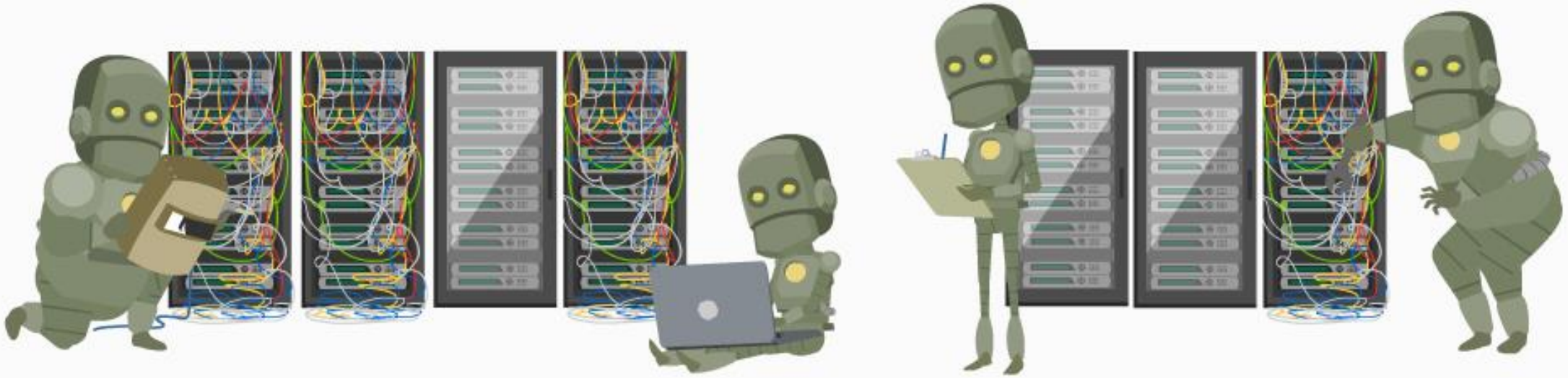
365 DataScience

NEURAL NETWORKS

CHEAT SHEET



Introduction

Most generally, a machine learning algorithm can be though of as a black box. It takes inputs and gives outputs.

The purpose of this course is to show you how to create this 'black box' and tailor it to your needs.

For example, we may create a model that predicts the weather tomorrow, based on meteorological data about the past few days.

The "black box" in fact is a mathematical model. The machine learning algorithm will follow a kind of trial-and-error method to determine the model that estimates the outputs, given inputs.

Once we have a model, we must **train** it. **Training** is the process through which, the model **learns** how to make sense of input data.

Types of machine learning

Supervised

It is called *supervised* as we provide the algorithm not only with the inputs, but also with the targets (desired outputs). This course focuses on supervised machine learning.

Based on that information the algorithm learns how to produce outputs as close to the **targets** as possible.

The objective function in supervised learning is called **loss function** (also cost or error). We are trying to minimize the loss as the lower the loss function, the higher the accuracy of the model.

Common methods:
- Regression
- Classification

Unsupervised

In *unsupervised* machine learning, the researcher feeds the model with inputs, but **not** with targets. Instead she asks it to find some sort of dependence or underlying logic in the data provided.

For example, you may have the financial data for 100 countries. The model manages to divide (cluster) them into 5 groups. You then examine the 5 clusters and reach the conclusion that the groups are: "Developed", "Developing but overachieving", "Developing but underachieving", "Stagnating", and "Worsening".

The algorithm divided them into 5 groups based on **similarities**, but you didn't know what similarities. It could have divided them by location instead.

Common methods:
- Clustering

Reinforcement

In reinforcement ML, the goal of the algorithm is to maximize its reward. It is inspired by human behavior and the way people change their actions according to incentives, such as getting a reward or avoiding punishment.

The objective function is called a **reward function.** We are trying to maximize the reward function.

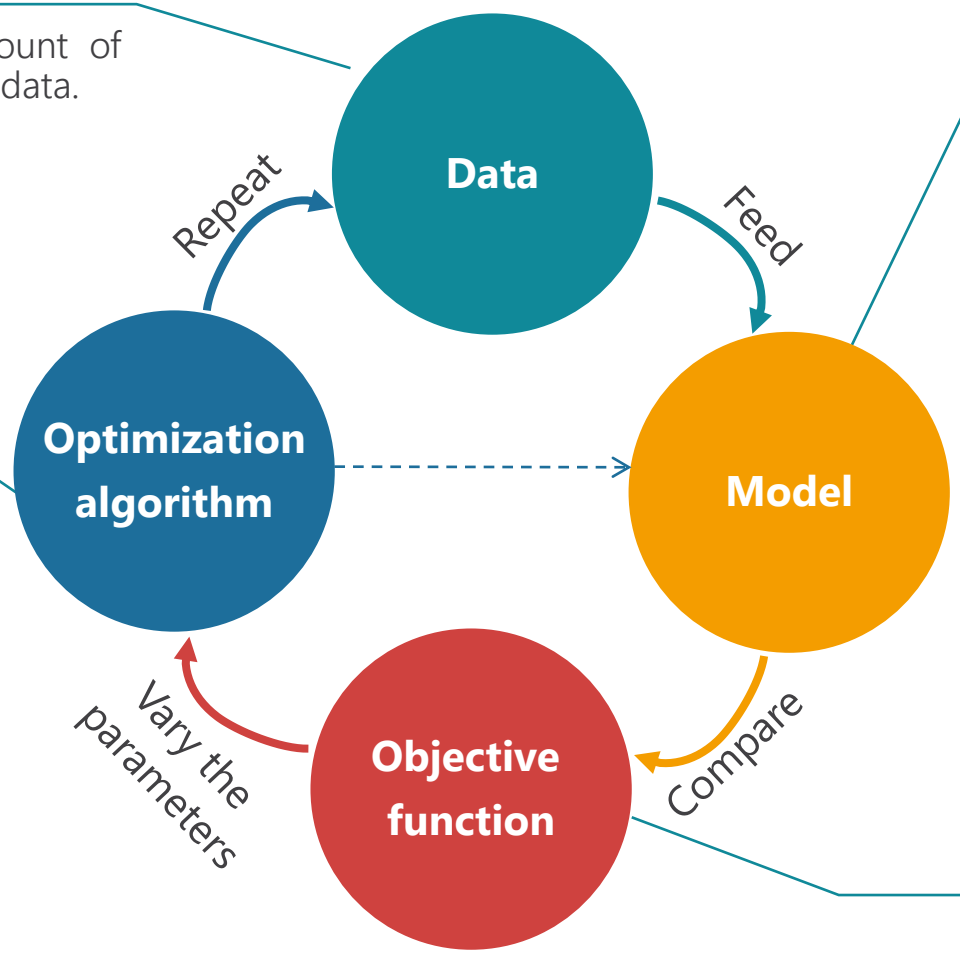
An example is a computer playing Super Mario. The higher the score it achieves, the better it is performing. The score in this case is the objective function.

Common methods:
- Decision process
- Reward system

Building blocks of a machine learning algorithm

The basic logic behind training an algorithm involves four ingredients: data, model, objective function, and optimization algorithm. They are **ingredients**, instead of steps, as the process is iterative.

First, we need to prepare a certain amount of **data** to train on. Usually, we take historical data.



We choose the type of **model**. Roughly speaking, this is some function, which is defined by the *weights* and the *biases*. We feed the input data into the model. Essentially, the idea of the machine learning algorithm is to find the *parameters* for which the model has the highest predictive power.

We achieve the optimization using an **optimization algorithm.** Using the value of the objective function, the optimization algorithm *varies the parameters* of the model. This operation is repeated until we find the values of the parameters, for which the objective function is optimal.

The **objective function** measures the predictive power of our model. Mathematically, the machine learning problem boils down to *optimizing* this function. For example, in the case of loss, we are trying to *minimize* it.

Types of supervised learning

Supervised learning could be split into two subtypes – **regression** and **classification**.

Regression

Regression outputs are continuous numbers.

Examples:

Predicting the EUR/USD exchange rate tomorrow. The output is a number, such as 1.02, 1.53, etc.

Predicting the price of a house, e.g. $234,200 or $512,414.

One of the main properties of the regression outputs is that they are ordered. As 1.02 < 1.53, and 243200 < 512414, we can surely say that one output is bigger than the other.

This distinction proves to be crucial in machine learning as the algorithm somewhat *gets additional information* from the outputs.

Classification

Classification outputs are labels from some sort of class.

Examples:

Classifying photos of animals. The classes are "cats", "dogs", "dolphins", etc.

Predicting conversions in a business context. You can have two classes of customers, e.g."will buy again" and "won't buy again".

In the case of classification, the labels are not ordered and cannot be compared at all. A dog photo is not "more" or "better" than a cat photo (objectively speaking) in the way a house worth $512,414 is "more" (more expensive) than a house worth $234,200.

This distinction proves to be crucial in machine learning, as the different classes are treated on **an equal footing.**

Model

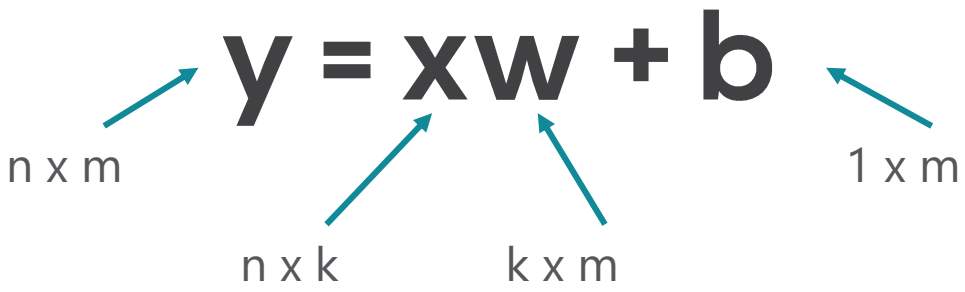
The simplest possible model is **a linear model**. Despite appearing unrealistically simple, in the deep learning context, it is the basis of more complicated models.

There are four elements.

- The input(s), x. That's basically the data that we feed to the model.
- The weight(s), w. Together with the biases, they are called **parameters.** The optimization algorithm will vary the weights and the biases, in order to produce the model that fits the data best.
- The bias(es), b. See weight(s).
- The output(s), y. y is a function of x, determined by w and b.

Each model is determined solely by its parameters (the weights and the biases). That is why we are interested in varying them using the (kind of) trial-and-error method, until we find a model that explains the data sufficiently well.

Model - Continued



Where:

- n is the number of samples (observations)
- m is the number of output variables
- k is the number of input variables

A linear model can represent multidimensional relationships. The shapes of y, x, w, and b are given above (notation is arbitrary).

The simplest linear model, where n = m = k = 1.

y	=	x	w	+	b

The simplest linear model, where m = k = 1, n > 1

y_1	=	$x_1w + b$	=	x_1	w	+	b
y_2		$x_2w + b$		x_2			
...		...		...			
y_n		$x_nw + b$		x_n			

Examples:

27	=	4	6	+	3

Since the weights and biases alone define a model, this example shows **the same model** as above but for many data points.

27	=	4	6	+	3
33		5			
...		...			
-3		-1			

Note that we add the bias to each row, essentially simulating an n x 1 matrix. That's how computers treat such matrices, that's why in machine learning we define the bias in this way.

Model multiple inputs

$$y = xw + b$$

n x m

n x k

k x m

1 x m

Where:

- n is the number of samples (observations)
- m is the number of output variables
- k is the number of input variables

We can extend the model to multiple inputs where n, k > 1, m = 1.

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ \dots \\ y_n \end{bmatrix} = \begin{bmatrix} x_{11}w_1 + x_{12}w_2 + \dots + x_{1k}w_k + b \\ x_{21}w_1 + x_{22}w_2 + \dots + x_{2k}w_k + b \\ \dots \\ \dots \\ x_{n1}w_1 + x_{n2}w_2 + \dots + x_{nk}w_k + b \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1k} \\ x_{21} & x_{22} & \dots & x_{2k} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & \dots & x_{nk} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_k \end{bmatrix} + \boxed{b}$$

n x 1

n x k

k x 1

1 x 1

Note that we add the bias to each row, essentially simulating an n x 1 matrix. That's how computers treat such matrices, that's why in machine learning we define the bias in this way.

Example on the next page.

Model multiple inputs

$$y = xw + b$$

n x m — n x k — k x m — 1 x m

Where:

- n is the number of samples (observations)
- m is the number of output variables
- k is the number of input variables

We can extend the model to multiple inputs where n, k > 1 m = 1. An example.

y_1	=	9 x (-2) + 12 x 5 +…+ 13 x (-1) + 4	=	9	12	…	13		-2	+	4
y_2		10 x (-2) + 6 x 5 +…+ 2 x (-1) + 4		10	6	…	2		5		
…		…		…	…	…	…		…		
…		…		…	…	…	…		-1		
y_n		7 x (-2) + 7 x 5 +…+ 1 x (-1) + 4		7	7	…	1				
n x 1				n x k					k x 1		1 x 1

Note that we add the bias to each row, essentially simulating an n x 1 matrix. That's how computers treat such matrices, that's why in machine learning we define the bias in this way.

Model multiple inputs and multiple outputs

$$y = xw + b$$

n x m — n x k — k x m — 1 x m

Where:

- n is the number of samples (observations)
- m is the number of output variables
- k is the number of input variables

We can extend the model to multiple inputs where n, k, m > 1.

$$\begin{bmatrix} y_{11} & \cdots & y_{1m} \\ y_{21} & \cdots & y_{2m} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ y_{n1} & \cdots & y_{nm} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1k} \\ x_{21} & x_{22} & \cdots & x_{2k} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix} \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ w_{21} & \cdots & w_{2m} \\ \cdots & \cdots & \cdots \\ w_{k1} & \cdots & w_{km} \end{bmatrix} + \begin{bmatrix} b_1 & \cdots & b_m \end{bmatrix}$$

n x m — n x k — k x m — 1 x m

Note that we add the bias to each row, essentially simulating an n x m matrix. That's how computers treat such matrices, that's why in machine learning we define the bias in this way.

Example on the next page.

Model multiple inputs and multiple outputs

$$y = xw + b$$

n x m — n x k — k x m — 1 x m

Where:

- n is the number of samples (observations)
- m is the number of output variables
- k is the number of input variables

We can extend the model to multiple inputs where n, k, m > 1.

$$\begin{bmatrix} y_{11} & \cdots & y_{1m} \\ y_{21} & \cdots & y_{2m} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ y_{n1} & \cdots & y_{nm} \end{bmatrix} = \begin{bmatrix} x_{11}w_{11} + x_{12}w_{21} + \ldots + x_{1k}w_{k1} + b_1 & \cdots & x_{11}w_{1m} + x_{12}w_{2m} + \ldots + x_{1k}w_{km} + b_m \\ x_{21}w_{11} + x_{22}w_{21} + \ldots + x_{2k}w_{k1} + b_1 & \cdots & x_{21}w_{1m} + x_{22}w_{2m} + \ldots + x_{2k}w_{km} + b_m \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ x_{n1}w_{11} + x_{n2}w_{21} + \ldots + x_{nk}w_{k1} + b_1 & \cdots & x_{n1}w_{1m} + x_{n2}w_{2m} + \ldots + x_{nk}w_{km} + b_m \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1k} \\ x_{21} & x_{22} & \cdots & x_{2k} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix} \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ w_{21} & \cdots & w_{2m} \\ \cdots & \cdots & \cdots \\ w_{k1} & \cdots & w_{km} \end{bmatrix} + \begin{bmatrix} b_1 & \cdots & b_m \end{bmatrix}$$

n x m — n x k — k x m — 1 x m

Note that we add the bias to each row, essentially simulating an n x m matrix. That's how computers treat such matrices, that's why in machine learning we define the bias in this way.

Objective function

The objective function is a measure of how well our model's outputs match the targets.

The **targets** are the "correct values" which we aim at. In the cats and dogs example, the targets were the "labels" we assigned to each photo (either "cat" or "dog").

Objective functions can be split into two types: **loss** (supervised learning) and **reward** (reinforcement learning). Our focus is supervised learning.

Common loss functions

L2-norm loss
/regression/

$$\sum_i (y_i - t_i)^2$$

The L2-norm of a vector, **a**, (Euclidean length) is given by

$$\|a\| = \sqrt{a^T \cdot a} = \sqrt{a_1^2 + \cdots + a_n^2}$$

The main rationale is that the L2-norm loss is basically the distance from the origin (0). So, the closer to the origin is the difference of the outputs and the targets, the lower the loss, and the better the prediction.

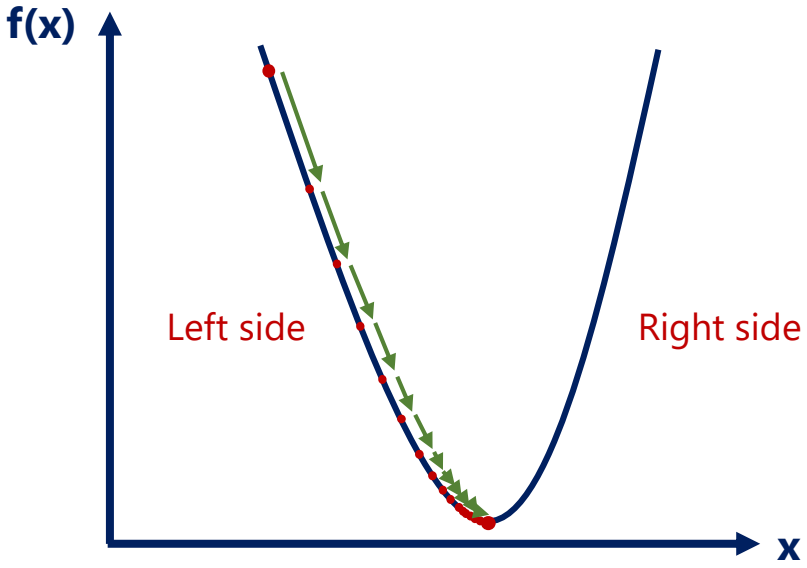
Cross-entropy loss
/classification/

$$-\sum_i \boldsymbol{t}_i \ln \boldsymbol{y}_i$$

The cross-entropy loss is mainly used for classification. Entropy comes from information theory, and measures how much information one is missing to answer a question. Cross-entropy (used in ML) works with probabilities – one is our opinion, the other – the true probability (the probability of a target to be correct is 1 by definition). If the cross-entropy is 0, then we are **not missing any information** and have a perfect model.

Gradient descent

The last ingredient is the optimization algorithm. The most commonly used one is the gradient descent. The main point is that we can find the minimum of a function by applying the rule: $x_{i+1} = x_i - \eta f'(x_i)$, where η is a small enough positive number. In machine learning, η, is called the learning rate. The rationale is that the first derivative at x_i , $f'(x_i)$ shows the slope of the function at x_i.



If the first derivative of f(x) at x_i , $f'(x_i)$, is negative, then we are on the left side of the parabola (as shown in the figure). Subtracting a negative number from x_i (as η is positive), will result in x_{i+1}, that is bigger than x_i. This would cause our next *trial* to be on the right; thus, closer to the sought minimum.

Alternatively, if the first derivative is positive, then we are on the right side of the parabola. Subtracting a positive number from x_i will result in a lower number, so our next trial will be to the left (again closer to the minimum).

So, either way, using this rule, we are approaching the minimum. When the first derivative is 0, we have reached the minimum. Of course, our update rule won't update anymore ($x_{i+1} = x_i - 0$).

The learning rate η, must be low enough so we don't oscillate (bounce around without reaching the minimum) and big enough, so we reach it in rational time.

In machine learning, f(x) is the **loss function**, which we are trying to minimize.

The variables that we are varying until we find the minimum are the **weights and the biases.** The proper update rules are:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \nabla_{\mathbf{w}} L(\boldsymbol{w}_i) = \boldsymbol{w}_i - \eta \sum_i \boldsymbol{x}_i \delta_i \quad \text{and} \quad b_{i+1} = b_i - \eta \nabla_b L(b_i) = b_i - \eta \sum_i \delta_i$$

Gradient descent. Multivariate derivation

The multivariate generalization of the gradient descent concept: $x_{i+1} = x_i - \eta f'(x_i)$ is given by: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \nabla_{\mathbf{w}} L(\boldsymbol{w}_i)$ (using this example, as we will need it). In this new equation, w is a matrix and we are interested in the gradient of w, w.r.t. the loss function. As promised in the lecture, we will show the derivation of the gradient descent formulas for the L2-norm loss divided by 2.

Model: $y = xw + b$

Loss: $L = \frac{1}{2}\sum_i (y_i - t_i)^2$

Update rule: $\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \nabla_{\mathbf{w}} L(\boldsymbol{w}_i)$

(opt. algorithm) $b_{i+1} = b_i - \eta \nabla_b L(b_i)$

Analogically, we find the update rule for the biases.

Please note that the division by 2 that we performed does not change the nature of the loss.

ANY function that holds the basic property of being higher for worse results and lower for better results can be a loss function.

$$\nabla_{\mathbf{w}} L = \nabla_{\mathbf{w}} \frac{1}{2} \sum_i (y_i - t_i)^2 =$$

$$= \nabla_{\mathbf{w}} \frac{1}{2} \sum_i \big((\boldsymbol{x}_i \boldsymbol{w} + b) - t_i\big)^2 =$$

$$= \sum_i \nabla_{\mathbf{w}} \frac{1}{2} (\boldsymbol{x}_i \boldsymbol{w} + b - t_i)^2 =$$

$$= \sum_i \boldsymbol{x}_i (\boldsymbol{x}_i \boldsymbol{w} + b - t_i) =$$

$$= \sum_i \boldsymbol{x}_i (y_i - t_i) \equiv$$

$$\equiv \sum_i \boldsymbol{x}_i \delta_i$$