Deep Learning Theoretical Assignment

Abhimanyu Bhowmik

Deep Learning

February 2, 2024

Dataset

1. The data used to develop a neural network can be split into 3 categories. What are they?

Training data, validation data, and test data.

2. Detail the role of each data category.?

- **Training data:** Used to train the neural network by adjusting its parameters to learn patterns and relationships in the data.
- **Validation data:** Used to fine-tune the model during training and make decisions on hyperparameters to prevent overfitting.
- **Testing data:** Used to evaluate the final model's performance on unseen data, providing an unbiased assessment of generalization ability.

3. How are they distributed? Give approximately the percentages for each category relative to the total data.

- Training data : Around 70-80% of the total dataset.
- Validation data : Around 10-15% of the total dataset.
- Testing data : The remaining 10-15% of the total dataset.

4. We distinguish 3 types of segmentation: Semantic vs. Instance vs. Panopticon. What is the difference between these three types of segmentation?

- **Semantic segmentation:** Assigns a specific label to each pixel in an image, classifying objects or regions into predefined categories (e.g., road, person, car).
- **Instance segmentation:** Extends semantic segmentation by distinguishing individual instances of objects, providing a unique label for each object instance within a category.
- **Panoptic segmentation:** Combines semantic and instance segmentation, aiming to label every pixel with a class label (semantic segmentation) and a unique instance ID

(instance segmentation), covering both ”stuff” (non-countable objects) and ”things” (countable objects). In a cityscape image, panoptic segmentation would label every pixel with a class (e.g., road, building) and provide a unique ID for each instance of countable objects (e.g., cars, bicycles).

5. What type of segmentation concerns us for TP? Semantic? Instance? Panopticon?

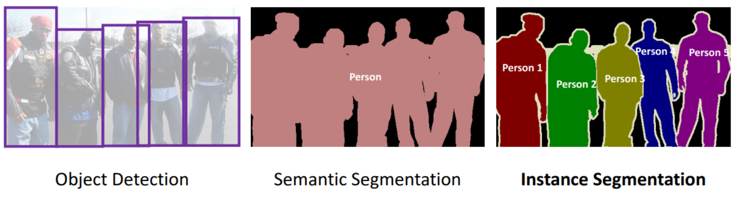
In our practical, we are dealing with semantic segmentation of horizon.

6. The data is usually first labelled/annotated in order for the algorithm to understand what the outcome needs to be. For the classification problem, each image is annotated according to its class. For the detection problem, each image is associated with bounding boxes positions and sizes. For the segmentation problem, what form does this labelling take?

For the segmentation problem, the data is typically labelled using pixel-level annotations. This means that each pixel in the image is assigned to a specific class or category.

There are two main types of pixel-level annotations:

- **Binary segmentation masks:** Masks are simply binary images(objects as 1 and background as 0) that represent the semantic segmentation of an object.
- **Instance segmentation masks:** They not only identify the objects in an image, but they also identify individual instances of the same object.

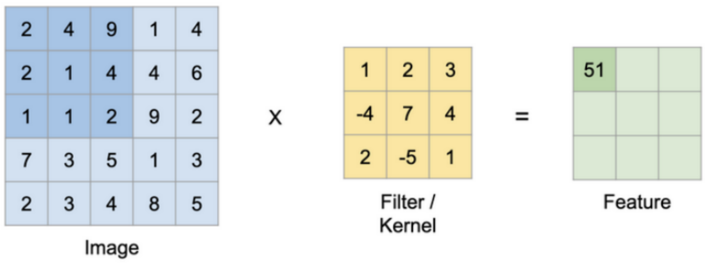


Model

1. What is the main function of the convolution layer?

The main function of the convolution layer in a neural network is to extract features from input data by applying filters to capture local patterns and spatial hierarchies, enabling the network to learn and recognize complex patterns in a hierarchical manner.

2. Complete the feature matrix by detailing the calculations.



For calculating the 2D Convolution between those two given matrices we will calculate the Hadamard product between the Image patches (deep blue selected part) with the kernel. We will sum all the values of the matrix and we will get the first value of the feature matrix. Then we will loop through the whole matrix patch by patch.

To illustrate this we can do the step-by-step operation for the initial value.

Step 1: We have two matrices for the convolution operation. The first matrix is the kernel projected onto the original matrix. The second matrix is the filter/kernel. For the first case:

$$\text{Image Patch} = \begin{bmatrix} 2 & 4 & 9 \\ 2 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix} \quad \text{Kernel} = \begin{bmatrix} 1 & 2 & 3 \\ -4 & 7 & 4 \\ 2 & -5 & 1 \end{bmatrix}$$

Step 2: To calculate the Hadamard product between the Image Patch and Kernel, we perform element-wise multiplication:

$$\text{Image Patch}\odot\text{Kernel} = \begin{bmatrix} 2 & 4 & 9 \\ 2 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix} \odot \begin{bmatrix} 1 & 2 & 3 \\ -4 & 7 & 4 \\ 2 & -5 & 1 \end{bmatrix} = \begin{bmatrix} 2\cdot 1 & 4\cdot 2 & 9\cdot 3 \\ 2\cdot(-4) & 1\cdot 7 & 4\cdot 4 \\ 1\cdot 2 & 1\cdot(-5) & 2\cdot 1 \end{bmatrix} = \begin{bmatrix} 2 & 8 & 27 \\ -8 & 7 & 16 \\ 2 & -5 & 2 \end{bmatrix}$$

Step 3: In the next step, calculate the sum of all elements of the output matrix.

$$\text{Sum of elements} = \sum_{i=1}^{3}\sum_{j=1}^{3}(\text{Image Patch} \odot \text{Kernel})_{ij} = 2 + 8 + 27 - 8 + 7 + 16 + 2 - 5 + 2 = 51$$

We need to repeat this process for each image patch in the Image matrix. Finally, we get:

$$\text{Image} * \text{Kernel} = \begin{bmatrix} 2 & 4 & 9 & 1 & 4 \\ 2 & 1 & 4 & 4 & 6 \\ 1 & 1 & 2 & 9 & 2 \\ 7 & 3 & 5 & 1 & 3 \\ 2 & 3 & 4 & 8 & 5 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ -4 & 7 & 4 \\ 2 & -5 & 1 \end{bmatrix} = \begin{bmatrix} 51 & 31 & 15 \\ 66 & 49 & 53 \\ 20 & 101 & -2 \end{bmatrix}$$

3. The convolution layer allows weight sharing. Explain what this involves. What is his interest?

Weight sharing is a crucial concept in CNNs that enables them to learn relevant features from input data effectively and efficiently. By sharing the same weights across multiple neurons, CNNs can identify patterns regardless of their location, leading to translation invariance and parameter efficiency

4. For deep learning, we stack a significant number of convolution layers, why do we do this?

Stacking convolution layers enables the network to progressively extract increasingly higher-level and abstract features from the input data. By iteratively applying filters to the output of previous layers, the network learns to recognize more complex patterns and capture the underlying structure and composition of the input data. Operations like max pooling or average pooling and stacked convolution layers can reduce the dimensionality of the feature maps while preserving the most significant information.

5. What is the point of the batch normalisation layer? Explain the batch normalization process.

Batch normalization (BN) is a regularization technique that aims to improve the training of deep neural networks by normalizing the activations of each layer. This normalization helps to stabilize the training process by reducing internal covariate shift, which is a phenomenon where the distribution of activations changes throughout training. BN also helps to improve the generalization performance of deep neural networks by making them less sensitive to initialization and input distribution.

The batch normalization process can be summarized as follows:

a) **Compute mean and standard deviation**: For each activation in the current layer, compute the mean and standard deviation across the current batch of inputs.

b) **Normalize activations**: Subtract the mean and divide by the standard deviation of each activation.

c) **Scale and shift**: Apply learnable scaling and shifting parameters, gamma and beta, respectively, to the normalized activations.

d) **Output**: Return the scaled and shifted activations.

By normalizing the activations in this way, BN helps to stabilize the training process and improve the generalization performance of deep neural networks.

6. What is the point of the layer normalisation layer? Explain the layer normalization process.

The purpose of Layer Normalization is to address the issue of internal covariate shift, aiming to improve the training stability and convergence of deep neural networks. Internal covariate shift refers to the change in the distribution of the input to a network's activation functions across training iterations, which can slow down the training process.

Layer Normalization helps in training deeper networks and allows for more straightforward application of gradient-based optimization algorithms.

The layer normalization process involves the following steps for each input feature in a given layer:

1. **Compute the Mean and Variance:**

Calculate the mean μ and variance σ^2 of the input values across the training data.

2. **Normalize the Inputs:**

Normalize the input values x using the mean and variance:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

where ϵ is a small constant added for numerical stability, making the distribution of each feature more stable and reducing the internal covariate shift.

3. **Scale and Shift:**

Introduce learnable parameters γ (scaling) and β (shifting):

$$y = \gamma \hat{x} + \beta$$

This step allows the network to learn the optimal scale and shift for each feature.

The layer normalization process is applied independently to each input feature, making it different from batch normalization, which operates over entire mini-batches.

7. Why do we use batch normalization in our case rather than layer normalization?

Batch Normalization (BN) is preferred over Layer Normalization (LN) in our case due to its effectiveness in handling batch statistics. In our case, we are processing images in batches, and BN normalizes activations based on batch-specific mean and variance, making it well-suited for tasks with varying input distributions.

BN also acts as regularization, aiding in preventing overfitting in image segmentation models. Its ability to handle internal covariate shift and adapt to mini-batch variability makes BN advantageous for the diverse content and structures encountered in image segmentation tasks.

Additionally, BN is more flexible with varying batch sizes, which is common in image segmentation applications. While the choice between BN and LN depends on factors like model architecture and task characteristics, BN's empirical success in image segmentation tasks underscores its practical utility.

8. Give some types of activation functions.

Several activation functions are commonly used in neural networks, including Convolutional Neural Networks (CNNs). Here are some types of activation functions:

1. **Sigmoid Activation Function:** Squeezes the output between 0 and 1, commonly used in the output layer for binary classification as it represents probabilities.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

2. **Hyperbolic Tangent (tanh) Activation Function:** Similar to the sigmoid, but with a range between -1 and 1, helping mitigate the vanishing gradient problem in deep networks.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

3. **Rectified Linear Unit (ReLU):** Simple and computationally efficient; widely used due to its effectiveness in promoting sparse activations.

$$f(x) = \max(0, x)$$

4. **Leaky ReLU:** Addresses the ”dying ReLU” problem by allowing a small negative slope for negative input values, preventing neurons from becoming inactive.

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases}$$

where α is a small positive constant.

5. **Parametric ReLU (PReLU):** Similar to Leaky ReLU but with the slope as a learnable parameter, providing more flexibility during training.

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases}$$

where α is a learnable parameter.

6. **Softmax Activation Function:** Converts raw scores into probabilities, suitable for multiclass classification, ensuring the output values sum to 1 and represent class probabilities. Used in the output layer for multiclass classification problems.

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

where N is the number of classes.

These activation functions introduce non-linearity to the neural network, enabling it to learn complex patterns and relationships in the data. The choice of activation function depends on the specific requirements and characteristics of the problem being addressed.

9. What are activation functions used for?

Activation functions in neural networks are used to introduce non-linearity. This allows the network to learn complex patterns beyond simple linear relationships, crucial for tasks like image recognition. Activation functions help maintain the signal during training and prevent vanishing gradients.

In short, they add power and flexibility to neural networks, enabling them to tackle complex tasks.

1 Loss function

1. In the formula below of binary cross enentropy , what do N, y_i, $p(y_i)$ represent? $-\frac{1}{N}\sum_{i=1}^{N}[y_i log(p(y_i)) + (1-y_i)log(1-p(y_i))]$

In the binary cross-entropy formula:

$$-\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p(y_i)) + (1-y_i)\log(1-p(y_i))]$$

- N represents the total number of samples or data points in the dataset. - y_i is the true label (ground truth) for the i-th sample, indicating whether it belongs to class 1 (positive) or class 0 (negative). - $p(y_i)$ is the predicted probability that the i-th sample belongs to class 1 according to the model.

The formula calculates the binary cross-entropy loss, measuring the dissimilarity between the true labels (y_i) and the predicted probabilities ($p(y_i)$) for each sample in the dataset. This loss function is commonly used in binary classification tasks.

2. What is $-y_i log(p(y_i))$ equal to in the following cases?

2.1. For the pixel whose target is equal to 1 and prediction is equal to 1?

2.2. For the pixel whose target is equal to 1 and prediction tend to 0?

In the binary cross-entropy loss term $-y_i \log(p(y_i))$, the expression evaluates differently based on the cases you mentioned:

1. For the pixel whose target is equal to 1 and prediction is equal to 1 (perfect prediction):
- $-y_i \log(p(y_i))$ becomes $-1 \cdot \log(1) = 0$.
- The loss is 0 because the model's prediction aligns perfectly with the true target.

2. For the pixel whose target is equal to 1 and prediction tends to 0 (imperfect prediction):
- As $p(y_i)$ tends to 0, $-y_i \log(p(y_i))$ approaches positive infinity.
- The loss becomes very large, indicating a significant disagreement between the predicted probability and the true label.

In summary, in the context of binary cross-entropy loss, perfect alignment between prediction and true label results in a loss of 0, while increasing disagreement leads to higher loss values, approaching infinity as the prediction tends to 0.

3. what do $-y_i log(p(y_i))$ représent?

The term $-y_i \log(p(y_i))$ measures the contribution of a single data point to the binary cross-entropy loss, penalizing the model more when its predicted probability diverges from the true label (1), and encouraging accurate predictions for the positive class.

4. What is $-(1 - y_i)log(1 - p(y_i))$ equal to in the following cases?

4.1. For the pixel whose target is equal to 0 and prediction is equal to 1?

4.2. For the pixel whose target is equal to 0 and prediction tend to 0?

In the binary cross-entropy loss term $-(1 - y_i) \log(1 - p(y_i))$, the expression evaluates differently based on the cases you mentioned:

1. For the pixel whose target is equal to 0 and prediction is equal to 1 (imperfect prediction):
- $-(1 - y_i) \log(1 - p(y_i))$ becomes $-(1 - 0) \log(1 - 1) = 0$.
- The loss is 0 because the model's prediction aligns perfectly with the absence of the true

target.

2. For the pixel whose target is equal to 0 and prediction tends to 0 (better prediction):
- As $1 - p(y_i)$ tends to 1, $-(1 - y_i)\log(1 - p(y_i))$ approaches 0.
- The loss becomes smaller, indicating better alignment between the predicted probability and the true absence of the target.

In summary, in the context of binary cross-entropy loss, a perfect prediction for the absence of the target results in a loss of 0, while increasing misalignment leads to higher loss values.

5. What do $-(1 - y_i)log(1 - p(y_i))$ représent?

The term $-(1 - y_i)\log(1 - p(y_i))$ in the binary cross-entropy loss function represents the contribution of a single data point to the overall loss. Breaking it down:

- y_i is the true label (ground truth) for the data point, taking a value of 0 or 1. - $p(y_i)$ is the predicted probability that the data point belongs to class 1 according to the model.

The term $-(1 - y_i)\log(1 - p(y_i))$ can be understood as follows:

- When the true label (y_i) is 0, this term penalizes the model more as the predicted probability $(p(y_i))$ diverges from 0. In other words, it measures how well the model predicts the negative class. - When the true label (y_i) is 1, this term becomes 0, as there is no penalty for predicting the positive class correctly.

The negative logarithm $\log(1 - p(y_i))$ penalizes predictions that are far from 0, encouraging the model to assign lower probabilities to true negative instances and higher probabilities to true positive instances. This term is summed and averaged over the entire dataset to compute the binary cross-entropy loss.

6. What is binary cross entropy supposed to measure?

Binary cross-entropy is designed to measure the dissimilarity between predicted probabilities and true binary labels in a classification task. Specifically, it quantifies how well the predicted probabilities align with the true labels, penalizing deviations from the actual classes. This loss function is commonly used in binary classification problems to guide the training of machine learning models, aiming to minimize the discrepancy between predicted and true outcomes.

7. The code uses BCEWithLogitsLoss instead of BCELoss. What is the difference between the two functions? why should we use BCEWithLogitsLoss?

BCELoss (Binary Cross-Entropy Loss) expects raw scores (logits) as input and requires an additional sigmoid activation layer in the network to convert logits to probabilities.

BCEWithLogitsLoss is more stable than BCELoss because it internally applies the sigmoid activation, ensuring numerical stability, avoiding explicit sigmoid issues, and simplifying network architecture. Additionally, it enables the use of gradient clipping techniques, which can further enhance stability during training.

8. What are the drawbacks of binary cross enntropy?

Here's a shorter summary of Binary Cross-Entropy (BCE) drawbacks:

a) **Sensitive to outliers:**Punishes confident wrong predictions heavily, which can be bad with outliers or imbalanced data.

b) **Not directly interpretable:** Doesn't tell you the exact nature of errors, making debugging difficult.

c) **Not accuracy-focused:** Minimizing BCE doesn't guarantee high accuracy.

9. Cite other loss functions suitable for image segmentation.

Several loss functions are suitable for image segmentation tasks in addition to binary cross-entropy. Some commonly used ones include:

a) **Dice Loss:** - Measures the overlap between predicted and true segmentation masks, calculated as $\frac{2\times\text{intersection}}{\text{union}+\epsilon}$, where ϵ is a small constant to avoid division by zero.

b) **Jaccard/IoU Loss:** - Computes the Intersection over Union (IoU) between predicted and true masks, given by $\frac{\text{intersection}}{\text{union}+\epsilon}$.

c) **Focal Loss:** - Addresses class imbalance by down-weighting well-classified examples, defined as $-(1-p_t)^\gamma \log(p_t)$, where p_t is the predicted probability and γ is a focusing parameter.

d) **Lovász-Softmax Loss:** - Based on the Lovász extension of submodular set functions, designed for non-differentiable metrics like IoU.

e) **Weighted Cross-Entropy Loss:** - Assigns different weights to different classes to address class imbalance.

f) **Boundary Loss:** - Focuses on minimizing errors in the boundary regions of segmented objects.

The choice of loss function depends on the specific characteristics of the segmentation task, the nature of the data, and the desired optimization objectives. Experimentation and validation are crucial for identifying the most effective loss function for a given scenario.

Training

1. What is the difference between the training phase and the inférence phase?

Training Phase: Training a CNN is like putting it through school. It devours massive datasets like textbooks, adjusting its internal connections to absorb the knowledge. This grueling phase equips the model with the ability to recognize patterns and relationships.

Inference Phase: Once the ”learning” is done, it’s test time! The model encounters new, unseen data, but now it can leverage its acquired knowledge to make predictions and inferences. Like a student applying their studies to solve a problem, the CNN uses its trained connections to analyze the new data and produce an output, showcasing its understanding of the world it learned from.

Training builds the foundation, inference applies it, making the CNN a powerful problem-solving tool.

2. The training task of a network is composed of gradient descent and back propagation. What is the goal of gradient descent?

The goal of gradient descent in training a neural network, including a Convolutional Neural Network (CNN), is to minimize the loss function. It does so by iteratively adjusting the model parameters in the opposite direction of the gradient of the loss with respect to those parameters. This process helps the model converge towards the optimal set of parameters that result in the lowest possible loss, improving the model’s performance on the given task.

3. What is the goal of back propagation?

The goal of backpropagation is to compute the gradients of the loss function with respect to the weights of the neural network. It involves propagating the error backward from the output layer to the input layer, using the chain rule of calculus. This process allows the network to update its weights during the training phase using the gradients calculated, helping the model learn and improve its performance over time. Essentially, backpropagation enables the adjustment of model parameters in the direction that minimizes the overall loss.

4. The new weights are computed with previous weights and gradients. Give the relation between the new weight, the previous weights and the cost function $Q : w \to Q(w)$.

The update rule for the weights in the context of gradient descent during training can be expressed as:

$$w_{\text{new}} = w_{\text{old}} - \alpha \cdot \nabla Q(w_{\text{old}})$$

Here: - w_{new} is the new set of weights.
- w_{old} is the previous set of weights.
- α is the learning rate, a hyperparameter that determines the size of the step taken during each iteration of gradient descent.
- $\nabla Q(w_{\text{old}})$ is the gradient of the cost function Q with respect to the weights at the current iteration.

This update rule ensures that the weights are adjusted in the direction opposite to the gradient, aiming to minimize the cost function $Q(w)$ during the training process.

5. Apply the gradient descent algorithm to the following simple function. Detail the calculation steps. $f : x \to 3x^2 + 5x + 4$

Applying the gradient descent algorithm to minimize the given function $f(x) = 3x^2+5x+4$. The gradient descent update rule is given by:

$$x_{\text{new}} = x_{\text{old}} - \alpha \cdot \nabla f(x_{\text{old}})$$

Here, $\nabla f(x)$ is the gradient of $f(x)$ with respect to x, and α is the learning rate.

Step 1: Initialize Parameters - Choose an initial guess for x_{old}.
- Choose a learning rate α.

Step 2: Calculate Gradient

$$\nabla f(x) = \frac{df}{dx} = 6x + 5$$

Step 3: Update Parameters

$$x_{\text{new}} = x_{\text{old}} - \alpha \cdot (6x_{\text{old}} + 5)$$

Step 4: Iterate Repeat steps 2 and 3 until convergence or a specified number of iterations.

Now, let's perform the first iteration for demonstration:

- Initial guess: $x_{\text{old}} = 1$ - Learning rate: $\alpha = 0.01$

Iteration 1:

$$\nabla f(1) = 6(1) + 5 = 11$$

$$x_{\text{new}} = 1 - 0.01 \cdot 11 = 0.89$$

Iteration 2:

$$\nabla f(0.89) = 6(0.89) + 5 = 10.34$$

$$x_{\text{new}} = 0.89 - 0.01 \cdot 10.34 = 0.7856$$

Iteration 3:

$$\nabla f(0.7856) = 6(0.7856) + 5 = 10.714$$

$$x_{\text{new}} = 0.7856 - 0.01 \cdot 10.714 = 0.6784$$

In subsequent iterations, we repeat steps 2 and 3 using x_{new} as the updated value of x_{old} until the algorithm converges to a minimum or reaches the maximum number of iterations. Adjust the learning rate as needed for convergence and stability.

6. In deep learning, gradient descent is said to be stochastic. Explain why we call the algorithm stochastic?

In deep learning, the term ”stochastic” in the context of gradient descent refers to the fact that the algorithm doesn’t necessarily use the entire dataset to compute the gradient at each iteration. Instead, it typically employs a randomly selected subset or a single data point to estimate the gradient. This introduces randomness in the parameter updates, making the optimization process less deterministic compared to traditional gradient descent.

7. What is the batch size?

The batch size refers to the number of training samples utilized in one iteration. In other words, it is the number of data points or samples from the dataset that are processed together in each forward and backward pass.

8. why is the batch size limited in practice?

The batch size in training is limited due to constraints such as available memory and computational efficiency. Larger batch sizes require more memory and may lead to diminished computational efficiency. Smaller batch sizes often result in faster training due to more frequent parameter updates. Extremely small batch sizes introduce noise and get stuck in local minima. The choice of batch size impacts convergence behavior, and researchers often experiment to find a balance that works best for their specific task, dataset, and computational resources.

9. What is the epochs number?

The number of epochs represents the total number of times a machine learning model iterates over the entire training dataset during training.

10. What is the steps per epochs?

The steps per epoch is the number of iterations or batches processed in one epoch during the training of a machine learning model.

11. What is the relation between the size of training dataset, the batch size and the steps number per epoch?

The relation between the size of the training dataset (N), the batch size (B), and the steps per epoch (S) is given by the formula:

$$S = \frac{N}{B}$$

where S represents the steps per epoch, N is the size of the training dataset, and B is the batch size. This formula calculates the number of batches needed to cover the entire training dataset in one epoch.

12. What is the effect of the learning rate on the evolution of the cost function?

The learning rate in training a machine learning model influences the convergence and evolution of the cost function; a higher learning rate may cause faster convergence but risks overshooting the optimal solution, while a lower learning rate may lead to slower convergence or potential convergence to a suboptimal solution. The choice of an appropriate learning rate is crucial for effective model training.

13. What is the effect of the batch size on the evolution of the cost function?

The batch size influences the evolution of the cost function during training by affecting the frequency of parameter updates; larger batch sizes provide smoother convergence but may converge to suboptimal solutions, while smaller batch sizes can lead to faster convergence but introduce more noise and variability in the optimization process. The choice of batch size involves a trade-off between computational efficiency and convergence behavior.

14. What happens when the number of epochs is too large?

When the number of epochs is too large, there is a risk of overfitting, where the model learns the training data too well, capturing noise and specifics that do not generalize to new data. This can result in reduced model performance on unseen data and increased computational costs without substantial benefits in learning.