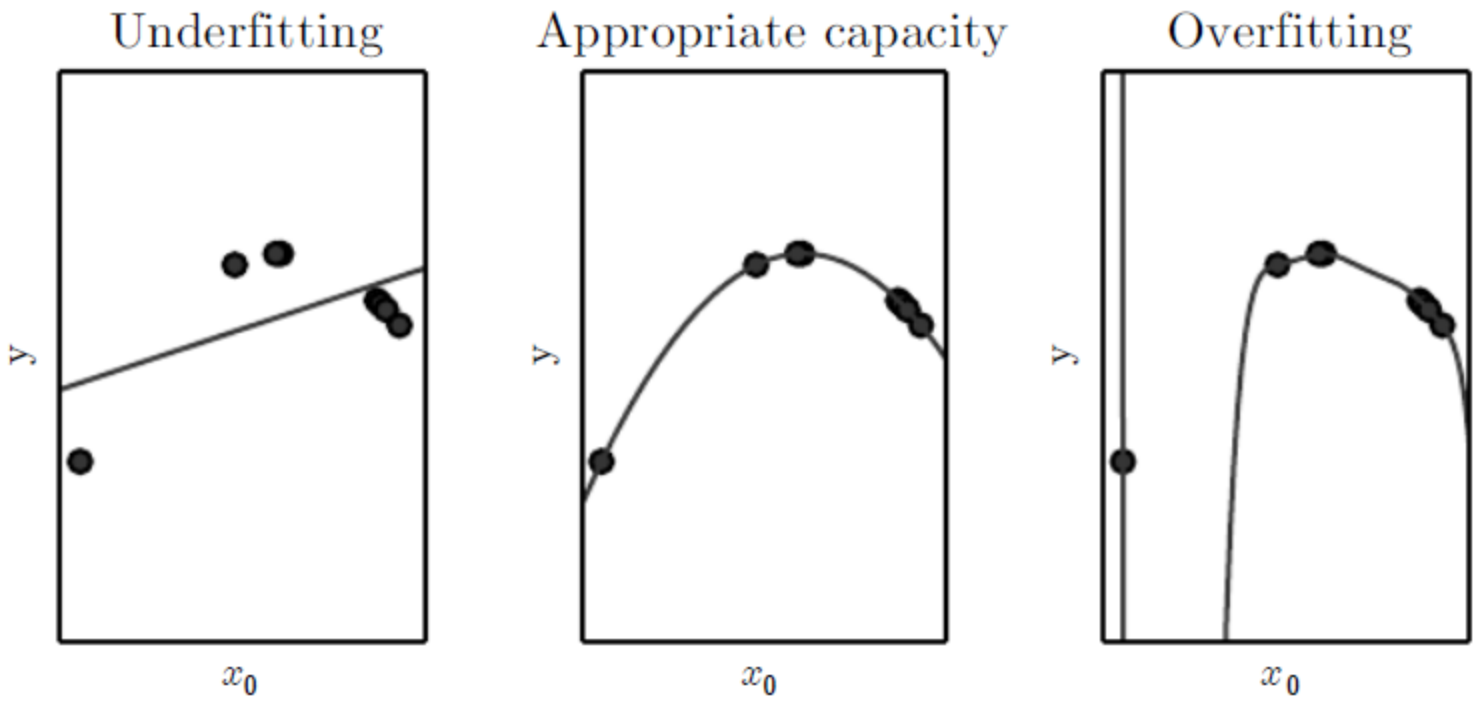
DEEP LEARNING
(CSE436)

BY: Nidhi S. Periwal,
Teaching Assistant,
DoCSE, SVNIT, Surat

Learning Algorithm

- A computer program is said to learn from **experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E**.
- The central challenge in machine learning is that we must perform well on *new, previously unseen* inputs—not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called **generalization**.
- When training a machine learning model, we have access to a training set, we can compute some error measure on the training set called the **training error**, and we reduce this training error. So far, what we have described is simply an optimization problem. What separates machine learning from optimization is that we want the **generalization error**, also called the **test error**, to be low as well.
- **The generalization error is defined as the expected value of the error on a new input.**

- Underfitting occurs when the model is not able to obtain a sufficiently low error value on the training set.
- Overfitting occurs when the gap between the training error and test error is too large.
- We can control whether a model is more likely to overfit or underfit by altering its **capacity**.



- **A model's capacity is its ability to fit a wide variety of functions. Models with low capacity may struggle to fit the training set.**
- **Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.**
- Machine learning algorithms will generally perform best when their capacity is appropriate for the true complexity of the task they need to perform and the amount of training data they are provided with.
- Models with insufficient capacity are unable to solve complex tasks. **Models with high capacity can solve complex tasks, but when their capacity is higher than needed to solve the present task they may overfit.**

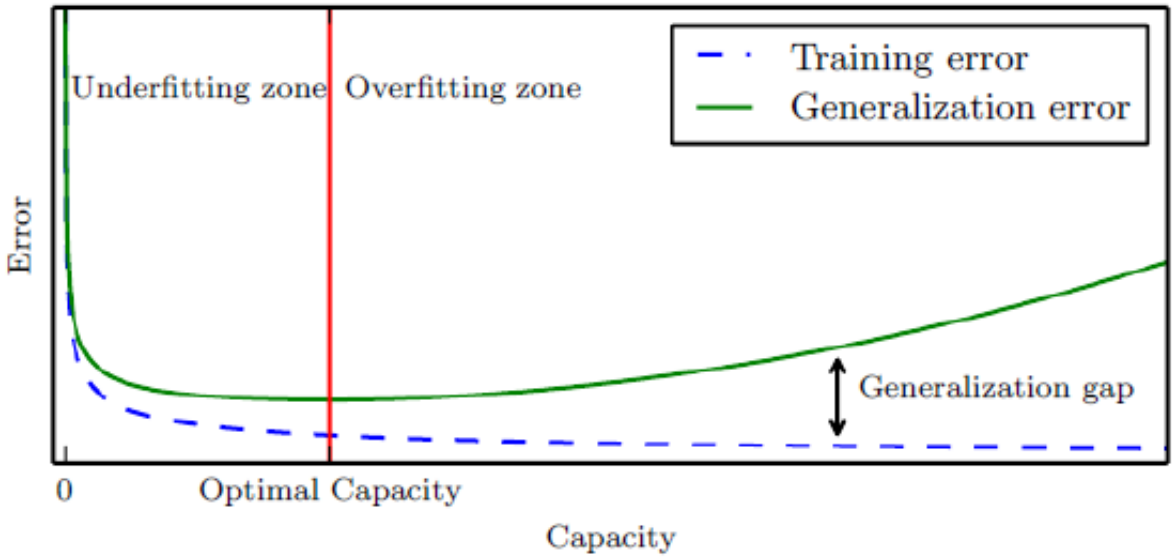


Figure 5.3: Typical relationship between capacity and error. Training and test error behave differently. At the left end of the graph, training error and generalization error are both high. This is the **underfitting regime**. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the **overfitting regime**, where capacity is too large, above the **optimal capacity**.

- **No free lunch theorem** for machine learning (Wolpert, 1996) states that, averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.
- **No machine learning algorithm is universally any better than any other**.
- ***Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.***
- The no free lunch theorem has made it clear that there is no best machine learning algorithm, and, in particular, no best form of regularization.

- The subset of data used to learn the parameters is still typically called the training set.
- The subset of data used to guide the selection of hyperparameters is called the validation set.
- One uses about 80% of the training data for training and 20% for validation.
- **Since the validation set is used to "train" the hyperparameters, the validation set error will underestimate the generalization error, though typically by a smaller amount than the training error.**
- After **all hyperparameter optimization is complete, the generalization error may be estimated using the test set.**

The key methods for avoiding overfitting in a neural network are as follows:

- **Penalty-based regularization**
  - The most common technique used by neural networks in order to avoid overfitting. The idea in regularization is to create a penalty or other types of constraints on the parameters in order to favor simpler models.
  - This will ensure simpler models.
  - However, since it is hard to impose such constraints explicitly, a simpler approach is to impose a softer penalty like $\lambda$and add it to the loss function.
  - Such an approach roughly amounts to multiplying each parameter $\boldsymbol{w}$ with a multiplicative decay factor of $(1 - \alpha\lambda)$ before each update at learning rate α.
  - Aside from **penalizing** parameters of the network, one can also choose to penalize the activations of hidden units. This approach often leads to sparse hidden representations

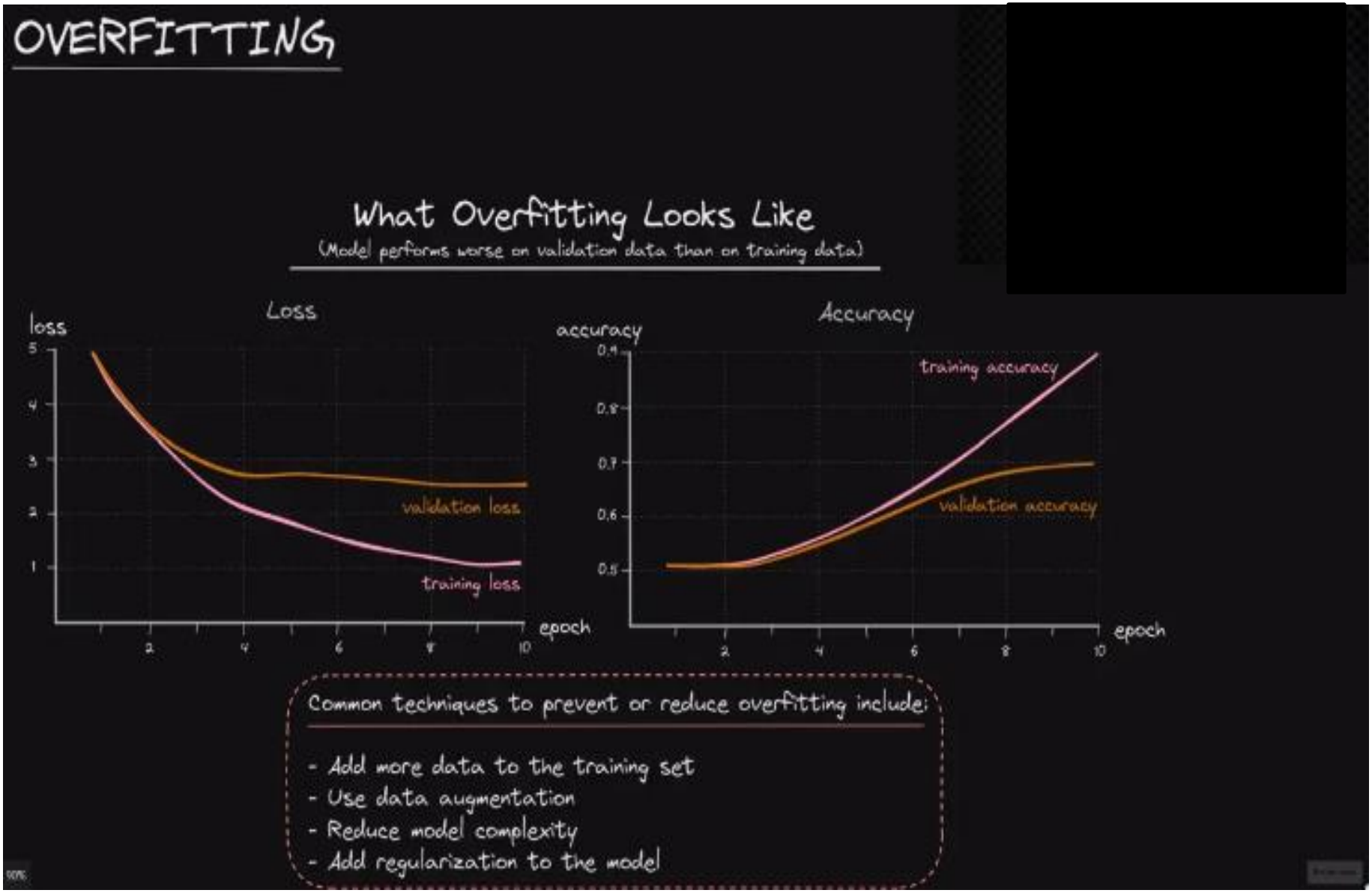
Generic and tailored ensemble methods:

- A straightforward approach is to average the predictions of different neural architectures obtained by quick and dirty hyper-parameter optimization.
- ***Dropout*** **is ensemble technique that is designed for neural networks.**
- This technique uses the selective dropping of nodes to create different neural networks.
- The predictions of different networks are combined to create the final result. Dropout reduces overfitting by indirectly acting as a regularizer.

- **Early stopping:**
- In early stopping, the iterative optimization method is terminated early without converging to the optimal solution on the training data.
- **The stopping point is determined using a portion of the training data that is not used for model building.**
- One terminates when the error on the held-out data begins to rise. **Even though this approach is not optimal for the training data, it seems to perform well on the test data because the stopping point is determined on the basis of the held-out data.**

- **Pretraining:**
- Pretraining is a form of learning in which a greedy algorithm is used to find a good initialization.
- The weights in different layers of the neural network are trained sequentially in greedy fashion.
- These trained weights are used as a good starting point for the overall process of learning. Pretraining can be shown to be an indirect form of regularization.

OVERFITTING



Regularization

- When learning model, emulates the training data closely and performs very well w.r.t. to training data but fails to perform w.r.t. test data, the model is said to be over-fitted model.
- Weights in NN are crucial parameter which lead to the performance of model on test data.
- Hence, if the weights are rightly assigned, the network will generalized well and give right prediction of test data.
- **However, in case of over-fitting, the weight values are assigned in such a way that it gives accurate predictions for training data but for test data predictions are grossly inaccurate.**
- **One potential symptom of such an over-fitted network is when certain values have excessively large values.**
- **A small change in those inputs may lead to a large change in the value of the output, which is not a good example of generalization model.**

Regularization

- **It is used to prevent the high value of weights.**
- **The learning algorithm should try to keep the weights small, which can be achieved by regularization.**
- **Regularization attempts to penalize large weights by increasing loss values.**
- The optimization algorithm **try to minimize the loss – they try to decrease the weights of the network.**
- Since, **the intent is to penalize higher weights & make them smaller & more regular weights, the process is known as regularization.**

Regularization

Penalty-based regularization

- Penalize the *parameters* of the neural network.
- Penalty-based regularization is the most common approach for reducing overfitting.

$$\hat{y} = \sum_{i=0}^{d} w_i x^i$$

- Eqn: Single-layer network with *d* inputs and a single bias neuron with weight w_0 in order to model this prediction.
- The i^{th} input is x_i. This neural network uses linear activations, and the squared loss function for a set of training instances (*x, y*) from data set *D* can be defined as follows:

$$L = \sum_{(x,y)\in\mathcal{D}} (y - \hat{y})^2$$

- **A large value of *d* tends to increase overfitting.**
- **One possible solution to this problem is to reduce the value of *d*.** In other words, using a model with *economy in parameters* leads to a simpler model.
- Reducing ***d* to 1 creates a linear model** that has fewer degrees of freedom and tends to fit the data in a similar way over different training samples. **However, doing so does lose some expressivity when the data patterns are indeed complex**.
- In other words**, oversimplification reduces the expressive power of a neural network, so that it is unable to adjust sufficiently to the needs of different types of data sets.**

- Instead of reducing the number of parameters in a hard way, one can use a ***soft* penalty on the use of parameters**
- **Large (absolute) values of the parameters are penalized more than small values, because small values do not affect the prediction significantly.**
- The most common choice is *L*2-regularization, which is also referred to as ***Tikhonov** regularization*.
- It uses squared norm penalty, is the most common approach for regularization.

L_2-Regularization

$$L = \sum_{(x,y)\in\mathcal{D}} (y - \hat{y})^2 + \lambda \cdot \sum_{i=0}^{d} w_i^2$$

- Increasing or decreasing the value of $\boldsymbol{\lambda}$ **reduces the softness of the penalty.**
- One advantage of this type of parameterized penalty is that **one can tune this parameter for optimum performance on a portion of the training data set that is not used for learning the parameters.** This type of approach is referred to as *model validation*.

- For any given weight w_i in the neural network, the updates are defined by gradient descent.

$$w_i \Leftarrow w_i - \alpha \frac{\partial L}{\partial w_i}$$

- Here, α is the learning rate. The use of $L2$-regularization is roughly equivalent to the use of decay imposition after each parameter update

$$w_i \Leftarrow w_i(1 - \alpha\lambda) - \alpha \frac{\partial L}{\partial w_i}$$

- The update above first multiplies **the weight with the decay factor $(1 - \alpha\lambda)$,** and then uses the gradient-based update.
- If we assume that the initial values of the weights are close to 0. One can view weight decay as a kind of **forgetting mechanism**, which brings the weights closer to their initial values. This ensures that only the repeated updates have a significant effect on the absolute magnitude of the weights.
- **A forgetting mechanism prevents a model from *memorizing* the training data,** because only significant and repeated updates will be reflected in the weights.

L_1-Regularization

- L_1-regularization uses a penalty on the sum of the absolute magnitudes of the coefficients. The new objective function is as follows:

$$L = \sum_{(x,y)\in\mathcal{D}} (y - \hat{y})^2 + \lambda \cdot \sum_{i=0}^{d} |w_i|_1$$

- Weight update equation:

$$w_i \Leftarrow w_i - \alpha\lambda s_i - \alpha\frac{\partial L}{\partial w_i}$$

The value of s_i, which is the partial derivative of $|w_i|$ (with respect to w_i), is as follows:

$$s_i = \begin{cases} -1 & w_i < 0 \\ +1 & w_i > 0 \end{cases}$$

- For the rare cases in which the value w_i is exactly 0, one can omit the regularization and simply set s_i to 0

L2 vs L1

- *L*2-regularization uses multiplicative decay as a forgetting mechanism, whereas *L*1-regularization uses additive updates as a forgetting mechanism.
- In both cases, the regularization portions of the updates tend to move the coefficients closer to 0.
- From an accuracy point of view, *L*2-regularization usually outperforms *L*1-regularization

- An interesting property of *L*1-regularization is that it creates *sparse* solutions in which the vast majority of the values of w_i are 0s (after ignoring computational errors).
- If the value **of w_i is zero for a connection incident on the input layer, then that particular input has no effect on the final prediction. In other words, such an input can be *dropped* and the *L*1-regularizer acts as a feature selector.**
- Therefore, one can use *L*1-regularization to estimate which features are predictive to the application at hand.
- These connections can be dropped, which results in a sparse neural network. Such sparse neural networks can be useful in cases where one repeatedly performs training on the same type of data set, but the nature and broader characteristics of the data set do not change significantly with time.
- Since **the sparse neural network will contain only a small fraction of the connections in the original neural network, it can be retrained much more efficiently whenever more training data is received**.

Regularization

- **L_1/L_2 Regularization:**
  - **If loss function is assumed to be L(w,b), the objective of any optimization algorithm is to minimize the value of loss function.**
  - For **L_1 /LASSO/ Linear or logistics regression , the sum of absolute values of the weights is added to the loss function.**
  - For L_2/ Ridge regression for linear or logistic regression/, the sum of **squared** values of weights is added to the loss function.

  - Lambda is regularization parameter, which is hyper-parameter.

- Extreme forms of overfitting are referred to as *memorization.*
- The ability of a learner to provide useful predictions for instances it has not seen before is referred to as *generalization.*

Penalizing Hidden Units

- An approach is to **penalize the *activations* of the neural network,** so that only a small subset of the neurons are activated for any given data instance.
- **Even though the neural network might be large and complex only a small part of it is used for predicting any given data instance.**
- The simplest way to achieve sparsity is to impose an *L*1-penalty on the hidden units. The original loss function *L* is modified to the regularized loss function *L* as follows:

$$L' = L + \lambda \sum_{i=1}^{M} |h_i|$$

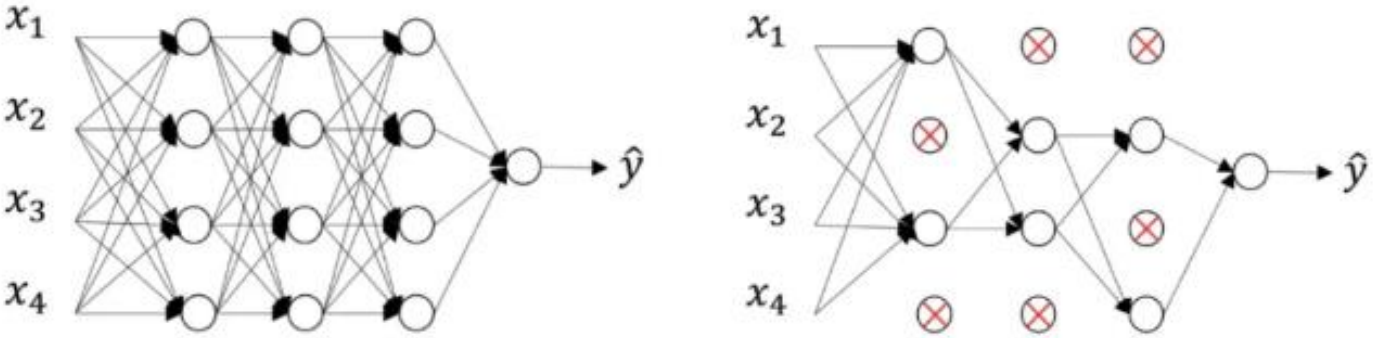
- *M* is the total number of units in the network, and h_i is the value of the i^{th} hidden unit. The regularization parameter is denoted by λ.
- In many cases, a single *layer* of the network is regularized, so that a sparse feature representation can be extracted from the activations of that particular layer

Randomized Connection Dropping

- The random dropping of connections between different layers in a multilayer neural network often leads to diverse models in which different combinations of features are used to construct the hidden variables.
- **Edge Sampling-** The dropping of connections between layers does tend to create less powerful models because of the addition of constraints to the model-building process.
- However, since different random connections are dropped from different models, the predictions from different models are very diverse.
- The averaged prediction from these different models is often highly accurate.
- It does not share any weights between ensemble components.

Dropout

- The strategy applied for dropout is simple.
- Dropout means that during training with some probability **P** a neuron of the neural network gets turned off during training.
- Assume on the left side we have a feedforward neural network with no dropout.
- Using dropout with let's say a probability of **P=0.5** that a random neuron gets turned off during training would result in a neural network on the right side.



Regularization

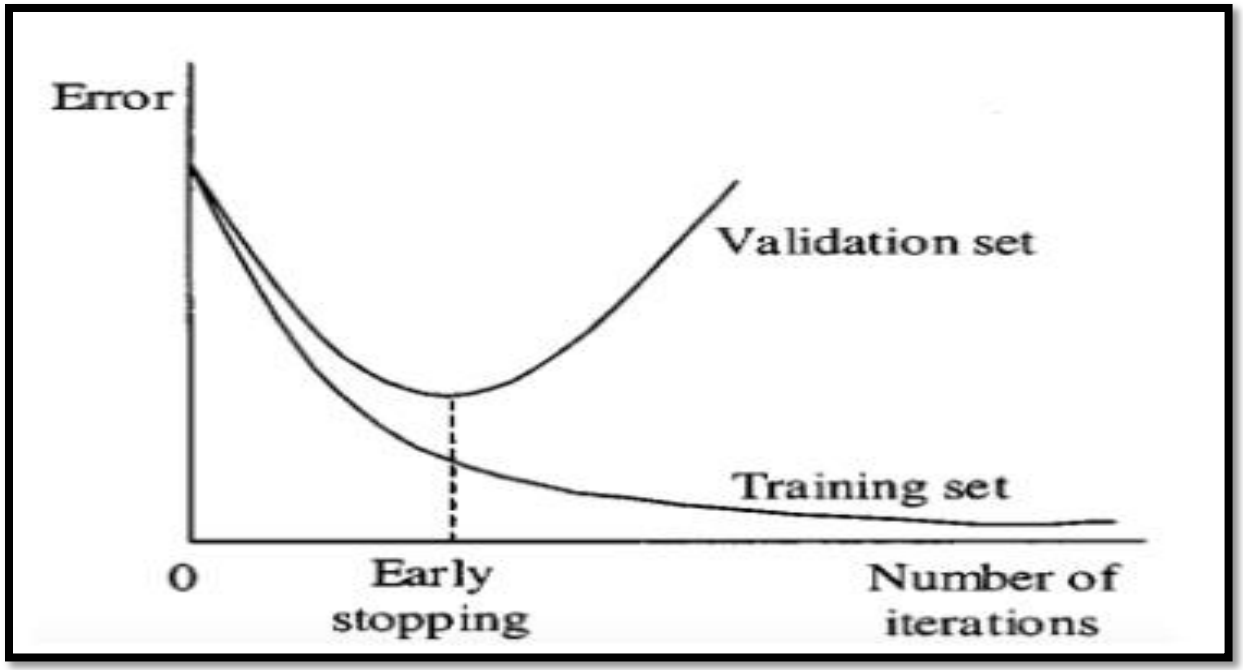
Dropout

- In this case, you can observe that approximately half of the neurons are not active and are not considered as a part of the neural network. And as you can observe the neural network becomes simpler.
- **A simpler version of the neural network results in less complexity that can reduce overfitting. The deactivation of neurons with a certain probability P is applied at each forward propagation and weight update step.**

Regularization

Early Stopping

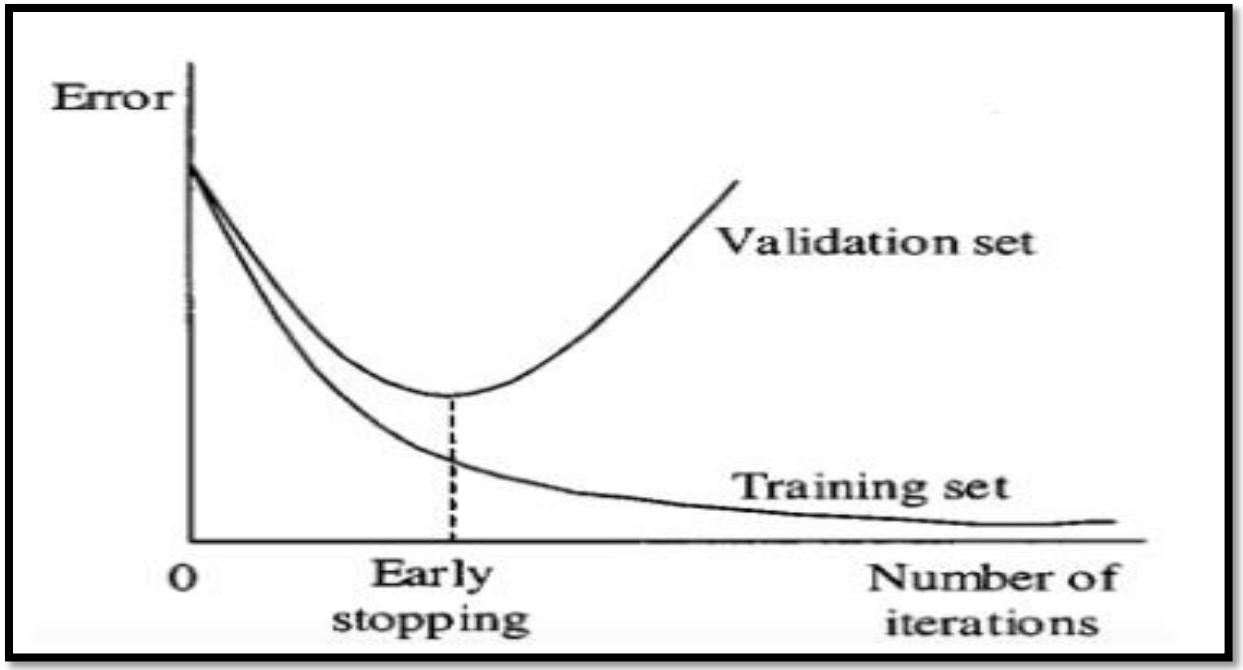
- If the training runs too long, i.e. the number of epochs are too high, the model tends to overfit.
- The model is run on the training and validation data simultaneously.
- Initially, both the training and validation loss keeps coming down.



Regularization

Early Stopping

- However, after a certain number of iterations or epochs, the validation loss will start increasing, whereas the training loss keeps decreasing.
- If the model training stops at this point, the possibility of model getting over fitted in training data is addressed. That is called early stopping and acts as a very effective regularizer.



Regularization

Early Stopping

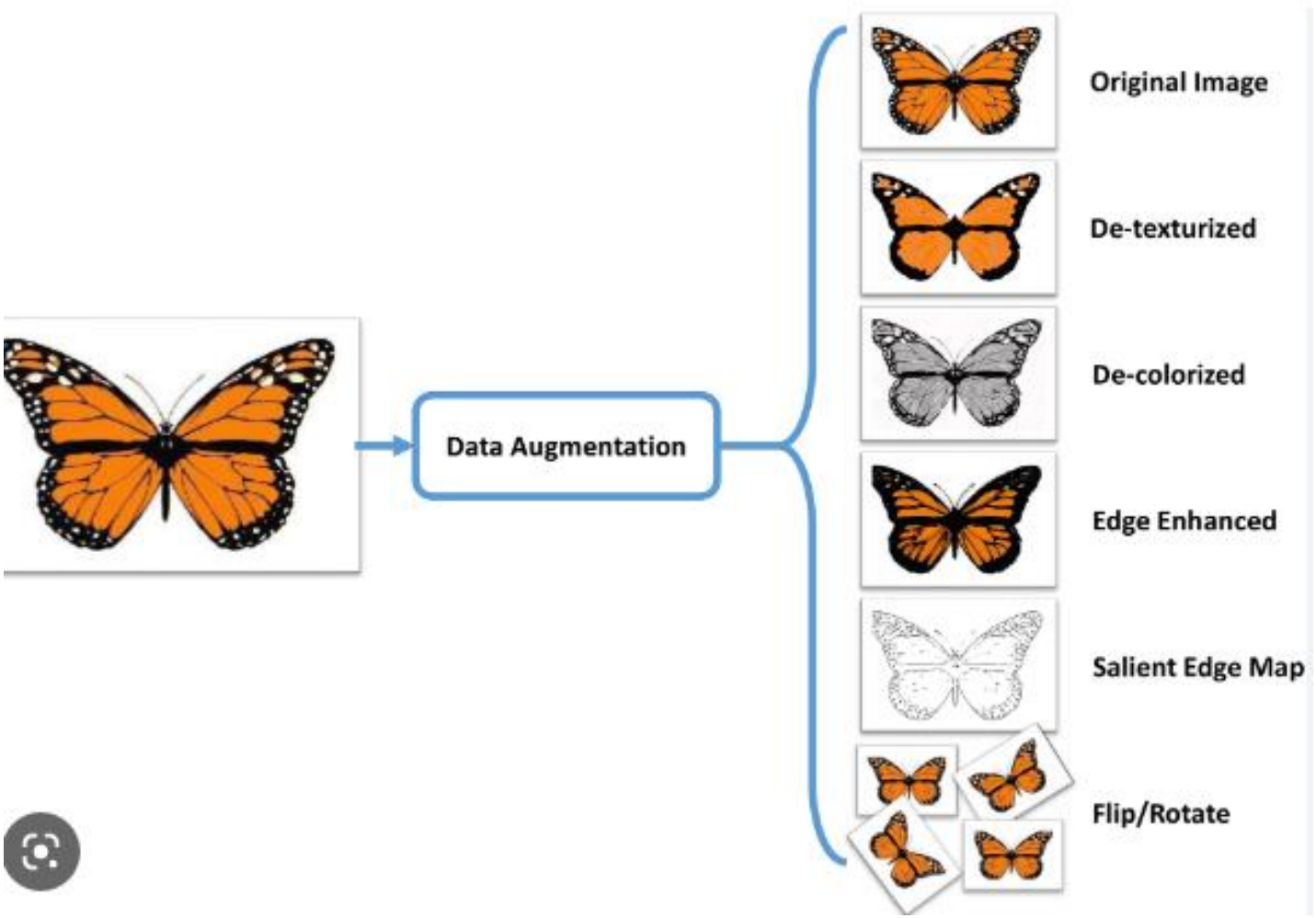
- A regularization technique for **deep neural networks that stops training when parameter updates no longer begin to yield improves on a validation set**.
- In essence, we store and update the current best parameters during training, and when parameter updates no longer yield an improvement (after a set number of iterations) we stop training and use the last best parameters.
- It works as a regularizer by restricting the optimization procedure to a smaller volume of parameter space.

Data Augmentation

- Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points.
- **Augmented vs. Synthetic data**
  - **Augmented data** is driven from original data with some minor changes. In the case of image augmentation, we make geometric and color space transformations (flipping, resizing, cropping, brightness, contrast) to increase the size and diversity of the training set.
  - **Synthetic data** is generated artificially without using the original dataset. It often uses DNNs (Deep Neural Networks) and GANs (Generative Adversarial Networks) to generate synthetic data.
  - **Note**: The augmentation techniques are not limited to images. You can augment audio, video, text, and other types of data too

Data Augmentation for image

- Commonly used in computer-vision related problems.
- **The training data is appended with more variations of original data in such a way that the number of instances of data is proportionate with number of parameters to be trained.**
- **Rotation**: New images are generated by rotating the original image clock-wise or anti-clockwise to left or to the right.
- **Cropping**: New images are generated by taking portions of the original images.
- **Re-sizing**: New images are generated by changing the size or re-scaling the original image.


Data Augmentation
Original Image
De-texturized
De-colorized
Edge Enhanced
Salient Edge Map
Flip/Rotate

Data Augmentation Uses

- To prevent models from overfitting.
- The initial training set is too small.
- To improve the model accuracy.
- To Reduce the operational cost of labeling and cleaning the raw dataset.

Limitations of Data Augmentation

- **Quality assurance for data augmentation is expensive.**
- Research and development are required to build a system with advanced applications.
- Finding an effective data augmentation approach can be challenging.

Audio Data Augmentation

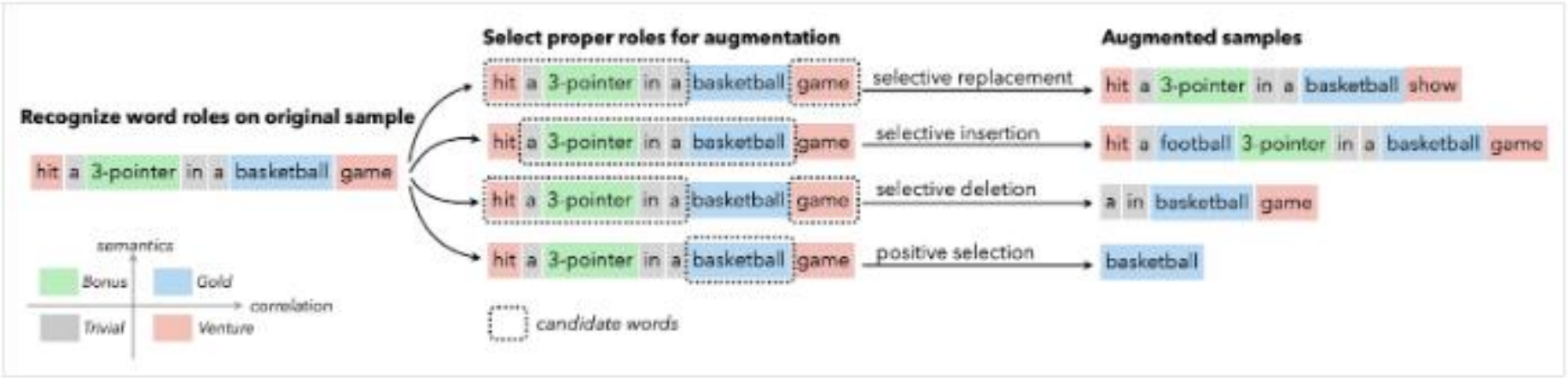
- **Noise Injection**: Add gaussian or random noise to the audio dataset to improve the model performance.
- **Shifting**: Shift audio left (fast forward) or right with random seconds.
- **Changing the speed**: Stretches times series by a fixed rate.
- **Changing the pitch**: Randomly change the pitch of the audio.

Text Data Augmentation

- **Word or sentence shuffling**: randomly changing the position of a word or sentence.
- **Word replacement**: replace words with synonyms.
- **Syntax-tree manipulation**: paraphrase the sentence using the same word.
- **Random word insertion**: inserts words at random.
- **Random word deletion**: deletes words at random.

Natural Language Processing

- Text data augmentation is generally used in situations with limited quality data, and improving the performance metric takes priority.
- Synonym augmentation, word embedding, character swap, and random insertion and deletion can be applied.
- These techniques are also valuable for low-resource languages.



Thank You!