GENERATIVNI ALGORITMI STROJNEGA UČENJA V FIZIKI

KORL ŽELEZNIKAR

Fakulteta za matematiko in fiziko
Univerza v Ljubljani

Generativni algoritmi so v zadnjih letih, zahvaljujoč uporabi v industriji, močno pridobili na popularnosti. Njihova sposobnost ustvarjanja prepričljivih novih podatkov pa se zdaj izkazuje tudi v fiziki osnovnih delcev, kjer zaradi tehnoloških nadgradenj na trkalnikih potreba po hitrejšem simuliranju odzivov detektorjev na trke delcev narašča. Uporaba teh algoritmov se tako ponuja kot obetavna rešitev. Članek predstavi osnovne principe strojnega učenja in se poglobi v delovanje difuzijskega generativnega algoritma, ki se uporablja za simulacije odziva kalorimetra pri trkih v fiziki osnovnih delcev kot nadomestek tradicionalnih Monte Carlo metod.

GENERATIVE MACHINE LEARNING ALGORITHMS IN PHYSICS

Generative algorithms have gained significant popularity in recent years, due to their widespread use in industry. Their ability to create convincing new data is now proving valuable in particle physics, where technological upgrades of colliders have increased the demand for faster simulations of detector responses to particle collisions. Thus, implementing these algorithms presents a promising solution. This article presents the fundamental principles of machine learning and delves into the workings of a diffusion-based generative algorithm that is used to simulate the calorimeter response in particle collision experiments as an alternative to traditional Monte Carlo methods.

1. Motivacija

Simulacije trkov v fiziki osnovnih delcev, ki temeljijo na teoretičnih modelih, predstavljajo ključno povezavo med teorijo in eksperimentom ter se uporabljajo tako za načrtovanje novih detektorjev, kot za analizo obsežnih količin podatkov med trki. V zadnjih letih se je zaradi tehnoloških nadgradenj velikega hadronskega trkalnika (HL – LHC [1]) in preobremenjenosti globalne računske mreže LHC (angl. *Worldwide LHC Computing Grid*) [2] pojavila potreba po hitrejših simulacijah odziva detektorjev na trke [2, 3].

Sodobni eksperimenti v fiziki visokih energij (HEP) uporabljajo metode Monte Carlo (MC) z visoko ločljivostjo za primerjavo eksperimentalnih meritev in teoretičnih napovedi, a so te računsko zahtevne, saj z njimi modeliramo dogodke od začetnega sipanja do kompleksnih interakcij delcev z detektorjem [4]. Ob tehnoloških nadgradnjah (večja svetilnost, boljše ločljivosti detektorjev [5]) in nedavnem porastu uporabe modelov strojnega učenja v industriji (Sora [6], ChatGPT [7], MidJourney [8], Dall-E [9] ...), so se kot potencialni odgovor na omenjene težave pojavili generativni modeli [2, 10]. Obstaja več vrst obetavnih algoritmov, s katerimi lahko ustvarjamo nove podatke, a se je v primeru simulacij kalorimetra pri trkih, ki zahtevajo največ računske moči [2], difuzijski algoritem izkazal za najuspešnejšega, zato mu bo posvečena posebna pozornost [3, 4, 11].

2. Osnove strojnega učenja

Da bi razumeli delovanje teh algoritmov, moramo najprej spoznati osnovne principe delovanja strojnega učenja. Model strojnega učenja lahko **opišemo kot funkcijo**, ki vektor vhodnih podatkov $\mathbf{x}$ slika v izhodni vektor $\mathbf{y} = \mathbf{f}(\mathbf{x}, \boldsymbol{\phi})$, kjer s $\boldsymbol{\phi}$ označimo parametre modela, ki določajo, kako se ta obnaša. Najprej si bomo pogledali, kako take funkcije sestaviti, kasneje pa kako optimizirati parameter $\boldsymbol{\phi}$, da izhodni vektor $\mathbf{f}$ (model) izpolni naše zahteve (na primer generira slike žalostnih surikat).

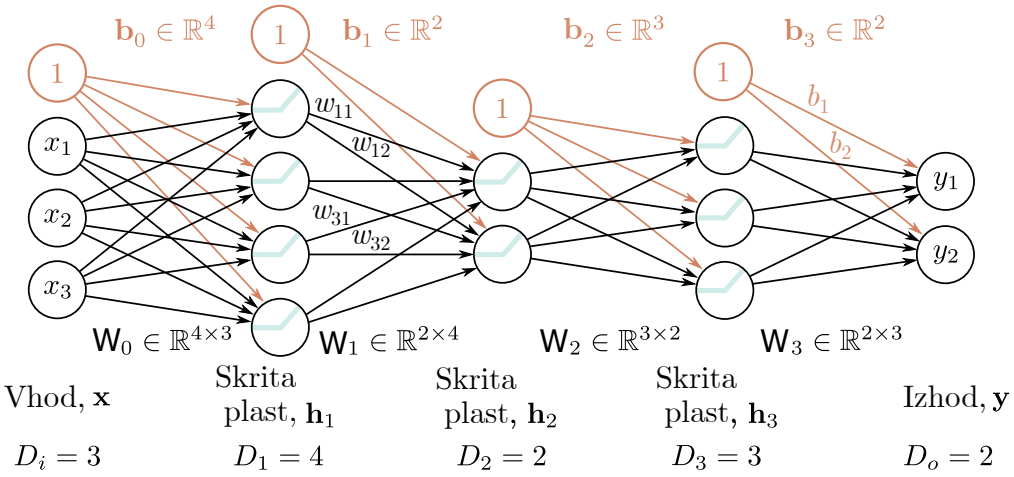
2.1 Nevronske mreže

Funkcijo $\mathbf{f}(\mathbf{x}, \boldsymbol{\phi})$ sestavimo z nevronsko mrežo (angl. *neural network* – *NN*). Zahvaljujoč strukturi nevronskih mrež je ta lahko poljubno zapletena, kar omogoča učenje kompleksnih odvisnosti.

Mrežo sestavljajo plasti nevronov, ki jih opišemo z vektorsko funkcijo $\mathbf{h}$. Vsaka plast pošlje naslednji plasti signal, ki ga v i-tem nevronu na plasti definira obtežen seštevek signalov iz prejšnje plasti z utežmi w_{ij} in pristranskostjo b_i, na katerega deluje še **nelinearna funkcija** σ. Če ima mreža K skritih plasti, lahko zapišemo funkcijo $\mathbf{y} = \mathbf{f}(\mathbf{x}, \boldsymbol{\phi})$ kot

$$\begin{aligned}
\mathbf{h}_1 &= \sigma(\mathbf{b}_0 + \mathsf{W}_0\mathbf{x})\,,\\
\mathbf{h}_2 &= \sigma(\mathbf{b}_1 + \mathsf{W}_1\mathbf{h}_1)\,,\\
&\vdots\\
\mathbf{h}_K &= \sigma(\mathbf{b}_{K-1} + \mathsf{W}_{K-1}\mathbf{h}_{K-1})\,,\\
\mathbf{y} &= \tilde{\sigma}(\mathbf{b}_K + \mathsf{W}_K\mathbf{h}_K)\,,
\end{aligned}$$

kjer so W_k matrika k-te plasti z utežmi $(\mathsf{W}_k)_{ij} = w_{ij}^{(k)}$, $\mathbf{b}_k$ vektor pristranskosti in $\mathbf{h}_k$ skrita plast nevronov za $k = 1, ..., K$. Vedenje take funkcije je popolnoma določeno z vrednostmi vseh uteži in pristranskosti, ki jih zato zložimo v vektor $\boldsymbol{\phi} = \{\mathbf{b}_k, \mathsf{W}_k\}_{k=0}^{K}$, preko katerega bomo model učili [12, str. 49].



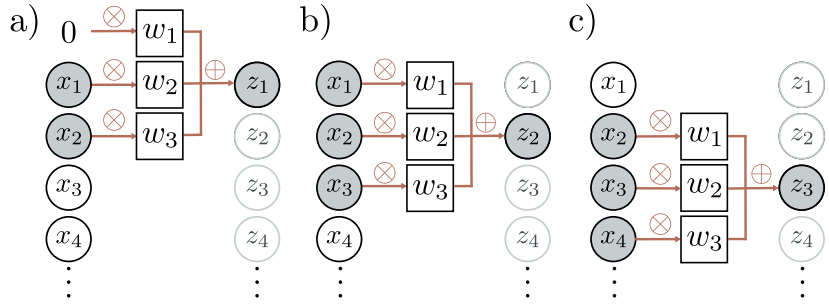
Slika 1. Primer nevronske mreže. Puščice, ki kažejo v isti nevron, simbolizirajo seštevanje signalov. Vsaki puščici pripišemo utež w_{ij} in pristranskosti b_i. Slika v vsakem nevronu prikazuje nelinearno aktivacijsko funkcijo, v tem primeru ReLU. Prirejeno po ref. [12].

S σ smo označili **aktivacijsko funkcijo**, ki deluje na uteženo vsoto signalov v prejšnji plasti. Brez nje bi bila funkcija $\mathbf{f}$ kompozicija linearnih funkcij in zato linearna, s čimer pa ne moremo modelirati kompleksnejših porazdelitev. V praksi se pogosto uporablja t.i. funkcija $\mathrm{ReLU}(x) = \max(0, x)$, saj se z njo modeli najbolje učijo (empirično raziskano) [13]. Lahko si predstavljamo, da ta doda kolena končni funkciji $\mathbf{f}$, kjer imajo linearne funkcije plasti ničle, zato je končna funkcija sestavljena iz odsekoma linearnih kosov [12, str. 25-28]. Na sliki 1 je prikazan primer mreže z aktivacijsko funkcijo ReLU in s $K = 3$ plastmi. Take funkcije so prevzele ime nevronske mreže, saj posnemajo obnašanje bioloških nevronov v možganih [14].

2.1.1 Konvolucijske nevronske mreže in *U-net* arhitektura

Če je vhodni podatek slika, predstavlja $\mathbf{x}$ urejene RGB vrednosti vseh pikslov v sliki, zato imajo v tem primeru že sami vhodni podatki velike dimenzije. Hitro ugotovimo, da polno povezane plasti

niso priročne za obdelavo slik. Na primer, slika velikosti 224×224 pikslov v RGB barvah bi v najpreprostejšem primeru, ko ima skrita plast enako število nevronov kot vhodna, zahtevala okoli $(224^2 \cdot 3)^2 \approx 22$ milijard uteži na plast. Poleg tega so bližnje slikovne točke med sabo vsebinsko povezane (v nasprotnem primeru bi bile slike le naključen šum). Rešitev predstavljajo konvolucijske mreže, ki izkoriščajo korelacije med bližnjimi slikovnimi točkami in se tako sproti učijo interpretacije slikovnih točk na vsakem položaju. Dodatno te že same po sebi upoštevajo, da je interpretacija slike invariantna na translacije, kar še zmanjša število parametrov [5, 12].



Slika 2. Primer enorazsežne konvolucijske mreže s korakom 1, tu velja enačba (1). Prirejeno po ref. [12].

Pri enorazsežni konvoluciji v splošnem na vhodni podatek $\mathbf{x}$ delujemo z manjšim vektorjem (v splošnem tenzorjem), ki ga premikamo za poljubno velik korak. Predpis i-tega nevrona za transformacijo vhoda z vektorjem s tremi utežmi in korakom ena je tako

$$h_i = \sigma\Bigg(b + \underbrace{\sum_{n=1}^{3} w_n x_{i+n-2}}_{z_i}\Bigg), \tag{1}$$

bolj zgovorna pa je tu slika 2. V primeru, ko korakamo za ena, postane matrika uteži W tridiagonalna. Postopek lahko posplošimo tudi na več-dimenzijske vhode (RGB slika 224×224 pikslov bi imela kot vhod dimenzije $\mathbb{R}^{224\times224\times3}$), kjer namesto z vektorji transformiramo s tenzorji višjih rangov.

V difuzijskih modelih je najbolj pogosta uporaba arhitekture *U-net*, ki je izpeljanka konvolucijske. Sestavljena je iz kodirnika, v katerem sliko večkrat zmanjšamo, in dekodirnika, v katerem jo povečamo. Posebnost leži v združevanju slik iz kodirnika s slikami iz dekodirnika. *U-net* so prvič uspešno uporabili leta 2015 v [15] za segmentacijo slik bioloških tkiv.

2.2 Kako se modeli učijo?

Pred uporabo moramo model izuriti. Ko govorimo o učenju, imamo v mislih **iskanje parametrov** $\boldsymbol{\phi}$, ki omogočajo smiselne napovedi iz vhodnih podatkov [12, str. 18]. Pri algoritmih za klasifikacijo poznamo predpisan nabor parov testnih vhodnih podatkov in pripadajočih oznak $\{\mathbf{x}_i, \mathbf{y}_i\}$[1]. Množica $\{\mathbf{x}_i\}$ na primer predstavlja slike žalostnih surikat, $\{\mathbf{y}_i\}$ pa nivo žalosti. Generativni algoritmi se učijo **nenadzorovano**, kar pomeni, da se model uči na množici podatkov $\{\mathbf{x}_i\}$ brez oznak. Predpostavimo, da slike izhajajo iz neznane verjetnostne porazdelitve $p(\mathbf{x})$ kot statistično neodvisni vzorci. **Modeli se učijo strukture testnih podatkov** $p(\mathbf{x}_i|\boldsymbol{\phi})$, iz katere lahko vzorčimo za generacijo novih vzorcev. S $p(\mathbf{x}|\boldsymbol{\phi})$ se skušamo približati pravi porazdelitvi $p(\mathbf{x})$, ki je ne znamo analitično izračunati.

Za lažjo predstavo si lahko zamislimo primer generiranja rezultatov metov neidealne igralne kocke. Če ima ta odkrušen vogal, ali pa je mogoče nehomogena, rečemo, da ni idealna in nam prava verjetnost $p(x)$, da pade številka x, ni znana ($p(x)_{\text{id.}} = 1/6$ bi bila porazdelitev za idealno

[1] Količine z indeksom i označujejo i-ti testni vhodni/izhodni vektor.

kocko). Z meritvami padlih cifer $\{x_i\}$ vzorčimo iz $p(x)$ in z modelom prilagodimo približek $p(x|\boldsymbol{\phi})$ za $p(x)$ glede na izmerke $\{x_i\}$, kjer so $\boldsymbol{\phi}$ parametri modelske porazdelitve. Če smo model dobro naučili, bodo nove padle cifre vzorčene iz $p(x|\boldsymbol{\phi})$ imele frekvence podobne tistim, ki smo jih namerili z metanjem kocke.

2.2.1 Funkcija izgube

Vsakemu algoritmu moramo **glede na cilj** prirediti t.i. **funkcijo izgube** $\mathcal{L}(\boldsymbol{\phi})$ (angl. *loss function*), ki karakterizira, kaj in kako se bo model učil. Tako kvantificiramo neskladnost med porazdelitvijo, ki jo izračuna mreža iz testnih podatkov $p(\mathbf{x}|\boldsymbol{\phi})$, in dejansko porazdelitvijo podatkov $p(\mathbf{x})$. Ker o $p(\mathbf{x})$ pogosto ne vemo ničesar, moramo najprej izbrati verjetnostno porazdelitev, s katero bomo podatke modelirali. Izbrano modelsko porazdelitev $p(\mathbf{x}|\boldsymbol{\phi})$ popolnoma karakterizirajo njeni momenti, ki jih zapišemo v vektor $\boldsymbol{\theta}$. Če bi izbrali Gaussovo porazdelitev, bi z modelom računali pričakovano vrednost (pr. vr.) in varianco $\boldsymbol{\theta} = (\mu, \sigma^2)^\mathsf{T}$. Mreža bo torej računala momente porazdelitve, s katero bomo opisali vhodne podatke:

$$\boldsymbol{\theta} = \mathbf{f}(\mathbf{x}, \boldsymbol{\phi}).$$

Pri **dobro naučenih parametrih** bo vrednost porazdelitve $p(\mathbf{x}_i|\boldsymbol{\phi}_{\text{opt.}})$ **maksimalna**, saj to pomeni, da je pri danih parametrih $\boldsymbol{\phi}_{\text{opt.}}$ verjetnost, da vzorčimo podatek $\mathbf{x}$, ki je podoben $\mathbf{x}_i$, velika (gl. sliko 3). Ker imamo testnih vektorjev več, recimo I, mora biti verjetnostna gostota **za vse testne vektorje naenkrat** $p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_I|\boldsymbol{\phi})$ največja. Pogosto predpostavimo, da so $\mathbf{x}_i$ neodvisni in imajo enake porazdelitve, da lahko verjetnosti za i-ti vzorec med sabo množimo, torej velja

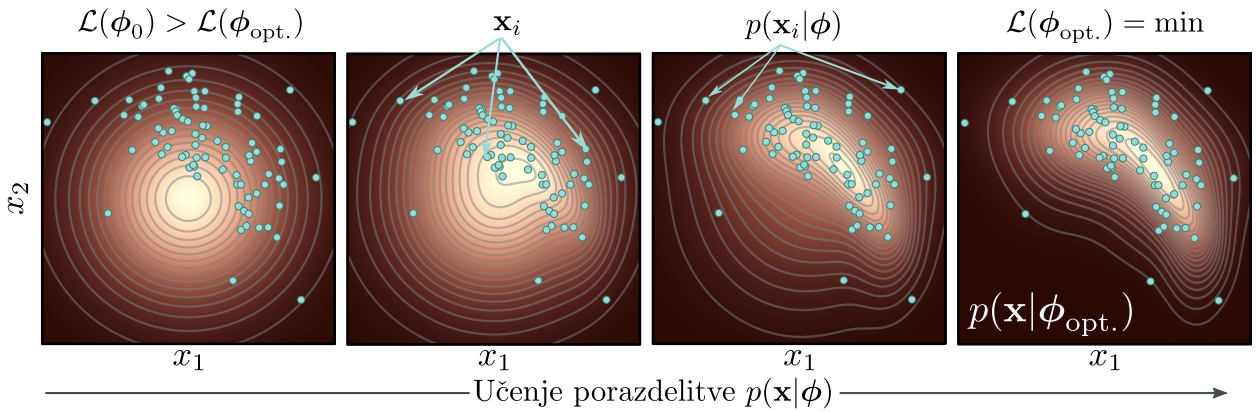
$$p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_I|\boldsymbol{\phi}) = \prod_{i=1}^{I} p(\mathbf{x}_i|\boldsymbol{\phi})\,.$$

Uporabimo še logaritem, da lahko zmnožek spremenimo v vsoto, saj je ta za veliko število vzorcev I lahko zelo majhen, kar povzroči numerične preglavice. V splošnem lahko tako za izgubo napišemo

$$\mathcal{L}(\boldsymbol{\phi}) = -\log\left(\prod_{i=1}^{I} p(\mathbf{x}_i|\boldsymbol{\phi})\right) = -\sum_{i=1}^{I} \log\Big(p(\mathbf{x}_i|\boldsymbol{\phi})\Big), \tag{2}$$

kjer smo dodali negativni predznak, saj želimo iskati minimum [12, str. 56-59].

Funkcija $\mathcal{L}(\boldsymbol{\phi})$ torej izhodu na mreži priredi realno število, s katerim modelu sporočimo, v kolikšni meri s trenutno vrednostjo $\boldsymbol{\phi}$ ta zadovolji svoj cilj. Če je vrednost $\mathcal{L}$ v primerjavi z minimumom velika, to pomeni, da dobljena verjetnost testnih podatkov ne opiše dobro.



Slika 3. Učenje porazdelitve $p(\mathbf{x}|\boldsymbol{\phi})$ podatkov $\{\mathbf{x}_i\}$ (turkizne točke) na primeru dvodimenzionalnih vhodnih vektorjev $\mathbf{x} = (x_1, x_2)^\mathsf{T}$. Pri naučeni porazdelitvi je funkcija izgube najmanjša možna. Prirejeno po ref. [12].

2.2.2 Stohastični gradientni spust

Ostane še vprašanje, kako posodabljati parameter $\boldsymbol{\phi}$. Negativni gradient funkcije izgube bo enak smeri, v kateri bo vrednost $\mathcal{L}$ najhitreje padala, zato je korak za izboljšanje parametrov zelo očiten:

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \frac{\partial \mathcal{L}(\boldsymbol{\phi})}{\partial \boldsymbol{\phi}}\bigg|_{\boldsymbol{\phi}_t} = \boldsymbol{\phi}_t - \alpha \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}\bigg|_{\boldsymbol{\phi}_t}. \tag{3}$$

Pri tem smo izgubo zapisali kot vsoto prispevkov posameznega vzorca za učenje $\mathcal{L}(\boldsymbol{\phi}) = \sum_{i=1}^{I} \ell_i$, α pa označuje parameter hitrosti učenja (angl. *learning rate*).

Izkaže se, da je sam gradientni spust za velike količine parametrov prepočasen (novejši modeli jih imajo $\sim 10^{12}$). Izboljšamo ga tako, da ob vsaki iteraciji izmed I testnih vzorcev **naključno** izberemo podmnožico ali serijo $\mathcal{B} \subset I$ (angl. *batch*) in odštejemo gradient funkcije izgube le podmnožice vseh vzorcev

$$\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t - \alpha \sum_{i \in \mathcal{B}_t} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}\bigg|_{\boldsymbol{\phi}_t}. \tag{4}$$

Pri tem $\mathcal{B}_t$ označuje izbrano podmnožico indeksov i ob času t. Gradient tako ne kaže več v smeri najhitrejšega spusta, saj smo s tem dodali nekaj šuma, a nas ta hitreje pripelje do minimuma.[2] Temu pravimo stohastični gradientni spust [12, str. 85].

Preprosto povedano lahko na modele strojnega učenja gledamo kot na funkcije $\mathbf{f}$, ki jih v osnovi implementiramo z nevronskimi mrežami. Lastnosti funkcije definirajo parametri $\boldsymbol{\phi}$. Pri generativnih algoritmih vhodnih podatkov $\{\mathbf{x}_i\}$ ne moremo primerjati z njihovimi oznakami, zato se modeli učijo porazdelitve podatkov $p(\boldsymbol{x}|\boldsymbol{\phi})$, iz katere bi lahko vzorčili. Funkcijo izgube nato definiramo tako, da bo pri optimalnih parametrih $\boldsymbol{\phi}$ funkcija $\mathbf{f}$ najbolje posnemala porazdelitev testnih podatkov. Iz naučene porazdelitve lahko nato vzorčimo nove podatke.

3. Predlagani generativni modeli

V okviru HEP so bile predlagane že štiri večje skupine generativnih algoritmov. Generativne antagonistične mreže (angl. *Generative adversarial networks – GAN*) so velikostne rede hitrejše kot klasične simulacije in se uporabljajo v detektorjih ATLAS [16], LHCb [17] in Belle II [18], a so nestabilne in imajo težave z učenjem celotnega prostora parametrov podatkov. Večji, a sorazmerno neuspešni skupini, predstavljajo variacijski avotenkoderji (angl. *Variational autoencoder – VAE*) in pretočni modeli (angl. *Nomralizing Flows – NF*), ki so se izkazali za slabše zaradi težav z natančnostjo in razširljivostjo na večje dimenzije. Zadnji predlagani so bili difuzijski modeli, ki so se zaradi enostavnega učenja, dobre natančnosti in razširljivosti ter obvladljivih računskih zahtev izkazali za uspešne v fizikalnem okolju [2, 3, 4, 11].

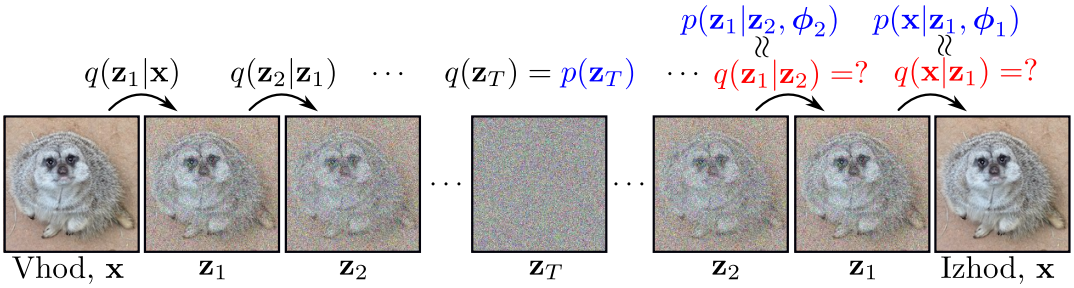
4. Difuzijski model

Najprej poskusimo na primeru generiranja slik intuitivno razumeti, kako taki modeli delujejo. V prostoru vseh slik fiksne velikosti jih velika večina predstavlja naključen šum (predstavljajte si, da izmed vseh možnih kombinacij slik 224×224 pikslov naključno izberete sliko žalostne surikate). Med zašumljenimi primeri so otoki slik, kjer imajo sosednji piksli pravilne korelacije in predstavljajo smiselne slike. Pri difuzijskih modelih začnemo z vzorci $\{\mathbf{x}_i\}$ iz nekega otoka (recimo slike žalostnih surikat) in jih postopoma zašumljamo, dokler ne ostane le še normali šum. S primerno funkcijo izgube model naučimo, kako se vrniti v otok logičnih slik. Drugače povedano, model se nauči obliko

[2] Dodani šum omogoča, da pri učenju model preskoči lokalne minimume, v katerih bi se sicer ujel. Lahko ga še izboljšamo, če korakom dodamo „gibalno količino“ [12, str. 83-88].

vektorskega polja gradienta modelske porazdelitve $p(\mathbf{x}|\boldsymbol{\phi})$, ki ga iz naključno izžrebanega začetnega položaja pripelje do želene skupine slik.

Arhitekturno so difuzijski modeli sestavljeni iz kodirnika in dekodirnika (*U-net*). Prvi je odgovoren za zašumljanje slik, slednji pa za rekonstrukcijo. Proces zašumljanja ali difuzije (angl. *forward diffusion*) poteka po v naprej določenih parametrih, zato se učenje mreže dogaja le v dekodirniku, kot je prikazano na sliki 4.



Slika 4. Proces difuzije v kodirniku in učenja vzvratnega procesa v dekodirniku. Prirejeno po ref. [12].

4.1 Kodirnik

Podobnosti s fizikalnim procesom difuzije postanejo očitne, ko si pogledamo matematični opis kodirnika. Merske podatke $\mathbf{x}$ zašumljamo z normalno porazdeljenim šumom, dokler niso ti popolnoma zašumljeni. Po dogovoru naj bo na vsakem koraku zašumljen podatek označen z $\mathbf{z}_t$, kjer časovni koraki tečejo po $t \in \{1, 2, \ldots, T\}$ in tako na koncu velja $\mathbf{z}_T \sim \mathcal{N}(0, \mathbb{1})$. Tako lahko za vsak korak zapišemo

$$\begin{aligned} \mathbf{z}_1 &= \sqrt{1-\beta_1}\mathbf{x} + \sqrt{\beta_1}\boldsymbol{\epsilon}\,, \\ \mathbf{z}_t &= \underbrace{\sqrt{1-\beta_t}\mathbf{z}_{t-1}}_{\text{nova pr. vr.}} + \underbrace{\sqrt{\beta_t}\boldsymbol{\epsilon}}_{\text{dodani šum}}\,, \end{aligned} \tag{5}$$

kjer smo z $\beta_t \in (0, 1)$ označili **v naprej določen** razporejevalnik variance šuma, s katerim reguliramo količino dodanega šuma na iteracijo. Dodani šum predstavlja $\boldsymbol{\epsilon}$, za katerega velja $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbb{1})$. Ekvivalentno lahko vsakemu koraku pripišemo normalno porazdelitev q

$$\begin{aligned} q(\mathbf{z}_1|\mathbf{x}) &= \mathcal{N}_{\mathbf{z}_1}(\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbb{1})\,,^{3} \\ q(\mathbf{z}_t|\mathbf{z}_{t-1}) &= \mathcal{N}_{\mathbf{z}_t}(\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbb{1})\,. \end{aligned} \tag{6}$$

Ta zapis je enakovreden enačbam (5). Vsak zaporedni korak je **odvisen le od prejšnjega**, zato takemu procesu rečemo veriga Markova. Vprašamo se, če obstaja ekspliciten predpis, ki omogoča izračun verjetnostne gostote za t-ti korak, neposredno iz $\mathbf{x}$, saj je iterativno ustvarjanje zašumljenih slik preko enačb (6) časovno potratno. Z rekurzivnim vstavljanjem v izraz (5) izpeljemo t.i. **difuzijsko jedro** (angl. *diffusion kernel*)

$$q(\mathbf{z}_t|\mathbf{x}) = \mathcal{N}_{\mathbf{z}_t}(\sqrt{\alpha_t}\mathbf{x}, (1-\alpha_t)\mathbb{1})\,, \tag{7}$$

kjer smo uvedli $\alpha_t := \prod_{s=1}^{t}(1-\beta_s)$. Iz formule za difuzijsko jedro lahko vidimo, da po veliko korakih $t = T \gg 1$ velja $q(\mathbf{z}_T|\mathbf{x}) = \mathcal{N}(0, \mathbb{1})$, kot smo si zaželeli.

[3] Tam kjer iz konteksta ni jasno katera spremenljivka je porazdeljena po Gaussovi porazdelitvi, bo ta podpisana, na primer $x \sim \mathcal{N}_x(\mu, \sigma^2)$.

4.2 Dekodirnik

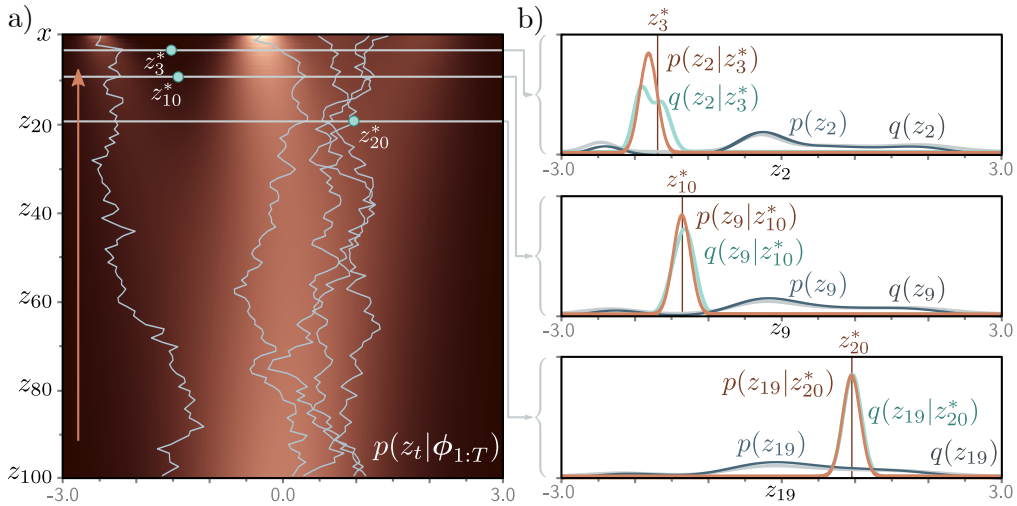
Dekodirnik je del modela, ki ga treniramo, da se nauči iz zašumljenih slik rekonstruirati prvotne podatke (gl. drugi del slike 4). Ideja je, da nam **majhni difuzijski koraki** omogočajo, da porazdelitve obratnih korakov $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$, ki jih analitično ne moremo izračunati, **aproksimiramo z Gaussovimi porazdelitvami**. Porazdelitve, ki se jih model uči, bomo označevali s $p(*|\boldsymbol{\phi})$, kjer bomo optimizirali parameter $\boldsymbol{\phi}$, da se približamo pravi obratni porazdelitvi $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$. Torej predpostavimo

$$p(\mathbf{z}_T) = q(\mathbf{z}_T|\mathbf{x}) = \mathcal{N}(0, \mathbb{1})\,, \tag{8}$$

$$p(\mathbf{z}_{t-1}|\mathbf{z}_t, \boldsymbol{\phi}_t) = \mathcal{N}_{\mathbf{z}_{t-1}}(\mathbf{f}_t(\mathbf{z}_t, \boldsymbol{\phi}_t), \sigma^2\mathbb{1})\,, \tag{9}$$

$$p(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\phi}_1) = \mathcal{N}_{\mathbf{x}}(\mathbf{f}_1(\mathbf{z}_1, \boldsymbol{\phi}_1), \sigma^2\mathbb{1})\,. \tag{10}$$

Mreža naj v vsakem koraku t izračuna povprečje $\boldsymbol{\mu}_{\boldsymbol{\phi}} := \mathbf{f}_t(\mathbf{z}_t, \boldsymbol{\phi}_t)$, kot smo definirali v poglavju 2.2.1, varianco pa fiksiramo [19].



Slika 5. Primer enorazsežnih porazdelitev (slika z enim pikslom), ki se jih model uči. Tu še lahko numerično izračunamo vzvratne porazdelitve $q(z_{t-1}|z_t)$ in opazimo, da so približki $p(z_{t-1}|z_t)$ upravičeni. Prikazani sta še celotni porazdelitvi $q(z_t)$ (siva) in naučena $p(z_t)$ (temno modra). Prirejeno po ref. [12].

Preden se lotimo minimiziranja izgube, moramo omeniti še, da medtem ko $q(\mathbf{z}_{t-1}|\mathbf{z}_t)$ ne znamo izračunati, pa lahko izračunamo obratno porazdelitev, če vemo, kakšen je bil vhodni podatek $\mathbf{x}$, saj po Bayesovem pravilu velja

$$q(\mathbf{z}_{t-1}|\mathbf{z}_t, \mathbf{x}) = \frac{q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x})q(\mathbf{z}_{t-1}|\mathbf{x})}{q(\mathbf{z}_t|\mathbf{x})} \propto q(\mathbf{z}_t|\mathbf{z}_{t-1})q(\mathbf{z}_{t-1}|\mathbf{x})\,.^4 \tag{11}$$

Obe porazdelitvi sta znani (gl. enačbi (6) in (7)). Enačba (11) se pojavi v poenostavitvi izraza za izgubo, zato je relevantna še njena pričakovana vrednost, ki je enaka pričakovani vrednosti zmnožka porazdelitev iz enačb (6) in (7)

$$\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{x}) = \frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}. \tag{12}$$

Sedaj se lotimo minimizacije izgube, ki je definirana enako kot v poglavju 2.2.1. Potrebujemo le konkreten predpis za izračun verjetnostne gostote $p(\mathbf{x}|\boldsymbol{\phi}_{1:T})$, kjer oznaka $1:T$ pomeni, da zajamemo vse korake med prvim in zadnjim. Slednjo lahko izrazimo s trikom projekcije (marginalizacije)

$$p(\mathbf{x}|\boldsymbol{\phi}_{1:T}) = \int p(\mathbf{x}, \mathbf{z}_{1:T}|\boldsymbol{\phi}_{1:T})\,\mathrm{d}\mathbf{z}_{1:T}\,,$$

[4] Ker sta zaporedna koraka odvisna le en od drugega, velja $q(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}) = q(\mathbf{z}_t|\mathbf{z}_{t-1})$.

kar nam omogoči, da po daljši izpeljavi prispemo do eksplicitnega zapisa za $\mathcal{L}(\boldsymbol{\phi}_{1:T})$, ki ga za razliko od začetnega znamo izvrednotiti:

$$\mathcal{L}(\boldsymbol{\phi}_{1:T}) = -\sum_{i=1}^{I} \log\Big(p\big(\mathbf{x}_i|\boldsymbol{\phi}_{1:T}\big)\Big) =$$

$$= -\sum_{i=1}^{I}\left(\underbrace{-\log\Big(\mathcal{N}_{\mathbf{x}_i}(\mathbf{f}_1(\mathbf{z}_{i1},\boldsymbol{\phi}_1),\sigma^2\mathbb{1})\Big)}_{\text{rekostrukcija slike / zadnji korak}} + \sum_{t=2}^{T}\frac{1}{2\sigma^2}\Big\|\underbrace{\boldsymbol{\mu}(\mathbf{z}_{it},\mathbf{x}_i)}_{\text{pr. vr. (11)}} - \underbrace{\mathbf{f}_t(\mathbf{z}_{it},\boldsymbol{\phi}_t)}_{\text{približek }\boldsymbol{\mu}_{\boldsymbol{\phi}}}\Big\|^2\right). \tag{13}$$

V prvem členu prepoznamo približek za obratno porazdelitev za zadnji korak enačbe (10), v drugem členu pa seštevamo prispevke vseh prejšnjih iterativnih korakov. Ker so porazdelitve Gaussove, se logaritmi verjetnostih gostot poenostavijo na kvadrat razlike ocenjenega povprečja porazdelitev na vsakem koraku $\mathbf{f}_t$ in pričakovane vrednosti porazdelitve (11).

S tako izgubo lahko zdaj učimo model s postopkom iz poglavja 2.2.2, a se izkaže, da se ta predpis še poenostavi, če $\mathcal{L}$ reparameteriziramo tako, da model predvidi dodani šum namesto pričakovane vrednosti. Iz jedra (7) lahko neposredno izrazimo $\mathbf{x}$ kot

$$\mathbf{x} = \frac{1}{\sqrt{\alpha_t}}\mathbf{z}_t - \frac{\sqrt{1-\alpha_t}}{\sqrt{\alpha_t}}\boldsymbol{\epsilon}\,,$$

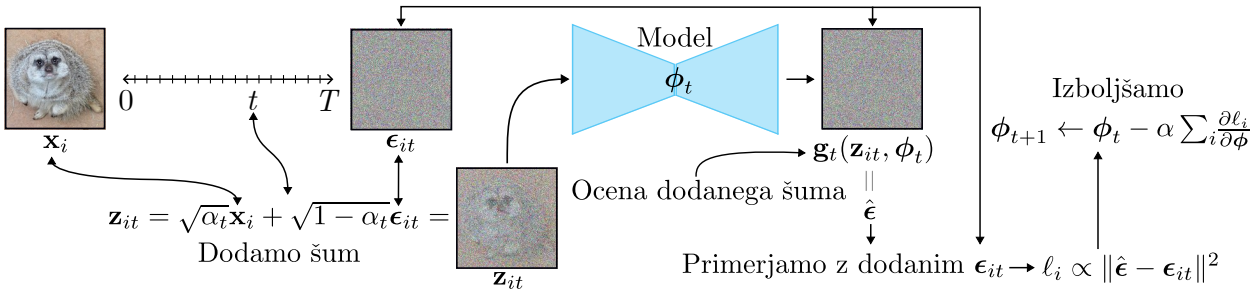
kar vstavimo v formulo za pričakovano vrednost $\boldsymbol{\mu}(\mathbf{z}_t,\mathbf{x})$. Členi v drugi vsoti v enačbi (13) se tako poenostavijo, če funkcijo $\mathbf{f}_t$, ki je prej računala povprečje, definiramo kot

$$\mathbf{f}_t(\mathbf{z}_{it},\boldsymbol{\phi}_t) = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\hat{\boldsymbol{\epsilon}}\,, \tag{14}$$

kjer z modelom zdaj računamo $\hat{\boldsymbol{\epsilon}} := \mathbf{g}_t(\mathbf{z}_t,\boldsymbol{\phi}_t)$ ali pričakovano vrednost šuma $\boldsymbol{\epsilon}_t$, ki smo ga uporabili, da smo $\mathbf{x}$ zašumili v $\mathbf{z}_t$. Bolj podroben prikaz je na sliki 6 [20]. Po ponovnem premetavanju izrazov pridemo do uporabne oblike funkcije izgube

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}_{1:T}) &= \sum_{i=1}^{I}\sum_{t=1}^{T} C_t\|\mathbf{g}_t(\mathbf{z}_{it},\boldsymbol{\phi}_t) - \boldsymbol{\epsilon}_{it}\|^2 = \\ &= \sum_{i=1}^{I}\sum_{t=1}^{T} C_t\|\mathbf{g}_t(\sqrt{\alpha_t}\mathbf{x}_i + \sqrt{1-\alpha_t}\,\boldsymbol{\epsilon}_{it},\boldsymbol{\phi}_t) - \boldsymbol{\epsilon}_{it}\|^2,\end{aligned} \tag{15}$$

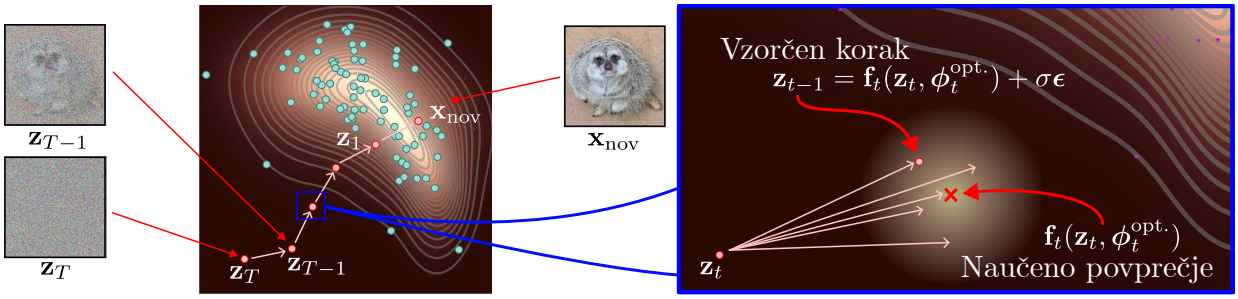
kjer se fenomenološko izkaže, da se model najbolje uči, če postavimo konstante C_t na 1 [12, 19].



Slika 6. Shematski prikaz učenja difuzijskega modela. Vhodne podatke $\mathbf{x}_i$ najprej zašumimo po enačbi (7) in jih pošljemo skozi model, ki skuša predvideti dodani šum ob koraku t za i-to testno sliko. S funkcijo izgube (15) izboljšamo napovedan šum.

Napovedovanje normaliziranega šuma omogoča, da izhod modela ostane v doslednem obsegu, kar mu omogoča, da se nauči izvajati fine izboljšave, saj šum dodajamo skoraj zvezno [2]. Nove slike

ustvarjamo tako, da iz Gaussovskega šuma vzorčimo $\mathbf{z}_T$ in z naučeno mrežo, ki predvidi šum po naučenih porazdelitvah $p(\mathbf{z}_{t-1}|\mathbf{z}_t, \boldsymbol{\phi}_t)$, postopoma vzorčimo vse manj zašumljene slike, kot prikazuje shema 7. Zdaj ostane le še apliciranje modela na fizikalnih meritvah.

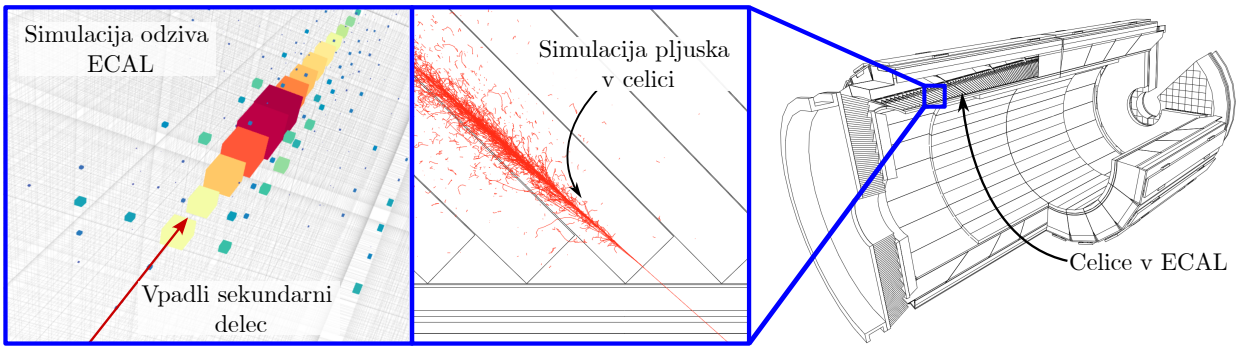


Slika 7. Generiranje novih podatkov. Začnemo z naključnim šumom $\mathbf{z}_T$ in se z naučenimi koraki vrnemo do novega podatka, ki leži nekje med vektorji $\{\mathbf{x}_i\}$, na katerih se je model učil (levo). Ko korakamo nazaj vzorčimo iz naučene Gaussove porazdelitve, kjer se je model naučil povprečje po enačbi (14), varianco σ^2 pa smo fiksirali (desno).

5. Kalorimetrični detektorji

V nadaljevanju se osredotočimo na generiranje novih eksperimentalnih podatkov v kalorimetrih, ki so narejeni za zajemanje energijskih pljuskov sekundarnih delcev, nastalih med trki v pospeševalniku. Elektromagnetni kalorimetri (ECAL), kot v detektorju CMS [21] (angl. *Compact muon solenoid*), merijo energije nastalih fotonov in elektronov s scintilacijskim kristalom (na primer $PbWO_4$). V scintilator prileti visokoenergijski sekundarni delec (γ ali $e^{\pm}$), kar povzroči plaz interakcij s snovjo, čemur pravimo elektromagnetni pljusk. Pri dovolj visokih energijah prevladujeta zavorno sevanje elektronov in nastanek parov e^+e^- iz fotonov. Vzbujen kristal ob relaksaciji odda scintilacijske fotone, ki jih izmerimo s fotodetektorji preko fotoelektričnega efekta. Nastali signal je tako sorazmeren z energijo vpadlih delcev, saj se ti v kristalih ustavijo in tako odložijo vso svojo kinetično energijo [22].

Detektorji v ECAL so ločeni v posamezne celice (angl. *voxel*), kjer vsaka nameri delež vpadne energije vpadlega delca (gl. sliko 8 levo). Meritve tako spominjajo na slike – celice na piksle, oddana energija pa na barvo, kar porodi idejo o uporabi generativnih algoritmov [2, 23].



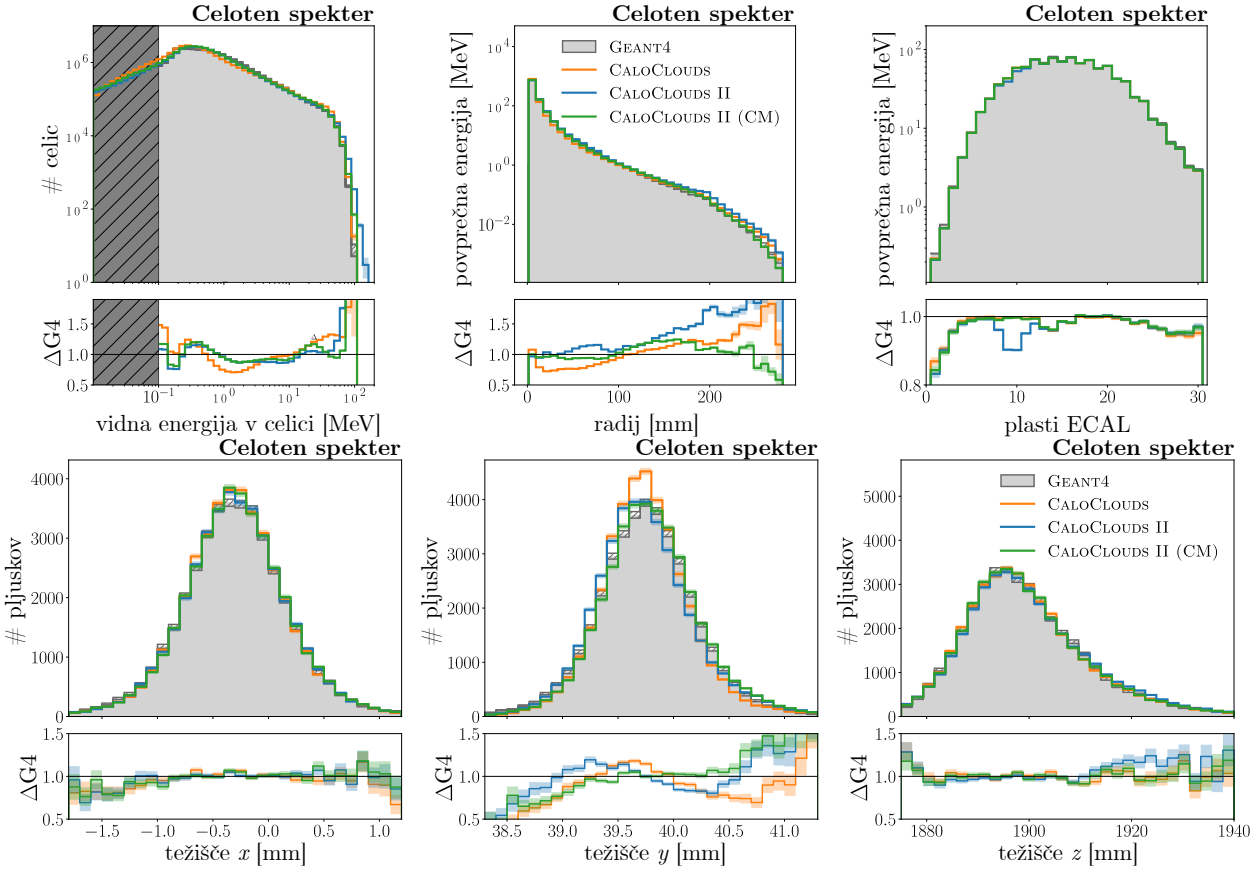
Slika 8. Shematski prikaz elektromagnetnega kalorimetra, ki je sestavljen iz posameznih scintilacijskih kristalov v obliki celic (desno). Slika na sredini prikazuje pljusk delca, ki priletel v celico, leva slika pa simulacijo odziva ECAL, kot bi ga namerili pri pljusku. Prirejeno po ref. [23].

6. Primeri uspešne uporabe

Kot merilo za kvaliteto ustvarjenih simulacij z algoritmi strojnega učenja se uporabljajo MC simulacije odprtokodnega programskega paketa za simulacijo prehoda delcev skozi snov GEANT4, ki temelji na fizikalnih interakcijah delcev z materialom. Program iz podane geometrije detektorja in podatkov o vpadlih delcih na podlagi interakcij s scintilatorjem (gl. 5. poglavje) simulira pljusk [4, 24]. Časovna zahtevnost teh simulacij predstavlja bistven problem, kar motivira iskanje alternativnih pristopov, kot so generativni modeli.

Bolj podrobno si poglejmo rezultate difuzijskih modelov CALODIFFUSION [2], CALOCLOUDS [4] in CALOCLOUDSII [11], ki so sodelovali na tekmovanju CALOCHALLENGE [3]. Vsi trije temeljijo na difuzijskem algoritmu iz 4. poglavja, a ima vsak rahlo drugačen pristop. Uspešnost se vrednoti s primerjavo različnih generiranih odzivov kalorimetra na vpadle delce, kot so porazdelitev energije po celicah, radialni in vzdolžni profili pljuskov, njihova težišča ($\overline{x} = \langle x_i E_i \rangle / \sum_i E_i$) in porazdelitev zajete energije.

CALODIFFUSION je model, ki uporablja standarden difuzijski algoritem iz 4. poglavja s 400 koraki. Čeprav običajno uporabljamo konvolucijske mreže, ki so invariantne na translacije, si tega tu ne želimo, saj nam prostorske porazdelitve kristalu oddane energije dajo pomembne informacije o trku. Zagato s periodičnostjo meritev v kotni smeri avtorji odpravijo s cilindričnimi konvolucijami (periodičnimi robnimi pogoji). Poleg tega ta model uporablja poseben algoritem, s katerim neenakomerno velike celice ECAL preslika v enako velike in jih po simulaciji z inverzom preslika na prvotno geometrijo. Model se je na tekmovanju uvrstil med vodilna mesta [2, 3].



Slika 9. Generirani rezultati modelov CALOCLOUDS, CALOCLOUDSII in CM različice v primerjavi z MC simulacijo na GEANT4. Simulacije so generirane iz celotnega spektra, torej so energije vstopnih delcev enakomerno vzorčene med 10 in 90 GeV. Prikazane so porazdelitev energij v ECAL, radialna ter vzdolžna porazdelitev deponiranih energij ter težišča trkov. Pod grafi so relativna odstopanja od MC simulacije (ΔG4) [11].

V ozadju modela CaloClouds se nahaja enak difuzijski algoritem, kot v 4. poglavju, ki ga že poznamo. Drugi, CaloCloudsII, pa nadgrajuje predhodnika, a ohranja enake ideje. Temelji na Langevinovi dinamiki in točkovnem ujemanju (angl. *score-based generative modeling*), ki predstavlja alternativni pogled na difuzijske algoritme. Razumemo ga lahko kot zvezno različico postopka, opisanega v 4. poglavju – po Langevinovi dinamiki zašumljanje podatkov zdaj opisuje stohastična diferencialna enačba, ki vsebuje člene z naključnim šumom. Točkovno ujemanje pomeni, da se model ne uči predvideti dodanega šuma, temveč se poskuša direktno naučiti obliko vektorskega polja gradienta logaritma porazdelitve podatkov. V tem primeru je izhod modela funkcija $s(\mathbf{z}_t, t, \boldsymbol{\phi})$, s katero želimo najbolje aproksimirati gradient. V skladu s tem lahko iz začetnega normalnega šuma model zvezno sestopa do novih podatkov prek reševanja diferencialne enačbe, v kateri je $\sigma^2(t)$ varianca, dodana ob času t. Tu jo podajmo brez izpeljave, bolj za občutek

$$\frac{\mathrm{d}\mathbf{z}_t}{\mathrm{d}t} = -\frac{1}{2}\sigma^2(t)s(\mathbf{z}_t, t, \boldsymbol{\phi}_{\text{opt.}})\,.$$

Esenco diskretnega modela torej ohranimo, od ustvarjanja novih podatkov pa nas loči le še numerična integracija.

CaloCloudsII uvede še model doslednosti (angl. *consistency model – CM*). Osnovna ideja je s pomočjo originalnega modela (CaloCloudsII) naučiti novega, kako generirati nove pljuske iz naključnega šuma le v enem koraku, namesto v več deset do sto kot pri osnovnih modelih. To posledično pospeši simulacijo, kar je razvidno iz tabele 1. Skladno s člankom [11] so omenjene različice med 1,2-krat do približno 1900-krat hitrejše od MC simulacij, odvisno od strojne opreme in uporabljenega modela. Prednost v hitrosti modelov postane očitna, ko jih poganjamo na grafičnih karticah zaradi zmožnosti paralelnega računanja. Trenutno se modele trenira na grafičnih karticah, zato lahko v praksi brez težav privzamemo pospešitev za vsaj dva reda velikosti.

Vse različice so se izkazale za uspešne, a so glede na metrike uspeha potrebne še nadgradnje, da bo natančnost modelov konkurenčna klasičnim simulacijam. Za občutek so nekateri končni rezultati generiranih porazdelitev merljivih količin v kalorimetru prikazani na sliki 9 [4, 11]. Čeprav na prvi pogled rezultati izgledajo obetavni, model za ločevanje med pljuski difuzijskih modelov in pljuski, simuliranimi z MC metodo, uporabljen kot ena od metrik uspeha, z lahkoto razloči sintetične podatke od pravih [11].

Tabela 1. Primerjava hitrosti treh različic modelov CaloClouds (CC) s klasično MC simulacijo na Geant4. Očitno hitrost močno variira s strojno opremo in različico modela – grafične kartice ponujajo velik skok v hitrosti zaradi zmožnosti paralelnega računanja. Prirejeno po ref. [11].

Strojna oprema	Simulator	Velikost serije	Čas / pjusk [ms]	Pospešitev
CPE	GEANT4		3914.80 ± 74.09	$\times 1$
	CC	1	3146.71 ± 31.66	$\times 1.2$
	CC II	1	651.68 ± 4.21	$\times 6.0$
	CC II (CM)	1	84.35 ± 0.22	$\times 46$
GPE	CC	64	24.91 ± 0.72	$\times 157$
	CC II	64	6.12 ± 0.13	$\times 640$
	CC II (CM)	64	2.09 ± 0.13	$\times 1873$

7. Sklep

Spoznali smo, da se z naraščajočimi računskimi potrebami pri simuliranju trkov odpirajo inovativne rešitve z generativnimi modeli strojnega učenja. Za uspešnejše so se izkazali novejši difuzijski algoritmi, ki temeljijo na odvzemanju informacij vhodnim podatkom (dodajanju naključnega šuma) in učenju obratnega procesa postopnega odstranjevanja dodanega šuma do rekonstrukcije testnih

vzorcev. Prek maksimiranja skupne verjetnostne porazdelitve podatkov glede na testne smo izpeljali funkcijo izgube v splošnem in si pogledali, kako jo izračunamo na dva načina v primeru difuzijskih algoritmov. Za konec smo se seznanili še s konkretnimi uspešnimi primeri uporabe teh modelov kot nadomestek za klasične MC simulacije.

LITERATURA

[1] CERN, *High-Luminosity LHC*, (2024). URL:https://home.cern/science/accelerators/high-luminosity-lhc. Dostop: 29. 3. 2025.

[2] O. Amram, K. Pedro, *Denoising diffusion models with geometry adaptation for high fidelity calorimeter simulation*, arXiv **2308.03876** (2023).

[3] C. Krause in sod, *CaloChallenge 2022: A Community Challenge for Fast Calorimeter Simulation*, arXiv **2410.21611**, (2024).

[4] E. Buhmann in sod, *CaloClouds: fast geometry-independent highly-granular calorimeter simulation*, Journal of Instrumentation **18.11** (2023).

[5] O. Amram, K. Pedro, *Simulating the CMS High Granularity Calorimeter with ML*, (2024).

[6] OpenAI. *Sora: Creating video from text*. https://openai.com/sora. 2024. Dostop: 19. 4. 2025.

[7] OpenAI. *ChatGPT*. https://openai.com/index/chatgpt/. 2022. Dostop: 19. 4. 2025.

[8] Midjourney, Inc. *Midjourney*. https://www.midjourney.com. 2022. Dostop: 19. 4. 2025.

[9] OpenAI. *DALL·E 3*. https://openai.com/dall-e-3. 2023. Dostop: 19. 4. 2025.

[10] S. Vallecorsa, *Generative models for fast simulation*, Journal of Physics: Conference Series **1085.2** (2018).

[11] E. Buhmann in sod, *CaloClouds II: Ultra-Fast Geometry-Independent Highly-Granular Calorimeter Simulation*, arXiv **2309.05704** (2024).

[12] S.J.D. Prince, *Understanding Deep Learning*, The MIT Press, 2023.

[13] T. Szandała. *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks*, *Bio-inspired Neurocomputing*, Ur. A.K. Bhoi in sod, Singapore: Springer Singapore (2021), 203–224.

[14] J.W. Lin. *Artificial neural network related to biological neuron network: a review*", Advanced Studies in Medical Sciences **5** (2017), 55–62.

[15] O. Ronneberger, P. Fischer in T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. Cham: Springer International Publishing (2015), 234–241.

[16] ATLAS kolaboracija, *The ATLAS Experiment at the CERN Large Hadron Collider*, Journal of Instrumentation **3.08** (2008).

[17] LHCb kolaboracija, *The LHCb Detector at the LHC*, Journal of Instrumentation **3.08** (2008).

[18] T. Abe in sod, *Belle II Technical Design Report*, arXiv **1011.0352** (2010).

[19] J. Ho, A. Jain in P. Abbeel, *Denoising Diffusion Probabilistic Models*, arXiv **2006.11239** (2020).

[20] L. Weng, *What are diffusion models?*, lilianweng.github.io (2021). URL: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/. Dostop: 27. 3. 2025.

[21] CMS kolaboracija, *The CMS experiment at the CERN LHC*, Journal of Instrumentation **3.08** (2008).

[22] J.B. Sauvan, *Calorimetry in HEP From concepts to experiments*, (2020). URL: https://indico.cern.ch/event/855973/contributions/3602188/attachments/1979913/3296643/calorimetry_sauvan_esipap2020.pdf. Dostop: 29. 3. 2025.

[23] CMS kolaboracija, *The CMS electromagnetic calorimeter project: Technical Design Report*, Technical design report. CMS. Geneva: CERN, (1997).

[24] S. Guatelli in sod, *Introduction to the geant4 simulation toolkit*, AIP Conference Proceedings **1345** (2011).