CNN

Version 0.2

**May 11, 2025**

Authors:

Mohammed Khalil mk3441@att.com

Contents

[Document Control 5](#_Toc197804646)

[1 Introduction 6](#_Toc197804647)

[1.1 Mathematical Foundation 6](#_Toc197804648)

[1.1.1 Hadamard product 6](#_Toc197804649)

[1.1.2 Similarity Concept 6](#_Toc197804650)

[1.1.2.1 Dot Product 6](#_Toc197804651)

[1.1.3 Jacobian Matrix 7](#_Toc197804652)

[1.1.3.1 Single Value Scalar Function 7](#_Toc197804653)

[1.1.4 Multiple Value Scaler Function 8](#_Toc197804654)

[1.1.5 Jacobian Matrix 8](#_Toc197804655)

[1.1.6 Binary Cross-Entropy 9](#_Toc197804656)

[1.1.7 Sigmoid Function 10](#_Toc197804657)

[1.1.7.1 Sigmoid Penalization Mechanism Analysis 11](#_Toc197804658)

[1.1.8 Softmax Function 11](#_Toc197804659)

[1.1.8.1 Softmax Penalization Mechanism Analysis 13](#_Toc197804660)

[1.2 Updating the Weight Matrix W with sigmoid + MSE 14](#_Toc197804661)

[1.2.1 Define Variables and Shapes 14](#_Toc197804662)

[1.2.2 Forward Propagation 14](#_Toc197804663)

[1.2.2.1 Compute Pre-activation Output Z 15](#_Toc197804664)

[1.2.3 Backward Propagation 15](#_Toc197804665)

[1.2.3.1 Sigmoid Activation Function 15](#_Toc197804666)

[1.2.3.2 Mean Square Error (MSE) 15](#_Toc197804667)

[1.2.3.3 Gradient of Loss : 16](#_Toc197804668)

[1.2.3.4 Compute Jacobian of sigmoid 16](#_Toc197804669)

[*1.2.3.5* Compute : 17](#_Toc197804670)

[*1.2.3.6* Chain Rule 18](#_Toc197804671)

[*1.2.3.7* Chain Rule 18](#_Toc197804672)

[1.2.3.8 Update Weights W Using Gradient Descent 19](#_Toc197804673)

[1.2.3.9 Update Biases b Using Gradient Descent 19](#_Toc197804674)

[1.3 Updating the Weight Matrix W with softmax and MSE 19](#_Toc197804675)

[1.3.1 Define Variables and Shapes 20](#_Toc197804676)

[1.3.2 Forward Propagation Calculation 20](#_Toc197804677)

[1.3.3 Backpropagation 21](#_Toc197804678)

[1.3.3.1 Compute Softmax Function 21](#_Toc197804679)

[1.3.3.2 Compute Mean Square Error (MSE) Loss 21](#_Toc197804680)

[1.3.3.3 Gradient of Loss : 21](#_Toc197804681)

[1.3.3.4 Compute Jacobian of softmax 22](#_Toc197804682)

[1.3.3.5 Compute Chain Rule 23](#_Toc197804683)

[1.3.3.6 Compute 24](#_Toc197804684)

[25](#_Toc197804685)

[1.3.3.7 Compute 25](#_Toc197804686)

[1.3.3.8 Updating W with Gradient Descent 26](#_Toc197804687)

[1.3.3.9 Updating b with Gradient Descent 26](#_Toc197804688)

[1.4 Updating the weight Matrix with softmax and Cross-Entropy 26](#_Toc197804689)

[1.4.1 Cross Entropy Loss 27](#_Toc197804690)

[1.4.2 Define Variables and Shapes 27](#_Toc197804691)

[1.4.3 Forward Propagation 27](#_Toc197804692)

[1.4.3.1 Compute Pre-activation Output Z 27](#_Toc197804693)

[1.4.4 Backpropagation 28](#_Toc197804694)

[1.4.4.1 softmax function 28](#_Toc197804695)

[1.4.4.2 Cross Entropy Loss 28](#_Toc197804696)

[1.4.4.2.1 Definition of Cross-Entropy Loss: 29](#_Toc197804697)

[1.4.4.3 The Jacobian Matrix of softmax 30](#_Toc197804698)

[1.4.4.4 Compute Chain Rule 31](#_Toc197804699)

[1.4.4.5 Compute Chain Rule 31](#_Toc197804700)

[1.4.4.6 Example of M = 3 Classes 33](#_Toc197804701)

[1.4.4.7 Compute 34](#_Toc197804702)

[34](#_Toc197804703)

[1.4.4.8 Compute 35](#_Toc197804704)

[1.4.4.9 Update W (Gradient Descent) 36](#_Toc197804705)

[1.4.4.10 Update b (Gradient descent) 36](#_Toc197804706)

[1.5 Updating the Weight Matrix W with sigmoid and Cross-Entropy 36](#_Toc197804707)

[1.5.1 Compute Pre-activation Output Z 36](#_Toc197804708)

[1.5.2 Backpropagation 37](#_Toc197804709)

[1.5.2.1 Binary Cross Entropy Loss 37](#_Toc197804710)

[1.5.2.2 Compute 37](#_Toc197804711)

[1.5.2.3 Sigmoid Activation Function 38](#_Toc197804712)

[1.5.2.4 Compute Jacobian of sigmoid 38](#_Toc197804713)

[1.5.2.5 Compute 40](#_Toc197804714)

[1.5.2.6 Compute 41](#_Toc197804715)

[1.5.2.7 Compute 41](#_Toc197804716)

[1.5.2.8 Updated Parameters (Gradient Descent) 41](#_Toc197804717)

[1.5.3 Example softmax with cross entropy 42](#_Toc197804718)

[1.5.3.1 Scenario 42](#_Toc197804719)

[1.5.3.2 Softmax + on-Hot Cross Entropy 42](#_Toc197804720)

[1.5.4 Example sigmoid with cross entropy 43](#_Toc197804721)

[1.5.4.1 Scenario 43](#_Toc197804722)

[1.5.4.2 softmax + on-Hot Cross Entropy 43](#_Toc197804723)

[1.6 Dynamic Edge-Conditional Filters in Convolutional Neural Network On Graphs 44](#_Toc197804724)

[1.6.1 Dynamic Edge-Conditioned Convolution (ECC) Formula 44](#_Toc197804725)

[1.6.2 Dynamic Filters Procedure 45](#_Toc197804726)

[1.6.3 Numerical example for greyscale image 48](#_Toc197804727)

[1.6.3.1 Assumptions 48](#_Toc197804728)

[1.6.3.2 Procedure Steps 48](#_Toc197804729)

[2 Acronyms 51](#_Toc197804730)

Document Control

Below are the publication date, author’s initials, and reason for publication for the current and any previous versions of this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Publication Date** | **Author** | **Reason for Publication** |
| 0.1 | <27 SEPTEMPER 2022> | MK,PV,GB,HM | NWDAF architecture description, phases, requirements |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

The "Attention Is All You Need" paper by Vaswani et al. introduced the Transformer model, a novel architecture designed for sequence-to-sequence tasks such as translation. The Transformer model revolutionized natural language processing by eliminating the need for recurrent or convolutional structures, instead relying entirely on attention mechanisms. This shift enables the model to capture global dependencies in input sequences more efficiently.

This document provides an in-depth explanation of the mathematical foundations underpinning the Transformer, focusing on key concepts like the attention mechanism, multi-head attention, positional encoding, and feed-forward networks.

## Mathematical Foundation

### Hadamard product

This section describes Hadamard product

to vector

=

### Similarity Concept

#### Dot Product



### Jacobian Matrix

#### Single Value Scalar Function

f(x) = ,  **= 2****x**

The first derivative of the function f of the independent variable x is the slope of the function f at certain value of x. Or how fast the function f changing at this particular x value. So if we change the value at x by  x, so x will cause a change of the output f(x). The slope of f(x) at x when it changes by x measures the sensitivity of changing the function f(x) when we change x by x.

**Slope of f(x) at x**

**x**

**x+ x** 1

**x**1

if we extend the concept of derivative to the concept of gradient, gradient means the derivative of scalar function to multiple independent variables (,, ….) feeding to the function:

### Multiple Value Scaler Function

the gradient is given as = =

is the partial derivative of the function f for each one of the independent variables of the vector . is the sensitivity or the rate function f changes in

Example f(, ) =

= 2, measure the sensitivity or the rate the function f changes with respect to

= 2 , measure the sensitivity or the rate the function f changes with respect to

=

### Jacobian Matrix

Instead of dealing with a single scaler function, we use a vector valued function. So instead of having scaler output we have a vector output. The figure below shows the output of 3 scaler functions on the top of each other:

***(, ) =***

***(, ) =***

***(, ) =***

**X1**

**X2**

***(,)***

***(,)***

*(*

***(,)***

The Jacobian matrix is expressed as follows:

***(***

Since the gradient of the function f with respect to is expressed as:

=  **=**

So, the Jacobian matrix J is stack of the transpose of each in the top of each other. Jacobian matrix measures the sensitivity or the rate of change of each function for each variable in vector **.**

J = =

is defined as the sensitivity of function  output to the change of .

### Binary Cross-Entropy

+ (

In machine learning:

: is the true label (either 0 or 1)

: is the model’s predicted probability

**Key Insight:**

The binary entropy-based loss punishes:

* **Low confidence in the correct class**: If there is low confidence in the correct class Loss will be high.
* **High-confidence in the wrong class:** if there is high confidence in the wrong class the Loss will be high

**Assume:**

|  |  |  |
| --- | --- | --- |
| **class** | **Label/class** | **Prediction** |
| Dog | 1 | .9 |
| Cat | 0 | .8 |
| Rabbit | 0 | .2 |

**Example 1: Input is a dog and prediction is good(class 1)**

True Label = 1

Prediction

+ 0x

**Small loss->** the model is %90 confidence that the input is a dog. Therefore it is %10 confidence the input is not a dog (which is correct in this case). The confidence is is high and loss is low.

**Example 2: Input is a dog but prediction is bad**

True Label = 1

Prediction

+ 0x

**High loss**-> the model is %20 confidence that the input is a dog. Therefore it is %80 confidence the input is not a dog (which is wrong in this case). The confidence is low and loss is hight.

**Example 3: Input is not dog (class 0)**

True Label = 0

Prediction

+ 0x

***moderate loss***-> the model is %35 confidence that the input is a dog. Therefore it is %70 confidence the input is not a dog (which is correct in this case). The confidence is moderate and loss is moderate.

### Sigmoid Function

Sigmoid function is used in multi-labels or multi-classes classification where:

* Each class is treated independently.
* An input can belong to multiple classes at the same time.
* Each output node gives the probability of that class independently of the others
* The activation is:

|  |  |  |
| --- | --- | --- |
| **class** | **Label/class** | **Prediction** |
| Dog | 1 | .9 |
| Cat | 1 | .8 |
| Rabbit | 0 | .2 |

Then Binary cross-entropy is computed for each class separately:

+ (

So, here Dog, Cat are correct, and sigmoid supports that:

L = -[1x

+ 1x

+ 0x

L = -[] = [-0.105-0.223-0.223] = .551

Each class contributes to the loss independently

#### Sigmoid Penalization Mechanism Analysis

There are 2 ways sigmoid penalizes prediction errors:

**Case 1:** True label (The class is present)

You want close to 1

If the model gives a low probability (say 0.2), then the term becomes:

L = -1x

High penalty for missing a relevant class

**Case 2:** True label (The class is not present)

You want close to 0

If the model gives a high probability (say 0.8), then the term becomes:

L = -(1-0)x

High penalty for predicting an irrelevant class as present.

### Softmax Function

Softmax function is used in single-labels or single-class classification where:

* The output for one class depends on the logits of all other classes.
* An input can only belong to a single class.
* The probability of class i, , depends on all other values. Each output probability in softmax is computed using sll other logits. Increasing the probability of one class can decrease the probabilities of other classes. They are in competition.
* The term “logit” comes from log-odds in statistics. However, in deep learning, we use it more generally to means: the value produced by the last linear layer of neural network before applying softmax (for multiclass) or sigmoid (for binary/multilabel).”
* The activation is:
* The logit is the raw score (or unnormalized prediction) output from the neural network before the any activation function such as sigmoid or softmax is applied.
* = weight vector for class i
* = input feature vector
* = bias for class i
* = logit for class j

|  |  |  |
| --- | --- | --- |
| **class** | **Label/class** | **Non normalized prediction** |
| Dog | 1 | 2.5 |
| Cat | 0 | 1.8 |
| Rabbit | 0 | 0.2 |

Then Binary cross-entropy is computed for each class separately:

1. **Softmax Calculation**

Given:

Z =

We compute softmax

We compute each term:

Total sum:

S = 12.18 + 6.05 + 1.22 = 19.45

No Calculate each probability:

1. **Cross Entropy Loss**

Only Dog is the correct class, so:

#### Softmax Penalization Mechanism Analysis

|  |  |  |
| --- | --- | --- |
| **class** | **Label/class** | **Prediction** |
| Dog | 1 | .9 |
| Cat | 1 | .08 |
| Rabbit | 0 | .02 |

So it appears that it penalize the log probability for the correct class.

**What about the wrong class:**

Even though we don’t penalize the wrong class, they do affect the loss ibdrectly through the softmax function**.**

**Why? Because softmax couples the prediction:**

So, if u increase the score for the wrong class (e,g, Cat). The dominator increases, which lower the probability for the correct class, that in turns, increase the loss.

**Visual Intuition**

Imagine a pie chart with 100% probability. Softmax distributes the probability across all classes:

* If the model gives 0.9 tp the correct class, the rest vonly get 0.1 to shar.
* If it gives high probability to the wrong lass, the correct class porbality drops, and loss increases.

So, the model is penalized for assigning high confidence to wrong classes-just not directly, but through the effect on the correct class.

## Updating the Weight Matrix W with sigmoid + MSE

Updating the weight Matrix W using the total loss in multi-label classification with the softmax+ MSE loss setup. We will do that for the case with:

* M output classes (e.g. Dog, Cat, Rabbit->M=3)
* N input features.
* X є : the input feature vector.
* W є : the weight matrix
* b є : the bias vector.
* Z є : logits
* Prediction outputs
* : true labels (0 or 1 for each class)

### Define Variables and Shapes

* Weight matrix W of shape (N,M):

є

* Input feature vector X of shape (M,1)

є

* Bias vector b of shape (N,1)

є

### Forward Propagation

#### Compute Pre-activation Output Z

Type : vector if one sample or Matrix if batch of samples

Z є

Z = WX + b

Where b є

=

=

### Backward Propagation

#### Sigmoid Activation Function

Where є

#### Mean Square Error (MSE)

**Type:** scalar

Now, we differentiate the loss function using chain rules

#### Gradient of Loss :

Type: vector є

=  (

=

#### Compute Jacobian of sigmoid

For diagonal terms i=j

For terms ij

0

#### Compute :

Z=WX+b

#### Chain Rule

=

=

#### Chain Rule

#### Update Weights W Using Gradient Descent

W = W -α

#### Update Biases b Using Gradient Descent

b = b - α

b = b - α

## Updating the Weight Matrix W with softmax and MSE

Updating the weight Matrix W using the total loss in single-label classification with the softmax+ MSE loss setup. We will do that for the case with:

* M output classes (e.g. Dog, Cat, Rabbit->M=3)
* N input features.
* X є : the input feature vector.
* W є : the weight matrix
* b є : the bias vector.
* Z є : logits
* Prediction outputs
* : true labels (0 or 1 for each class)

### Define Variables and Shapes

* Weight matrix W of shape (N,M):
* Input feature vector X of shape (N,1)
* Bias vector b of shape (N,1)

### Forward Propagation Calculation

Z = WX + b

Where b є

Z=

Z=

### Backpropagation

#### Compute Softmax Function

The softmax function is used to convert the logits into probability

The predicted probability for each class i is given by:

; i = 1,2,…….N

#### Compute Mean Square Error (MSE) Loss

Now, we differentiate the loss function using chain rules

#### Gradient of Loss :

* Type: vector є

=  (

=

#### Compute Jacobian of softmax

For diagonal terms i=j

)

)

For diagonal terms ij

#### Compute Chain Rule

Compute the Chain Rule

є

#### Compute

Z = WX+b

#### Compute

#### Updating W with Gradient Descent

W = W -α

W= W- α

#### Updating b with Gradient Descent

b = b - α

b=b- α

where α is the learning rate.

## Updating the weight Matrix with softmax and Cross-Entropy

Updating the weight Matrix W using the total loss in single-label classification with the softmax+ cross-entropy loss setup. We will do that for the case with:

* M output classes (e.g. Dog, Cat, Rabbit->M=3)
* N input features.
* X є : the input feature vector.
* W є : the weight matrix
* b є : the bias vector.
* Z є : logits
* Prediction outputs
* : true labels (0 or 1 for each class)

### Cross Entropy Loss

### Define Variables and Shapes

* Weight matrix W of shape (N,M): N is number of Neurons and M is the number of inputs to each Neurons:

є

* Input feature vector X of shape (M,1)

є

* Bias vector b of shape (N,1)

є

### Forward Propagation

#### Compute Pre-activation Output Z

Type : vector if one sample or Matrix if batch of samples

Z є where N is number of neurons

Z = WX + b

Where b є

=

=

### Backpropagation

Backpropagation procedure is depicted in this section

#### softmax function

The softmax function is used to convert the logits into probability

The predicted probability fo each class i is given by:

; i = 1,2,…….N

#### Cross Entropy Loss

Let Y be the one-hot encoded true label vector:

Where

* Let be the predicted probability distribution from softmax:

Where is the predicted probability for class i

##### Definition of Cross-Entropy Loss:

The cross entropy for a single input X sample is defined as:

Since Y is one one-hot encoded, only one and all others , so only the term corresponding to the correct class contributes to the loss, if the true class is k; then:

loss as a function of

Since ,then the final expression for the loss function L is :

#### The Jacobian Matrix of softmax

The Jacobian matrix is essential when we compute the gradients of the loss function wit respect to the logits, Z. The Jacobian matrix for the softmax function is a matrix of partial derivative of each predicted class probability with respect to each logit . The Jacobian matrix of softmax function is given by:

For diagonal terms i=j

)

)

For diagonal terms ij

#### Compute Chain Rule

Since ,

=

#### Compute Chain Rule

є

Since one-shot encoding only one component is 1. Let us assume class =2 is the correct class, then when class =1 is:

Assume N =3

= 1 (Since Y is one-hot encoded), then

= 1 (Since Y is one-hot encoded), then

= 1 (Since Y is one-hot encoded), then

=

#### Example of M = 3 Classes

This vector shows the error in predictions:

* 0.2 is the predicted probability for class 1 (which should be 0).
* −0.3 shows an **overestimation correction** for class 2 (since it should be 1 but was predicted as 0.7).
* is the predicted probability for class 3 (which should be 0).

**General Interpretation**

* is a probability error vector.
* This is the gradient of the **cross-entropy loss** with respect to the logits Z:

* It tells us how much the predicted probabilities deviate from the correct labels.

#### Compute

Z = WX+b

#### Compute

=

=

#### Update W (Gradient Descent)

W = W -α

W = W--α

where α is the learning rate

#### Update b (Gradient descent)

b = b - α

b = b - α

where α is the learning rate

## Updating the Weight Matrix W with sigmoid and Cross-Entropy

Updating the weight Matrix W using the total loss in multi-label classification with the sigmoid+ cross-entropy loss setup. We will do that for the case with:

* M output classes (e.g. Dog, Cat, Rabbit->M=3)
* N input features.
* X є : the input feature vector.
* W є : the weight matrix
* b є : the bias vector.
* Z є : logits
* Prediction outputs
* : true labels (0 or 1 for each class)

### Compute Pre-activation Output Z

Type : vector if one sample or Matrix if batch of samples

Z є where N is number of neurons

Z = WX + b

Where b є

=

=

### Backpropagation

#### Binary Cross Entropy Loss

The cross entropy for a single input X sample for all classes is:

+ (

#### Compute

#### Sigmoid Activation Function

Where є

#### Compute Jacobian of sigmoid

For diagonal terms i=j

For terms ij

0

#### Compute

#### Compute

Z = WX+b

#### Compute

Z = WX+b

#### Updated Parameters (Gradient Descent)

W = W -α

W = W -α

b = b - α

b = b - α

where α is the learning rate.

multiple elements of Y could be 1, so multiple rows of W get updated in the positive direction.

### Example softmax with cross entropy

#### Scenario

We have 3 classes: dog, cat, Rabbit

Input feature :

W=

#### Softmax + on-Hot Cross Entropy

1. **Assume**

true label Cat

1. **Compute Z = WX**

Z=

1. **softmax on Z**

1. **Compute Gradient**

=

Only one label (Cat) was 1, so the update pushes down Cat’s probability and adjust others.

### Example sigmoid with cross entropy

#### Scenario

We have 3 classes: dog, cat, Rabbit

Input feature :

W=

#### softmax + on-Hot Cross Entropy

1. **Assume**

true label Dog & Rabbit

1. **Compute Z = WX**

Z=

1. **softmax on Z**
2. **Compute Gradient**

=

Multiple classes (dog & Rabbit) contributed to the loss, so both their rows in W are updated

## Dynamic Edge-Conditional Filters in Convolutional Neural Network On Graphs

### Dynamic Edge-Conditioned Convolution (ECC) Formula

The node-Level ECC operation:

. , where k is the filter index

Where is the filter function such as:

🔹 **θ₁(e) = 0.2 · e**

* **Type:** Linear
* **Effect:** Linearly increases with edge weight.
* **Use case:** Emphasizes stronger edges proportionally.

🔹 **θ₂(e) = 0.5 − 0.1 · e**

* **Type:** Linear (decreasing)
* **Effect:** Linearly decreases as edge value increases.
* **Use case:** Suppresses messages from stronger edges.

🔹 **θ₃(e) = max(0, 0.3 · e − 0.5)**

* **Type:** ReLU-style threshold
* **Effect:** Zero weight when e< ; increases linearly afterward.
* **Use case:** Acts like an attention gate—only passes strong enough edges.

🔹 **θ₄(e) = log(1 + e)**

* **Type:** Logarithmic
* **Effect:** Grows sub-linearly with edge value.
* **Use case:** Smoothly increases with edge importance; reduces impact of large edge weights.

🔹 **θ₅(e) = 1 / (1 + e)**

* **Type:** Inverse
* **Effect:** Decreases with increasing edge value.
* **Use case:** Strong edges are down-weighted; useful for emphasizing small weights (e.g., distances).

🔹 **θ₆(e) = 1 if e > 2, else 0**

* **Type:** Step function (indicator)
* **Effect:** Hard threshold at e=2; either fully pass or block.
* **Use case:** Selects only edges above a certain strength; binary gating.

🔹 **θ₇(e) = e · exp(−0.5(e − 2)²)**

* **Type:** Gaussian-bell shaped around e=2e = 2e=2
* **Effect:** Peaks when e=2; falls off symmetrically.
* **Use case:** Emphasizes edge values close to 2; filters out all others.

🔹 **θ₈(e) = 0.5 · max(0, 0.7e − 1) + 0.2**

* **Type:** Leaky-activation-style threshold
* **Effect:** Outputs constant 0.2 when e < ; grows slowly afterward.
* **Use case:** Always passes a base amount; increases for strong edges.

### Dynamic Filters Procedure

#### Grey Image Use dynamic filters to compute node-level outputs

**Step 1: per pixel computation (each pixel = node)**

. , where k is the filter index

* **Color Image dynamic filters to compute node-level outputs**

**Option 1: Sum of Absolute Differences (L1 norm)**

+

**Option 2: Euclidean Distance (L2 norm)**

Gives smoother, magnitude-based similarity.

, ,

**Step 2: global aggregation**

* **Grey Image**

Pool over all nodes to get the graph/image-level feature vector where K is the number of filters:

….. }

* **Option 1 : SUM**

* **Option 2 : MEAN**

* **Option 3 : MAX**

1..M, 1..K where is the largest

**Step 3: Final Global Feature Vector**

x = =  **OR** x = =  **OR** x = =

**Step 3: Classification Layer**

z=X= W = b=

**Step 4 : Training Goal**

Train the model to classify the input image at one of C classes

**Step 4 : Forward Pass :**

A global feature vector X=from the edge-conditioned convolution and global aggregation.

You have learnable weights W  **and bias b**

**Compute logits Z:**  is **unnormalized final scores of your model**. You apply softmax to it to get a probability distribution over your classes:

z= +

**Step 5: Compute Probability Distribution over Classes**

To convert logits to probabilities

predicted probability distribution

**Step 6: Compute Cross Entropy**

Let assume the true class is then the loss is :

= ) =

Only the prediction probability for the correct class contributes to the loss

**Step 7: Backpropagation (Gradient Descent)**

=

= -y)

=

=

**Step 8: Update Parameters**

W = W -α

W = W -α

b = b - α

b = b - α

#### Color Image

**Step 1: per pixel computation (each pixel = node)**

**Option 1: Sum of Absolute Differences (L1 norm)**

+

**Option 2: Euclidean Distance (L2 norm)**

Gives smoother, magnitude-based similarity.

**Step 2: global aggregation**

Pool over all nodes to get the graph/image-level feature vector where K is the number of filters:

….. }

….. }

….. }

* **Option 1 : SUM**
* **Option 2 : MEAN**
* **Option 3 : MAX**

1..M, 1..K where is the largest

1..M, 1..K where is the largest

1..M, 1..K where is the largest

**Step 3: Final Global Feature Vector**

x = =  **OR**  =  **OR**

**Step 3: Classification Layer**

z=X= W = b=

**Step 4 : Training Goal**

Train the model to classify the input image at one of C classes

**Step 4 : Forward Pass :**

A global feature vector X=from the edge-conditioned convolution and global aggregation.

You have learnable weights W  **and bias b**

**Compute logits Z:**  is **unnormalized final scores of your model**. You apply softmax to it to get a probability distribution over your classes:

z= +

**Step 5: Compute Probability Distribution over Classes**

To convert logits to probabilities

predicted probability distribution

**Step 6: Compute Cross Entropy**

Let assume the true class is then the loss is :

= ) =

Only the prediction probability for the correct class contributes to the loss

**Step 7: Backpropagation (Gradient Descent)**

=

= -y)

=

=

**Step 8: Update Parameters**

W = W -α

W = W -α

b = b - α

b = b - α

### Numerical example for greyscale image

#### Assumptions

An Image the pixels from 1-16 and the value of pixels from 1-8

Image =

#### Procedure Steps

**Step 1: Calculate Edge Features**

Assume :

* Node Feature  are scalar intensity values.
* Edge features direction vector from

We are computing the output of Node 6 in grayscale

Neighbors of node 6 are:

* Node 2:
* Node 5:
* Node 7:
* Node 10:

We have to 2 filters ( ) that we will use to generate features for all nodes from 1-16, our example will cover Node 6

|  |  |
| --- | --- |
| Edge | Edge feature  = |
| (6,2) |  |
| (6,5) |  |
| (6,7) |  |
| (6,10) |  |

**Step 2: Applying the filters (**

We apply the 2 dynamic filters each filter depends on the edge feature and node value

* Filter 1 (

.

Then Compute :

* Filter 2 (

(1+

Then Compute :

* Filter 1 (calculation

**For node 2:**

= 0.2x2 = 0.4

0.4x2 = 0.8

**For node 5:**

= 0.2x2 = 0.4

0.4x2 = 0.8

**For node 7:**

= 0.2x1 = 0.5

0.2x5 = 1.0

**For node 10:**

= 0.2x2 = 0.4

0.4x6 = 2.4

Summing each these contributions

* Filter 2 (calculation

**For node 2:**

= log (1+2) = log (3) = 1.0986

1.0986x2 = 2.1972

**For node 5:**

= log (1+2) = log (3) = 1.0986

1.0986x2 = 2.1972

**For node 7:**

= log (1+1) = 0.6931

0.6931x5 = 3.4655

**For node 10:**

= log (1+2) = 1.0986

1.0986x6 = 6.4655

Summing each these contributes

**Step 3: global Aggregation**

,….,

….. }

**Step 4: Classification Layer**

Now that we have 2 feature values from 2 filters, we need to map them to the 3 classes dog (1,0,0), cat (0,1,0), and man (0,0,1). Let us assume the following weights (W) for the final classification.

W=

And Bias vector

b =

we need to compute the logits for each class. If we aasume that we apply only for node 6, then:

= W.

= .

=

**Step 4: Apply softmax for node 6 as example**

* if we apply global aggregation
* for example if we apply only for node 6 were C = 1,2,3 in our example

= 37.75 + 0.640+88.46 = 121.85

==

= softmaz () =

**Final Classification for node 6:**

* + Dog:26.9%
  + Cat:0.5%
  + Man 72.26% predicted class Man

# Acronyms

CSC Communication Service Customer

NWDAF 5G Network Data Analytical Function

AF Application Function

NF Network Function

CSMF Communication Service Management Function

CSP Communication Service Provider

DN Data Network

MNO Mobile Network Operator

NOP Network Operator

NSaaS Network Slice as a Service

NSaasC Network Slice as a Service Customer

NSaaSP Network Slice as a Service Provider

NSMF Network Slice Management Function

NSC Network Slice Customer

NSSMF Network Slice Subnet Management Function

NSP Network Slice Provider

SLA Service Level Agreement

SLS Service Level Specification

TN Transport Network