Lossy Compression of HEP data through Normalizing Flows

Presentation for the exam project of the course

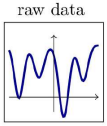
Computing Methods for Experimental Physics and Data Analysis

Luca Callisti, Marco Carotta, Igor Di Tota

Indice

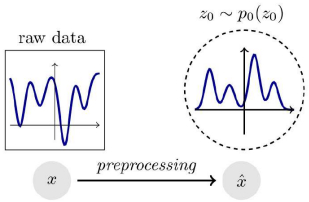
Idea del progetto

Lo scopo di questo progetto è implementare una lossy compression dei dati, attraverso i *Normalizing Flows*.



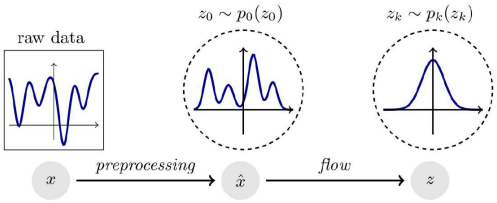
Idea del progetto

Lo scopo di questo progetto è implementare una lossy compression dei dati, attraverso i *Normalizing Flows*.



Idea del progetto

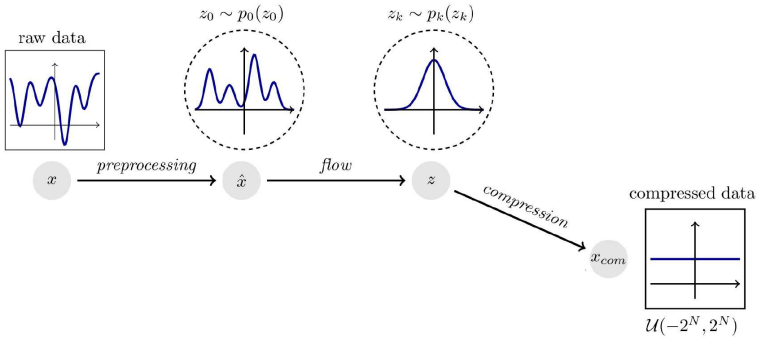
Lo scopo di questo progetto è implementare una lossy compression dei dati, attraverso i *Normalizing Flows*.



Idea del progetto

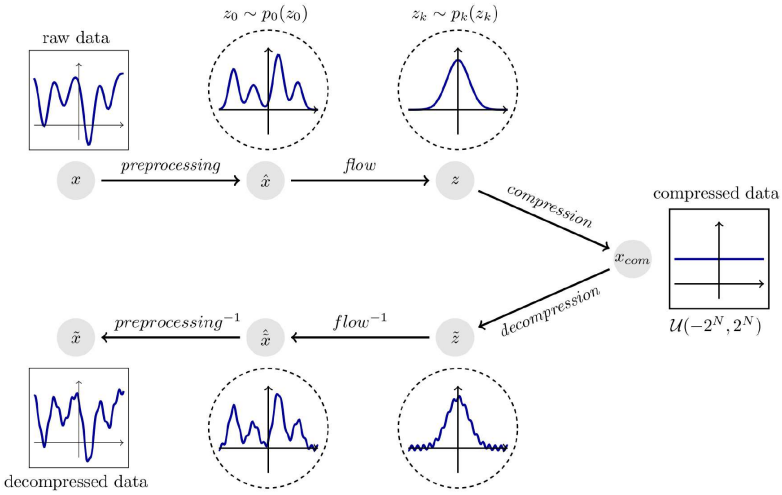
Attraverso la funzione *Erf* portiamo i dati in una distribuzione uniforme.

Per ottenere la compressione riduciamo il numero di bit usati per salvarli.



Idea del progetto

Otteniamo i dati decompressi invertendo gli step precedenti. Infine, per valutare il modello, confronteremo i dati originali con quelli decompressi.



Elementi di Teoria - Normalizing Flows

I*Normalizing Flows* sono un modello generativo e forniscono un metodo per costruire distribuzioni di probabilità su variabili aleatorie continue.
L'idea principale è esprimere $\mathbf{x} \in \mathbb{R}^n$ come immagine, attraverso una trasformazione T, di $\mathbf{u} \sim p_u(\mathbf{u})$:

$$\mathbf{x} = T(\mathbf{u}), \qquad \text{where} \qquad \mathbf{u} \sim p_u(\mathbf{u})$$

in questo modo abbiamo un'espressione esplicita della distribuzione di $\mathbf{x}$

$$p_x(\mathbf{x}) = p_u(\mathbf{u}) \; |\det J_T(\mathbf{u})|^{-1} = p_u(T^{-1}(\mathbf{x})) \; |\det J_{T^{-1}}(\mathbf{x})|$$

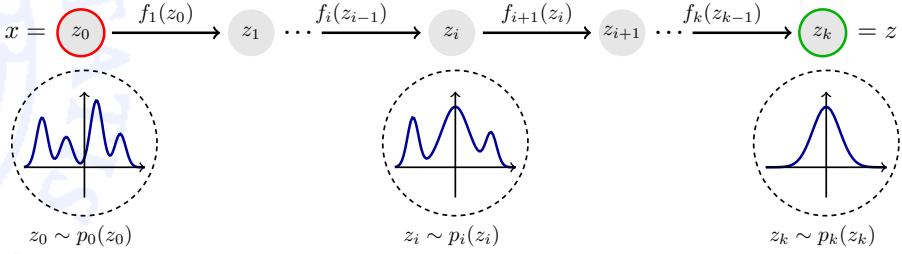
Per usare una rete neurale per approssimare T, si può usare come loss:

$$\mathcal{L}(\phi) = -\mathbb{E}_{p_x^*(\mathbf{x})}[\log\big(p_u(T^{-1}(\mathbf{x};\phi))\big) + \log(\det \mathbb{J}_{T^{-1}}(\mathbf{x};\phi))]$$

che è viene dalla divergenza di Kullback-Leibler.

Elementi di Teoria - Normalizing Flows

Nel seguito useremo la trasformazione inversa per modellare i dati nella distribuzione gaussiana e considereremo questo come flusso, cioè $F = T^{-1}$.
Il flusso può essere visto come composizioni di più trasformazioni che gradualmente modificano la distribuzione mappandola nella *base distribution*.



Elementi di Teoria - Valutazione del modello

Abbiamo valutato il nostro modello in tre modi diversi:

- **Rapporto di compressione**: definito come $R = \frac{\text{size(input file)}}{\text{size(compressed file)}}$
- **Metriche**: confronteremo i dati originali con quelli compressi attraverso:
  - *Difference = original - decompressed*
  - *Response* $= \frac{difference}{original}$
  - *Errore relativo* $= \sqrt{\left(\frac{difference}{std(original)}\right)^2}$
- **Test di Normalità**:
  - statistics: definito come $s^2 + k^2$ dove s è la skewness e k è la kurtosis,
  - p-value: è la probabilità di rigettare l'ipotesi nulla fissato un livello di significatività $\alpha = 0.05$.

Implementazione - Dataset

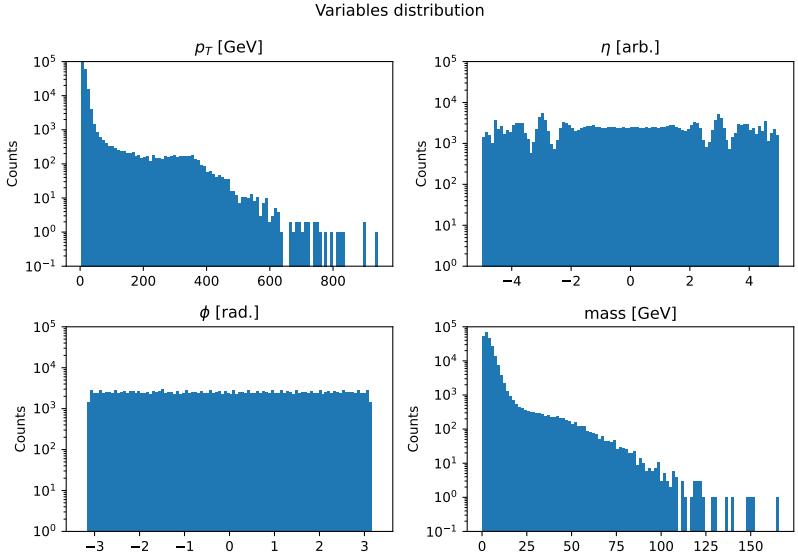
I dati utilizzati sono presi da JetHT primary dataset in AOD format from Run of 2012, scaricati dal CERN Open Data Portal usando la versione CMSSW_5_3_32 della CERN Virtual Machine.

Essi sono costituiti da collisioni protone-protone, in cui sono stati selezionati solo gli eventi contenenti jet. Ogni evento è caratterizzato da 24 variabili ma solo 4 sono state studiate, le componenti del quadrivettore *jet*:

$$j^{\mu} = (p_T, \eta, \phi, m)$$

Il numero di eventi per il dataset di training è 236413.

Implementazione - Dataset



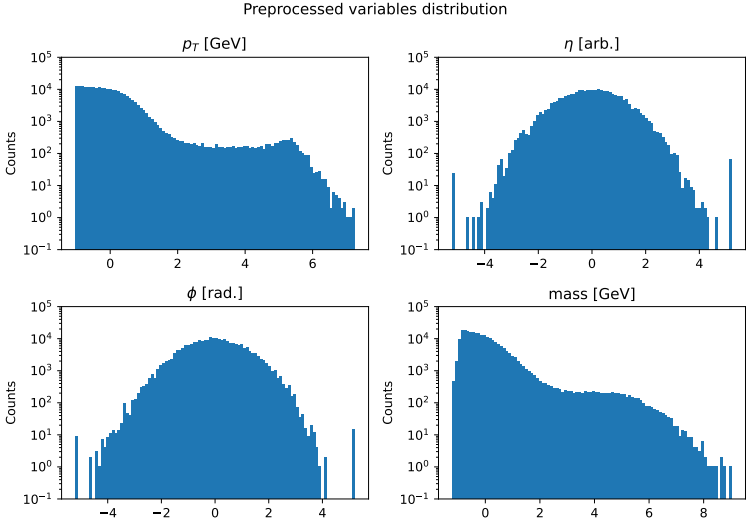
Implementazione - Preprocessing

Alle features p_T e $mass$ abbiamo applicato le seguenti trasformazioni:

- $x \to \log(x + 10)$
- `StandardScaler`

Mentre a ϕ e η si è applicato:

- `StandardScaler`
- `QuantileTrasformer`



Implementazione - Iperparametri

Prima del training, abbiamo fatto alcune run con diversi valori degli iperparametri, in particolare abbiamo fatto variare:

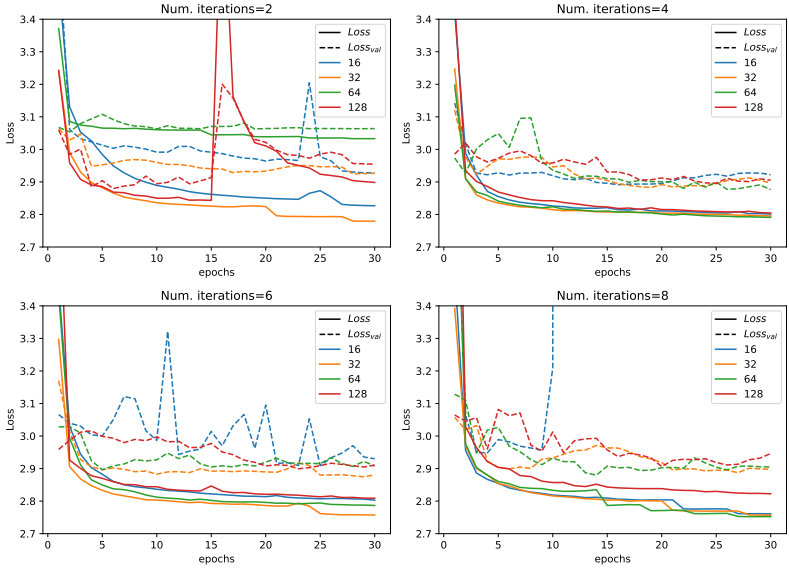
- Numero di funzioni f_i che compongono il flusso: `num_iterations = 2,4,6,8`
- Numero di hidden features: `range_hidden_features = 16,32,64,128`

Questi modelli sono stati allenati per 30 epoche e con `batch_size = 1024` per il train e `val_batch_size = 10000` per la validation.

Implementazione - Iperparametri

Abbiamo scelto i seguenti modelli per il training:

- `model1`: `iterations = 6` `hidden features = 64`
- `model2`: `iterations = 4` `hidden features = 32`

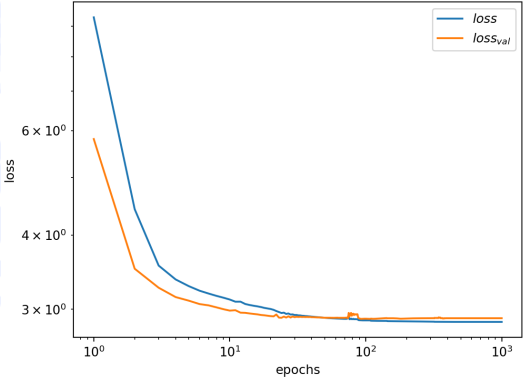


Implementazione - Training

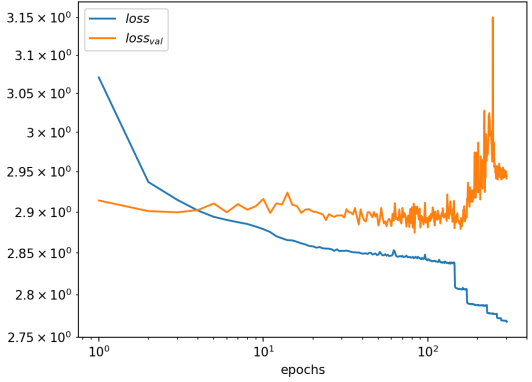
I risultato del train sono:

Name	Epochs	Batch size	Validation batch size	Loss	Validation Loss	Time [min]
model1	1000	10000	10000	2.857	2.899	37
model2	300	32	10000	2.768	2.946	311

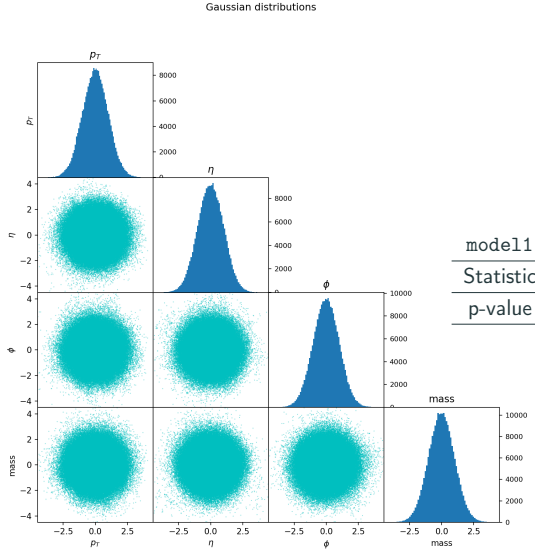
model1



model2

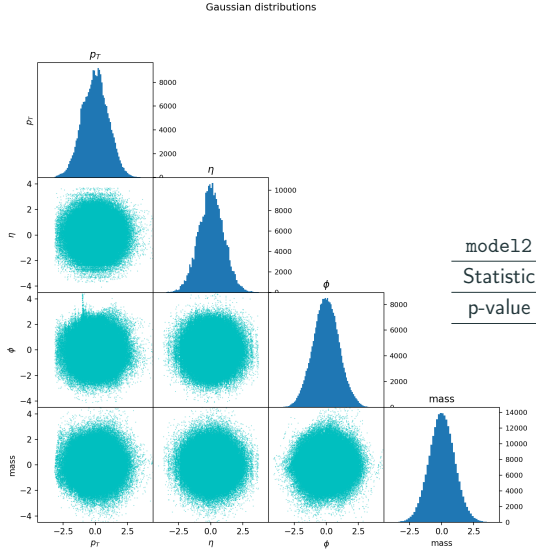


Implementazione - Compressione - Test di Normalità



model1	p_T	η	ϕ	m
Statistic	7	37	128	61
p-value	2.79×10^{-2}	1.16×10^{-8}	1.41×10^{-28}	5.11×10^{-14}

Implementazione - Compressione - Test di Normalità



model2	p_T	η	ϕ	m
Statistic	85	383	129	439
p-value	3.98×10^{-19}	5.25×10^{-84}	9.28×10^{-29}	5.51×10^{-96}

Implementazione - Compressione

Gli step per ottenere la lossy compression sono i seguenti:

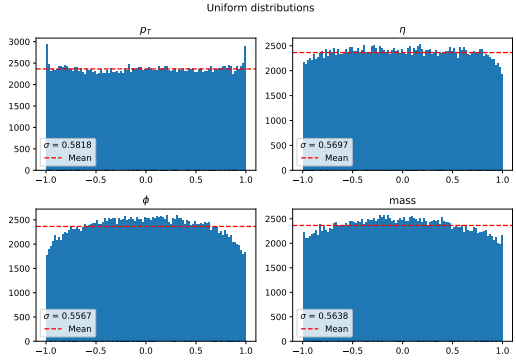
- si usa `MaxAbsScaler` sui dati distribuiti in modo gaussiano per riscalarli tra [-3, 3],
- si applica la funzione *erf* per ottenere le distribuzioni uniformi,
- si moltiplica per 2^N e si prende la parte intera delle distribuzioni uniformi.

I risultati di questa procedura sono i dati compressi.

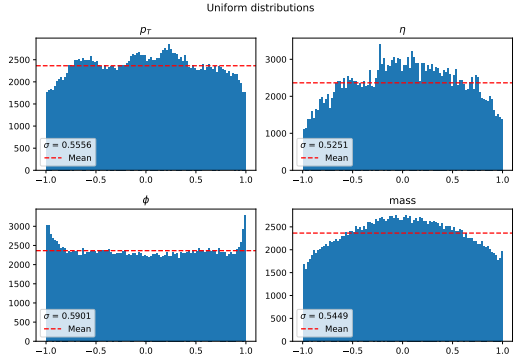
Per ottenere i dati decompressi facciamo le inverse delle trasformazioni precedenti. L'uso di `MaxAbsScaler`è giustificato dalla possibilità di ottenere valori `Inf` come risultati di *ErfInv*.

Implementazione - Compressione - Distribuzione uniforme

Distribuzioni uniformi ottenute applicando *Erf* alle distribuzioni gaussiane:

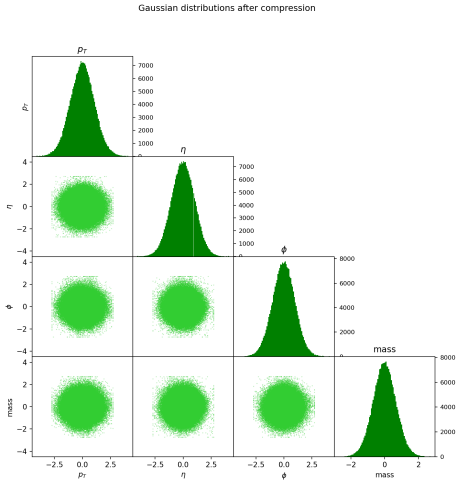


`model1`

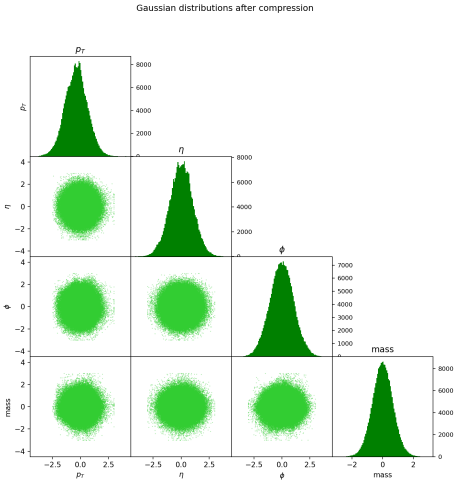


`model2`

Implementazione - Compressione



`model1`, *N=13*



`model2`, *N=20*

Risultati - Rapporto di compressione

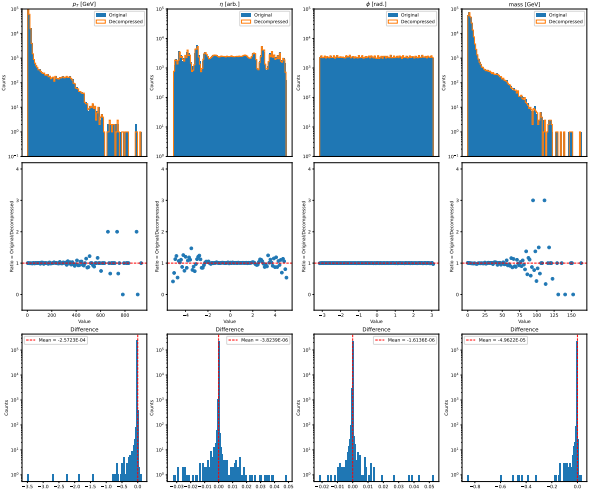
Si è calcolato R utilizzando due formati diversi: `.txt` e `.npz`.

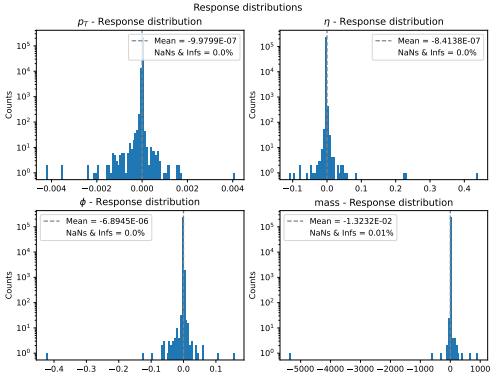
Si notano due proprietà:

- R non dipende dal modello perché Le dimensioni dei file non vi dipendono,
- Le dimensioni dei file di output non dipende dal numero di bit di compressione N.

File		size(`.txt`) [MB]	size(`.npz`) [MB]
input		12.90	5.36
output		16.97	6.69
compressed	$N = 13$	4.83	2.34
	$N = 20$	6.69	3.18
R	$N = 13$	2.67	2.29
	$N = 20$	1.93	1.68

Risultati - `model1`, $N = 20$

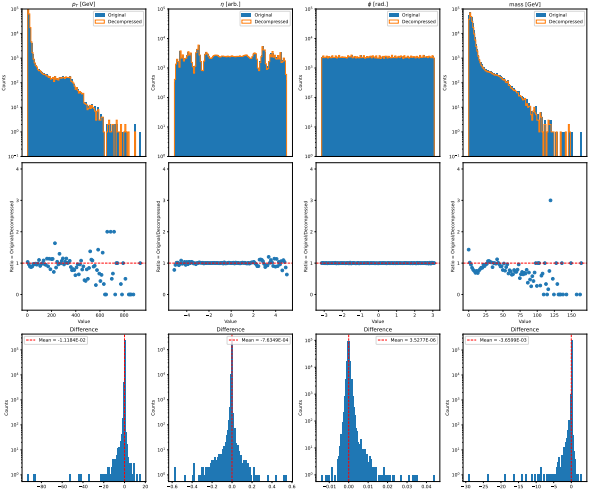


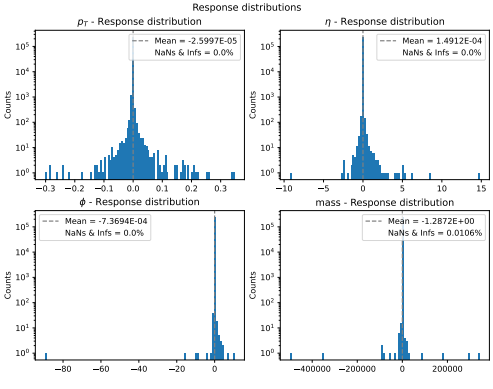


Media del modulo dei residui $= 2.60 \times 10^{-5}$

Errore relativo $= 1.60 \times 10^{-5}$

Risultati - `model2`, $N = 13$





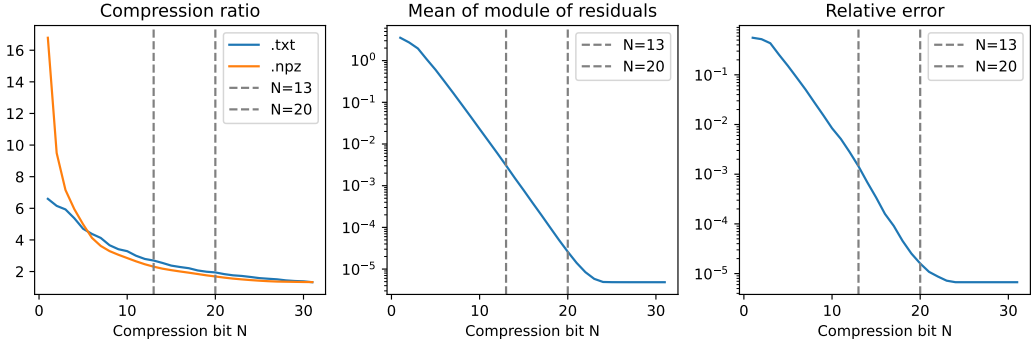
Media del modulo dei residui $= 2.44 \times 10^{-3}$

Errore relativo $= 7.80 \times 10^{-4}$

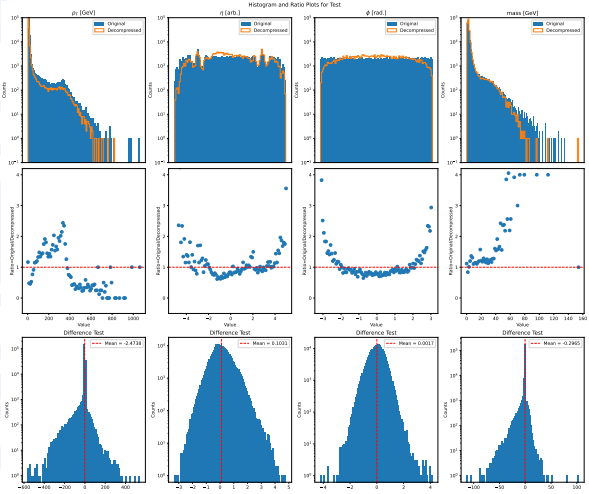
Risultati

Si sono studiate le seguenti quantità al variare di N: il rapporto di compressione R, la media del modulo dei residui e l'errore relativo.

I plot di seguito si riferiscono al `model1`:



Test



Test: `model1`, $N = 13$

Si è testato il modello su 5 dataset da 200000 entrate. Si è calcolato il rapporto di compressione R per ogni dataset con $N = 13$ fissato.

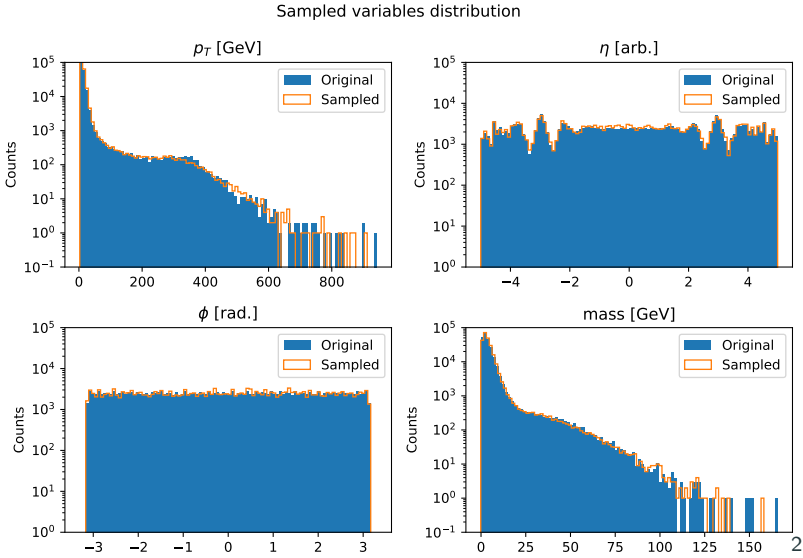
Riportiamo la media e la deviazione standard per ogni modello:

	`.txt`	`.npz`
`model1`	2.682 ± 0.011	2.309 ± 0.048
`model2`	2.702 ± 0.007	2.308 ± 0.004

Altre applicazioni - Sample

Come applicazione principale dei *Normalizing Flows* è possibile generare nuovi dati.

Riportiamo i risultati per `model1`.



Conclusioni

I risultati ottenuti mostrano che c'è una riduzione delle dimensioni dei file, la compressione funziona. Posso essere messe in luce le seguenti osservazioni:

- Il rapporto di compressione R dipende solo dalla dimensione del file compresso. In particolare non dipende dal modello.
- Le dimensioni del file di output non dipendono dal numero di bit di compressione N.
- Il p-value delle distribuzioni gaussiane, nonostante sia sempre minore di 0.05, non influisce sulla compressione.
- Il valore statistics appare accettabile in circa la metà delle features.

Conclusioni

Infine, è possibile un confronto con i risultati ottenuti da Baler - Machine Learning Based Compression of Scientific Data, con $R = 1.7$ fissato:

$R = 1.7$	p_T	η	ϕ	m
Difference	$-1.44 \times 10^{-2} \pm 1.04 \times 10^{-1}$	$-1.12 \times 10^{-3} \pm 2.67 \times 10^{-3}$	$2.45 \times 10^{-4} \pm 1.80 \times 10^{-3}$	$-8.05 \times 10^{-3} \pm 2.51 \times 10^{-2}$
Response	$-1.07 \times 10^{-3} \pm 1.34 \times 10^{-2}$	$3.75 \times 10^{-4} \pm 6.11 \times 10^{-4}$	$3.44 \times 10^{-4} \pm 8.64 \times 10^{-4}$	$2.39 \times 10^{-1} \pm 7.87$

I nostri risultati sono:

model1		p_T	η	ϕ	m
Difference	$N = 13$	-2.83×10^{-2}	-3.86×10^{-4}	-1.75×10^{-4}	-4.85×10^{-3}
	$N = 20$	-2.57×10^{-4}	-3.82×10^{-6}	-1.61×10^{-6}	-4.96×10^{-5}
Response	$N = 13$	-1.22×10^{-4}	-1.52×10^{-4}	-4.96×10^{-4}	-3.67×10^{-1}
	$N = 20$	-9.98×10^{-7}	-8.41×10^{-7}	-6.89×10^{-6}	-1.32×10^{-2}

model2		p_T	η	ϕ	m
Difference	$N = 13$	-1.12×10^{-2}	-7.63×10^{-4}	3.53×10^{-6}	-3.66×10^{-3}
	$N = 20$	-1.16×10^{-4}	-9.29×10^{-6}	1.02×10^{-7}	-3.53×10^{-5}
Response	$N = 13$	-2.60×10^{-5}	1.49×10^{-4}	-7.37×10^{-4}	-1.29
	$N = 20$	-1.83×10^{-7}	-1.16×10^{-6}	-2.54×10^{-6}	-1.98×10^{-1}

dove $N = 13$ corrisponde a $R = 2.29$, mentre $N = 20$ corrisponde a $R = 1.68$.

Conclusioni

Infine, è possibile un confronto con i risultati ottenuti da Baler - Machine Learning Based Compression of Scientific Data, con $R = 1.7$ fissato:

$R = 1.7$	p_T	η	ϕ	m
Difference	$-1.44 \times 10^{-2} \pm 1.04 \times 10^{-1}$	$-1.12 \times 10^{-3} \pm 2.67 \times 10^{-3}$	$2.45 \times 10^{-4} \pm 1.80 \times 10^{-3}$	$-8.05 \times 10^{-3} \pm 2.51 \times 10^{-2}$
Response	$-1.07 \times 10^{-3} \pm 1.34 \times 10^{-2}$	$3.75 \times 10^{-4} \pm 6.11 \times 10^{-4}$	$3.44 \times 10^{-4} \pm 8.64 \times 10^{-4}$	$2.39 \times 10^{-1} \pm 7.87$

I nostri risultati sono:

model1		p_T	η	ϕ	m
Difference	$N = 13$	-2.83×10^{-2}	-3.86×10^{-4}	-1.75×10^{-4}	-4.85×10^{-3}
	$N = 20$	-2.57×10^{-4}	-3.82×10^{-6}	-1.61×10^{-6}	-4.96×10^{-5}
Response	$N = 13$	-1.22×10^{-4}	-1.52×10^{-4}	-4.96×10^{-4}	-3.67×10^{-1}
	$N = 20$	-9.98×10^{-7}	-8.41×10^{-7}	-6.89×10^{-6}	-1.32×10^{-2}

model2		p_T	η	ϕ	m
Difference	$N = 13$	-1.12×10^{-2}	-7.63×10^{-4}	3.53×10^{-6}	-3.66×10^{-3}
	$N = 20$	-1.16×10^{-4}	-9.29×10^{-6}	1.02×10^{-7}	-3.53×10^{-5}
Response	$N = 13$	-2.60×10^{-5}	1.49×10^{-4}	-7.37×10^{-4}	-1.29
	$N = 20$	-1.83×10^{-7}	-1.16×10^{-6}	-2.54×10^{-6}	-1.98×10^{-1}

dove $N = 13$ corrisponde a $R = 2.29$, mentre $N = 20$ corrisponde a $R = 1.68$.