# PROGETTO LINGUAGGI FORMALI E COMPILATORI

Andrea Tresoldi [redacted]

Università di Bergamo, Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

4 settembre 2023

## 1 Json To Neural Networks - Presentazione

### 1.1 Scenario e decisore

Lo scenario che ho immaginato è quello di presentare l'applicazione "Json To Neural Networks" al Ministro dell'istruzione, con l'obiettivo, poi, di renderla disponibile per l'utilizzo nelle scuole superiori con indirizzo informatico. Essendo l'applicativo pensato e progettato per utenti che non hanno nessuna conoscenza pregressa in ambito machine learning, questa applicazione potrebbe essere una potente risorsa da fornire ai professori per introdurre gli studenti al mondo intelligenza artificiale, competenze che saranno sicuramente utili per il loro futuro professionale.

### 1.2 Introduzione

Onorevole Ministro, è un onore essere qui oggi per presentarvi una straordinaria opportunità che può potenziare notevolmente l'insegnamento dell'informatica nelle nostre scuole superiori: "Json To Neural Networks". Immaginate di poter introdurre facilmente nei programmi didattici argomenti avanzati di machine learning, fornendo agli studenti delle nozioni strutturate su argomenti attuali ed innovativi come le reti neurali.

### 1.3 Problema

Nel panorama educativo attuale, il mondo dell'IA e delle rete neurali viene introdotto agli studenti solamente in corsi di laurea magistrale come ingegneria e statistica. In un momento storico come questo, in cui il mondo dell'apprendimento automatico si sta espandendo in modo esponenziale, ritengo sia opportuno ed importante avvicinare gli studenti a queste tematiche in modo progressivo e costante nel tempo al fine di formare futuri professionisti del settore con competenze solide e ben strutturate. Ma come possiamo integrare argomenti complessi ed avanzati nella didattica superiore superando l'ostacolo della complessità della programmazione di reti neurali in python?

### 1.4 Soluzione

"Json To Neural Networks" è nata proprio come soluzione a questo problema. L'applicativo consente agli studenti di progettare e sperimentare differenti architetture di reti neurali in modo intuitivo e flessibile senza la necessità di conoscere ambienti e librerie di sviluppo python

tipicamente utilizzate per lo sviluppo di modelli di apprendimento automatico. Partendo da un semplice documento in ingresso in cui vengono specificati alcuni parametri dell'architettura di rete, l'applicativo è in grado di generare in modo automatico tutto il codice necessario per poter allenare una rete neurale a risolvere un task specifico.

### 1.5 Vantaggi

I vantaggi nell'usare questo applicativo sono molteplici: prima di tutto, come già accennato, non è necessario scrivere nessuna riga di codice, un lavoro che richiede competenze che studenti di scuola superiore non possiedono. Questo è un aspetto fondamentale in quanto consente agli utenti di concentrarsi esclusivamente sulla comprensione dei concetti base su cui si fonda la disciplina. Inoltre grazie ai numerosi controlli semantici presenti, ogni volta che vengono inseriti nel documento di input errori concettualmente significativi, questi vengono rilevati indicando la correzione più appropriata.

### 1.6 Conclusioni

Per concludere, "Json To Neural Networks" è lo strumento e l'invesitmento giusto per rendere l'intelligenza artificiale accessibile a tutti gli studenti, permettendo loro di sviluppare fin da subito competenze chiave per il futuro e per formare una generazione pronta ad affrontare queste tematiche sul mondo del lavoro.

## 2 Documentazione di Progetto

### 2.1 Tecnologie utilizzate

Le principali teconologie che sono state utilizzate per lo sviluppo dell'applicazione sono le seguenti:

- ANTLRWorks: è un'ambiente di sviluppo integrato per ANTLR che permette di progettare lexer e parser Java partendo da specifiche grammaticali, nel nostro caso semplici espressioni regolari.
- Eclipse e Java: l'applicazione è stata sviluppata utilizzando Eclipse come IDE di sviluppo e Java come linguaggio di programmazione principale.

### 2.2 Lessico, grammatica non decorata e decorata

I token del nostro linguaggio e i relativi identificatori associati sono i seguenti:

- token del linguaggio:

```
// key
TASK :'"task"';
HIDDEN_LAYERS :'"hidden_layers"';
NUMBER_OF_NEURONS :'"neurons"';
ACTIVATION_FUNCTION :'"activation_function"';
OUTPUT_LAYER :'"output_layer"';
NUMBER_OF_EPOCHS :'"epochs"';
EARLY_STOPPING :'"early_stopping"';
LEARNING_RATE :'"learning_rate"';
```

```
LOSS_FUNCTION :'"loss_function"';

fragment
ESC_SEQ :'\\'('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\');

fragment
DIGIT :'0'..'9';

//value
INTEGER :'-'? (DIGIT)+;
FLOAT :(INTEGER) DOT DIGIT+;
STRING                  :'"'(ESC_SEQ|~('\\'|'"'))*'"';
BOOLEAN :'true'|'false';
NULL :'null';

// punctuation
CL :':';
CM  :',';
DOT  :'.';

// parenthesis
LB :'[';
RB :']';
LBR :'{';
RBR :'}';

//white spaces
WS :(' '|'\t'|'\r'|'\n')+{$channel=HIDDEN;};

//token error
ERROR_TK :.;
```

- identificatori:

```
ACTIVATION_FUNCTION=4
BOOLEAN=5
CL=6
CM=7
DIGIT=8
DOT=9
EARLY_STOPPING=10
ERROR_TK=11
ESC_SEQ=12
FLOAT=13
HIDDEN_LAYERS=14
INTEGER=15
LB=16
LBR=17
LEARNING_RATE=18
LOSS_FUNCTION=19
NULL=20
```

```
NUMBER_OF_EPOCHS=21
NUMBER_OF_NEURONS=22
OUTPUT_LAYER=23
RB=24
RBR=25
STRING=26
TASK=27
WS=28
```

Invece, per quanto concerne la grammatica sintattica (grammatica non decorata) e le relative azioni semantiche (grammatica decorata)[1]:

- grammatica decorata e non decorata:

```
parseJson
@init {initParser();}
:
principalObjectRule
;

principalObjectRule
:
LBR
taskType = taskRule
CM
hiddenLayersRule[taskType]
CM
outputLayersRule[taskType]
CM
numberOfEpochs=numberOfEpochsRule
CM
earlyStopping = earlyStoppingRule
CM
learningRateRule
CM
lossFunctionRule[taskType, earlyStopping, numberOfEpochs]
RBR
;

taskRule returns[Token taskType]
:
TASK CL task=STRING
{taskType=$task; h.checkTask(taskType);}
;

hiddenLayersRule[Token taskType]
:
hiddenLayers=HIDDEN_LAYERS CL
LB
```

[1]Per ulteriori dettagli ed informazioni inerenti alla grammatica di progetto visionare i documenti "Json.g" e "Json.token."

```
numberOfNeuron=layerRule[taskType, $hiddenLayers, null]
(CM numberOfNeuron=layerRule[taskType, $hiddenLayers, numberOfNeuron])*
RB
;

outputLayersRule[Token taskType]
:
outputLayer=OUTPUT_LAYER CL layerRule[taskType, $outputLayer, null]
;

layerRule[Token taskType, Token layerType, Token precNumberOfNeurons]
returns[Token numberOfNeurons]
:
LBR
NUMBER_OF_NEURONS CL nOfN=INTEGER CM
{numberOfNeurons=$nOfN; h.checkNeurons(numberOfNeurons, layerType);}
{h.checkDecreasingNeuron(precNumberOfNeurons, numberOfNeurons);}
ACTIVATION_FUNCTION CL activationFunction=STRING
{h.checkActivationFunction(layerType, taskType, $activationFunction);}
RBR
{h.buildLayer(numberOfNeurons, layerType, $activationFunction);}
;

numberOfEpochsRule returns[Token numberOfEpochs]
:
NUMBER_OF_EPOCHS CL nOfE=INTEGER
{numberOfEpochs=$nOfE; h.checkNumberOfEpochs(numberOfEpochs);}
;

earlyStoppingRule returns[Token earlyStopping]
:
EARLY_STOPPING CL eStopping=BOOLEAN
{earlyStopping=$eStopping;}
;

learningRateRule
:
LEARNING_RATE CL learningRate=FLOAT
{h.checkLearningRate($learningRate);}
;

lossFunctionRule[Token taskType, Token earlyStopping,
Token numberOfEpochs]
:
LOSS_FUNCTION CL lossFunction=STRING
{h.checkLossFunction(taskType, $lossFunction, earlyStopping,
numberOfEpochs);}
;
```

## 2.3 Controlli semantici

Come è possibile osservare nella sezione precedente, oltre ai controlli lessicali e sintattici realizzati da lexer e parser, sul documento in ingresso vengono eseguite numerose azioni e controlli semantici cablati all'interno dell'analizzatore sintattico stesso. I principali sono:

- `h.checkTask(taskType)`:

  controlla se il task che si desidera eseguire con la rete neurale è appropriato e supportato dall'applicativo.

- `h.checkNeurons(numberOfNeurons, layerType)`:

  controlla se il numero di neuroni che si vogliono istanziare all'interno di ogni layer è corretto.

- `h.checkDecreasingNeuron(precNumberOfNeurons, numberOfNeurons)`:

  controlla se il numero di neuroni all'interno di ogni layer è minore uguale al numero di neuroni all'interno del layer precedente.

- `h.checkActivationFunction(layerType, taskType, $activationFunction)`:

  controlla, in funzione del tipo di layer, se le funzioni di attivazione selezionate sono adeguate e coerenti con il task selezionato.

- `h.checkNumberOfEpochs(numberOfEpochs)`:

  controlla se il numero di epoche è maggiore di 49.

- `h.checkLearningRate($learningRate)`:

  controlla se il valore di learning rate utilizzato durante l'addestramento è maggiore di zero.

- `h.checkLossFunction(taskType, $lossFunction, earlyStopping, numberOfEpochs)`:

  controlla se la funzione di costo da ottimizzare è compatibile con il task che la rete deve imparare a risolvere.

Tutti questi metodi sono necessari per controllare e verificare la correttezza semantica del documento di input contenete la specifica della rete che si vuole implementare.

## 2.4 Diagramma UML

Nel diagramma UML mostrato qui sotto, sono presentate le classi principali dell'applicazione:


Figura 1: Diagramma UML dell'applicazione

- Le classi JsonLexer.java e JsonParser.java contengono rispettivamente tutto il codice Java del lexer, responsabile dell'analisi lessicale del documento di ingresso, e del parser, responsabile invece dell'analisi sintattica e dei controlli semantici che sono stati cablati al suo interno. É fondamentale sottolineare che il codice presente in queste classi viene generato in modo automatico da ANTLRWorks sulla base della specifica fornita[2], quindi,

[2]la specifica è visibile all'interno del file Json.g

nel caso in cui si vogliano eseguire delle modifiche, è necessario modificare la specifica all'interno di ANTLRWorks e generare nuovamente il codice.

- all'interno della classe Handler.java vengono implementati tutti i metodi utilizzati dal parser per eseguire l'analisi semantica e la gestione degli errori.
- la classe Output.java contiene tutta una serie di metodi per la creazione e formattazione del documento di output "output.file" contenente il codice python della rete neurale.
- infine, la classe ParserTester contiene il main dell'applicazione che esegue l'analisi lessicale, sintattica e semantica del documento in ingresso e, nel caso in cui non siano presenti errori, genera il documento di output.

# 3 Manuale Utente

## 3.1 Funzionamento

Come prima cosa, è fondamentale accertarsi che il file eseguibile "JsonToNeuralNetworks.jar" e il file di input "input.file" contenete la specifica della rete si trovino all'interno della stessa cartella. Una volta verificato questo, è sufficiente aprire una finestra del terminale, raggiungere il folder contenente l'eseguibile e il documento di input, e lanciare il comando:

```
java -jar JsonToNeuralNetworks.jar
```

Nel caso in cui il documento di input non presenti errori lessicali, sintattici o semantici, la console ci mostrerà a schermo il seguente messaggio:

```
Parsing with ANTLR lexer
-----------------------
Parsing completed successfully
```

e, all'interno dello stesso folder, verrà automaticamente generato un file "output.file" contenente il codice python desiderato. A questo punto non resta che copiare il codice generato all'interno di un qualsiasi notebook python ed eseguirlo[3]. Invece, nel caso in cui siano presenti degli errori nel file di input contenente la specifica della rete neurale, verrano mostrati a schermo tutti gli errori commessi insieme ad una descrizione del tipo di errore (lessicale, sintattico o semantico) e la correzione appropriata.

# 4 Funzionalità

Come già descritto nelle sezioni precedenti, la funzionalità dell'applicazione "Json To Neural Networks" è quella di permettere all'utente di generare in modo automatico tutto il codice python necessario per implementare e addestrare su un dataset la rete neurale, partendo da un semplice file Json contenete la specifica di alcuni parametri del modello che si desidera realizzare (vedi sezione successiva). Inoltre il codice generato permette di visualizzare un resoconto della rete realizzata, le performance che ha sul dataset di test e le curve di addestramento.

[3]Per eseguire il codice python generato all'interno di un notebook è necessario aver installato le seguenti librerie: pyplot, pandas, numpy, tensorflow e sklearn.

## 4.1 Documento di input

Il documento di input utilizzato per la specifica della rete neurale che si desidera implementare è un semplice file Json con la seguente struttura:

```
{
  "task":"binary classification",
  "hidden_layers": [
    {
      "neurons":30,
      "activation_function":"relu"
    },
    {
      "neurons":20,
      "activation_function":"relu"
    },
    {
      "neurons":10,
      "activation_function":"relu"
    }
],
  "output_layer": {
    "neurons": 1,
    "activation_function":"sigmoid"
  },
  "epochs": 1000,
  "early_stopping": true,
  "learning_rate": 0.01,
  "loss_function": "binary_crossentropy"
}
```

all'interno di ogni campo è necessario specificare:

- "task": il compito che si desidera eseguire con la rete neurale. I task supportati sono "univariate regression" e "binary classification".
- "hidden layers": per ogni layer profondo, il numero di neuroni che si vogliono istanziare e la funzione di attivazione da utilizzare. Ogni layer profondo deve avere un numero di neuroni maggiore di zero ma inferiore o uguale al numero di neuroni del layer precedente.
- "output layer": il numero di neuroni che si vogliono istanziare e la funzione di attivazione da utilizzare nel layer di output. L'output layer deve avere un solo neurone e la funzione di attivazione deve essere coerente con il task specificato.
- "epochs": il numero di epoche di addestramento.
- "early stopping": se vogliamo, o meno, utilizzare l'arresto anticipato durante l'addestramento.
- "learning rate": il valore di learning rate da utilizzare durante l'addestramento.
- "loss function": la funzione di costo da utilizzare per l'apprendimento. La funzione di costo deve essere coerente con il task specificato.

## 4.2 Esempi di utilizzo

Mostriamo adesso degli esempi di utilizzo in cui mostriamo le peculiarità dell'applicazione:

### 4.2.1 Esempio di funzionamento corretto

1. Specifica della rete corretta ("input.file"):

```
{
  "task":"univariate regression",
  "hidden_layers": [
    {
      "neurons":20,
      "activation_function":"relu"
    },
    {
      "neurons":10,
      "activation_function":"relu"
    }
],
  "output_layer": {
    "neurons": 1,
    "activation_function":"linear"
  },
  "epochs": 1000,
  "early_stopping": true,
  "learning_rate": 0.001,
  "loss_function": "mae"
}
```

2. Lancio dell'applicazione:

```
JsonToNeuralNetworks % java -jar JsonToNeuralNetworks.jar
Parsing with ANTLR lexer
-----------------------
Parsing completed successfully
```

3. Output prodotto ("output.file"):

```
#classic env
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
#fix fig size
plt.rcParams["figure.figsize"]=16,9

dataset = pd.read_csv('regression.csv')
dataset = dataset.drop(columns=['Extracurricular Activities'])

X = dataset.iloc[:, :4]
```

```
y = dataset.iloc[:, -1]

X_train, X_temp, y_train, y_temp = train_test_split(X, y,
    test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
    test_size=0.5, random_state=42)

tf.keras.utils.set_random_seed(42)

model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=X_train.shape[1:]),
        tf.keras.layers.Dense(20, activation="relu"),
        tf.keras.layers.Dense(10, activation="relu"),
        tf.keras.layers.Dense(1, activation="linear")
])

early_stopping_cb =
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
    patience=100, restore_best_weights=True)
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss="mae", optimizer=opt)

print('LEARNING IN PROGRESS...')
history = model.fit(X_train, y_train, epochs=1000,
    validation_data=(X_val, y_val), callbacks=early_stopping_cb,
    verbose=0)
print('')

print('TRAIN PERFORMANCE:')
print(model.evaluate(X_train, y_train, verbose=0))
print('')

print('VALIDATION PERFORMANCE:')
print(model.evaluate(X_val, y_val, verbose=0))
print('')

print('TEST PERFORMANCE:')
print(model.evaluate(X_test, y_test, verbose=0))
print('')

print('NEURAL NETWORK ARCHITECTURE:')
model.summary()
print('')

print('LEARNING CURVES:')
pd.DataFrame(history.history).plot(xlim=[0,
    early_stopping_cb.stopped_epoch + 1], grid=True,
    xlabel="Epoch", style=["r--", "b-"])
plt.show()
```

4. Notebook python e risultati:

```
LEARNING IN PROGRESS...

TRAIN PERFORMANCE:
1.6502649784088135

VALIDATION PERFORMANCE:
1.612596869468689

TEST PERFORMANCE:
1.6360418796539307

NEURAL NETWORK ARCHITECTURE:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_4 (Dense)              (None, 20)                100

 dense_5 (Dense)              (None, 10)                210

 dense_6 (Dense)              (None, 1)                 11

=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
```

Figura 2: Performance di train, validazione e test sul dataset di esempio e riassunto dell'architettura di rete implementata con il relativo numero di parametri stimati.

LEARNING CURVES:

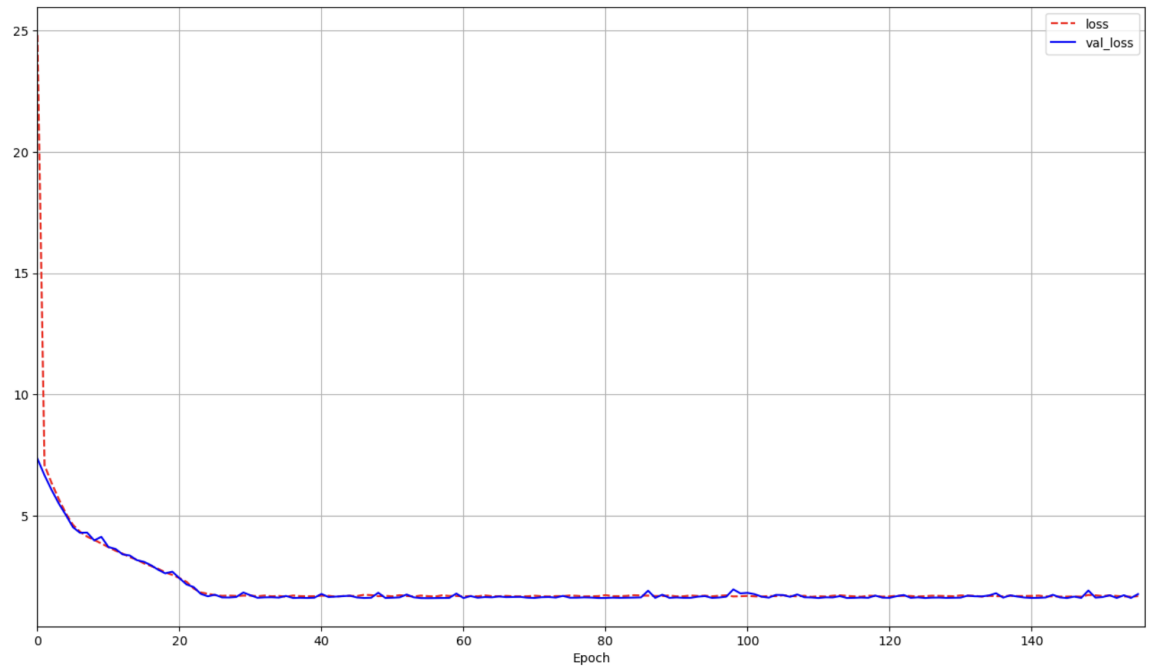


Figura 3: Learning Curves

### 4.2.2 Esempio di funzionamento errato

1. Specifica della rete errata ("input.file"):

```
{
  "task":"univariate regression",
  "hidden_layers": [
    {
      "neurons":50,
      "activation_function":"error"
    },
    {
      "neurons":60,
      "activation_function":"relu"
    },
    {
      "neurons":-1,
      "activation_function":"relu"
    }
],
  "output_layer": {
    "neurons": 10,
    "activation_function":"sigmoid"
  },
  "epochs": -1,
  "early_stopping": true,
  "learning_rate": -0.001,
  "loss_function": "binary_crossentropy"
}
```

2. Lancio dell'applicazione:

```
JsonToNeuralNetworks % java -jar JsonToNeuralNetworks.jar
Parsing with ANTLR lexer
-----------------------
Errore 1: Semantic Error 6 at [6, 29]: The activation function "error"
is not compatible for an hidden layer.
Please select one between tanh, relu, linear or sigmoid.

Errore 2: Semantic Error 5 at [9, 19]: The number of neurons in a
hidden layer at position i+1 must be less equal of the number of neuron
in the hidden layer at position i.

Errore 3: Semantic Error 3 at [13, 17]: The number of neurons within an
hidden layer must be greater than 0.
The value -1 you have inserted is not correct.

Errore 4: Semantic Error 4 at [18, 16]: The number of neurons within the
output layer must be 1.
The value 10 you have inserted is not correct.

Errore 5: Semantic Error 7 at [19, 27]: The activation function
"sigmoid" is not compatible for the output layer or the task
you have selected.
Please select relu/linear for univariate regression task or sigmoid for
binary classification.

Errore 6: Semantic Error 8 at [21, 13]: The number of training epochs
must be greater than 49 to ensure a good training.
The value -1 you have inserted is not correct.

Errore 7: Semantic Error 9 at [23, 20]: The learning rate must be
greater than 0.
The value -0.001 you have inserted is not correct.

Errore 8: Semantic Error 10 at [24, 20]: The loss function
"binary_crossentropy" is not compatible for the task you have selected.
Please select mae for univariate regression task or binary_crossentropy
for binary classification.
```