MLAI 504

NEURAL NETWORK AND DEEP LEARNING

Dr. Zein Al Abidin IBRAHIM

zein.ibrahim@ul.edu.lb

NEURAL NETWORKS: A QUICK OVERVIEW

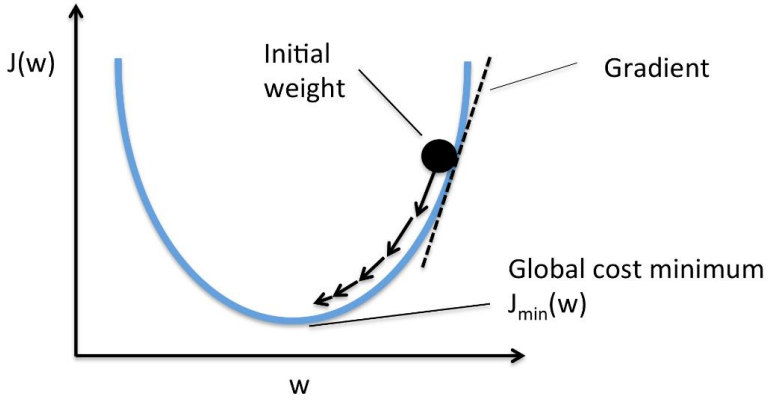
DEEP LEARNING

COST FUNCTIONS AND OPTIMIZATION

NEURAL NETWORK

GRADIENT DESCENT - RECAP

- One technique that can be used for minimising functions is **gradient descent**.
- Can we use this on our **error function** E?



We would like a learning rule that tells us how to update weights, like this:

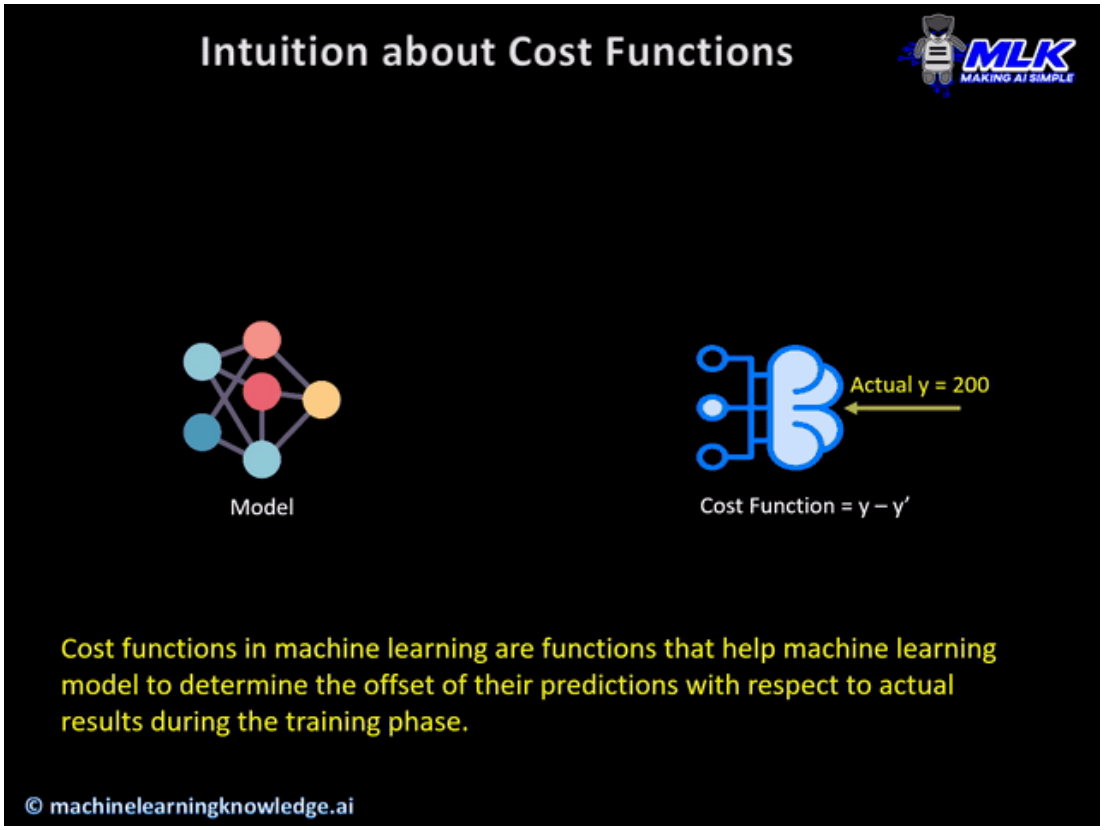
$$w'_{ij} = w_{ij} + \Delta w_{ij}$$

But what should Δw_{ij} be?

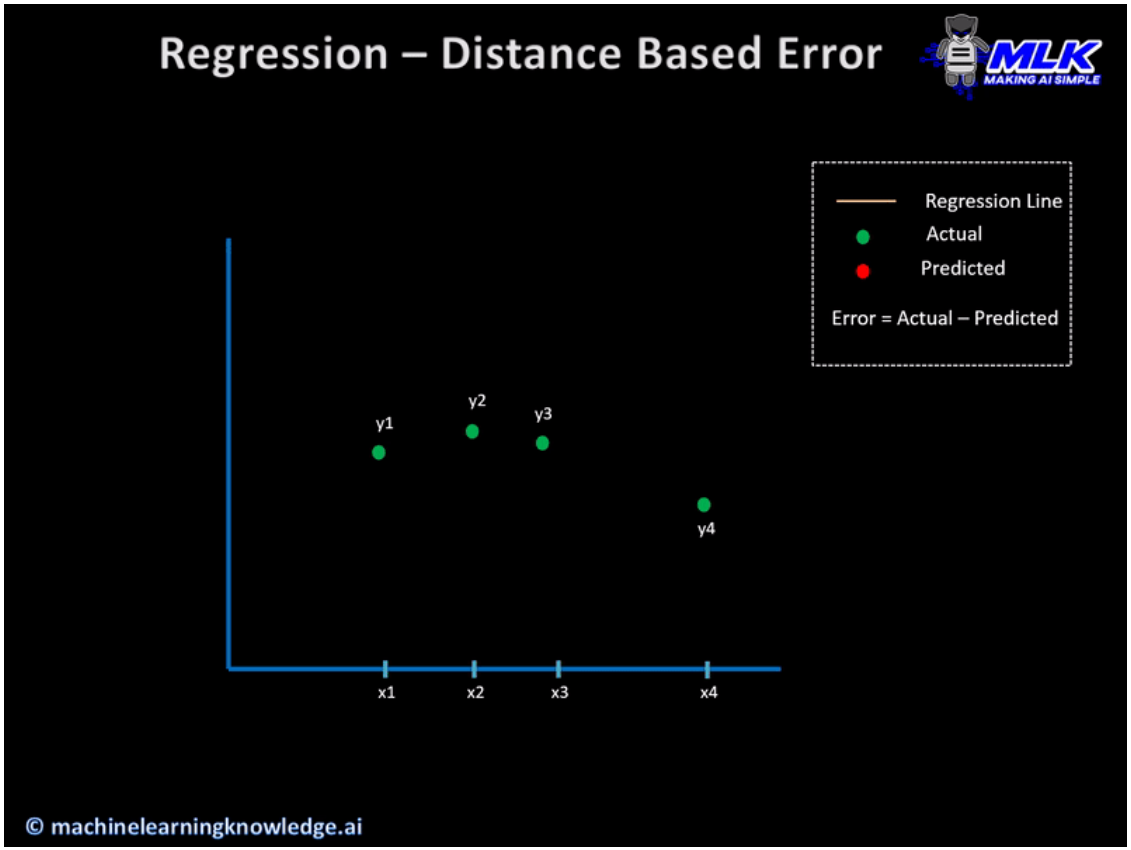
OBJECTIVE FUNCTIONS

- also known as loss functions or cost functions
- measure difference between the predicted outputs of the network and the actual desired outputs
- aim to minimize the objective function
- Purpose
  - Quantify prediction error
  - Guide network in the training process
- Common Objective Functions
  - Mean Squared Error (MSE) → commonly used for regression
  - Cross Entropy Loss or log loss → commonly used for classification
  - Binary Cross Entropy Loss → like 2 but when output is binary
  - Mean Absolute Error (MAE) → like MSE for regression tasks
  - Sparse Categorical Cross Entropy
  - Kullback-Leibler (KL) Divergence

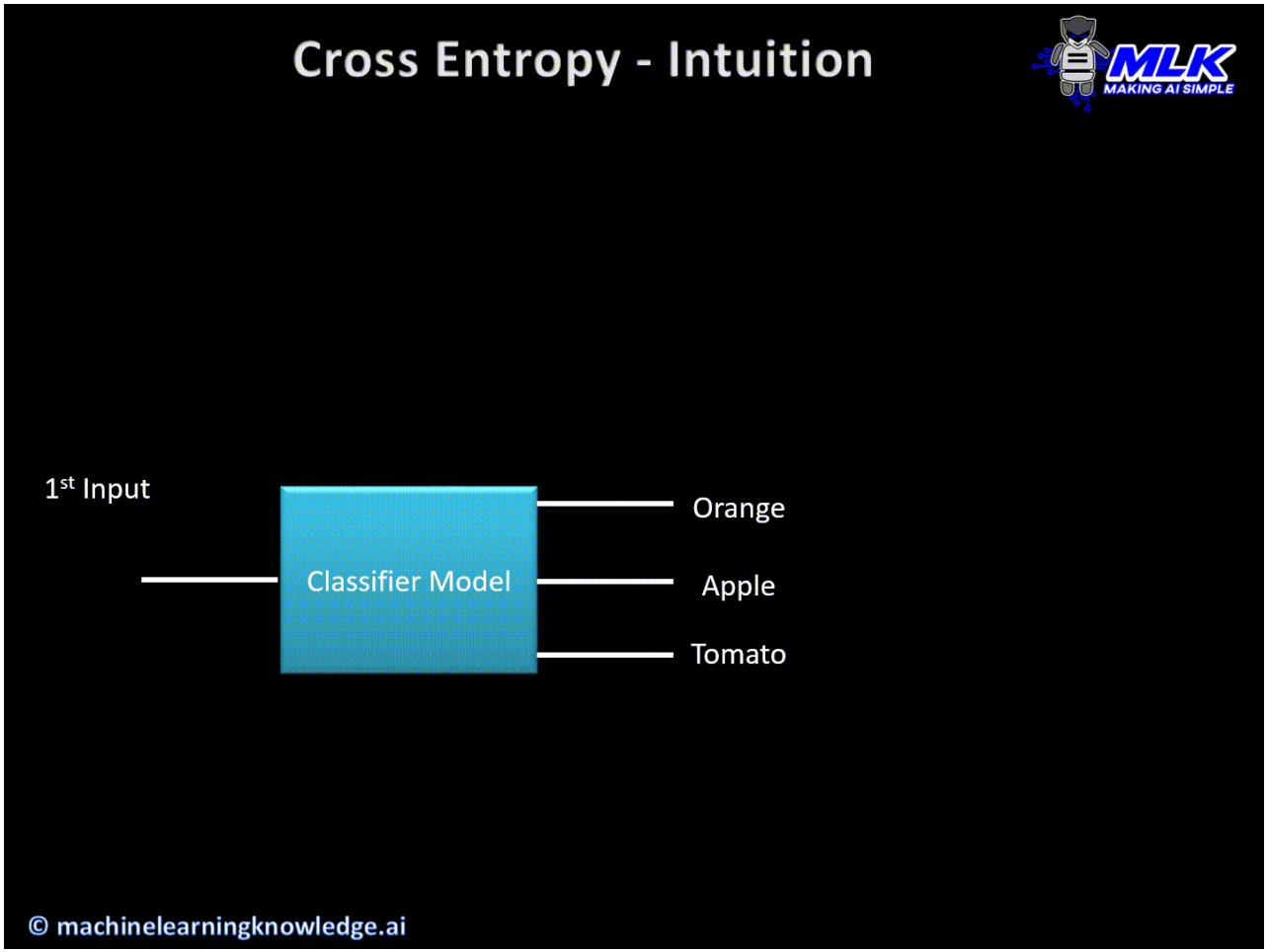
OBJECTIVE FUNCTIONS



OBJECTIVE FUNCTIONS



OBJECTIVE FUNCTIONS



OBJECTIVE FUNCTIONS

- MSE $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$
- MAE $$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$
- Cross Entropy $$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} log(\hat{y}_{ij})$$
- Binary Cross Entropy $$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

LEARNING APPROACHES

- **Online learning** → weights updated after processing each sample
  - Useful when training dataset is large
  - Less storage requirement
  - Take advantage of redundancy in the data
  - Tracks small changes in data
  - Simple to implement
- **Stochastic learning** → variant of online learning where order of samples is randomized and weights updated after each sample.
  - Prevent network from stacking into local optima but less stable than other approaches

LEARNING APPROACHES

- **Batch learning** → weights updated after processing all samples.
  - more computationally efficient than online learning (can be done in parallel)
  - can result in smoother convergence to an optimal solution.
- **Mini-batch learning** → compromise between online and batch. Weights updated after processing a small subset of the samples.
  - more computationally efficient and can still allow for updates that are more fine-grained than those in batch learning.

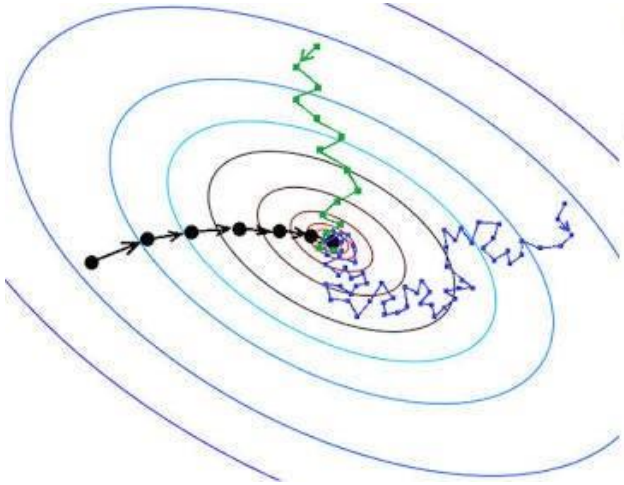
ONLINE/STOCHASTIC BACKPROPAGATION

1: Initialize all weights to small random values.
2: **repeat**
3: **for** each training example **do**
4: Forward propagate the input features of the example to determine the MLP's outputs.
5: Back propagate the error to generate Δw_{ij} for all weights w_{ij}.
6: Update the weights using Δw_{ij}.
7: **end for**
8: **until** stopping criteria reached.

BATCH BACKPROPAGATION

1: Initialize all weights to small random values.
2: **repeat**
3: **for** each training example **do**
4: Forward propagate the input features of the example to determine the MLP's outputs.
5: Back propagate the error to generate Δw_{ij} for all weights w_{ij}.
6: **end for**
7: Update the weights based on the accumulated values Δw_{ij}.
8: **until** stopping criteria reached.

LEARNING



Batch GD
- Slowest
- Perfect gradient

Stochastic GD
- Fastest
- Rough-estimate grad

Mini-batch GD
- Compromise

OPTIMIZER

- There are various improved versions of gradient descent algorithms. In object-oriented language implementation, different gradient descent algorithms are often encapsulated into an object which is called an **optimizer**.
- The **purpose** of algorithm improvement includes but is not limited to:
  - ✓ Accelerates algorithm convergence.
  - ✓ Avoids or overshoots local extrema.
  - ✓ Simplifies manual parameter setting, especially the learning rate.
- Common optimizers: common **GD optimizer**, momentum optimizer, Nesterov, Adagrad, Adadelta, **RMSprop**, **Adam**, AdaMax, and Nadam

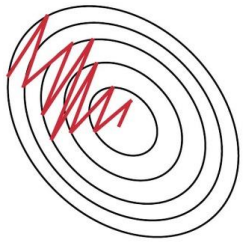
MOMENTUM OPTIMIZER

- The backpropagation algorithm provides a trajectory in the weight space computed by the method of the steepest descent.
- The smaller the parameter η, the smaller the changes to the synaptic weights in the network.
- If η is too large to speed up the rate of learning then the resulting changes in the synaptic weight may become unstable (i.e. Oscillatory)
- As simple method of increasing the rate of learning while avoiding the danger of instability is to modify the delta rule by including a momentum term.
- This is called the generalized delta rule

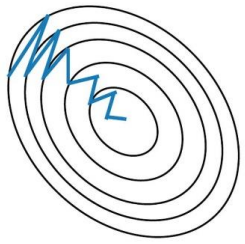
$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

ADVANTAGES AND DISADVANTAGES OF MOMENTUM OPTIMIZER

- Advantages:
  - ✓ Enhances the stability of the gradient correction direction and reduces Oscillations.
  - ✓ In areas where the gradient direction is stable, the ball rolls faster and faster (there is a speed upper limit because $\alpha < 1$), which helps the ball quickly overshoot the flat area and accelerates convergence.
  - ✓ A small ball with inertia is more likely to roll over some narrow local extrema.
- Disadvantage:
  - ✓ The learning rate η and momentum α need to be manually set, which often requires more experiments to determine the appropriate value.



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

ADAM OPTIMIZER

- Adaptive Moment Estimation (Adam): Developed based on Adagrad and Adadelta, Adam maintains two additional variables m_t and v_t for each variable to be trained:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where t represents t-th iteration and g_t is the calculated gradient. m_t and v_t are moving averages of the gradient and the squared gradient. From the statistical perspective, m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method.

Adam attempts to compute adaptive learning rates for each parameter.

ADAM OPTIMIZER

- As m_t and v_t are initialized as vectors of 0's, they are biased towards 0, especially during the initial steps, and especially when β_1 and β_2 are close to 1. To solve this problem, we use $\hat{m}_t$ and $\hat{v}_t$:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

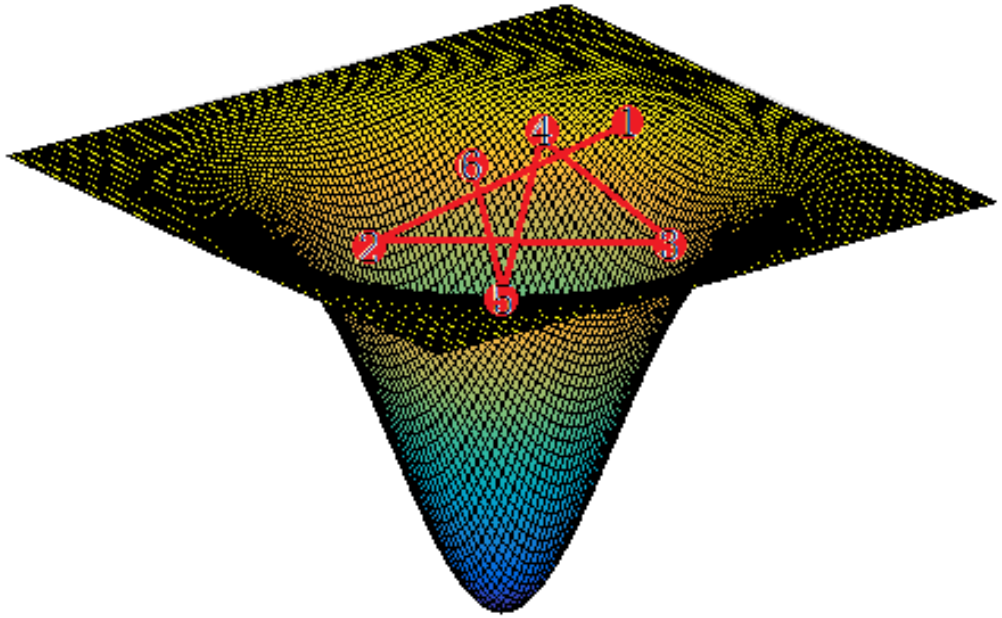
- Adam's weight update rule:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

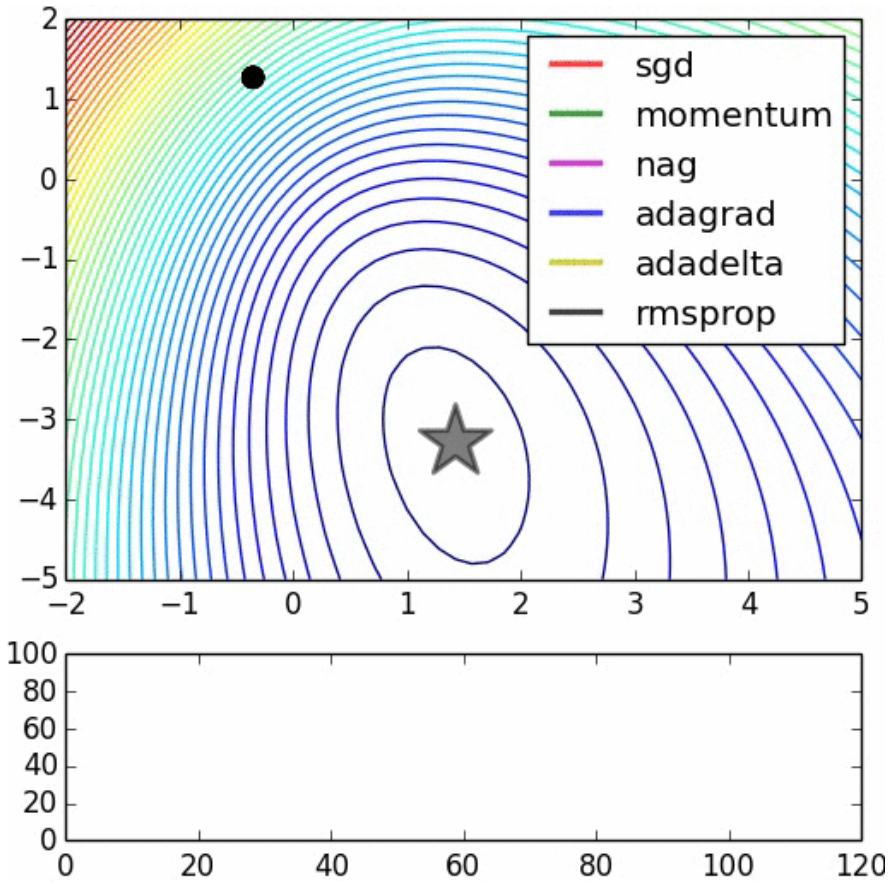
- Although the rule involves manual setting of η, β_1, and β_2, this is much simpler. According to experiments, the default setting are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and η = 0.001. In practice, Adam will converge quickly. For convergence saturation, reduce η. After several times of reduction, the satisfying local extrema will be converged. Other parameters do not need adjustment.

ADAM OPTIMIZER

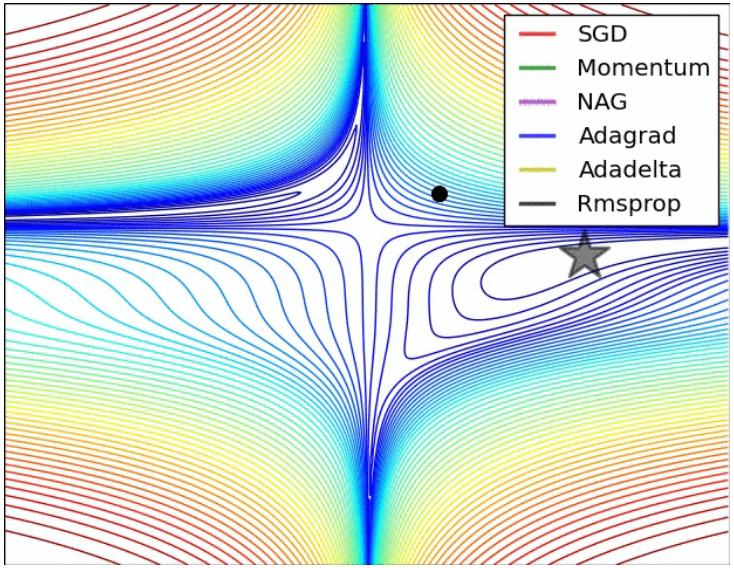
- $\widehat{m}_t$ is the current gradient with a momentum term and $\frac{\eta}{\sqrt{\widehat{v}_t}+\epsilon}$ is equivalent to the current learning rate. If the gradient model is too large, the weight cannot jump out of the extrema or converge. Then we need to reduce the learning rate to facilitate convergence.



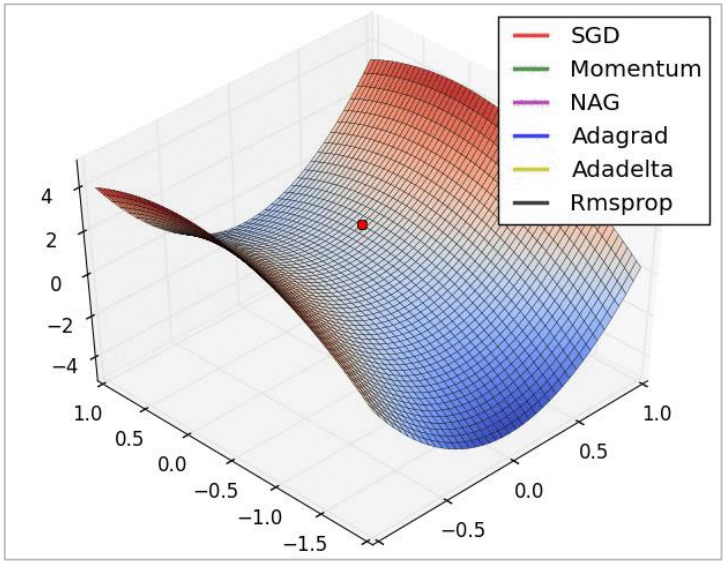
VISUALIZED COMPARISON OF OPTIMIZERS



VISUALIZED COMPARISON OF OPTIMIZERS



Comparison of optimization algorithms in contour maps of loss functions



Comparison of optimization algorithms at the saddle point

LEARNING

Gradient Descent Variant	Advantages	Disadvantages
Batch Gradient Descent	Thorough consideration of all data points, stable convergence	Computationally expensive for large datasets
Stochastic Gradient Descent (SGD)	Faster than batch gradient descent, less prone to local minima	Noisy convergence, may require more iterations
Mini-batch Gradient Descent	Balances speed of SGD with stability of batch gradient descent	Requires careful selection of mini-batch size
Momentum-Based Gradient Descent	Helps accelerate convergence and avoid local minima, improves stability	Requires careful selection of momentum coefficient β
RMSprop	Adapts learning rate for each parameter, reduces oscillations, improves stability	Requires careful selection of momentum coefficient β and smoothing constant, computationally expensive

WHEN TO STOP LEARNING

- Several stopping criteria
  - **Number of epochs**: Maximum number of epochs reached
  - **Early stopping**: validation error starts to increase after it has reached a minimum → use cross-validation to perform that.
  - **Plateaued performance**: performance on validation set does not improve beyond a certain threshold for a specified number of epochs.
  - **Minimum error threshold**: training error or validation error falls below a certain threshold.
  - **Computing resource constraints**: limited computing resources.
  - **Convergence of weights**: weights converge and do not change significantly over a certain number of epochs.

HEURISTICS FOR MAKING BP PERFORM BETTER

- **Stochastic versus batch update:**
  - Stochastic (sequential) mode is computationally faster per iteration than batch mode but need more iteration
    - Especially when training data sample is large and highly redundant.
- **Maximizing information content:**
  - As a general rule every training example of the BP algorithm should be chosen on the basis that its information content is the largest possible for the task at hand
  - To realize this choice we have two ways:
    - Use an example that results in the largest training error.
    - Use an example that is radically different from all those previously used.
  - This is because it drives the algorithm to search more in the weight space.
    - Example: shuffle examples between two epochs

CHOICE OF ACTIVATION FUNCTION

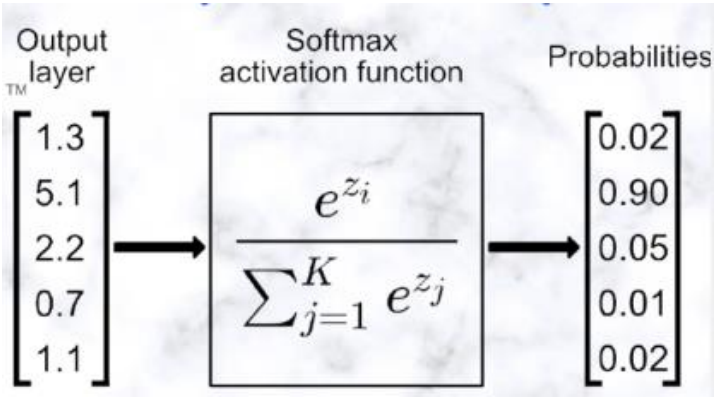
Function	Function	Derivative
ReLU (Rectified Linear Unit)	$\phi(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$	$\phi'(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$
Leaky ReLU	$\phi(x) = \begin{cases} \alpha x & x \leq 0 \\ x & x > 0 \end{cases}$	$\phi'(x) = \begin{cases} \alpha & x \leq 0 \\ 1 & x > 0 \end{cases}$
ELU (Exponential Linear Unit)	$f(x) = \begin{cases} \alpha(e^x - 1) \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
Tanh	$\phi(x) = \frac{2}{1+e^{-2x}} - 1$	$\phi'(x) = 1 - \phi(x)^2$
Sigmoid	$\phi(x) = \frac{1}{1+e^{-x}}$	$\phi'(x) = \phi(x)(1 - \phi(x))$
Softmax	$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$	$\frac{\partial S(z_i)}{\partial z_j} = \begin{cases} S(z_i) \times (1 - S(z_i)) & \text{if } i = j \\ -S(z_i) \times S(z_j) & \text{if } i \neq j \end{cases}$

CHOICE OF ACTIVATION FUNCTION

Function	Advantages	Disadvantages
ReLU (Rectified Linear Unit)	• Efficient gradient flow • Simple and computationally efficient • Works well for deep networks	• Outputs are not bounded • Can lead to "dead neurons"
Leaky ReLU	• Non-zero gradient for negative values • Prevents dead neurons • Suitable for deep networks	• Less efficient than ReLU • May require careful tuning of the leak factor
ELU (Exponential Linear Unit)	• Smooth, continuous output • Fast activation and good gradient flow • Suitable for deep networks	• More computationally expensive than ReLU or Leaky ReLU
Tanh	• Outputs are bounded between -1 and 1 • Useful for regression tasks	• Saturating outputs can lead to vanishing gradients
Sigmoid	• Outputs are probabilities (between 0 and 1) • Useful for binary classification	• Saturating outputs can lead to vanishing gradients • Less efficient than ReLU or Leaky ReLU

https://www.analyticsvidhya.com/blog/2021/04/activation-functions-and-their-derivatives-a-quick-complete-guide/

SOFTMAX



$$\frac{\partial S(z_i)}{\partial z_j} = \begin{cases} S(z_i) \times (1 - S(z_i)) & \text{if } i = j \\ -S(z_i) \times S(z_j) & \text{if } i \neq j \end{cases}$$

HEURISTICS FOR MAKING BP PERFORM BETTER

- **Normalizing the inputs:**
  - Accelerate BP learning process → normalization of the inputs
  - Two measures:
    - The input variables contained in the training set should be uncorrelated
      - This can be done by PCA or any other feature extraction method (ICA, …)
    - The decorrelated input variables should be scaled so that their variance are approximately equal

HEURISTICS FOR MAKING BP PERFORM BETTER

- **Normalizing the inputs:**
  - **Min-Max Scaling (Normalization):**

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

    - Scales the data between 0 and 1.
  - **Z-Score (Standardization):**

$$z = \frac{x - \mu}{\sigma}$$

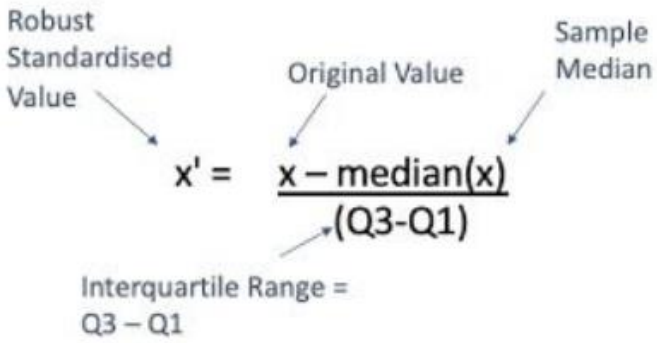
μ = Mean

σ = Standard Deviation

    - Shifts and scales the data to have a mean (μ) of 0 and a standard deviation (σ) of 1.

HEURISTICS FOR MAKING BP PERFORM BETTER

- **Normalizing the inputs:**
  - **Robust Scaling:**



- X is the original data point.
- Median is the 50th percentile of the feature.
- Q3 is the 75th percentile of the feature.
- Q1 is the 25th percentile of the feature.
- **It is less sensitive to outliers.**

HEURISTICS FOR MAKING BP PERFORM BETTER

- **Normalizing the inputs:**
  - **Log Transformation:**
    - Applies a logarithmic function to the data. Useful when the data has a wide range of values.
    - x' = log(x + 1) (+1 to avoid having log(0) → UNDEFINED)
  - **Softmax Scaling:**
    - Commonly used in neural networks for multiclass classification.
    - It exponentiates each element and normalizes them to sum to 1.

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

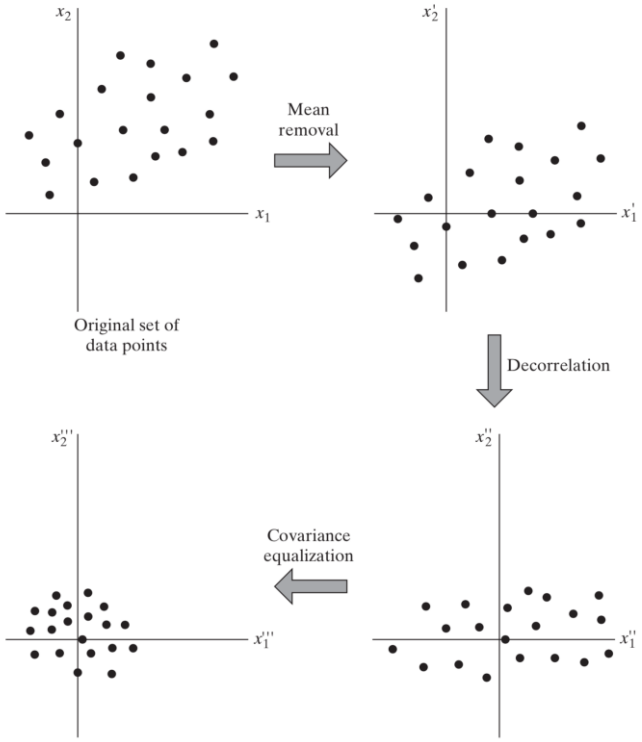
  - **Unit Vector Scaling (Vector Normalization):**
    - Scales each data point to have a magnitude of 1.

$$x' = \frac{x}{\|x\|}$$

$\|x\|$ Is the Euclidean length of the Feature Vector.

HEURISTICS FOR MAKING BP PERFORM BETTER

- **Normalizing the inputs:**
  - Results of three normalization steps:
    - Mean removal, decorrelation and covariance equalization



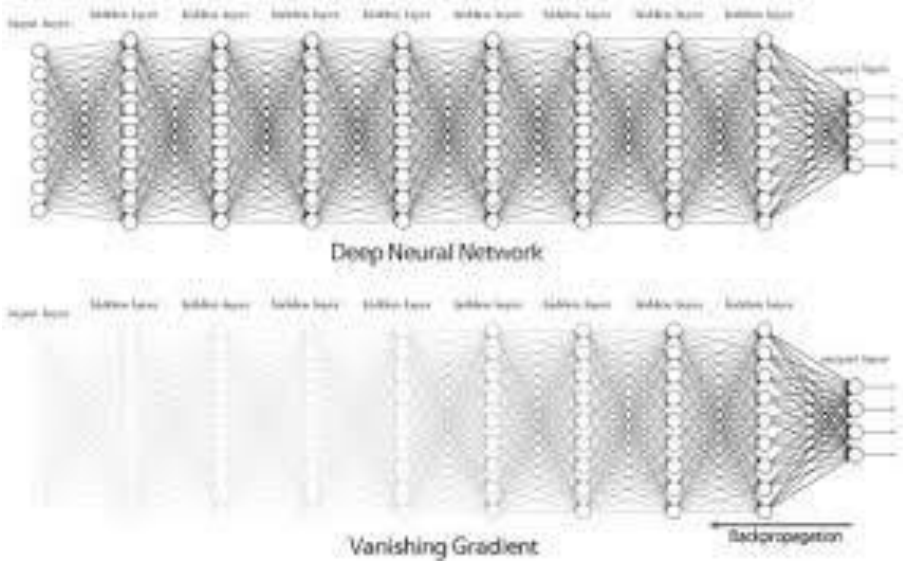
HEURISTICS FOR MAKING BP PERFORM BETTER

- **Weights learning problems:**

1. **Vanishing or exploding gradients** → When weights are initialized improperly → gradients can become too small or too large, causing the optimization algorithm to get stuck or diverge.
   - lead to slow convergence or complete failure of the training process.
2. **Saturated neurons** → When weights are initialized too large or too small, the neurons in the network can become saturated, meaning that the output of the neuron is close to the minimum or maximum value that can be represented by the activation function.
   - can cause the gradient of the activation function to become very small, leading to slow learning or no learning at all.
3. **Slow convergence** → When weights are initialized randomly, it can take a long time for the optimization algorithm to converge to a good solution, especially for deep neural networks with many layers.
4. **Poor performance** → Improper weight initialization can lead to poor performance of the neural network, with lower accuracy or higher error rates on the validation or test set.

VANISHING GRADIENTS

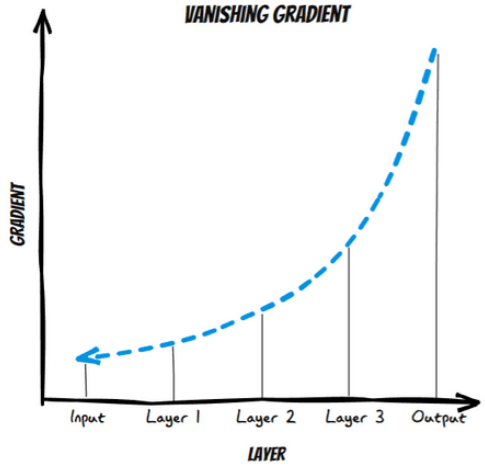
- During the application of backpropagation
  - calculating and updating the weights in the backward direction, the gradients tend to get smaller and smaller as we keep on moving backward in the network.
  - This means the weights of the neurons in the earlier layers learn very slowly or sometimes they won't change at all.
  - Earlier layers in the network are much important because they are responsible for detecting simple patterns.
  - This problem is called a ***vanishing gradient problem.***

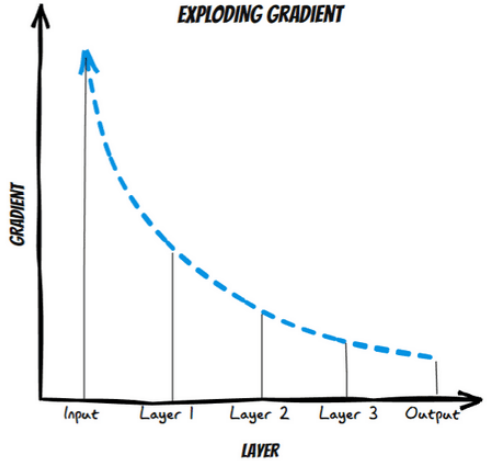


EXPLODING GRADIENTS

- During the application of backpropagation
  - if the gradients become larger and larger, then the weights of the neurons in the earlier stages change much.
  - Earlier layers are much important.
  - Because of this larger weights, the neurons in the earlier layers will give inappropriate results.
  - This problem is called the ***exploding gradients problem***

1. *Vanishing gradient* $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 < 1$
2. *Exploding gradient* $\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\|_2 > 1$





HEURISTICS FOR MAKING BP PERFORM BETTER

- **Weights initialization solutions:**
- **Random initialization** → randomly initialize the weights with small values drawn from a uniform or normal distribution.
  - range of values should be small enough to prevent saturation of the activation function, but large enough to ensure that the weights are not too close to zero.
  - Commonly used ranges include [-0.5, 0.5] or [-1/sqrt(n), 1/sqrt(n)], where n is the number of inputs to a neuron.
- **Xavier initialization**: technique that sets the initial weights such that the variance of the outputs of each layer is equal to the variance of the inputs.
  - can help to avoid the vanishing and exploding gradient problems.
  - weights are initialized by drawing them from a normal distribution with mean 0 and variance of 2/n, where n is the number of inputs to a neuron.

HEURISTICS FOR MAKING BP PERFORM BETTER

- **Weights initialisation solutions:**
- **He initialization** → similar to Xavier initialization but takes into account the activation function being used.
  - sets the initial weights such that the variance of the outputs of each layer is equal to twice the variance of the inputs.
  - weights are initialized by drawing them from a normal distribution with mean 0 and variance of 2/n, where n is the number of inputs to a neuron multiplied by a scaling factor that depends on the activation function being used.
- **Transfer learning** → involves using pre-trained weights from a model that has been trained on a similar task or domain.
  - can help to speed up training and improve performance, especially when the available training data is limited.

So: weights should not be too large or too small.

HEURISTICS FOR MAKING BP PERFORM BETTER

Method	Formula
Zero Initialization	$w_{ij} = 0$
Random Initialization	$w_{ij} = R[-a, a]$
Xavier Initialization	$w_{ij} = N(0, \sqrt{2 / (n_{in} + n_{out})})$
He Initialization	$w_{ij} = N(0, \sqrt{2 / n_{in}})$
Orthogonal Initialization	$w_{ij} = Q_{ij} / \sqrt{n_{in}}$
Kaiming Initialization	$w_{ij} = N(0, \sqrt{2 / (a * n_{in})})$

HEURISTICS FOR MAKING BP PERFORM BETTER

Method	Advantages	Disadvantages
Zero Initialization	➢ Simple and fast	➢ Symmetry breaking, all neurons learn the same features
Random Initialization	➢ Easy to implement	➢ Vanishing/exploding gradients
Xavier Initialization	➢ Stable gradients for sigmoid/tanh	➢ Doesn't consider other activations
He Initialization	➢ Stable gradients for ReLU, prevents dead neurons	➢ More complex than Xavier
Orthogonal Initialization	➢ Reduces information redundancy, improves performance	➢ More computationally expensive
Kaiming Initialization	➢ Efficient for ReLU/Leaky ReLU, prevents dying neurons	➢ More complex than Xavier/He

HEURISTICS FOR MAKING BP PERFORM BETTER

- Learning rates:
  - All neurons in the MLP should learn in ideally the same rate.
  - The last layer usually have larger local gradients than layers at the front end of the network.
  - Thus the learning rate η should be assigned a smaller value in the last layer than in the front layers
  - Neurons with many inputs should have a smaller learning rate parameter than neurons with few inputs so as to maintain a similar learning time to all neurons in the network.
  - **It is suggested that for a given neuron the learning rate should be inversely proportional to its square root of synaptic connections made to that neuron.**

GENERALIZATION

- The aim of the BP to generalize well the problem at hand by computing the weights.
- This is done by presenting as many training examples as possible.

- The network is said to generalize well if the response of input-output mapping of the test data (not used in training) is correct.
- The learning process can be viewed as curve fitting problem.
  - The network can be seen as a non linear input output mapping.
  - Thus generalization can be viewed as simply a good non linear interpolation of the input data.

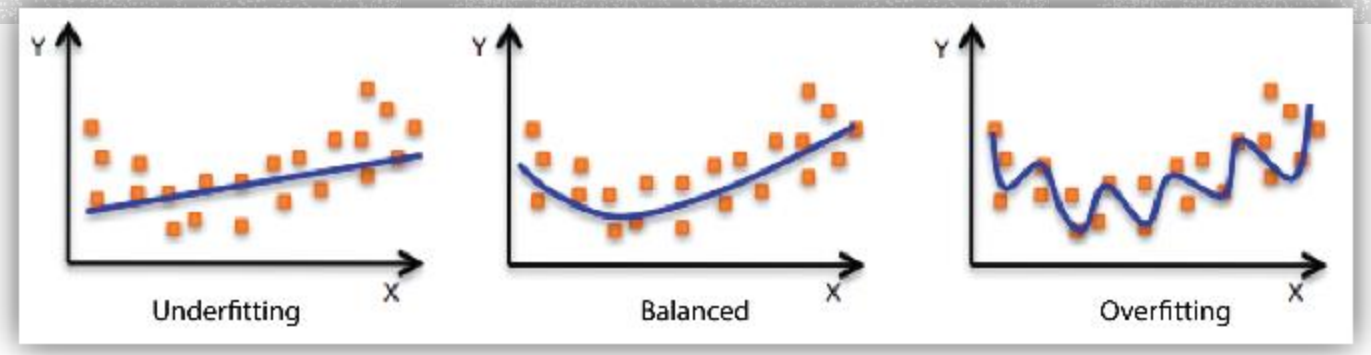
GENERALIZATION

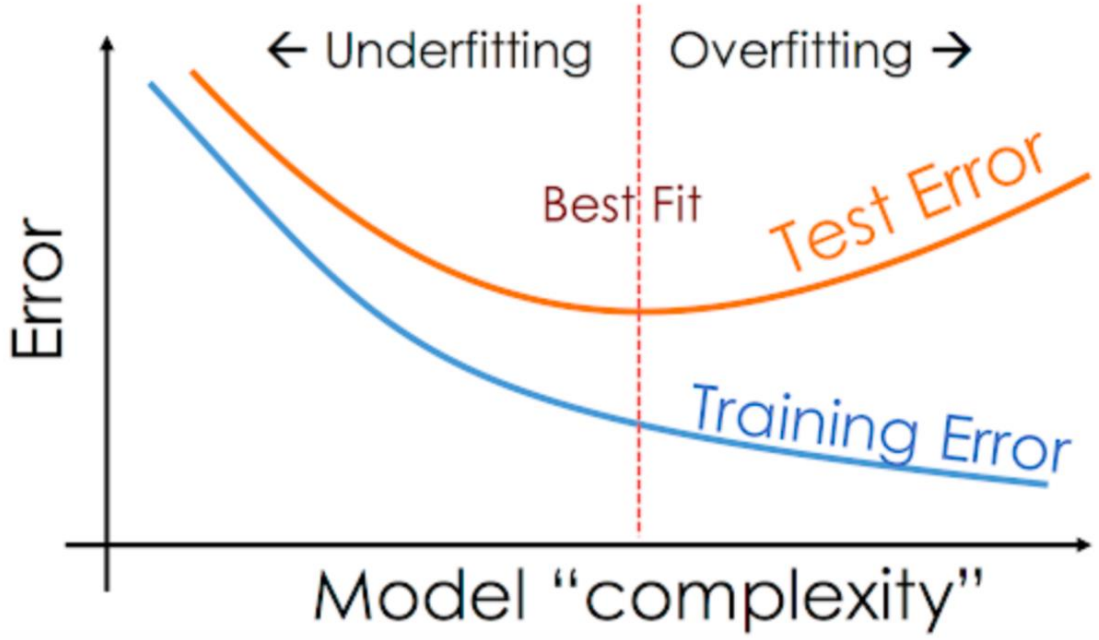
- **Sufficient Training-Sample Size for a Valid Generalization**
  - To have good generalization we need to have the size of the training sample N satisfy the below condition

$$N = O\left(\frac{W}{\varepsilon}\right)$$

  - Where W is the total number of free parameters (weights and biases), ε is the fraction of classification error permitted on test data. $O(.)$ is the order of quantity.
  - For example with an error of 10 percent, the number of training examples needed should be 10 times the number of free parameters in the network.

OVERFITTING PROBLEM





REGULARIZATION IN DEEP LEARNING

- Regularization is a very important and effective technology to reduce generalization errors in machine learning. It is especially useful for deep learning models which tend to have overfitting due to diverse parameters. Researchers have also proposed many effective technologies to prevent overfitting, including:
  - Adding constraints to parameters, such as L_1 and L_2 norms
  - Expanding the training set, such as adding noise and transforming data
  - Dropout

PARAMETER PENALTIES

- Many regularization approaches add a parameter penalty $\Omega(\theta)$ to the objective function J, limiting the learning capability of models. We denote the regularized objective function as $\tilde{J}$:

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

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

L_1 REGULARIZATION

- Add L_1 norm constraint to model parameters:

$$\tilde{J}(w; X, y) = J(w; X, y) + \alpha \|w\|_1,$$

- If the gradient method is used to solve the problem, the parameter gradient is

$$\nabla \tilde{J}(w) = \propto sign(w) + \nabla J(w).$$

L_2 REGULARIZATION

- Add L_2 parameter norm penalty to prevent overfitting.

$$\tilde{J}(w; X, y) = J(w; X, y) + \frac{1}{2}\alpha\|w\|^2,$$

The parameter optimization method can be inferred using the optimization technology (such as gradient correlation method):

$$w = (1 - \varepsilon\alpha)\omega - \varepsilon\nabla J(w),$$

where ε is the learning rate. This multiplies parameters by a reduction factor compared with the common gradient optimization function.

L1 VS L2

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

- *The main intuitive difference between the L1 and L2 regularization is that L1 regularization tries to estimate the median of the data while the L2 regularization tries to estimate the mean of the data to avoid overfitting.*
- *Another difference between them is that L1 regularization helps in feature selection by eliminating the features that are not important. This is helpful when the number of feature points are large in number.*

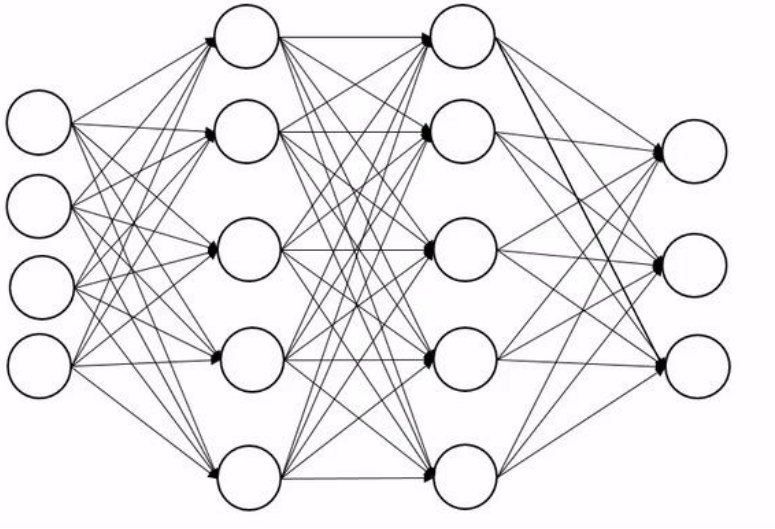
DATASET EXPANSION

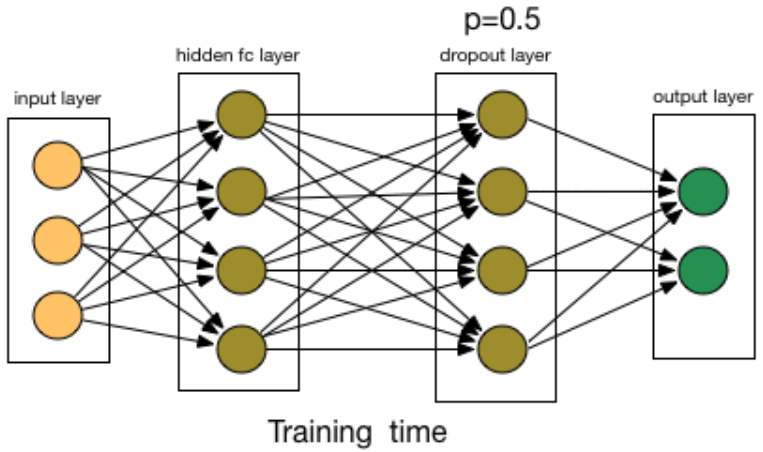
- The most effective way to prevent overfitting is to add a training set. A larger training set has a smaller overfitting probability. Dataset expansion is a time-saving method, but it varies in different fields.
  - ✓ A common method in the object recognition field is to rotate or scale images. (The prerequisite to image transformation is that the type of the image cannot be changed through transformation. For example, for handwriting digit recognition, category 6 and 9 can be easily changed after rotation).
  - ✓ Random noise is added to the input data in speech recognition.
  - ✓ The common idea of NLP is to replace words with their synonyms.
  - ✓ Noise injection can add noise to the input or to the hidden layer or output layer. For example, for the softmax classification problem, noise can be added by using the label smoothing technology. If noise is added to the 0-1 category, the corresponding probability is changed to $\frac{\varepsilon}{k}$ and $1 - \frac{k-1}{k}\varepsilon$.

DROPOUT

- Dropout is a regularization technique used in machine learning and deep learning to prevent overfitting in neural networks. It involves randomly 'dropping out,' which means temporarily removing, a number of neurons in a layer during a training step. Only the remaining neurons contribute to the forward pass and backward pass.
- Dropout is usually specified as a percentage rate (between 0.0 and 1.0), which is the fraction of the neurons to drop. For example, a dropout rate of 0.5 means half of the neurons in that layer will be dropped out.
- The idea behind dropout is that it forces the network to learn more robust and generalized representations. By removing neurons at random during training, it prevents the model from becoming too reliant on any single neuron or combination of neurons.
- Then during testing or evaluation of the model, no neurons are dropped out; but the layer's output values are scaled down by the dropout rate, to balance the fact that more neurons are active than during training. It's a kind of "ensemble" technique within a single network.

DROPOUT




p=0.5
hidden fc layer
dropout layer
input layer
output layer
Training time