COMP 432 Machine Learning

Linear Models

Computer Science & Software Engineering
Concordia University, Fall 2024

Summary of the last episode....

- Capacity, Underfitting, Overfitting, Regularization
- Gradient Descent (and variants)
- Linear Least Square
- Hyperparameters and Validation set
- Gradient Descent Variants

What we are going to learn today:

- Linear models for regression
- Linear models for classification

Linear Models for Regression

Linear Regression

- A linear model can be written as:

$$\boxed{\hat{y}(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} = w_0 + \sum_{j=1}^{D} w_j x_j}$$

$$\mathbf{x} = [1, x_1, x_2, \ldots, x_D]^T$$ ➡ Input Features

Concatenating a 1 is needed to implement the bias term.

$$\mathbf{w} = [w_0, w_1, \ldots, w_D]^T$$ ➡ Parameters

- The output is a **weighted sum** of the inputs.

Linear Regression

$$\hat{y}(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} = w_0 + \sum_{j=1}^{D} w_j x_j$$

- To train the model, we need:

1. To define an **objective function**.

2. Compute the **gradient** of the objective wrt the parameters.

Linear Regression

- The natural objective function for a regression problem is the **Mean Squared Error** (MSE):

$$\underbrace{J(\mathbf{w})}_{\text{Scalar}} = \frac{1}{2}\sum_{i=1}^{\overbrace{N}^{\text{Number of training samples}}} \Big(\underbrace{y_i}_{\text{Label}} - \underbrace{\hat{y}_i(\mathbf{x}_i, \mathbf{w})}_{\text{Prediction}}\Big)^2$$

Let's assume our model provides the following prediction of the i-th input:

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$

- In this case, we can write the objective as:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - w_0 - w_1 x_1 - w_2 x_2)^2$$

Linear Regression

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \underbrace{w_0 - w_1 x_1 - w_2 x_2}_{\hat{y}_i})^2$$

We now compute the gradient:

$$\nabla J(\mathbf{w}) = \left[\boxed{\frac{\partial J}{w0}}, \boxed{\frac{\partial J}{w1}}, \boxed{\frac{\partial J}{w2}}\right]^T$$

$$\boxed{\frac{\partial J}{\partial w_0}} = 2 \cdot \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i) \cdot (-1)$$

$$= \boxed{\sum_{i=1}^{N}(\hat{y}_i - y_i)}$$

$$\boxed{\frac{\partial J}{w1}} = 2 \cdot \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i) \cdot (-x_1)$$

$$= \boxed{\sum_{i=1}^{N}(\hat{y}_i - y_i) \cdot (-x_1)}$$

$$\boxed{\frac{\partial J}{w2} = \sum_{i=1}^{N}(\hat{y}_i - y_i) \cdot (-x_2)}$$

Linear Regression

- In the multidimensional case, we can generalize the gradient as:

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_0} \\ \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \dots \\ \frac{\partial J(\mathbf{w})}{\partial w_D} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i) \\ \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)x_{i1} \\ \dots \\ \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)x_{iD} \end{bmatrix} = \boxed{\sum_{i=1}^{N}(\underbrace{\underbrace{\mathbf{w}^T\mathbf{x}_i}_{\text{Prediction}} - \underbrace{y_i}_{\text{Label}}}_{\text{Error}})\mathbf{x}_i}$$

$$\mathbf{x}_i = [1, x_0, \dots, x_D]^T \quad \mathbf{w} = [w_0, w_1, \dots, w_D]^T$$

- If we set:

$$\mathbf{x} = [1, x_1, x_2]^T \quad \mathbf{w} = [w0, w_1, w_2]^T$$

$\Rightarrow$ We obtain the equations seen for the 2d case

Linear Regression

- We can also write the gradient equations in **matrix form**:

$$\nabla J(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)\mathbf{x}_i = \boxed{\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})}$$

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,D} \\ 1 & x_{2,1} & \dots & x_{2,D} \\ 1 & \dots & \dots & \dots \\ 1 & x_{N,1} & \dots & x_{N,D} \end{bmatrix}$$

Data Matrix (all samples)

$$\mathbf{w} = [w_0, w_1, \dots, w_D]^T$$

$$\mathbf{y} = [y_1, y_2, \dots, y_N]^T$$

Label Vector (all samples)

Linear Regression

- Let's do a **sanity check** on the dimensionalities to convince us better that the two expressions are equivalent.

$$\nabla J(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)\mathbf{x}_i = \boxed{\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})}$$

- Let's assume we have 3 parameters w_0, w_1, and w_2
- *What is the expected dimensionality for the gradient?* (3,1)
- *What is the dimensionality of **w**?* (3,1)
- *Let's assume we have 4 examples. What will be the dimensionality of the feature matrix **X**?* (4,3)
- *What is the dimensionality of **y**?* (4,1)

Linear Regression

- Let's do a **sanity check** on the dimensionalities to convince us better that the two expressions are equivalent.

$$\nabla J(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T\mathbf{x}_i - y_i)\mathbf{x}_i = \boxed{\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})}$$

(4,3) (3,1)

(4,1) (4,1)

(3,4) (4,1)

(3,1)

Why do we want expressions that contain matrices?

- For the sake of compactness.
- We can take advantage of fast matrix multiplication libraries.

This is the expected dimensionality for the gradient.

Linear Regression

- We now have a compact expression of the gradient:

$$\nabla J(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)\mathbf{x}_i = \boxed{\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})}$$

- We can use it for optimizing the parameters with **gradient descent**:

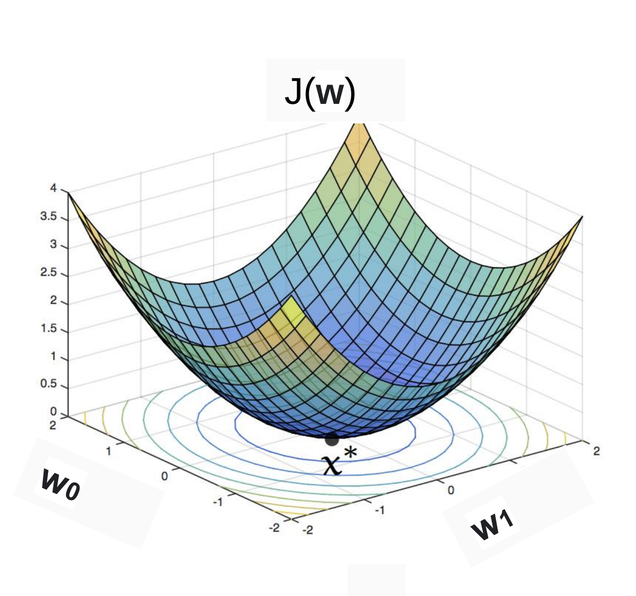
$$\mathbf{w}_{new} = \mathbf{w} - \eta \nabla J(\mathbf{w})$$

- This model is very simple though. We can see if a direct (analytical solution exists).

Linear Regression

- Let’s take a look at the parameter space that we are optimizing.

- If we consider this simple linear model coupled with the **mean squared error**, the **parameter space** is **convex**!



- The function has only **one minimum** which is the **global** one.

- Instead of using gradient descent, we can see if an **analytical solution** exists!

Linear Regression

- This is the expression for the gradient.

$$\nabla J(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x}_i - y_i)\mathbf{x}_i = \mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y})$$

- We can look for an analytical solution by solving:

$$\nabla J(\mathbf{w}) = 0$$

Linear Regression

$$\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

$$(\mathbf{X}^T\mathbf{X})\mathbf{w} - \mathbf{X}^T\mathbf{y} = 0$$ (Expansion)

$$(\mathbf{X}^T\mathbf{X})\mathbf{w} = \mathbf{X}^T\mathbf{y}$$ (Manipulation)

$$(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X})\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$ (Manipulation)

$$\boxed{\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$ (Direct solution)

Linear Regression

This problem is simple enough and can be solved in a **closed form** with an **analytical expression**:

$$\boxed{\mathbf{w} = \underbrace{(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T}_{\text{Moore-Penrose Pseudoinverse}}\mathbf{y}}$$

(Direct solution)

Moore-Penrose Pseudoinverse → Generalization of the notion of "inverse" matrix to non-square matrices

- We can solve this simple problem in **one shot** just by solving a system of linear equations.
- Even though gradient descent works well in this case too, we do not necessarily need to use it as an analytical solution exists.

Linear Regression

- Let's do a dimensionality check:

$$\boxed{\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$$ (Direct solution)

$$\underbrace{\underbrace{\underbrace{\underbrace{\mathbf{X}^T}_{(3,4)}\ \underbrace{\mathbf{X}}_{(4,3)}}_{(3,3)}\ \underbrace{\mathbf{X}^T}_{(3,4)}}_{(3,4)}\ \underbrace{\mathbf{y}}_{(4,1)}}_{(3,1)}$$

This is the expected dimensionality for the parameter vector **w**. → (3,1)

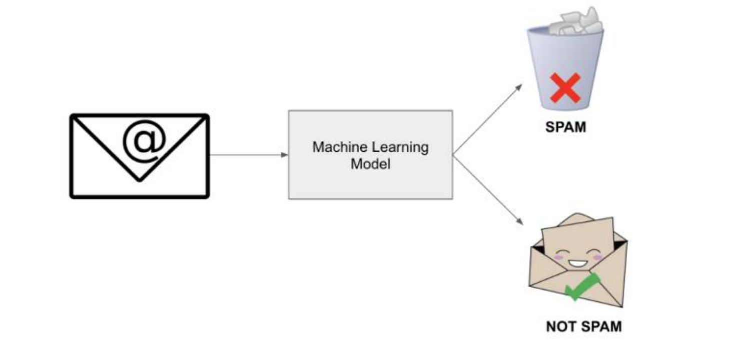
Logistic Regression

Binary Classification

- So far, we have seen linear models for **regression**. But, what about **classification**?

Classification: the model has to specify which of the *k* categories some input belongs to.

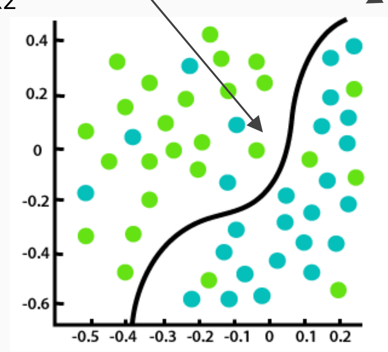
$$\boxed{\hat{y} = f(\mathbf{x})} \quad \mathbf{x} = [x_1, x_2, .., x_D]^T \quad \mathbf{x} \in \mathbb{R}^D \quad \hat{y} = \{1, ..., K\}$$



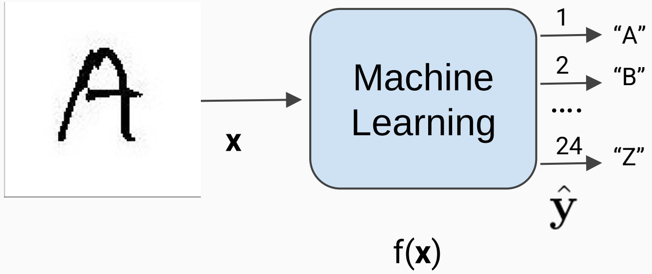
Decision boundaries

f(**x**)

X2



X1



Binary Classification

- If we only have two classes (e.g., *cat/dog*) we have a **binary classification** problem.

$$y_i = \{0, 1\} \quad \hat{y}_i = \{0, 1\}$$

“Cat” “Dog”

How many outputs are needed for a binary classification?

Input

Machine Learning

0/1

$\mathbf{x}_i$ $f(\mathbf{x})$

One output is enough!

If the output is “zero”, we predict class 0

If the output is “one”, we predict class 1

Binary Classification

- Often, we want the model to output a **probability** (soft prediction) rather than a discrete one (hard prediction),

Input

$\mathbf{x}_i$

Machine Learning

f(**x**)

output

$P(y_i = 1 \mid \mathbf{x}_i)$

0.1

>0.5 $\hat{y}_i = 1$

Else $\hat{y}_i = 0$

Cat

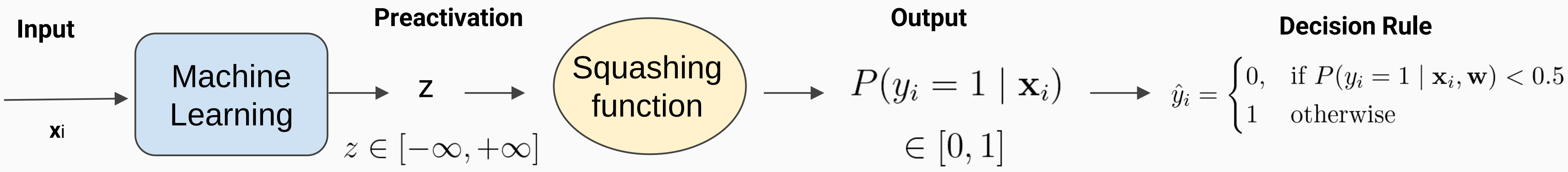
- You can compute the probability of class 0 from the probability of class 1:

$$P(y_i = 0 \mid \mathbf{x}_i) = 1 - P(y_i = 1 \mid \mathbf{x}_i)$$

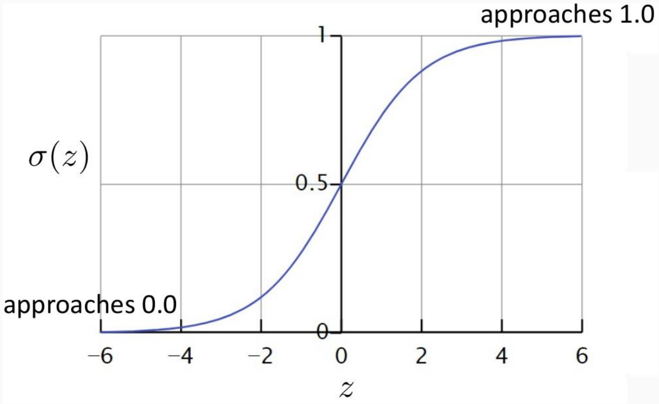
The probability of class 1 determines the probability of class 0

Binary Classification

- How can we “force” the machine learning model to output a probability?



- To interpret our output as a probability, we need a “squashing” function that makes sure that the output is bounded between 0 and 1.

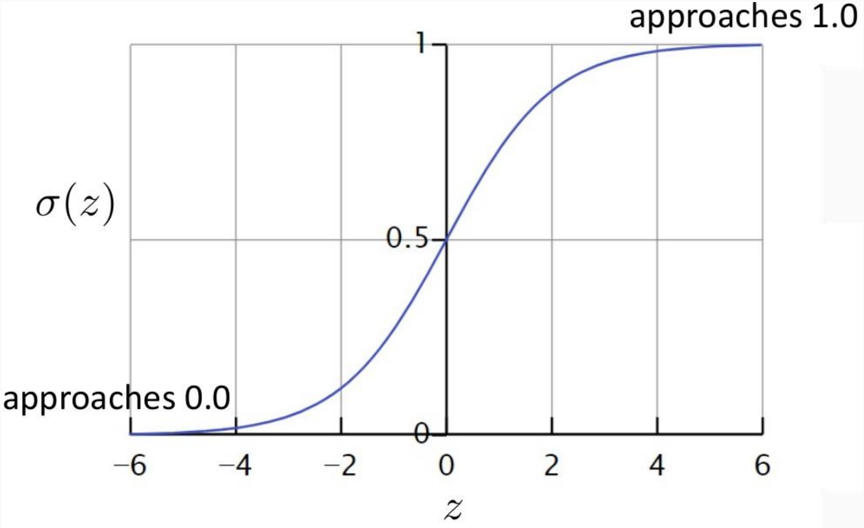


Sigmoid (logistic function):

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

Binary Classification

- The sigmoid has some nice properties:



Sigmoid (logistic function):

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

Derivative:

$$\boxed{\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))}$$

Binary Classification

Sigmoid (logistic function):

$$\sigma(z) = \frac{1}{1+e^{-z}} = (1+e^{-z})^{-1}$$ (Algebraic manipulation)

$$\frac{d\sigma(z)}{dz} = -(1+e^{-z})^{-2} \cdot -e^{-z}$$ (Derivative computation)

$$= \frac{e^{-z}}{(1+e^{-z})^2}$$ (Algebraic manipulation)

$$= \frac{1}{(1+e^{-z})} \cdot \frac{e^{-z}}{(1+e^{-z})}$$ (Algebraic manipulation)

$$= \frac{1}{(1+e^{-z})} \cdot \frac{e^{-z}+1-1}{(1+e^{-z})} = \boxed{\sigma(z)(1-\sigma(z))}$$

Logistic Regression

- Logistic Regression is a linear model used for **binary classification**.

- It is based on a **linear model** whose output is squashed by a **sigmoid**.

$$\overbrace{P(y_i = 1|\mathbf{x}_i, \mathbf{w})}^{P(y_i = 1|\mathbf{x}_i, \mathbf{w}) \in [0, 1]} = \sigma(\overbrace{\mathbf{w}^T \mathbf{x}_i}^{\text{Linear Model } z \in \mathbb{R}})$$

Output **Sigmoid** **Parameters** **Input**

- The objective function is **Binary Cross Entropy**:

- This is a somewhat unfortunate name for a model that we use for classification and not for regression.

Logistic Regression

- Logistic Regression can only draw **linear boundaries** between the two classes:

Decision Rule

$$P(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}_i) \longrightarrow \hat{y}_i = \begin{cases} 0, & \text{if } P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) < 0.5 \\ 1 & \text{otherwise} \end{cases}$$

The decision boundaries are all those points where there is **maximum uncertainty**:

$$P(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}_i) = 0.5$$

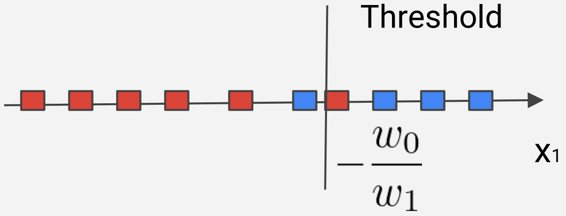
$$\Downarrow$$

$$\boxed{\mathbf{w}^T \mathbf{x}_i = 0}$$

Logistic Regression

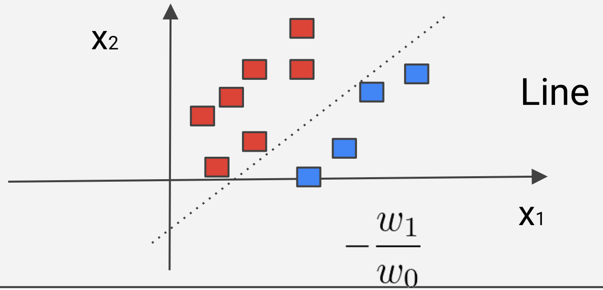
1D space:

$$w_0 + w_1 x_1 = 0 \;\Rightarrow\; x_1 = -\frac{w_0}{w1}$$



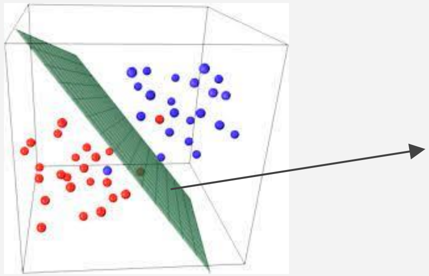
2D space:

$$w_0 + w_1 x_1 + w_2 x_2 = 0 \;\Rightarrow\; x_2 = -\frac{w0}{w2} - \frac{w1}{w2} x_1$$



3D space:

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 = 0$$



Plane

D-dimensional space:

$$\mathbf{w}^T \mathbf{x}_i = 0$$

Hyperplane

Logistic Regression

$$P(y_i = 1 | \mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}_i)$$

- We can train logistic regression with **gradient descent**.

- To make this possible, we need:

1. To define an **objective function**
2. Compute the **gradient** of the objective wrt the parameters.

Logistic Regression

- To train a binary classifier, we need an **objective function**.

- For classification, we can use the **categorical cross-entropy**:

$$CCE = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} ln(p_{ik})$$

$$\textbf{K=2} \begin{cases} y_{i1} = 1, \quad y_{i2} = 1 - y_{i1} \\ p_{i1} = P(y_i = 1 | \mathbf{w}, \mathbf{x}_i) \\ p_{i2} = 1 - P(y_i = 1 | \mathbf{w}, \mathbf{x}_i) \end{cases}$$

$$\Downarrow$$

$$BCE = -\frac{1}{N}\sum_{i=1}^{N} y_{i1} ln(p_{i1}) + (1 - y_{i1}) ln(1 - p_{i1})$$

Lower bound: 0

Upper bound: +inf

- The binary cross-entropy is a special case of the categorical cross-entropy

Logistic Regression

- We can now compute the gradient

$$BCE = -\frac{1}{N}\sum_{i=1}^{N} \underbrace{y_{i1} ln(p_{i1}) + (1 - y_{i1}) ln(1 - p_{i1})}_{\text{Log-Likelihood (LL)}}$$

- We focus first on the term highlighted called **Log-Likelihood** (LL) and compute the derivative wrt to a generic parameter w_j

$$\frac{\partial LL_i(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j} y_i \ln \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \ln\left[1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right]$$

$$= \frac{\partial}{\partial w_j} y_i \ln \sigma(\mathbf{w}^T \mathbf{x}_i) + \frac{\partial}{\partial w_j}(1 - y_i) \ln\left[1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right]$$

Logistic Regression

$$\frac{\partial LL_i(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j} y_i \ln \sigma(\mathbf{w}^T \mathbf{x}_i) + \frac{\partial}{\partial w_j}(1 - y_i) \ln \left[1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right]$$

$$= y_i \frac{1}{\sigma(\mathbf{w}^T \mathbf{x}_i)} \frac{\partial}{\partial w_j} \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \frac{1}{1 - \sigma(\mathbf{w}^T \mathbf{x}_i)} \cdot (-1) \frac{\partial}{\partial w_j} \sigma(\mathbf{w}^T \mathbf{x}_i)$$

(derivation)

$$= \left[\frac{y_i}{\sigma(\mathbf{w}^T \mathbf{x}_i)} - \frac{1 - y_i}{1 - \sigma(\mathbf{w}^T \mathbf{x}_i)}\right] \frac{\partial}{\partial w_j} \sigma(\mathbf{w}^T \mathbf{x}_i)$$

(manipulation)

$$= \left[\frac{y_i}{\sigma(\mathbf{w}^T \mathbf{x}_i)} - \frac{1 - y_i}{1 - \sigma(\mathbf{w}^T \mathbf{x}_i)}\right] \sigma(\mathbf{w}^T \mathbf{x}_i)\left(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)\right) \boxed{x_j}$$

(derivative of the sigmoid)

Why do we have the <u>scalar</u> xj?

$$\mathbf{W}^T \mathbf{x}_i = w_0 + w_1 x_1 + w_2 x_2 + \dots + \boxed{w_j x_j} + \dots + w_D x_D$$

$$\frac{\partial \mathbf{W}^T \mathbf{x}_i}{\partial w_j} = x_j$$

Logistic Regression

$$= \left[\frac{y_i}{\sigma(\mathbf{w}^T\mathbf{x}_i)} - \frac{1-y_i}{1-\sigma(\mathbf{w}^T\mathbf{x}_i)}\right]\sigma(\mathbf{w}^T\mathbf{x}_i)\big(1-\sigma(\mathbf{w}^T\mathbf{x}_i)\big)x_j$$

$$= \left[\frac{y_i(1-\sigma(\mathbf{w}^T\mathbf{x}_i)) - (1-y_i)\sigma(\mathbf{w}^T\mathbf{x}_i)}{\sigma(\mathbf{w}^T\mathbf{x}_i)(1-\sigma(\mathbf{w}^T\mathbf{x}_i))}\right]\sigma(\mathbf{w}^T\mathbf{x}_i)(1-\sigma(\mathbf{w}^T\mathbf{x}_i)x_j$$ (Manipulation)

$$\frac{\partial LL_i(\mathbf{w})}{\partial w_j} = \left[y_i - \sigma(\mathbf{w}^T\mathbf{x}_i)\right]x_j$$ (Simplification)

Gradient (over all parameters **w**):

$$\nabla LL_i(\mathbf{w}) = \left[\frac{\partial LL_i}{\partial w_0}, \frac{\partial LL_i}{\partial w_1}, \ldots, \frac{\partial LL_i}{\partial w_j}, \ldots, \frac{\partial LL_i}{\partial w_D}\right]^T$$

Logistic Regression

$$\frac{\partial LL_i(\mathbf{w})}{\partial w_j} = \left[y_i - \sigma(\mathbf{w}^T \mathbf{x}_i)\right] x_j$$

Gradient (over all parameters **w**):

$$\nabla LL_i(\mathbf{w}) = \left[\frac{\partial LL_i}{\partial w_0}, \frac{\partial LL_i}{\partial w_1}, \ldots, \frac{\partial LL_i}{\partial w_j}, \ldots, \frac{\partial LL_i}{\partial w_D}\right]^T$$

$$\frac{\partial LL_i(\mathbf{w})}{\partial w_0} = \left[y_i - \sigma(\mathbf{w}^T \mathbf{x}_i)\right] \qquad \frac{\partial LL_i(\mathbf{w})}{\partial w_1} = \left[y_i - \sigma(\mathbf{w}^T \mathbf{x}_i)\right] x_1 \qquad \frac{\partial LL_i(\mathbf{w})}{\partial w_j} = \left[y_i - \sigma(\mathbf{w}^T \mathbf{x}_i)\right] x_j$$

The vectorized expression is thus:

$$\nabla LL_i(\mathbf{w}) = \left[y_i - \sigma(\mathbf{w}^T \mathbf{x}_i)\right] \mathbf{x}_i$$

Weights:

$$\mathbf{w} = [w_0, w_1, .., w_j, .., w_D]^T$$

Input:

$$\mathbf{x}_i = [1, x_1, .., x_j, .., x_D]^T$$

Logistic Regression

Gradient of LL (over all parameters **w**):

$$\nabla LL_i(\mathbf{w}) = \left[y_i - \sigma(\mathbf{w}^T \mathbf{x}_i)\right]\mathbf{x}_i$$

$$\nabla BCE(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} \nabla LL_i(\mathbf{w})$$

Vectorized form:

$$\nabla BCE(\mathbf{w}) = \mathbf{X}^T(\sigma(\mathbf{X}\mathbf{w}) - \mathbf{y})$$

$$\nabla BCE(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} \left[\sigma(\mathbf{w}^T \mathbf{x}_i) - y_i\right]\mathbf{x}_i$$

What does this remind you of?

Logistic Regression

Gradient (Logistic Regression):

$$\nabla BCE(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}[\underbrace{\sigma(\mathbf{w}^T\mathbf{x}_i) - y_i}_{\text{Prediction - Label}}]\mathbf{x}_i$$

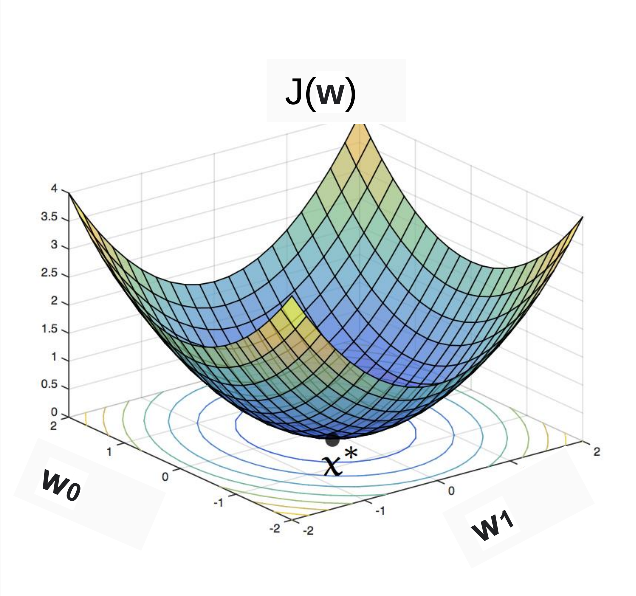
Gradient (Linear Regression):

$$\nabla J(\mathbf{w}) = \sum_{i=1}^{N}(\underbrace{\mathbf{w}^T\mathbf{x}_i - y_i}_{\text{Prediction - Label}})\mathbf{x}_i$$

- The gradient is the same as the one computed for linear regression:
- In both cases, the gradient depends on the difference between the predicted output and the target label (**error**).

Logistic Regression

- How does the optimization space look like in this case?



- If we use **logistic regression** (linear model + BCE) the optimization space is still **convex**.

- This is good news for **optimization**.

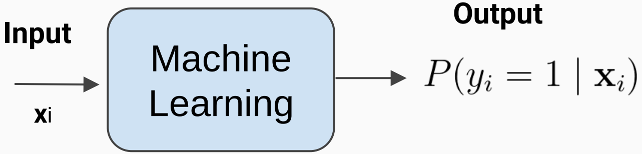
- *Do we have a direct solution?*

A direct **closed-form solution does not exist**.

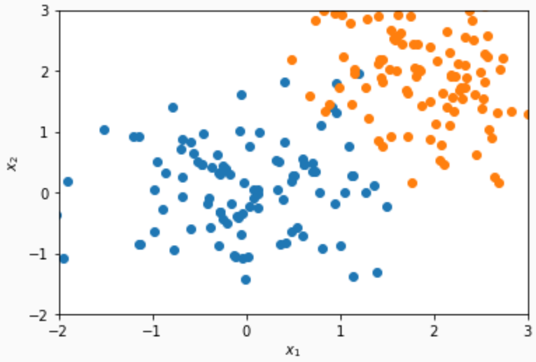
We have to use gradient descent!

Logistic Regression

- Let's now use logistic regression to solve a binary classification task:



Training set:



Model:

$$p(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) = \sigma(\mathbf{w}^T \mathbf{x}_i) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$$

$$p(\mathbf{y} = 1 \mid \mathbf{X}, \mathbf{w}) = \sigma(\mathbf{X}\mathbf{w})$$ (Vectorized form)

$$\mathbf{w} = [w_0, w_1, w_2]^T$$ (Parameters)

Inputs

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} \\ 1 & x_{2,1} & x_{2,2} \\ 1 & \dots & \dots \\ 1 & x_{N,1} & x_{N,2} \end{bmatrix}$$

Labels

$$\mathbf{y} = [y_1, y_2, .., y_i, ..., y_N]^T$$

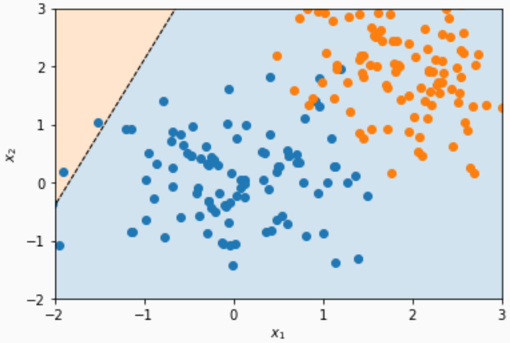
$$y_i = \{0, 1\}$$

Objective:

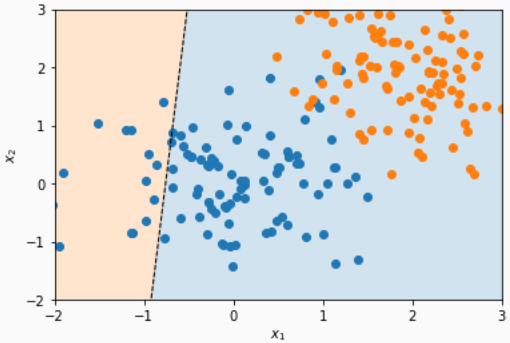
$$J(\mathbf{w}) = -\sum_{i=1}^{N} y_i \ln P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}) + (1 - y_i) \ln(1 - P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}))$$

Logistic Regression

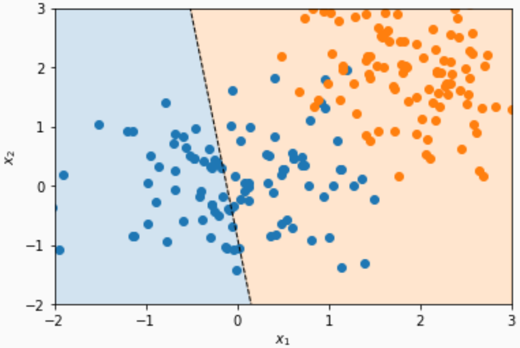
```python
# Parameters Initialization
w = np.array([2.0, 0.5, -0.5])

# Hyperparameters
N_epochs = 100
lr = 0.01

for epoch in range(N_epochs):
  # compute the predictions
  y_hat = logistic_regression(X_train, w)

  # compute the gradient
  grad = np.dot(X_train.T,(y_hat - y_train))

  # parameter updates with gradient descend
  w = w -lr * grad
```



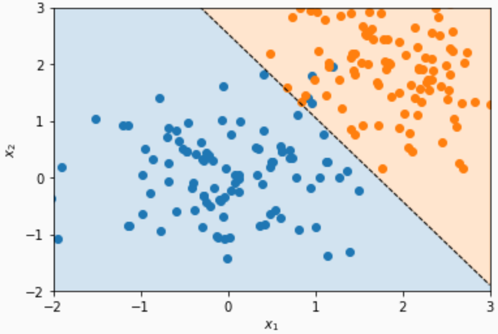
Epoch 1



Epoch 2



Epoch 3

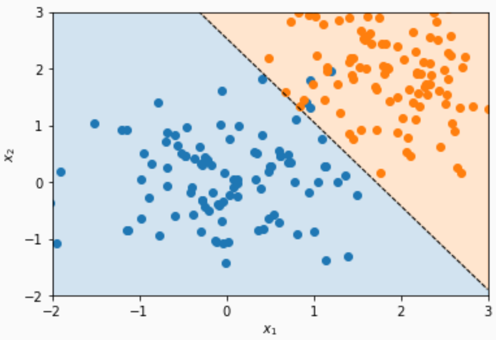


Epoch 100

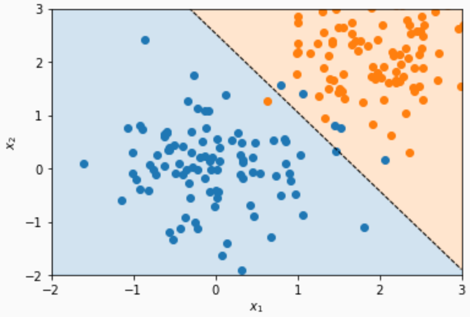
Logistic Regression

```
# Parameters Initialization
w = np.array([2.0, 0.5, -0.5])

# Hyperparameters
N_epochs = 100
lr = 0.01

for epoch in range(N_epochs):
 # compute the predictions
 y_hat = logistic_regression(X_train, w)

 # compute the gradient
 grad = np.dot(X_train.T,(y_hat - y_train))

 # parameter updates with gradient descend
 w = w -lr * grad
```

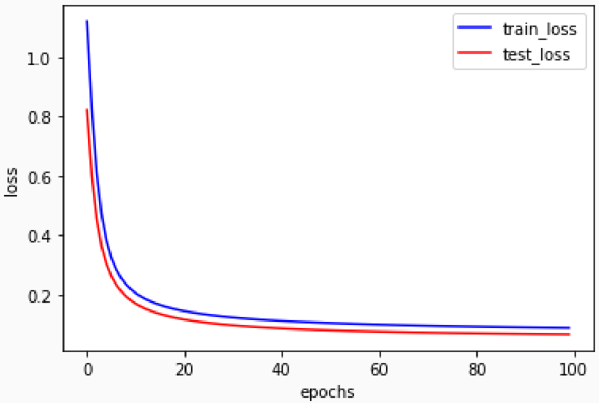
Training Data



Test Data



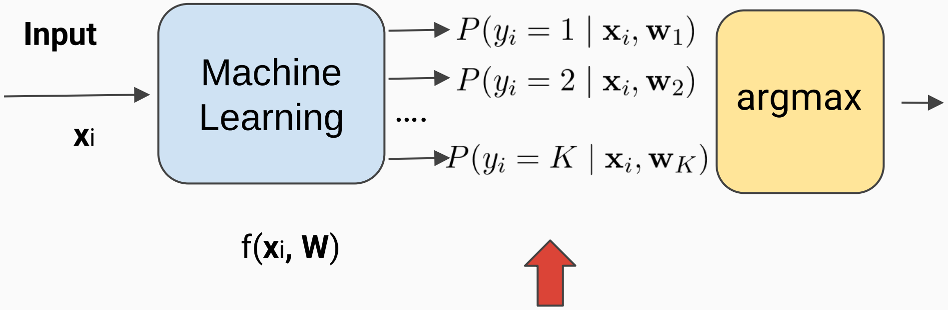
Training Curve



Multiclass Logistic Regression

Multiclass Logistic Regression

- In multiclass classification, we want a machine learning model that outputs K **probabilities**.



Prediction

$$\hat{y}_i = \underset{k}{\operatorname{argmax}}\ P(y_i = k \mid \mathbf{x}_i, \mathbf{w}_k)$$

We want these probabilities to sum up to 1

A single class is predicted for each input

- We can then select the **class** with the **highest probability** with an **argmax** operation.

Multiclass Logistic Regression

- How can we “force” the machine learning model to output **probabilities**?

Input → **Machine Learning** →

$\mathbf{x}_i$

$P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}_1)$

$P(y_i = 2 \mid \mathbf{x}_i, \mathbf{w}_2)$

....

$P(y_i = K \mid \mathbf{x}_i, \mathbf{w}_K)$

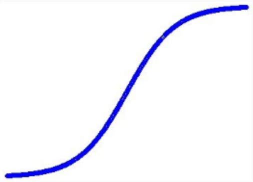
f($\mathbf{x}_i$, $\mathbf{W}$)

- For **binary classification** with logistic regression (where we have a single output), we can use a **sigmoid**:

Input → **Linear Transformation** $\mathbf{w}^T\mathbf{x}_i$ → **Preactivation** z → **Sigmoid** → **Output**

$\mathbf{x}_i$

$z \in [-\infty, +\infty]$

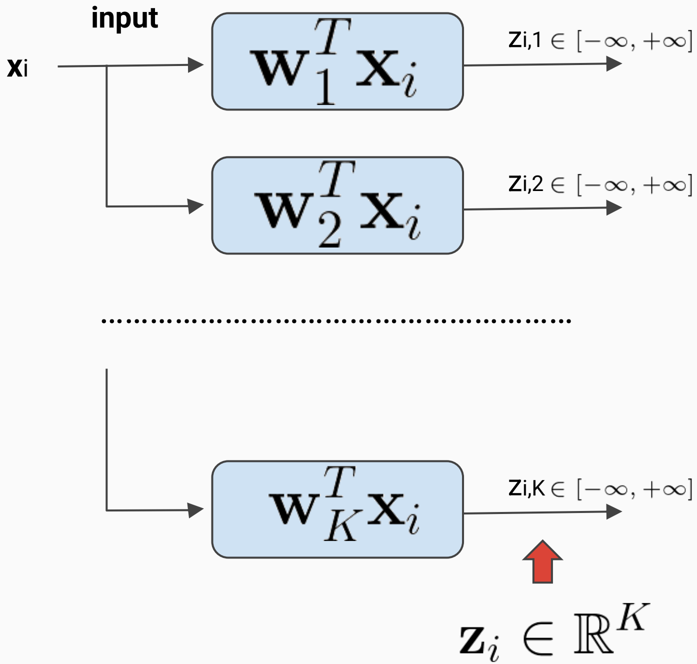


$P(y_i = 1 \mid \mathbf{x}_i)$

$\in [0, 1]$

Multiclass Logistic Regression

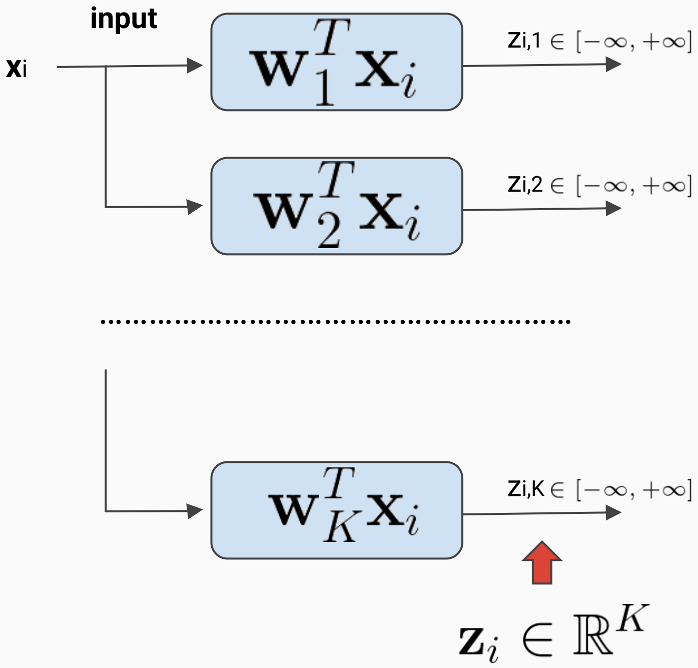
- For multiclass classification, we can map the input $\mathbf{x}_i$ into a **K dimensional space** $\mathbf{z}_i$
- This mapping can be done by applying **K linear transformations** in **parallel**:



We can **vectorize** this operation in this way:

$$\mathbf{z}_i = \mathbf{W}^T \mathbf{x}_i$$

Multiclass Logistic Regression



We can **vectorize** the K linear transformation in this way:

$$\mathbf{z}_i = \mathbf{W}^T \mathbf{x}_i$$

(K, 1) (K, P) (P, 1)

Due to intercept term

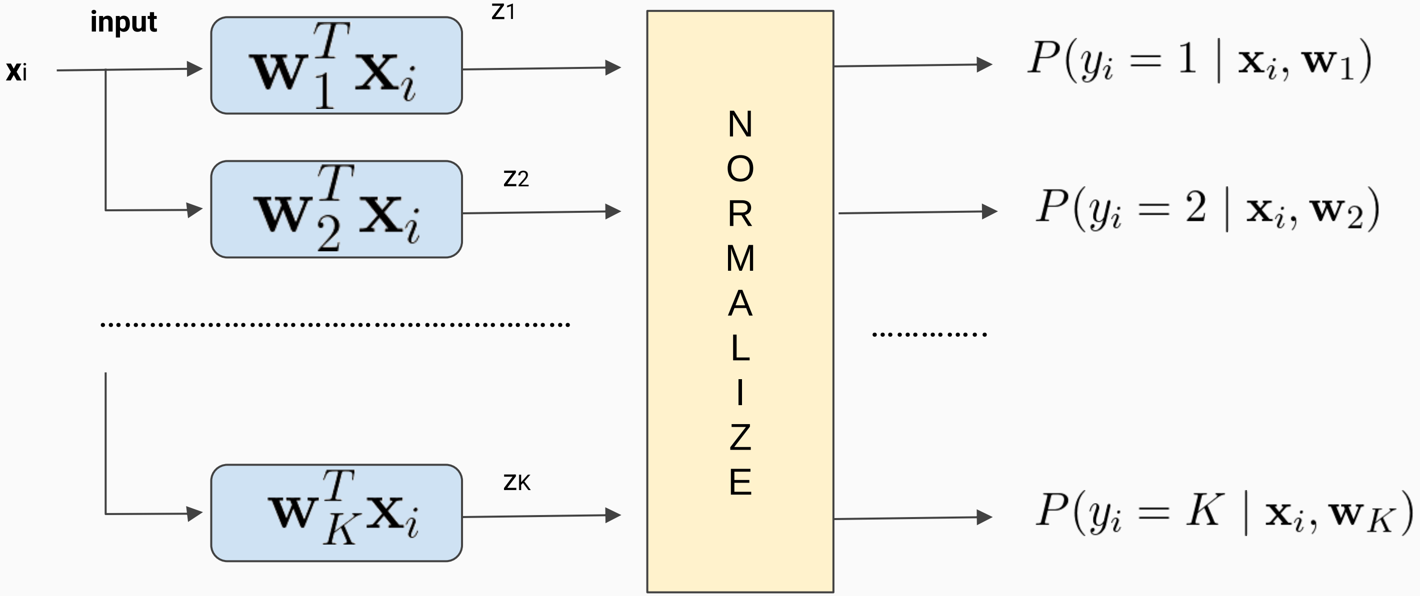
$$\mathbf{x}_i = [1, x_{i1}, x_{i2}, \dots, x_{iD}]^T$$

P = D + 1

$$\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K] = \begin{bmatrix} w_{0,1} & w_{0,2} & \dots & w_{0,K} \\ w_{1,1} & w_{1,2} & \dots & w_{1,K} \\ \dots & \dots & \dots & \dots \\ w_{D,1} & w_{D,2} & \dots & w_{D,K} \end{bmatrix}$$

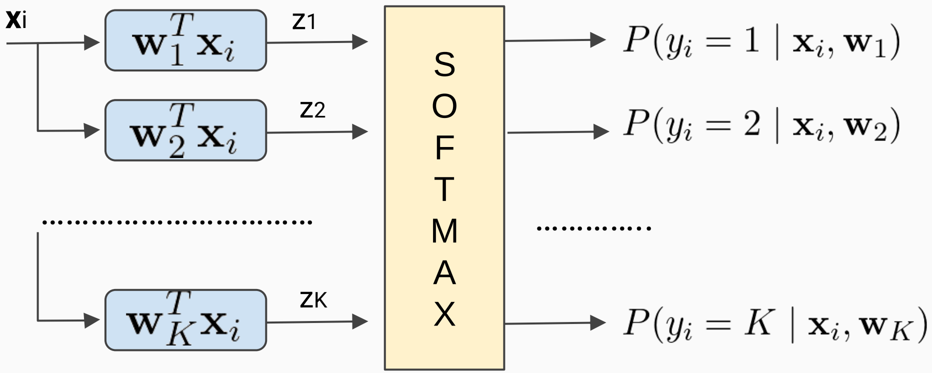
Multiclass Logistic Regression

- Now we can apply a **squashing function** to output the **probabilities** over the K classes.



Multiclass Logistic Regression

- A popular choice is the **softmax function**:



$$softmax(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}$$

$$P(y = k \mid \mathbf{x}_i, \mathbf{W}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}_i)}{\sum_{j=1}^{K} \exp(\mathbf{w}_k^T \mathbf{x}_i)}$$

The softmax ensures that:

1. *Each probability ranges between 0 and 1.*
2. *The sum of all the K probabilities is 1.*

The softmax introduces **competition** within the output units: the **increase** of one probability leads to a **decrease** in the others.

Linear Transformation + Softmax = Multiclass logistic regression or **Softmax classification**.

Multiclass Logistic Regression

- A popular choice is the **softmax function**:

$$softmax(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}$$

$\mathbf{x}_i \rightarrow \mathbf{w}_1^T \mathbf{x}_i \xrightarrow{z_1} 3.4 \longrightarrow$ exp(3.4)/(exp(3.4)+ exp(-1.1)+exp(1.6)) = **0.8500**

$\mathbf{x}_i \rightarrow \mathbf{w}_2^T \mathbf{x}_i \xrightarrow{z_2} -1.1 \longrightarrow$ exp(-1.1)/(exp(3.4)+ exp(-1.1)+exp(1.6)) = **0.0094**

$\mathbf{x}_i \rightarrow \mathbf{w}_3^T \mathbf{x}_i \xrightarrow{z_3} 1.6 \longrightarrow$ exp(1.6)/(exp(3.4)+ exp(-1.1)+exp(1.6)) = **0.1405**

They sum to one

$$\mathbf{z} = [z_1, .., z_k, ..., z_K]^T$$

Multiclass Logistic Regression

- The **sigmoid** is a special case of the softmax function:

$$\boxed{softmax(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}}$$

K = 2

$$\Downarrow$$

$$softmax(z_1, z_2)_1 = \frac{exp(z_1)}{exp(z_1) + exp(z_2)} = \frac{1}{1 + exp(z_2 - z_1)}$$

Only one output is needed in a binary classification

$$z_2 - z_1 = \mathbf{w}_2^T\mathbf{x} - \mathbf{w}_1^T\mathbf{x} = (\mathbf{w}_2 - \mathbf{w}_1)^T\mathbf{x} = \mathbf{w}^T\mathbf{x}$$

$$softmax(z) = \frac{1}{1 + e^{-z}} = \boxed{\sigma(z)}$$

Multiclass Logistic Regression

- The softmax has some nice properties. For instance:

$$softmax(\mathbf{z}) = softmax(\mathbf{z} + c)$$

- This is helpful because we can improve the **numerical stability** of the softmax:

$$softmax(\mathbf{z}) = softmax(\mathbf{z} - \max_j z_j)$$

$\xrightarrow{z_1}$ 50

$\xrightarrow{z_2}$ 49 **NaN**

$\xrightarrow{z_3}$ -15

$\xrightarrow{z_1}$ 50 - 50 = 0 **0.73**

$\xrightarrow{z_2}$ 49 - 50 = -1 **0.27**

$\xrightarrow{z_3}$ -15 -50 = -45 **0.00**

We will see other nice properties in the training part.

Multiclass Logistic Regression

- Let's now analyze the **decision boundaries** of multiclass **logistic regression**.
- The **boundary condition** for two classes k and j selected within the pool of K classes is given by:

$$P(y = k \mid \mathbf{x}, \mathbf{w}) = P(y = j \mid \mathbf{x}, \mathbf{w})$$ (equal probability)

$$softmax(\mathbf{z})_k = softmax(\mathbf{z})_j$$ (replace with softmax)

$$softmax(\mathbf{W}^T\mathbf{x})_k = softmax(\mathbf{W}^T\mathbf{x})_j$$ (replace z)

$$\frac{\exp(\mathbf{w}_k^T\mathbf{x})}{\sum_{i=1}^{K}\exp(\mathbf{w}_i^T\mathbf{x})} = \frac{\exp(\mathbf{w}_j^T\mathbf{x})}{\sum_{i=1}^{K}\exp(\mathbf{w}_i^T\mathbf{x})}$$ (expand softmax)

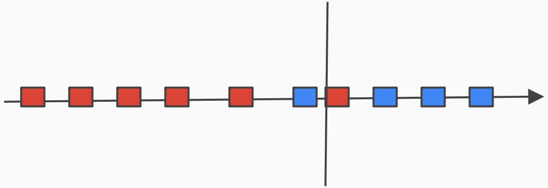
$$\mathbf{w}_k^T\mathbf{x} = \mathbf{w}_j^T\mathbf{x}$$ (algebraic manipulation)

$$\boxed{(\mathbf{w}_k - \mathbf{w}_j)^T\mathbf{x} = 0}$$ (boundary condition)

Multiclass Logistic Regression

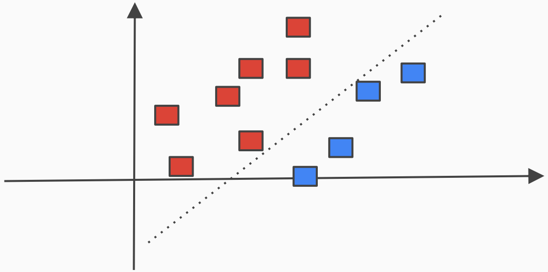
$$(\mathbf{w}_k - \mathbf{w}_j)^T \mathbf{x} = 0$$

1D space:



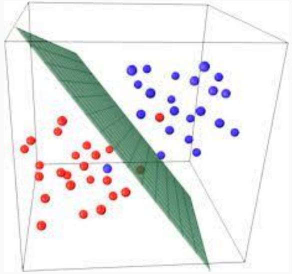
Threshold

2D space:



Line

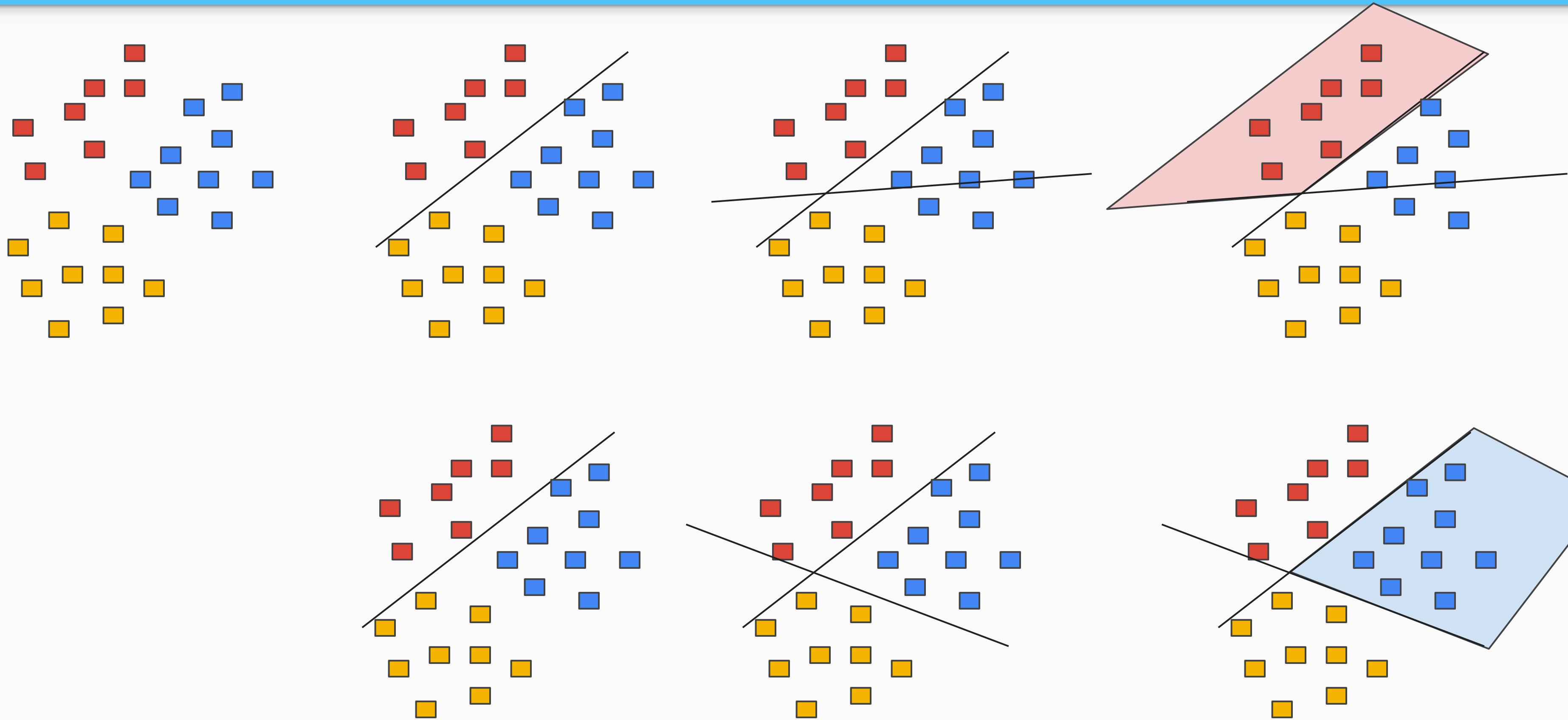
3D space:



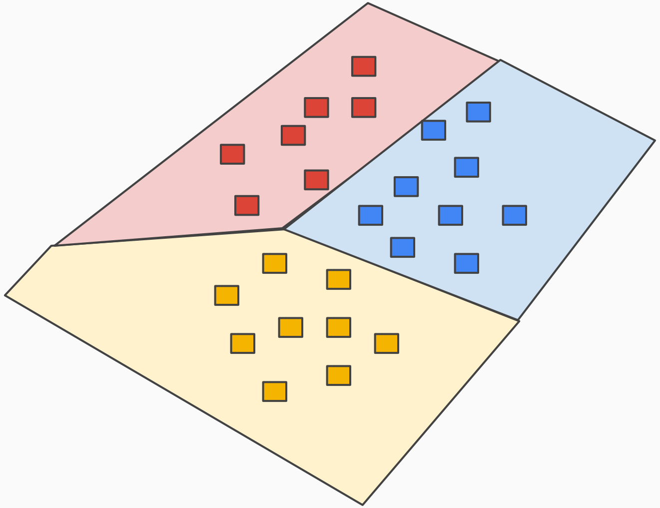
Plane

This is the same as **binary classification**. In our multiclass context, we have to jointly consider all the boundaries across classes

Multiclass Logistic Regression



Multiclass Logistic Regression



For multiclass logistic regression, the boundaries are defined by:

- 1D case *Multiple thresholds*
- 2D case *Line intersections*
- 3D case *Plane intersections*
- D > 3 case *Hyperplane intersections*

Multiclass Logistic Regression

- As usual, we can train our multiclass classifier with **gradient descent**.

- To make this possible, we need:

1. To define an **objective function** ⇨ Categorical Cross Entropy (Negative Log-Likelihood)
2. Compute the **gradient** of the objective wrt the parameters.

Multiclass Logistic Regression

- Let’s now plug the softmax in the expression of NLL:

$$NLL = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \ln P(y_i = k \mid \mathbf{x}_i, \mathbf{w})$$

$$NLL = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \ln \frac{\exp(z_{ik})}{\sum_{j=1}^{K}\exp(z_{ij})}$$

$$NLL = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\left(z_{ik} - \ln \sum_{j=1}^{K}\exp(z_{ij})\right)$$

One-hot label representation

$$\begin{array}{cc} & \text{\# Classes K} \\ \text{\# Examples N} & \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

Multiclass Logistic Regression

$$NLL = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\Big(\underbrace{z_{ik} - \ln\sum_{j=1}^{K}\exp(z_{ij})}\Big)$$

To minimize NLL, we have to **maximize** this term (the bigger the better)

- Intuitively, when training the model to minimize the NLL:

1. We **push up** the output corresponding to the **right label** z_{ik}.

2. At the same time we **push down** the contribution of all the **other outputs** (second term of the equation)

Multiclass Logistic Regression

- Now, we have an **objective function**. We also need the **gradient** to **train** the model.

$$NLL = -\sum_{i=1}^{N} \underbrace{\sum_{k=1}^{K} y_{ik}\left(z_{ik} - \ln \sum_{j=1}^{K} \exp(z_{ij})\right)}_{\text{LLi}}$$

One-hot label representation

$$\begin{array}{c c} & \text{\# Classes K} \rightarrow \\ \text{\# Examples N} \downarrow & \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}$$

$$LL_i = \sum_{k=1}^{K} y_{ik}\left(z_{ik} - \ln \sum_{j=1}^{K} \exp(z_{ij})\right)$$

$$= \sum_{k=1}^{K} y_{ik} z_{ik} - \sum_{k=1}^{K} y_{ik} \ln \sum_{j=1}^{K} \exp(z_{ij})$$

1 →

Remember: we are using the one-hot label representation.

(Algebraic Manipulation)

Multiclass Logistic Regression

$$LL_i = \sum_{k=1}^{K} y_{ik} z_{ik} - \ln \sum_{j=1}^{K} \exp(z_{ij})$$

- We can now compute the **derivative** wrt the generic activation $\mathbf{z_{il}}$ for the sample i and class l :

$$y_{i1}z_{i1} + y_{i2}z_{i2} + \ldots + y_{il}z_{il} + \ldots + y_{iK}z_{iK}$$

$$\frac{\partial LL_i}{\partial z_{il}} = \frac{\partial}{\partial z_{il}} \sum_{k=1}^{K} y_{ik} z_{ik} - \frac{\partial}{\partial z_{il}} \ln \sum_{j=1}^{K} \exp(z_{ij})$$

$$= y_{il} - \frac{\exp(z_{il})}{\sum_{j=1}^{K} \exp(z_{il})}$$

This is the **softmax**!

Multiclass Logistic Regression

$$\boxed{\frac{\partial LL_i}{\partial z_{il}} = y_{il} - softmax(\mathbf{z}_i)_l}$$

- We want to compute the derivative over the generic weight w_{ml} connecting the input feature m to class l:

$$\mathbf{W} = [\mathbf{w}_1, ..., \mathbf{w}_l, ..., \mathbf{w}_K] = \begin{bmatrix} w_{0,1} & \dots & w_{0,l} & \dots & w_{0,K} \\ \dots & \dots & \dots & \dots & \dots \\ w_{m,1} & \dots & w_{m,l} & \dots & w_{m,K} \\ \dots & \dots & \dots & \dots & \dots \\ w_{D,1} & \dots & w_{D,l} & \dots & w_{D,K} \end{bmatrix}$$

$$z_{il} = \mathbf{w}_l^T \mathbf{x}_i = w_{0l} + w_{1l} x_{i1} + \dots + w_{ml} x_{im} + \dots + w_{Dl} x_{iD}$$

Multiclass Logistic Regression

$$\frac{\partial LL_i}{\partial z_{il}} = y_{il} - softmax(\mathbf{z}_i)_l$$

$$\frac{\partial LL_i}{\partial w_{ml}} = \frac{\partial LL_i}{\partial z_{il}} \cdot \frac{\partial z_{il}}{\partial w_{ml}}$$

$$z_{il} = \mathbf{w}_l^T \mathbf{x}_i = w_{0l} + w_{1l}x_{i1} + \ldots + w_{ml}x_{im} + \ldots + w_{Dl}x_{iD}$$

$$\boxed{\frac{\partial LL_i}{\partial w_{ml}} = \left(y_{il} - softmax(\mathbf{z}_i)_l\right) x_{im}}$$

Multiclass Logistic Regression

- We can now collect all the partial derivatives for all the features M:

$$\boxed{\frac{\partial LL_i}{\partial w_{ml}} = \left(y_{il} - softmax(\mathbf{z}_i)_l\right) x_{im}}$$

$$\nabla LL_i(\mathbf{w}_m) = \left(y_{il} - softmax(\mathbf{z}_i)_l\right) \mathbf{x}_i$$

Gradient for all the **features**.

$$\mathbf{J}_{LL_i}(\mathbf{W}) = \underbrace{\overbrace{\mathbf{x}_i}^{\text{P x 1}} \underbrace{\left(\overbrace{\mathbf{y}_i}^{\text{K x 1 (one-hot)}} - \overbrace{softmax(\mathbf{z}_i)}^{\text{K x 1}}\right)^T}_{\text{1 x K}}}_{\text{P x K}}$$

Gradient (Jacobian) for all the **features** and **outputs**.

Transpose is needed here to match the dimensionality

Multiclass Logistic Regression

$$\mathbf{J}_{LL_i}(\mathbf{W}) = \mathbf{x}_i\Big(\mathbf{y}_i - softmax(\mathbf{z}_i)\Big)^T$$

Gradient (Jacobian for all the features and outputs)

- We can now compute the gradient of the **NLL** for all the inputs:

$$\mathbf{J}_{NLL}(\mathbf{W}) = -\sum_{i=1}^{N} \mathbf{J}_{LL_i}(\mathbf{W}) = \underbrace{\sum_{i=1}^{N} \overbrace{\mathbf{x}_i}^{\text{P x 1}} \underbrace{\Big(\overbrace{softmax(\overbrace{\mathbf{W}^T}^{\text{K x P}} \overbrace{\mathbf{x}_i}^{\text{P x 1}})}^{\text{K x 1}} - \overbrace{\mathbf{y}_i}^{\text{K x 1 (one hot)}}\Big)^T}_{\text{1 x K}}}_{\text{P x K}}$$

$$\boxed{\mathbf{J}_{NLL}(\mathbf{W}) = \overbrace{\mathbf{X}^T}^{\text{P x N}} \underbrace{\Big(\overbrace{softmax(\overbrace{\mathbf{X}}^{\text{N x P}}\overbrace{\mathbf{W}}^{\text{P x K}})}^{\text{N x K}} - \overbrace{\mathbf{Y}}^{\text{N x K}}\Big)}_{\text{predictions - labels}}}$$

Vectorized form

Gradient

$$\boxed{\mathbf{J}_{NLL}(\mathbf{W}) = \underbrace{\mathbf{X}^T}_{\text{P x N}} \underbrace{\left(softmax(\underbrace{\underbrace{\mathbf{X}}_{\text{N x P}}\underbrace{\mathbf{W}}_{\text{P x K}}}_{\text{N x K}}) - \underbrace{\mathbf{Y}}_{\text{N x K}}\right)}_{\text{N x K}}}$$

$$\mathbf{Y} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{X} = \overbrace{\begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,D} \\ 1 & x_{2,1} & \dots & x_{2,D} \\ 1 & \dots & \dots & \dots \\ 1 & x_{N,1} & \dots & x_{N,D} \end{bmatrix}}^{\text{P = D + 1}} \quad \mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K] = \begin{bmatrix} w_{0,1} & w_{0,2} & \dots & w_{0,K} \\ w_{1,1} & w_{1,2} & \dots & w_{1,K} \\ \dots & \dots & \dots & \dots \\ w_{D,1} & w_{D,2} & \dots & w_{D,K} \end{bmatrix}$$

Multiclass Logistic Regression

$$\underbrace{\boxed{\mathbf{J}_{NLL}(\mathbf{W}) = \mathbf{X}^T(softmax(\mathbf{X}\mathbf{W}) - \mathbf{Y})}}_{\text{P x K}}$$

⚠ Do not confuse the Jacobian $\mathbf{J}_f(\mathbf{W})$ with the objective function $J(\mathbf{W})$

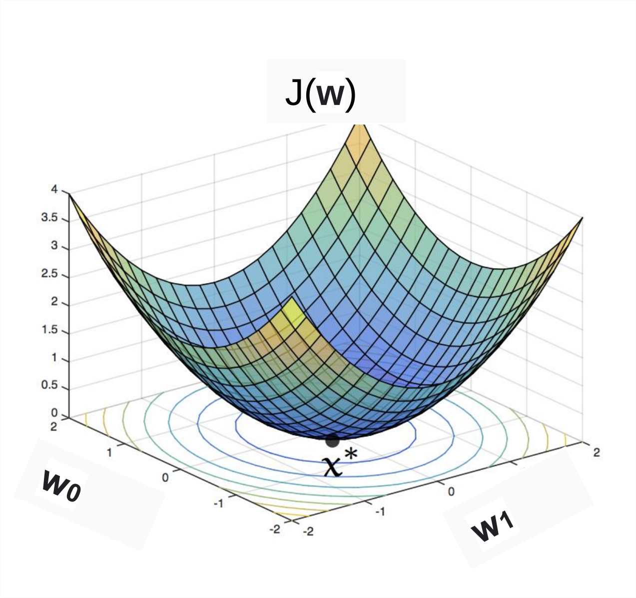
$$\begin{array}{c c} & \text{K} \\ \text{P=D+1} & \begin{bmatrix} w_{0,1} & w_{0,2} & \cdots & w_{0,K} \\ w_{1,1} & w_{1,2} & \cdots & w_{1,K} \\ \cdots & \cdots & \cdots & \cdots \\ w_{D,1} & w_{D,2} & \cdots & w_{D,K} \end{bmatrix} \end{array}$$

$$\mathbf{J}_f(\mathbf{W}) = \begin{array}{c c} \text{P=D+1} & \begin{bmatrix} \frac{\partial f(\mathbf{W})}{\partial w_{0,1}} & \frac{\partial f(\mathbf{W})}{\partial w_{0,2}} & \cdots & \frac{\partial f(\mathbf{W})}{\partial w_{0,K}} \\ \frac{\partial f(\mathbf{W})}{\partial w_{1,1}} & \frac{\partial f(\mathbf{W})}{\partial w_{1,2}} & \cdots & \frac{\partial f(\mathbf{W})}{\partial w_{1,K}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f(\mathbf{W})}{\partial w_{D,1}} & \frac{\partial f(\mathbf{W})}{\partial w_{D,2}} & \cdots & \frac{\partial f(\mathbf{W})}{\partial w_{D,K}} \end{bmatrix} \end{array}$$

- In multiclass logistic regression, the weights are gathered in an M x K **matrix** and not anymore in an M-dimensional vector (due to the multiple classes K).
- As a result, also the gradient will be an M x K matrix.
- This matrix containing all the partial derivatives is called **Jacobian.**
- It extends the concept of gradients to functions with **multiple inputs** and **multiple outputs.**

Multiclass Logistic Regression

- How the optimization space looks like in this case?



- If we use **multiclass logistic regression** (linear model + CCE) the optimization space is still **convex**.
- This is a good news for **optimization**.

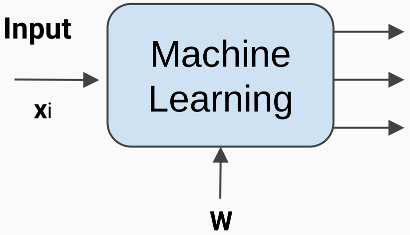
- *Do we have a direct solution?*

A direct **closed-form solution does not exist**.

We have to use gradient descent!

Multiclass Logistic Regression

- Let's now use multiclass logistic regression to solve a multiclass classification task:



Outputs

$$P(y_i = 1 \mid \mathbf{x}_i, \mathbf{w}_1)$$
$$P(y_i = 2 \mid \mathbf{x}_i, \mathbf{w}_1)$$
$$P(y_i = 3 \mid \mathbf{x}_i, \mathbf{w}_1)$$

Model:

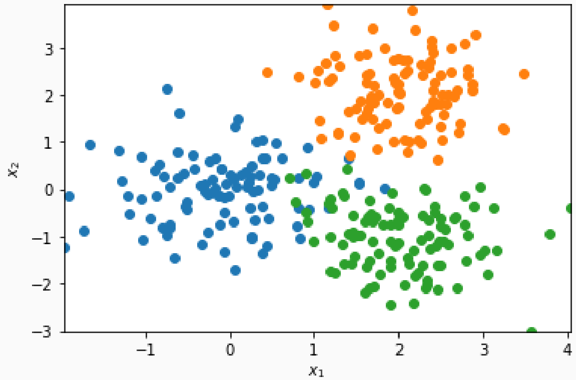
$$p(y_i = k \mid \mathbf{x}_i, \mathbf{W}) = softmax(\mathbf{W}^T \mathbf{x}_i)_k$$

$$\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_3] = \begin{bmatrix} w_{0,1} & w_{0,2} & w_{0,3} \\ w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{bmatrix}$$

Objective:

$$NLL(\mathbf{W}) = -\ln P(\mathbf{Y} \mid \mathbf{X}, \mathbf{w}) = -\sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \ln softmax(\mathbf{W}^T \mathbf{x}_i)_k$$

Training set:



Inputs

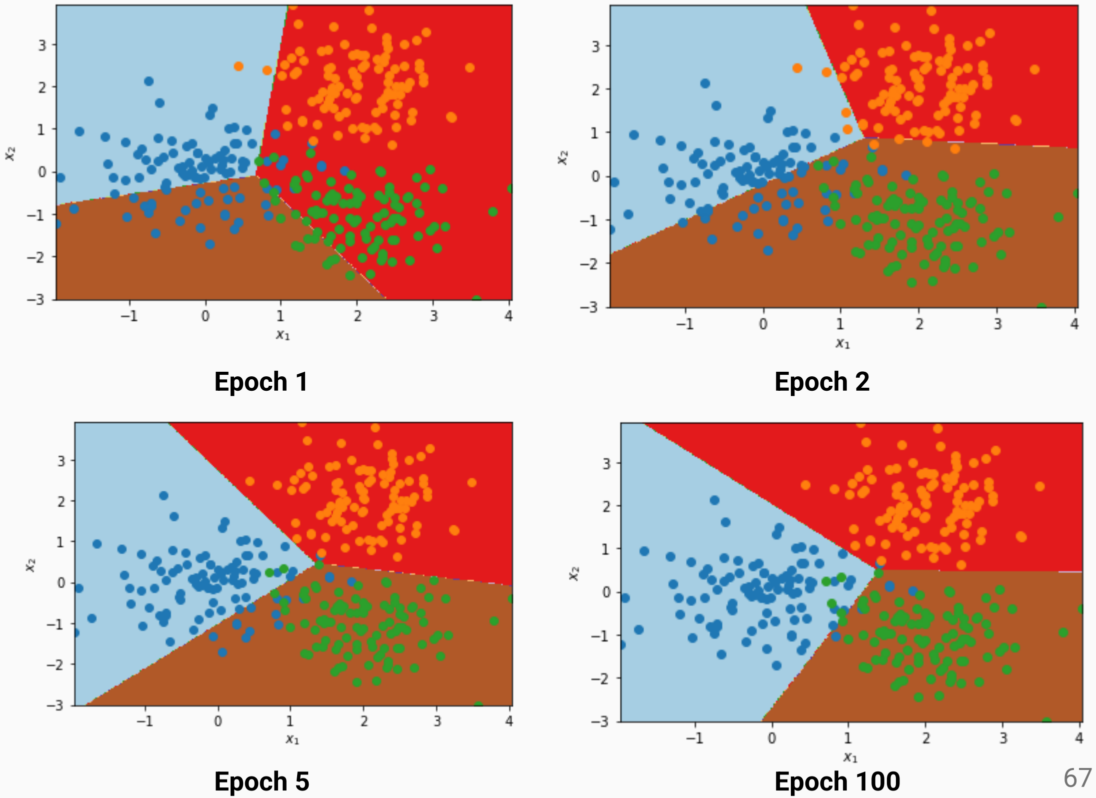
$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} \\ 1 & x_{2,1} & x_{2,2} \\ 1 & \ldots & \ldots \\ 1 & x_{N,1} & x_{N,2} \end{bmatrix}$$

Labels (one-hot)

$$\mathbf{Y} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ \ldots & \ldots & \ldots \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

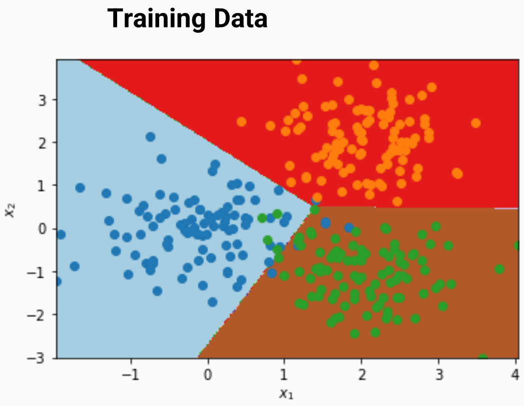
Example

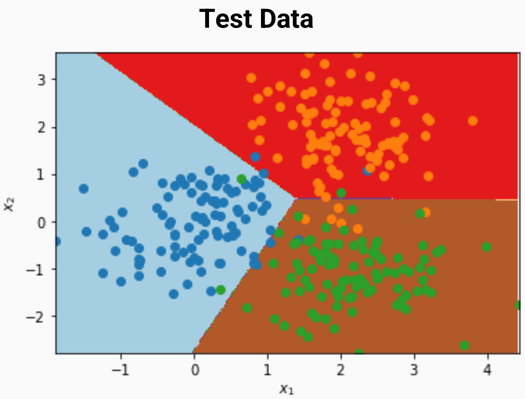
```python
# Initial Values
W = np.random.randint(-1, 1, (3, 3))
N_epochs = 100
lr = 0.01

for epoch in range(N_epochs):
 # compute the predictions
 Y_hat = multiclass_lr(X_train, W)

 # compute the gradient
 grad = np.dot(X_train.T,(Y_hat - Y_train))

 # parameter updates with gradient descend
 W = W - lr * grad
```

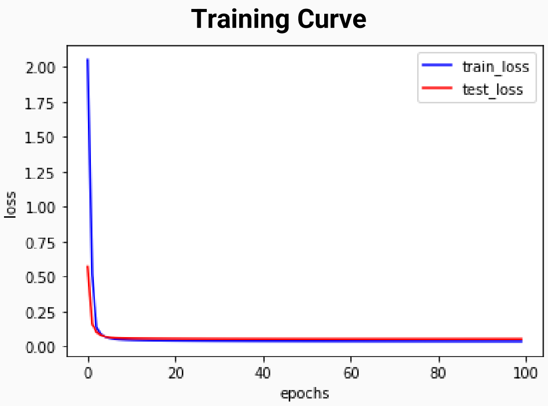


Example

```
# Initial Values
W = np.random.randint(-1, 1, (3, 3))
N_epochs = 100
lr = 0.01

for epoch in range(N_epochs):
 # compute the predictions
 Y_hat = multiclass_lr(X_train, W)

 # compute the gradient
 grad = np.dot(X_train.T,(Y_hat - Y_train))

 # parameter updates with gradient descend
 W = W - lr * grad
```







Final Remarks on Linear Models

- The **capacity** of the model depends on the **input dimensionality D**. Remember, the **VC dimension** is **D + 1** for logistic regression without any transformation.

- This means that for **low-dimensional data**, there is an increased risk of **underfitting**.

- Instead, for **high-dimensional data** it is possible to **overfit**.

Linear models are easy to optimize

Fast training and predictions

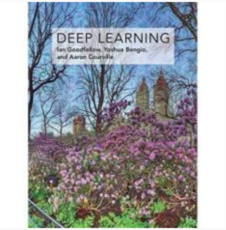
Interpretability

Suitable for linearly separable classes

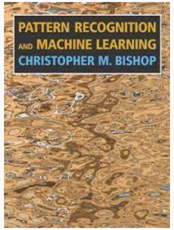
Linear models are very basic machine learning models but are still used a lot in machine learning.

In the next lectures, we will see improvements and extensions of linear models:
neural networks and **support vector machines**.

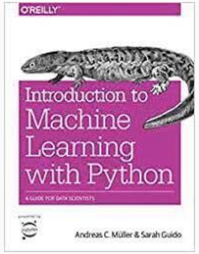
Additional Material



Chapter 2: Linear Algebra
Chapter 3: Probability and information theory
Chapter 5: Machine Learning Basics



1.1.0 Example: Polynomial Curve Fitting
1.2.0 Probability Theory
3.1.0 Linear Basis Function Models
3.1.1 Maximum likelihood and least squares
4.3.2 Logistic regression



Introduction (page 1-27)
Linear Models (page 47-70)

Lab Session

- During the weekly lab session, we will do:

colab Tutorial on Linear Regression

colab Tutorial on Logistic Regression

colab Tutorial on Multiclass Logistic Regression

Major Assignment No.1 Deadline:
Monday 11:59PM, September 1st, 2024
(Submission from Moodle)

Appendix

Scale of the Objective Function

- In the previous slides and lectures, we have seen the objective functions written with different **normalization factors**.

- For instance:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2 \qquad MSE = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2 \qquad MSE = \sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2$$

$$CCE = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} ln(p_{ik}) \qquad CCE = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} ln(p_{ik})$$

- The normalization factor does not play an important role and we can choose the one that is more convenient for us.

Scale of the Objective Function

- The normalization factor of the objective only affects the **scale** of the gradient, but **not where the gradient** is pointing.

$$\tilde{J}(\mathbf{w}) = kJ(\mathbf{w})$$

(k is assumed a positive scaling factor)

$$\nabla\tilde{J}(\mathbf{w}) = k\nabla J(\mathbf{w})$$

- When we use gradient descend, this term gets multiplied by the **learning rate**:

$$\mathbf{w}_{new} = \mathbf{w} - \eta k \nabla J(\mathbf{w}) \longrightarrow$$

You can choose the scale you want and tune the learning rate for your problem

Scale of the Objective Function

- A common choice is to scale the objective function with the number of training samples N:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2 \qquad CCE = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} ln(p_{ik})$$

- This way, the scale of the loss function (and those of the corresponding gradient) does not depend on the number of samples in the training set.

- When using gradient descend, we can set up a similar learning rate even if we change the number of training samples.