Deep Learning Principles & Applications

Chapter 2 – Linear Classifiers

Sudarsan N.S. Acharya (sudarsan.acharya@manipal.edu)

Classification in Practice

Classification in Practice

Classifying a sample into one of the known categories (or classes) is a common challenge across different domains:

Classification in Practice

Classifying a sample into one of the known categories (or classes) is a common challenge across different domains:

Computer
Vision

Classification in Practice

Classifying a sample into one of the known categories (or classes) is a common challenge across different domains:

Computer Vision

Classification in Practice

Classifying a sample into one of the known categories (or classes) is a common challenge across different domains:

Computer Vision

is this sample a $\underbrace{\text{lion/tiger/leopard}}_{\text{known 3 classes}}$?

Classification in Practice

Classifying a sample into one of the known categories (or classes) is a common challenge across different domains:

Computer Vision

is this sample a $\underbrace{\text{lion/tiger/leopard}}_{\text{known 3 classes}}$?

Recall that this color image is internally represented as a $337 \times 600 \times 3$-tensor of integer values ranging from 0 to 255.

Classification in Practice – continued

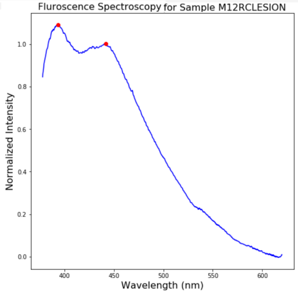
Classification in Practice – continued

Medical Signal
Processing

Classification in Practice – continued

Medical Signal Processing

	0	1	...	1017	1018
M12RCLESION	0.845890	0.861290	...	0.000037	0.001354

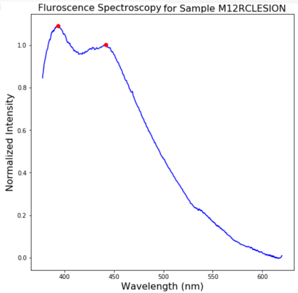


Classification in Practice – continued

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

Medical Signal Processing

	0	1	...	1017	1018
M12RCLESION	0.845890	0.861290	...	0.000037	0.001354



What kind of an oral tumor does this patient have: benign / premalignant / malignant?

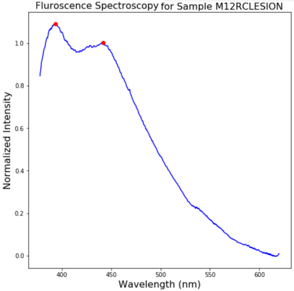
known 3 classes

Classification in Practice – continued

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

Medical Signal Processing

	0	1	...	1017	1018
M12RCLESION	0.845890	0.861290	...	0.000037	0.001354



What kind of an
oral tumor does this patient have:
benign/premalignant/malignant?

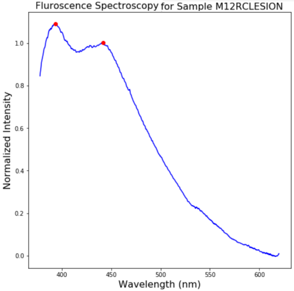
known 3 classes

Language Application

Classification in Practice – continued

Medical Signal Processing

	0	1	...	1017	1018
M12RCLESION	0.845890	0.861290	...	0.000037	0.001354



What kind of an
oral tumor does this patient have:
benign / premalignant / malignant?

known 3 classes

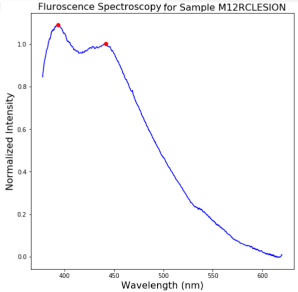
Language Application

The movie was goat

Classification in Practice – continued

Medical Signal Processing

	0	1	...	1017	1018
M12RCLESION	0.845890	0.861290	...	0.000037	0.001354



What kind of an oral tumor does this patient have: benign/premalignant/malignant?

known 3 classes

Language Application

The movie was goat

Is this movie review positive/negative?

known 2 classes

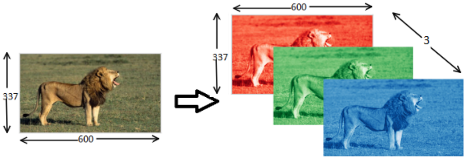
Linear Classifier: Basic Idea

Linear Classifier: Basic Idea

Quantify the process of training-to-classify a sample into lion/tiger/leopard:

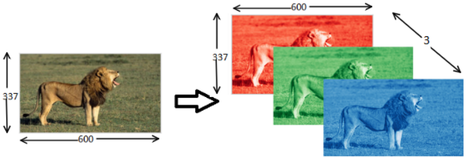
Linear Classifier: Basic Idea

Quantify the process of training-to-classify a sample into lion/tiger/leopard:



Linear Classifier: Basic Idea

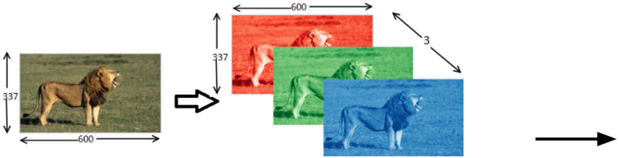
Quantify the process of training-to-classify a sample into lion/tiger/leopard:



a training image that can be seen as
a vector $\mathbf{x}$ with
$337 \times 600 \times 3 = 606600$ numbers

Linear Classifier: Basic Idea

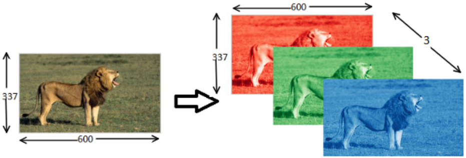
Quantify the process of training-to-classify a sample into lion/tiger/leopard:



a training image that can be seen as a vector $\mathbf{x}$ with $337 \times 600 \times 3 = 606600$ numbers

Linear Classifier: Basic Idea

Quantify the process of training-to-classify a sample into lion/tiger/leopard:

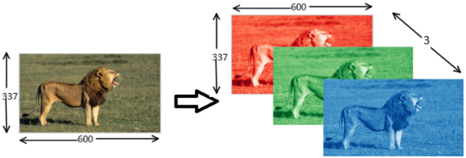


a training image that can be seen as a vector $\mathbf{x}$ with $337 \times 600 \times 3 = 606600$ numbers

Calculate 3 class scores as

Linear Classifier: Basic Idea

Quantify the process of training-to-classify a sample into lion/tiger/leopard:



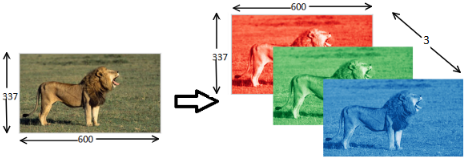
a training image that can be seen as a vector $\mathbf{x}$ with $337 \times 600 \times 3 = 606600$ numbers

Calculate 3 class scores as

$$\underbrace{\mathbf{W}}_{3\times 606600\text{-matrix}}$$

Linear Classifier: Basic Idea

Quantify the process of training-to-classify a sample into lion/tiger/leopard:



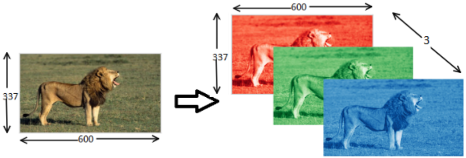
a training image that can be seen as a vector $\mathbf{x}$ with $337 \times 600 \times 3 = 606600$ numbers

Calculate 3 class scores as

$$\underbrace{\mathbf{W}}_{3\times 606600\text{-matrix}} \quad \underbrace{\mathbf{x}}_{606600\text{-vector}}$$

Linear Classifier: Basic Idea

Quantify the process of training-to-classify a sample into lion/tiger/leopard:



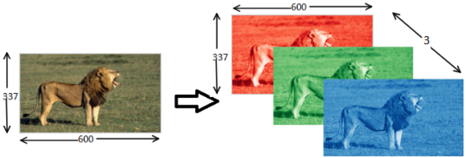
a training image that can be seen as a vector $\mathbf{x}$ with $337 \times 600 \times 3 = 606600$ numbers

Calculate 3 class scores as

$$\underbrace{\mathbf{W}}_{3\times 606600\text{-matrix}} \underbrace{\mathbf{x}}_{606600\text{-vector}} + \underbrace{\mathbf{b}}_{3\text{-vector}}$$

Linear Classifier: Basic Idea

Quantify the process of training-to-classify a sample into lion/tiger/leopard:



a training image that can be seen as a vector $\mathbf{x}$ with
$337 \times 600 \times 3 = 606600$ numbers

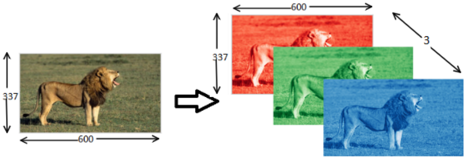
Calculate 3 class scores as

$$\underbrace{\mathbf{W}}_{3\times 606600\text{-matrix}} \; \underbrace{\mathbf{x}}_{606600\text{-vector}} + \underbrace{\mathbf{b}}_{3\text{-vector}}$$

that can be used to assess how good the choices of $\mathbf{W}$ and $\mathbf{b}$ are.

Linear Classifier: Basic Idea

Quantify the process of training-to-classify a sample into lion/tiger/leopard:



a training image that can be seen as a vector $\mathbf{x}$ with $337 \times 600 \times 3 = 606600$ numbers

Calculate 3 class scores as

$$\underbrace{\mathbf{W}}_{3\times 606600\text{-matrix}} \; \underbrace{\mathbf{x}}_{606600\text{-vector}} + \underbrace{\mathbf{b}}_{3\text{-vector}}$$

that can be used to assess how good the choices of $\mathbf{W}$ and $\mathbf{b}$ are.

What are $\mathbf{W}$ and $\mathbf{b}$ (the parameters), and how do we know what they are?

Weights, Bias, & Raw Scores

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} \quad & \quad & \quad \\ & & \\ & & \end{bmatrix}}_{\mathbf{W}}$$

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ & & & \\ & & & \end{bmatrix}}_{\mathbf{W}}$$

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ & & & \end{bmatrix}}_{\mathbf{W}}$$

Weights, Bias, & Raw Scores

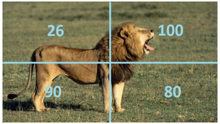
Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}}$$

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}}$$



Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as } 4\text{-vector } \mathbf{x}}$$

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as } 4\text{-vector } \mathbf{x}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}}$$

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as 4-vector } \mathbf{x}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} 32.6 \\ 287.8 \\ 25 \end{bmatrix}}_{\mathbf{z}}$$

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as } 4\text{-vector } \mathbf{x}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} 32.6 \\ 287.8 \\ 25 \end{bmatrix}}_{\mathbf{z}} \begin{matrix} \text{lion raw score} \\ \text{tiger raw score} \\ \text{leopard raw score} \end{matrix}$$

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as } 4\text{-vector } \mathbf{x}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} 32.6 \\ 287.8 \\ 25 \end{bmatrix}}_{\mathbf{z}} \begin{array}{c} \text{lion raw score} \\ \text{tiger raw score} \\ \text{leopard raw score} \end{array}$$

Using the language of linear algebra, raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$.

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as 4-vector } \mathbf{x}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} 32.6 \\ 287.8 \\ 25 \end{bmatrix}}_{\mathbf{z}} \begin{matrix} \text{lion raw score} \\ \text{tiger raw score} \\ \text{leopard raw score} \end{matrix}$$

Using the language of linear algebra, raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$. The current set of weights and bias values lead to a maximum raw score (287.8) for the (*incorrect*) tiger class ☹.

Weights, Bias, & Raw Scores

Using the training samples, devise a computational approach for calculating the *optimal* weights matrix $\mathbf{W}$ and the bias vector $\mathbf{b}$:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as 4-vector } \mathbf{x}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} 32.6 \\ 287.8 \\ 25 \end{bmatrix}}_{\mathbf{z}} \begin{array}{c} \text{lion raw score} \\ \text{tiger raw score} \\ \text{leopard raw score} \end{array}$$

Using the language of linear algebra, raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$. The current set of weights and bias values lead to a maximum raw score (287.8) for the (*incorrect*) tiger class ☹. Can we quantify the *unhappiness*?

Loss Function – Intuition

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score   

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6

Happy with $\mathbf{W}$ & $\mathbf{b}$?

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6
Happy with $\mathbf{W}$ & $\mathbf{b}$?			

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6
Happy with $\mathbf{W}$ & $\mathbf{b}$?			

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6
Happy with $\mathbf{W}$ & $\mathbf{b}$?			

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6
Happy with $\mathbf{W}$ & $\mathbf{b}$?			

Quantifying loss for each sample:

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6
Happy with $\mathbf{W}$ & $\mathbf{b}$?			

Quantifying loss for each sample: *incorrect class scores greater than correct class scores contribute to the loss.*

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6
Happy with $\mathbf{W}$ & $\mathbf{b}$?			

Quantifying loss for each sample: *incorrect class scores greater than correct class scores contribute to the loss.*

Loss for Sample-1

$$L_1 = \begin{cases} \max(0, 6.4 - 5.6) \\ \quad + \\ \max(0, -4.6 - 5.6) \end{cases}$$

$$= 0.8$$

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6
Happy with $\mathbf{W}$ & $\mathbf{b}$?			

Quantifying loss for each sample: *incorrect class scores greater than correct class scores contribute to the loss.*

Loss for Sample-2

$$L_2 = \begin{cases} \max(0, -1.8 - 10.2) \\ + \\ \max(0, 3.5 - 10.2) \end{cases}$$

$$= 0$$

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6
Happy with $\mathbf{W}$ & $\mathbf{b}$?			

Quantifying loss for each sample: *incorrect class scores greater than correct class scores contribute to the loss.*

Loss for Sample-3

$$L_3 = \begin{cases} \max(0, 2.0 - (-8.6)) \\ + \\ \max(0, 5.4 - (-8.6)) \end{cases}$$
$$= 24.6$$

Loss Function – Intuition

Given that we know the true output class for a set of training samples, we can quantify the unhappiness for a particular set of weights $\mathbf{W}$ and $\mathbf{b}$ values using the raw scores for 3 training samples as follows:

Raw score			
Lion	5.6	-1.8	2.0
Tiger	6.4	10.2	5.4
Leopard	-4.6	3.5	-8.6
Happy with $\mathbf{W}$ & $\mathbf{b}$?			

Quantifying loss for each sample: *incorrect class scores greater than correct class scores contribute to the loss.*

Average training loss

$$\frac{0.8 + 0 + 24.6}{3} = 8.5$$

Hinge Loss Function

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.

Hinge Loss Function

- Suppose there are n training samples $\left(\underset{\substack{\uparrow\\\text{sample vector}}}{\mathbf{x}^{(i)}}, y^{(i)}\right)$.

Hinge Loss Function

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.

correct class/label

Hinge Loss Function

MANIPAL ACADEMY of HIGHER EDUCATION (Institution of Eminence Deemed to be University)

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.

Hinge Loss Function

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

$$L_i = \sum_{j \neq y^{(i)}} \max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

Hinge Loss Function

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

$$L_i = \sum_{j \neq y^{(i)}} \max\left(0, \underset{\substack{\uparrow \\ \text{incorrect class raw score}}}{z_j^{(i)}} - z_{y^{(i)}}^{(i)} + 1\right)$$

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

$$L_i = \sum_{j \neq y^{(i)}} \max\left(0, z_j^{(i)} - \underbrace{z_{y^{(i)}}^{(i)}}_{\text{correct class raw score}} + 1\right)$$

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

$$L_i = \sum_{j \neq y^{(i)}} \max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + \underset{\text{offset}}{\underset{\uparrow}{1}}\right)$$

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

$$L_i = \sum_{j \neq y^{(i)}} \max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

$$= \sum_{j \neq y^{(i)}} \max\left(0, \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_j - \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_{y^{(i)}} + 1\right)$$

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

$$L_i = \sum_{j \neq y^{(i)}} \max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

$$= \sum_{j \neq y^{(i)}} \max\left(0, \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_j - \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_{y^{(i)}} + 1\right)$$

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

$$L_i = \sum_{j \neq y^{(i)}} \max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

$$= \sum_{j \neq y^{(i)}} \max\left(0, \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_j - \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_{y^{(i)}} + 1\right)$$

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

$$L_i = \sum_{j \neq y^{(i)}} \max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

$$= \sum_{j \neq y^{(i)}} \max\left(0, \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_j - \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_{y^{(i)}} + 1\right)$$

- The average training data loss is $\frac{1}{n}\sum_{i=1}^{n} L_i$,

Hinge Loss Function

- Suppose there are n training samples $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$.
- Choose an initial weights matrix $\mathbf{W}$ and bias vector $\mathbf{b}$.
- The ith sample's raw score vector $\mathbf{z}^{(i)} = \mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}$.
- The multiclass SVM hinge loss function for the ith sample is

$$L_i = \sum_{j \neq y^{(i)}} \max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

$$= \sum_{j \neq y^{(i)}} \max\left(0, \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_j - \left[\mathbf{W}\mathbf{x}^{(i)} + \mathbf{b}\right]_{y^{(i)}} + 1\right)$$

- The average training data loss is $\frac{1}{n}\sum_{i=1}^{n} L_i$, which is a function of the weights and bias values.

Visualizing Loss Functions

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

Perceptron Loss

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

Perceptron Loss **Hinge Loss**

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

Perceptron Loss **Hinge Loss** **Squared Hinge Loss**

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

Perceptron Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)}\right)$$

Hinge Loss

Squared Hinge Loss

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

Perceptron Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)}\right)$$

Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

Squared Hinge Loss

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

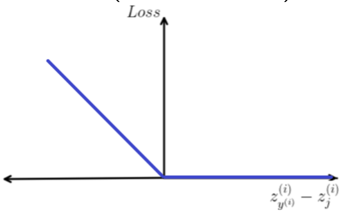
Perceptron Loss	Hinge Loss	Squared Hinge Loss
$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)}\right)$	$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$	$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)^2$

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

Perceptron Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)}\right)$$



Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

Squared Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)^2$$

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

Perceptron Loss

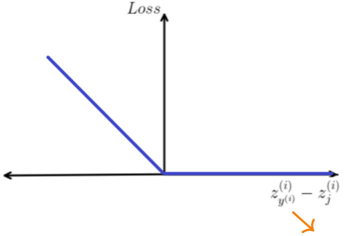
$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)}\right)$$

Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$

Squared Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)^2$$



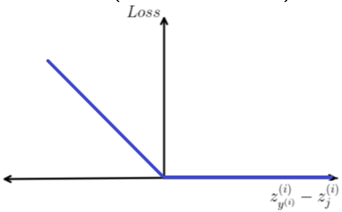
difference between correct and incorrect class raw scores

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

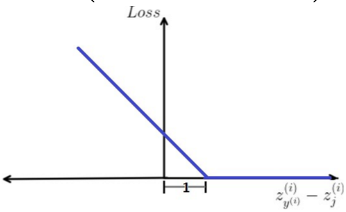
Perceptron Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)}\right)$$



Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$



Squared Hinge Loss

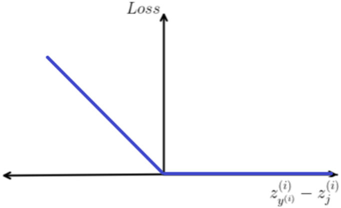
$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)^2$$

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

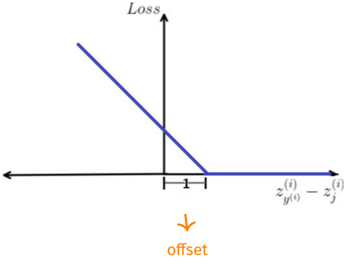
Perceptron Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)}\right)$$



Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$



Squared Hinge Loss

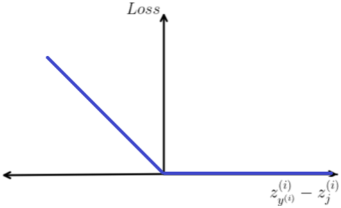
$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)^2$$

Visualizing Loss Functions

Visualizing different loss functions considering contribution from one incorrect class:

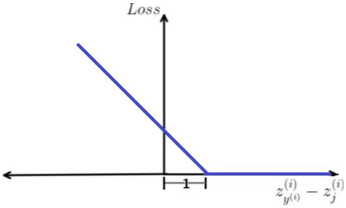
Perceptron Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)}\right)$$



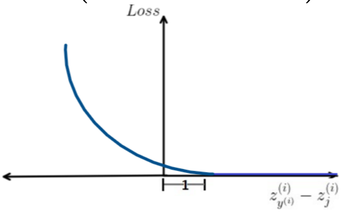
Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)$$



Squared Hinge Loss

$$\max\left(0, z_j^{(i)} - z_{y^{(i)}}^{(i)} + 1\right)^2$$



Softmax Function

Softmax Function

It is possible to turn the raw scores vector into a a vector of probabilities:

Softmax Function

It is possible to turn the raw scores vector into a a vector of probabilities:

Raw score	
Lion score	5.6
Tiger score	6.4
Leopard score	-4.6

Softmax Function

It is possible to turn the raw scores vector into a a vector of probabilities:

Raw score	
Lion score	5.6
Tiger score	6.4
Leopard score	-4.6

Raise to power of e →

Softmax Function

It is possible to turn the raw scores vector into a a vector of probabilities:

Raw score			Exponentiated raw score
Lion score	5.6	Raise to power of e →	$e^{5.6}$
Tiger score	6.4		$e^{6.4}$
Leopard score	-4.6		$e^{-4.6}$

Softmax Function

It is possible to turn the raw scores vector into a a vector of probabilities:

Raw score			Exponentiated raw score	
Lion score	5.6	Raise to power of e $\rightarrow$	$e^{5.6}$	normalize $\rightarrow$
Tiger score	6.4		$e^{6.4}$	
Leopard score	-4.6		$e^{-4.6}$	

Softmax Function

It is possible to turn the raw scores vector into a a vector of probabilities:

Raw score		Raise to power of e →	**Exponentiated raw score**	normalize →	**Probabilities**
Lion score	5.6		$e^{5.6}$		$\frac{e^{5.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0.31$
Tiger score	6.4		$e^{6.4}$		$\frac{e^{6.4}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0.69$
Leopard score	-4.6		$e^{-4.6}$		$\frac{e^{-4.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0$

Softmax Function

It is possible to turn the raw scores vector into a a vector of probabilities:

Raw score			Exponentiated raw score		Probabilities
Lion score	5.6		$e^{5.6}$		$\frac{e^{5.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0.31$
		Raise to power of e $\rightarrow$		normalize $\rightarrow$	
Tiger score	6.4		$e^{6.4}$		$\frac{e^{6.4}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0.69$
Leopard score	-4.6		$e^{-4.6}$		$\frac{e^{-4.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0$

Formally, the softmax function takes a vector as input,

Softmax Function

It is possible to turn the raw scores vector into a a vector of probabilities:

Raw score		Raise to power of e →	Exponentiated raw score	normalize →	Probabilities
Lion score	5.6		$e^{5.6}$		$\frac{e^{5.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0.31$
Tiger score	6.4		$e^{6.4}$		$\frac{e^{6.4}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0.69$
Leopard score	-4.6		$e^{-4.6}$		$\frac{e^{-4.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0$

Formally, the softmax function takes a vector as input, and outputs a vector (of the same size) of probabilities through exponentiation and normalization.

Softmax Function

It is possible to turn the raw scores vector into a a vector of probabilities:

Raw score			Exponentiated raw score		Probabilities
Lion score	5.6		$e^{5.6}$		$\frac{e^{5.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0.31$
Tiger score	6.4	Raise to power of e $\rightarrow$	$e^{6.4}$	normalize $\rightarrow$	$\frac{e^{6.4}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0.69$
Leopard score	-4.6		$e^{-4.6}$		$\frac{e^{-4.6}}{e^{5.6}+e^{6.4}+e^{-4.6}} \approx 0$

Formally, the softmax function takes a vector as input, and outputs a vector (of the same size) of probabilities through exponentiation and normalization. The lion probability is not 1.0 rather $0.39 \Rightarrow$

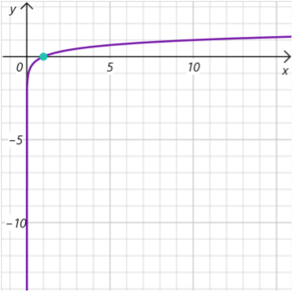
Softmax Loss Function

Softmax Loss Function

- The natural logarithm $\log(x)$ is a very useful function:

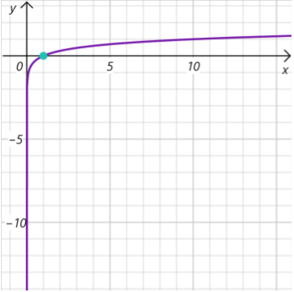
Softmax Loss Function

- The natural logarithm $\log(x)$ is a very useful function:



Softmax Loss Function

- The natural logarithm $\log(x)$ is a very useful function:



- Note that $\begin{cases} \log(1) = 0, \\ \log(x) \to -\infty \text{ as } x \to 0. \end{cases}$

Softmax Loss Function – continued

Softmax Loss Function – continued

- Suppose a training sample has raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ and belongs to correct class y.

Softmax Loss Function – continued

- Suppose a training sample has raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ and belongs to correct class y.
- softmax$(\mathbf{z})$ gives the class probabilities vector.

Softmax Loss Function – continued

- Suppose a training sample has raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ and belongs to correct class y.
- $\text{softmax}(\mathbf{z})$ gives the class probabilities vector.
- $[\text{softmax}(\mathbf{z})]_y$ is the correct class's probability;

Softmax Loss Function – continued

- Suppose a training sample has raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ and belongs to correct class y.
- softmax$(\mathbf{z})$ gives the class probabilities vector.
- $[\text{softmax}(\mathbf{z})]_y$ is the correct class's probability; closer this probability is to 1, we are ☺;

Softmax Loss Function – continued

- Suppose a training sample has raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ and belongs to correct class y.
- $\text{softmax}(\mathbf{z})$ gives the class probabilities vector.
- $[\text{softmax}(\mathbf{z})]_y$ is the correct class's probability; closer this probability is to 1, we are ☺; closer it is to 0, we are ☹ with $\mathbf{W}$ and $\mathbf{b}$.

Softmax Loss Function – continued

- Suppose a training sample has raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ and belongs to correct class y.
- $\text{softmax}(\mathbf{z})$ gives the class probabilities vector.
- $[\text{softmax}(\mathbf{z})]_y$ is the correct class's probability; closer this probability is to 1, we are ☺; closer it is to 0, we are ☹ with $\mathbf{W}$ and $\mathbf{b}$.
- The softmax loss function for this sample is defined as $-\log\left([\text{softmax}(\mathbf{z})]_y\right)$.

Softmax Loss Function – continued

- Suppose a training sample has raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ and belongs to correct class y.
- softmax$(\mathbf{z})$ gives the class probabilities vector.
- $[\text{softmax}(\mathbf{z})]_y$ is the correct class's probability; closer this probability is to 1, we are ☺; closer it is to 0, we are ☹ with $\mathbf{W}$ and $\mathbf{b}$.
- The softmax loss function for this sample is defined as $-\log\left([\text{softmax}(\mathbf{z})]_y\right)$.
- In plain English, it is the negative of the logarithm of the correct class's probability.

Softmax Loss Function – continued

- Suppose a training sample has raw scores vector $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ and belongs to correct class y.
- $\text{softmax}(\mathbf{z})$ gives the class probabilities vector.
- $[\text{softmax}(\mathbf{z})]_y$ is the correct class's probability; closer this probability is to 1, we are ☺; closer it is to 0, we are ☹ with $\mathbf{W}$ and $\mathbf{b}$.
- The softmax loss function for this sample is defined as $-\log\left([\text{softmax}(\mathbf{z})]_y\right)$.
- In plain English, it is the negative of the logarithm of the correct class's probability.
- Note that $\begin{cases} [\text{softmax}(\mathbf{z})]_y = 1 & \Rightarrow ☺ \Rightarrow \text{loss} = -\log(1) = 0, \\ [\text{softmax}(\mathbf{z})]_y = 0 & \Rightarrow ☹ \Rightarrow \text{loss} = -\log(0) \to \infty. \end{cases}$

Optimization Setup

Optimization Setup

Given training samples, the goal is to find optimal values for the weights and biases that minimize the average training loss.

Optimization Setup

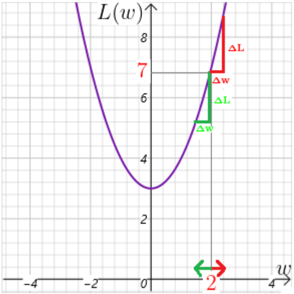
Given training samples, the goal is to find optimal values for the weights and biases that minimize the average training loss.

Consider $L(w) = w^2 + 3$:

Optimization Setup

Given training samples, the goal is to find optimal values for the weights and biases that minimize the average training loss.

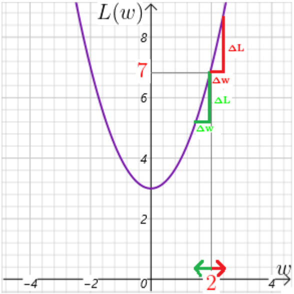
Consider $L(w) = w^2 + 3$:



Optimization Setup

Given training samples, the goal is to find optimal values for the weights and biases that minimize the average training loss.

Consider $L(w) = w^2 + 3$:

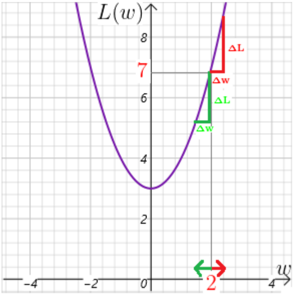


- How can we tweak the input w from it's current value of 2 so that the output L <u>decreases</u> from its current value of 7?

Optimization Setup

Given training samples, the goal is to find optimal values for the weights and biases that minimize the average training loss.

Consider $L(w) = w^2 + 3$:

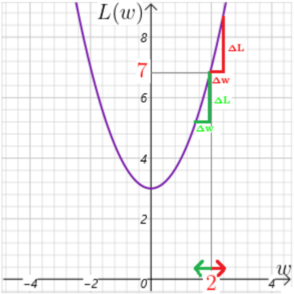


- How can we tweak the input w from it's current value of 2 so that the output L decreases from its current value of 7?
- w can be increased (move right) or decreased (move left) from the current value 2.

Optimization Setup

Given training samples, the goal is to find optimal values for the weights and biases that minimize the average training loss.

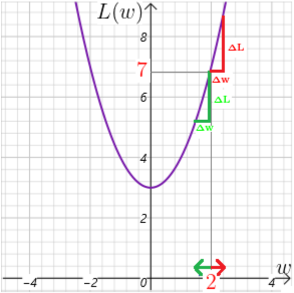
Consider $L(w) = w^2 + 3$:



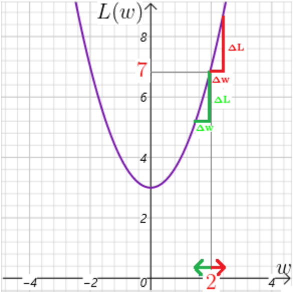
- How can we tweak the input w from it's current value of 2 so that the output L decreases from its current value of 7?
- w can be increased (move right) or decreased (move left) from the current value 2.
- Can we quantify the sensitivity of the output L w.r.t. small changes in the input w?

Sensitivity

Sensitivity

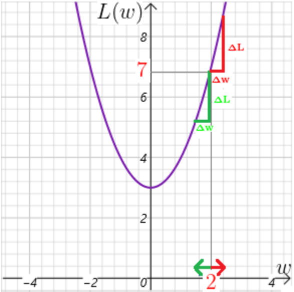


Sensitivity



The sensitivity of the output L w.r.t. a small change in the input w is

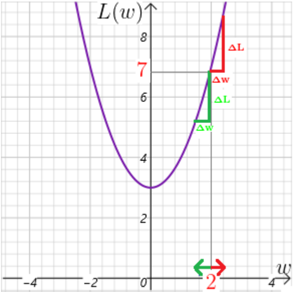
Sensitivity



The sensitivity of the output L w.r.t. a small change in the input w is

$$\frac{\text{change in output}}{\text{change in input}}$$

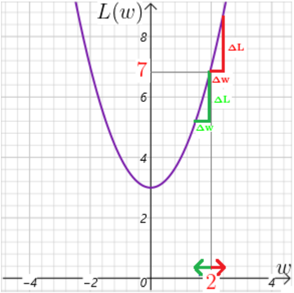
Sensitivity



The sensitivity of the output L w.r.t. a small change in the input w is

$$\frac{\text{change in output}}{\text{change in input}} : \left\{ \vphantom{\begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \end{array}} \right.$$

Sensitivity

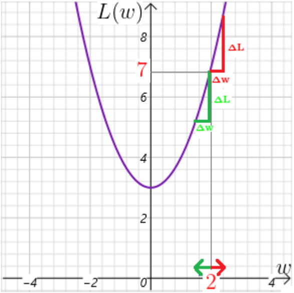


The sensitivity of the output L w.r.t. a small change in the input w is

$$\frac{\text{change in output}}{\text{change in input}} : \begin{cases} \text{moving right} & = \\ & \\ & \end{cases}$$

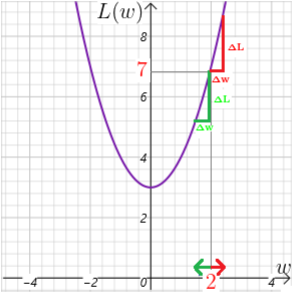
Sensitivity

The sensitivity of the output L w.r.t. a small change in the input w is



$$\frac{\text{change in output}}{\text{change in input}} : \begin{cases} \textcolor{red}{\text{moving right}} & = \dfrac{\underbrace{\textcolor{red}{\Delta L}}_{+ve}}{\underbrace{\textcolor{red}{\Delta w}}_{+ve}} \\ & \end{cases}$$

Sensitivity

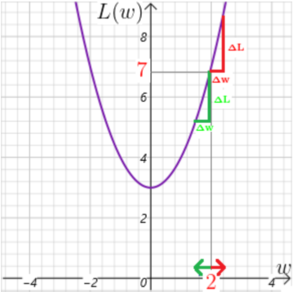


The sensitivity of the output L w.r.t. a small change in the input w is

$$\frac{\text{change in output}}{\text{change in input}} : \begin{cases} \text{moving right} & = \dfrac{\underbrace{\Delta L}_{+ve}}{\underbrace{\Delta w}_{+ve}} = +ve \end{cases}$$

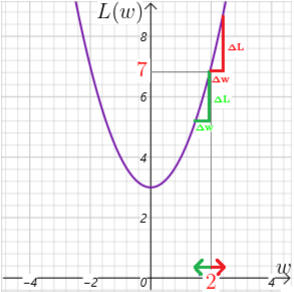
Sensitivity

The sensitivity of the output L w.r.t. a small change in the input w is



$$\frac{\text{change in output}}{\text{change in input}} : \begin{cases} \text{moving right} & = \dfrac{\underbrace{\Delta L}_{+ve}}{\underbrace{\Delta w}_{+ve}} = +ve \\ \text{moving left} & = \end{cases}$$

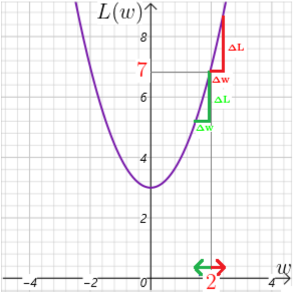
Sensitivity



The sensitivity of the output L w.r.t. a small change in the input w is

$$\frac{\text{change in output}}{\text{change in input}} : \begin{cases} \text{moving right} & = \dfrac{\underbrace{\Delta L}_{+ve}}{\underbrace{\Delta w}_{+ve}} = +ve \\ \text{moving left} & = \dfrac{\underbrace{\Delta L}_{-ve}}{\underbrace{\Delta w}_{-ve}} \end{cases}$$

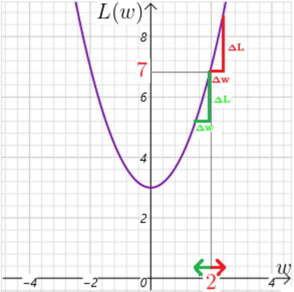
Sensitivity



The sensitivity of the output L w.r.t. a small change in the input w is

$$\frac{\text{change in output}}{\text{change in input}} : \begin{cases} \text{moving right} & = \frac{\underbrace{\Delta L}_{+ve}}{\underbrace{\Delta w}_{+ve}} = +ve \\ \text{moving left} & = \frac{\underbrace{\Delta L}_{-ve}}{\underbrace{\Delta w}_{-ve}} = +ve \end{cases}$$

Sensitivity



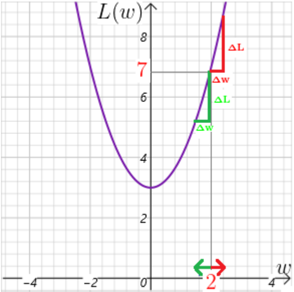
The sensitivity of the output L w.r.t. a small change in the input w is

$$\frac{\text{change in output}}{\text{change in input}} : \left\{ \begin{array}{ll} \text{moving right} & = \dfrac{\overbrace{\Delta L}^{+ve}}{\underbrace{\Delta w}_{+ve}} = +ve \\ \text{moving left} & = \dfrac{\overbrace{\Delta L}^{-ve}}{\underbrace{\Delta w}_{-ve}} = +ve \end{array} \right\} = +ve.$$

Sensitivity

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)
INSPIRED BY LIFE

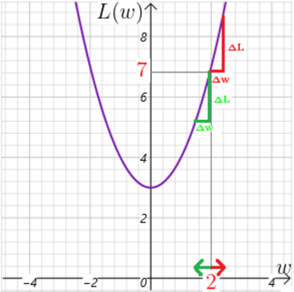
The sensitivity of the output L w.r.t. a small change in the input w is



$$\frac{\text{change in output}}{\text{change in input}} : \left\{ \begin{array}{ll} \text{moving right} & = \dfrac{\overbrace{\Delta L}^{+ve}}{\underbrace{\Delta w}_{+ve}} = +ve \\ \text{moving left} & = \dfrac{\overbrace{\Delta L}^{-ve}}{\underbrace{\Delta w}_{-ve}} = +ve \end{array} \right\} = +ve.$$

$\boxed{+ve \text{ sensitivity}}$ w increases $\Rightarrow$ L increases & w decreases $\Rightarrow$ L decreases

Sensitivity



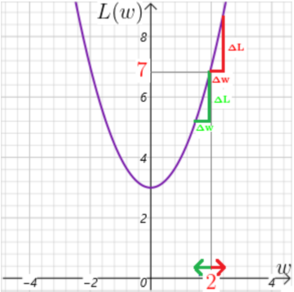
The sensitivity of the output L w.r.t. a small change in the input w is

$$\frac{\text{change in output}}{\text{change in input}} : \left\{ \begin{array}{ll} \text{moving right} & = \dfrac{\overbrace{\Delta L}^{+ve}}{\underbrace{\Delta w}_{+ve}} = +ve \\ \text{moving left} & = \dfrac{\overbrace{\Delta L}^{-ve}}{\underbrace{\Delta w}_{-ve}} = +ve \end{array} \right\} = +ve.$$

$+ve$ sensitivity	w increases $\Rightarrow$ L increases & w decreases $\Rightarrow$ L decreases
$-ve$ sensitivity	w increases $\Rightarrow$ L decreases & w decreases $\Rightarrow$ L increases

Sensitivity

The sensitivity of the output L w.r.t. a small change in the input w is



$$\frac{\text{change in output}}{\text{change in input}} : \left\{ \begin{array}{ll} \text{moving right} & = \dfrac{\overbrace{\Delta L}^{+ve}}{\underbrace{\Delta w}_{+ve}} = +ve \\ \text{moving left} & = \dfrac{\overbrace{\Delta L}^{-ve}}{\underbrace{\Delta w}_{-ve}} = +ve \end{array} \right\} = +ve.$$

$+ve$ sensitivity	w increases $\Rightarrow$ L increases & w decreases $\Rightarrow$ L decreases
$-ve$ sensitivity	w increases $\Rightarrow$ L decreases & w decreases $\Rightarrow$ L increases

In this case, we move left (decrease) w to decrease L.

Gradient

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases}$$

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

Gradient rules in 1D using examples:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

$$\Rightarrow \nabla_w(L) = \nabla_w(L)\left(w^2\right) + \nabla_w(L)\left(5\log(w)\right) + \nabla_w(L)(6)$$

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

$$\Rightarrow \nabla_w(L) = \nabla_w(L)\left(w^2\right) + \nabla_w(L)\left(5\log(w)\right) + \nabla_w(L)(6)$$

$$\Rightarrow \nabla_w(L) = 2w + \frac{5}{w} + 0$$

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

$$\Rightarrow \nabla_w(L) = \nabla_w(L)\left(w^2\right) + \nabla_w(L)\left(5\log(w)\right) + \nabla_w(L)(6)$$

$$\Rightarrow \nabla_w(L) = 2w + \frac{5}{w} + 0$$

- $L(w) = \dfrac{1}{1+e^{-w}}$

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \quad \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

$$\Rightarrow \nabla_w(L) = \nabla_w(L)\left(w^2\right) + \nabla_w(L)\left(5\log(w)\right) + \nabla_w(L)(6)$$

$$\Rightarrow \nabla_w(L) = 2w + \frac{5}{w} + 0$$

- $L(w) = \dfrac{1}{1 + e^{-w}}$

$$\Rightarrow \nabla_w(L) \stackrel{?}{=}$$

Gradient

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \quad \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

$$\Rightarrow \nabla_w(L) = \nabla_w(L)\left(w^2\right) + \nabla_w(L)\left(5\log(w)\right) + \nabla_w(L)(6)$$

$$\Rightarrow \nabla_w(L) = 2w + \frac{5}{w} + 0$$

- $L(w) = \dfrac{1}{1+e^{-w}}$

$$\Rightarrow \nabla_w(L) \stackrel{?}{=} \frac{\nabla_w(1)}{\nabla_w\left(1+e^{-w}\right)}$$

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \quad \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

$$\Rightarrow \nabla_w(L) = \nabla_w(L)\left(w^2\right) + \nabla_w(L)\left(5\log(w)\right) + \nabla_w(L)(6)$$

$$\Rightarrow \nabla_w(L) = 2w + \frac{5}{w} + 0$$

- $L(w) = \frac{1}{1+e^{-w}}$

$$\Rightarrow \nabla_w(L) \stackrel{?}{=} \cancel{\frac{\nabla_w(1)}{\nabla_w\left(1+e^{-w}\right)}}$$: No! we use the chain rule

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \quad \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

$$\Rightarrow \nabla_w(L) = \nabla_w(L)\left(w^2\right) + \nabla_w(L)\,(5\log(w)) + \nabla_w(L)(6)$$

$$\Rightarrow \nabla_w(L) = 2w + \frac{5}{w} + 0$$

- $L(w) = \dfrac{1}{1 + e^{-w}}$

$$\Rightarrow \nabla_w(L) \stackrel{?}{=} \cancel{\frac{\nabla_w(1)}{\nabla_w\,(1 + e^{-w})}}$$: No! we use the chain rule

Gradient quantifies sensitivity:

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

$$\Rightarrow \nabla_w(L) = \nabla_w(L)\left(w^2\right) + \nabla_w(L)\left(5\log(w)\right) + \nabla_w(L)(6)$$

$$\Rightarrow \nabla_w(L) = 2w + \frac{5}{w} + 0$$

- $L(w) = \dfrac{1}{1+e^{-w}}$

$$\Rightarrow \nabla_w(L) \stackrel{?}{=} \cancel{\frac{\nabla_w(1)}{\nabla_w\left(1+e^{-w}\right)}}$$: No! we use the chain rule

Gradient quantifies sensitivity: for example, $L(w) = w^2 + 3 \Rightarrow$ sensitivity at $w = 2$ is $\nabla_w(L)|_{w=2} = 2w|_{w=2} = 4$

Gradient

The sensitivity of L w.r.t. w can be functionally represented using the gradient represented as $\nabla_w(L)$.

Useful gradients in 1D:

$$L(w) = \begin{cases} a \\ aw \\ w^2 \\ w^n \\ e^w \\ e^{-w} \\ \log(w) \\ \frac{1}{w} \end{cases} \Rightarrow \nabla_w(L) = \begin{cases} 0 \\ a \\ 2w \\ nw^{n-1} \\ e^w \\ -e^{-w} \\ \frac{1}{w} \\ -\frac{1}{w^2} \end{cases}$$

Gradient rules in 1D using examples:

- $L(w) = w^2 + 5\log(w) + 6$

$$\Rightarrow \nabla_w(L) = \nabla_w(L)\left(w^2\right) + \nabla_w(L)\left(5\log(w)\right) + \nabla_w(L)(6)$$

$$\Rightarrow \nabla_w(L) = 2w + \frac{5}{w} + 0$$

- $L(w) = \dfrac{1}{1+e^{-w}}$

$$\Rightarrow \nabla_w(L) \stackrel{?}{=} \cancel{\frac{\nabla_w(1)}{\nabla_w\left(1+e^{-w}\right)}}$$: No! we use the chain rule

Gradient quantifies sensitivity: for example, $L(w) = w^2 + 3 \Rightarrow$ sensitivity at $w = 2$ is $\nabla_w(L)\big|_{w=2} = 2w\big|_{w=2} = 4 \Rightarrow \Delta L \approx 4 \times \Delta w$ at $w = 2$.

Chain Rule

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right)$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}}$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1+e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \left\{ \right.$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1+e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ \end{cases}$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1 + e^{-w}. \end{cases}$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1 + e^{-w}. \end{cases}$

Computation graph:

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1 + e^{-w}. \end{cases}$

<u>Computation graph:</u>

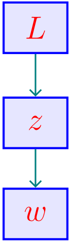
$$
\begin{tikzcd}
\boxed{L} \arrow[d] \\
\boxed{z} \arrow[d] \\
\boxed{w}
\end{tikzcd}
$$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1+e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) &= 1/z, \\ z(w) &= 1+e^{-w}. \end{cases}$

<u>Computation graph:</u> <u>Gradient calculation using chain rule:</u>

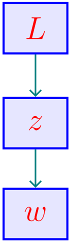


Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1+e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) &= 1/z, \\ z(w) &= 1+e^{-w}. \end{cases}$

<u>Computation graph:</u> <u>Gradient calculation using chain rule:</u>



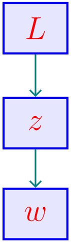
$\nabla_w(L) =$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) &= 1/z, \\ z(w) &= 1 + e^{-w}. \end{cases}$

<u>Computation graph:</u> <u>Gradient calculation using chain rule:</u>



$$\nabla_w(L) = \qquad\qquad \nabla_z(L)$$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1 + e^{-w}. \end{cases}$

Computation graph: **Gradient calculation using chain rule:**

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{z} \arrow[d] \\ \boxed{w} \end{tikzcd}$$

$$\nabla_w(L) = \nabla_w(z) \times \nabla_z(L)$$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1 + e^{-w}. \end{cases}$

Computation graph:

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{z} \arrow[d] \\ \boxed{w} \end{tikzcd}$$

Gradient calculation using chain rule:

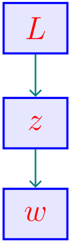
$$\nabla_w(L) = \nabla_w(z) \times \nabla_z(L)$$
$$=$$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1+e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1+e^{-w}. \end{cases}$

<u>Computation graph:</u> <u>Gradient calculation using chain rule:</u>



$$\begin{aligned} \nabla_w(L) &= \nabla_w(z) \times \nabla_z(L) \\ &= \qquad\qquad \nabla_z\left(\frac{1}{z}\right) \end{aligned}$$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1 + e^{-w}. \end{cases}$

<u>Computation graph:</u> <u>Gradient calculation using chain rule:</u>

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{z} \arrow[d] \\ \boxed{w} \end{tikzcd}$$

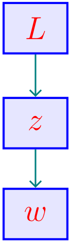
$$\begin{aligned} \nabla_w(L) &= \nabla_w(z) \times \nabla_z(L) \\ &= \nabla_w\left(1 + e^{-w}\right) \times \nabla_z\left(\frac{1}{z}\right) \\ &= \end{aligned}$$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1+e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1+e^{-w}. \end{cases}$

Computation graph:



Gradient calculation using chain rule:

$$\begin{aligned} \nabla_w(L) &= \nabla_w(z) \times \nabla_z(L) \\ &= \nabla_w\left(1+e^{-w}\right) \times \nabla_z\left(\frac{1}{z}\right) \\ &= \qquad\qquad -\frac{1}{z^2} \end{aligned}$$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1 + e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1 + e^{-w}. \end{cases}$

Computation graph:

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{z} \arrow[d] \\ \boxed{w} \end{tikzcd}$$

Gradient calculation using chain rule:

$$\begin{aligned} \nabla_w(L) &= \nabla_w(z) \times \nabla_z(L) \\ &= \nabla_w\left(1 + e^{-w}\right) \times \nabla_z\left(\frac{1}{z}\right) \\ &= \left[\nabla_w(1) + \nabla_w\left(e^{-w}\right)\right] \times -\frac{1}{z^2} \end{aligned}$$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1+e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1+e^{-w}. \end{cases}$

Computation graph:

$$\begin{array}{c} \boxed{L} \\ \downarrow \\ \boxed{z} \\ \downarrow \\ \boxed{w} \end{array}$$

Gradient calculation using chain rule:

$$\begin{aligned}
\nabla_w(L) &= \nabla_w(z) \times \nabla_z(L) \\
&= \nabla_w\left(1+e^{-w}\right) \times \nabla_z\left(\frac{1}{z}\right) \\
&= \left[\nabla_w(1) + \nabla_w\left(e^{-w}\right)\right] \times -\frac{1}{z^2} \\
&= \left[0 + \left(-e^{-w}\right)\right] \times -\frac{1}{z^2}
\end{aligned}$$

Chain Rule

The chain rule is used to calculate gradients in a hierarchical way by using intermediate variable(s).

E.g. $L(w) = 1/\left(1+e^{-w}\right) \xrightarrow{\text{use } z=1+e^{-w}} \begin{cases} L(z) & = 1/z, \\ z(w) & = 1+e^{-w}. \end{cases}$

Computation graph:

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{z} \arrow[d] \\ \boxed{w} \end{tikzcd}$$

Gradient calculation using chain rule:

$$\begin{aligned} \nabla_w(L) &= \nabla_w(z) \times \nabla_z(L) \\ &= \nabla_w\left(1+e^{-w}\right) \times \nabla_z\left(\frac{1}{z}\right) \\ &= \left[\nabla_w(1) + \nabla_w\left(e^{-w}\right)\right] \times -\frac{1}{z^2} \\ &= \left[0 + \left(-e^{-w}\right)\right] \times -\frac{1}{z^2} = \frac{e^{-w}}{\left(1+e^{-w}\right)^2}. \end{aligned}$$

Gradient in Higher Dimensions

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$			

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$		

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input: n$ $output: 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector			

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^\mathrm{T}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^\mathrm{T}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$		

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$
$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$			

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$
$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$	$input : 2$ $output : 1$		

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$
$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$	$input : 2$ $output : 1$	2×1	

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$
$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$	$input : 2$ $output : 1$	2×1	$\nabla_{\mathbf{w}}(L) = \begin{bmatrix} \nabla_{w_1}(L) \\ \nabla_{w_2}(L) \end{bmatrix} = \begin{bmatrix} 2(w_1 - 2) \\ 2(w_2 + 3) \end{bmatrix}$

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$
$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$	$input : 2$ $output : 1$	2×1	$\nabla_{\mathbf{w}}(L) = \begin{bmatrix} \nabla_{w_1}(L) \\ \nabla_{w_2}(L) \end{bmatrix} = \begin{bmatrix} 2(w_1 - 2) \\ 2(w_2 + 3) \end{bmatrix}$
$\mathbf{l}(\mathbf{z}) = \mathbf{W}\mathbf{z}$ W known $p \times k$-matrix			

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$
$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$	$input : 2$ $output : 1$	2×1	$\nabla_{\mathbf{w}}(L) = \begin{bmatrix} \nabla_{w_1}(L) \\ \nabla_{w_2}(L) \end{bmatrix} = \begin{bmatrix} 2(w_1 - 2) \\ 2(w_2 + 3) \end{bmatrix}$
$\mathbf{l}(\mathbf{z}) = \mathbf{W}\mathbf{z}$ W known $p \times k$-matrix	$input : k \times 1$ $output : p \times 1$		

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$
$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$	$input : 2$ $output : 1$	2×1	$\nabla_{\mathbf{w}}(L) = \begin{bmatrix} \nabla_{w_1}(L) \\ \nabla_{w_2}(L) \end{bmatrix} = \begin{bmatrix} 2(w_1 - 2) \\ 2(w_2 + 3) \end{bmatrix}$
$\mathbf{l}(\mathbf{z}) = \mathbf{W}\mathbf{z}$ W known $p \times k$-matrix	$input : k \times 1$ $output : p \times 1$	$k \times p$	

Gradient in Higher Dimensions

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$
$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$	$input : 2$ $output : 1$	2×1	$\nabla_{\mathbf{w}}(L) = \begin{bmatrix} \nabla_{w_1}(L) \\ \nabla_{w_2}(L) \end{bmatrix} = \begin{bmatrix} 2(w_1 - 2) \\ 2(w_2 + 3) \end{bmatrix}$
$\mathbf{l}(\mathbf{z}) = \mathbf{W}\mathbf{z}$ W known $p \times k$-matrix	$input : k \times 1$ $output : p \times 1$	$k \times p$	$\nabla_{\mathbf{z}}(\mathbf{l}) = \mathbf{W}^{\mathrm{T}}$

Gradient in Higher Dimensions

MANIPAL ACADEMY of HIGHER EDUCATION (Institution of Eminence Deemed to be University)

Gradient $\nabla_{input}(output)$ has shape $input\ shape \times output\ shape$:

Function	I/O Shapes	Grad. Shape	Gradient
$L(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{w}$	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = 2\mathbf{w}$
$L(\mathbf{w}) = \mathbf{x}^{\mathrm{T}}\mathbf{w}$ $\mathbf{x}$ known n-vector	$input : n$ $output : 1$	$n \times 1$	$\nabla_{\mathbf{w}}(L) = \mathbf{x}$
$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$	$input : 2$ $output : 1$	2×1	$\nabla_{\mathbf{w}}(L) = \begin{bmatrix} \nabla_{w_1}(L) \\ \nabla_{w_2}(L) \end{bmatrix} = \begin{bmatrix} 2(w_1 - 2) \\ 2(w_2 + 3) \end{bmatrix}$
$\mathbf{l}(\mathbf{z}) = \mathbf{W}\mathbf{z}$ W known $p \times k$-matrix	$input : k \times 1$ $output : p \times 1$	$k \times p$	$\nabla_{\mathbf{z}}(\mathbf{l}) = \mathbf{W}^{\mathrm{T}}$ note the transpose

Chain Rule Example in Higher Dimensions

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$:

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^\mathrm{T}\mathbf{x}}\right)$: $\begin{cases} & \\ z_2(\mathbf{w}) & = -\mathbf{w}^\mathrm{T}\mathbf{x}. \end{cases}$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^\mathrm{T}\mathbf{x}}\right)$: $\begin{cases} z_1(z_2) & = 1 + e^{z_2}, \\ z_2(\mathbf{w}) & = -\mathbf{w}^\mathrm{T}\mathbf{x}. \end{cases}$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) & = 1/z_1, \\ z_1(z_2) & = 1 + e^{z_2}, \\ z_2(\mathbf{w}) & = -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) &= 1/z_1, \\ z_1(z_2) &= 1 + e^{z_2}, \\ z_2(\mathbf{w}) &= -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Computation graph:

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) & = 1/z_1, \\ z_1(z_2) & = 1 + e^{z_2}, \\ z_2(\mathbf{w}) & = -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

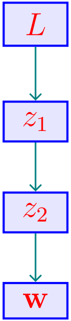
Computation graph:

$$
\begin{tikzcd}
\boxed{L} \arrow[d] \\
\boxed{z_1} \arrow[d] \\
\boxed{z_2} \arrow[d] \\
\boxed{\mathbf{w}}
\end{tikzcd}
$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) &= 1/z_1, \\ z_1(z_2) &= 1 + e^{z_2}, \\ z_2(\mathbf{w}) &= -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Computation graph:



Gradient calculation using chain rule:

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) &= 1/z_1, \\ z_1(z_2) &= 1 + e^{z_2}, \\ z_2(\mathbf{w}) &= -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Computation graph:

$$\begin{array}{c} \boxed{L} \\ \downarrow \\ \boxed{z_1} \\ \downarrow \\ \boxed{z_2} \\ \downarrow \\ \boxed{\mathbf{w}} \end{array}$$

Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) =$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) &= 1/z_1, \\ z_1(z_2) &= 1 + e^{z_2}, \\ z_2(\mathbf{w}) &= -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Computation graph:

$$
\begin{tikzcd}
\boxed{L} \arrow[d] \\
\boxed{z_1} \arrow[d] \\
\boxed{z_2} \arrow[d] \\
\boxed{\mathbf{w}}
\end{tikzcd}
$$

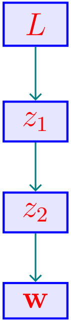
Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \qquad\qquad \nabla_{z_1}(L)$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^\mathrm{T}\mathbf{x}}\right)$: $\begin{cases} L(z_1) &= 1/z_1, \\ z_1(z_2) &= 1 + e^{z_2}, \\ z_2(\mathbf{w}) &= -\mathbf{w}^\mathrm{T}\mathbf{x}. \end{cases}$

Computation graph:



Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \qquad \nabla_{z_2}(z_1) \times \nabla_{z_1}(L)$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) & = 1/z_1, \\ z_1(z_2) & = 1 + e^{z_2}, \\ z_2(\mathbf{w}) & = -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Computation graph:

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{z_1} \arrow[d] \\ \boxed{z_2} \arrow[d] \\ \boxed{\mathbf{w}} \end{tikzcd}$$

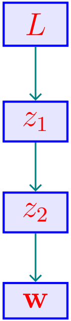
Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{w}}(z_2) \times \nabla_{z_2}(z_1) \times \nabla_{z_1}(L)$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^\mathrm{T}\mathbf{x}}\right)$: $\begin{cases} L(z_1) & = 1/z_1, \\ z_1(z_2) & = 1 + e^{z_2}, \\ z_2(\mathbf{w}) & = -\mathbf{w}^\mathrm{T}\mathbf{x}. \end{cases}$

Computation graph:



Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{w}}(z_2) \times \nabla_{z_2}(z_1) \times \nabla_{z_1}(L)$$

$$\nabla_{z_1}\left(\frac{1}{z_1}\right)$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) &= 1/z_1, \\ z_1(z_2) &= 1 + e^{z_2}, \\ z_2(\mathbf{w}) &= -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Computation graph:

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{z_1} \arrow[d] \\ \boxed{z_2} \arrow[d] \\ \boxed{\mathbf{w}} \end{tikzcd}$$

Gradient calculation using chain rule:

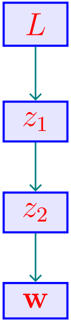
$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{w}}(z_2) \times \nabla_{z_2}(z_1) \times \nabla_{z_1}(L)$$

$$\nabla_{z_1}\left(1 + e^{z_2}\right) \times \nabla_{z_1}\left(\frac{1}{z_1}\right)$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) & = 1/z_1, \\ z_1(z_2) & = 1 + e^{z_2}, \\ z_2(\mathbf{w}) & = -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Computation graph:



Gradient calculation using chain rule:

$$\begin{aligned} \nabla_{\mathbf{w}}(L) &= \nabla_{\mathbf{w}}(z_2) \times \nabla_{z_2}(z_1) \times \nabla_{z_1}(L) \\ &= \nabla_{\mathbf{w}}\left(-\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) \times \nabla_{z_1}\left(1 + e^{z_2}\right) \times \nabla_{z_1}\left(\frac{1}{z_1}\right) \end{aligned}$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^\mathrm{T}\mathbf{x}}\right)$: $\begin{cases} L(z_1) & = 1/z_1, \\ z_1(z_2) & = 1 + e^{z_2}, \\ z_2(\mathbf{w}) & = -\mathbf{w}^\mathrm{T}\mathbf{x}. \end{cases}$

<u>Computation graph:</u>

$$
\begin{tikzcd}
L \arrow[d] \\
z_1 \arrow[d] \\
z_2 \arrow[d] \\
\mathbf{w}
\end{tikzcd}
$$

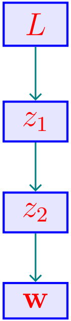
<u>Gradient calculation using chain rule:</u>

$$
\begin{aligned}
\nabla_\mathbf{w}(L) &= \nabla_\mathbf{w}(z_2) \times \nabla_{z_2}(z_1) \times \nabla_{z_1}(L) \\
&= \nabla_\mathbf{w}\left(-\mathbf{w}^\mathrm{T}\mathbf{x}\right) \times \nabla_{z_1}\left(1 + e^{z_2}\right) \times \nabla_{z_1}\left(\frac{1}{z_1}\right) \\
& \qquad\qquad\qquad\qquad - 1/z_1^2
\end{aligned}
$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) &= 1/z_1, \\ z_1(z_2) &= 1 + e^{z_2}, \\ z_2(\mathbf{w}) &= -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Computation graph:



Gradient calculation using chain rule:

$$\nabla_{\mathbf{w}}(L) = \nabla_{\mathbf{w}}(z_2) \times \nabla_{z_2}(z_1) \times \nabla_{z_1}(L)$$

$$= \nabla_{\mathbf{w}}\left(-\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) \times \nabla_{z_1}\left(1 + e^{z_2}\right) \times \nabla_{z_1}\left(\frac{1}{z_1}\right)$$

$$e^{z_2} \times -1/z_1^2$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$: $\begin{cases} L(z_1) &= 1/z_1, \\ z_1(z_2) &= 1 + e^{z_2}, \\ z_2(\mathbf{w}) &= -\mathbf{w}^{\mathrm{T}}\mathbf{x}. \end{cases}$

Computation graph:

$$\begin{tikzcd} \boxed{L} \arrow[d] \\ \boxed{z_1} \arrow[d] \\ \boxed{z_2} \arrow[d] \\ \boxed{\mathbf{w}} \end{tikzcd}$$

Gradient calculation using chain rule:

$$\begin{aligned} \nabla_{\mathbf{w}}(L) &= \nabla_{\mathbf{w}}(z_2) \times \nabla_{z_2}(z_1) \times \nabla_{z_1}(L) \\ &= \nabla_{\mathbf{w}}\left(-\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) \times \nabla_{z_1}\left(1 + e^{z_2}\right) \times \nabla_{z_1}\left(\frac{1}{z_1}\right) \\ &= -\mathbf{x} \times e^{z_2} \times -1/z_1^2 \end{aligned}$$

Chain Rule Example in Higher Dimensions

Calculate the gradient of $L(\mathbf{w}) = 1/\left(1 + e^{-\mathbf{w}^T\mathbf{x}}\right)$: $\begin{cases} L(z_1) & = 1/z_1, \\ z_1(z_2) & = 1 + e^{z_2}, \\ z_2(\mathbf{w}) & = -\mathbf{w}^T\mathbf{x}. \end{cases}$

Computation graph:

$$\begin{tikzcd} L \arrow[d] \\ z_1 \arrow[d] \\ z_2 \arrow[d] \\ \mathbf{w} \end{tikzcd}$$

Gradient calculation using chain rule:

$$\begin{aligned}
\nabla_{\mathbf{w}}(L) &= \nabla_{\mathbf{w}}(z_2) \times \nabla_{z_2}(z_1) \times \nabla_{z_1}(L) \\
&= \nabla_{\mathbf{w}}\left(-\mathbf{w}^T\mathbf{x}\right) \times \nabla_{z_1}\left(1 + e^{z_2}\right) \times \nabla_{z_1}\left(\frac{1}{z_1}\right) \\
&= -\mathbf{x} \times e^{z_2} \times -1/z_1^2 \\
&= \frac{e^{-\mathbf{w}^T\mathbf{x}}}{\left(1 + e^{-\mathbf{w}^T\mathbf{x}}\right)^2}\mathbf{x}.
\end{aligned}$$

Gradient Descent Method

Gradient Descent Method

The gradient (seen as a vector) is the direction of steepest ascent

Gradient Descent Method

The gradient (seen as a vector) is the direction of steepest ascent $\Rightarrow$ the negative of the gradient is the direction of steepest descent.

Gradient Descent Method

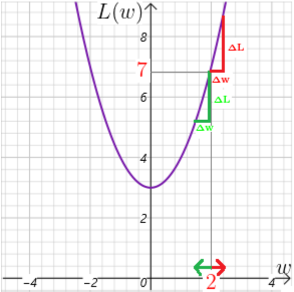
The gradient (seen as a vector) is the direction of steepest ascent $\Rightarrow$ the negative of the gradient is the direction of steepest descent.

Consider $L(w) = w^2 + 3$ at $w = 2$:

Gradient Descent Method

The gradient (seen as a vector) is the direction of steepest ascent $\Rightarrow$ the negative of the gradient is the direction of steepest descent.

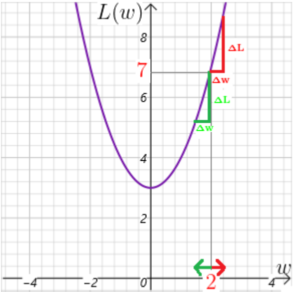
Consider $L(w) = w^2 + 3$ at $w = 2$:



Gradient Descent Method

The gradient (seen as a vector) is the direction of steepest ascent $\Rightarrow$ the negative of the gradient is the direction of steepest descent.

Consider $L(w) = w^2 + 3$ at $w = 2$:

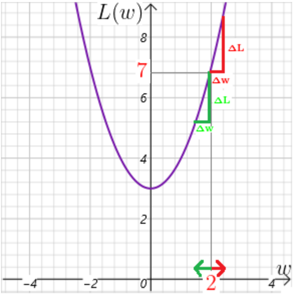


- The gradient at $w = 2$ is

Gradient Descent Method

The gradient (seen as a vector) is the direction of steepest ascent $\Rightarrow$ the negative of the gradient is the direction of steepest descent.

Consider $L(w) = w^2 + 3$ at $w = 2$:

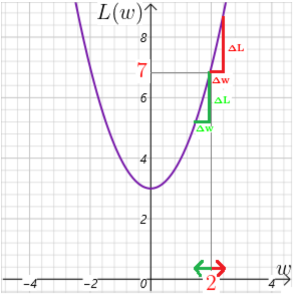


- The gradient at $w = 2$ is $\nabla_w(L)|_{w=2} = 2w|_{w=2} = \begin{bmatrix}4\end{bmatrix}$.

Gradient Descent Method

The gradient (seen as a vector) is the direction of steepest ascent $\Rightarrow$ the negative of the gradient is the direction of steepest descent.

Consider $L(w) = w^2 + 3$ at $w = 2$:

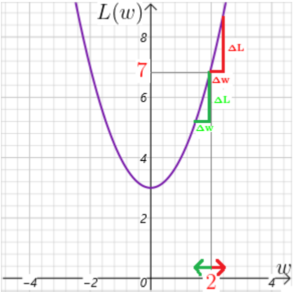


- The gradient at $w = 2$ is $\nabla_w (L)|_{w=2} = 2w|_{w=2} = \left[4\right].$
- This means, we move in the negative gradient direction: $-\nabla_w (L)|_{w=2} = -\left[4\right].$

Gradient Descent Method

The gradient (seen as a vector) is the direction of steepest ascent $\Rightarrow$ the negative of the gradient is the direction of steepest descent.

Consider $L(w) = w^2 + 3$ at $w = 2$:

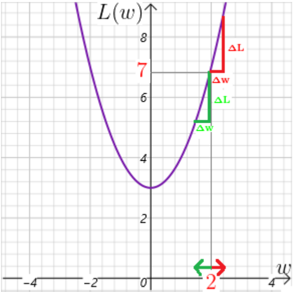


- The gradient at $w = 2$ is $\nabla_w(L)|_{w=2} = 2w|_{w=2} = [4]$.
- This means, we move in the negative gradient direction: $-\nabla_w(L)|_{w=2} = -[4]$.
- How much should we move?

Gradient Descent Method

The gradient (seen as a vector) is the direction of steepest ascent $\Rightarrow$ the negative of the gradient is the direction of steepest descent.

Consider $L(w) = w^2 + 3$ at $w = 2$:



- The gradient at $w = 2$ is $\nabla_w(L)|_{w=2} = 2w|_{w=2} = \left[4\right]$.
- This means, we move in the negative gradient direction: $-\nabla_w(L)|_{w=2} = -\left[4\right]$.
- How much should we move? We move by a specific small amount known as the learning rate α.

Gradient Descent Method – continued

Gradient Descent Method – continued

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.

Gradient Descent Method – continued

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.

Gradient Descent Method – continued

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.
- The gradient descent iteration: $w = w - \alpha \times \nabla_w(L)|_w$.

Gradient Descent Method – continued

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.
- The gradient descent iteration: $w = w - \alpha \times \nabla_w(L)\big|_w$.

new w value

Gradient Descent Method – continued

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.
- The gradient descent iteration: $w = \underbrace{w}_{\text{old } w \text{ value}} - \alpha \times \nabla_w(L)|_w$.

Gradient Descent Method – continued

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.
- The gradient descent iteration: $w = w - \alpha \times \nabla_w(L)|_w$.

learning rate

Gradient Descent Method – continued

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.
- The gradient descent iteration: $w = w - \alpha \times \nabla_w(L)|_w$.

gradient at old w value

Gradient Descent Method – continued

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.
- The gradient descent iteration: $w = w - \alpha \times \nabla_w(L)|_w$.
- The same update works in higher dimensions as well:

Gradient Descent Method – continued

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.
- The gradient descent iteration: $w = w - \alpha \times \nabla_w(L)|_w$.
- The same update works in higher dimensions as well:

$$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$$

Gradient Descent Method – continued

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.
- The gradient descent iteration: $w = w - \alpha \times \nabla_w(L)|_w$.
- The same update works in higher dimensions as well:

$$L(\mathbf{w}) = (w_1 - 2)^2 + (w_2 + 3)^2$$

$$\Rightarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \left. \begin{bmatrix} \nabla_{w_1}\left((w_1 - 2)^2 + (w_2 + 3)^2\right) \\ \nabla_{w_2}\left((w_1 - 2)^2 + (w_2 + 3)^2\right) \end{bmatrix} \right|_{\mathbf{w}}$$

Gradient Descent Method – continued

- We keep repeating this process iteratively until the gradient is close to zero and/or there is no significant change in the w values.
- This will take us, hopefully, to the point w at which L has the smallest value.
- The gradient descent iteration: $w = w - \alpha \times \nabla_w(L)\big|_w$.
- The same update works in higher dimensions as well:

$$
\begin{aligned}
L(\mathbf{w}) &= (w_1 - 2)^2 + (w_2 + 3)^2 \\
\Rightarrow \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} &= \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \left.\begin{bmatrix} \nabla_{w_1}\left((w_1 - 2)^2 + (w_2 + 3)^2\right) \\ \nabla_{w_2}\left((w_1 - 2)^2 + (w_2 + 3)^2\right) \end{bmatrix}\right|_{\mathbf{w}} \\
&= \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \alpha \left.\begin{bmatrix} 2(w_1 - 2) \\ 3(w_2 + 3) \end{bmatrix}\right|_{\mathbf{w}}.
\end{aligned}
$$

Bias Trick

Bias Trick

In calculating the raw score of a sample
$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

Bias Trick

In calculating the raw score of a sample
$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} \quad \\ \quad \\ \quad \end{bmatrix}}_{\mathbf{W}}$$

Bias Trick

In calculating the raw score of a sample $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ & & & \\ & & & \end{bmatrix}}_{\mathbf{W}}$$

Bias Trick

In calculating the raw score of a sample
$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ & & & \end{bmatrix}}_{\mathbf{W}}$$

Bias Trick

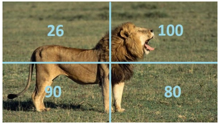
In calculating the raw score of a sample
$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}}$$

Bias Trick

In calculating the raw score of a sample $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}}$$



Bias Trick

In calculating the raw score of a sample
$\mathbf{z} = \mathbf{W}\mathbf{x}+\mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \qquad \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as } 4\text{-vector } \mathbf{x}}$$

Bias Trick

In calculating the raw score of a sample

$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as } 4\text{-vector}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}}$$

Bias Trick

In calculating the raw score of a sample
$\mathbf{z} = \mathbf{W}\mathbf{x}+\mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as } 4\text{-vector}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 & \boxed{1.0} \\ 2.3 & 0.8 & 1.2 & 0.5 & \boxed{-1.5} \\ 0 & -1 & 0.5 & 1.0 & \boxed{1.0} \end{bmatrix}}_{\mathbf{W}\text{bias column added}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \\ \boxed{1} \end{bmatrix}}_{\mathbf{x}\text{with bias feature}}$$

Bias Trick

In calculating the raw score of a sample
$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as 4-vector}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 & \boxed{1.0} \\ 2.3 & 0.8 & 1.2 & 0.5 & \boxed{-1.5} \\ 0 & -1 & 0.5 & 1.0 & \boxed{1.0} \end{bmatrix}}_{\mathbf{W}\text{ bias column added}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \\ \boxed{1} \end{bmatrix}}_{\mathbf{x}\text{ with bias feature}}$$

- Last column of the weights matrix hold the bias values.

Bias Trick

In calculating the raw score of a sample
$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix}0.1 & -0.1 & 0 & 0.5\\ 2.3 & 0.8 & 1.2 & 0.5\\ 0 & -1 & 0.5 & 1.0\end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix}26\\ 100\\ 90\\ 80\end{bmatrix}}_{\text{image as 4-vector}} + \underbrace{\begin{bmatrix}1.0\\ -1.5\\ 1.0\end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix}0.1 & -0.1 & 0 & 0.5 & \boxed{1.0}\\ 2.3 & 0.8 & 1.2 & 0.5 & \boxed{-1.5}\\ 0 & -1 & 0.5 & 1.0 & \boxed{1.0}\end{bmatrix}}_{\mathbf{W}\text{bias column added}} \underbrace{\begin{bmatrix}26\\ 100\\ 90\\ 80\\ \boxed{1}\end{bmatrix}}_{\mathbf{x}\text{with bias feature}}$$

- Last column of the weights matrix hold the bias values.
- The bias feature with value 1 gets *appended* to the sample vector.

Bias Trick

In calculating the raw score of a sample
$\mathbf{z} = \mathbf{W}\mathbf{x}+\mathbf{b}$, it is possible to absorb the bias values into the weights matrix:

$$\underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 \\ 2.3 & 0.8 & 1.2 & 0.5 \\ 0 & -1 & 0.5 & 1.0 \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \end{bmatrix}}_{\text{image as } 4\text{-vector}} + \underbrace{\begin{bmatrix} 1.0 \\ -1.5 \\ 1.0 \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} 0.1 & -0.1 & 0 & 0.5 & \boxed{1.0} \\ 2.3 & 0.8 & 1.2 & 0.5 & \boxed{-1.5} \\ 0 & -1 & 0.5 & 1.0 & \boxed{1.0} \end{bmatrix}}_{\mathbf{W}_{\text{bias column added}}} \underbrace{\begin{bmatrix} 26 \\ 100 \\ 90 \\ 80 \\ \boxed{1} \end{bmatrix}}_{\mathbf{x}_{\text{with bias feature}}}$$

- Last column of the weights matrix hold the bias values.
- The bias feature with value 1 gets *appended* to the sample vector.
- This helps in simplifying gradient calculations for computing optimal weights and bias values without having to account for the bias separately.

Softmax Classifier Setup

Softmax Classifier Setup

- Suppose we have n samples each with p features: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$, with labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ belonging to k classes.

Softmax Classifier Setup

- Suppose we have n samples each with p features: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$, with labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ belonging to k classes.
- Assume that bias trick has been performed resulting in a $k \times (p+1)$-weights matrix $\mathbf{W}$ and a $(p+1) \times n$-data matrix $\mathbf{X}$.

Softmax Classifier Setup

- Suppose we have n samples each with p features: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$, with labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ belonging to k classes.
- Assume that bias trick has been performed resulting in a $k \times (p+1)$-weights matrix $\mathbf{W}$ and a $(p+1) \times n$-data matrix $\mathbf{X}$.
- Note that the columns of the data matrix $\mathbf{X}$ correspond to the samples with the last row equal to ones (the bias feature).

Softmax Classifier Setup

- Suppose we have n samples each with p features: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$, with labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ belonging to k classes.
- Assume that bias trick has been performed resulting in a $k \times (p+1)$-weights matrix $\mathbf{W}$ and a $(p+1) \times n$-data matrix $\mathbf{X}$.
- Note that the columns of the data matrix $\mathbf{X}$ correspond to the samples with the last row equal to ones (the bias feature).
- The raw scores for all samples can be computed as the $k \times n$-matrix

Softmax Classifier Setup

- Suppose we have n samples each with p features: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$, with labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ belonging to k classes.
- Assume that bias trick has been performed resulting in a $k \times (p+1)$-weights matrix $\mathbf{W}$ and a $(p+1) \times n$-data matrix $\mathbf{X}$.
- Note that the columns of the data matrix $\mathbf{X}$ correspond to the samples with the last row equal to ones (the bias feature).
- The raw scores for all samples can be computed as the $k \times n$-matrix

$$\mathbf{Z} = \mathbf{W}\mathbf{X}$$

Softmax Classifier Setup

- Suppose we have n samples each with p features: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$, with labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ belonging to k classes.
- Assume that bias trick has been performed resulting in a $k \times (p+1)$-weights matrix $\mathbf{W}$ and a $(p+1) \times n$-data matrix $\mathbf{X}$.
- Note that the columns of the data matrix $\mathbf{X}$ correspond to the samples with the last row equal to ones (the bias feature).
- The raw scores for all samples can be computed as the $k \times n$-matrix

$$\mathbf{Z} = \mathbf{W}\mathbf{X} = \mathbf{W}\begin{bmatrix}\mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \ldots & \mathbf{x}^{(n)}\end{bmatrix}$$

Softmax Classifier Setup

- Suppose we have n samples each with p features: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$, with labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ belonging to k classes.
- Assume that bias trick has been performed resulting in a $k \times (p+1)$-weights matrix $\mathbf{W}$ and a $(p+1) \times n$-data matrix $\mathbf{X}$.
- Note that the columns of the data matrix $\mathbf{X}$ correspond to the samples with the last row equal to ones (the bias feature).
- The raw scores for all samples can be computed as the $k \times n$-matrix

$$\mathbf{Z} = \mathbf{W}\mathbf{X} = \mathbf{W}\begin{bmatrix}\mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \ldots & \mathbf{x}^{(n)}\end{bmatrix} = \begin{bmatrix}\mathbf{W}\mathbf{x}^{(1)} & \mathbf{W}\mathbf{x}^{(2)} & \ldots & \mathbf{W}\mathbf{x}^{(n)}\end{bmatrix}$$

Softmax Classifier Setup

- Suppose we have n samples each with p features: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$, with labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ belonging to k classes.
- Assume that bias trick has been performed resulting in a $k \times (p+1)$-weights matrix $\mathbf{W}$ and a $(p+1) \times n$-data matrix $\mathbf{X}$.
- Note that the columns of the data matrix $\mathbf{X}$ correspond to the samples with the last row equal to ones (the bias feature).
- The raw scores for all samples can be computed as the $k \times n$-matrix

$$\mathbf{Z} = \mathbf{W}\mathbf{X} = \mathbf{W}\begin{bmatrix}\mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \ldots & \mathbf{x}^{(n)}\end{bmatrix} = \begin{bmatrix}\mathbf{W}\mathbf{x}^{(1)} & \mathbf{W}\mathbf{x}^{(2)} & \ldots & \mathbf{W}\mathbf{x}^{(n)}\end{bmatrix} = \begin{bmatrix}\mathbf{z}^{(1)} & \mathbf{z}^{(2)} & \ldots & \mathbf{z}^{(n)}\end{bmatrix}.$$

Softmax Classifier Setup

- Suppose we have n samples each with p features: $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$, with labels $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ belonging to k classes.
- Assume that bias trick has been performed resulting in a $k \times (p+1)$-weights matrix $\mathbf{W}$ and a $(p+1) \times n$-data matrix $\mathbf{X}$.
- Note that the columns of the data matrix $\mathbf{X}$ correspond to the samples with the last row equal to ones (the bias feature).
- The raw scores for all samples can be computed as the $k \times n$-matrix

$$\mathbf{Z} = \mathbf{W}\mathbf{X} = \mathbf{W}\begin{bmatrix}\mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \ldots & \mathbf{x}^{(n)}\end{bmatrix} = \begin{bmatrix}\mathbf{W}\mathbf{x}^{(1)} & \mathbf{W}\mathbf{x}^{(2)} & \ldots & \mathbf{W}\mathbf{x}^{(n)}\end{bmatrix} = \begin{bmatrix}\mathbf{z}^{(1)} & \mathbf{z}^{(2)} & \ldots & \mathbf{z}^{(n)}\end{bmatrix}.$$

- The ith sample's softmax loss is $-\log\left(\left[\text{softmax}\left(\mathbf{z}^{(i)}\right)\right]_{y^{(i)}}\right)$.

Softmax Classifier Gradient

Softmax Classifier Gradient

- The average training softmax loss

$$L(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L_i = \frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{j=1}^{k} e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}^{(i)}}}\right).$$

Softmax Classifier Gradient

- The average training softmax loss

$$L(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L_i = \frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{j=1}^{k} e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}^{(i)}}}\right).$$

- The gradient $\nabla_{\mathbf{W}}(L)$ has shape $\underbrace{k \times (p+1)}_{\text{input shape}} \times \underbrace{1}_{\text{output shape}} = k \times (p+1)$.

Softmax Classifier Gradient

- The average training softmax loss

$$L(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L_i = \frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{j=1}^{k} e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}^{(i)}}}\right).$$

- The gradient $\nabla_{\mathbf{W}}(L)$ has shape $\underbrace{k \times (p+1)}_{\text{input shape}} \times \underbrace{1}_{\text{output shape}} = k \times (p+1)$.

- Weights matrix

Softmax Classifier Gradient

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

- The average training softmax loss

$$L(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L_i = \frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{j=1}^{k} e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}^{(i)}}}\right).$$

- The gradient $\nabla_{\mathbf{W}}(L)$ has shape $\underbrace{k \times (p+1)}_{\text{input shape}} \times \underbrace{1}_{\text{output shape}} = k \times (p+1)$.

- Weights matrix $\mathbf{W} = \begin{bmatrix} \underbrace{\mathbf{w}_1}_{1\times(p+1)}{}^{\mathrm{T}} \\ \underbrace{\mathbf{w}_2}_{1\times(p+1)}{}^{\mathrm{T}} \\ \vdots \\ \underbrace{\mathbf{w}_k}_{1\times(p+1)}{}^{\mathrm{T}} \end{bmatrix}$,

Softmax Classifier Gradient

- The average training softmax loss

$$L(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L_i = \frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{j=1}^{k} e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}^{(i)}}}\right).$$

- The gradient $\nabla_{\mathbf{W}}(L)$ has shape $\underbrace{k \times (p+1)}_{\text{input shape}} \times \underbrace{1}_{\text{output shape}} = k \times (p+1)$.

- Weights matrix $\mathbf{W} = \begin{bmatrix} \underbrace{\mathbf{w}_1{}^{\mathrm{T}}}_{1\times(p+1)} \\ \underbrace{\mathbf{w}_2{}^{\mathrm{T}}}_{1\times(p+1)} \\ \vdots \\ \underbrace{\mathbf{w}_k{}^{\mathrm{T}}}_{1\times(p+1)} \end{bmatrix}$, gradient $\nabla_{\mathbf{W}}(L) = \begin{bmatrix} \underbrace{\left(\nabla_{\mathbf{w}_1}(L)\right)^{\mathrm{T}}}_{1\times(p+1)} \\ \underbrace{\left(\nabla_{\mathbf{w}_2}(L)\right)^{\mathrm{T}}}_{1\times(p+1)} \\ \vdots \\ \underbrace{\left(\nabla_{\mathbf{w}_k}(L)\right)^{\mathrm{T}}}_{1\times(p+1)} \end{bmatrix}$.

Softmax Classifier Gradient

- The average training softmax loss

$$L(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L_i = \frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{j=1}^{k} e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}^{(i)}}}\right).$$

- The gradient $\nabla_{\mathbf{W}}(L)$ has shape $\underbrace{k \times (p+1)}_{\text{input shape}} \times \underbrace{1}_{\text{output shape}} = k \times (p+1)$.

- Weights matrix $\mathbf{W} = \begin{bmatrix} \underbrace{\mathbf{w}_1^{\mathrm{T}}}_{1\times(p+1)} \\ \underbrace{\mathbf{w}_2^{\mathrm{T}}}_{1\times(p+1)} \\ \vdots \\ \underbrace{\mathbf{w}_k^{\mathrm{T}}}_{1\times(p+1)} \end{bmatrix}$, gradient $\nabla_{\mathbf{W}}(L) = \begin{bmatrix} \underbrace{\left(\nabla_{\mathbf{w}_1}(L)\right)^{\mathrm{T}}}_{1\times(p+1)} \\ \underbrace{\left(\nabla_{\mathbf{w}_2}(L)\right)^{\mathrm{T}}}_{1\times(p+1)} \\ \vdots \\ \underbrace{\left(\nabla_{\mathbf{w}_k}(L)\right)^{\mathrm{T}}}_{1\times(p+1)} \end{bmatrix}$.

focus on term like this

Softmax Classifier Gradient – continued

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L)$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right)$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

$$\log(a/b) = \log(a) - \log(b)$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

$$= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right)$$

Softmax Classifier Gradient – continued

$$
\begin{aligned}
\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) &= \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right) \\
&= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right) \\
&= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]
\end{aligned}
$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

$$= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$\log(e^{a}) = a$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

$$= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

Softmax Classifier Gradient – continued

$$
\begin{aligned}
\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) &= \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right) \\
&= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right) \\
&= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right] \\
&= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\underset{\uparrow}{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]
\end{aligned}
$$

$$
\begin{cases} = \mathbf{x}^{(i)} & \text{if } j = y^{(i)}, \\ = 0 & \text{if } j \neq y^{(i)}. \end{cases}
$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

$$= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$I\left(y^{(i)} = j\right)\mathbf{x}^{(i)}$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

$$= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

use chain rule: $\begin{cases} L &= \log(z), \\ z &= \sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}. \end{cases}$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

$$= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}\right) - \underbrace{\nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)}_{\boxed{\frac{e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}}\mathbf{x}^{(i)}}\right]$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

$$= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left[-I\left(y^{(i)} = j\right)\mathbf{x}^{(i)} + \hat{p}_{ji}\mathbf{x}^{(i)}\right]$$

Softmax Classifier Gradient – continued

$$\nabla_{\mathbf{w}_j}(L) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} L_i\right) = \nabla_{\mathbf{w}_j}\left(\frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}}{\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}}\right)\right)$$

$$= -\frac{1}{n}\nabla_{\mathbf{w}_j}\left(\sum_{i=1}^{n}\left[\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right) - \log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right]\right)$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\log\left(e^{\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}\left[\nabla_{\mathbf{w}_j}\left(\mathbf{w}_{y^{(i)}}^{\mathrm{T}}\mathbf{x}^{(i)}\right) - \nabla_{\mathbf{w}_j}\left(\log\left(\sum_{r=1}^{k} e^{\mathbf{w}_r^{\mathrm{T}}\mathbf{x}^{(i)}}\right)\right)\right]$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left[-I\left(y^{(i)} = j\right)\mathbf{x}^{(i)} + \underbrace{\hat{p}_{ji}}_{\text{predicted probability that sample-}i\text{ belongs to class }j}\mathbf{x}^{(i)}\right]$$

Softmax Classifier Gradient – continued

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{W}}(L) = \begin{bmatrix} \left(\nabla_{\mathbf{w}_1}(L)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_j}(L)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_k}(L)\right)^{\mathrm{T}} \end{bmatrix}$$

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{W}}(L) = \begin{bmatrix} \left(\nabla_{\mathbf{w}_1}(L)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_j}(L)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_k}(L)\right)^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_{1i} - I\left(y^{(i)} = 1\right)\right]\mathbf{x}^{(i)}\right)^{\mathrm{T}} \\ \vdots \\ \left(\frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_{ji} - I\left(y^{(i)} = j\right)\right]\mathbf{x}^{(i)}\right)^{\mathrm{T}} \\ \vdots \\ \left(\frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_{ki} - I\left(y^{(i)} = k\right)\right]\mathbf{x}^{(i)}\right)^{\mathrm{T}} \end{bmatrix}$$

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{W}}(L) = \begin{bmatrix} \left(\nabla_{\mathbf{w}_1}(L)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_j}(L)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_k}(L)\right)^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_{1i} - I\left(y^{(i)} = 1\right)\right]\mathbf{x}^{(i)}\right)^{\mathrm{T}} \\ \vdots \\ \left(\frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_{ji} - I\left(y^{(i)} = j\right)\right]\mathbf{x}^{(i)}\right)^{\mathrm{T}} \\ \vdots \\ \left(\frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_{ki} - I\left(y^{(i)} = k\right)\right]\mathbf{x}^{(i)}\right)^{\mathrm{T}} \end{bmatrix} = \frac{1}{n}\sum_{i=1}^{n}\left(\begin{bmatrix} \left[\hat{p}_{1i} - I\left(y^{(i)} = 1\right)\right]\mathbf{x}^{(i)\mathrm{T}} \\ \vdots \\ \left[\hat{p}_{ji} - I\left(y^{(i)} = j\right)\right]\mathbf{x}^{(i)\mathrm{T}} \\ \vdots \\ \left[\hat{p}_{ki} - I\left(y^{(i)} = k\right)\right]\mathbf{x}^{(i)\mathrm{T}} \end{bmatrix}\right)$$

Softmax Classifier Gradient – continued

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

$$\Rightarrow \nabla_{\mathbf{W}}(L) = \begin{bmatrix} \left(\nabla_{\mathbf{w}_1}(L)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_j}(L)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_k}(L)\right)^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \left(\frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_{1i} - I\left(y^{(i)}=1\right)\right]\mathbf{x}^{(i)}\right)^{\mathrm{T}} \\ \vdots \\ \left(\frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_{ji} - I\left(y^{(i)}=j\right)\right]\mathbf{x}^{(i)}\right)^{\mathrm{T}} \\ \vdots \\ \left(\frac{1}{n}\sum_{i=1}^{n}\left[\hat{p}_{ki} - I\left(y^{(i)}=k\right)\right]\mathbf{x}^{(i)}\right)^{\mathrm{T}} \end{bmatrix} = \frac{1}{n}\sum_{i=1}^{n}\left(\begin{bmatrix} \left[\hat{p}_{1i} - I\left(y^{(i)}=1\right)\right]\mathbf{x}^{(i)\,\mathrm{T}} \\ \vdots \\ \left[\hat{p}_{ji} - I\left(y^{(i)}=j\right)\right]\mathbf{x}^{(i)\,\mathrm{T}} \\ \vdots \\ \left[\hat{p}_{ki} - I\left(y^{(i)}=k\right)\right]\mathbf{x}^{(i)\,\mathrm{T}} \end{bmatrix}\right)$$

$$= \frac{1}{n}\left(\begin{bmatrix} \hat{p}_{11} - I\left(y^{(1)}=1\right) \\ \vdots \\ \hat{p}_{j1} - I\left(y^{(1)}=j\right) \\ \vdots \\ \hat{p}_{k1} - I\left(y^{(1)}=k\right) \end{bmatrix}\mathbf{x}^{(1)\,\mathrm{T}} + \begin{bmatrix} \hat{p}_{12} - I\left(y^{(2)}=1\right) \\ \vdots \\ \hat{p}_{j2} - I\left(y^{(2)}=j\right) \\ \vdots \\ \hat{p}_{k2} - I\left(y^{(2)}=k\right) \end{bmatrix}\mathbf{x}^{(2)\,\mathrm{T}} + \cdots + \begin{bmatrix} \hat{p}_{1n} - I\left(y^{(n)}=1\right) \\ \vdots \\ \hat{p}_{jn} - I\left(y^{(n)}=j\right) \\ \vdots \\ \hat{p}_{kn} - I\left(y^{(n)}=k\right) \end{bmatrix}\mathbf{x}^{(n)\,\mathrm{T}}\right)$$

Softmax Classifier Gradient – continued

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{w}}(L) = \frac{1}{n} \begin{bmatrix} \hat{p}_{11} - I\left(y^{(1)} = 1\right) & \hat{p}_{12} - I\left(y^{(2)} = 1\right) & \ldots & \hat{p}_{1n} - I\left(y^{(n)} = 1\right) \\ \vdots & \vdots & \vdots & \\ \hat{p}_{j1} - I\left(y^{(1)} = j\right) & \hat{p}_{j2} - I\left(y^{(2)} = j\right) & \ldots & \hat{p}_{jn} - I\left(y^{(n)} = j\right) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{p}_{k1} - I\left(y^{(1)} = k\right) & \hat{p}_{k2} - I\left(y^{(2)} = k\right) & \ldots & \hat{p}_{kn} - I\left(y^{(n)} = k\right) \end{bmatrix} \begin{bmatrix} \mathbf{x}^{(1)^{\mathrm{T}}} \\ \mathbf{x}^{(2)^{\mathrm{T}}} \\ \vdots \\ \vdots \\ \mathbf{x}^{(n)^{\mathrm{T}}} \end{bmatrix}$$

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{w}}(L) = \frac{1}{n}\begin{bmatrix} \hat{p}_{11} - I\left(y^{(1)} = 1\right) & \hat{p}_{12} - I\left(y^{(2)} = 1\right) & \dots & \hat{p}_{1n} - I\left(y^{(n)} = 1\right) \\ \vdots & \vdots & \vdots & \\ \hat{p}_{j1} - I\left(y^{(1)} = j\right) & \hat{p}_{j2} - I\left(y^{(2)} = j\right) & \dots & \hat{p}_{jn} - I\left(y^{(n)} = j\right) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{p}_{k1} - I\left(y^{(1)} = k\right) & \hat{p}_{k2} - I\left(y^{(2)} = k\right) & \dots & \hat{p}_{kn} - I\left(y^{(n)} = k\right) \end{bmatrix}\begin{bmatrix} \mathbf{x}^{(1)^{\mathrm{T}}} \\ \mathbf{x}^{(2)^{\mathrm{T}}} \\ \vdots \\ \vdots \\ \mathbf{x}^{(n)^{\mathrm{T}}} \end{bmatrix}$$

$$= \frac{1}{n}\mathbf{P}_{\text{adjusted}}\mathbf{X}^{\mathrm{T}}.$$

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{w}}(L) = \frac{1}{n}\begin{bmatrix} \hat{p}_{11} - I\left(y^{(1)} = 1\right) & \hat{p}_{12} - I\left(y^{(2)} = 1\right) & \dots & \hat{p}_{1n} - I\left(y^{(n)} = 1\right) \\ \vdots & \vdots & \vdots & \\ \hat{p}_{j1} - I\left(y^{(1)} = j\right) & \hat{p}_{j2} - I\left(y^{(2)} = j\right) & \dots & \hat{p}_{jn} - I\left(y^{(n)} = j\right) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{p}_{k1} - I\left(y^{(1)} = k\right) & \hat{p}_{k2} - I\left(y^{(2)} = k\right) & \dots & \hat{p}_{kn} - I\left(y^{(n)} = k\right) \end{bmatrix} \begin{bmatrix} \mathbf{x}^{(1)\,\mathrm{T}} \\ \mathbf{x}^{(2)\,\mathrm{T}} \\ \vdots \\ \vdots \\ \mathbf{x}^{(n)\,\mathrm{T}} \end{bmatrix}$$

$$= \frac{1}{n}\mathbf{P}_{\text{adjusted}}\mathbf{X}^{\mathrm{T}}.$$

for each sample, correct class predicted probability minus one; incorrect class predicted probabilities untouched

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{w}}(L) = \frac{1}{n}\begin{bmatrix} \hat{p}_{11} - I\left(y^{(1)} = 1\right) & \hat{p}_{12} - I\left(y^{(2)} = 1\right) & \dots & \hat{p}_{1n} - I\left(y^{(n)} = 1\right) \\ \vdots & \vdots & \vdots & \\ \hat{p}_{j1} - I\left(y^{(1)} = j\right) & \hat{p}_{j2} - I\left(y^{(2)} = j\right) & \dots & \hat{p}_{jn} - I\left(y^{(n)} = j\right) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{p}_{k1} - I\left(y^{(1)} = k\right) & \hat{p}_{k2} - I\left(y^{(2)} = k\right) & \dots & \hat{p}_{kn} - I\left(y^{(n)} = k\right) \end{bmatrix} \begin{bmatrix} \mathbf{x}^{(1)\,\mathrm{T}} \\ \mathbf{x}^{(2)\,\mathrm{T}} \\ \vdots \\ \vdots \\ \mathbf{x}^{(n)\,\mathrm{T}} \end{bmatrix}$$

$$= \frac{1}{n}\mathbf{P}_{\text{adjusted}}\mathbf{X}^{\mathrm{T}}.$$

check shape : $(k \times n) \times (n \times (p+1)) = (k \times (p+1))$-matrix

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{w}}(L) = \frac{1}{n}\begin{bmatrix} \hat{p}_{11} - I\left(y^{(1)} = 1\right) & \hat{p}_{12} - I\left(y^{(2)} = 1\right) & \dots & \hat{p}_{1n} - I\left(y^{(n)} = 1\right) \\ \vdots & \vdots & \vdots & \\ \hat{p}_{j1} - I\left(y^{(1)} = j\right) & \hat{p}_{j2} - I\left(y^{(2)} = j\right) & \dots & \hat{p}_{jn} - I\left(y^{(n)} = j\right) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{p}_{k1} - I\left(y^{(1)} = k\right) & \hat{p}_{k2} - I\left(y^{(2)} = k\right) & \dots & \hat{p}_{kn} - I\left(y^{(n)} = k\right) \end{bmatrix}\begin{bmatrix} {\mathbf{x}^{(1)}}^{\mathrm{T}} \\ {\mathbf{x}^{(2)}}^{\mathrm{T}} \\ \vdots \\ \vdots \\ {\mathbf{x}^{(n)}}^{\mathrm{T}} \end{bmatrix}$$

$$= \frac{1}{n}\mathbf{P}_{\text{adjusted}}\mathbf{X}^{\mathrm{T}}.$$

Gradient descent iteration for softmax:

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{W}}(L) = \frac{1}{n}\begin{bmatrix} \hat{p}_{11} - I\left(y^{(1)} = 1\right) & \hat{p}_{12} - I\left(y^{(2)} = 1\right) & \dots & \hat{p}_{1n} - I\left(y^{(n)} = 1\right) \\ \vdots & \vdots & \vdots & \\ \hat{p}_{j1} - I\left(y^{(1)} = j\right) & \hat{p}_{j2} - I\left(y^{(2)} = j\right) & \dots & \hat{p}_{jn} - I\left(y^{(n)} = j\right) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{p}_{k1} - I\left(y^{(1)} = k\right) & \hat{p}_{k2} - I\left(y^{(2)} = k\right) & \dots & \hat{p}_{kn} - I\left(y^{(n)} = k\right) \end{bmatrix} \begin{bmatrix} \mathbf{x}^{(1)\,\mathrm{T}} \\ \mathbf{x}^{(2)\,\mathrm{T}} \\ \vdots \\ \vdots \\ \mathbf{x}^{(n)\,\mathrm{T}} \end{bmatrix}$$

$$= \frac{1}{n}\mathbf{P}_{\text{adjusted}}\mathbf{X}^{\mathrm{T}}.$$

Gradient descent iteration for softmax:

$$\mathbf{W} = \mathbf{W} - \alpha \times \nabla_{\mathbf{W}}(L)|_{\mathbf{W}}$$

Softmax Classifier Gradient – continued

$$\Rightarrow \nabla_{\mathbf{W}}(L) = \frac{1}{n}\begin{bmatrix} \hat{p}_{11} - I\left(y^{(1)} = 1\right) & \hat{p}_{12} - I\left(y^{(2)} = 1\right) & \dots & \hat{p}_{1n} - I\left(y^{(n)} = 1\right) \\ \vdots & \vdots & \vdots & \\ \hat{p}_{j1} - I\left(y^{(1)} = j\right) & \hat{p}_{j2} - I\left(y^{(2)} = j\right) & \dots & \hat{p}_{jn} - I\left(y^{(n)} = j\right) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{p}_{k1} - I\left(y^{(1)} = k\right) & \hat{p}_{k2} - I\left(y^{(2)} = k\right) & \dots & \hat{p}_{kn} - I\left(y^{(n)} = k\right) \end{bmatrix} \begin{bmatrix} {\mathbf{x}^{(1)}}^{\mathrm{T}} \\ {\mathbf{x}^{(2)}}^{\mathrm{T}} \\ \vdots \\ \vdots \\ {\mathbf{x}^{(n)}}^{\mathrm{T}} \end{bmatrix}$$

$$= \frac{1}{n}\mathbf{P}_{\text{adjusted}}\mathbf{X}^{\mathrm{T}}.$$

Gradient descent iteration for softmax:

$$\mathbf{W} = \mathbf{W} - \alpha \times \nabla_{\mathbf{W}}(L)|_{\mathbf{W}} = \mathbf{W} - \alpha \times \frac{1}{n}\mathbf{P}_{\text{adjusted}}\mathbf{X}^{\mathrm{T}}.$$

Logistic Regression Classifier Setup

Logistic Regression Classifier Setup

- A classifier for binary classification:

Logistic Regression Classifier Setup

- A classifier for binary classification: two output classes labeled as 0 and 1.

Logistic Regression Classifier Setup

- A classifier for binary classification: two output classes labeled as 0 and 1.
- Just like the softmax classifier, we calculate raw scores for each sample.

Logistic Regression Classifier Setup

- A classifier for binary classification: two output classes labeled as 0 and 1.
- Just like the softmax classifier, we calculate raw scores for each sample.
- The raw score for the ith sample is $\mathbf{z}^{(i)} = \mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}$ (assuming bias trick):

Logistic Regression Classifier Setup

- A classifier for binary classification: two output classes labeled as 0 and 1.
- Just like the softmax classifier, we calculate raw scores for each sample.
- The raw score for the ith sample is $\mathbf{z}^{(i)} = \mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}$ (assuming bias trick): note that the weight matrix $\mathbf{W}$ has only one row $\mathbf{w}^{\mathrm{T}}$.

Logistic Regression Classifier Setup

- A classifier for binary classification: two output classes labeled as 0 and 1.
- Just like the softmax classifier, we calculate raw scores for each sample.
- The raw score for the ith sample is $\mathbf{z}^{(i)} = \mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}$ (assuming bias trick): note that the weight matrix $\mathbf{W}$ has only one row $\mathbf{w}^{\mathrm{T}}$.
- The raw score $\mathbf{z}^{(i)}$ is used to calculate the predicted probability that the ith sample belongs to its correct class, which in turn is used to define the loss for the ith sample.

Logistic Regression Classifier Setup

- A classifier for binary classification: two output classes labeled as 0 and 1.
- Just like the softmax classifier, we calculate raw scores for each sample.
- The raw score for the ith sample is $\mathbf{z}^{(i)} = \mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}$ (assuming bias trick): note that the weight matrix $\mathbf{W}$ has only one row $\mathbf{w}^{\mathrm{T}}$.
- The raw score $\mathbf{z}^{(i)}$ is used to calculate the predicted probability that the ith sample belongs to its correct class, which in turn is used to define the loss for the ith sample.
- But how do we get the predicted probability that a sample belongs to its correct class?

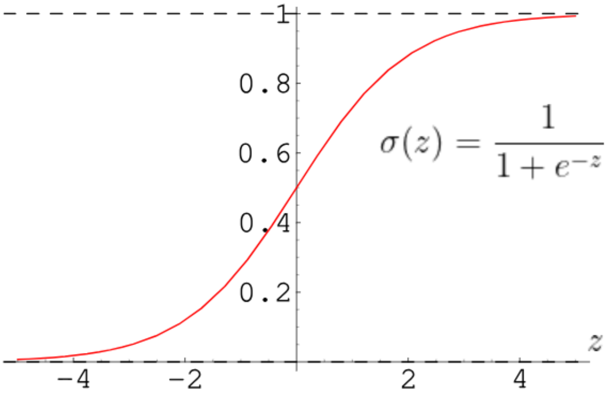
Logistic Regression Classifier Setup – continued

Logistic Regression Classifier Setup – continued

The sigmoid function $\sigma(z)$:

Logistic Regression Classifier Setup – continued

The sigmoid function $\sigma(z)$:



Logistic Regression Classifier Setup – continued

Logistic Regression Classifier Setup – continued

- The predicted probability that the ith sample belongs to its correct class $y^{(i)}$ is denoted as $\hat{y}^{(i)}$.

Logistic Regression Classifier Setup – continued

- The predicted probability that the ith sample belongs to its correct class $y^{(i)}$ is denoted as $\hat{y}^{(i)}$.
- The sigmoid function applied to the raw score $\mathbf{z}^{(i)}$ is used to define the predicted probability:

Logistic Regression Classifier Setup – continued

- The predicted probability that the ith sample belongs to its correct class $y^{(i)}$ is denoted as $\hat{y}^{(i)}$.
- The sigmoid function applied to the raw score $\mathbf{z}^{(i)}$ is used to define the predicted probability:

$$\hat{y}^{(i)} = \begin{cases} \sigma\left(\mathbf{z}^{(i)}\right) = \sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right) & \text{if correct class } y^{(i)} = 1, \\ 1 - \sigma\left(\mathbf{z}^{(i)}\right) = 1 - \sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right) & \text{if correct class } y^{(i)} = 0 \end{cases}$$

Logistic Regression Classifier Setup – continued

- The predicted probability that the ith sample belongs to its correct class $y^{(i)}$ is denoted as $\hat{y}^{(i)}$.
- The sigmoid function applied to the raw score $\mathbf{z}^{(i)}$ is used to define the predicted probability:

$$\hat{y}^{(i)} = \begin{cases} \sigma\left(\mathbf{z}^{(i)}\right) = \sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right) & \text{if correct class } y^{(i)} = 1, \\ 1 - \sigma\left(\mathbf{z}^{(i)}\right) = 1 - \sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right) & \text{if correct class } y^{(i)} = 0 \end{cases}$$

- Note that we can write this compactly as

$$\hat{y}^{(i)} = \left(\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{y^{(i)}} \left(1 - \sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{1-y^{(i)}}.$$

Logistic Regression Classifier Loss Function and Gradient

Logistic Regression Classifier Loss Function and Gradient

- Just like the softmax classifier, the loss for the logistic regression classifier for the ith sample is

Logistic Regression Classifier Loss Function and Gradient

- Just like the softmax classifier, the loss for the logistic regression classifier for the ith sample is the negative log of the predicted probability that the ith sample belongs to it correct class $y^{(i)} \Rightarrow$

Logistic Regression Classifier Loss Function and Gradient

- Just like the softmax classifier, the loss for the logistic regression classifier for the ith sample is the negative log of the predicted probability that the ith sample belongs to it correct class $y^{(i)} \Rightarrow$

$$L_i(\mathbf{w}) = -\log\left(\hat{y}^{(i)}\right) = -\log\left(\left(\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{y^{(i)}}\left(1-\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{1-y^{(i)}}\right).$$

Logistic Regression Classifier Loss Function and Gradient

- Just like the softmax classifier, the loss for the logistic regression classifier for the ith sample is the negative log of the predicted probability that the ith sample belongs to it correct class $y^{(i)} \Rightarrow$

$$L_i(\mathbf{w}) = -\log\left(\hat{y}^{(i)}\right) = -\log\left(\left(\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{y^{(i)}}\left(1-\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{1-y^{(i)}}\right).$$

- The average training loss $L(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{w})$ has the gradient

Logistic Regression Classifier Loss Function and Gradient

- Just like the softmax classifier, the loss for the logistic regression classifier for the ith sample is the negative log of the predicted probability that the ith sample belongs to it correct class $y^{(i)} \Rightarrow$

$$L_i(\mathbf{w}) = -\log\left(\hat{y}^{(i)}\right) = -\log\left(\left(\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{y^{(i)}}\left(1-\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{1-y^{(i)}}\right).$$

- The average training loss $L(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{w})$ has the gradient

$$\nabla_{\mathbf{w}}(L) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\mathbf{w}}(L_i) = \frac{1}{n}\mathbf{X}\left[\sigma\left(\mathbf{X}^{\mathrm{T}}\mathbf{w}\right) - \mathbf{y}\right].$$

Logistic Regression Classifier Loss Function and Gradient

- Just like the softmax classifier, the loss for the logistic regression classifier for the ith sample is the negative log of the predicted probability that the ith sample belongs to it correct class $y^{(i)} \Rightarrow$

$$L_i(\mathbf{w}) = -\log\left(\hat{y}^{(i)}\right) = -\log\left(\left(\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{y^{(i)}}\left(1-\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{1-y^{(i)}}\right).$$

- The average training loss $L(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{w})$ has the gradient

$$\nabla_{\mathbf{w}}(L) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\mathbf{w}}(L_i) = \frac{1}{n}\mathbf{X}\left[\sigma\left(\mathbf{X}^{\mathrm{T}}\mathbf{w}\right)-\mathbf{y}\right].$$

sigmoid broadcasted to all elements of vector

Logistic Regression Classifier Loss Function and Gradient

- Just like the softmax classifier, the loss for the logistic regression classifier for the ith sample is the negative log of the predicted probability that the ith sample belongs to it correct class $y^{(i)} \Rightarrow$

$$L_i(\mathbf{w}) = -\log\left(\hat{y}^{(i)}\right) = -\log\left(\left(\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{y^{(i)}}\left(1-\sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right)\right)^{1-y^{(i)}}\right).$$

- The average training loss $L(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{w})$ has the gradient

$$\nabla_{\mathbf{w}}(L) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\mathbf{w}}(L_i) = \frac{1}{n}\mathbf{X}\left[\sigma\left(\mathbf{X}^{\mathrm{T}}\mathbf{w}\right) - \mathbf{y}\right].$$

vector of correct classes

Logistic Regression Classifier Loss Function and Gradient – continued

Logistic Regression Classifier Loss Function and Gradient – continued

Following intermediate variables and computation graph can be used to derive the ith samples logistic regression loss function's gradient $\nabla_{\mathbf{w}}(L_i)$:

Logistic Regression Classifier Loss Function and Gradient – continued

Following intermediate variables and computation graph can be used to derive the ith samples logistic regression loss function's gradient $\nabla_{\mathbf{w}}(L_i)$:

$$L_i = -\log\left(\left(a^{(i)}\right)^{y^{(i)}} \times \left(1 - a^{(i)}\right)^{1-y^{(i)}}\right),$$

Logistic Regression Classifier Loss Function and Gradient – continued

Following intermediate variables and computation graph can be used to derive the ith samples logistic regression loss function's gradient $\nabla_{\mathbf{w}}(L_i)$:

$$L_i = -\log\left(\left(a^{(i)}\right)^{y^{(i)}} \times \left(1 - a^{(i)}\right)^{1-y^{(i)}}\right),$$

$$a^{(i)} = \sigma\left(\mathbf{z}^{(i)}\right),$$

Logistic Regression Classifier Loss Function and Gradient – continued

Following intermediate variables and computation graph can be used to derive the ith samples logistic regression loss function's gradient $\nabla_{\mathbf{w}}(L_i)$:

$$
\begin{aligned}
L_i &= -\log\left(\left(a^{(i)}\right)^{y^{(i)}} \times \left(1 - a^{(i)}\right)^{1-y^{(i)}}\right), \\
a^{(i)} &= \sigma\left(\mathbf{z}^{(i)}\right), \\
\mathbf{z}^{(i)} &= \mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}.
\end{aligned}
$$

Logistic Regression Classifier Loss Function and Gradient – continued

Following intermediate variables and computation graph can be used to derive the ith samples logistic regression loss function's gradient $\nabla_{\mathbf{w}}(L_i)$:

$$L_i = -\log\left(\left(a^{(i)}\right)^{y^{(i)}} \times \left(1 - a^{(i)}\right)^{1-y^{(i)}}\right),$$

$$a^{(i)} = \sigma\left(\mathbf{z}^{(i)}\right),$$

$$\mathbf{z}^{(i)} = \mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}.$$

$$\begin{array}{c} \boxed{L_i} \\ \downarrow \\ \boxed{a^{(i)}} \\ \downarrow \\ \boxed{\mathbf{z}^{(i)}} \\ \downarrow \\ \boxed{\mathbf{w}} \end{array}$$

Why Regularization?

Why Regularization?

- Regularization is an approach to prevent the model from doing too well on training data.

Why Regularization?

- Regularization is an approach to prevent the model from doing too well on training data.
- Models that overfit the training data typically perform poorly on unseen test data.

Why Regularization?

- Regularization is an approach to prevent the model from doing too well on training data.
- Models that overfit the training data typically perform poorly on unseen test data.
- Model overfitting happens when specific features of the training samples drive the learning process and/or the model learns the noise in the data.

Why Regularization?

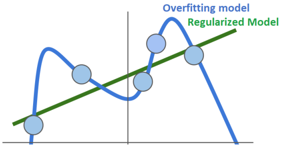
- Regularization is an approach to prevent the model from doing too well on training data.
- Models that overfit the training data typically perform poorly on unseen test data.
- Model overfitting happens when specific features of the training samples drive the learning process and/or the model learns the noise in the data.
- Regularization is particularly helpful for high-dimensional datasets (more features than samples).

Why Regularization?

- Regularization is an approach to prevent the model from doing too well on training data.
- Models that overfit the training data typically perform poorly on unseen test data.
- Model overfitting happens when specific features of the training samples drive the learning process and/or the model learns the noise in the data.
- Regularization is particularly helpful for high-dimensional datasets (more features than samples).

Potentially overfit feature

Oil	Density	Crispy	Fracture	Hardness	Taste
16.5	2955	10	23	97	fair
17.7	2660	14	9	139	excellent
16.2	2870	12	17	143	poor
16.7	2920	10	31	95	good
16.3	2975	11	26	143	fair
19.1	2790	13	16	189	good
18.4	2750	13	17	114	poor



Regularization Approaches

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}}$$

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} + \left\{ \right.$$

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} \quad + \begin{cases} \lambda \sum_{r,s} |w_{rs}|^2 \\ (\text{ridge or } L_2) \end{cases}$$

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} + \begin{cases} \lambda \sum_{r,s} |w_{rs}|^2 \\ (\text{ridge or } L_2) \\ \quad \text{or} \\ \\ \\ \\ \end{cases}$$

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} + \begin{cases} \lambda \sum_{r,s} |w_{rs}|^2 \\ (\text{ridge or } L_2) \\ \quad \text{or} \\ \lambda \sum_{r,s} |w_{rs}| \\ (\text{lasso or } L_1) \end{cases}$$

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} + \begin{cases} \lambda \sum_{r,s} |w_{rs}|^2 \\ (\text{ridge or } L_2) \\ \quad \text{or} \\ \lambda \sum_{r,s} |w_{rs}| \\ (\text{lasso or } L_1) \\ \quad \text{or} \\ \end{cases}$$

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} + \begin{cases} \lambda \sum_{r,s} |w_{rs}|^2 \\ (\text{ridge or } L_2) \\ \quad\text{or} \\ \lambda \sum_{r,s} |w_{rs}| \\ (\text{lasso or } L_1) \\ \quad\text{or} \\ \lambda \left[\alpha \sum_{r,s} |w_{rs}| + (1-\alpha) \sum_{r,s} |w_{rs}|^2\right] \\ (\text{elastic-net}), \end{cases}$$

Regularization Approaches

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} + \begin{cases} \lambda \sum_{r,s} |w_{rs}|^2 \\ (\text{ridge or } L_2) \\ \quad \text{or} \\ \lambda \sum_{r,s} |w_{rs}| \\ (\text{lasso or } L_1) \\ \quad \text{or} \\ \lambda \left[\alpha \sum_{r,s} |w_{rs}| + (1-\alpha) \sum_{r,s} |w_{rs}|^2\right] \\ (\text{elastic-net}), \end{cases}$$

where $\lambda > 0$ is the strength of regularization.

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} + \begin{cases} \lambda \sum_{r,s} |w_{rs}|^2 \\ (\text{ridge or } L_2) \\ \quad\text{or} \\ \lambda \sum_{r,s} |w_{rs}| \\ (\text{lasso or } L_1) \\ \quad\text{or} \\ \lambda \left[\alpha \sum_{r,s} |w_{rs}| + (1-\alpha) \sum_{r,s} |w_{rs}|^2\right] \\ (\text{elastic-net}), \end{cases}$$

where $\lambda > 0$ is the strength of regularization.

- λ is a hyperparameter that has to be tuned.

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} + \begin{cases} \lambda \sum_{r,s} |w_{rs}|^2 \\ (\text{ridge or } L_2) \\ \quad\text{or} \\ \lambda \sum_{r,s} |w_{rs}| \\ (\text{lasso or } L_1) \\ \quad\text{or} \\ \lambda \left[\alpha \sum_{r,s} |w_{rs}| + (1-\alpha) \sum_{r,s} |w_{rs}|^2\right] \\ (\text{elastic-net}), \end{cases}$$

- λ is a hyperparameter that has to be tuned.
- All regularization approaches tend to drive the weight values close to zero.

where $\lambda > 0$ is the strength of regularization.

Regularization Approaches

Regularization is achieved by
adding to the average training data loss constraints on the weights (not to the bias values):

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} L_i(\mathbf{W})}_{\text{average training data loss}} + \begin{cases} \lambda \sum_{r,s} |w_{rs}|^2 \\ (\text{ridge or } L_2) \\ \quad \text{or} \\ \lambda \sum_{r,s} |w_{rs}| \\ (\text{lasso or } L_1) \\ \quad \text{or} \\ \lambda \left[\alpha \sum_{r,s} |w_{rs}| + (1-\alpha) \sum_{r,s} |w_{rs}|^2\right] \\ (\text{elastic-net}), \end{cases}$$

where $\lambda > 0$ is the strength of regularization.

- λ is a hyperparameter that has to be tuned.
- All regularization approaches tend to drive the weight values close to zero.
- L_1-regularization typically results in a smaller subset of nonzero weights than L_2-regularization.

Regularization Loss & Gradient

MANIPAL ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

Regularization Loss & Gradient

Consider the L_2-regularization term (also the regularization loss) for a weights matrix corresponding to k output classes and p features:

Regularization Loss & Gradient

Consider the L_2-regularization term (also the regularization loss) for a weights matrix corresponding to k output classes and p features:

$$L_{\text{reg}}(\mathbf{W}) = \sum_{r,s} |w_{rs}|^2 =$$

Regularization Loss & Gradient

Consider the L_2-regularization term (also the regularization loss) for a weights matrix corresponding to k output classes and p features:

$$L_{\text{reg}}(\mathbf{W}) = \sum_{r,s} |w_{rs}|^2 = \|\mathbf{w}_1\|^2 + \|\mathbf{w}_2\|^2 + \cdots + \|\mathbf{w}_k\|^2$$

Regularization Loss & Gradient

Consider the L_2-regularization term (also the regularization loss) for a weights matrix corresponding to k output classes and p features:

$$
\begin{aligned}
L_{\text{reg}}(\mathbf{W}) &= \sum_{r,s} |w_{rs}|^2 = \|\mathbf{w}_1\|^2 + \|\mathbf{w}_2\|^2 + \cdots + \|\mathbf{w}_k\|^2 \\
&= \mathbf{w}_1^{\mathrm{T}}\mathbf{w}_1 + \mathbf{w}_2^{\mathrm{T}}\mathbf{w}_2 + \cdots + \mathbf{w}_k^{\mathrm{T}}\mathbf{w}_k.
\end{aligned}
$$

Regularization Loss & Gradient

Consider the L_2-regularization term (also the regularization loss) for a weights matrix corresponding to k output classes and p features:

$$L_{\text{reg}}(\mathbf{W}) = \sum_{r,s} |w_{rs}|^2 = \|\mathbf{w}_1\|^2 + \|\mathbf{w}_2\|^2 + \cdots + \|\mathbf{w}_k\|^2$$
$$= \mathbf{w}_1^{\mathrm{T}}\mathbf{w}_1 + \mathbf{w}_2^{\mathrm{T}}\mathbf{w}_2 + \cdots + \mathbf{w}_k^{\mathrm{T}}\mathbf{w}_k.$$

Note that input shape is $k \times p$ and output shape is 1

Regularization Loss & Gradient

Consider the L_2-regularization term (also the regularization loss) for a weights matrix corresponding to k output classes and p features:

$$
\begin{aligned}
L_{\text{reg}}(\mathbf{W}) &= \sum_{r,s} |w_{rs}|^2 = \|\mathbf{w}_1\|^2 + \|\mathbf{w}_2\|^2 + \cdots + \|\mathbf{w}_k\|^2 \\
&= \mathbf{w}_1^{\mathrm{T}}\mathbf{w}_1 + \mathbf{w}_2^{\mathrm{T}}\mathbf{w}_2 + \cdots + \mathbf{w}_k^{\mathrm{T}}\mathbf{w}_k.
\end{aligned}
$$

Note that input shape is $k \times p$ and output shape is 1 $\Rightarrow$ gradient shape is $k \times p$.

Regularization Loss & Gradient

Consider the L_2-regularization term (also the regularization loss) for a weights matrix corresponding to k output classes and p features:

$$
\begin{aligned}
L_{\text{reg}}(\mathbf{W}) &= \sum_{r,s} |w_{rs}|^2 = \|\mathbf{w}_1\|^2 + \|\mathbf{w}_2\|^2 + \cdots + \|\mathbf{w}_k\|^2 \\
&= \mathbf{w}_1^{\mathrm{T}}\mathbf{w}_1 + \mathbf{w}_2^{\mathrm{T}}\mathbf{w}_2 + \cdots + \mathbf{w}_k^{\mathrm{T}}\mathbf{w}_k.
\end{aligned}
$$

Note that input shape is $k \times p$ and output shape is 1 $\Rightarrow$ gradient shape is $k \times p$.

$$
\nabla_{\mathbf{W}} \left(L_{\text{reg}} \right) =
$$

Regularization Loss & Gradient

Consider the L_2-regularization term (also the regularization loss) for a weights matrix corresponding to k output classes and p features:

$$L_{\text{reg}}(\mathbf{W}) = \sum_{r,s} |w_{rs}|^2 = \|\mathbf{w}_1\|^2 + \|\mathbf{w}_2\|^2 + \cdots + \|\mathbf{w}_k\|^2$$
$$= \mathbf{w}_1^{\mathrm{T}}\mathbf{w}_1 + \mathbf{w}_2^{\mathrm{T}}\mathbf{w}_2 + \cdots + \mathbf{w}_k^{\mathrm{T}}\mathbf{w}_k.$$

Note that input shape is $k \times p$ and output shape is 1 $\Rightarrow$ gradient shape is $k \times p$.

$$\nabla_{\mathbf{W}}\left(L_{\text{reg}}\right) = \begin{bmatrix} \left(\nabla_{\mathbf{w}_1}\left(L_{\text{reg}}\right)\right)^{\mathrm{T}} \\ \left(\nabla_{\mathbf{w}_2}\left(L_{\text{reg}}\right)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_k}\left(L_{\text{reg}}\right)\right)^{\mathrm{T}} \end{bmatrix}$$

Regularization Loss & Gradient

Consider the L_2-regularization term (also the regularization loss) for a weights matrix corresponding to k output classes and p features:

$$
\begin{aligned}
L_{\text{reg}}(\mathbf{W}) &= \sum_{r,s} |w_{rs}|^2 = \|\mathbf{w}_1\|^2 + \|\mathbf{w}_2\|^2 + \cdots + \|\mathbf{w}_k\|^2 \\
&= \mathbf{w}_1^{\mathrm{T}}\mathbf{w}_1 + \mathbf{w}_2^{\mathrm{T}}\mathbf{w}_2 + \cdots + \mathbf{w}_k^{\mathrm{T}}\mathbf{w}_k.
\end{aligned}
$$

Note that input shape is $k \times p$ and output shape is 1 $\Rightarrow$ gradient shape is $k \times p$.

$$
\nabla_{\mathbf{W}}\left(L_{\text{reg}}\right) = \begin{bmatrix} \left(\nabla_{\mathbf{w}_1}\left(L_{\text{reg}}\right)\right)^{\mathrm{T}} \\ \left(\nabla_{\mathbf{w}_2}\left(L_{\text{reg}}\right)\right)^{\mathrm{T}} \\ \vdots \\ \left(\nabla_{\mathbf{w}_k}\left(L_{\text{reg}}\right)\right)^{\mathrm{T}} \end{bmatrix} = 2\begin{bmatrix} \mathbf{w}_1^{\mathrm{T}} \\ \mathbf{w}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{w}_k^{\mathrm{T}} \end{bmatrix} = 2\mathbf{W}.
$$

Batch Processing

Batch Processing

- The average training data loss $\frac{1}{n}\sum_{i=1}^{n} L_i$ has contribution from all the training samples.

Batch Processing

- The average training data loss $\frac{1}{n}\sum_{i=1}^{n} L_i$ has contribution from all the training samples.
- In some cases, such as the training samples being images, it could be computationally expensive the calculate the loss for all training samples and then the average training loss.

Batch Processing

- The average training data loss $\frac{1}{n}\sum_{i=1}^{n} L_i$ has contribution from all the training samples.
- In some cases, such as the training samples being images, it could be computationally expensive the calculate the loss for all training samples and then the average training loss.
- The work around is called batch processing where the training samples are randomly shuffled and split into batches of a specific batch size (typically $16, 32, \ldots$) and then the weights are trained using the smaller set of samples in the training batches.

Batch Processing

- The average training data loss $\frac{1}{n}\sum_{i=1}^{n} L_i$ has contribution from all the training samples.
- In some cases, such as the training samples being images, it could be computationally expensive the calculate the loss for all training samples and then the average training loss.
- The work around is called batch processing where the training samples are randomly shuffled and split into batches of a specific batch size (typically $16, 32, \ldots$) and then the weights are trained using the smaller set of samples in the training batches.
- The weights will be updated more frequently (and initially inaccurately) using the gradient descent method because of the small number of batch training samples.

Batch Processing

- The average training data loss $\frac{1}{n}\sum_{i=1}^{n} L_i$ has contribution from all the training samples.
- In some cases, such as the training samples being images, it could be computationally expensive the calculate the loss for all training samples and then the average training loss.
- The work around is called batch processing where the training samples are randomly shuffled and split into batches of a specific batch size (typically $16, 32, \ldots$) and then the weights are trained using the smaller set of samples in the training batches.
- The weights will be updated more frequently (and initially inaccurately) using the gradient descent method because of the small number of batch training samples.
- An epoch is when the weights have been updated using all the training samples through batches;

Batch Processing

- The average training data loss $\frac{1}{n}\sum_{i=1}^{n} L_i$ has contribution from all the training samples.
- In some cases, such as the training samples being images, it could be computationally expensive the calculate the loss for all training samples and then the average training loss.
- The work around is called batch processing where the training samples are randomly shuffled and split into batches of a specific batch size (typically $16, 32, \ldots$) and then the weights are trained using the smaller set of samples in the training batches.
- The weights will be updated more frequently (and initially inaccurately) using the gradient descent method because of the small number of batch training samples.
- An epoch is when the weights have been updated using all the training samples through batches; that is, the model has seen all the training samples once.