaky ReLU activation function has been shown to perform better than ReLU in some deep neural network architectures. It is particularly useful in networks with a large number of negative inputs. However, it can be slower to compute than ReLU.



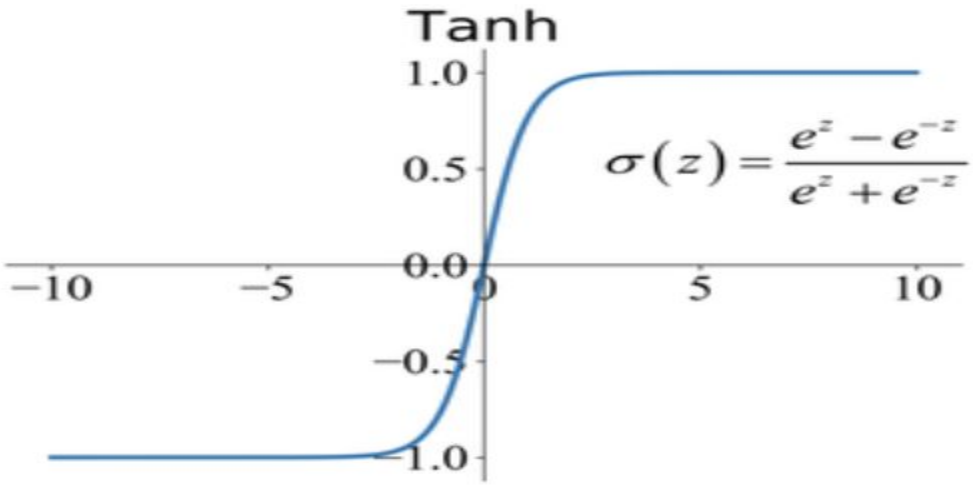
3. ELU (Exponential Linear Unit) Activation Function

- ELU is a more complex activation function than ReLU or Leaky ReLU. It returns the input if it is positive, and an exponential function of the input minus one otherwise. The exponential function helps to avoid the "dead neurons" problem that can occur with ReLU.
- ELU has been shown to perform better than ReLU and Leaky ReLU in some deep neural network architectures. It has a smooth curve, which makes it easier to optimize than other activation functions. However, it can be slower to compute than ReLU and Leaky ReLU.



4. Tanh (Hyperbolic Tangent) Activation Function

- Tanh is a non-linear activation function that returns values between -1 and 1. It is similar to the sigmoid activation function, but it is centered at zero, which makes it more symmetric. Tanh is useful in some types of neural networks, such as those used for language modeling and speech recognition.
- Tanh has been shown to perform well in some deep neural network architectures, particularly those with recurrent connections. However, it can suffer from the same vanishing gradient problem that can occur with sigmoid activation functions.



5. Softmax Activation Function

- Softmax is a special type of activation function used in the output layer of a neural network for multi-class classification tasks. It returns a probability distribution over the possible classes, with the sum of the probabilities equal to one. Softmax is useful for tasks such as image recognition, where the network needs to classify an image into one of several possible categories.
- Softmax has been shown to perform well in many deep neural network architectures, particularly for multi-class classification tasks. However, it can be sensitive to outliers in the input data.