Unit 02: Deep Feedforward Networks

—

Chapter - Regularization for Deep Learning

Overfitting and Regularization

Poor performance in machine learning models comes from either overfitting or underfitting. **Overfitting** happens when the learned hypothesis is fitting the training data so well that it hurts the model's performance on unseen data.

To prevent overfitting, we use **regularization**. Broken down, the word "regularize" states that we're making something regular. In a mathematical or ML context, we make something regular by adding information which creates a solution that prevents overfitting.

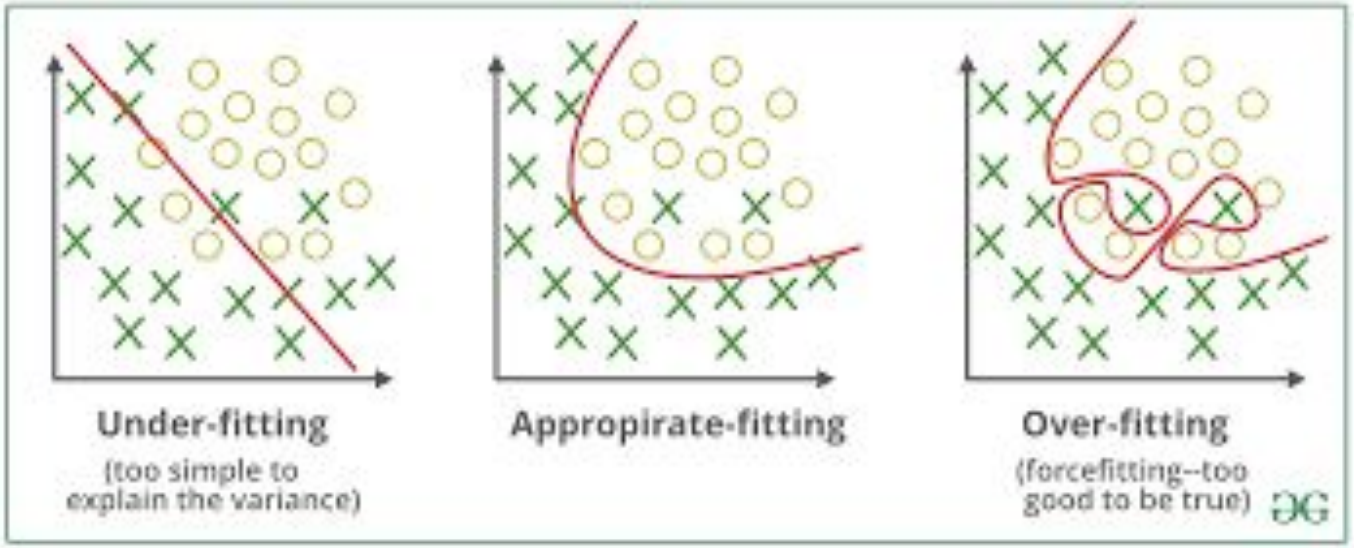


Figure: Underfitting, appropriate fitting and overfitting

https://neptune.ai/blog/fighting-overfitting-with-l1-or-l2-regularization
https://www.youtube.com/watch?v=uisWo_LXMwY

To detect overfitting using cross validation

To detect overfitting in our ML model, we need a way to test it on unseen data. We often leverage a technique called **“Cross Validation”** whenever we want to evaluate the performance of a model on unseen instances.

The most basic type of cross validation implementation is the hold-out based cross validation. This implementation splits the available data into training and testing sets. To evaluate our model using hold-out based cross validation, we would first build and train a model on the training split of our hold-out set, and then use the model to make predictions using the test set, so we can evaluate how it performs.

Parameter Norm Penalties

Many regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J. We denote the regularized objective function by J:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

where $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term, Ω, relative to the standard objective function J. Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.

L^1 Parameter Regularization

L1 regularization, also known as L1 norm or Lasso (in regression problems), combats overfitting by shrinking the parameters towards 0. This makes some features obsolete.

It's a form of feature selection, because when we assign a feature with a 0 weight, we're multiplying the feature values by 0 which returns 0, eradicating the significance of that feature. If the input features of our model have weights closer to 0, our L1 norm would be sparse. A selection of the input features would have weights equal to zero, and the rest would be non-zero.

Mathematically, we express L1 regularization by extending our loss function like such:

$$LossFunction = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y} - Y)^2 + \lambda \sum_{i=1}^{N} \mid \theta_i \mid$$

Essentially, when we use L1 regularization, we are penalizing the absolute value of the weights.

In real world environments, we often have features that are highly correlated. For example, the year our home was built and the number of rooms in the home may have a high correlation. Something to consider when using L1 regularization is that when we have highly correlated features, the L1 norm would select only 1 of the features from the group of correlated features in an arbitrary nature, which is something that we might not want.

L^2 Parameter Regularization

L^2 regularization, or the L^2 norm, or Ridge (in regression problems), combats overfitting by forcing weights to be small, but not making them exactly 0.

So, if we're predicting house prices again, this means the less significant features for predicting the house price would still have some influence over the final prediction, but it would only be a small influence.

The regularization term that we add to the loss function when performing L2 regularization is the sum of squares of all of the feature weights:

$$LossFunction = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y} - Y)^2 + \lambda\sum_{i=1}^{N}\theta_i^2$$

Summary

In optimization and machine learning, a penalty refers to an additional term added to the cost function to discourage or penalize certain characteristics of the model parameters. The goal of introducing a penalty is to control the complexity of the model and improve generalization to unseen data.

- The penalty is added to the original cost function J(θ), resulting in a new, regularized cost function: J_hat(θ)=J(θ)+penalty
- The penalty term depends on the model parameters θ nd is used to adjust the cost function. The objective is to minimize this regularized cost function.

Types of Penalties:

- **L1 Penalty (Lasso Regularization):** Adds the sum of the absolute values of the weights:

  Encourages sparsity in the model (many weights become zero).

$$\text{Penalty} = \lambda \sum_i |\theta_i|$$

- **L2 Penalty (Ridge Regularization):** Adds the sum of the squared values of the weights:

  Encourages smaller weights but does not force them to be zero.

$$\text{Penalty} = \lambda \sum_i \theta_i^2$$

Adding a penalty term alters the optimization problem. Instead of just minimizing the original cost function $J(\theta)$, the model now minimizes the regularized cost function $\tilde{J}(\theta)$. This encourages the model to find a balance between fitting the training data and keeping the weights small or sparse.

Example

Imagine you are training a model to fit some data. Without a penalty, the model might find a solution that fits the training data very well but has overly large weights, leading to poor performance on new data (overfitting). By adding a penalty term to the cost function, you discourage the model from using large weights, leading to a simpler model that generalizes better.

Norm Penalties as Constrained Optimization

Methods for regularizing a cost function J(θ;X,y) by adding a parameter norm penalty. This is formulated as: $\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$

here Ω(θ) is a regularization term, and α is the regularization coefficient.

To impose constraints on Ω(θ), such as ensuring it remains below a constant k, a generalized Lagrange function can be constructed: $L(\theta, \alpha; X, y) = J(\theta; X, y) + \alpha(\Omega(\theta) - k)$

The solution to this constrained problem is found by minimizing L(θ,α)with respect to θ and α: $\theta^* = \arg\min_\theta \max_{\alpha \geq 0} L(\theta, \alpha)$

In regularized optimization problems, especially those involving constraints, the Lagrangian approach is used to handle constraints by introducing penalty terms.

This approach allows for incorporating constraints into the optimization process by converting them into an optimization problem where both the objective function and the constraints are handled simultaneously.

Here, θ^* represents the optimal parameters of the model after considering both the original objective function and the penalty for violating the constraints. This formulation ensures that the solution θ^* is found in a way that balances the objective function with the imposed constraints through the penalty term.

This resembles regularized training with a penalty on $\Omega(\theta)$. The choice of Ω determines the type of regularization; for instance, the L2 norm constraint weights within an L2 ball, while the L1 norm restricts them to a region with limited L1 norm.

Using explicit constraints instead of penalties can sometimes be more effective, especially when k is known. This approach involves projecting θ back to the feasible region after each gradient update, which avoids the problem of local minima caused by penalties that might lead to very small weights (dead units). Explicit constraints also help in maintaining optimization stability, especially with high learning rates, by preventing weights from growing unboundedly.

Dataset Augmentation

- To help a machine learning model generalize better, training it on more data is ideal. However, if you have limited data, you can create new, fake data to augment your training set.
- **Creating Fake Data:** For classification tasks, it's relatively easy to generate new data by transforming existing data. For instance, you can slightly alter images (like translating, rotating, or scaling) to create variations that improve model performance. However, this approach is less straightforward for tasks like density estimation.
- **Effective Techniques for Image Data:** In object recognition, small transformations to images (like shifting, rotating, or scaling) are very useful. But, transformations that might change the category of the image (like flipping or rotating in certain ways) should be avoided if they alter the label.
- **Challenges and Other Uses:** Some transformations, such as out-of-plane rotations, are complex and not easily applied. Data augmentation is also useful in speech recognition. Adding random noise to inputs, known as noise injection, can help improve a model's robustness. This technique is also used in unsupervised learning (like denoising autoencoders).
- **Dropout and Noise Injection:** Injecting noise into inputs or hidden units can be a form of data augmentation at different abstraction levels. Dropout, a specific technique for regularization, is essentially a way of creating new training data by randomly turning off certain units during training.
- **Controlled Experiments:** When comparing machine learning algorithms, it's crucial to ensure that both are evaluated with the same data augmentation methods. Differences in performance may be due to the augmentation techniques rather than the algorithms themselves. Generally, adding noise is considered part of the algorithm, while domain-specific transformations are seen as preprocessing steps.

Noise Robustness

For some models, the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights.

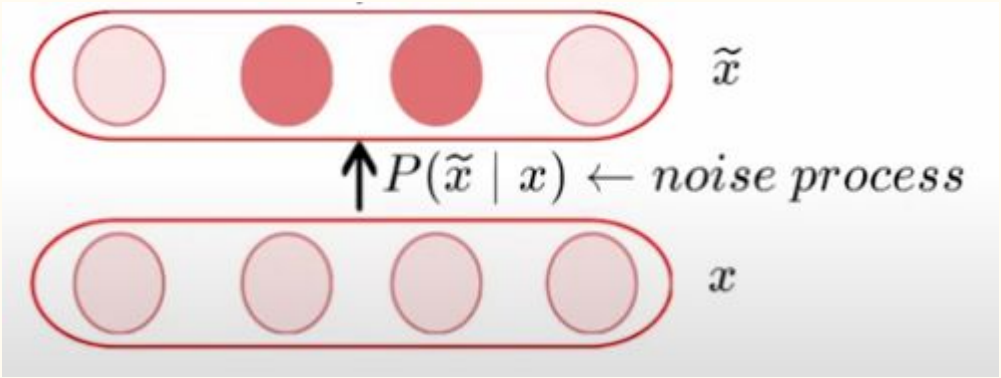
One way is to implement dropout algorithm. Another way that noise has been used in the service of regularizing models is by adding it to the weights. This can be interpreted as a stochastic implementation of Bayesian inference over the weights. The Bayesian treatment of learning would consider the model weights to be uncertain and representable via a probability distribution that reflects this uncertainty. Adding noise to the weights is a practical, stochastic way to reflect this uncertainty.

In machine learning, especially in supervised learning, the goal is often to model the relationship between input features x and output labels y by learning a function that maps x to y. The likelihood function $p(y|x)$ represents the probability of observing a particular label y given the input x. Maximizing the log likelihood $\log p(y|x)$ helps in finding the parameters of the model that best fit the data.

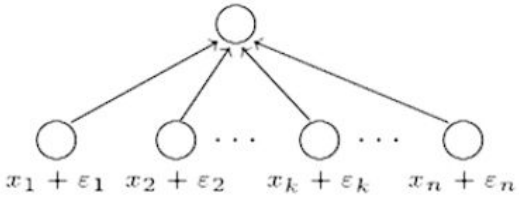
However, if there are mistakes or noise in the labels y, directly maximizing $\log p(y|x)$ might lead to suboptimal models because the model could learn from incorrect labels and thus produce biased or erroneous predictions.

Adding noise to the input

- Increase our dataset
- Increases complexity
- Corrupted dataset must also give us the same output



https://www.youtube.com/watch?v=agGUR06jM_g



$$\varepsilon \sim \mathcal{N}(0,\, \sigma^2)$$

$$\widetilde{x}_i = x_i + \varepsilon_i$$

$$\widehat{y} = \sum_{i=1}^{n} w_i x_i$$

$$\widetilde{y} = \sum_{i=1}^{n} w_i \widetilde{x}_i$$

$$= \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} w_i \varepsilon_i$$

$$= \widehat{y} + \sum_{i=1}^{n} w_i \varepsilon_i$$

We are interested in $E[(\widetilde{y} - y)^2]$

$$E\left[(\widetilde{y} - y)^2\right] = E\left[\left(\widehat{y} + \sum_{i=1}^{n} w_i \varepsilon_i - y\right)^2\right]$$

$$= E\left[\left(\left(\widehat{y} - y\right) + \left(\sum_{i=1}^{n} w_i \varepsilon_i\right)\right)^2\right]$$

$$= E\left[(\widehat{y} - y)^2\right] + E\left[2(\widehat{y} - y)\sum_{i=1}^{n} w_i \varepsilon_i\right] + E\left[\left(\sum_{i=1}^{n} w_i \varepsilon_i\right)^2\right]$$

$$= E\left[(\widehat{y} - y)^2\right] + 0 + E\left[\sum_{i=1}^{n} w_i^2 \varepsilon_i^2\right]$$

($\because \varepsilon_i$ is independent of ε_j and ε_i is independent of $(\widehat{y}\text{-}y)$)

$$= \left(E\left[(\widehat{y} - y)^2\right] + \sigma^2 \sum_{i=1}^{n} w_i^2\right.$$ (same as L_2 norm penalty)

Adding Noise to the output

0	0	1	0	0	0	0	0	0	0

Hard targets

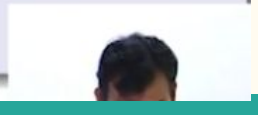
$$\text{minimize} : \sum_{i=0}^{9} p_i \log q_i$$

$$\text{true distribution} : p = \{0, 0, 1, 0, 0, 0, 0, 0, 0, 0\}$$

$$\text{estimated distribution} : q$$

Intuition

- Do not trust the true labels, they may be noisy
- Instead, use soft targets



$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$1-\varepsilon$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	$\frac{\varepsilon}{9}$	Soft targets

$$\varepsilon = \text{small positive constant}$$

$$\text{minimize} : \sum_{i=0}^{9} p_i \log q_i$$

$$\text{true distribution + noise} : p = \left\{\frac{\varepsilon}{9}, \frac{\varepsilon}{9}, 1-\varepsilon, \frac{\varepsilon}{9}, \dots\right\}$$

$$\text{estimated distribution} : q$$

https://www.youtube.com/watch?v=TtU9GhLeOk4

Semi-Supervised Learning

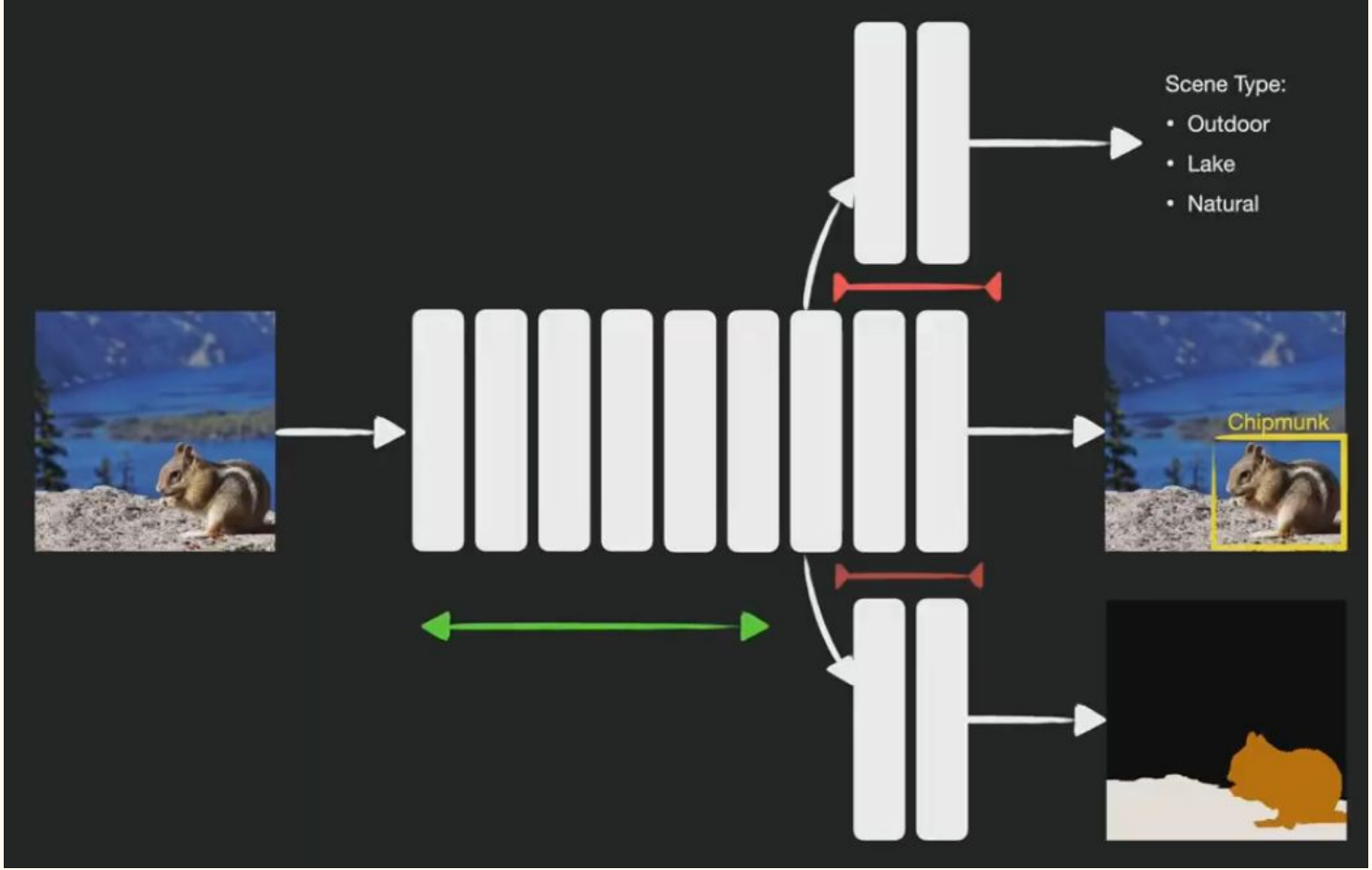
In the paradigm of semi-supervised learning, both unlabeled examples from P(x) and labeled examples from P (x, y) are used to estimate P (y | x) or predict y from x.

https://www.youtube.com/watch?v=b-yhKUINb7o refer the video to explain the example

Multi-Task Learning

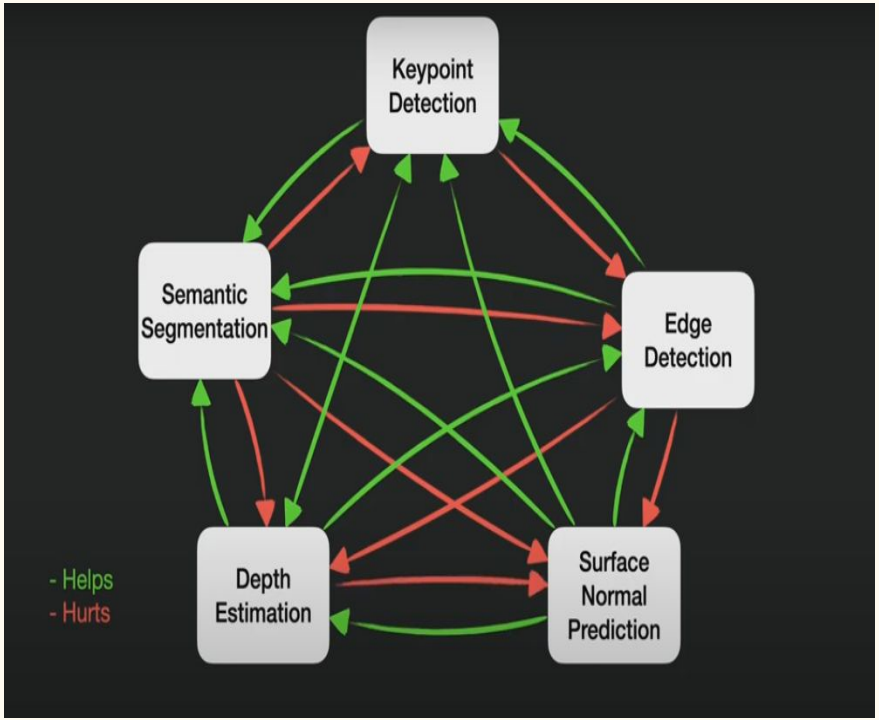
- It is about teaching machine learning model to do multiple things at a time.
- We may want to use a single model to solve multiple problems for various reasons such as efficiency and better generalization.
- Hard Sharing - The tasks are forced to share more features therefore they become more tightly coupled intuitively the closer the tasks are the more features they can share but how close the tasks are may not be always straightforward seemingly related tasks can hurt each other's accuracy when trained together so in practice it takes some trial and error to design a multi-task model
- Soft sharing– Each model has their own sets of weights and biases and the distance between these parameters in different models is regularized so that the parameters become similar and can represent all the tasks.

https://www.youtube.com/watch?v=ckQuvrksP4k

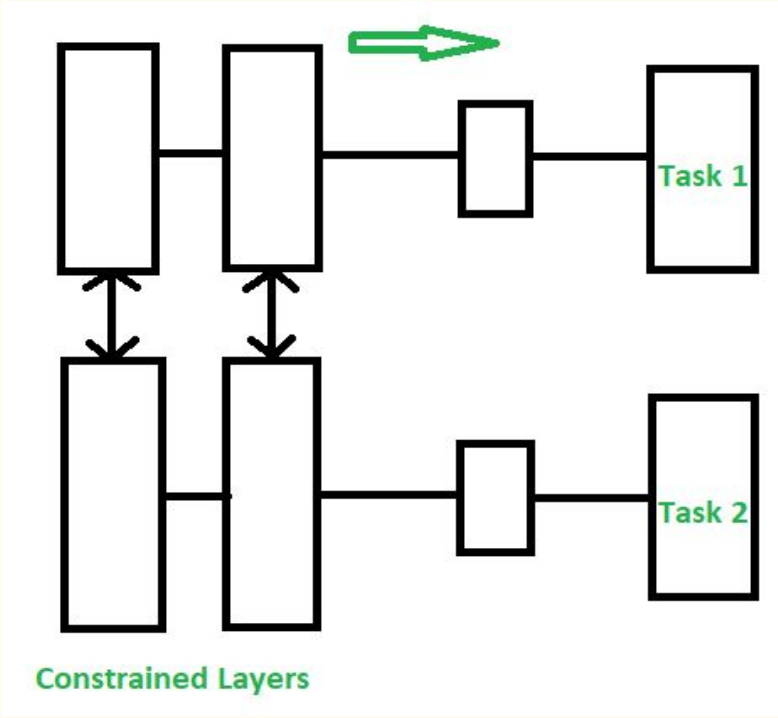


Example for MultiTask learning - To detect scene, object and segmentation from the image

Hard Sharing



Soft sharing



Early Stopping

Early Stopping is a regularization technique used to prevent overfitting in machine learning models, particularly deep neural networks. It involves monitoring the model's performance on a validation set during training and stopping the training process when performance on the validation set begins to deteriorate.

By halting training before the model starts to memorize the training data excessively, early stopping helps to improve the model's ability to generalize to unseen data. It is a simple yet effective method for enhancing model performance.

Validation set: A separate portion of data used to assess the model's performance during training.

Overfitting: When a model becomes too complex and fits the training data too closely, leading to poor performance on new data.

Generalization: The ability of a model to perform well on unseen data.

Parameter Tying

Two models are doing the same classification task (with the same set of classes), but their input distributions are somewhat different.

We have two model A and B transfer the input to two different but related outputs.

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

and

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

Assume the tasks are comparable enough (possibly with similar input and output distributions) that the model parameters should be near to each other: $\forall i, w_i^{(A)}$ should be close to $w_i^{(B)}$

We can take advantage of this data by regularising it. We can apply a parameter norm penalty of the following form of

$$\Omega\left(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}\right) = \left\|\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}\right\|_2^2$$

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.

Convolutional neural networks (CNNs) used in computer vision are by far the most widespread and extensive usage of parameter sharing. Many statistical features of natural images are translation insensitive. A shot of a cat, for example, can be translated one pixel to the right and still be a shot of a cat. By sharing parameters across several picture locations, CNNs take this property into account. Different locations in the input are computed with the same feature (a hidden unit with the same weights). This indicates that whether the cat appears in column i or column i + 1 in the image, we can find it with the same cat detector.

CNN's have been able to reduce the number of unique model parameters and raise network sizes greatly without requiring a comparable increase in training data thanks to parameter sharing.

Parameter sharing is a technique in deep learning that involves using the same parameters for more than one function in a model. This can be done in a few ways:

Sharing weights

In a feature map, all neurons can share the same weights, which reduces the number of parameters in the system and makes it more computationally efficient. This is also known as weight replication.

Multi-task learning (MTL)

In MTL, the model is trained simultaneously for all tasks, with constraints or regularization on the relationship between related parameters. This can be done using hard or soft parameter sharing, which aim to learn shared or similar hidden representations for the different tasks.

Sparse Representations

Weight decay acts by placing a penalty directly on the model parameters. Another strategy is to place a penalty on the activations of the units in a neural network, encouraging their activations to be sparse. This indirectly imposes a complicated penalty on the model parameters.

L1 penalization induces a sparse parametrization—meaning that many of the parameters become zero (or close to zero). Representational sparsity, on the other hand, describes a representation where many of the elements of the representation are zero (or close to zero). A simplified view of this distinction can be illustrated in the context of linear regression:

$$\underbrace{\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix}}_{\boldsymbol{y} \in \mathbb{R}^m} = \underbrace{\begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix}}_{\boldsymbol{A} \in \mathbb{R}^{m \times n}} \underbrace{\begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}}_{\boldsymbol{x} \in \mathbb{R}^n} \tag{7.46}$$

$$\underbrace{\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix}}_{\boldsymbol{y} \in \mathbb{R}^m} = \underbrace{\begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix}}_{\boldsymbol{B} \in \mathbb{R}^{m \times n}} \underbrace{\begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}}_{\boldsymbol{h} \in \mathbb{R}^n} \tag{7.47}$$

In the first expression, we have an example of a sparsely parametrized linear regression model. In the second, we have linear regression with a sparse representation h of the data x. That is, h is a function of x that, in some sense, represents the information present in x, but does so with a sparse vector.

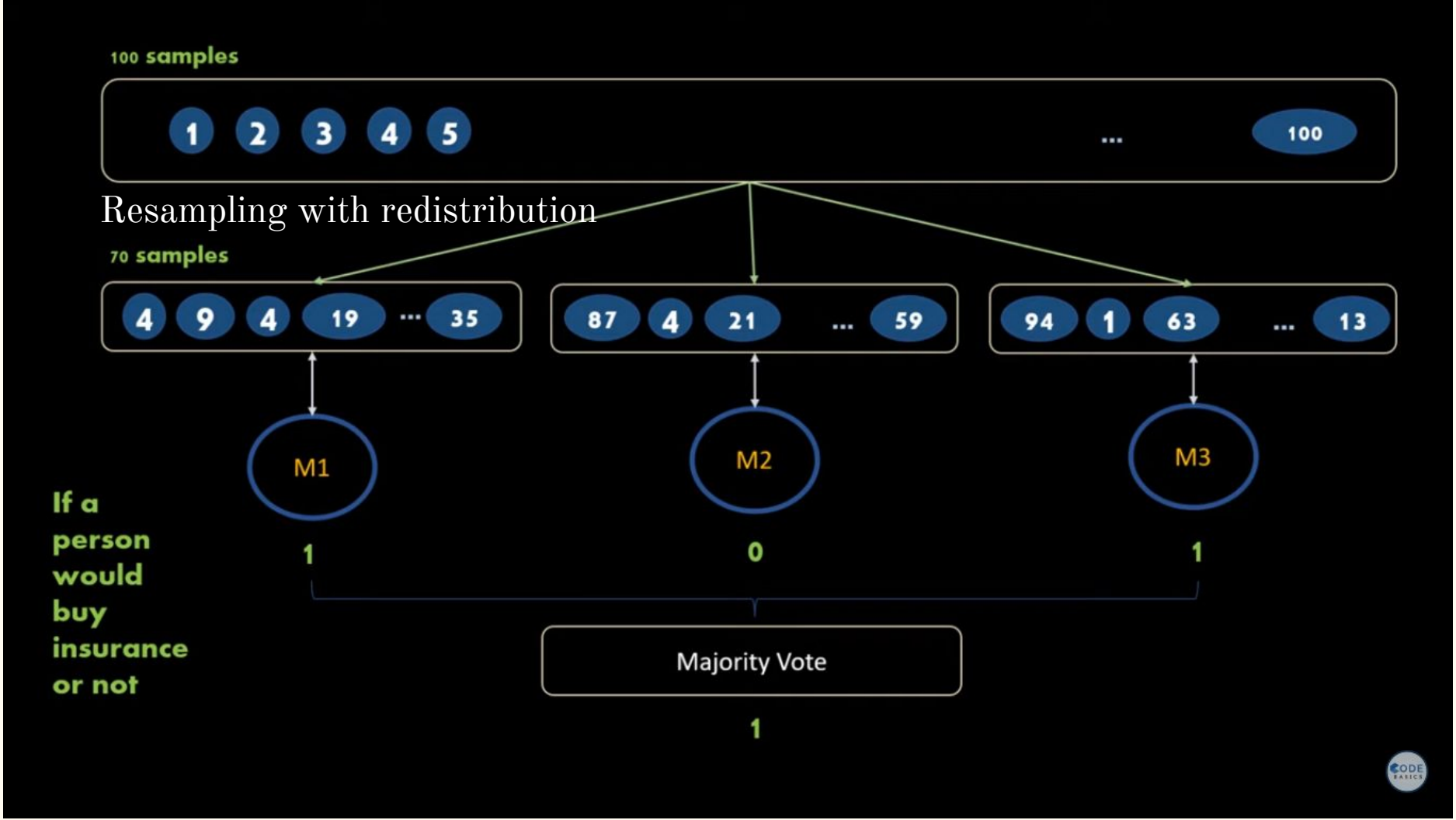
Bagging and Other Ensemble Methods

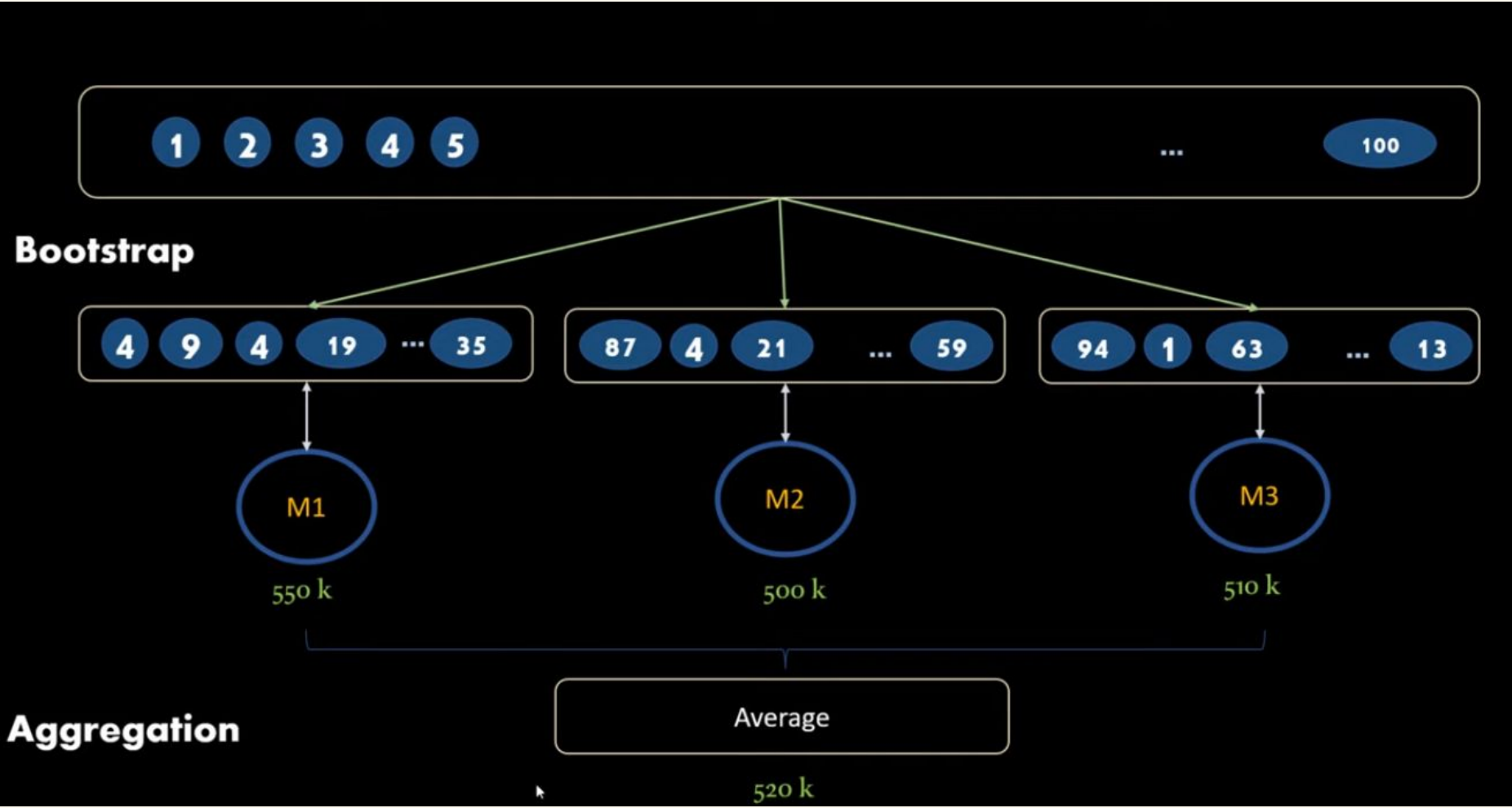
Bagging (short for bootstrap aggregating) is a technique for reducing generalization error by combining several models. The idea is to train several different models separately, then have all of the models vote on the output for test examples. This is an example of a general strategy in machine learning called model averaging. Techniques employing this strategy are known as ensemble methods.

From the figure: Logistic regression model using classification - M1, M2, M3 are the weak learner models because they are trained on the subset of a dataset so they will generalize better, not overfit and when you combine them they will provide you a good quality result.

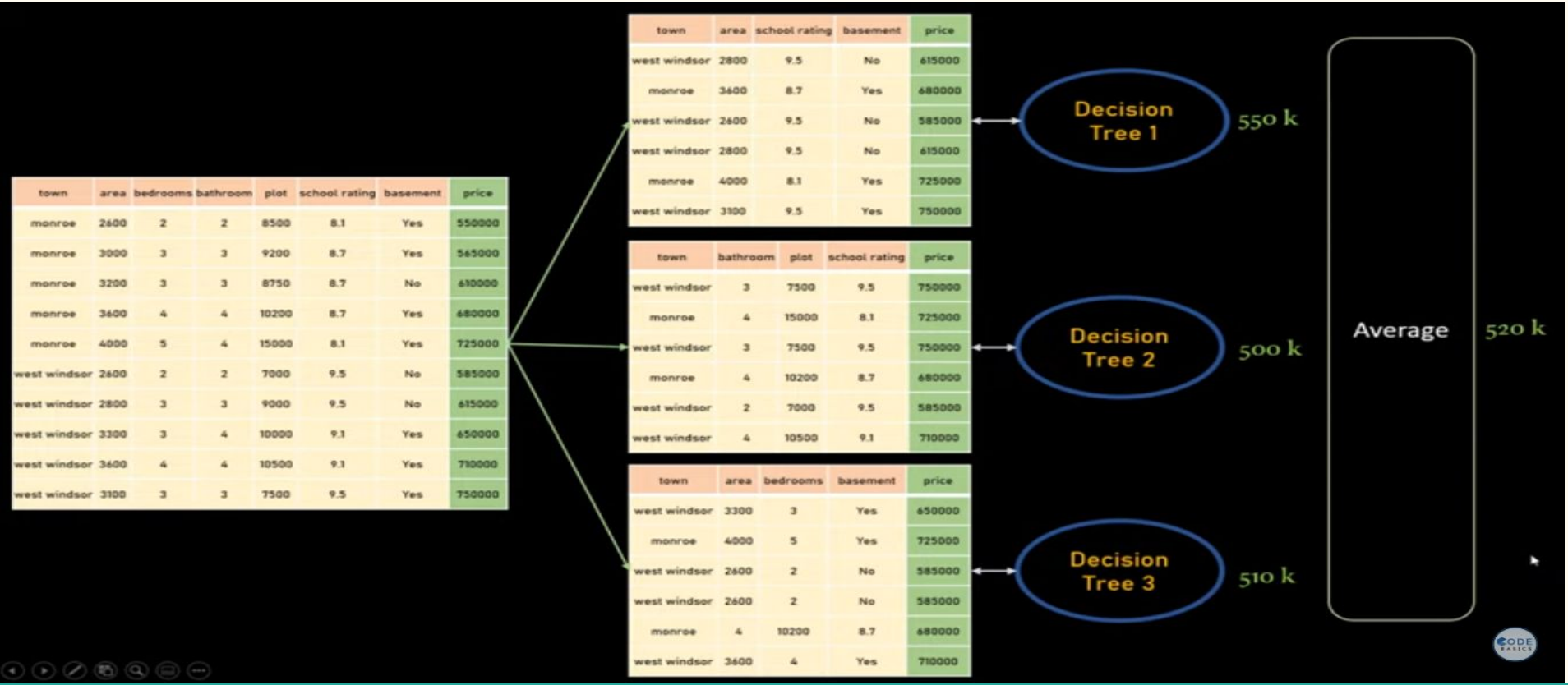
In Bagging, underlying model can be anything like SVM, KNN, Logistic regression, etc, where as in bagged tree, each model is a tree

https://youtu.be/RtrBtAKwcxQ?si=5VfZs37lP-ZEFpS9


100 samples
1
2
3
4
5
...
100
Resampling with redistribution
70 samples
4
9
4
19
...
35
87
4
21
...
59
94
1
63
...
13
M1
M2
M3
If a person would buy insurance or not
1
0
1
Majority Vote
1


1 2 3 4 5 ... 100
Bootstrap
4 9 4 19 ... 35
87 4 21 ... 59
94 1 63 ... 13
M1
M2
M3
550 k
500 k
510 k
Aggregation
Average
520 k

Bagging Technique - Random Forest



Dropout

https://youtu.be/lcI8ukTUEbo?si=XMHsz8h-50EsaWbB

Adversarial Training

Adversarial training is a machine learning technique that helps protect models from adversarial examples, which are inputs that are slightly altered in a way that is imperceptible to humans but can be misclassified by a machine learning model. The goal of adversarial training is to make models robust enough to withstand adversarial attacks, even if the attacker has full knowledge of the model.

https://www.javatpoint.com/adversarial-machine-learning

Tangent Distance, Tangent Prop, and Manifold Tangent Classifier

https://medium.com/invterate-learner/deep-learning-book-chapter-7-regularization-for-deep-learning-937ff261875c

Reference link

https://github.com/purvasingh96/Deep-learning-with-neural-networks/blob/master/Chapter-wise%20notes/Ch_6_Regularization_for_Deep_Learning/Readme.md covers entire syllabus

https://archive.nptel.ac.in/courses/106/106/106106184/ Deep Learning NPTEL course

https://www.youtube.com/watch?v=8tLhvAs-TLI – practical backpropagation

https://www.youtube.com/watch?v=mH9GBJ6og5A - theory backpropagation