Deep learning is the branch of machine learning which is based on artificial neural network architecture which makes it capable of learning complex patterns and relationships within data. An artificial neural network or ANN uses layers of interconnected nodes called neurons that work together to process and learn from the input data.

In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after the other. Each neuron receives input from the previous layer neurons or the input layer. The output of one neuron becomes the input to other neurons in the next layer of the network, and this process continues until the final layer produces the output of the network.

An artificial neural network is inspired by the networks and functionalities of human biological neurons. ANN uses layers of interconnected nodes called artificial neurons that work together to process and learn the input data.

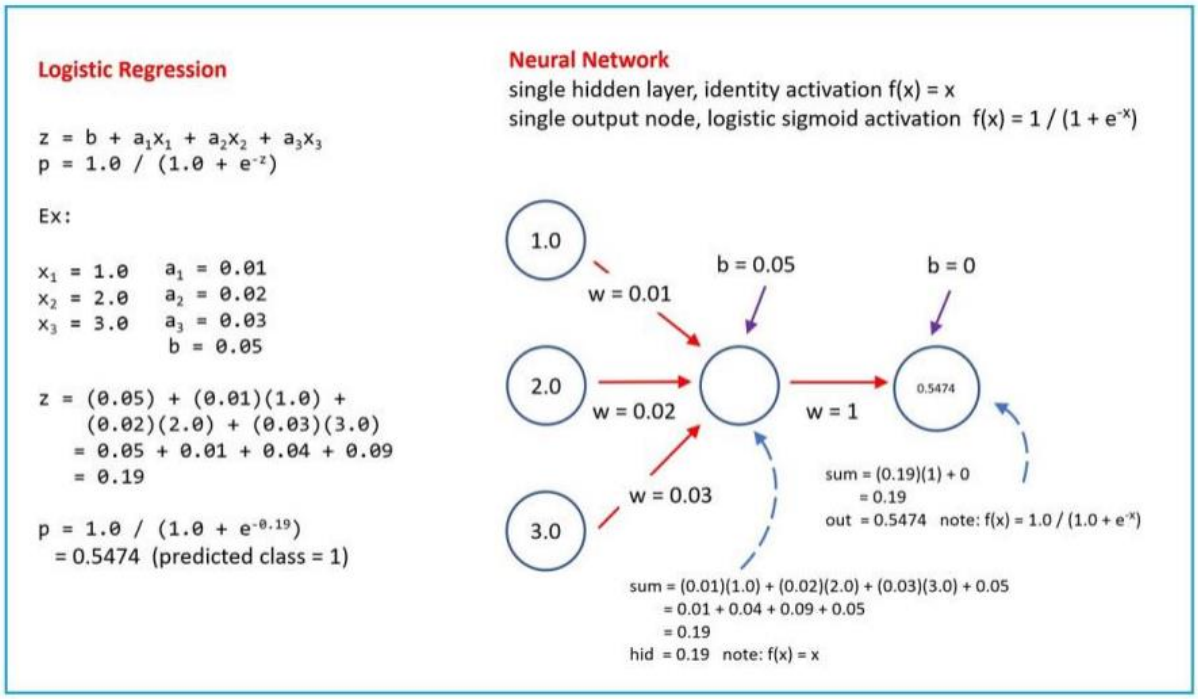
Machine Learning	Deep Learning
• Enables machines to take decisions on their own, based on past data • It needs only a small amount of data for training • Works well on the low-end system, so you don't need large machines • Most features need to be identified in advance and manually coded • The problem is divided into two parts and solved individually and then combined	• Enables machines to take decisions with the help of artificial neural networks • It needs a large amount of training data • Needs high-end machines because it requires a lot of computing power • The machine learns the features from the data it is provided • The problem is solved in an end-to-end manner

Logistic Regression

Definition:
Logistic regression is a statistical method used for binary classification tasks, where the outcome variable can take on two possible values (commonly 0 and 1). It models the relationship between one or more independent variables (predictors) and a binary dependent variable by estimating probabilities using the logistic function.

Sigmoid Function

Definition:
that maps any real-valued number into a value between 0 and 1. It is used in logistic regression to convert linear combinations of the input variables into probabilities.



Loss Function for Logistic Regression

Definition:
In deep learning, a **loss function** (also known as a cost function) is a mathematical function that measures the difference between the predicted output of a model and the actual target output. The purpose of a loss function is to quantify how well or poorly a model is performing.

By minimizing the loss function, a model can improve its predictions over time during the training process.

Formula

For a single instance, the binary cross-entropy loss is defined as:

$$\text{Loss} = -\left[y \log(P) + (1 - y) \log(1 - P)\right]$$

Example

Let's consider a scenario with a single data point where:

- The actual outcome $y = 1$ (the email is spam).
- The predicted probability $P = 0.8$ (the model predicts an 80% chance of being spam).

Step 1: Calculate the Loss

Using the formula for loss:

$$\text{Loss} = -\left[1 \cdot \log(0.8) + (1 - 1) \cdot \log(1 - 0.8)\right]$$

Since $y = 1$, the second term becomes zero:

$$\text{Loss} = -\log(0.8)$$

Calculating $\log(0.8)$:

$$\log(0.8) \approx -0.2231 \quad \text{(using natural logarithm)}$$

Thus:

$$\text{Loss} = -(-0.2231) = 0.2231$$

Example: Mean Squared Error (MSE) for Regression

For regression tasks, a common loss function is the **Mean Squared Error (MSE)**, which is defined as:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Here, the squared difference between the true and predicted values is averaged over all samples.

Example Calculation:

Let's say we have 3 samples with true values $y = [3, -0.5, 2]$ and predicted values $\hat{y} = [2.5, 0.0, 2]$.

1. Calculate the squared errors for each sample:
   - $(3 - 2.5)^2 = 0.25$
   - $(-0.5 - 0.0)^2 = 0.25$
   - $(2 - 2)^2 = 0$
2. Sum and average these squared errors:

$$MSE = \frac{1}{3}(0.25 + 0.25 + 0) = \frac{0.5}{3} \approx 0.167$$

Step 2: Another Example for $y = 0$

Now, let's consider another instance where:

- The actual outcome $y = 0$ (the email is not spam).
- The predicted probability $P = 0.3$ (the model predicts a 30% chance of being spam).

Step 3: Calculate the Loss

Using the same formula:

$$\text{Loss} = -\left[0 \cdot \log(0.3) + (1 - 0) \cdot \log(1 - 0.3)\right]$$

Since $y = 0$, the first term becomes zero:

$$\text{Loss} = -\log(0.7)$$

Calculating $\log(0.7)$:

$$\log(0.7) \approx -0.3567$$

Thus:

$$\text{Loss} = -(-0.3567) = 0.3567$$

Deep learning has made significant advancements in various fields, but there are still some challenges that need to be addressed. Here are some of the main challenges in deep learning:

1. Data availability: It requires large amounts of data to learn from. For using deep learning it's a big concern to gather as much data for training.
2. Computational Resources: For training the deep learning model, it is computationally expensive because it requires specialized hardware like GPUs and TPUs.
3. Time-consuming: While working on sequential data depending on the computational resource it can take very large even in days or months.
4. Interpretability: Deep learning models are complex, it works like a black box. it is very difficult to interpret the result.
5. Overfitting: when the model is trained again and again, it becomes too specialized for the training data, leading to overfitting and poor performance on new data.
6. Activation: In biological neurons, activation is the firing rate of the neuron which happens when the signals are strong enough to reach the threshold. and in artificial neural networks, activations are done by mathematical functions known as activations functions which map the input to the output.

- **Input Layer:** This is the layer that receives the input data and passes it on to the next layer. The input layer is typically not counted as one of the hidden layers of the network.
- **Hidden Layers:** The input layer is the one that receives input data and transfers it to the next layer. Usually, the input layer is not included in the list of the hidden layers of the neural network.
- **Output Layer:** This is the output-producing layer of the network. A binary classification problem might only have one output neuron, but a multi-class classification problem might have numerous output neurons, one for each class. The number of neurons in the output layer depends on the type of problem being solved.

The cost function is the mathematical function that is used to measure the quality of prediction during training in deep neural networks. It measures the differences between the generated output of the forward pass of the neural network to the actual outputs, which are known as losses or errors. During the training process, the weights of the network are adjusted to minimize the losses. which is achieved by computing the gradient of the cost function with respect to weights and biases using backpropagation algorithms.

Deep learning uses activation functions, which are mathematical operations that are performed on each neuron's output in a neural network to provide nonlinearity to the network. The goal of activation functions is to inject non-linearity into the network so that it can learn the more complex relationships between the input and output variables.

In deep learning, several different-different types of activation functions are used. Each of them has its own strength and weakness. Some of the most common activation functions are as follows.

- **Sigmoid function:** It maps any value between 0 and 1. It is mainly used in binary classification problems. where it maps the output of the preceding hidden layer into the probability value.
- **Softmax function:** It is the extension of the sigmoid function used for multi-class classification problems in the output layer of the neural network, where it maps the output of the previous layer into a probability distribution across the classes, giving each class a probability value between 0 and 1 with the sum of the probabilities over all classes is equal to 1. The class which has the highest probability value is considered as the predicted class.
- **ReLU (Rectified Linear Unit) function:** It is a non-linear function that returns the input value for positive inputs and 0 for negative inputs. Deep neural networks frequently employ this function since it is both straightforward and effective.
- **Leaky ReLU function:** It is similar to the ReLU function, but it adds a small slope for negative input values to prevent dead neurons.
- **Tanh (hyperbolic tangent) function:** It is a non-linear activations function that maps the input's value between -1 to 1. It is similar to the sigmoid function but it provides both positive and negative results. It is mainly used for regression tasks, where the output will be continuous values.
- During the training, in forward pass the input data passes through the network and generates output. then the cost function compares this generated output to the actual output. then the backpropagation computes the gradient of the cost function with respect to the output of the neural network. This gradient is then propagated backward through the network to compute the gradient of the loss function with respect to the weights and biases of each layer. Here chain rules of differentiations are applied with respect to the parameters of each layer to find the gradient.
- Once the gradient is computed, The optimization algorithms are used to update the parameters of the network. Some of the most common optimization algorithms are stochastic gradient descent (SGD), mini-batch, etc.

  How to select number of hidden layers

- The number of hidden layers can be determined by the complexity of the problem being solved. Simple problems can be solved with just one hidden layer whereas more complicated problems may require two or more hidden levels. However adding more layers also increases the risk of overfitting, so the number of layers should be chosen based on the trade-off between model complexity and generalization performance.
-
- 9. What is overfitting and how to avoid it?
- Overfitting occurs when the model learns to fit the training data too close to the point that it starts catching up on noise and unimportant patterns. Because of this, the model performs well on training data but badly on fresh, untested data, resulting in poor generalization performance.

- To avoid overfitting in deep learning we can use the following techniques:
- Simplify the model: Overfitting may be less likely in a simpler model with fewer layers and parameters. In practical applications, it is frequently beneficial, to begin with a simple model and progressively increase its complexity until the desired performance is attained.
- Regularization: Regularization is a technique to prevent the overfitting of a model by adding a penalty term, it imposes the constraint on the weight of the model. It occurs when a model learns the training data too well, including its noise and outliers, and fails to generalize to unseen data.
- Some of the most common regularization techniques are as follows:
- L1 and L2 regularization: L1 regularization sparse the model by equating many model weights equal to 0 while L2 regularization constrains the weight of the neural network connection.
- L1 Regularization (Lasso):
- Adds a penalty equal to the absolute value of the weights.
- Encourages sparsity in the model (some weights may become exactly zero).

$$\text{Loss} = \text{Original Loss} + \lambda \sum |w_i|$$

Example of L1 Regularization (Lasso)

Scenario:

Suppose we have a simple model with the following parameters:

- **Original Loss** (e.g., cross-entropy loss): $\text{Original Loss} = 0.4$
- **Weights:** $w_1 = 0.5, w_2 = -0.3$
- **Regularization Strength:** $\lambda = 0.1$

Step 1: Calculate L1 Penalty

The L1 penalty is calculated as follows:

$$\text{L1 Penalty} = \lambda \sum |w_i| = 0.1 \times (|0.5| + |-0.3|) = 0.1 \times (0.5 + 0.3) = 0.1 \times 0.8 = 0.08$$

Step 2: Calculate Total Loss with L1 Regularization

$$\text{Total Loss} = \text{Original Loss} + \text{L1 Penalty} = 0.4 + 0.08 = 0.48$$

L2 Regularization (Ridge):

- Adds a penalty equal to the square of the weights.
- Encourages smaller weights but doesn't necessarily force them to be zero.

$$\text{Loss} = \text{Original Loss} + \lambda \sum w_i^2$$

Example of L2 Regularization (Ridge)

Scenario:

Let's use the same original loss and weights, but now apply L2 regularization:

- **Original Loss:** $\text{Original Loss} = 0.4$
- **Weights:** $w_1 = 0.5, w_2 = -0.3$
- **Regularization Strength:** $\lambda = 0.1$

Step 1: Calculate L2 Penalty

The L2 penalty is calculated as follows:

$$\text{L2 Penalty} = \lambda \sum w_i^2 = 0.1 \times (0.5^2 + (-0.3)^2) = 0.1 \times (0.25 + 0.09) = 0.1 \times 0.34 = 0.034$$

Step 2: Calculate Total Loss with L2 Regularization

$$\text{Total Loss} = \text{Original Loss} + \text{L2 Penalty} = 0.4 + 0.034 = 0.434$$

Dropout:

Dropout is one of the most popular regularization techniques used in deep learning to prevent overfitting. The basic idea behind this is to randomly drop out or set to zero some of the neurons of the previously hidden layer so that its contribution is temporarily removed during the training for both forward and backward passes.

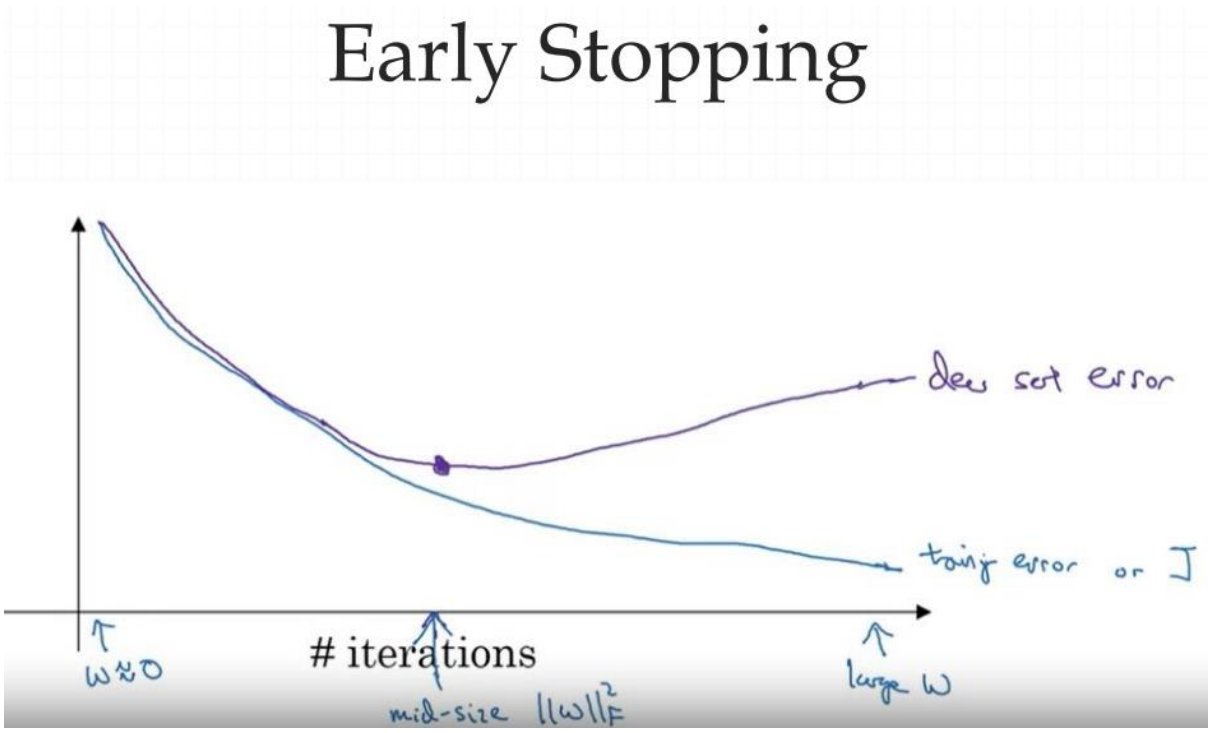
In each iteration, neurons for the dropout are selected randomly and their values are set to zero so that it doesn't affect the downstream neurons of upcoming next-layer neurons during the forward pass, And during the backpropagation, there is no weight update for these randomly selected neurons in current iterations. In this way, a subset of randomly selected neurons is completely ignored during that particular iteration.

Prevents co-adaptation of neurons, making the network more robust.

Typically, a fraction of neurons is dropped during each training iteration.

Early Stopping:

- Monitoring the model's performance on a validation set and stopping training when performance starts to degrade.
- Helps to prevent the model from overfitting to the training data.



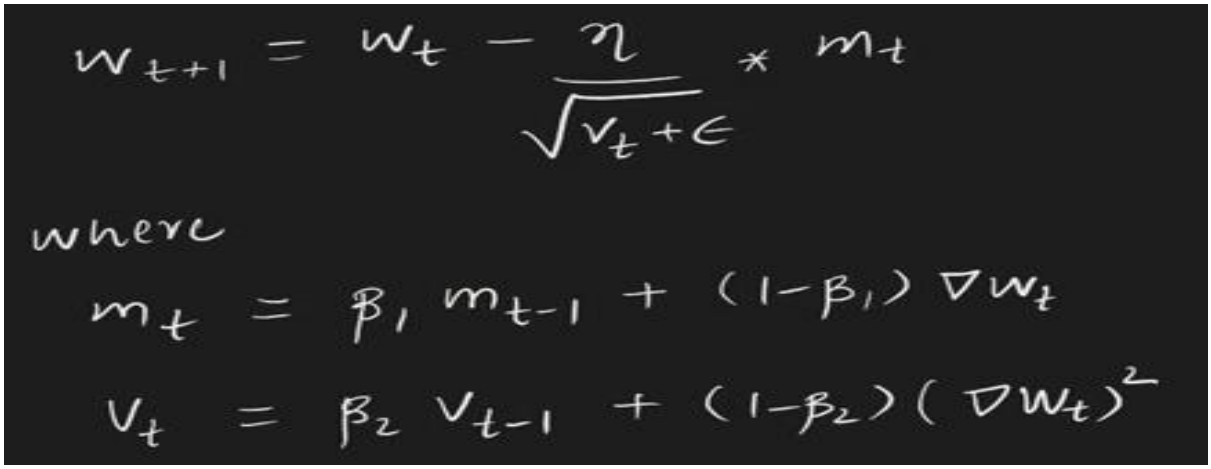
Data Augmentation:

While not a traditional regularization technique, augmenting the training dataset with modified versions of the input data (like rotated or flipped images) can help improve generalization.

Max-Norm Regularization: It constrains the magnitude of the weights in a neural network by setting a maximum limit (or norm) on the weights of the neurons, such that their values cannot exceed this limit.

Adam Optimization

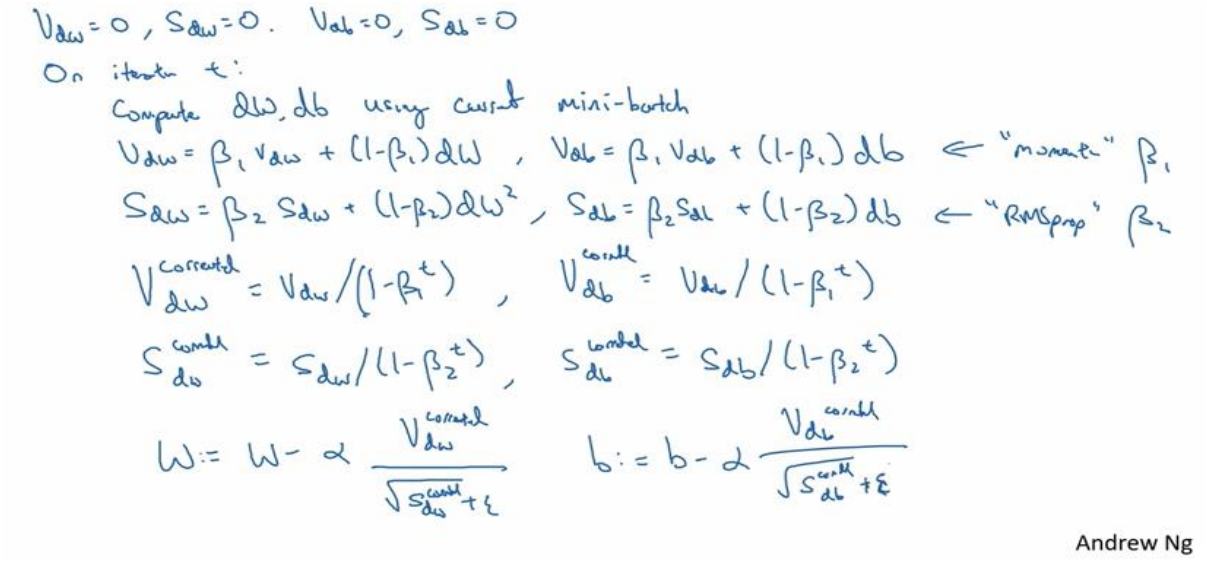
- Adam merges momentum and learning decay
- Uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network
- In Statistics, Moments are popularly used to describe the characteristic of a distribution
- These are very useful in statistics because they tell you much about your data

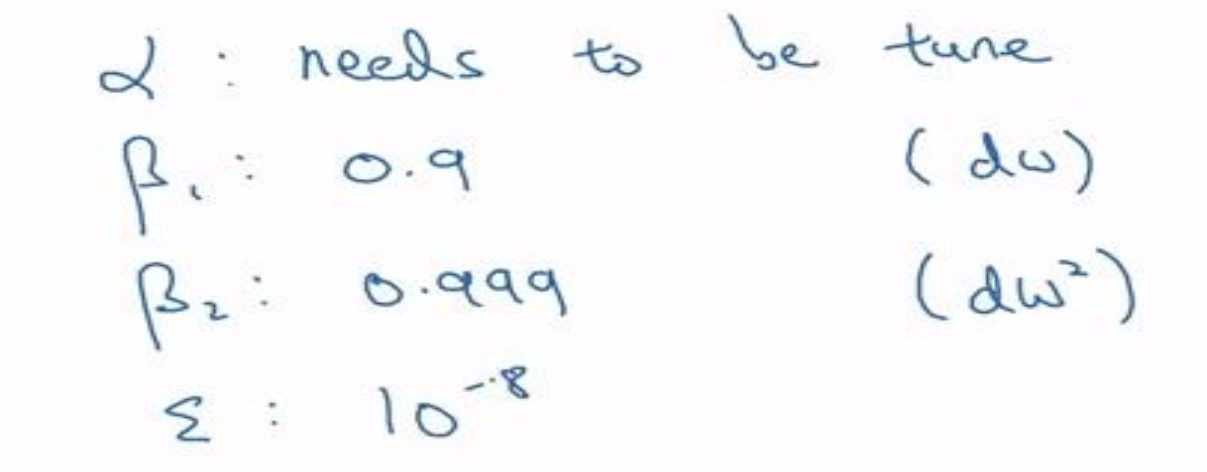


Recall Bias Collection

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

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





Definition and Working

The AdaGrad (Adaptive Gradient) optimizer is a gradient-based optimization algorithm designed to improve training efficiency and convergence speed in deep learning models. Unlike standard gradient descent, which uses a fixed learning rate for all parameters, AdaGrad adjusts the learning rate dynamically by scaling it inversely proportional to the square root of the sum of all past squared gradients for each parameter. This means that parameters with frequently large gradients receive smaller updates, while those with infrequent updates receive larger ones.

Example Calculation

Let's assume the following values:

- $\eta = 0.1$
- $\epsilon = 1 \times 10^{-8}$
- Initial parameters: $w_{t-1}^1 = 0.5$, $w_{t-1}^2 = -0.3$, $b_{t-1} = 0.1$
- Gradients: $\frac{\partial L}{\partial w_{t-1}^1} = 0.2$, $\frac{\partial L}{\partial w_{t-1}^2} = -0.4$, $\frac{\partial L}{\partial b_{t-1}} = 0.1$
- Accumulated squared gradients up to t: $\alpha_t^1 = 0.04$, $\alpha_t^2 = 0.16$, $\alpha_t = 0.01$

Now, let's substitute these values into the formula.

1. Calculate the individual learning rates for each parameter:
   - For w_t^1: $\frac{\eta}{\sqrt{\alpha_t^1 + \epsilon}} = \frac{0.1}{\sqrt{0.04 + 1 \times 10^{-8}}} \approx 0.5$
   - For w_t^2: $\frac{\eta}{\sqrt{\alpha_t^2 + \epsilon}} = \frac{0.1}{\sqrt{0.16 + 1 \times 10^{-8}}} \approx 0.25$
   - For b_t: $\frac{\eta}{\sqrt{\alpha_t + \epsilon}} = \frac{0.1}{\sqrt{0.01 + 1 \times 10^{-8}}} \approx 1.0$
2. Update each parameter:
   - $w_t^1 = w_{t-1}^1 - 0.5 \times 0.2 = 0.5 - 0.1 = 0.4$
   - $w_t^2 = w_{t-1}^2 - 0.25 \times (-0.4) = -0.3 + 0.1 = -0.2$
   - $b_t = b_{t-1} - 1.0 \times 0.1 = 0.1 - 0.1 = 0.0$

Results

After one update step with AdaGrad:

- $w_t^1 = 0.4$
- $w_t^2 = -0.2$
- $b_t = 0.0$

A **batch** in deep learning is a subset of the training data that is used to modify the weights of a model during training. In batch training, the entire training set is divided into smaller groups, and the model is updated after analyzing each batch. An epoch can be made up of one or more batches.

- The batch size will be more than one and always less than the number of samples.
- Batch size is a hyperparameter, it is set by the user. where the number of iterations per epoch is calculated by dividing the total number of training samples by the individual batch size.

23. What is gradient descent?

It is the method used to minimize the cost or loss function by iteratively adjusting the model parameters i.e. weight and biases of the neural layer. The objective is to reduce this disparity, which is represented by the cost function as the difference between the model's anticipated

output and the actual output. The magnitude of the update is determined by the learning rate, which controls the step size of the update.

25. Define Batch, Stochastic, and Mini gradient descent.

Definition:

- SGD is a variant of gradient descent where the parameter updates are performed for each individual training example, rather than using the entire dataset at once (as in batch gradient descent).
- This makes SGD faster, especially for large datasets, but it can result in noisy updates that may oscillate around the minimum rather than converging smoothly.

Update Rule:

- For each training sample, the parameters θ are updated as:

$$\theta = \theta - \eta \cdot \nabla L(\theta)$$

where:

- η is the learning rate,
- $\nabla L(\theta)$ is the gradient of the loss function with respect to the parameters.

Advantages:

- Fast updates due to only one sample being used at each step.
- Introduces stochasticity, which can help escape local minima.

- ↓

SGD with Momentum

Definition:

- SGD with Momentum aims to improve the convergence speed and stability of SGD by adding a "momentum" term to the parameter updates.
- Momentum helps to smooth out the updates by incorporating a fraction of the previous update into the current one. This allows the optimizer to maintain direction across updates and dampens oscillations.

Update Rule:

- The parameter updates are influenced by both the current gradient and the previous update:

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla L(\theta)$$

$$\theta = \theta - v_t$$

where:

- v_t is the velocity term at step t,
- γ (momentum coefficient, typically between 0.5 and 0.9) controls how much of the previous update contributes to the current update.

Mini-batch Gradient Descent is a variant of gradient descent that splits the training dataset into smaller subsets, called mini-batches, and performs updates on the model parameters for each mini-batch. This approach combines aspects of both Batch Gradient Descent and Stochastic Gradient Descent (SGD), aiming to achieve a balance between their advantages and drawbacks.

Key Concepts

- Mini-batch: A subset of the training data, typically containing a fixed number of examples (e.g., 32, 64, or 128).
- Epoch: One full pass through the entire training dataset.
- Iteration: One update of the model parameters, which corresponds to processing one mini-batch. In one epoch, the number of iterations equals the number of mini-batches.

How it Works

1. Divide the dataset: The entire dataset is divided into several mini-batches.
2. Compute gradients: For each mini-batch, the gradient of the loss function is computed.
3. Update parameters: The model parameters are updated based on the average gradient over the mini-batch.
4. Repeat: This process is repeated for all mini-batches in an epoch, and the entire procedure is repeated across multiple epochs until the model converges.
5. **RMSProp** (Root Mean Square Propagation) is an adaptive learning rate optimization algorithm designed to address some of the issues faced by other optimization methods like SGD. It adjusts the learning rate for each parameter dynamically, allowing the model to perform more stable updates, especially in the presence of noisy or sparse gradients.

6. Key Concepts

7. RMSProp was developed to deal with the issue of **oscillations** in the optimization path, particularly in deep networks. It achieves this by maintaining a **moving average of the squared gradients** for each parameter, which helps to normalize the updates and keep them within a reasonable range.

Overfitting and Underfitting

- **Overfitting**:
    - The model learns the training data too well, including noise and outliers, which results in poor generalization to new data.
    - Common in models with high capacity (large or complex networks) and limited training data.
    - **Solutions**: Use regularization techniques like dropout, L2 regularization, or early stopping. Increasing the training data and using data augmentation can also help.

- **Underfitting**:
    - The model is too simple to capture the underlying patterns in the data, resulting in poor performance on both training and validation data.
    - Common when the model is not complex enough or lacks enough capacity to learn the data's patterns.
    - **Solutions**: Use a more complex model, increase the number of parameters, or use a more sophisticated network architecture.

36. What are Convolutional Neural Networks (CNNs)?

Convolutional Neural Networks (CNNs) are the type of neural network commonly used for Computer Vision tasks like image processing, image classification, object detection, and segmentation tasks. It applies filters to the input image to detect patterns, edges, and textures and then uses these features to classify the image.

It is the type of feedforward neural network (FNN) used to extract features from grid-like datasets by applying different types of filters also known as the kernel. For example visual datasets like images or videos where data patterns play an extensive role. It uses the process known as convolution to extract the features from images.

It is composed of multiple layers including the convolution layer, the pooling layer, and the fully connected layer. In the convolutional layers, useful features are extracted from the input data by applying a kernel, The kernel value is adjusted during the training process, and it helps to identify patterns and structures within the input data.

The pooling layers then reduce the spatial dimensionality of the feature maps, making them more manageable for the subsequent layers. Finally, the fully connected layers use the extracted features to make a prediction or classification.

41. Define the same and valid padding.

Padding is a technique used in convolutional neural networks to preserve the spatial dimensions of the input data and prevent the loss of information at the edges of the image. it is done by adding additional layers of zeros around the edges of the input matrix.

There are two main types of padding: same padding and valid padding.

- **Same Padding:** The term "same padding" describes the process of adding padding to an image or feature map such that the output has the same spatial dimensions as the input. The same padding adds additional rows and columns of pixels around the edges of the input data so that the size of the output feature map will be the same as the size of the input data. This is achieved by adding rows and columns of pixels with a value of zero around the edges of the input data before the convolution operation.
- **Valid Padding:** Convolutional neural networks (CNNs) employ the valid padding approach to analyze the input data without adding any extra rows or columns of pixels around the input data's edges. This means that the size of the output feature map is smaller than the size of the input data. Valid padding is used when it is desired to reduce

the size of the output feature map in order to reduce the number of parameters in the model and improve its computational efficiency.

50. What is an Encoder-Decoder network in Deep Learning?

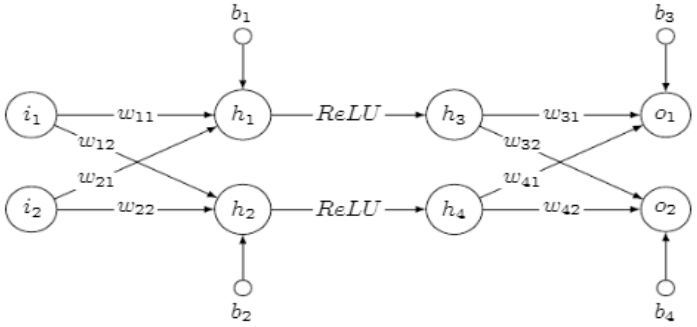
An encoder-decoder network is a kind of neural network that can learn to map an input sequence to a different length and structure output sequence. It is made up of two primary parts: an encoder and a decoder.

- Encoder: The encoder takes a variable-length input sequence (such as a sentence, an image, or a video) and processes it step by step steps to build a fixed-length context or encoded vector or representation that captures the important information from the input sequence. The encoded vector condenses the information from the entire input sequence.
- Decoder: Decoder is another neural network that takes the encoded vector as input and generates an output sequence (such as another sentence, an image, or a video) that is related to the input sequence. The decoder generates an output and modifies its internal hidden state based on the encoded vector and previously generated outputs at each step.

51. What is an autoencoder?
Autoencoders are a type of neural network architecture used for unsupervised learning tasks like dimensionality reduction, feature learning, etc. Autoencoders work on the principle of learning a low-dimensional representation of high-dimensional input data by compressing it into a latent representation and then reconstructing the input data from the compressed representation. It consists of two main parts an encoder and a decoder. The encoder maps an input to a lower-dimensional latent representation, while the decoder maps the latent representation back to the original input space. In most cases, neural networks are used to create the encoder and decoder, and they are trained in parallel to reduce the difference between the original input data and the reconstructed data.

Part III: Backpropagation (9 points)

1. (9 points) Given the following neural network with fully connection layer and ReLU activations, including two input units (i_1, i_2), four hidden units (h_1, h_2) and (h_3, h_4). The output units are indicated as (o_1, o_2) and their targets are indicated as (t_1, t_2). The weights and bias of fully connected layer are called w and b with specific sub-descriptors.



The values of variables are given in the following table:

Variable	i_1	i_2	w_{11}	w_{12}	w_{21}	w_{22}	w_{31}	w_{32}	w_{41}	w_{42}	b_1	b_2	b_3	b_4	t_1	t_2
Value	2.0	-1.0	1.0	-0.5	0.5	-1.0	0.5	-1.0	-0.5	1.0	0.5	-0.5	-1.0	0.5	1.0	0.5

(a) (3 points) Compute the output (o_1, o_2) with the input (i_1, i_2) and network paramters as specified above. Write down all calculations, including intermediate layer results.

Solution:

Forward pass:

$$h_1 = i_1 \times w_{11} + i_2 \times w_{21} + b_1 = 2.0 \times 1.0 - 1.0 \times 0.5 + 0.5 = 2.0$$

$$h_2 = i_1 \times w_{12} + i_2 \times w_{22} + b_2 = 2.0 \times -0.5 + -1.0 \times -1.0 - 0.5 = -0.5$$

$$h_3 = max(0, h_1) = h_1 = 2$$

$$h_4 = max(0, h_2) = 0$$

$$o_1 = h_3 \times w_{31} + h_4 \times w_{41} + b_3 = 2 \times 0.5 + 0 \times -0.5 - 1.0 = 0$$

$$o_2 = h_3 \times w_{32} + h_4 \times w_{42} + b_4 = 2 \times -1.0 + 0 \times 1.0 + 0.5 = -1.5$$

(b) (1 point) Compute the mean squared error of the output (o_1, o_2) calculated above and the target (t_1, t_2).

Solution:

$$MSE = \frac{1}{2} \times (t_1 - o_1)^2 + \frac{1}{2} \times (t_2 - o_2)^2 = 0.5 \times 1.0 + 0.5 \times 4.0 = 2.5$$

(c) (5 points) Update the weight w_{21} using gradient descent with learning rate 0.1 as well as the loss computed previously. (Please write down all your computations.)

Solution:

Backward pass (Applying chain rule):

$$\frac{\partial MSE}{\partial w_{21}} = \frac{\partial \frac{1}{2}(t_1 - o_1)^2}{\partial o_1} \times \frac{\partial o_1}{\partial h_3} \times \frac{\partial h_3}{\partial h_1} \times \frac{\partial h_1}{\partial w_{21}} + \frac{\partial \frac{1}{2}(t_2 - o_2)^2}{\partial o_2} \times \frac{\partial o_2}{\partial h_3} \times \frac{\partial h_3}{\partial h_1} \times \frac{\partial h_1}{\partial w_{21}}$$

$$= (o_1 - t_1) \times w_{31} \times 1.0 \times i_2 + (o_2 - t_2) \times w_{32} \times 1.0 \times i_2$$

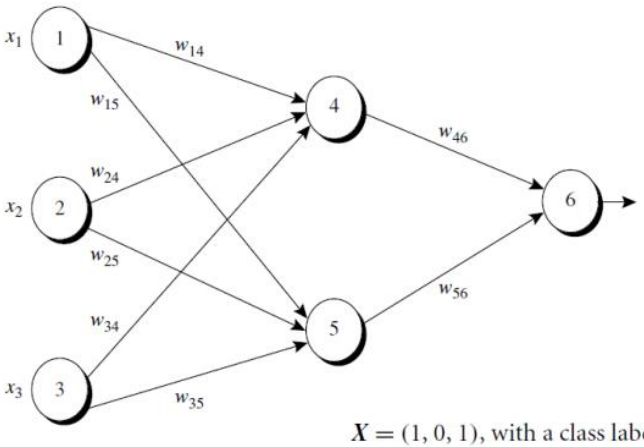
$$= (0 - 1.0) \times 0.5 \times -1.0 + (-1.5 - 0.5) \times -1.0 \times -1.0$$

$$= 0.5 + -2.0 = -1.5$$

Update using gradient descent:

$$w_{21}^{+} = w_{21} - lr * \frac{\partial MSE}{\partial w_{21}} = 0.5 - 0.1 * -1.5 = 0.65$$

Worked example



$\boldsymbol{X} = (1, 0, 1)$, with a class label of 1.

Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Net Input and Output Calculations

Unit, j	Net Input, I_j	Output, O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

Calculation of the Error at Each Node

Unit, j	Err_j
6	$(0.474)(1-0.474)(1-0.474)=0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2)=-0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087$

Calculations for Weight and Bias Updating

Weight or Bias	New Value
w_{46}	$-0.3+(0.9)(0.1311)(0.332)=-0.261$
w_{56}	$-0.2+(0.9)(0.1311)(0.525)=-0.138$
w_{14}	$0.2+(0.9)(-0.0087)(1)=0.192$
w_{15}	$-0.3+(0.9)(-0.0065)(1)=-0.306$
w_{24}	$0.4+(0.9)(-0.0087)(0)=0.4$
w_{25}	$0.1+(0.9)(-0.0065)(0)=0.1$
w_{34}	$-0.5+(0.9)(-0.0087)(1)=-0.508$
w_{35}	$0.2+(0.9)(-0.0065)(1)=0.194$
θ_6	$0.1+(0.9)(0.1311)=0.218$
θ_5	$0.2+(0.9)(-0.0065)=0.194$
θ_4	$-0.4+(0.9)(-0.0087)=-0.408$

Advantage

- Advantages of neural networks, however, include their high tolerance of noisy data as well as their ability to classify patterns on which they have not been trained. They can be used when you may have little knowledge of the relationships between attributes and classes.
- They are well suited for continuous-valued inputs *and outputs, unlike most* decision tree algorithms. They have been successful on a wide array of real-world data, including handwritten character recognition, pathology and laboratory medicine, and training a computer to pronounce English text

---

- Neural network algorithms are inherently parallel; parallelization techniques can be used to speed up the computation process

- It can perform tasks which a linear classifier cannot
- Multilayer feed-forward networks, given enough hidden units and enough training samples, can closely approximate any function

7. (0 points) Optional: Given a Convolution Layer with 8 filters, a filter size of 6, a stride of 2, and a padding of 1. For an input feature map of $32 \times 32 \times 32$, what is the output dimensionality after applying the Convolution Layer to the input?

Solution:

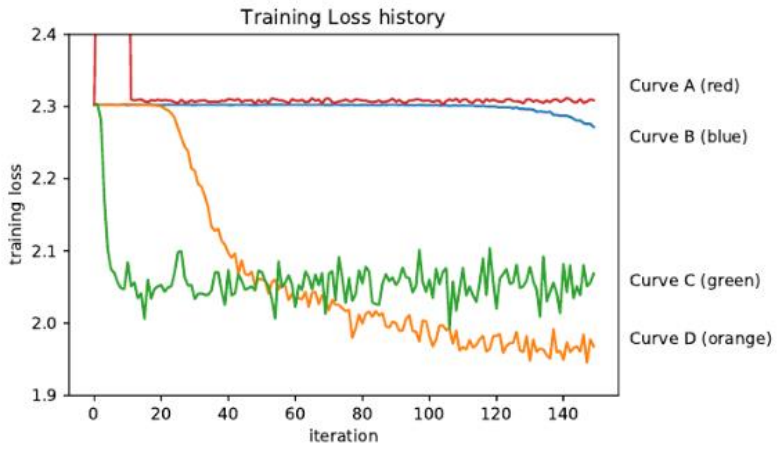
$\frac{32-6+2\cdot1}{2} + 1 = 14 + 1 = 15$ (1 pt.)

$15 \times 15 \times 8$ (1 pt.)

5. (1 point) Explain why we need activation functions.

Solution:

Without non-linearities, our network can only learn linear functions, because the composition of linear functions is again linear.



Solution:

Curve A: 4e-1 = 0.4 (Learning Rate is way too high)

Curve B: 2e-5 = 0.00002 (Learning Rate is too low)

Curve C: 8e-3 = 0.008 (Learning Rate is too high)

Curve D: 3e-4 = 0.0003 (Good Learning Rate)

Part II: Short Questions (12 points)

1. (2 points) You're training a neural network and notice that the validation error is significantly lower than the training error. Name two possible reasons for this to happen.

Solution:

The model performs better on unseen data than on training data - this should not happen under normal circumstances. Possible explanations:

- Training and Validation data sets are not from the same distribution
- Error in the implementation
- ...

- √ **Batch Normalization uses two trainable parameters that allow the network to undo the normalization effect of this layer if needed.**
- √ **Batch Normalization makes the gradients more stable so that we can train deeper networks.**
- √ **At test time, Batch Normalization uses a mean and variance computed on training samples to normalize the data.**
- √ **Batch Normalization has learnable parameters.**

4. (2 points) Which of the following optimization methods use first order momentum?
   - ☐ Stochastic Gradient Descent
   - √ **Adam**
   - ☐ RMSProp
   - ☐ Gauss-Newton

5. (2 points) Making your network deeper by adding more parametrized layers will always...
   - √ **slow down training and inference speed.**
   - ☐ reduce the training loss.
   - ☐ improve the performance on unseen data.
   - √ **(Optional: make your model sound cooler when bragging about it at parties.)**

Part I: Multiple Choice (10 points)

1. (2 points) To avoid overfitting, you can...
   - ☐ increase the size of the network.
   - √ **use data augmentation.**
   - ☐ use Xavier initialization.
   - √ **stop training earlier.**

2. (2 points) What is true about Dropout?
   - ☐ The training process is faster and more stable to initialization when using Dropout.
   - ☐ You should not use weaky ReLu as non-linearity when using Dropout.
   - √ **Dropout acts as regularization.**
   - √ **Dropout is applied differently during training and testing.**

3. (2 points) What is true about Batch Normalization?
   - √ **Batch Normalization uses two trainable parameters that allow the network to undo the normalization effect of this layer if needed.**