VI. Reti Neurali per l’ampiezze

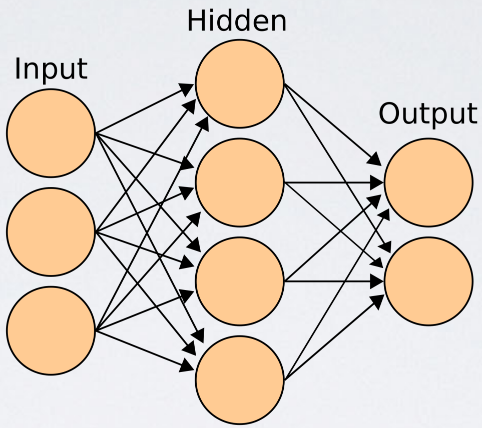
1) Primo Tentativo

2) Errori? - Trained network reliability

3) Cosa habbiamo sbagliato?

I)

The amplitude Neural Network

momenta → 

→ $\langle |\mathcal{A}|^2 \rangle$

ideal answer

universal interface to amplitude generators → NeuralNetwork → universal interface to event generators

approximation sufficiently robust against changes in cuts, PDFs etc.

phasespace

not sure where this goes…

1.1)

Code structure

```
NNnamps  – pstools    – __init__.py
                      – rambo.py
         – njettools  – __init__.py
                      – njet_interface.py
         – nntools    – __init__.py
                      – model.py

         – NJ_order_ee3j_tree.lh
```

1.1)

Choosing the amplitude from NJET

```
njet.py -o NJ_contract_ee3j_tree.lh NJ_order_ee3h_tree.lh
```

```
(base) DA-PHY-61:NNamps xqjk42$ cat NJ_order_ee3j_tree.lh
# OLE_order for ee+3jet production

CorrectionType          QCD
IRregularisation        CDR
AlphasPower             1
AlphaPower              2

NJetReturnAccuracy yes
AmplitudeType tree

SetParameter mass(23) 10e9
SetParameter mass(24) 10e9

NJetNc 3

# process list
11 -11 -> 2 -2 21
```

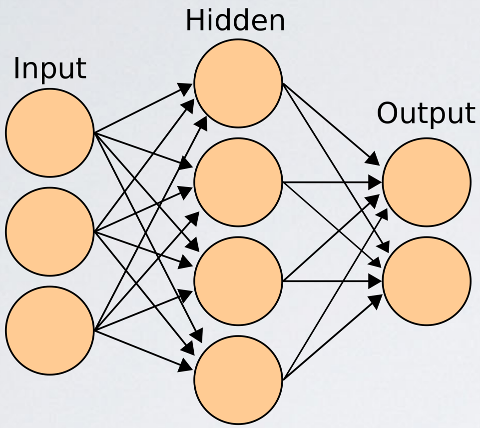
```
(base) DA-PHY-61:NNamps xqjk42$ cat NJ_contract_ee3j_tree.lh
# OLE_order for ee+3jet production
# Generated by njet.py, do not edit by hand.
# Signed by NJet 2636665947.
# 1025 1 1e-05 0.01 2 3 1 1 3.0
CorrectionType QCD | OK
IRregularisation CDR | OK
AlphasPower 1 | OK
AlphaPower 2 | OK
NJetReturnAccuracy yes | OK
AmplitudeType tree | OK
SetParameter mass(23) 10e9 | OK
SetParameter mass(24) 10e9 | OK
NJetNc 3 | OK
# process list
11 -11 -> 2 -2 21 | 1 1 # 72 4 0 2 (4 3 5 -2 -1)
```

particle codes:

11 electron, -11 positron
21 gluon, 22 photon, 23 W, 24 Z
1 down quark, 2 up quark

1.2)

Network architecture




```
def baseline_model(self, layers, lr=0.001, activation='tanh', loss='mean_squared_error'):
    'define and compile model with a fixed dataset but random weights'
    # create model
    # at some point can use new Keras tuning feature for optimising this model
    model = Sequential()
    model.add(Dense(layers[0], input_dim=(self.input_size)))
    if activation == 'tanh':
        model.add(Activation(activations.tanh))
    elif activation == 'relu':
        model.add(Activation(activations.relu))
    else:
        raise ValueError('activation supported are either tanh or relu, you have used {}'.format(activation))

    for i in range(1, len(layers)):
        model.add(Dense(layers[i]))
        if activation == 'tanh':
            model.add(Activation(activations.tanh))
        elif activation == 'relu':
            model.add(Activation(activations.relu))

    model.add(Dense(1))
    # Compile model
    model.compile(optimizer = Adam(lr=lr, beta_1=0.9, beta_2=0.999, amsgrad=False), loss = loss)

    return model
```

hidden layers: tanh activation

output layer: relu activation

optimiser: ADAM with default learning rate=0.001

1.3)

Input variables

- generate momenta with RAMBO algorithm:
  cut with minimum value of $d_{ij} = 2\, p_i.p_j$

- 'flatten' momenta: list of 4n elements

- standardise input and output values

1.4)

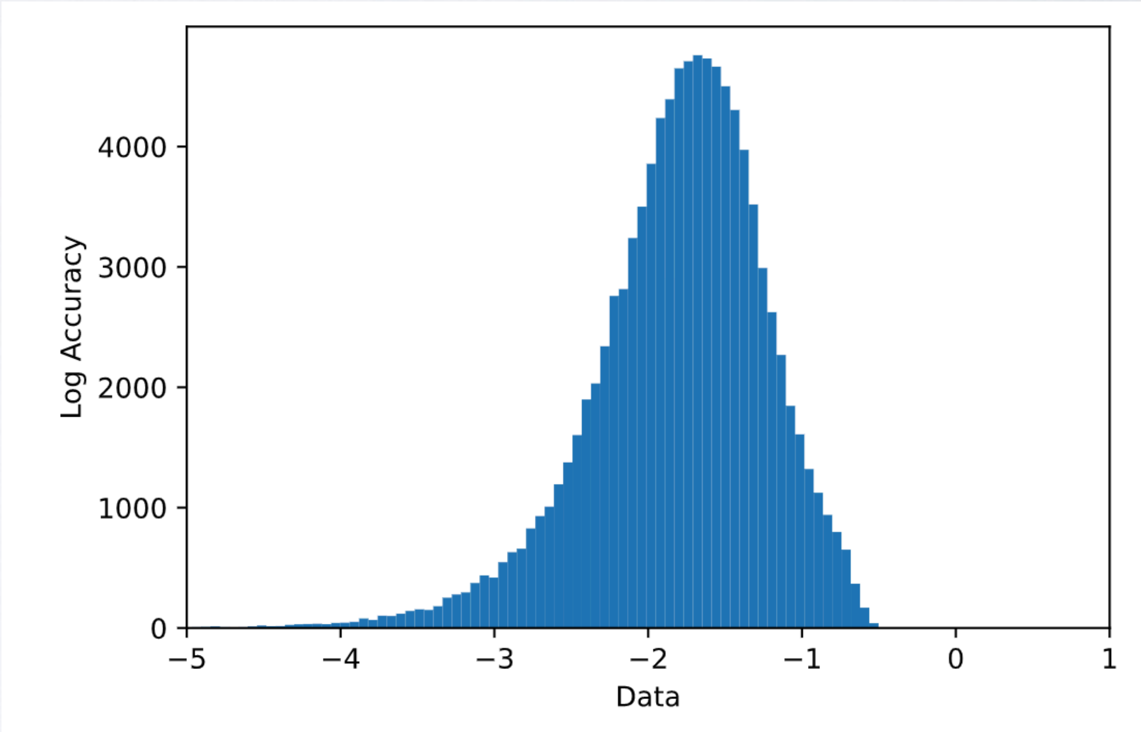
First results

$$\langle|\mathcal{A}|^2\rangle = 4\left(\frac{\alpha}{4\pi}\right)^2\left(\frac{\alpha_s}{4\pi}\right) N_c C_F Q_q^2 \frac{s_{a1}^2+s_{a2}^2+s_{b1}^2+s_{b2}^2}{s_{ab}s_{13}s_{23}}$$

tree-level validation - amplitude evaluation is very fast so no chance of optimisation here…

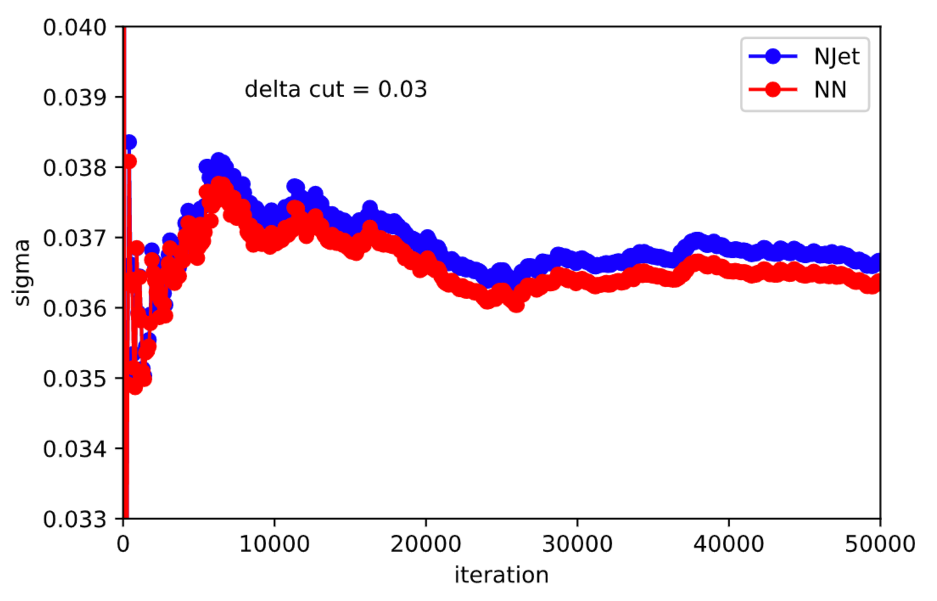
10000 training (80:20 split), 1M interpolation

most points around 10% (1 digit) accurate

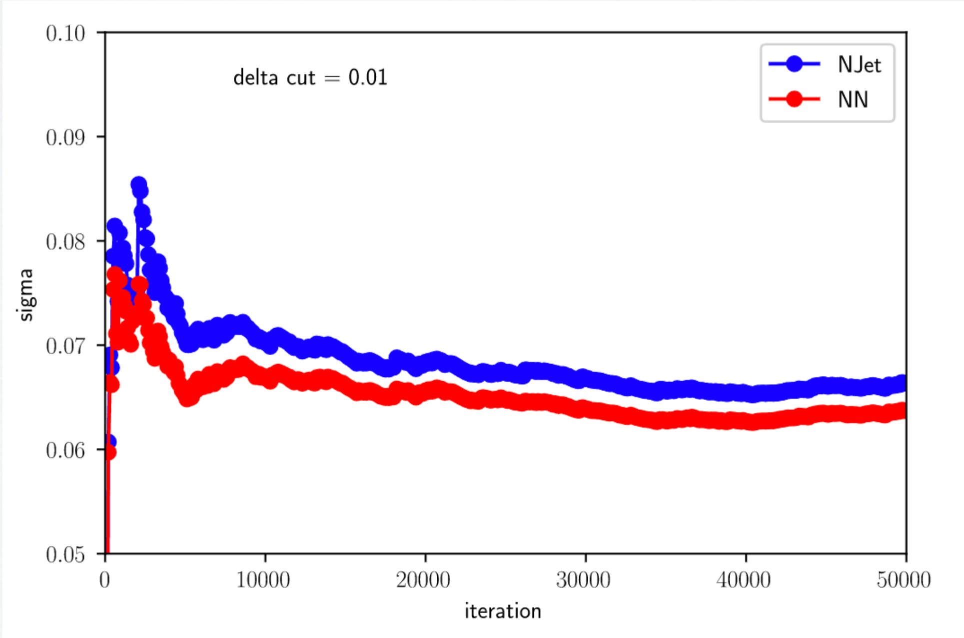


1.4)

First results



NN approximation is less good for small δ



2)

Errors

The MC integration has a well defined error

$$x_i = \langle |\mathcal{A}(p_i)|^2 \rangle$$

$$\sigma = \langle x \rangle \pm \sqrt{\langle x^2 \rangle - \langle x \rangle^2}$$

The dominant error from the NN is the uncertainty in the fit **not** the standard deviation of the interpolated/extrapolated values

Since the NN is fast to call the 'MC' error can be practically zero

2.1)

Neural Network Errors

As far as I know, quantifying errors from a Neural Network is not a standard task.

Especially in this case where the input data is free of experimental noise

Each hyperparameter variation will affect the network -
we can use this to ascertain the reliability of the fit

Shuffle the testing/training splitting (default with `sklearn.model_selection.train_test_split`) while fixing the parameter initialisation ()