DIABETES HEALTH CLASSIFICATOR (DHC)
A NEURAL NETWORK FOR DIABETES RISK PREDICTION

AGENDA

INTRODUCTION

1. Objective
   a. Develop a neural network to classify individuals as diabetic or non-diabetic based on health indicators.
2. Dataset
   a. Source: Diabetes Health Indicators Dataset (Kaggle)
   b. Size: 70,693 records
   c. Features: 21 variables
      - Demographics: Age, Sex
      - Behavior: Smoking, Physical Activity
      - Health Conditions: High Blood Pressure, High Cholesterol
      - Medical Indicators: BMI, Depressio
3. Model Goals
   a. Build an accurate and generalizable predictive model
   b. Analyze feature importance
   c. Visualize training behavior and performance metrics
4. Implementation
   a. Language: Python
   b. Libraries: NumPy, Pandas, Machine Learning tools
5. Task Type:
   a) Binary Classification: Diabetic vs Non-Diabetic
6. Relevance
   a. Early diabetes detection can significantly improve clinical outcomes.

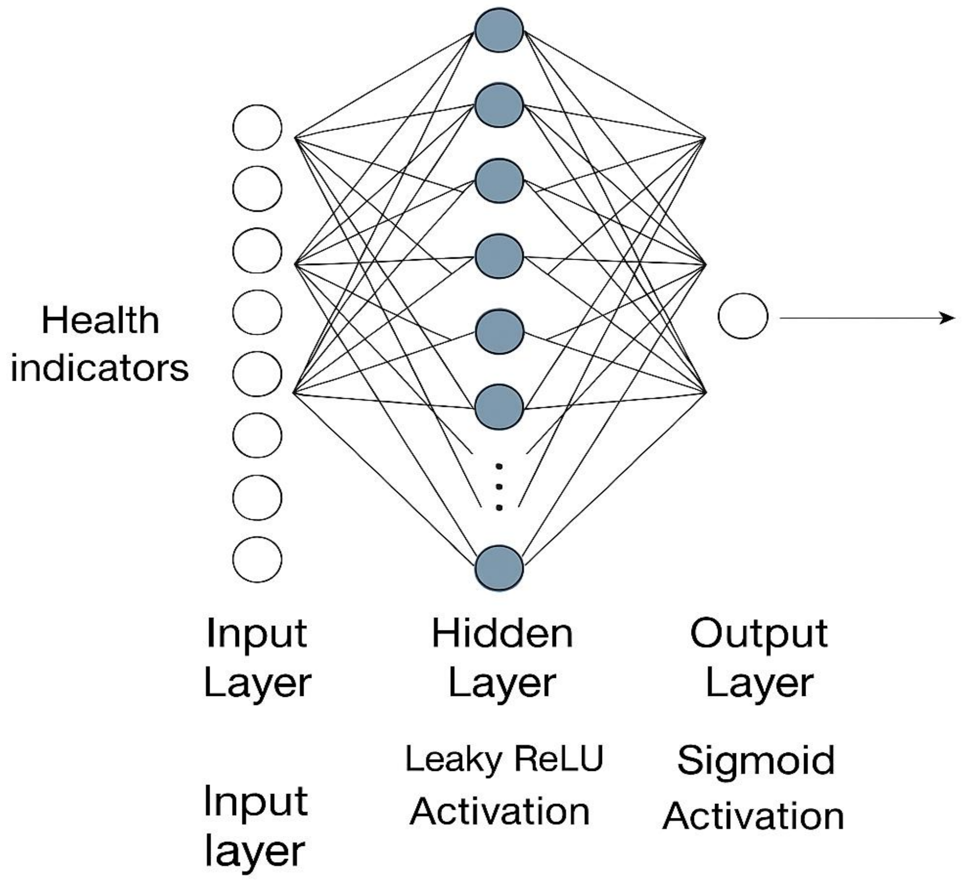
ARCHITECTURE AND COMPONENTS OF DHC NEURAL NETWORK

ARCHITECTURE AND COMPONENTS OF DHC NEURAL NETWORK

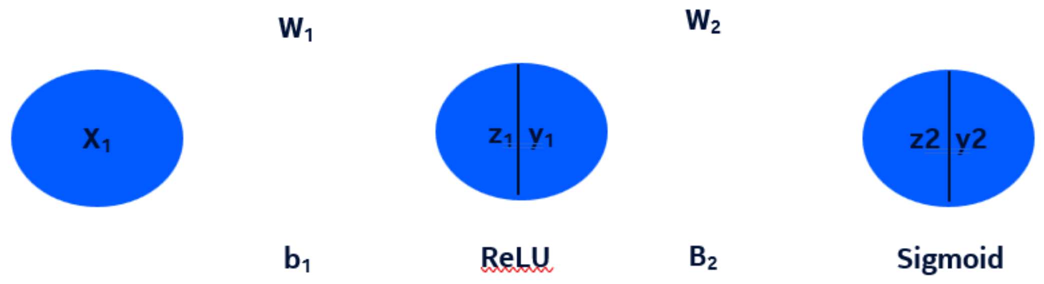
- The neural network designed for the Diabetes Health Classifier (DHC) project is a simple but effective feedforward network for a binary classification task (presence or absence of diabetes).

Layer	Number of Neurones	Activaction function
Input layer	21	None
Hidden layer	64	Leaky ReLU
Output layer	1	Sigmoid

ARCHITECTURE AND COMPONENTS OF DHC NEURAL NETWORK



- DHC graphical and logical rapresentation



ARCHITECTURE AND COMPONENTS OF DHC NEURAL NETWORK

- Activation function: ReLU Leaky, which introduces non-linearity

$$f(x) = \begin{cases} x \; se \; x > 0 \\ 0.01x \; se \; x \leq 0 \end{cases}$$

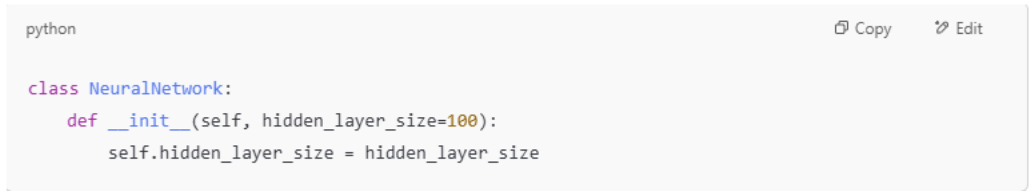
- Activation function: Sigmoid, which converts the output into a probability between 0 and 1, suitable for binary classification (diseased / not diseased)

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

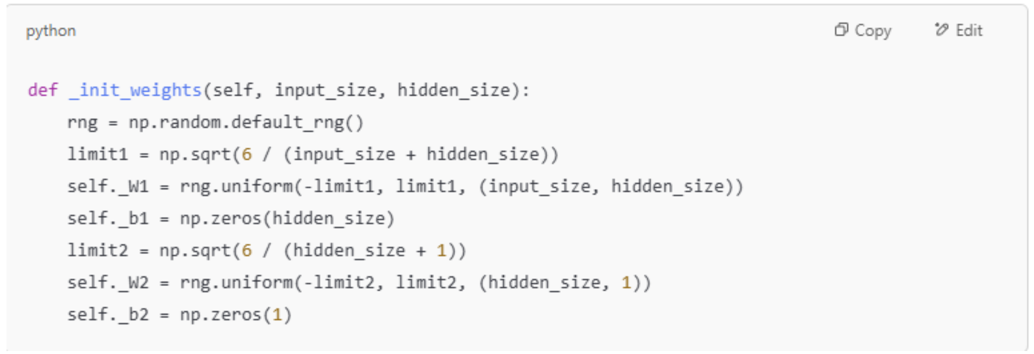
- Loss function: Logarithmic Loss o Binary Cross-Entropy Loss performs a sort of probabilistic weighted average between the output values of the neural network $\hat{y}_i$ with respect to the real values y_i and takes into account the fact that what it predicts and what is to be predicted are probabilities between zero and one.

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

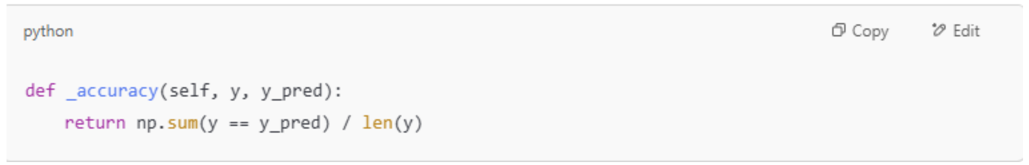
ARCHITECTURE AND COMPONENTS OF DHC NEURAL NETWORK



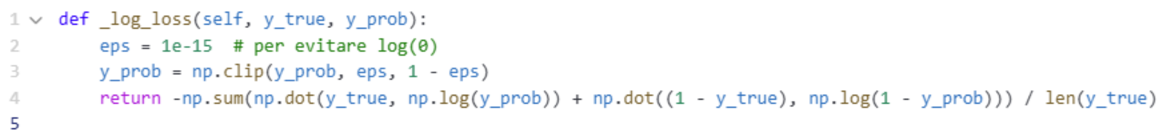
```python
class NeuralNetwork:
    def __init__(self, hidden_layer_size=100):
        self.hidden_layer_size = hidden_layer_size
```



```python
def _init_weights(self, input_size, hidden_size):
    rng = np.random.default_rng()
    limit1 = np.sqrt(6 / (input_size + hidden_size))
    self._W1 = rng.uniform(-limit1, limit1, (input_size, hidden_size))
    self._b1 = np.zeros(hidden_size)
    limit2 = np.sqrt(6 / (hidden_size + 1))
    self._W2 = rng.uniform(-limit2, limit2, (hidden_size, 1))
    self._b2 = np.zeros(1)
```



```python
def _accuracy(self, y, y_pred):
    return np.sum(y == y_pred) / len(y)
```



```python
def _log_loss(self, y_true, y_prob):
    eps = 1e-15  # per evitare log(0)
    y_prob = np.clip(y_prob, eps, 1 - eps)
    return -np.sum(np.dot(y_true, np.log(y_prob)) + np.dot((1 - y_true), np.log(1 - y_prob))) / len(y_true)
```

- Main functions implemented
  - *_init_* : Constructs the "neural network" object, initializing the parameter that defines the number of neurons in the hidden layer
  - *_init_weights* : Initialize the weights (_W1, _W2) and biases (_b1, _b2) randomly, using Xavier initialization (uniform over a range depending on the number of inputs and outputs)
  - *_accurancy* : Calculate the percentage of correct predictions. Compare element by element y and y_pred. Count how many values match. Divide the number of successes by the total number of samples
  - *_log_loss* : Loss function implementation

INPUT DATA

INPUT DATA

- The dataset used to train and validate the Diabetes Health Classifier (DHC) neural network is publicly available on the Kaggle platform. The specific file used is *diabetes_binary_5050split_health_indicators_BRFSS2015.csv*, containing:
    - Over 70,693 samples (instances)
    - 21 features for each individual
    - Binary label (Diabetes_binary) as target
- Features include information on:
    - Medical indicators (e.g. high blood pressure, high cholesterol, body mass index)
    - Lifestyle (e.g. smoking, physical activity)
    - Psychological conditions (e.g. depression)

INPUT DATA

- Preprocessing. To prepare the data for neural network training, two main preprocessing operations were performed:
  - Splits the dataset into a training set (70%) and a validation set (30%), based on a specified percentage. Training set is used for training phase, validation set is used for validation phase. Validation is used to verify that the training carried out with the training set does not generate overfitting, that is, the network is too attached to the data in the training set, or that it only knows how to classify the data in the training set
  - Normalization
    - When training a neural network (or a machine learning algorithm in general), the input data must be scaled and normalized. Main reason: different feature scales can slow down, prevent or damage learning. *Min-Max* normalization scales each feature in the range [0, 1]:

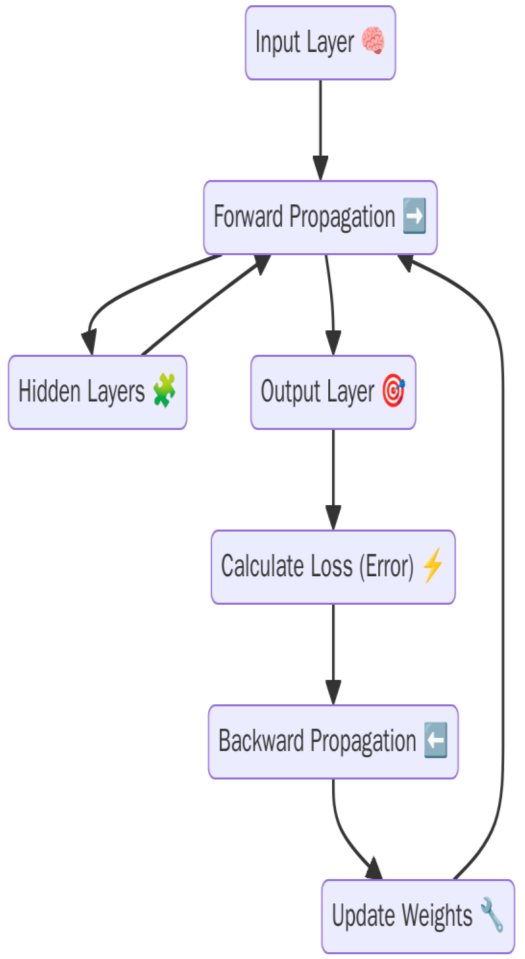
$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

    - Summarizing: stable learning, balanced gradients, better generalization, all features treated equally

TRAINING PROCESS

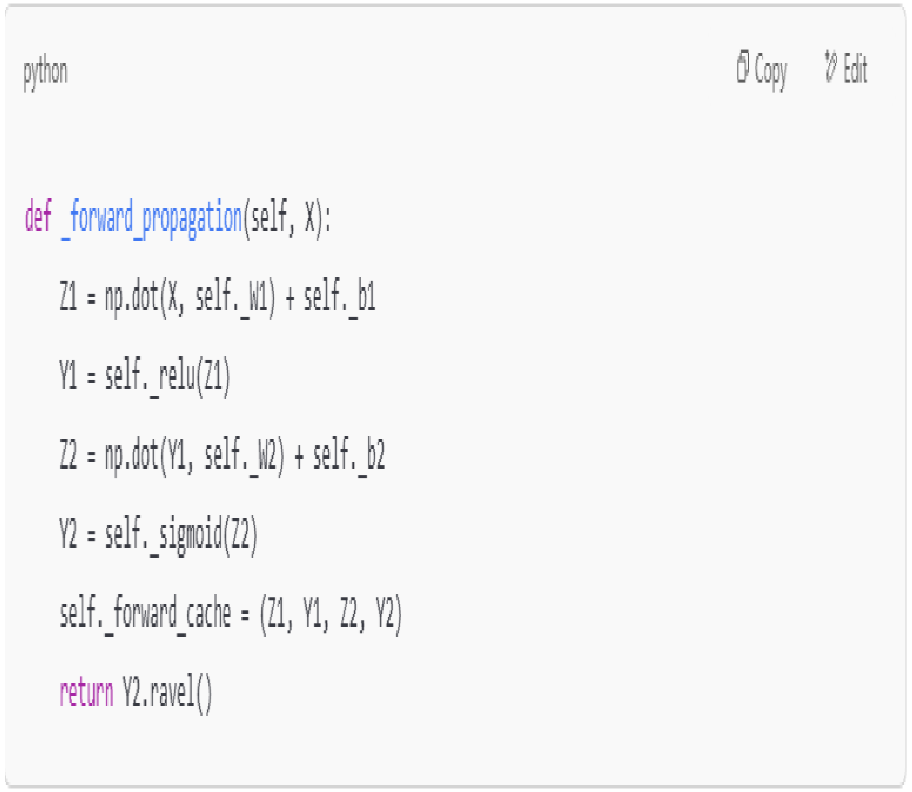
TRAINING PROCESS

- The training of the Diabetes Health Classifier (DHC) neural network occurs through two main phases:
  - Forward Propagation
  - Backward Propagation
- Training cycle according following parameters:
  - epochs: number of iterations on the entire dataset
  - lr: learning rate (how quickly to update the weights, the higher the value of lr, the stronger the push in the descent of the gradient)



TRAINING PROCESS

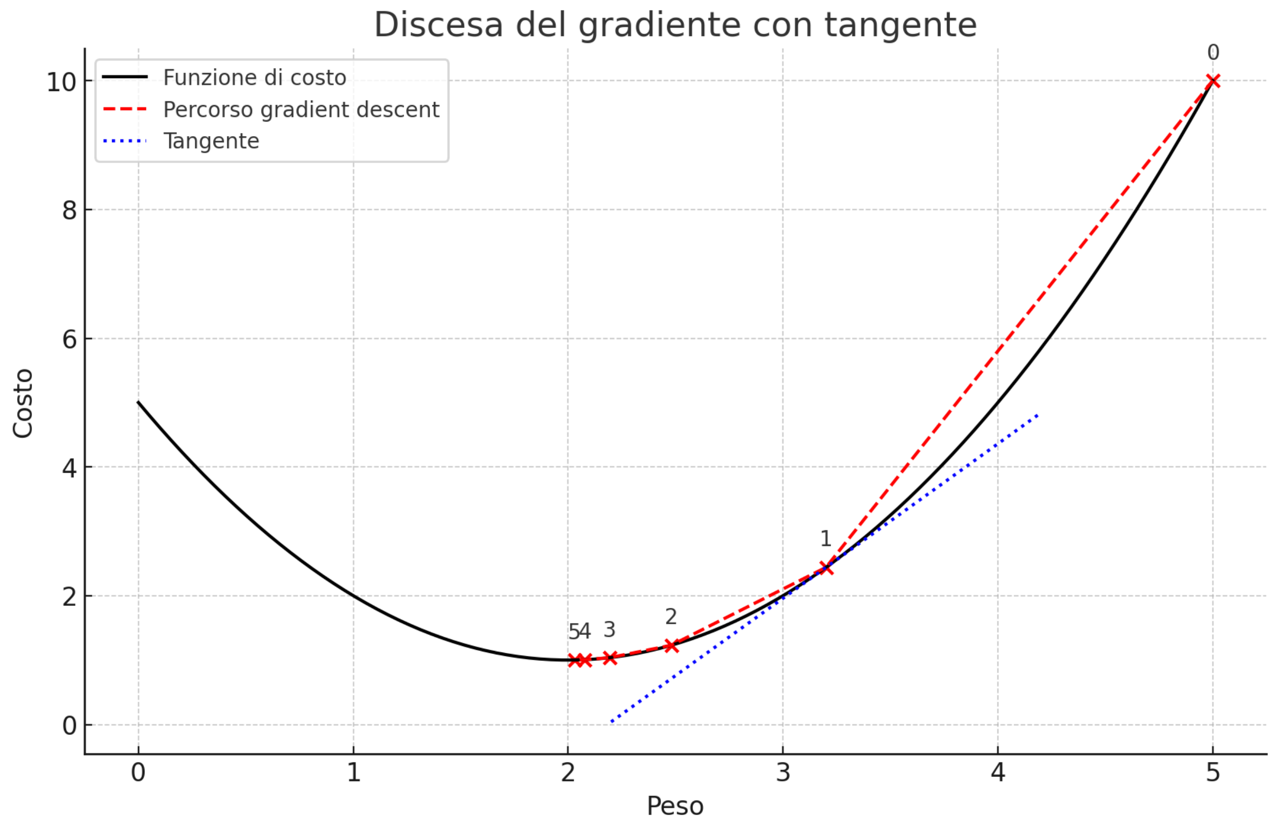
- Forward propagation
  - It computes the complete forward pass from input to output, that is, it makes an approximate prediction of the value to be attributed to each single input array.
    - Multiply input × weights + bias.
    - Apply Leaky ReLU.
    - Multiply hidden output × weights + bias.
    - Apply Sigmoid.
    - Return the vector of predicted probabilities
  - This prediction will inevitably be approximate: however, it can be optimized. How is optimization performed?



```python
def _forward_propagation(self, X):
    Z1 = np.dot(X, self._W1) + self._b1
    Y1 = self._relu(Z1)
    Z2 = np.dot(Y1, self._W2) + self._b2
    Y2 = self._sigmoid(Z2)
    self._forward_cache = (Z1, Y1, Z2, Y2)
    return Y2.ravel()
```

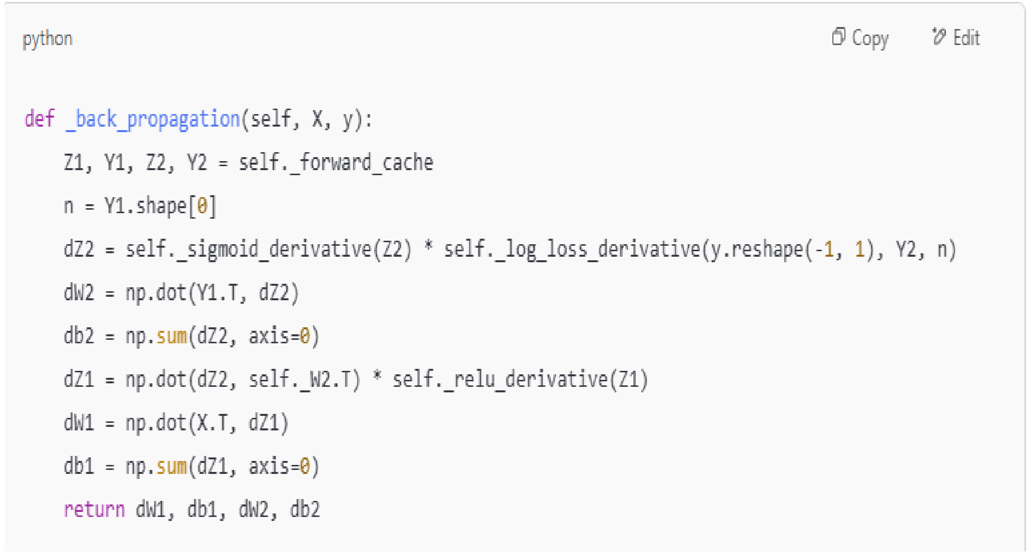
TRAINING PROCESS

- Backward propagation
  - First calculating with back propagation how much to correct the weights and biases,
  - Second carrying out the correction with the formula old value = old value - learning rate * correction rate
  - This iterative process of correcting the weight and threshold matrices, i.e. optimizing the prediction, is called gradient descent:



TRAINING PROCESS

- Backward propagation
  - Calculate the gradients (i.e. partial derivatives of the loss function with respect to the weights and biases) to update the neural network and minimize the error. This function works after the forward pass and prepares everything for the weight update step.
  - Output
    - dW1: Gradient of the weights of the first layer.
    - db1: Gradient of the thresholds of the first layer.
    - dW2: Gradient of the weights of the second layer.
    - db2: Gradient of the thresholds of the second layer.



```python
def _back_propagation(self, X, y):
    Z1, Y1, Z2, Y2 = self._forward_cache
    n = Y1.shape[0]
    dZ2 = self._sigmoid_derivative(Z2) * self._log_loss_derivative(y.reshape(-1, 1), Y2, n)
    dW2 = np.dot(Y1.T, dZ2)
    db2 = np.sum(dZ2, axis=0)
    dZ1 = np.dot(dZ2, self._W2.T) * self._relu_derivative(Z1)
    dW1 = np.dot(X.T, dZ1)
    db1 = np.sum(dZ1, axis=0)
    return dW1, db1, dW2, db2
```

RESULTS AND THEIR ANALYSIS

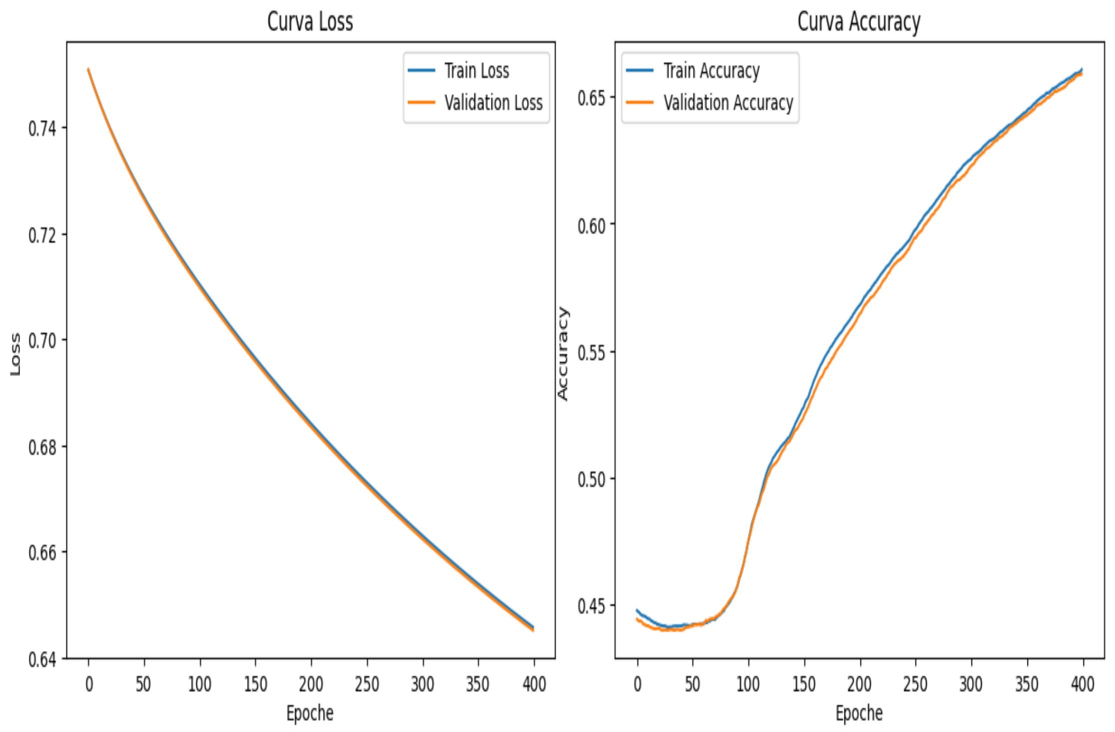
RESULTS AND THEIR ANALYSIS

- The evaluate function is used to evaluate the performance of the model on a given data set (X, y), calculating two key metrics:
    - percentage of correct classifications (Accurancy)
    - and the measure of the goodness of the predicted probabilities, strongly penalizing incorrect and unconfident predictions (Log Loss).
- It is used after training to measure how well the model has learned, both in terms of classification and probabilistic reliability.

```
def evaluate(self, X, y):
    y_pred, proba = self.predict(X, classify=True)
    accuracy = self._accuracy(y, y_pred)
    log_loss = self._log_loss(y, proba)
    return (accuracy, log_loss)
```

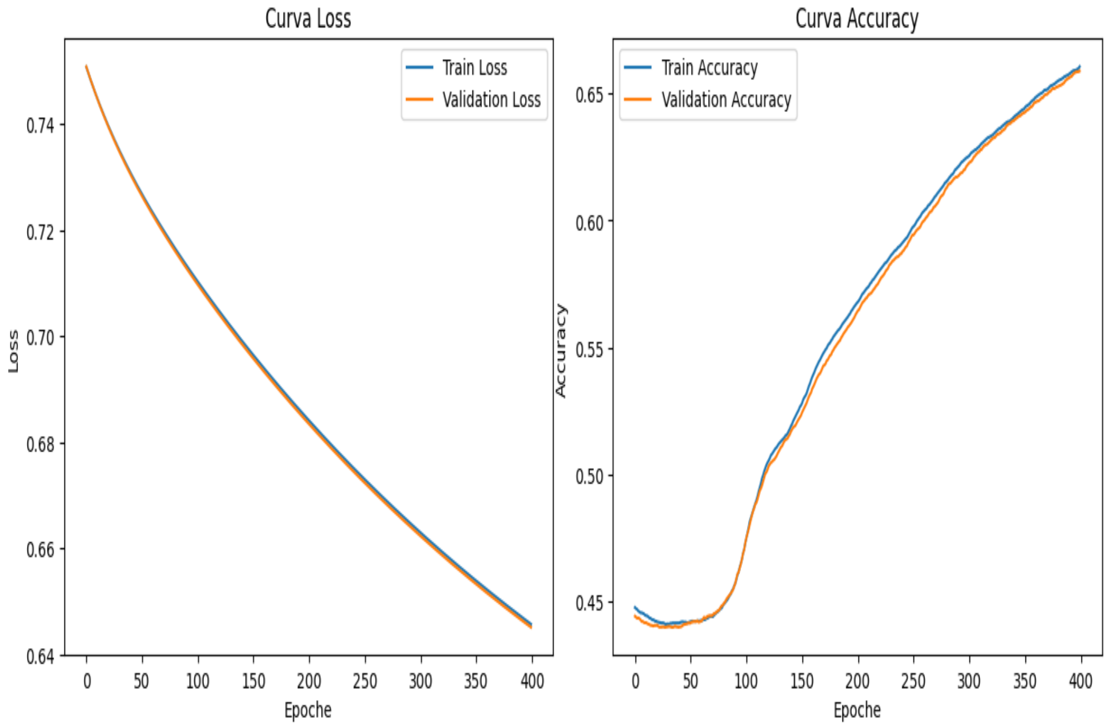
RESULTS AND THEIR ANALYSIS

- X-axis:
  - Represents the number of epochs: one epoch corresponds to a complete pass through the entire training dataset.
  - The graph shows 400 training epochs.
- Y-axis:
  - On the left in the "Loss Curve" graph: value of the calculated loss (loss function).
  - On the right in the "Accuracy Curve" graph: value of the accuracy (percentage of correct classifications).



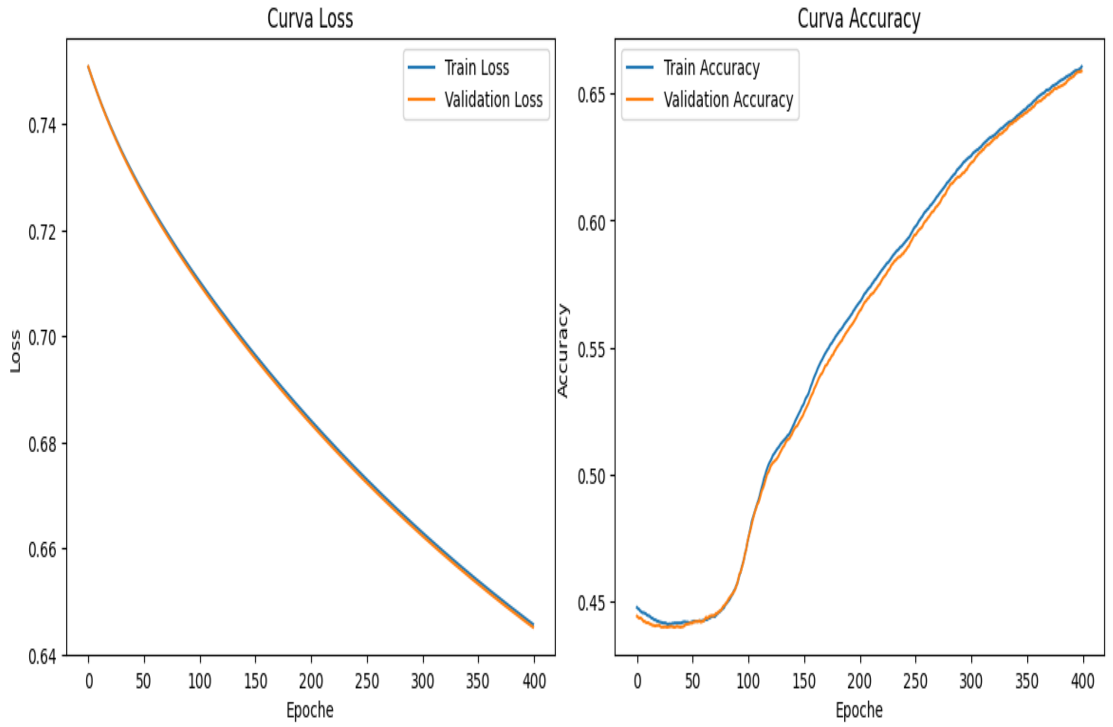
RESULTS AND THEIR ANALYSIS

- Loss Curve Analysis
  - Trend: the loss curve decreases continuously and regularly for both the training set and the validation set.
  - Interpretation: a decreasing loss indicates that the network is learning correctly to reduce the predictive error.
  - Behavior between training and validation: the close overlap of the training and validation curves suggests that the model is not overfitting (i.e. it does not fit the training data too well).



RESULTS AND THEIR ANALYSIS

- Accuracy Curve Analysis
  - Trend: accuracy increases gradually as the epochs progress.
  - Initial low: accuracy is initially low (around 44%) but begins to improve significantly after around 100 epochs.
  - Continuous progression: accuracy increases constantly until it exceeds 65% at the end of training.
  - Train vs Validation: here too, the training and validation curves are very close, indicating good generalization.



RESULTS AND THEIR ANALYSIS

- Accuracy: How many times the model guessed right out of the total. And the percentage of correct predictions/classifications across the entire dataset (both 0 and 1). The value of 67.85% means that the network correctly classifies about 2 out of 3 examples, either as "diabetic" or "non-diabetic".
- Log Loss: Log loss (logarithmic loss or binary cross-entropy) is a metric that measures how correct and confident the probabilities predicted by the model are. A value like 0.6427 indicates generally correct behavior, but with some uncertainty in the predicted probabilities, suggesting that the model can still improve in its "confidence".

Metric	Value
Accurancy	0.6785
Log Loss	0.64

RESULTS AND THEIR ANALYSIS

Classe	Precision	Recall	F1-score	Supporto
0.0 (non diabetico)	0.70	0.63	0.66	10.659
1.0 (diabetico)	0.66	0.72	0.69	10.548

- Precision
  - When the model says 'yes', how many times is it right?
  - Precision = True Positive / True Positive + False Positive
  - Indicate how many of the predicted positives are correct.
- Recall (Sensitivity)
  - How many times does the model actually recognize positive cases?
  - Recall = True Positive / True Positive + False Negative
  - Indicates how many of all the true positives were captured.
- F1-score
  - Balance precision and recall in a single number.
  - It is the harmonic mean between precision and recall.
  - It is used to rebalance models that perhaps have high precision but low recall (or vice versa).

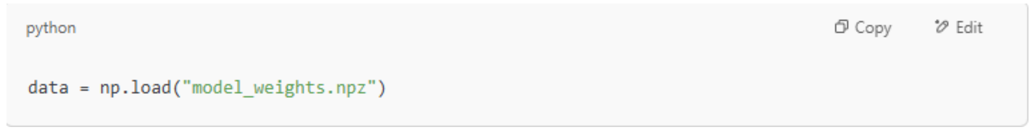
RESULTS AND THEIR ANALYSIS

- Possible improvements:
  - Architecture optimization: add more hidden layers or vary the number of neurons.
  - Regularization techniques: dropout, L2 penalty to further reduce possible overfitting.
  - Data augmentation: data augmentation or additional data collection to improve generalization.
  - Parameter fine-tuning: lower learning rate or use of scheduler.

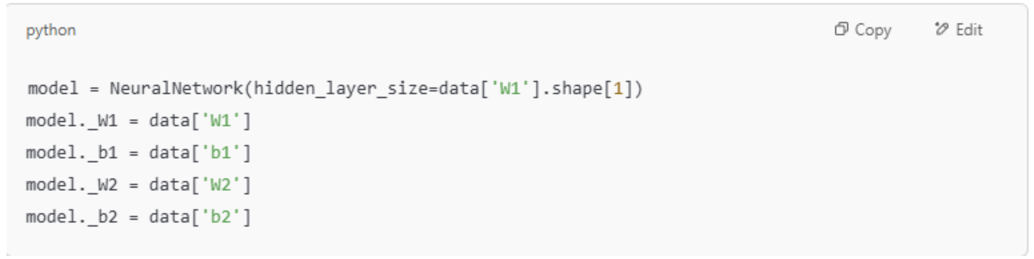
TESTING PHASE

TESTING PHASE

- Preliminary actions:
    - Preparing a new script that loads the model saved during the training phase
    - Rebuild the model (first layer weights and biases, second layer weights and biases)
    - Normalization is also reloaded, ensuring that the test data is processed in the same numerical space as the training data.
    - Preparing 2 different dataset for testing: 2000 cases dataset and 5000 cases dataset



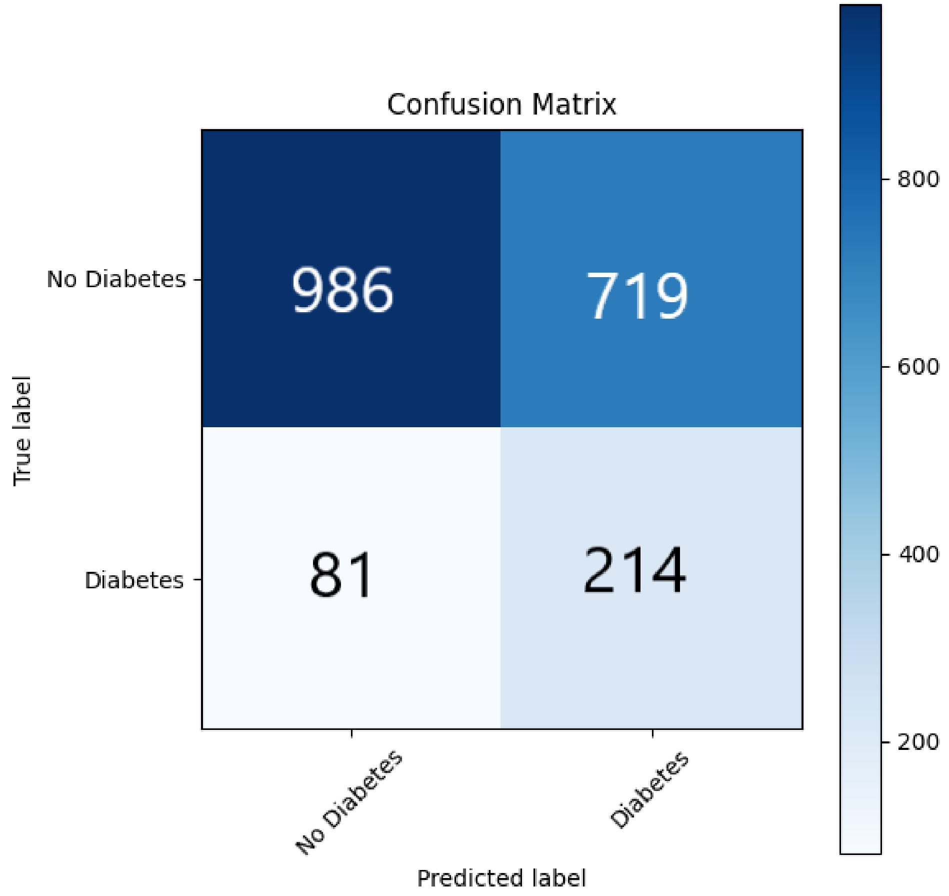
```python
data = np.load("model_weights.npz")
```



```python
model = NeuralNetwork(hidden_layer_size=data['W1'].shape[1])
model._W1 = data['W1']
model._b1 = data['b1']
model._W2 = data['W2']
model._b2 = data['b2']
```

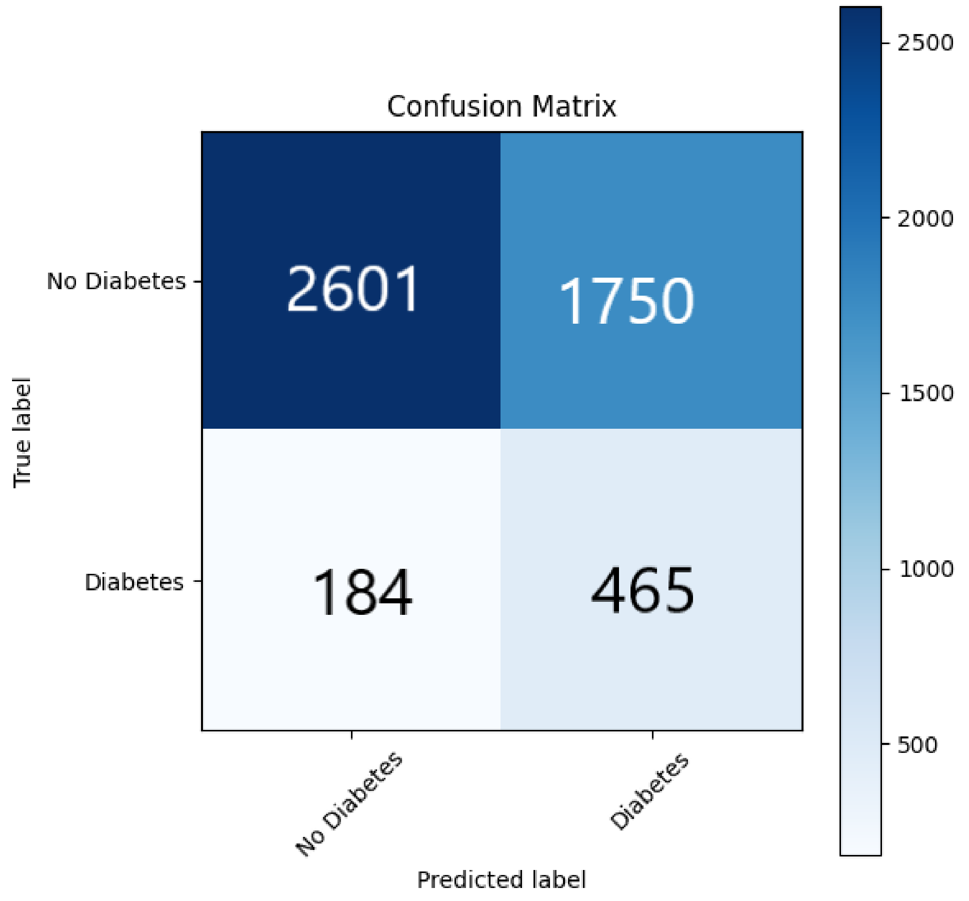
TESTING PHASE

- 2000 samples. Confusion matrix:
  - True positive (TP = 214): the model correctly identified 214 diabetic patients.
  - False negative (FN = 81): it missed 81 real diabetics → this is the most critical data in medicine.
  - False positive (FP = 719): it predicted diabetes in 719 healthy people.
  - True negative (TN = 986): it correctly identified 986 non-diabetic people.
- Recall diabete = 214 / (81 + 214) = 73%
- Precision diabete = 214 / (214 + 719) ≈ 23%
- Accuracy totale = (986 + 214) / 2000 = 60%



TESTING PHASE

- 5000 samples. Confusion matrix:
  - True positive (TP = 465): the model correctly identified 465 diabetic patients.
  - False negative (FN = 184): it missed 184 real diabetics → this is the most critical data in medicine.
  - False positive (FP = 1750): it predicted diabetes in 1750 healthy people.
  - True negative (TN = 2601): it correctly identified 2601 non-diabetic people.
- Recall diabete = 465 / (184 + 465) = 72%
- Precision diabete = 465 / (465 + 1750) ≈ 21%
- Accuracy totale = (2601 + 465) / 5000 = 61%



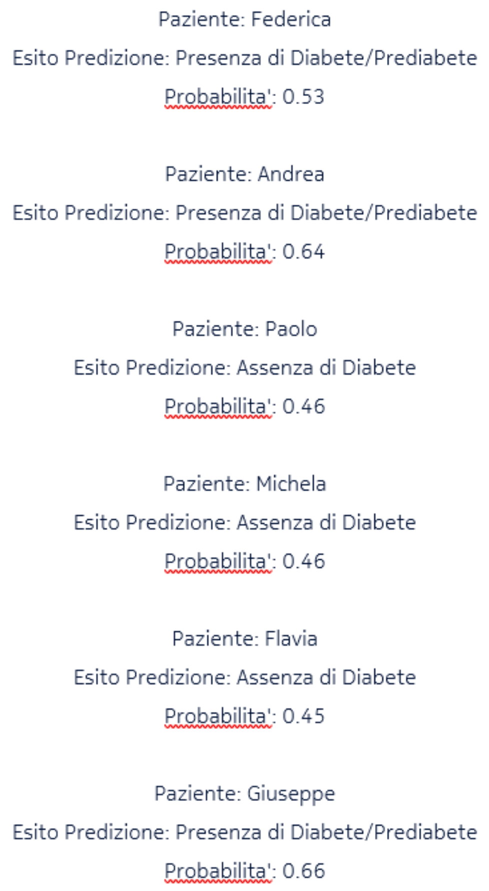
PREDICTION ON NEW PATIENTS

PREDICTION ON NEW PATIENTS

- After the Diabetes Health Classifier (DHC) neural network was trained and validated, the model was used to make predictions on 20 new patients never seen before.
- The goal was to verify the model's ability to correctly generalize to unknown real data.
- Preliminary actions:
    - Preparing a new script that loads the model saved during the training phase
    - Rebuild the model (first layer weights and biases, second layer weights and biases)
    - Normalization is also reloaded, ensuring that the test data is processed in the same numerical space as the training data.
    - The dataset contains 20 real patients, each described by 21 health indicators (the same ones used for training), EXCEPT the true label data.

PREDICTION ON NEW PATIENTS

- Overall summary:
  - Total number of patients analyzed: 20
  - Predicted patients at risk (diabetes/prediabetes): 12
  - Predicted healthy patients: 8
  - Percentage of patients at risk: 60.0%
  - Mean predicted probability at risk: 0.60



```
Paziente: Federica
Esito Predizione: Presenza di Diabete/Prediabete
Probabilita': 0.53

Paziente: Andrea
Esito Predizione: Presenza di Diabete/Prediabete
Probabilita': 0.64

Paziente: Paolo
Esito Predizione: Assenza di Diabete
Probabilita': 0.46

Paziente: Michela
Esito Predizione: Assenza di Diabete
Probabilita': 0.46

Paziente: Flavia
Esito Predizione: Assenza di Diabete
Probabilita': 0.45

Paziente: Giuseppe
Esito Predizione: Presenza di Diabete/Prediabete
Probabilita': 0.66
```

CONCLUSION

CONCLUSION

- In this project, a feedforward neural network called Diabetes Health Classifier (DHC) was developed, trained and validated with the aim of supporting the early identification of subjects at risk of diabetes.
- The main stages of the work were:
  - Definition of the network architecture (21 inputs, 64 hidden layer neurons, 1 output).
  - Data preprocessing: data normalization and subdivision into training/validation.
  - Training: performed on over 70,000 real cases, monitoring loss curve and accuracy.
  - Evaluation: obtained good global metrics (accuracy around 68%).
  - Testing on new datasets: application of the model to data never seen before (2000 and 5000 cases), confirming the ability to generalize without evident overfitting.
  - Prediction on real patients: tests on 20 new real cases, demonstrating good sensitivity in identifying subjects at risk.

CONCLUSION

- Although the results are satisfactory, areas for improvement have emerged:
  - Improved accuracy:
    - Optimization of the architecture (number of layers, number of neurons).
    - Adoption of regularization techniques such as Dropout or L2 penalty.
  - Management of data imbalance:
    - Implementation of oversampling or undersampling techniques.
    - Balancing of classes in the training data.
  - Customization of classification thresholds:
    - Adaptation of the decision threshold to optimize precision or recall depending on clinical needs.
  - Integration of new clinical data:
    - Inclusion of additional medical parameters to refine the model (e.g. blood tests, family history).
  - Development of an application interface:
    - Creation of a web or mobile application to use the model in the clinical setting as a rapid screening tool.

CONCLUSION

- Detailed Technical Specification Document (TSD) is available: *Diabetes Health Classificator (DHC). Una rete neurale per la predizione del rischio di diabete,* Corrado Vaccaro, 1° edition, 2025-05-12
- The document includes the description of the architecture of DHC, the implemented functions, scripts regarding training, validation, testing and prediction, several dataset in csv format and exaustive explanation regarding learning curves and metrics. The document is Italian language.

THANK YOU

Corrado Vaccaro

Senior Customer Program Manager, PMP®

corrado.vaccaro@gmail.com