

Universidade de São Paulo
Instituto de Física

Estudo da reação de breakup ${}^4\text{He}({}^{17}\text{F}, {}^{16}\text{O}+\text{p}){}^4\text{He}$
usando o alvo ativo pAT-TPC: uma abordagem
usando técnicas de Machine Learning

Guilherme Ferrari Fortino

Orientador: Prof. Dr. Valdir Guimarães _____

Coorientador: Dr. Juan Carlos Zamora Cardona _____

Dissertação de mestrado apresentada ao Instituto de
Física da Universidade de São Paulo, como requisito
parcial para a obtenção do título de Mestre(a) em Ciências.

Banca Examinadora:

Prof(a). Dr(a). Nome do(a) Professor(a) - Orientador (instituição de trabalho)

Prof(a). Dr(a). Nome do(a) Professor(a) (instituição de trabalho)

Prof(a). Dr(a). Nome do(a) Professor(a) (instituição de trabalho)

São Paulo
2022

Sumário

1	Introdução	8
2	O experimento	9
2.1	Produção do feixe secundário ^{17}F usando o sistema TWINSOL	9
2.2	O alvo ativo pAT-TPC	13
3	Desenvolvimento de ferramentas de <i>machine learning</i> para análise de dados	17
3.1	Tipos de redes neurais	18
3.2	Construção de uma rede neural	20
3.3	Sistemas de <i>machine learning</i>	22
3.4	Aplicações de <i>machine learning</i> na física nuclear	24
3.4.1	Análise de espectros para identificação de partículas (<i>particle identification</i> , PID)	24
3.4.2	Estimativa de raios e massas nucleares	25
3.4.3	Decaimento beta e processo r	27
3.4.4	Alvos ativos	28
4	Reconstrução de nuvens de pontos	30
4.1	Análise dos pulsos com algoritmos tradicionais	30
4.1.1	Remoção do fundo	31
4.1.2	Deconvolução do sinal	34
4.2	Análise dos pulsos com <i>machine learning</i>	36
4.2.1	Rede neural para o fundo	37
4.2.2	Rede neural para a deconvolução	39
4.2.3	Unificando as duas redes neurais	41
4.2.4	Detecção de picos	45
5	Análise das nuvens de pontos	49
5.1	Método usual	51

5.2	Métodos com <i>machine learning</i>	56
5.2.1	Clusterização hierárquica	56
6	Resultados	59
7	Conclusão	61

Listas de figuras

2.1	Sistema TWINSOL à esquerda e pAT-TPC à direita. O feixe estável de ^{16}O entra à esquerda do TWINSOL, para então ser produzido o feixe secundário ^{17}F que irá ser conduzido até o alvo ativo pAT-TPC. Todo o sistema está localizado na University of Notre Dame.	10
2.2	Simulação computacional do sistema RIBRAS, onde as partículas carregadas em laranja surgem do ponto à direita daq figura e passam pelos dois solenoides em verde, para então serem focalizada em um ponto do plano focal em vermelho. A parte em preto dos solenoides corresponde aos limites físicos da bobina[5].	11
2.3	Valor do campo magnético B em tesla em função da posição em z em centímetro da bobina. A linha vertical tracejada preta indica o limite físico da bobina. O campo foi calculado à uma distância de 8 cm do eixo do solenoide. É possível ver claramente o efeito de borda que há em um solenoide finito[9].	11
2.4	Espectro biparamétrico $\Delta E - E$ de identificação de partículas. Nele é possível identificar que o feixe possui a presença de ^{17}F , ^{16}O e uma pequena parte de ^{17}O	12
2.5	Figura esquemática do pAT-TPC e o detector <i>Micromegas</i> [2].	13
2.6	Plano do <i>Micromegas</i> com a esquematização das <i>thick gems</i> . Os elétrons quando passam para um campo elétrico mais intenso ionizam o gás, produzindo ainda mais elétrons (evento chamado de avalanche de elétrons). Cada canal da eletrônica de saída produz um pulso como mostrado na parte de baixo da figura.	15
2.7	Evento reconstruído a partir da análise dos pulsos gerados pelo <i>Micromegas</i> . A cor representa a carga integrada de cada ponto de interação com o gás. . .	16

3.1	Exemplo de FFNN. A camada de entrada na esquerda propaga a informação para a direita (camada de saída). Todos os neurônios entre camadas estão conectados entre si.	18
3.2	Processo de convolução entre sinal azul em cima e o filtro em verde, resultando no sinal azul embaixo. A multiplicação é feita ponto a ponto e está indicada na caixa azul-clara.	19
3.3	Processo de convolução entre o sinal azul em cima e o filtro em verde, resultando no sinal azul embaixo. Agora são acrescentados zeros no inicio e no final do vetor para que o vetor saída tenha o mesmo tamanho do vetor de entrada (nesse caso 9).	20
3.4	Funções de ativação e seus respectivos gráficos.	21
3.6	Histograma de identificação de partículas, a partir do conjunto 1 identificado na figura 3.5b. Em a temos o primeiro pico que corresponde ao estamos fundamental do ^{14}N e em b temos o primeiro estado excitado do ^{14}N	25
3.5	25
3.7	Painel acima mostra a localização dos núcleos usados para o treino da rede neural. No painel de baixo temos o erro entre a previsão e o valor experimental da energia de ligação do núcleo para os núcleos usados como dados de validação. σ é o erro quadrático médio da rede neural.	26
3.8	Previsões para o raio de carga para isótopos do chumbo ($Z = 82$) para diferentes modelos teóricos ou que usam redes neurais. A previsão que contém barras de erro é a que foi brevemente descrita no texto.	27
3.9	Meias-vidas de decaimento β para isótonos com $N = 126$. A região hachurada verde mostra as previsões de uma rede neural. A região hachurada em azul mostra os resultados da mesma rede neural, porém seus dados de aprendizado são estendidos para incluir três meias-vidas extras de decaimento β para cada isótopo (indicado por círculos abertos) em direção à <i>drip-line</i> de nêutrons[43].	28
3.10	Projeções no plano xy de partículas dentro do alvo ativo, onde quanto mais escura for a cor, mais carga tem o ponto, e quanto mais clara a cor, menos carga tem o ponto. O objetivo da rede neural pode ser classificar corretamente se a imagem à direita corresponde à uma trajetória de um próton, carbono ou outra partícula, ou pode ser uma rede neural para classificação binária caso seja necessário classificar apenas entre próton ou carbono[44].	29

4.1	Exemplos de sinais produzidos pelos canais do detector. Em 4.1a o sinal possui apenas um pulso, enquanto em 4.1b há vários pulsos em sobreposição, formando um único pulso com largura maior que em 4.1a.	30
4.2	Ilustração que mostra a variação no formato da carga coletada a partir da passagem de uma partícula carregada dentro do TPC, onde o plano do detector está embaixo. No lado esquerdo de cada imagem, a distribuição do sinal coletado por um único pad (escuro) do plano de coleta é mostrado (o canal eletrônico de leitura é representado pela seta cinza em negrito). No caso de uma trajetória quase horizontal (a), o sinal é uma distribuição estreita, enquanto para uma trajetória próxima a uma direção vertical (b), a distribuição deve ser muito mais ampla (vários picos devem ser extraídos desse sinal). A última imagem ilustra o caso em mais de uma trajetória de partículas contribui para o sinal[14].	31
4.3	Histogramas com as respectivas <i>baselines</i> (linhas tracejadas) estimadas pelo método da convolução. O espectro resultante (sem o fundo) está em verde.	33
4.4	Histogramas com as respectivas <i>baselines</i> (linhas tracejadas) calculadas pelo <i>TSpectrum</i>	34
4.5	Histogramas sem as <i>baselines</i> antes (em azul) e depois da deconvolução (em vermelho). Os picos (em verde) e o limiar (linha tracejada preta) de detecção também estão indicados.	35
4.6	Exemplos de eventos reconstruídos através da análise dos sinais. A seta vermelha indica o sentido do feixe.	36
4.7	Arquitetura da rede neural que faz a inferência do fundo. O vetor de entrada deve ter dimensionalidade 512 x 1. Todas as partes com convolução não possuem o parâmetro <i>bias</i>	37
4.8	Resultados do treino da rede neural dada por 4.7. A rede atingiu seu melhor resultado a partir da <i>epoch</i> 20 aproximadamente, quando começa um platô no <i>loss</i>	38
4.9	Exemplos da rede neural dada por 4.7 em comparação com a saída do <i>TSpectrum</i>	39
4.10	Arquitetura da rede neural que faz a inferência da deconvolução do espectro. O vetor de entrada deve ter dimensionalidade 512 x 1. Todas as partes com convolução não possuem o parâmetro <i>bias</i>	40
4.11	Resultados do treino da rede neural dada por 4.7.	41
4.12	Exemplos de deconvolução da rede neural dada por 4.7.	41

4.13 Arquitetura da rede neural que faz a inferência da <i>baseline</i> e depois faz a deconvolução do espectro. O vetor de entrada deve ter dimensionalidade 512 x 1.	42
4.14 Histograma bidimensional que mostra em <i>x</i> a contagem do número de picos detectados por sinal (dado pelo <i>TSpectrum</i>) pela deconvolução e em <i>y</i> a contagem do número de picos detectados por sinal (resultante da rede neural) pelo <i>scipy</i> . O número de contagens está marcado em cima de cada <i>bin</i>	43
4.15 Resultados da reconstrução de eventos por <i>machine learning</i>	44
4.16 Histograma bidimensional que mostra a relação, de cada pico detectado, entre a amplitude do pico após a deconvolução no eixo <i>y</i> e antes da deconvolução no eixo <i>x</i> . A linha tracejada indica a tendência da maior parte dos pontos. Já a linha sólida indica a região de corte dos pontos.	45
4.17 Sinal aps a deconvolução que mostra o pico detectado mais os pontos adicionais que irão facilitar o trabalho da rede neural (evitar o desbalanço de classe). Foram acrescentados 2 pontos à esquerda e à direita.	46
4.18 Arquitetura da rede neural que faz o recorte das regiões com picos. O vetor de entrada deve ter dimensionalidade 512 x 1.	47
4.19 Resultados do treino da rede neural dada por 4.18.	47
4.20 Exemplos de deconvolução da rede neural dada por 4.7.	48
 5.1 Sequênciа de análise de um evento. Em 5.1a temos o evento que é recebido para ser analisado, em 5.1b temos o mesmo evento aps o RANSAC (antes da clusterização) e 5.1c mostra depois da correção. As cores das retas são arbitrárias e servem apenas para a diferenciação.	50
5.2 Evento em que não foi detectado o feixe, apenas a partícula espalhada. O triângulo azul é o local calculo do vértice de reação dado pela equação 5.7.	54
5.3 Exemplos dos resultados para o HDBSCAN. Percebe-se que em 5.3c e 5.3d o algoritmo falhou, juntando diferentes <i>clusters</i> ou simplesmente detectando ruído junto da <i>track</i>	57
 6.1 Histograma de comprimento de <i>track</i> no eixo <i>y</i> e ângulo de espalhamento no eixo <i>x</i> . O histograma foi feito coletando eventos que possuíam duas trajetórias com o mesmo vértice de reação, indicando a detecção simultânea do ^{16}O e do próton.	59

Capítulo 1

Introdução

Capítulo 2

O experimento

Esse capítulo irá descrever sobre a parte experimental desse trabalho que foi desenvolvido na Universidade de Notre Dame em Outubro de 2019. Nesse experimento o feixe radioativo ^{17}F foi produzido e selecionado por rigidez magnética pelo sistema TWINSOL (TWIN SOLenids)[1]. O alvo ativo *prototype Active Target - Time Projection Chamber* (pAT-TPC) [2] foi usado como alvo e detector simultaneamente. Na continuação, será descrito como foi feito o sistema de produção do feixe secundário ^{17}F e a detecção das reações induzidas por esse feixe no alvo ativo.

2.1 Produção do feixe secundário ^{17}F usando o sistema TWINSOL

A produção do feixe radioativo ^{17}F foi feita a partir da reação do feixe estável de ^{16}O com um alvo de deutério. O feixe primário ^{16}O foi produzido e acelerado por um acelerador tipo Tandem, que possui uma tensão terminal de até 10 MV[3, 4]. O feixe é conduzido até a câmara alvo de produção, no início do sistema TWINSOL, onde partículas emergentes de reações nucleares irão surgir. A figura 2.1 mostra o sistema TWINSOL acoplado com o pAT-TPC.

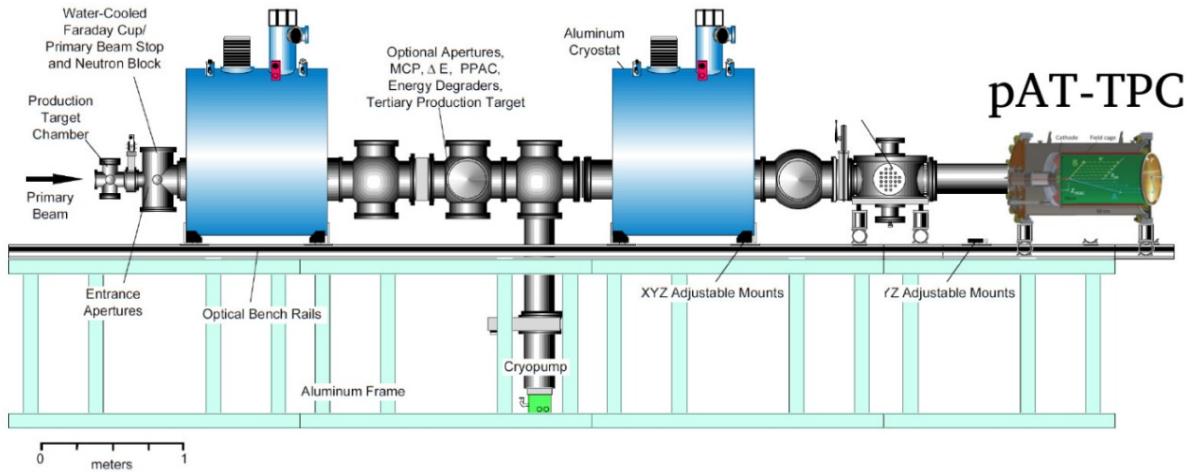


Figura 2.1: Sistema TWINSOL à esquerda e pAT-TPC à direita. O feixe estável de ^{16}O entra à esquerda do TWINSOL, para então ser produzido o feixe secundário ^{17}F que irá ser conduzido até o alvo ativo pAT-TPC. Todo o sistema está localizado na University of Notre Dame.

O TWINSOL é um sistema de produção de feixes radioativos em voo que possui dois solenoides supercondutores alinhados que são usados para produzir, coletar, transportar, focar e analisar feixes estáveis e radioativos. O sistema se baseia na seleção de partículas a partir da sua rigidez magnética ($B\rho$)[1, 5, 6]. Cada solenoide possui 30 cm de raio interno e 1 m de comprimento[1]. O fato de ser um solenoide finito faz com que surjam efeitos de borda na componente radial do campo magnético do solenoide, cujo efeito faz com que o solenoide seja capaz de focalizar partículas. Para entender melhor o efeito de borda no campo magnético, e consequentemente o funcionamento do TWINSOL, simulações computacionais usando a biblioteca GEANT4[7] foram feitas usando a geometria do sistema “irmão” do TWINSOL, o Radioactive Ion Beams in Brasil (RIBRAS), que também possui dois solenoides supercondutores alinhados[5, 8]. A simulação da figura 2.2 mostra os dois solenoides de cor verde focalizando as partículas de cor laranja em um ponto do plano focal em vermelho. O campo magnético em função da posição do eixo, de cada solenoide, está na figura 2.3.

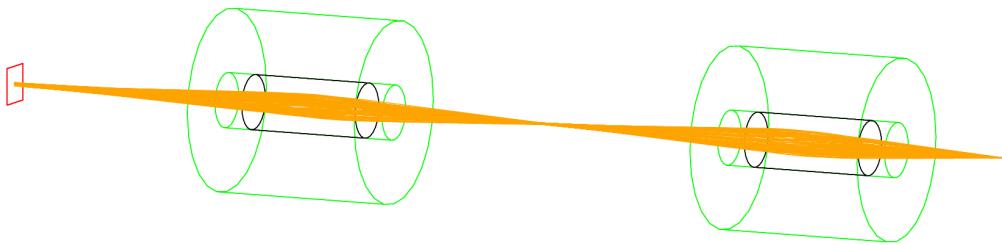


Figura 2.2: Simulação computacional do sistema RIBRAS, onde as partículas carregadas em laranja surgem do ponto à direita da figura e passam pelos dois solenoides em verde, para então serem focalizada em um ponto do plano focal em vermelho. A parte em preto dos solenoides corresponde aos limites físicos da bobina[5].

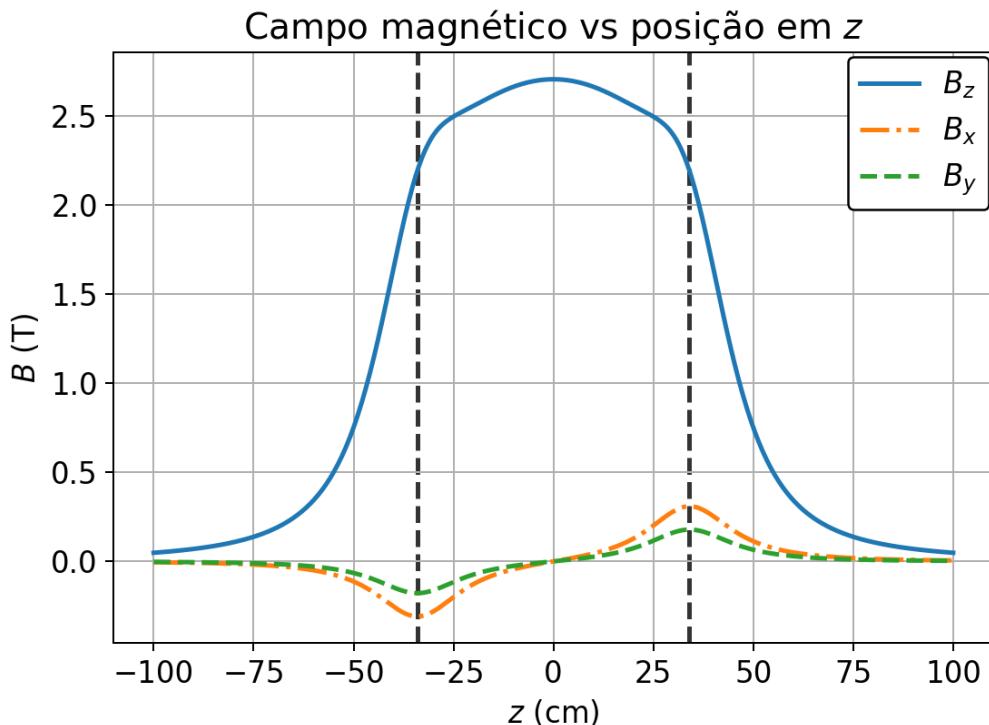


Figura 2.3: Valor do campo magnético B em tesla em função da posição em z em centímetro da bobina. A linha vertical tracejada preta indica o limite físico da bobina. O campo foi calculado à uma distância de 8 cm do eixo do solenoide. É possível ver claramente o efeito de borda que há em um solenoide finito[9].

A trajetória das partículas dentro do solenoide é helicoidal e, como os solenoides funcionam como uma lente óptica, é possível calibrar o ponto focal. Em uma aproximação de um solenoide como uma lente grossa, o foco depende da rigidez magnética da partícula através da relação[3, 6]:

$$\frac{1}{f} = \frac{B_z^2}{(B\rho)^2}, \quad (2.1)$$

onde f é o ponto focal, B_z a componente z do campo magnético, e $B\rho$ é dado por:

$$B\rho = \frac{mv}{q} = \frac{\sqrt{2mE}}{q}, \quad (2.2)$$

onde E é a energia, m sua massa e q seu estado de carga.

Para produzir ^{17}F , uma célula gasosa localizada na câmara alvo de produção[1] preenchida com deutério foi bombardeada pelo feixe primário. Além de ^{17}F , outras partículas também são geradas, como o ^{17}O , além do feixe de ^{16}O que pode apenas ser espalhado. Outras reações, por exemplo, com a estrutura da célula gasosa (janela feita de $5\ \mu\text{m}$ de titânio[1]) podem ocorrer, mas o papel do TWINSOL é de selecionar e focalizar apenas as partículas desejadas.

Mesmo para partículas diferentes, o B_ρ pode ser muito próximo ou igual. Isso faz com que não seja possível obter um feixe de ^{17}F com 100% de pureza, e sim um coquetel de partículas[6]. O coquetel de partículas produzido possui 54% de ^{17}F , 41% de ^{16}O e cerca de 5% de ^{17}O . A figura 2.4 mostra o espectro biparamétrico de identificação de partículas, onde é possível identificar as partículas que estão presentes no feixe (coquetel de partículas). Por fim, o feixe produzido pelo TWINSOL é conduzido até o pAT-TPC.

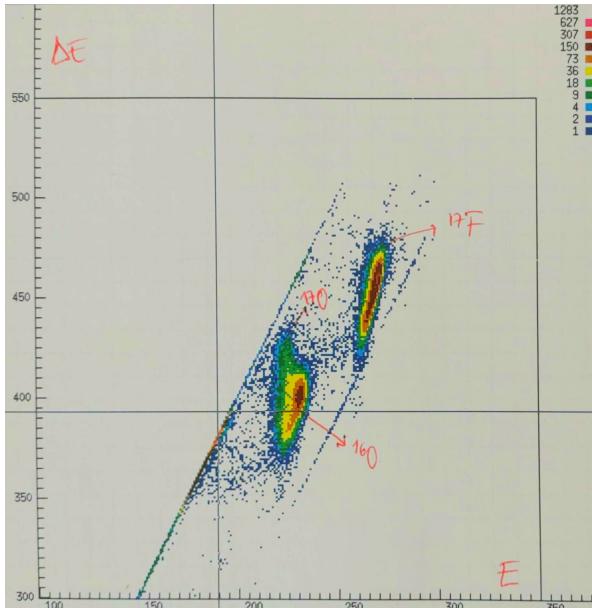
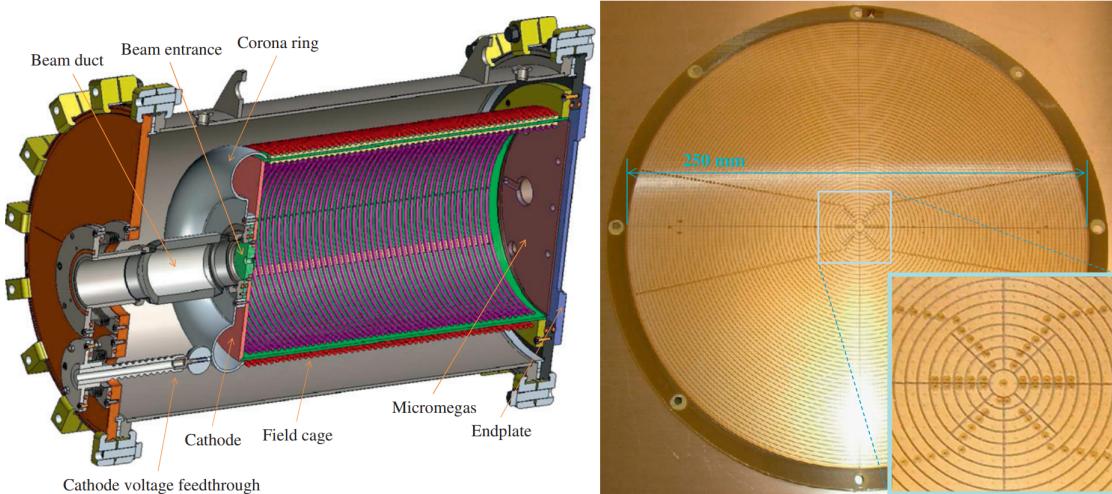


Figura 2.4: Espectro biparamétrico $\Delta E - E$ de identificação de partículas. Nele é possível identificar que o feixe possui a presença de ^{17}F , ^{16}O e uma pequena parte de ^{17}O .

2.2 O alvo ativo pAT-TPC

Essa seção irá descrever brevemente propriedades geométricas e físicas relacionadas ao sistema de detecção usado neste experimento, o alvo ativo pAT-TPC.

A figura 2.5a mostra o desenho esquemático do pAT-TPC. O detector possui uma cela cilíndrica de 50 cm de comprimento e 28 cm de diâmetro, onde o seu eixo é alinhado com o eixo do feixe[2], que entra pelo duto central. A câmara é preenchida com o ${}^4\text{He}$ gasoso puro à uma pressão de 350 Torr que serve tanto para alvo de reações nucleares, quanto para a própria medição e detecção dos produtos da reação[2, 10]. Tanto o feixe quanto partículas originadas da reação ionizam o gás e os elétrons que surgem dessa ionização são conduzidos por um campo elétrico de 1 kV/cm perpendicular ao eixo da câmara até o plano detector (*pad plane*), o *Micromegas*[11], mostrado na figura 2.5b.



(a) Visão transversal do pAT-TPC. O gás é preenchido dentro da cela que possui um campo elétrico perpendicular ao plano do *Micromegas*, à direita da figura. O feixe incide na câmara entrando pelo duto de feixe à esquerda da figura.

(b) Foto do *Micromegas*. O detector é multi-pixelado com uma maior densidade no centro, parte destacada na imagem. O *pad* central tem diâmetro de 5 mm enquanto que as faixas coaxiais possuem passo de 2mm[12, 13].

Figura 2.5: Figura esquemática do pAT-TPC e o detector *Micromegas*[2].

O *Micromegas* é um dispositivo de amplificação de elétrons, que consiste em um plano detector com 2048 canais (*pads*) triangulares com eletrônica independente, que usa o *Generic Electronics for TPCs* (GET)[14]. Detalhes sobre a eletrônica podem ser encontrados nas Refs. [14, 13]. O formato triangular dos canais tem como objetivo maximizar a resolução espacial do detector. Cada canal possui uma posição (x, y) fixa e a terceira coordenada z será determinada a partir do tempo de deriva dos elétrons no gás[2, 10, 12, 13]. Isso só é possível pois a velocidade de deriva (*drift*) dos elétrons é constante[15], portanto a posição em z da partícula é diretamente proporcional ao tempo

de voo. Esse princípio que deu origem ao nome de *Time Projection Chamber*, pois o evento é projetado no tempo de deriva dos elétrons no gás. A equação 2.3 (equação de Langevin) descreve o movimento de um elétron com massa m e carga e é descrito por[15]

$$m \frac{d\vec{v}}{dt} = e (\vec{E} + \vec{v} \times \vec{B}) - \frac{m}{\tau} \vec{v}, \quad (2.3)$$

onde \vec{E} é o vetor campo elétrico, \vec{B} o vetor campo magnético, \vec{v} é o vetor de velocidade do elétron e τ é o tempo de colisão médio, que depende das propriedades termodinâmicas do gás. No caso do pAT-TPC, \vec{B} é zero e a solução estacionária para a velocidade de *drift* do elétron é

$$\vec{v} = \frac{\tau}{m} e \vec{E}. \quad (2.4)$$

A velocidade de deriva depende das propriedades termodinâmicas do gás (temperatura, pressão) e também de sua condutividade elétrica[15]. Isso significa que é a calibração da velocidade envolve a calibração não só do campo elétrico, mas também das propriedades do gás dentro do alvo ativo[2, 15]. Para acharmos a coordenada z , basta integrar a equação 2.4 para obter

$$z = \frac{\tau}{m} e \vec{E} (t - t_0), \quad (2.5)$$

onde no tempo $t_0 = 0$ o elétron está no plano do detector ($z = 0$).

O *Micromegas* possui *thick gems*. *Thick gems* usam do fato de que, no momento em que o elétron passa para uma região de campo elétrico ordens de grandeza maior que de sua origem, ocorre a ionização secundária (quando o elétron ioniza o gás). Isso provoca o que é chamado de avalanche de elétrons, amplificando a intensidade do sinal recebido[14]. A figura 2.6 mostra a esquematização do *Micromegas*, onde na eletrônica de saída é produzido um sinal em função do tempo.

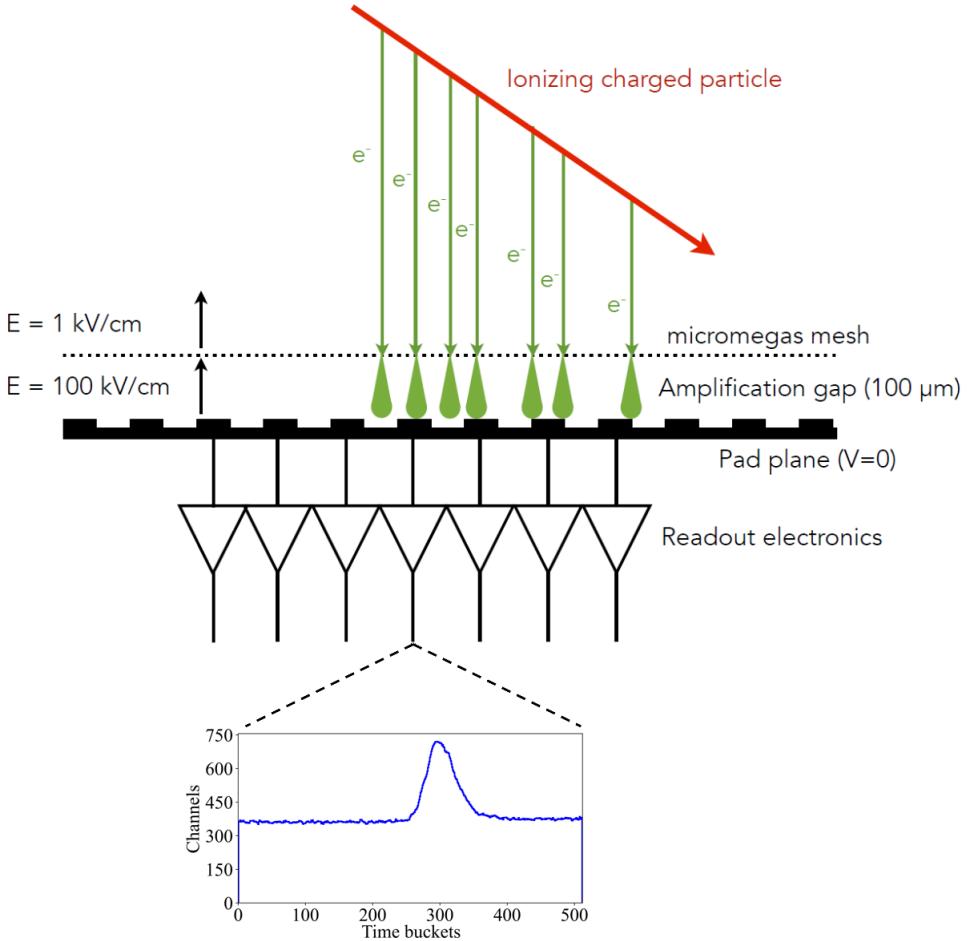


Figura 2.6: Plano do *Micromegas* com a esquematização das *thick gems*. Os elétrons quando passam para um campo elétrico mais intenso ionizam o gás, produzindo ainda mais elétrons (evento chamado de avalanche de elétrons). Cada canal da eletrônica de saída produz um pulso como mostrado na parte de baixo da figura.

O sinal é discretizado no tempo levando em conta a velocidade de deriva dos elétrons, dividindo em 512 canais o tempo que o elétron leva para percorrer toda câmara do TPC[13, 2]. A velocidade de deriva do elétron no gás ${}^4\text{He}$ é da ordem de $5 \text{ mm}/\mu\text{s}$ [2], onde o elétron percorre os 50 cm da câmara em cerca de $100 \mu\text{s}$. Dividindo esse tempo pelos 512 canais chegamos que cada canal (*time bucket*) possui 192 ns de largura. Cada centroide detectado é um ponto de interação de uma partícula carrega com o gás. A carga acumulada Q dessa interação é a área do pulso associado ao centroide. Cada centroide então representa um ponto no espaço (x, y, t, Q). Um exemplo de evento reconstruído está na figura 2.7.

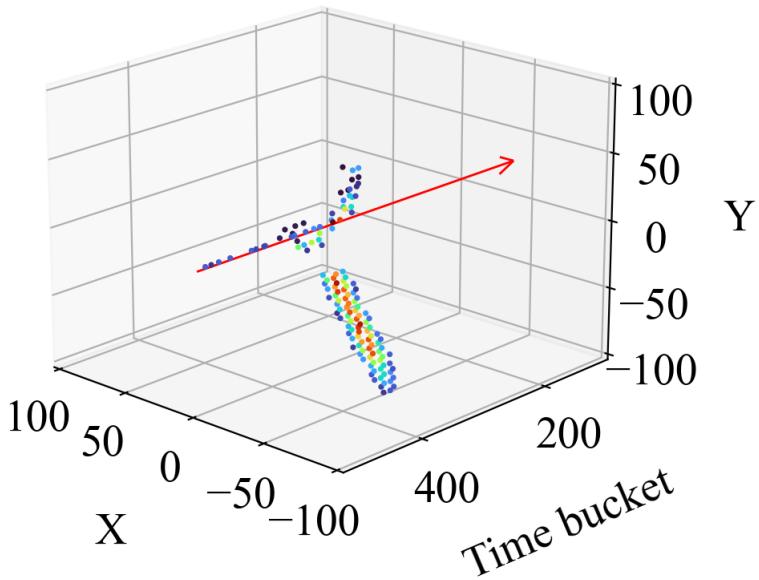


Figura 2.7: Evento reconstruído a partir da análise dos pulsos gerados pelo *Micromegas*. A cor representa a carga integrada de cada ponto de interação com o gás.

Para reconstituir eventos como o da figura 2.7, são analisados cerca de 300 pulsos. Existem canais auxiliares que servem para evitar armazenar canais sem detecção. Caso haja detecção além do centro do *Micromegas* então os sinais gerados pelo evento são armazenados[13, 12]. O número de eventos reconstruídos é da ordem de milhões, portanto a quantidade de sinais que precisam ser analisados é muito grande, o que gera a necessidade de desenvolvimento de algoritmos extremamente eficientes em tempo para a análise. Para a análise completa do experimento precisamos seguir as seguintes etapas:

- Análise dos pulsos de cada interação das partículas com o gás. Isso envolve remover o fundo, localizar os picos e obter os tempos e carga integrada de cada caso;
- Reconstruir eventos em 3D (nuvens de pontos) a partir da análise de sinais. As nuvens de pontos precisam ser analisadas com algoritmos de reconhecimento de padrões que permitem ajustar as trajetórias das partículas em 3D;
- Reconstruir a cinemática das partículas com as trajetórias e energia depositada no gás. Isto permite obter as distribuições angulares.

A análise dos pulsos, reconstituição de eventos e resultados serão mostradas nos próximos capítulos.

Capítulo 3

Desenvolvimento de ferramentas de *machine learning* para análise de dados

Um dos objetivos desse trabalho está em desenvolver ferramentas baseadas em *machine learning* para a análise do grande volume de dados gerada pelo alvo ativo. Nesse capítulo será explicada a metodologia usada para a implementação dessas ferramentas, e algumas aplicações na física nuclear.

Machine learning é a área de estudo que desenvolve algoritmos para que eles possam aprender com os dados, sem serem explicitamente programados para isso[16]. Supõe-se que uma rede neural imite um sistema biológico, em que os neurônios interajam enviando sinais na forma de funções matemáticas entre as camadas. Isso inspirou um modelo matemático simples para um neurônio artificial:

$$y = f \left(\sum_{i=1}^n \omega_i x_i + b_i \right) = f(z), \quad (3.1)$$

onde y é a saída do neurônio, que corresponde à função de ativação f que depende da soma ponderada, onde o peso é ω_i , das entradas x_i dos outros n neurônios. O termo b_i corresponde ao parâmetro *bias*. A ideia é fazer um neurônio receber a informação de todos os outros neurônios da camada anterior, fazendo uma média ponderada (onde o peso que será estimado pelo algoritmo de *machine learning*) e somando com um termo independente (*bias*, que também é estimado). Os parâmetros ω_i e b_i serão estimados através de um determinado procedimento, chamado de minimização (o treino da rede neural).

3.1 Tipos de redes neurais

Uma rede neural artificial, *Artificial Neural Network* (ANN), é um modelo computacional que consiste de camadas de neurônios. Muitas ANNs foram desenvolvidas[16, 17], mas grande parte consiste em uma camada de entrada (*input layer*), uma camada de saída (*output layer*) e eventuais camadas entre essas duas, chamadas de camadas ocultas (*hidden layers*). Os tipos mais comuns são:

Feed-Forward Neural Networks

A *Feed-forward neural networks* (FFNN) é a primeira e mais simples rede neural desenvolvida[18, 19]. Nessa rede a informação se move apenas para frente através de camadas (da camada de entrada até a camada de saída). A figura 3.1 mostra uma representação de rede, onde os neurônios são representados por círculos, enquanto que as linhas mostram as conexões entre os neurônios. Cada neurônio recebe informação de todos os neurônios da camada anterior, portanto a rede é chamada de totalmente conectada, *fully-connected* (FC), FFNN.

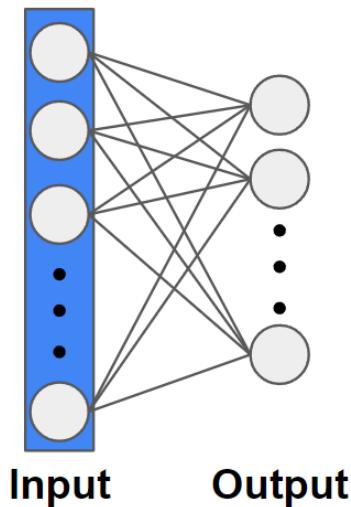


Figura 3.1: Exemplo de FFNN. A camada de entrada na esquerda propaga a informação para a direita (camada de saída). Todos os neurônios entre camadas estão conectados entre si.

Convolutional Neural Network

Uma variante da FFNN é a chamada de rede neural convolucional, *convolutional neural network* (CNN). Do ponto de vista matemático sobre convoluções, a convolução descrita como $(f * g)(t)$ de uma função $f(t)$ e outra $g(t)$ é definida como:

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau. \quad (3.2)$$

Para o caso discreto, com g sendo uma função resposta finita de tamanho $2M$, temos

$$(f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m]. \quad (3.3)$$

Convoluçãoções são invariantes sobre rotação e translação, portanto são muito utilizadas para processamento de sinais e imagens[20]. Para a convolução discreta se escolhe um filtro que irá atuar no vetor desejado. Para ilustrar o que significa isso, no caso discreto e unidimensional, a figura 3.2 mostra o processo de convolução de um vetor de tamanho 9 e um filtro de tamanho 3.

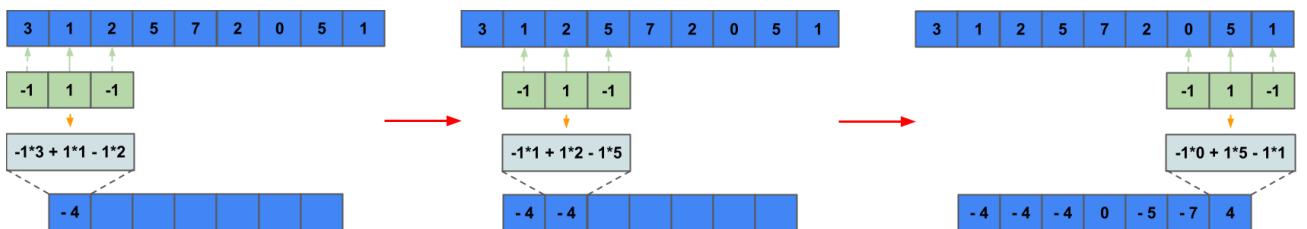


Figura 3.2: Processo de convolução entre sinal azul em cima e o filtro em verde, resultando no sinal azul embaixo. A multiplicação é feita ponto a ponto e está indicada na caixa azul-clara.

Percebe-se que o sinal resultante tem dimensão menor que o sinal original. O filtro (também chamado de *kernel*) atua em um ponto que possua vizinhos o suficiente para o restante do filtro poder fazer a multiplicação ponto a ponto. Esse tipo de convolução tem o chamado emparelhamento válido (*valid padding*). O tamanho n_2 resultante do vetor de saída é

$$n_2 = n_1 - m + 1, \quad (3.4)$$

onde n_1 é o tamanho do vetor de entrada e m o tamanho do filtro (*kernel size*). Para que o vetor de saída tenha o mesmo tamanho do vetor de entrada, são acrescentados zeros em torno da entrada, de forma que a saída tenha o mesmo tamanho da entrada. Esse é o chamado emparelhamento igual (*same padding*). A figura 3.3 ilustra esse processo.

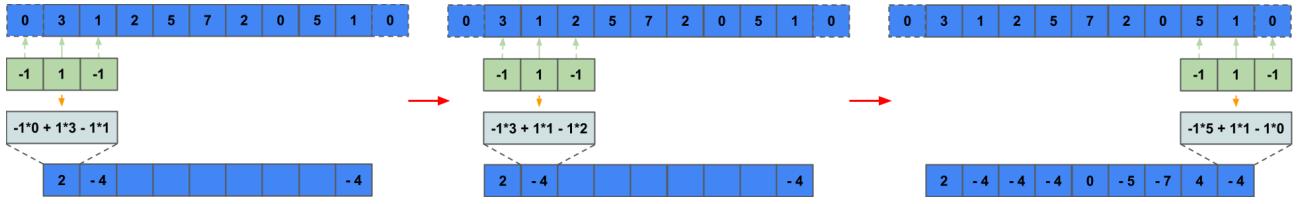


Figura 3.3: Processo de convolução entre o sinal azul em cima e o filtro em verde, resultando no sinal azul embaixo. Agora são acrescentados zeros no inicio e no final do vetor para que o vetor saída tenha o mesmo tamanho do vetor de entrada (nesse caso 9).

Como estamos no contexto de *machine learning* (inteligência artificial), *a priori* não sabemos quais os valores dos filtros que devem ser aplicados, apenas seus tamanhos e como agem. A ideia é estimar os valores do filtro (através do treino da rede neural) que deve ser aplicado para se obter o resultado desejado.

Cada filtro aplicado gera um mapa característico (*feature map*), que é o resultado da atuação do filtro em um vetor. Usualmente, em uma CNN se escolhe o tamanho do filtro, *padding* (*valid* ou *same*) e quantos filtros serão aplicados (para saber quantos *feature maps* serão gerados). Como temos vários mapas gerados por cada filtro, isso acarreta em um aumento de dimensionalidade. Para filtrar/selecionar os mapas é usado um critério, como por exemplo selecionar valores máximos dos mapas gerados dada uma janela de atuação (quantos mapas serão comparados para selecionar o máximo valor). O *Max-Pooling* faz isso, selecionando valores máximos para uma determinada quantidade de mapas sendo comparados (*pool size*).

Existem outros tipos de redes neurais que não serão discutidas aqui, porém podem ser encontradas nas referências [21, 22].

3.2 Construção de uma rede neural

Para a construção de uma rede neural (nesse caso em específico de uma rede neural supervisionada, que será discutida mais para frente), precisamos primeiro entender sobre os dados que estamos trabalhando. Grande parte das redes neurais possuem um *input* que deve ter dimensão fixa. O mesmo vale para o *output*.

Para cada camada da arquitetura devemos escolher sua função de ativação. Tanto FNNNs quanto CNNs podem possuir funções de ativação (função f da equação 3.1). Dentre muitas funções de ativação podemos citar a *Rectified Linear Units* (ReLU)[23], sigmoide[24], linear e tangente hiperbólica[25]. A figura 3.4 mostra os gráficos dessas funções de ativação, onde no eixo x é o argumento e no eixo y o resultado da função.

Funções de ativação

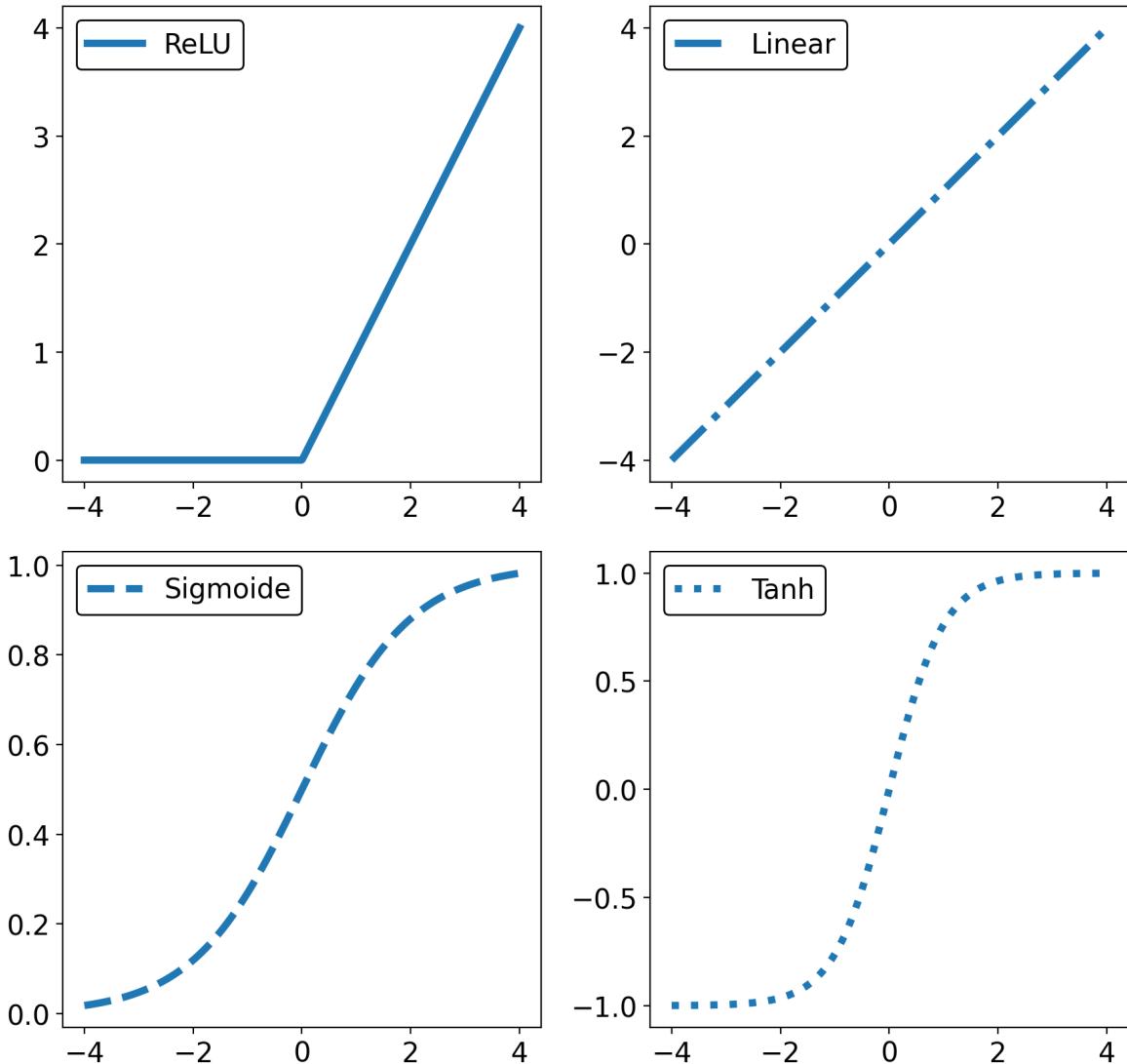


Figura 3.4: Funções de ativação e seus respectivos gráficos.

O próximo passo é definir a função custo (também chamada de *loss*) e o otimizador. A função custo tem o papel de retornar valores altos para previsões erradas e valores baixos para previsões corretas. Por exemplo, se queremos treinar uma rede neural para classificação binária (que prevê duas saídas possíveis), devemos usar a função custo chamada de *binary cross-entropy* dada por[26]

$$C(p(y_i)) = -\frac{1}{N} \sum_{i=1}^N y_i \log(p(y_i)) + (1 - y_i) \log(1 - p(y_i)), \quad (3.5)$$

onde y_i é o rótulo (*label*), $p(y_i)$ é a probabilidade do ponto y_i ser 1 e N é o número de pontos. O objetivo da rede neural é achar o mínimo da função $C(p(y_i))$, o que implica

diretamente na melhor solução para o conjunto de dados. Isso é feito pelo método de retropropagação do erro (*backpropagation*[27]) por um otimizador. Outros exemplos de *loss* são o erro quadrático médio ou *categorical cross-entropy*[28].

O otimizador tem o objetivo de otimizar os parâmetros presentes na rede neural, buscando o mínimo global da função custo, o que nem sempre acontece, pois a minimização pode parar em um mínimo local da função. Existem diversos otimizadores, como por exemplo o *Stochastic Gradient Descent* (SGD), ADAM, ADAMAX[29], entre outros[30]. Para o SGD, temos que a atualização de parâmetros é dada por

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} C(\theta), \quad (3.6)$$

onde θ_j é o parâmetro a ser atualizado, α é a *learning rate* e $C(\theta)$ é a *loss* que depende dos parâmetros θ .

Para enfim treinar a rede neural, se escolhe o *batch size*, que é o tamanho de amostras que será usada para o treino, por iteração em cada rodada de treino (*epoch*). Por exemplo, se usamos 1000 dados para o treino, e o *batch size* é 500, cada *epoch* terá duas iterações. No geral, se usam *batch sizes* pequenos, pois o consumo de memória é mais eficiente.

Para avaliação do modelo se usam dados de validação, que servem para verificar o comportamento da rede neural que está sendo treinada. Esse conjunto de dados usados não é usado para o treino, são usados apenas para verificar possíveis problemas como o *overfit*. O *overfit* ocorre quando a rede neural começa a se adequar perfeitamente aos dados de treino, perdendo a capacidade de previsão em dados que não estão sendo usados no treino pela rede neural.

Além dos dados de validação, podemos escolher métricas que auxiliam a visualização do comportamento da rede neural durante o treino e nos retornam informações importantes sobre sua qualidade. Exemplos importantes de métricas são: acurácia binária, erro médio absoluto e acurácia categórica[31]. Por exemplo, caso seja necessário verificar se uma rede neural está fazendo previsões certas em um problema cuja classificação é binária, então a métrica deve ser a acurácia binária. Tudo depende do objetivo da rede neural.

3.3 Sistemas de *machine learning*

Podemos dividir os sistemas de *machine learning* em quatro tipos:

Aprendizado Supervisionado

Aprendizado supervisionado é quando fornecemos para a rede neural um conjunto de dados para o treino com a solução desejada (chamados de *labels*). Um uso típico é para problemas de classificação[16]. Por exemplo, classificação de imagens (identificação de figuras), previsão de valores numéricos etc. Exemplos de algoritmos supervisionados são:

- *k-Nearest Neighbors*[32]
- Regressão linear
- *Support Vector Machines* (SVMs)
- *Decision Trees and Random Forests*
- Redes neurais

Aprendizado não supervisionado

Aprendizado não supervisionado é quando fornecemos o conjunto de dados para o treino, porém sem solução. A ideia é aprender sem supervisão. Um problema comum, por exemplo, é quando queremos identificar *clusters* em um conjunto de dados (*clustering*)[33, 34].

Aprendizado semi supervisionado

Aprendizado supervisionado é quando apenas parte do conjunto de dados para o treino possui *labels*. Isso é comum quando se obtém conjuntos de dados diferentes e apenas parte deles foi classificado[35].

Aprendizado por reforço

Aprendizado por reforço é quando um sistema, chamado de *agente* nesse contexto, aprende através do ambiente, realizando ações que maximizam sua recompensa. Por exemplo, caso o sistema realize uma ação incorreta, ele recebe uma penalidade, fazendo com que procure outra maneira de realizar a ação, dessa vez de maneira correta, para poder ganhar uma recompensa[36]. Esse tipo de sistema é muito usado, por exemplo, em automatização robótica, como carros que pilotam sozinhos, robôs que aprendem a andar etc[37].

3.4 Aplicações de *machine learning* na física nuclear

Em física nuclear, o uso de técnicas de *machine learning* tem se mostrado cada vez mais importante. Na continuação serão citados alguns exemplos.

3.4.1 Análise de espectros para identificação de partículas (*particle identification, PID*)

O uso de técnicas de aprendizado não supervisionado pode ser usado para a identificar de partículas em espectros $\Delta E - E$. Neste tipo de espectros, as partículas são detectadas e separadas por regiões que facilmente podem ser identificadas visualmente[38]. No entanto, a identificação visual desses espectros com os sistemas de detecção com milhares de canais é inviável de forma manual. Assim, ferramentas de *machine learning* (ou inteligencia artificial) são de grande relevância para esse tipo de analise.

Os resultados apresentados nessa seção foram obtidos na etapa inicial desta dissertação. Para o presente estudo, o espectro biparamétrico ($\Delta E-E$) mostrado na Figura 3.5a será analisado. Os dados correspondem a reação de espalhamento e transferência no sistema ${}^6\text{Li} + {}^{12}\text{C}$.

O objetivo é identificar conjuntos (ou *clusters*) diferentes a partir do espectro. É claro para o olho humano que existem conjuntos diferentes, e eles podem ser identificados usando algoritmos de aprendizado não supervisionado, como por exemplo o *density-based spatial clustering of applications with noise* (DBSCAN)[39]. O DBSCAN consegue identificar *clusters* apenas com as informações características de densidade existente pelos conjuntos que existem nos dados. O resultado da aplicação do algoritmo está na figura 3.5b, onde é mostrado que o espectro agora está separado por diferentes conjuntos.

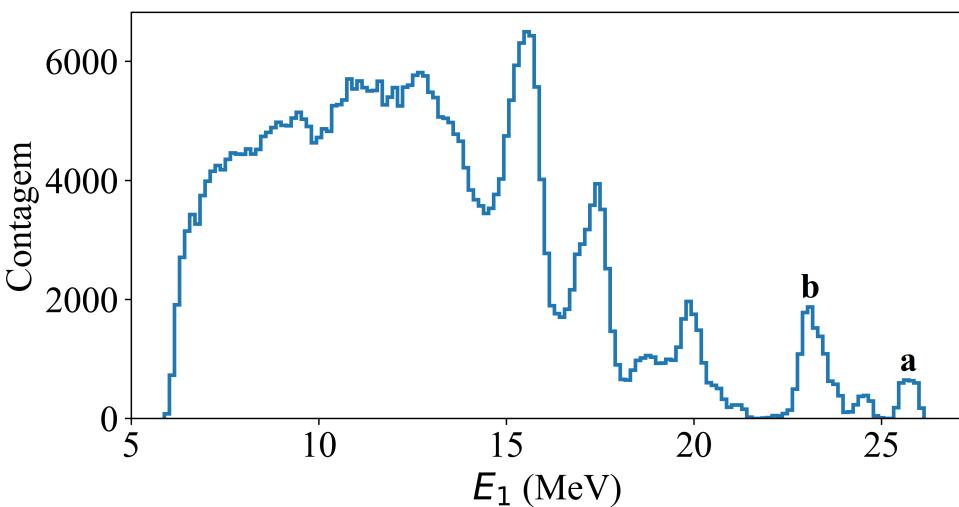
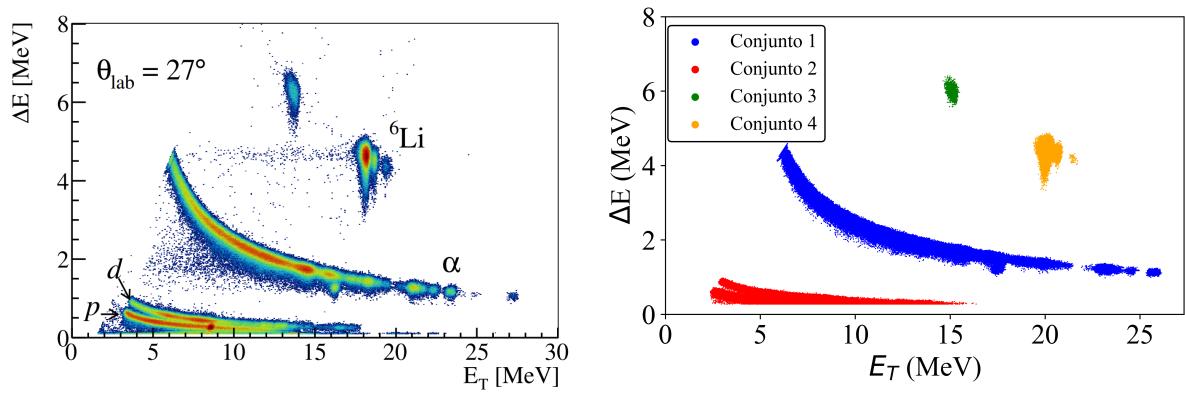


Figura 3.6: Histograma de identificação de partículas, a partir do conjunto 1 identificado na figura 3.5b. Em **a** temos o primeiro pico que corresponde ao estado fundamental do ^{14}N e em **b** temos o primeiro estado excitado do ^{14}N .



(a) Espectro biparamétrico $\Delta E \times E$ (energia no referencial do centro de massa).

(b) Espectro biparamétrico $\Delta E \times E$ (energia no referencial do centro de massa) com os conjuntos identificados com o uso do DBSCAN.

Figura 3.5

Usando o conjunto 1, por exemplo, é possível criar o histograma de identificação de partículas dado pela figura 3.6.

3.4.2 Estimativa de raios e massas nucleares

Frequentemente é necessário calcular com alta precisão observáveis que ainda não foram medidos, para contribuir com dados já existentes. É possível estimar propriedades de núcleos usando modelos e dados experimentais já existentes em conjunto com técnicas de *machine learning*.

É possível usar redes neurais que façam previsão de massa nucleares com informações de massa já disponíveis. Isso foi feito usando o algoritmo *Light Gradient Boosting Machine* (LightGBM)[40]. O algoritmo é uma rede neural cujo aprendizado é supervisionado e minimiza o erro entre a energia de ligação teórica e a energia de ligação experimental dos núcleos (essa diferença é chamada de resíduo), usando 10 quantidades físicas como dados de entrada (*input*)[41]. O desvio quadrático médio para a massa dos núcleos é de 0.234 ± 0.022 MeV, valor melhor que muitos modelos físicos de massa nuclear. A figura 3.7 mostra os resíduos calculados entre a energia de ligação calculada pelo rede neural e a energia de ligação determinada experimentalmente.

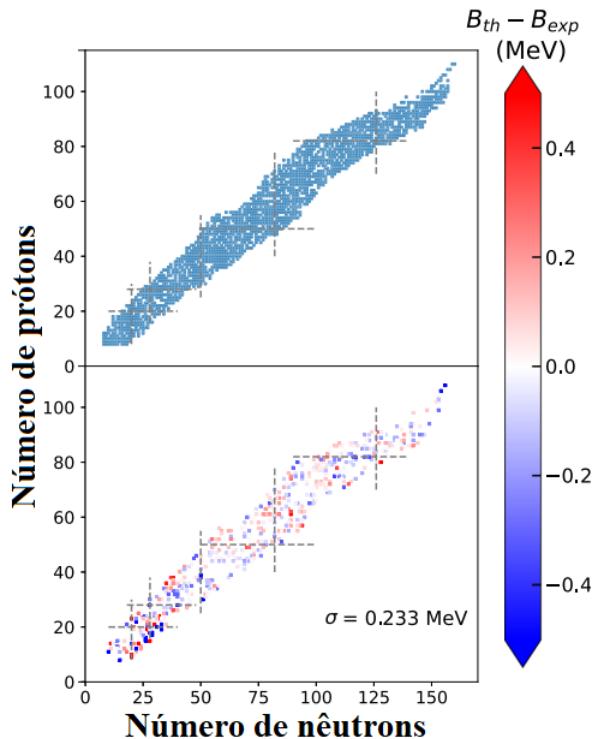


Figura 3.7: Painel acima mostra a localização dos núcleos usados para o treino da rede neural. No painel de baixo temos o erro entre a previsão e o valor experimental da energia de ligação do núcleo para os núcleos usados como dados de validação. σ é o erro quadrático médio da rede neural.

Outro exemplo de uso é para a estimativa de raio de carga nuclear. Usando uma rede neural (chamada de *Bayesian neural network extended liquid drop* - BNN-ELD) FC com duas variáveis de entrada, o número de prótons Z e o número de massa A , com a condição de que $Z \geq 20$ e $A \geq 40$. A rede é supervisionada e usa 722 núcleos para dados de treino e 98 núcleos como validação. Resultados para o desvio quadrático médio do raio de carga são de cerca de 0.02 fm para os 820 núcleos utilizados. A figura 3.8 mostra a diferença entre o raio de carga calculado teoricamente e o determinado experimentalmente para

isótopos do chumbo, usando diferentes neurais e modelos teóricos. Mais detalhes podem ser encontrados na Ref. [42].

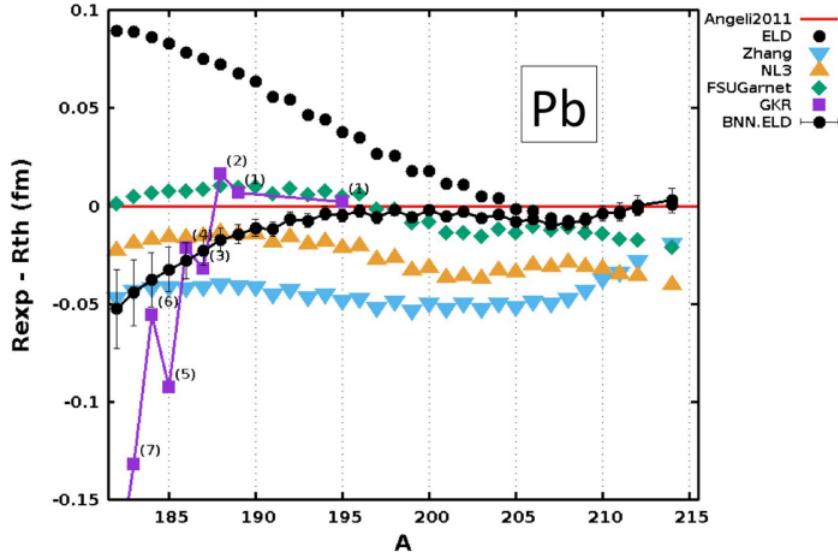


Figura 3.8: Previsões para o raio de carga para isótopos do chumbo ($Z = 82$) para diferentes modelos teóricos ou que usam redes neurais. A previsão que contém barras de erro é a que foi brevemente descrita no texto.

3.4.3 Decaimento beta e processo r

O decaimento β é fundamental para entender a origem dos elementos pesados. Prever o tempo de meia vida do decaimento β é de grande importância para simulações do processo r (captura rápida de nêutrons). Com redes neurais é possível fazer previsões que levam em conta a física do problema, como visto na Ref. [43]. O modelo de inteligência artificial leva em conta a teoria de Fermi para o decaimento beta, onde a função f que é minimizada (função custo) é dada por

$$f = \log_{10}(T_{1/2}^a/T_{1/2}^b), \quad (3.7)$$

onde $T_{1/2}^a$ é o tempo de meia vida do decaimento beta medido experimentalmente e $T_{1/2}^b$ é o tempo de meia vida do decaimento beta determinado teoricamente. As variáveis de entrada são o número de prótons Z , o número de nêutrons N , a energia total do decaimento beta e o parâmetro de paridade $\delta = 1, 0, -1$, para núcleos par-par, ímpar-par e ímpar-ímpar, respectivamente.

A figura 3.9 mostra a previsão do tempo de meia vida ($T_{1/2}$) do decaimento β (em segundos) para isótonos com número de nêutrons $N = 126$ usando redes neurais.

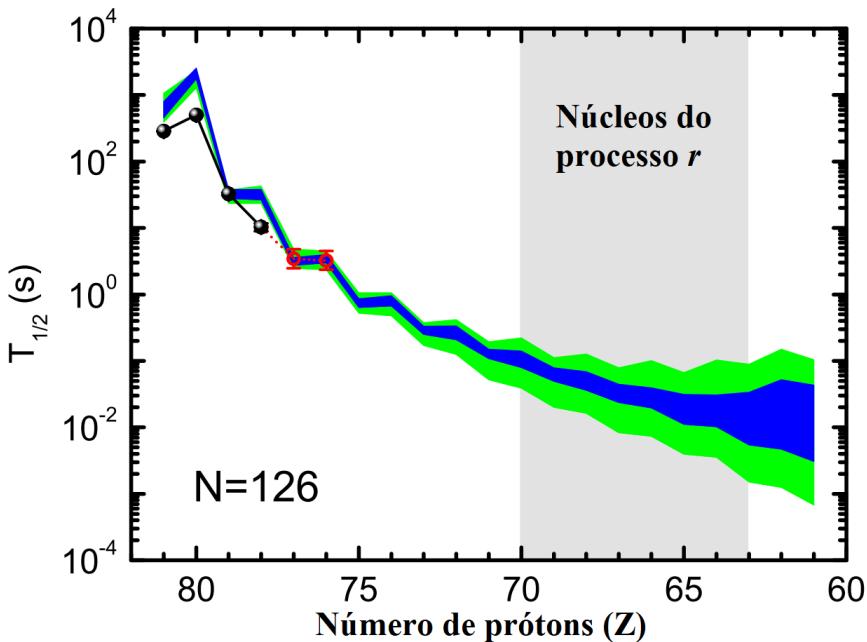


Figura 3.9: Meias-vidas de decaimento β para isótonos com $N = 126$. A região hachurada verde mostra as previsões de uma rede neural. A região hachurada em azul mostra os resultados da mesma rede neural, porém seus dados de aprendizado são estendidos para incluir três meias-vidas extras de decaimento β para cada isótopo (indicado por círculos abertos) em direção à *drip-line* de nêutrons[43].

3.4.4 Alvos ativos

Experimentos com alvos ativos geram enormes quantidades de dados. Uma semana de experimento pode gerar até 10Tb de dados[44], o que gera a necessidade do uso de algoritmos de *machine learning* para diminuir o consumo de tempo da análise desses dados.

A análise dos dados envolve a reconstrução tridimensional dos eventos e posteriormente a reconstrução da cinemática, com a identificação de partículas e de reações nucleares. O uso de técnicas de *machine learning* pode diminuir o consumo de tempo dessas etapas.

O uso de CNNs em imagens feitas a partir das projeções de eventos pode ser útil para a classificação de eventos, sem a necessidade de reconstruir a cinemática em uma etapa anterior. A figura 3.10 mostra exemplos de projeções de trajetórias de partículas dentro do alvo ativo, onde a partir da projeção no plano xy é possível identificar a partícula que a originou.

Primeiro, com o objetivo de classificar as projeções entre próton ou carbono, é possível construir uma rede neural para a classificação binária. Para isso foi usada uma arquitetura

com camadas convolucionais seguidas de *max-pooling* e por fim uma camada FC[44]. A função a ser minimizada é a dada pela equação 3.5 e a métrica para entender o comportamento da rede neural é a acurácia binária.

Caso o objetivo seja classificar entre três ou mais possibilidades (como por exemplo próton, carbono e outros), então a classificação não será binária, passará a ser categórica, pois será necessário classificar a imagem em alguma categoria. A função custo passará a ser a *categorical cross-entropy* e a métrica será a acurácia categórica.

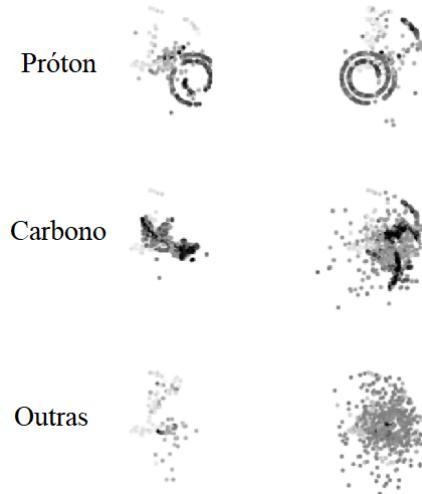


Figura 3.10: Projeções no plano xy de partículas dentro do alvo ativo, onde quanto mais escura for a cor, mais carga tem o ponto, e quanto mais clara a cor, menos carga tem o ponto. O objetivo da rede neural pode ser classificar corretamente se a imagem à direita corresponde à uma trajetória de um próton, carbono ou outra partícula, ou pode ser uma rede neural para classificação binária caso seja necessário classificar apenas entre próton ou carbono[44].

A rede criada para a classificação binária (entre próton e carbono) atingiu cerca de 90% de acurácia. Somado ao fato de que redes neurais são muito eficientes em tempo, isso mostra uma possibilidade de que seja possível utilizar redes neurais como essa para o processamento em tempo real dos dados, com pouca supervisão humana durante o processo[44].

Outro problema que pode ser resolvido com técnicas de *machine learning* é a remoção de ruído de eventos. Para isso, pode ser usado o algoritmo não supervisionado *Hough transform*[45], que é capaz de detectar padrões de interesse nos dados e evitar regiões ou pontos que não seguem o padrão desejado.

Com essa breve descrição sobre *machine learning* e exemplos do seu uso em problemas de física nuclear, podemos seguir adiante e entender seu uso dentro deste trabalho, que será feito nos próximos capítulos.

Capítulo 4

Reconstrução de nuvens de pontos

Esse capítulo irá mostrar como são recriadas as nuvens de pontos (*pointclouds*) a partir dos pulsos gerados por cada pixel do *micromegas*. Primeiro será mostrado como os sinais são analisados a partir de métodos mais tradicionais e depois usando algoritmos de *machine learning*.

4.1 Análise dos pulsos com algoritmos tradicionais

Essa seção irá descrever o processo de análise dos sinais usando algoritmo mais tradicionais. Os sinais são gerados pelo detector *micromegas* e cada um de seus canais possui eletrônica independente e os sinais recebidos são como os mostrados na figura 4.1.

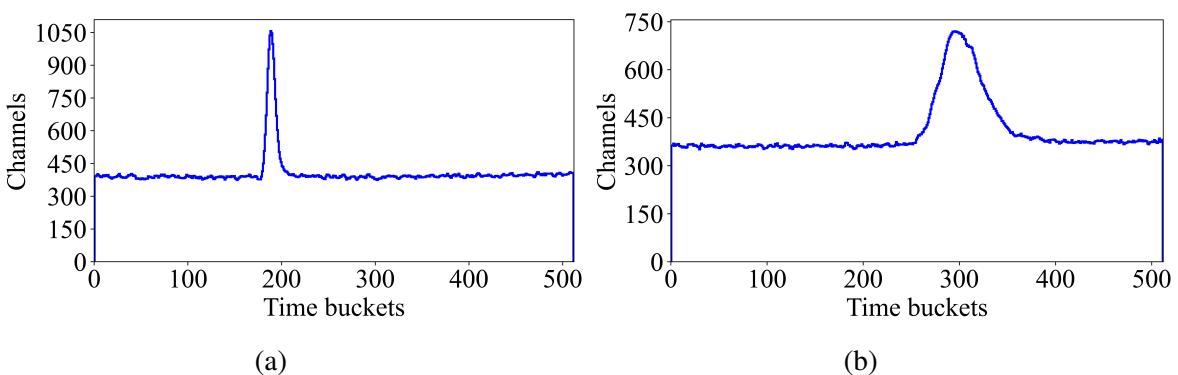


Figura 4.1: Exemplos de sinais produzidos pelos canais do detector. Em 4.1a o sinal possui apenas um pulso, enquanto em 4.1b há vários pulsos em sobreposição, formando um único pulso com largura maior que em 4.1a.

No eixo x , cada um dos 512 *time buckets* possui largura de 195 ns. No eixo y temos a carga acumulada no detector para cada *time bucket*. Em 4.1a vemos um sinal com um pedestal (fundo ou *baseline*) com altura entre 300 e 450, e um pulso estreito em

cima. Como dito na seção 2, os elétrons que surgem da ionização do gás são conduzidos perpendicularmente pelo campo elétrico até o detector. A interação da partícula com o gás é evidenciada justamente pelo pulso presente em 4.1a. Cada pixel i do detector está em uma posição (x_i, y_i) , o centroide de cada gaussiana fornece a coordenada em t (*bucket*) para então ser convertida na posição em z da partícula detectada, e a carga acumulada Q do ponto é a área do pulso (gaussiana com centroide t) sem o fundo.

Para 4.1a temos apenas um pulso, o que significa que há um feixe paralelo àquele canal do histograma, afinal, temos apenas um único ponto (x, y, z) . Já para 4.1b temos o que é chamado de mistura de gaussianas (*gaussian mixture*), que é a presença de várias gaussianas sobrepostas. Esse tipo de sinal corresponde ao feixe indo perpendicularmente ao pixel do detector. A ilustração desse processo está na figura 4.2, que mostra o processo da passagem de uma partícula carregada e como o sinal é gerado a partir disso.

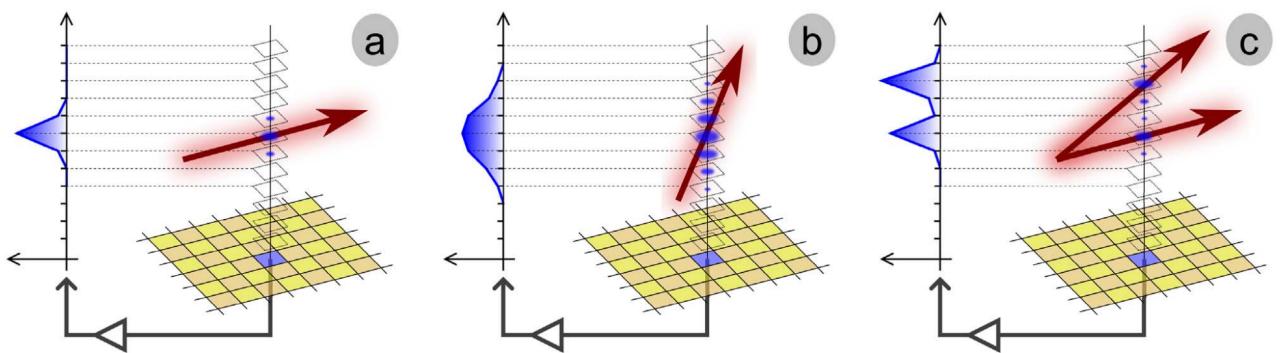


Figura 4.2: Ilustração que mostra a variação no formato da carga coletada a partir da passagem de uma partícula carregada dentro do TPC, onde o plano do detector está embaixo. No lado esquerdo de cada imagem, a distribuição do sinal coletado por um único pad (escuro) do plano de coleta é mostrado (o canal eletrônico de leitura é representado pela seta cinza em negrito). No caso de uma trajetória quase horizontal (a), o sinal é uma distribuição estreita, enquanto para uma trajetória próxima a uma direção vertical (b), a distribuição deve ser muito mais ampla (vários picos devem ser extraídos desse sinal). A última imagem ilustra o caso em mais de uma trajetória de partículas contribui para o sinal[14].

Para analisar os pulsos devemos então remover o fundo (pedestal ou *baseline*) dos sinais. O fundo não é trivial de se determinar, pois não é analítico, oscilando muito entre os canais.

4.1.1 Remoção do fundo

A primeira tentativa de estimar o sinal sem o fundo é usando transformada de Fourier e um filtro passa-baixa. Seja $f(t)$ uma função qualquer, sua transformada de Fourier é

dada por

$$\hat{f}(v) = \mathcal{F}[f(t)] = \int_{-\infty}^{\infty} f(t)e^{-2\pi i vt} dt. \quad (4.1)$$

Primeiro calculamos a transformada de Fourier $\hat{f}(v)$ do sinal, em seguida multiplicamos por $\text{sinc}(\nu/a)$, onde

$$\text{sinc } x \equiv \frac{\sin(\pi x)}{\pi x}, \quad (4.2)$$

onde a é um fator de escala. A função sinc foi escolhida para tirar vantagem do Teorema da Convolução, dado por[46]

$$\mathcal{F}^{-1}[\hat{f}(\nu)\hat{g}(\nu)] = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau, \quad (4.3)$$

onde $(f * g)(t)$ é a convolução entre $f(t)$ e $g(t)$. Multiplicar o sinal transformado por $\text{sinc}(\nu/a)$ e depois inverter inverte a transformação é o mesmo que convoluir o sinal original com a transformação inversa de $\text{sinc}(\nu/a)$, pois

$$\mathcal{F}^{-1}[\text{sinc}(\nu)] = \text{rect}(t) \equiv \begin{cases} 1, & -\frac{1}{2} < t < \frac{1}{2} \\ 0, & \text{qualquer outro } t \end{cases}, \quad (4.4)$$

é uma função que representa uma janela retangular, o que seria um exemplo de função resposta do detector. Caso saibamos essa função resposta, então é possível reconstruir completamente o sinal. Resultados desse procedimento estão na figura 4.3.

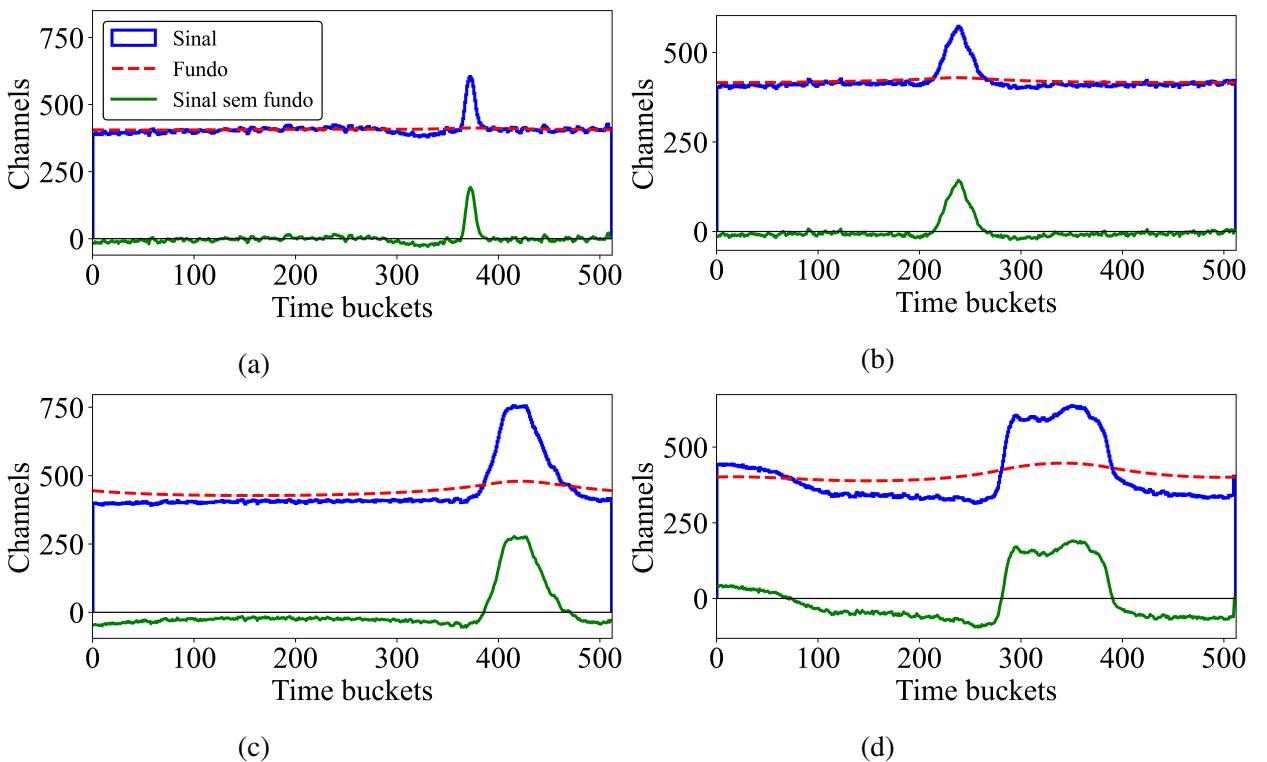


Figura 4.3: Histogramas com as respectivas *baselines* (linhas tracejadas) estimadas pelo método da convolução. O espectro resultante (sem o fundo) está em verde.

Esse não é o melhor método pois não sabemos qual é função resposta para o detector, pois o fundo é não analítico. Para um resultado mais adequado, o fundo será determinado usando o algoritmo *background removal* da biblioteca *TSpectrum* do *ROOT* [47]. A função tem a capacidade de separar o fundo dos picos presentes no espectro[48, 49, 50]. Exemplos de estimativa do fundo estão na figura 4.4.

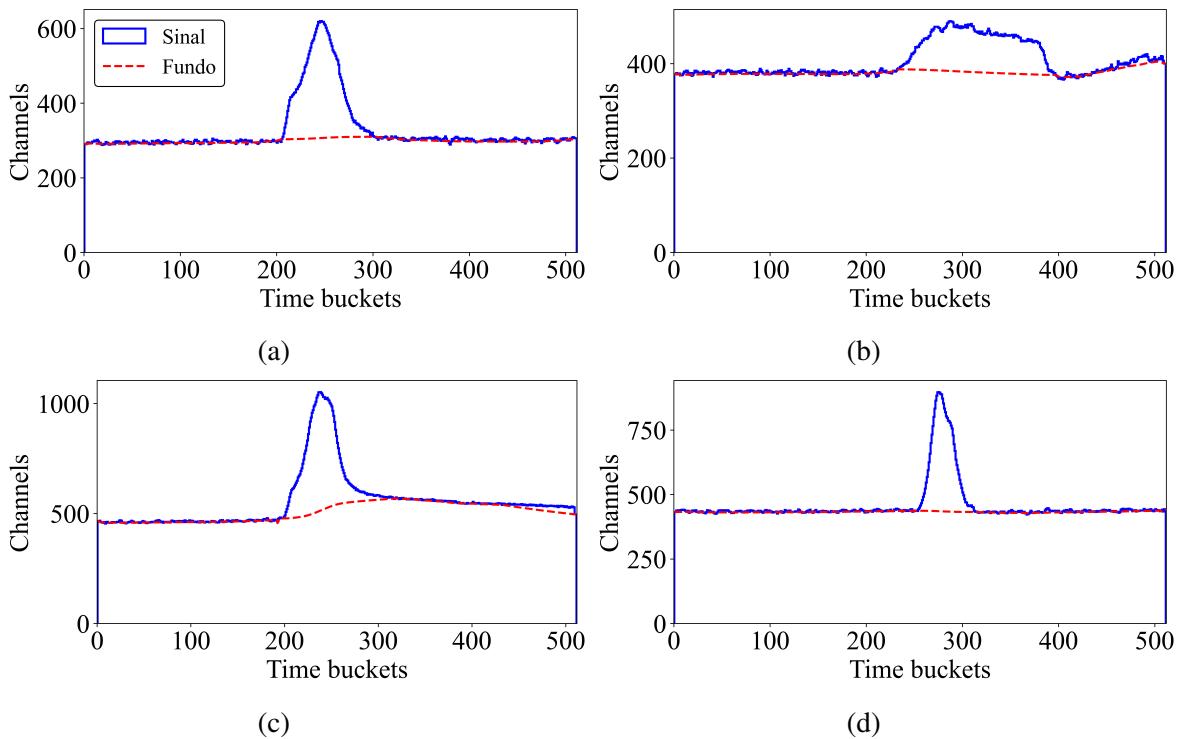


Figura 4.4: Histogramas com as respectivas *baselines* (linhas tracejadas) calculadas pelo *TSpectrum*.

Para evitar valores negativos após a remoção do fundo, o valor mínimo do sinal sem o fundo é zero. Sem o fundo podemos buscar por todos os picos e suas cargas correspondentes no sinal. Não podemos detectar diretamente todos os picos pois muitos deles estão em sobreposição. Para isso será feita a deconvolução do sinal.

4.1.2 Deconvolução do sinal

Para aumentar a resolução dos picos será usado o algoritmo *gold deconvolution* presente na biblioteca *TSpectrum* do *ROOT*[51]. O algoritmo tem como objetivo fazer a deconvolução do espectro, gerando uma função (nesse caso um sinal) resposta de acordo com o sigma esperado para os pulsos. O sinal resposta corresponde ao espectro com as gaussianas não sobrepuestas. Isso significa que devemos descobrir qual o valor de sigma dos pulsos para buscar a função resposta.

O sigma dos pulsos deve ser o mesmo de um sinal que possui apenas um pico. Ou seja, podemos determinar o sigma fazendo a análise de sinais que possuem apenas 1 pico, fazendo um ajuste pelo método dos mínimos quadrados (MMQ) de uma gaussiana. Para buscar espectros com apenas um pico foi usado o algoritmo de detecção de picos do *scipy*[52] e para o ajuste da gaussiana foi usada o pacote *lmfit* [53]. O valor de sigma

encontrado foi de 4.09 (17) *time buckets*.

O sigma escolhido foi ligeiramente maior pois, verificando empiricamente, em alguns casos o algoritmo separava o que deveria ser uma única gaussiana em duas. O valor de sigma usado na deconvolução foi de 4.30 *time buckets*. Devemos também escolher o número de iterações do algoritmo de deconvolução. O número de iterações escolhido foi de 700, menos que isso o algoritmo não estava separando totalmente picos sobrepostos. O limiar para a escolha de um ponto como um pico é ter altura maior que 20% do valor máximo do sinal. Resultados da deconvolução estão na figura 4.5.

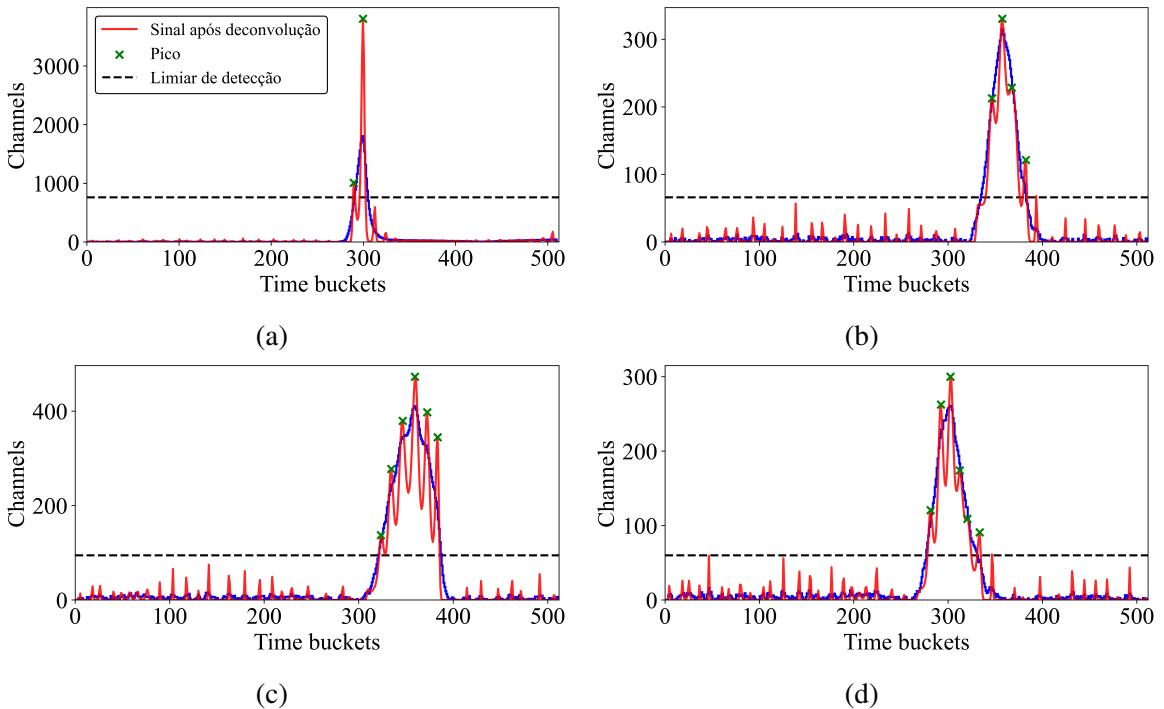


Figura 4.5: Histogramas sem as *baselines* antes (em azul) e depois da deconvolução (em vermelho). Os picos (em verde) e o limiar (linha tracejada preta) de detecção também estão indicados.

O algoritmo de deconvolução também retorna a posição dos centroides encontrados. A execução de 200.000 sinais, desde a estimativa e remoção do fundo, até a detecção dos centroides, demora cerca de 23.25 minutos, usando o processador Ryzen 5 3600X.

Para determinar a carga de cada ponto temos que calcular a área do centroide do pico detectado. A área do sinal antes e depois da deconvolução é a mesma, mesmo para a região dos pulsos, portanto podemos olhar diretamente para o sinal após a deconvolução. Para achar a área podemos calcular o sigma dos pulsos após a deconvolução, para determinar a área como uma simples integral gaussiana. O sigma dos pulsos após a deconvolução é $\sigma_{dd} = 1.1543$ (44) *time buckets*. Com isso podemos calcular a carga acumulada para

cada ponto descoberto do evento. A carga acumulada Q para cada ponto i é dada por:

$$Q = \int_{-\infty}^{\infty} Ae^{-(t' - t_i)^2 / 2\sigma_{dd}^2} dt' = A |\sigma_{dd}| \sqrt{2\pi}, \quad (4.5)$$

onde A é a amplitude do ponto com centroide t_i e desvio padrão após a deconvolução σ_{dd} . Com esse procedimento podemos então reconstituir os eventos (nuvens de pontos), obtendo, para cada evento, todas as coordenadas x, y, t e Q de cada ponto. A figura 4.6 mostra exemplos de eventos reconstruídos.

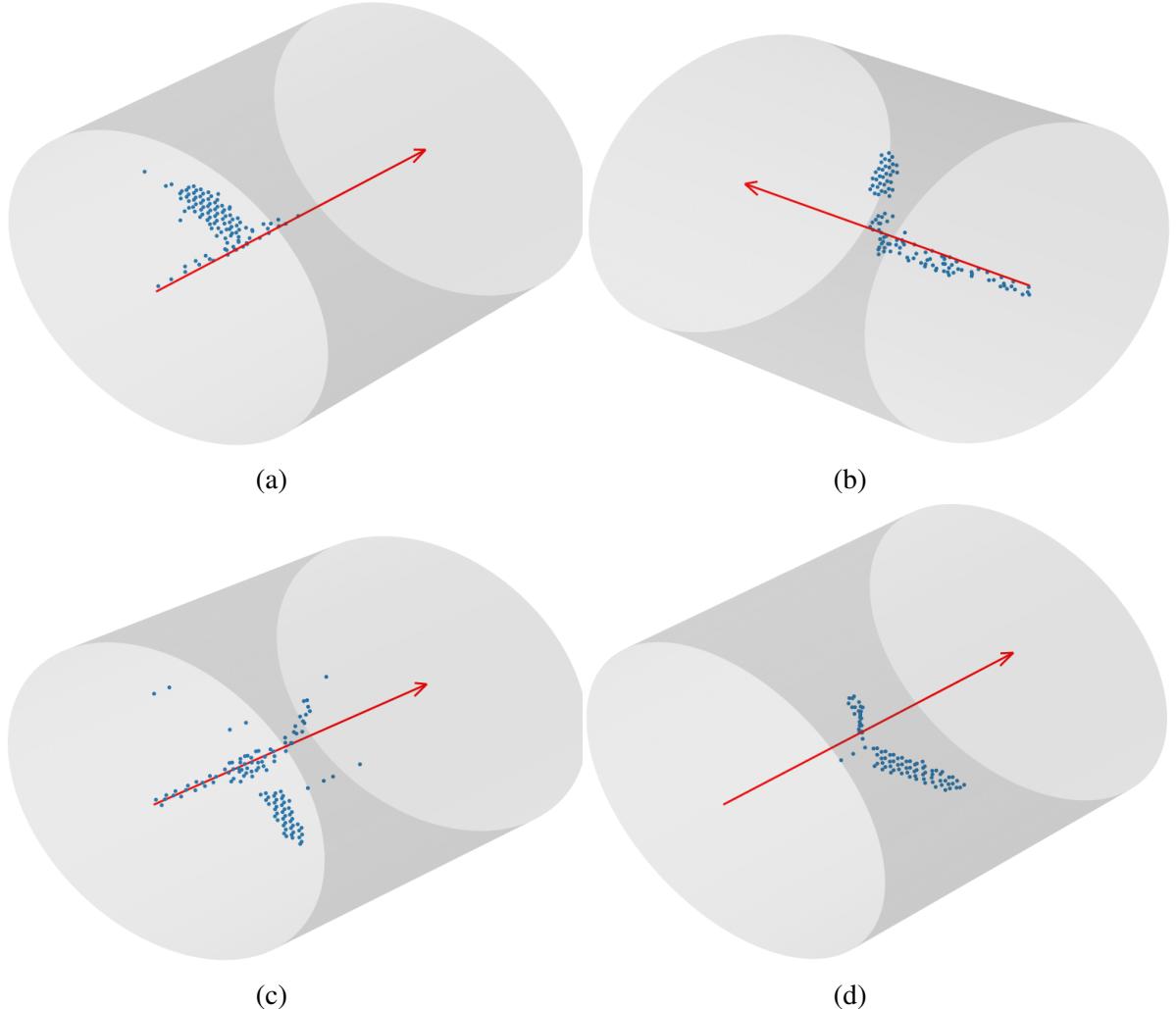


Figura 4.6: Exemplos de eventos reconstruídos através da análise dos sinais. A seta vermelha indica o sentido do feixe.

4.2 Análise dos pulsos com *machine learning*

Com *machine learning* temos a possibilidade de criar algoritmos extremamente complexos sem definir operações explícitas. Podemos nos basear na metodologia da seção

anterior que é dividir o problema em três etapas: remover o fundo, fazer a deconvolução e por fim detectar os picos. A subseções seguintes irão descrever uma rede neural para cada etapa.

4.2.1 Rede neural para o fundo

O objetivo é criar uma rede neural que reproduza o comportamento do algoritmo *background removal* que estima o fundo do sinal, tentando reproduzir resultados muito similares. A rede neural é supervisionada, onde os dados de entrada são os sinais brutos e as saídas devem ser os fundos de cada sinal. A arquitetura está na figura 4.7.

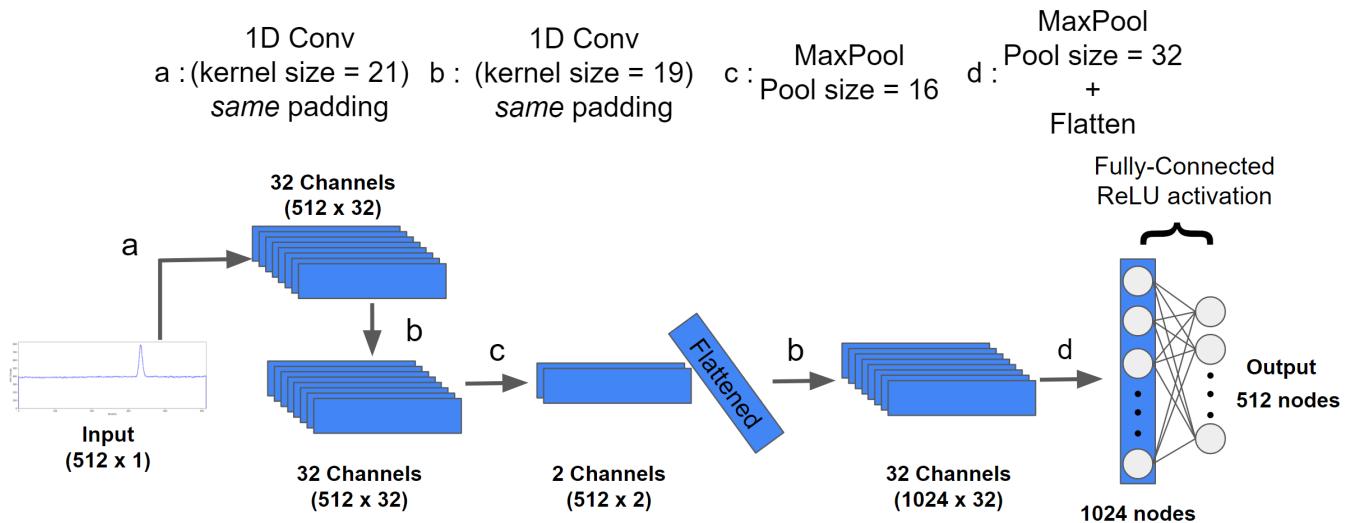


Figura 4.7: Arquitetura da rede neural que faz a inferência do fundo. O vetor de entrada deve ter dimensionalidade 512×1 . Todas as partes com convolução não possuem o parâmetro *bias*.

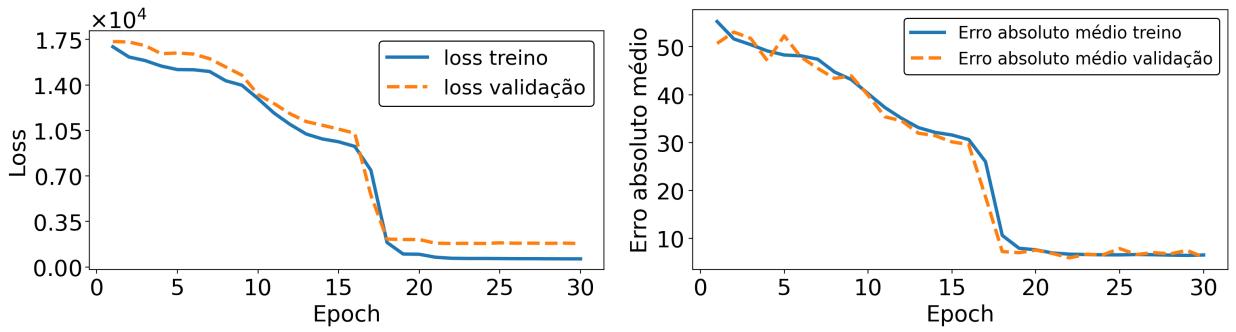
A entrada da rede é o sinal com dimensionalidade 512×1 . Há duas convoluções seguidas (passagens *a* e *b*) com *padding same*, seguida de um *Max pooling*. Os dois canais restantes sofrem uma planificação (ou *flat*), diminuindo sua dimensionalidade, para então passar mais uma convolução com *padding same* e filtros de tamanho 19 seguido de *Max pooling* e *Fully Connected* com ativação *ReLU*. Toda a rede foi construída usando o TensorFlow 2 e possui um total de 545.536 parâmetros, todos treináveis. O tamanho dos filtros das convoluções devem levar em conta a largura do pulso, devendo ser no mínimo maior que a largura, a fim de cada *kernel* visualizar um pulso completo na convolução.

A camada final com ativação *ReLU* garante o valor mínimo de saída em 0 e, principalmente, pelo fato de não causar problemas à minimização do gradiente [54]. Foram testadas diversas combinações e a mostrada na figura 4.7 é a que obteve os melhores resultados.

Para o treino foram usados 160.000 sinais para treino e 40.000 para validação. O *loss* foi escolhido como sendo o erro quadrático médio (equação 4.6, o otimizador foi o *ADAMAX* [29], com *learning rate* de 0.0005, e métrica para avaliação foi o erro médio absoluto, dado por

$$E = \frac{1}{N} \sum_{i=1}^N |x_i - \hat{x}_i|, \quad (4.6)$$

onde E é o erro absoluto médio, N é o número de pontos e x_i o ponto da saída da rede para ser comparado com o ponto original \hat{x}_i . Foram 30 *epochs* e o *batch-size* foi 8. O treino foi realizado no Google Colaboratory [55] usando a GPU (graphics processing unit) NVIDIA Tesla P100 e durou cerca de 34 minutos. Os resultados do treino estão na figura 4.8.



(a) *Loss* dos dados de treino (linha contínua) e dos dados de validação (linha tracejada) em função da *epoch* no treino da rede dada por 4.7.

(b) Erro absoluto médio dos dados de treino (linha contínua) e dos dados de validação (linha tracejada) em função da *epoch* no treino da rede dada por 4.7.

Figura 4.8: Resultados do treino da rede neural dada por 4.7. A rede atingiu seu melhor resultado a partir da *epoch* 20 aproximadamente, quando começa um platô no *loss*.

A arquitetura da figura 4.7 (assim como as próximas desse capítulo) foi determinada de forma empírica. Uma arquitetura com menos passagens e/ou menos parâmetros fornece resultados piores com relação à arquitetura apresentada. No caso de mais parâmetros e /ou passagens (consequentemente com aumento no tempo de execução), a rede neural não demonstrou melhora substancial.

Exemplos de resultados da rede podem ser vistos na figura 4.9. A previsão do fundo possui um erro absoluto nos dados de treino de 6.5315 Channels e nos dados de validação de 6.0783 Channels. Podemos subtrair o fundo do sinal original (colocando o valor mínimo da subtração em 0). Comparando o erro médio absoluto de 200.000 sinais sem o respectivo fundo (resultante do algoritmo do TSpectrum) com o resultado da rede neural obtemos 4.5 Channels.

Rede neurais convolucionais têm a vantagem de usarem poucas variáveis e serem facilmente paralelizadas em sua execução. Uma vantagem de redes neurais é o seu tempo de execução. Empiricamente a rede neural pode processar 200.000 sinais em apenas 8s (ou 25.000 sinais por segundo), sendo extremamente eficiente em tempo.

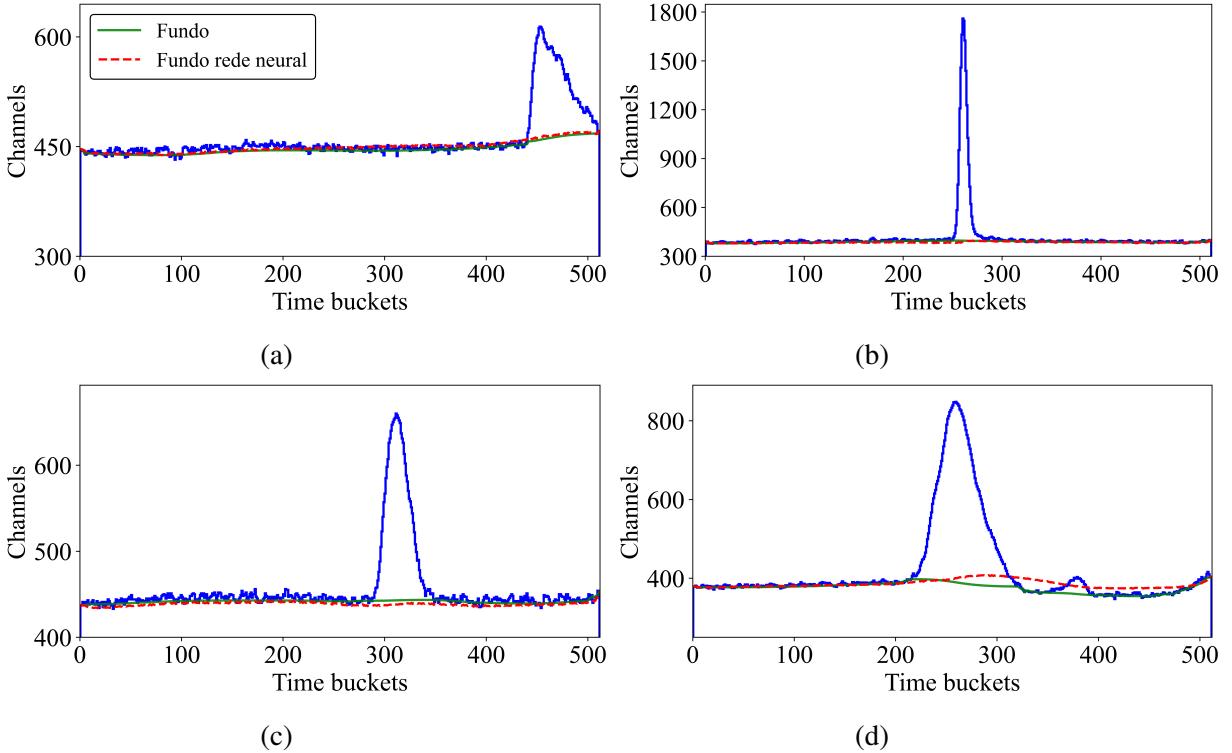


Figura 4.9: Exemplos da rede neural dada por 4.7 em comparação com a saída do *TSpec-trum*.

Nos exemplos mostrados nas figuras 4.9a, 4.9b e 4.9a os fundos dos sinais possuem grande flutuação e a rede neural se mostrou eficaz na previsão. No exemplo 4.9d a *baseline* do sinal é extremamente complexo de se determinar pois o sinal, do canal 300 a 500 aproximadamente, varia em cerca de 50 unidades em y . Apesar da rede neural determinar o fundo acima do fundo original, ao subtrair o espectro do fundo e colocar o valor mínimo em 0, o pulso presente entre os canais 200 e 300 é praticamente inalterado.

Com os resultados da rede podemos então seguir a diante para criar a rede neural que faz a deconvolução do espectro sem a *baseline*.

4.2.2 Rede neural para a deconvolução

Podemos usar a mesma abordagem da rede neural anterior, fazer uma sequencia de convoluções e por fim uma *fully connected* com ativação *ReLU*, pois precisamos ter o

valor mínimo do espectro em 0. Os filtros das convoluções precisam ter tamanho mínimo de duas vezes o sigma das gaussianas para atuarem sobre cada pulso do espectro. A entrada da rede é o sinal com o fundo subtraído e com mínimo em 0. A saída é a deconvolução dada pelo algoritmo *gold deconvolution* na biblioteca *TSpectrum* do ROOT. A figura 4.10 mostra a arquitetura da rede de deconvolução.

A rede é a sequência de duas convoluções com 32 filtros, *valid padding* e *kernels* de tamanho 19 e 17, respectivamente, seguida de *Max pooling* com *pool size* igual à 16. No final há o *flat* na camada para seguir com uma *fully connected* com ativação *ReLU*. O *valid padding* se mostrou mais eficiente para a convergência da rede. Toda a rede foi construída usando o TensorFlow 2, possuindo 508.000 parâmetros treináveis.

Assim como na rede anterior, foram usados 160.000 sinais para treino e 40.000 para validação. O *loss* foi escolhido como sendo o erro quadrático médio, o otimizador foi o *ADAM*, com *learning rate* de 0.0005 porém com o parâmetro *clipnorm* igual a 0.45. A métrica para avaliação foi o erro médio absoluto. Foram 75 *epochs* e o *batch-size* foi 8. Os resultados do treino estão na figura 4.11.

Alterar a norma do gradiente (usar o parâmetro *clipnorm* = 0.45) significa que, caso a norma do vetor do gradiente exceda 0.45, então o valor da norma é reajustado para o limiar (*threshold*) escolhido (0.45). Isso faz com que não ocorra problemas comuns como o gradiente sumir[54, 29], o que estava acontecendo especificamente nesse caso.

O treino foi realizado no Google Colaboratory [55] usando a GPU NVIDIA Tesla P100 e demorou cerca de 54 minutos. Os resultados do treino estão na figura 4.8. Empiricamente a rede é capaz de executar 200.000 sinais em 5.4 segundos (ou 37.000 sinais por segundo).

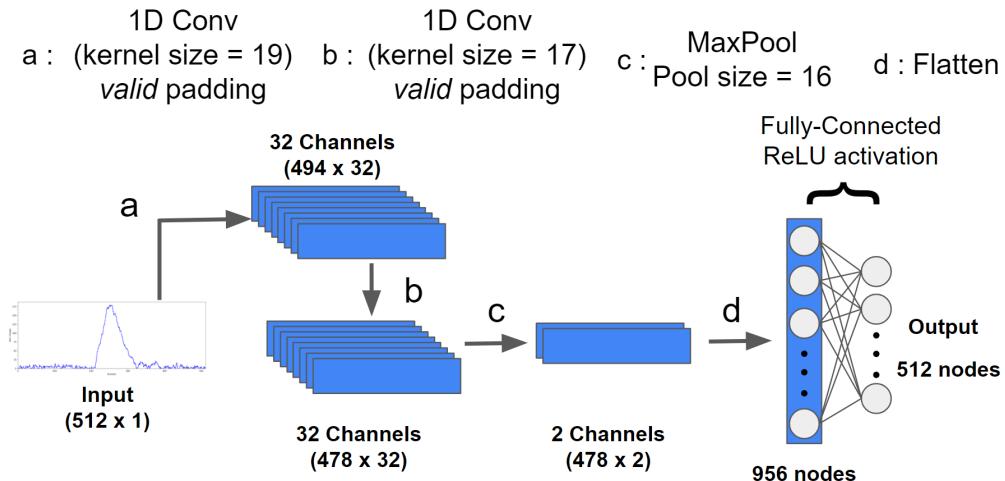
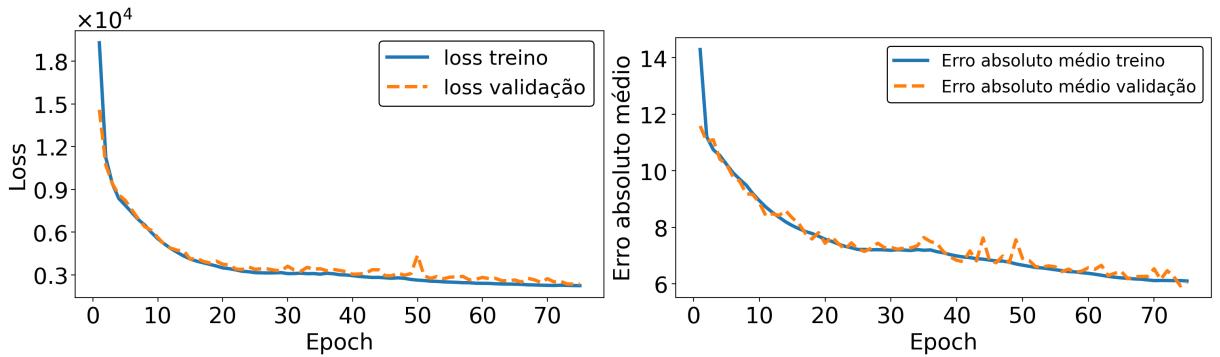


Figura 4.10: Arquitetura da rede neural que faz a inferência da deconvolução do espectro. O vetor de entrada deve ter dimensionalidade 512 x 1. Todas as partes com convolução não possuem o parâmetro *bias*.



(a) *Loss* dos dados de treino (linha contínua) e dos dados de validação (linha tracejada) em função da *epoch* no treino da rede dada por 4.10.

(b) Erro absoluto médio dos dados de treino (linha contínua) e dos dados de validação (linha tracejada) em função da *epoch* no treino da rede dada por 4.10.

Figura 4.11: Resultados do treino da rede neural dada por 4.7.

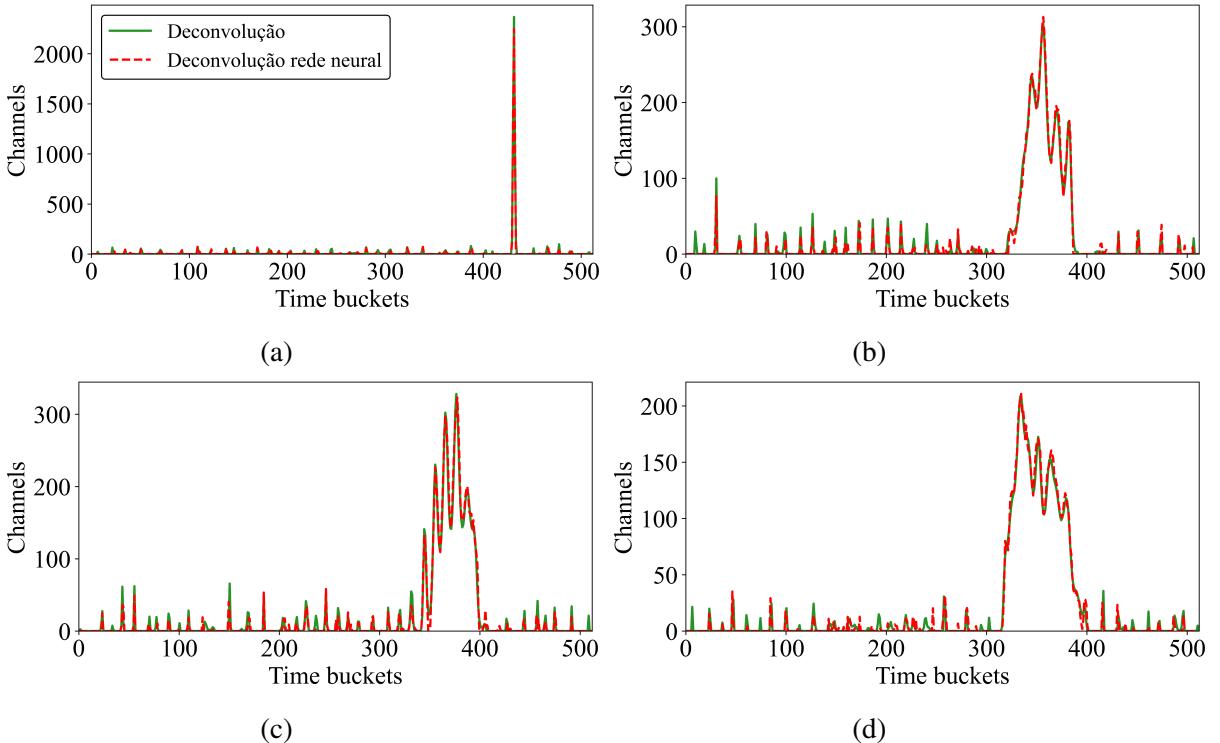


Figura 4.12: Exemplos de deconvolução da rede neural dada por 4.7.

4.2.3 Unificando as duas redes neurais

Usando novamente o TensorFlow 2 podemos carregar as redes neurais já treinadas e usar como se fosse uma única rede neural. A rede receberá de entrada o espectro original e devolverá tanto o fundo quanto o espectro deconvoluído. A arquitetura da rede está na figura 4.13.

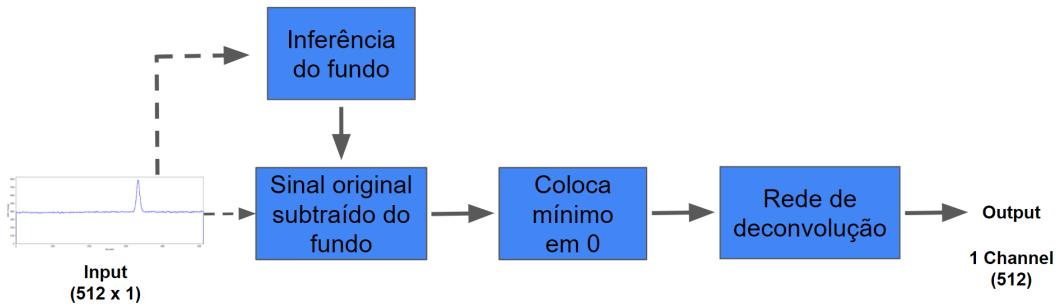


Figura 4.13: Arquitetura da rede neural que faz a inferência da *baseline* e depois faz a deconvolução do espectro. O vetor de entrada deve ter dimensionalidade 512 x 1.

A rede é apenas a sequencia das redes anteriores, ou seja, o espectro passa pelo cálculo da *baseline*, então o espectro original é subtraído dessa *baseline* (colocando o valor mínimo em 0) e por fim ocorre a deconvolução. Pelo fato de cada parte ser treinada de modo separado não há necessidade de treinar a rede unificada, apenas carregar os valores das variáveis treinadas de cada parte. A rede unificada possui 1.053.536 de parâmetros.

Agora devemos comparar a saída dessa nova rede com a saída desejada. Podemos usar novamente o erro médio absoluto para fins de comparação. O erro médio absoluto, para os 200.000 sinais, é de 7.45 Channels. O valor é um pouco maior que o encontrado anteriormente com a rede de deconvolução, portanto podemos verificar a detecção de picos no espectro resultante.

Com o espectro deconvoluído podemos então detectar todos os picos em cada sinal. A detecção será feita com o *SciPy* que possui rotinas para a detecção. A função usada é a “*find_peaks*”, onde é possível determinar altura mínima de detecção, distância mínima entre picos, dentre outros parâmetros, o que torna possível ajustar melhor a detecção em comparação com a saída pelo algoritmo presente no *TSpectrum*.

O histograma da figura 4.14 mostra que, apesar da variação no número de picos detectados pelo *SciPy* (eixo y) em comparação ao número de picos detectados pelo *TSpectrum*, na maioria dos casos o número de detecções é o mesmo, especialmente para os casos em que há um único pico.

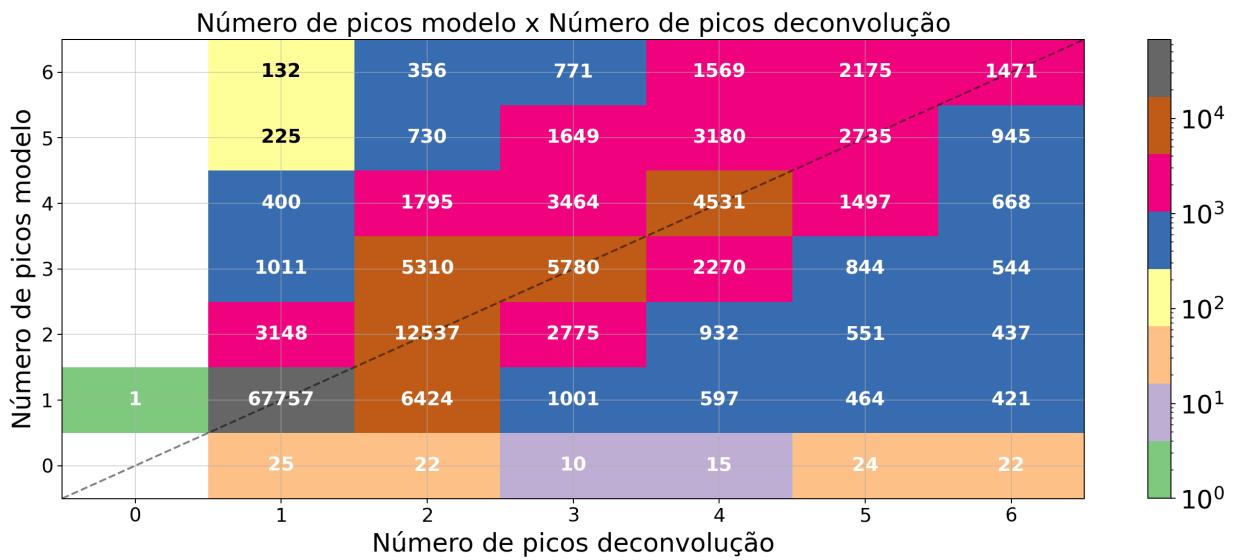


Figura 4.14: Histograma bidimensional que mostra em x a contagem do número de picos detectados por sinal (dado pelo *TSpectrum*) pela deconvolução e em y a contagem do número de picos detectados por sinal (resultante da rede neural) pelo *scipy*. O número de contagens está marcado em cima de cada *bin*.

Com relação a eficiência em tempo, a rede neural da figura 4.13 processa 200.000 sinais em 8 segundos. Somando esse tempo com a detecção de picos com o *SciPy*, o tempo total para processar 200 mil sinais é de cerca de 25 s. Nem foi preciso uma rede neural para os picos e o consumo de tempo já é 60 vezes maior em comparação aos métodos tradicionais mostrados anteriormente.

Exemplos da reconstrução das nuvens de pontos usando *machine learning* com detecção de picos pelo *SciPy* estão na figura 4.15. É possível perceber uma pequena melhora no ruído dos eventos, pois com o *SciPy* podemos calibrar melhor a detecção. Além disso há uma correlação de amplitude do pico detectado entre espectro sem o fundo e o espectro sem o fundo após a deconvolução, como mostrado na figura 4.16. Colocando a condição de que, para cada pico detectado, a razão entre a amplitude do espectro após a deconvolução e do espectro original para o ponto seja maior que 3.8 faz com que a quantidade de pontos ruidosos por evento caia em cerca de 35% em relação aos métodos anteriores.

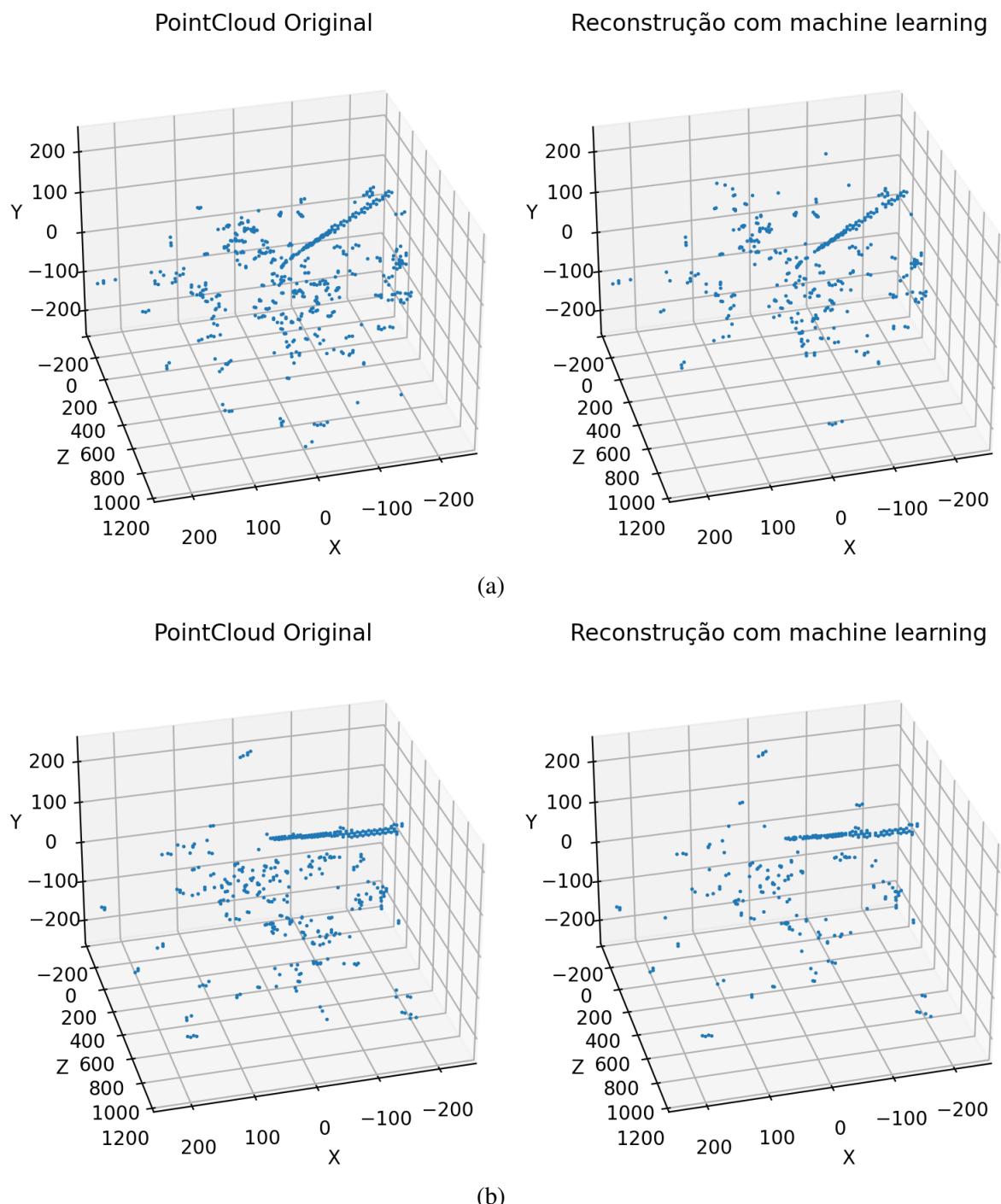


Figura 4.15: Resultados da reconstrução de eventos por *machine learning*.

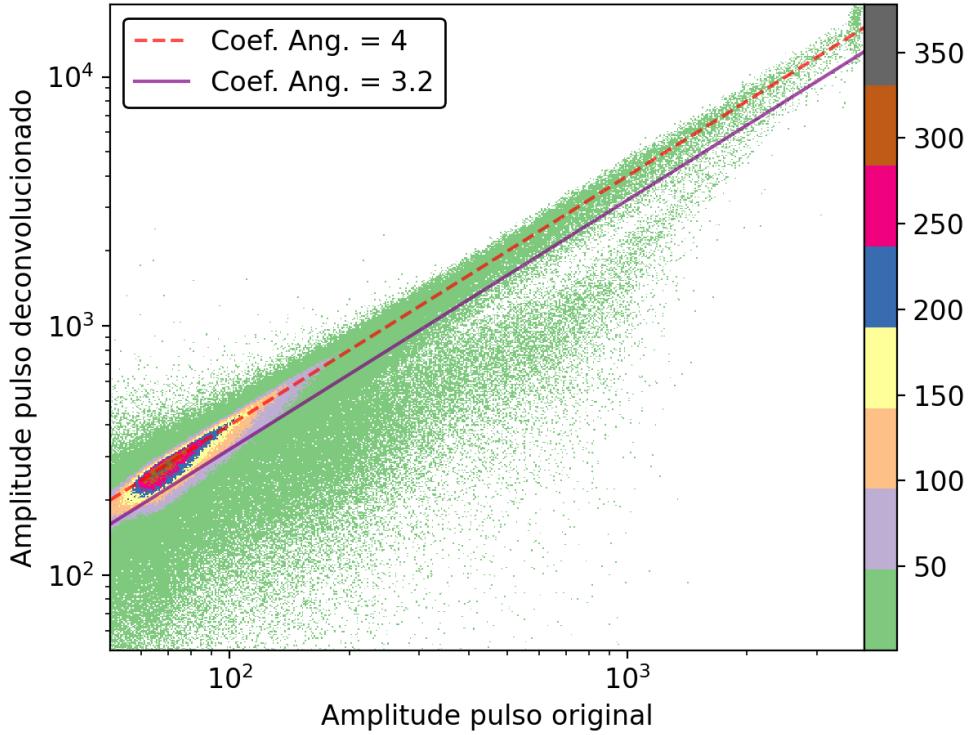


Figura 4.16: Histograma bidimensional que mostra a relação, de cada pico detectado, entre a amplitude do pico após a deconvolução no eixo y e antes da deconvolução no eixo x . A linha tracejada indica a tendência da maior parte dos pontos. Já a linha sólida indica a região de corte dos pontos.

4.2.4 Detecção de picos

A última etapa dessa análise com *machine learning* é analisar possíveis soluções para a detecção de picos. Esse é um problema muito complexo, pois há uma variação muito grande na quantidade de picos por sinal e também há um desbalanço muito grande na quantidade de pontos comuns (aqueles que não são picos) e pontos que são picos. Por exemplo, caso haja um sinal que possui apenas um pico, devemos detectar uma posição, dar o valor de saída como 1, por exemplo, dentre 512 pontos, onde 511 terão o valor de saída como 0. Caso a rede determine que todos os pontos são não picos, ainda assim a acurácia binária seria maior que 99%. Isso é conhecido como desbalanço de classe[56].

Para corrigir esse desbalanço, basta acrescentar pontos simetricamente em torno do pulso. Isso faz com que não seja preciso detectar um único ponto, mas sim uma região em torno do pico. Com isso podemos nos basear na ideia de segmentar o sinal para destacar regiões de interesse[57]. Segmentar significa ter uma rede neural com a saída com o mesmo tamanho do vetor de entrada (512) e saída com valores entre 0 e 1, onde 1

indica uma região com um pico e 0 não. A figura 4.17 mostra um exemplo de um sinal após a deconvolução onde há o pico detectado e os pontos acrescentados para representar a região do pulso.

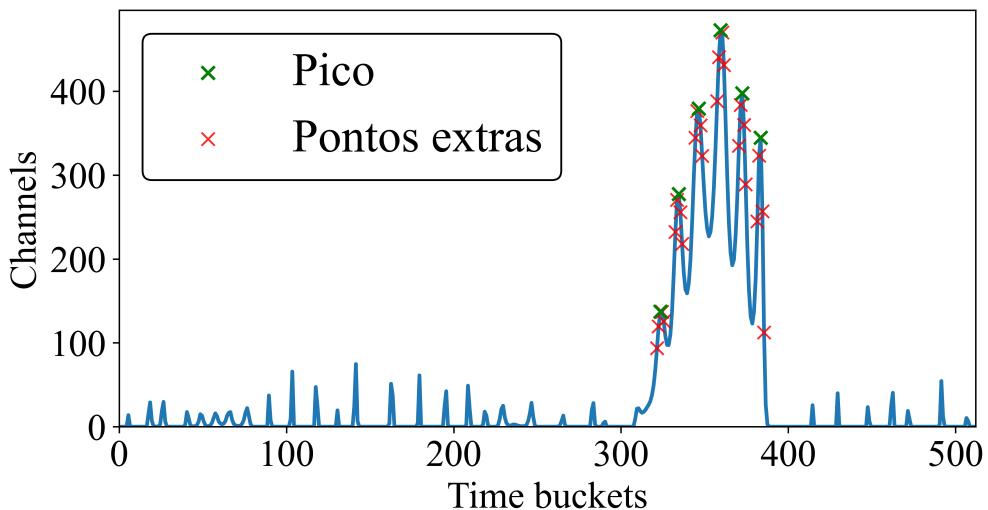


Figura 4.17: Sinal após a deconvolução que mostra o pico detectado mais os pontos adicionais que irão facilitar o trabalho da rede neural (evitar o desbalanço de classe). Foram acrescentados 2 pontos à esquerda e à direita.

A rede construída é a sequência de uma convolução com *kernel* de tamanho 13 e *same padding* seguida de *Max-Pooling* e uma *fully-connected* com função de ativação sigmoide, possuindo um total de 263.104 parâmetros treináveis. Para o treino foram usados sinais que possuíam entre 1 e 6 picos, resultantes da saída do algoritmo *peak_finder* do *SciPy*, o que resultou em 120.024 de dados para o treino e 30.006 para validação. A escolha pelos picos detectados pelo *peak_finder* ao invés do algoritmo de detecção do TSpectrum é pela maior flexibilidade de ajuste fino do algoritmo, tornando a detecção de picos muito melhor. A função custo escolhida foi a *binary cross-entropy* (dada pela equação 3.5), o otimizador o *ADAM* com *learning rate* de 0.001. A métrica utilizada foi a acurácia binária. O treino também foi realizado por uma GPU NVIDIA Tesla P100 e durou cerca de 8 minutos com 12 *epochs*. A arquitetura da rede está na figura 4.18 e os resultados do treino estão na figura 4.19.

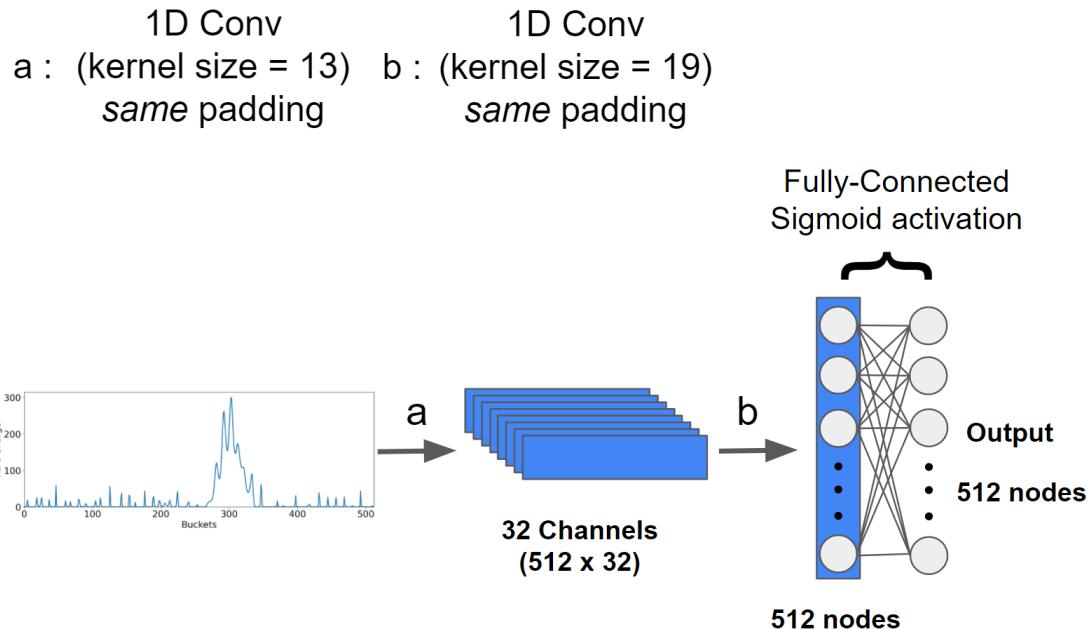
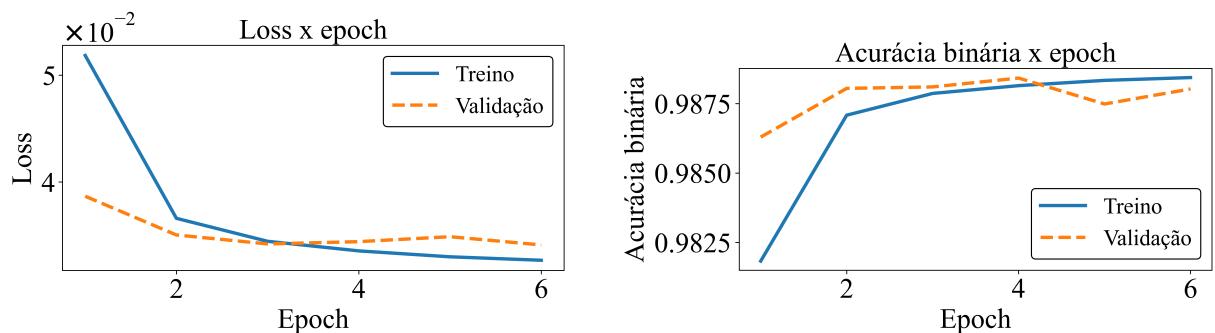


Figura 4.18: Arquitetura da rede neural que faz o recorte das regiões com picos. O vetor de entrada deve ter dimensionalidade 512×1 .



(a) *Loss* dos dados de treino (linha contínua) e dos dados de validação (linha tracejada) em função da *epoch*.

(b) *Acurácia binária* dos dados de treino (linha contínua) e dos dados de validação (linha tracejada) em função da *epoch*.

Figura 4.19: Resultados do treino da rede neural dada por 4.18.

A saída da rede neural é um vetor de tamanho 512 com valores entre 0 e 1, onde valores maiores que 0.5 são considerados como pertencentes à uma região com um centroide. Com as regiões identificadas podemos fazer uma média ponderada com o espectro de entrada para achar o centroide. O tempo de processamento da rede neural é de 150.030 sinais em cerca de 4.11 segundos (aproximadamente 36.500 sinais por segundo). Para determinar os picos a partir da saída da rede o tempo é de aproximadamente 4.3 segundos, onde o algoritmo pode ser ainda mais rápido se for paralelizado.

Unificando a rede neural da figura 4.13 com a rede de segmentação da figura 4.18 o tempo para processar 200 mil sinais e depois identificar os picos é de aproximadamente

16.5 segundos, usando a GPU Tesla P100 e o processador (para processar a segmentação) Ryzen 5 3600X. Há um pequeno ganho de tempo em relação ao algoritmo *peak_finder*. Resultados para picos detectados usando as três redes acopladas estão na figura 4.20.

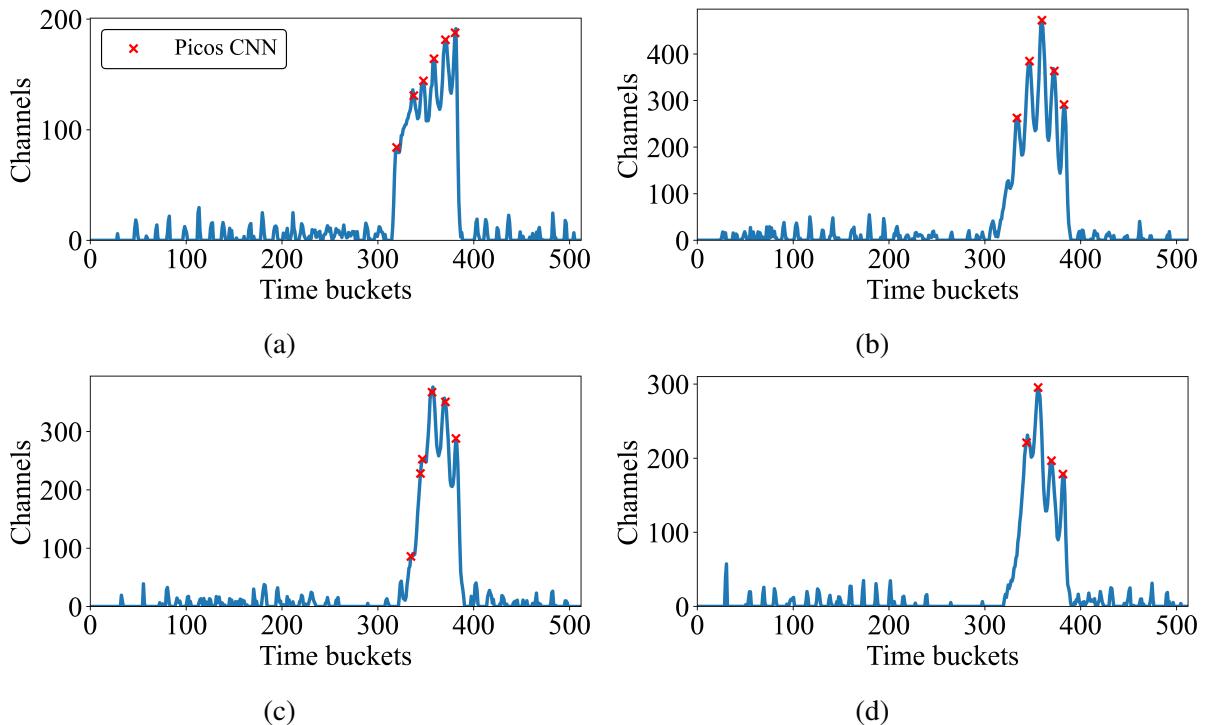


Figura 4.20: Exemplos de deconvolução da rede neural dada por 4.7.

Com as redes neurais desenvolvidas, o consumo de tempo para o processamento dos pulsos diminui entre 60 e 80 vezes em relação à algoritmos mais tradicionais, abrindo possibilidades para a análise em tempo real de um experimento. Com as nuvens de pontos reconstruídas, podemos seguir adiante e identificar as reações nucleares nos eventos

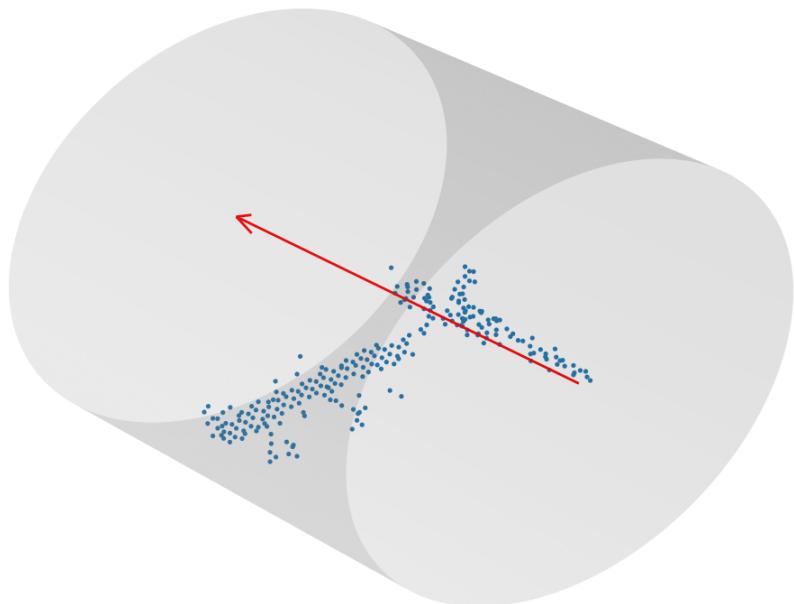
Capítulo 5

Análise das nuvens de pontos

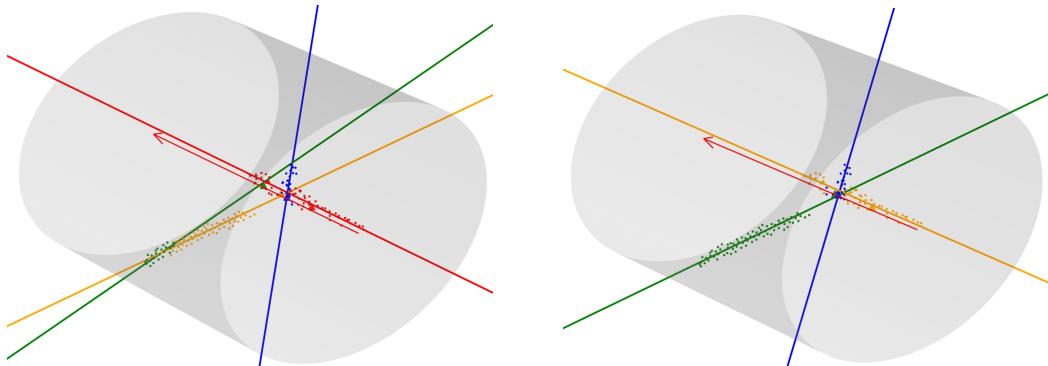
Esse capítulo irá mostrar como são analisadas as nuvens de pontos. O objetivo desta etapa, do ponto de vista computacional, é detectar *clusters*, que nesse caso são retas tridimensionais. Com a informação da física do problema podemos distinguir cada reta detectada e determinar o vértice de reação entre uma partícula originada originada de uma reação nuclear, e o feixe. Após a detecção, devemos selecionar os eventos que possuam reações nucleares.

Na seção 5.1 será mostrado como identificar os *clusters* a partir de estimadores robustos. Na seção ?? a mesma identificação será feita a partir de algoritmos de *machine learning*, mostrando diferenças em relação ao método anterior.

Um exemplo de evento pode ser visto na figura 5.1a.



(a) Exemplo de evento a ser analisado. Os pontos em azul são das partículas detectadas pelo TPC, a seta vermelha indicando o sentido do feixe e o TPC está representado pelo cilindro cinza.



(b) Evento sem a clusterização. As retas de cor amarela e verde são de um único *cluster*.

(c) Evento após a clusterização. Agora o evento possui as duas retas corretas, a azul e a verde.

Figura 5.1: Sequência de análise de um evento. Em 5.1a temos o evento que é recebido para ser analisado, em 5.1b temos o mesmo evento após o RANSAC (antes da clusterização) e 5.1c mostra depois da correção. As cores das retas são arbitrárias e servem apenas para a diferenciação.

A etapa anterior nos fornece arquivos no formato *ROOT* para cada *run* do experimento. O arquivo é dividido por eventos em que estão contidas informações, em *trees*, como coordenadas x, y, z, t, carga, tempo de voo, além de canais adicionais (indicadores de partículas presentes além da parte central do *micromegas*). Os arquivos foram lidos em Python usando a biblioteca Uproot[58].

A figura 5.1a nos mostra um exemplo que parece indicar mais de uma trajetória dentro do TPC, e é preciso separar os pontos de cada trajetória identificada. O objetivo é separar

os pontos de *clusters* que contenham a informação do momento da partícula, obtida a partir das propriedades geométricas do versor das retas e da energia inicial da trajetória. A identificação pode ser feita sem o uso de algoritmos de machine learning, que chamarei de método usual, e usando machine learning.

5.1 Método usual

Para identificar as retas (*clusters*) presentes em eventos como mostrados na figura 5.1a, devemos usar estimadores robustos que consigam fazer a detecção mesmo com uma presença grande de *outliers*, pontos que são considerados ruídos. Existem muitos estimadores, que são apenas variações do Random sample consensus (*RANSAC*) [59], como o MLESAC[60], *RRANSAC*, *PROSAC*, dentre outros, que estão presentes no *Point Cloud Library* (PCL)[61]. Todos esses se baseiam na ideia de estimar várias possíveis retas de forma aleatória e selecionam a melhor usando uma estimativa (por isso o nome *consensus*). Existe ainda a opção do *Hough Transform*[45], porém é um algoritmo de difícil expansão e não demonstrou resultados bons.

Os algoritmos do PCL foram testados e os que mostraram os melhores resultados foram o *RANSAC* e o *PROSAC*, não havendo muita diferença entre os dois. O estimador usado foi uma variação do *RANSAC*, desenvolvida para fazer melhores estimativas em dados como os do pAT-TPC[62]. O algoritmo 1, que chamei de *Enhanced RANSAC* (Para você Juan: só quis diferenciar o nome, até pq eu quero colocar no pypi com esse nome, pode ser?), mostra o seu funcionamento. A melhor estimativa original (que retorna a melhor reta) era simplesmente o número total de *inliers* de um *cluster*. A nova estimativa *C* passou a ser

$$C = \sum_{i=0}^N \frac{d_i^2}{N}, \quad (5.1)$$

onde *N* é o número total de pontos de uma reta e d_i é a distância do i-ésimo ponto à reta.

O *RANSAC* consegue identificar apenas pontos de uma única reta, porém cada evento pode ter mais de uma reta. O novo algoritmo se baseia na ideia de usar o *RANSAC* sequencialmente, ou seja, no momento que um *cluster* tem um tamanho mínimo, então ele é guardado para depois, então, poder ser escolhidos dentre as melhores estimativas *C*. Com isso podemos selecionar quantas retas forem possíveis.

Para diminuir o número de pontos a serem analisados pelo algoritmo 1, passamos a *pointcloud* por dois filtros. O primeiro filtro exclui pontos baseado na sua carga. O

Algoritmo 1: Enhanced RANSAC

Dados: pointcloud, N , d_{min} , tam_{min}

1 **para** cada iteração $i = 1, 2, \dots, N$ **faz**

2 Seleciona dois pontos da *pointcloud* de modo aleatório (*Random Sampling*);

3 Estima versor v e um ponto P_b que passe pela reta r formada pelos dois pontos;

4 **para** cada ponto P **faz**

5 Calcula a distância d do ponto à reta r ;

6 **se** $d < d_{min}$ **então**

7 Guarda P como pertencente à r ;

8 **se** Número de pontos de $r > tam_{min}$ **então**

9 Guarda v , P_b e C ;

10 Ordena as retas do menor para o maior C ;

11 **para** cada reta r ordenada **faz**

12 **se** Número de pontos de $r > tam_{min}$ **então**

13 Guarda v , P_b e pontos $P \in r$;

14 **retorna** Retas r selecionadas na última etapa;

segundo filtro, chamado de *outlier removal*, elimina pontos considerados *outliers* globais, analisando a vizinhança ao redor do ponto. O *outlier removal* está presente na biblioteca Open3D[63] no Python e funciona excluindo pontos muito isolados uns dos outros.

Uma mudança na eficiência do algoritmo 1 é na linha 2, chamada de *Random Sampling*. Podemos nos beneficiar do *Monte Carlo Rejecting* na escolha de dois pontos aleatórios. Por exemplo, caso sejam selecionados pontos muito próximos, o que implicaria em uma reta com poucos pontos, podemos descarta-los. Essa ideia faz com que a primeira etapa seja mais eficiente.

O algoritmo 1 possui uma eficiência muito alta, acertando quais os *clusters* que existem em um evento, porém ainda assim não é perfeito. Uma das falhas do algoritmo é que ele pode selecionar retas muito próximas e entender que não são um único *cluster*, e sim dois distintos. A figura 5.1b mostra um exemplo em que foram detectadas três retas, onde deveria existir apenas duas (é necessário combinar duas delas).

A etapa de correção da saída do *Enhanced Ransac* é chamada de clusterização. A ideia é comparar, dois a dois, todos os *clusters* resultantes da saída e usar algum critério para juntar ou não os dois clusters. O problema precisa abordado avaliando a semelhança entre dois *clusters*. Uma métrica possível é o coeficiente de silhueta [64].

O coeficiente de silhueta compara dois *clusters* e retorna um valor entre -1 e 1 que informa se estão separados corretamente. O valor -1 informa que a separação dos *clus-*

ters estão errados, 0 indica sobreposição e 1 significa que a separação está correta. A métrica se mostra muito viável para análise de duas dimensões, porém para o caso de três dimensões essa métrica não se mostrou muito eficiente. Mesmo colocando um *threshold* muito baixo o algoritmo indicava que deveria juntar *clusters* mesmo que muito distantes um do outro.

A abordagem correta se dá primeiro comparando o ângulo entre as duas retas. Caso a diferença absoluta entre os ângulos com relação ao versor $(0, 0, 1)$ seja menor que um valor dado, que nesse caso foi considerado 9° (determinado empiricamente), então as duas retas serão combinadas se obedecerem a condição dada pela equação 5.2.

$$\sum_{i=0}^{N_1} \frac{d_{i2}}{N_1} < \alpha d_{min}, \quad (5.2)$$

onde N_1 é o número de pontos da reta 1, d_{i2} a distância do ponto i da reta 1 em relação à reta 2, α é um parâmetro com valor a ser escolhido e d_{min} é a distância mínima de ponto a reta usada no algoritmo 1. Os valores escolhidos foram tais que $\alpha = 1.75$ e $d_{min} = 15$ mm. A figura 5.1c mostra o antes e depois da clusterização ser aplicada em um evento.

Após a etapa de clusterização é necessário classificar cada reta como sendo ou o feixe, ou uma partícula espalhada. O feixe deve incidir no TPC com um ângulo muito pequeno com relação ao versor $(0, 0, 1)$. Além disso, mesmo se o ângulo for muito pequeno, a reta do feixe deve cruzar o plano da janela do TPC muito próximo do ponto mais provável da entrada o feixe. O ponto mais provável foi calculado usando a posição média da projeção dos pontos de uma *run* no plano *xy*. Disso obtemos que a posição inicial mais provável do feixe é tal que $x = -3.4$ (6.7) mm e $y = -0.9$ (6.3) mm. A incerteza é alta pois o feixe incide em muitas posições diferentes da janela.

Portanto se o ângulo entre o versor \hat{v}_i de uma reta i for menor que 5° (determinado de modo empírico novamente) e a distância d entre o ponto P_i que intercepta o plano e o ponto $(x, y, 0)$, dados no parágrafo anterior, for menor que 15 mm (pouco mais que duas vezes a incerteza de cada ponto), então a reta é considerada como o feixe do evento. No caso de não satisfazer essas condições, então ela classificada como uma provável partícula originada da reação do feixe com o gás.

Importante notar que, pelo baixo ganho da parte central do *micromegas*, há eventos que não possuem feixe, como mostrado na figura 5.2. Neste caso é necessário assumir as propriedades da reta mais provável para o feixe, ou seja, precisa passar pelo ponto $(x, y, 0)$ e ter versor $(0, 0, 1)$.

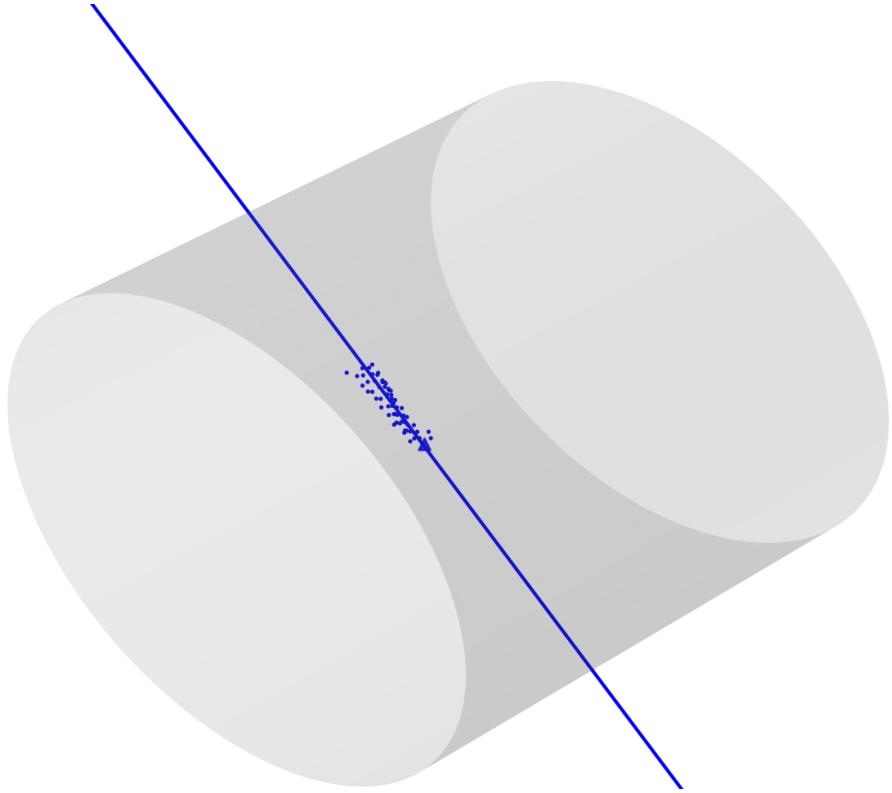


Figura 5.2: Evento em que não foi detectado o feixe, apenas a partícula espalhada. O triângulo azul é o local calculo do vértice de reação dado pela equação 5.7.

Nem todas as retas que não são o feixe são originadas da reação do feixe com o gás. Para determinar sua origem é necessário determinar o vértice de reação com o feixe. O vértice de reação é o ponto médio do segmento de reta que conecta a reta a ser analisada e o feixe no ponto de menor distância entre as retas. Ele permite ainda fazer correções futuras e diferenciar o começo e o fim de uma reta, pois é possível determinar o ponto tal onde ocorreu uma reação.

Temos as seguintes equações das retas \vec{P}_1 e \vec{P}_2 como vetores:

$$\begin{aligned}\vec{P}_1 &= \vec{A}_1 + \vec{V}_1 * t_1 \\ \vec{P}_2 &= \vec{A}_2 + \vec{V}_2 * t_2,\end{aligned}\tag{5.3}$$

onde \vec{A}_1 e \vec{A}_2 são pontos arbitrários que pertencem as retas 1 e 2, respectivamente, \vec{V}_1 e \vec{V}_2 são os versores, t_1 e t_2 são os hiperparâmetros das retas.

A reta que conecta a menor distância possui versor

$$\vec{V}_c = \frac{\vec{V}_1 \times \vec{V}_2}{|\vec{V}_1 \times \vec{V}_2|}.\tag{5.4}$$

Podemos então construir uma reta \vec{P}_3 que conecta \vec{P}_1 e \vec{P}_2 . Essa reta deve começar no ponto de menor distância da reta 1 e terminar no ponto de menor distância da reta 2. Ou seja, temos o seguinte sistema linear:

$$\vec{A}_2 + \vec{V}_2 * \tilde{t}_2 = \vec{A}_1 + \vec{V}_1 * \tilde{t}_1 + \vec{V}_c * \tilde{t}_3.$$

Rearranjando temos que

$$\vec{V}_1 * \tilde{t}_1 - \vec{V}_2 * \tilde{t}_2 + \vec{V}_c * \tilde{t}_3 = \vec{A}_2 - \vec{A}_1, \quad (5.5)$$

onde \tilde{t}_1 , \tilde{t}_2 e \tilde{t}_3 são os hiperparâmetros a serem determinados. Caso \vec{V}_1 seja paralelo à \vec{V}_2 , então não há solução (não há vértice de reação). Achando os valores, achamos os pontos de menor distância nas duas retas:

$$\begin{aligned} \vec{P}_1 &= \vec{A}_1 + \vec{V}_1 * \tilde{t}_1 \\ \vec{P}_2 &= \vec{A}_2 + \vec{V}_2 * \tilde{t}_2. \end{aligned} \quad (5.6)$$

Conseguimos então determinar que o vértice de reação \vec{V}_r é dado por

$$\vec{V}_r = \frac{1}{2}(\vec{P}_1 + \vec{P}_2). \quad (5.7)$$

Também podemos definir a distância de máxima aproximação d_{max} das retas, dada pela equação 5.8.

$$d_{max} = |\vec{P}_1 - \vec{P}_2|. \quad (5.8)$$

$$d_{max} = |(\vec{P}_1 - \vec{P}_2)|. \quad (5.9)$$

Da equação 5.8 podemos estabelecer um limite máximo de distância que uma reta pode ter do feixe, para então definir se houve realmente a reação no vértice de reação definido pelo equação 5.7. Retas que possuíam d_{max} maior ou igual que 25mm foram automaticamente descartadas.

Existem ainda situação em que a reta e o feixe tem uma distância de máxima aproximação muito pequena, porém o vértice de reação estava fora dos limites do TPC ($|x| < 140\text{mm}$, $|y| < 140\text{mm}$ e $|t| < 512$), o que também indica que a reta deve ser descartada.

Agora com apenas as retas certas selecionadas devemos apenas tomar cuidado de excluir eventos em que apenas o feixe foi detectado, pois neste caso não há o evento

físico para ser analisado.

5.2 Métodos com *machine learning*

Na seção 5.1 descrevemos o algoritmo completo de seleção de eventos para a análise. Agora o objetivo é aplicar algoritmos de *machine learning* para melhorar o funcionamento de algo já existente ou resolver o mesmo problema de uma maneira totalmente diferente.

5.2.1 Clusterização hierárquica

Como queremos classificar pontos, dando *labels* que diferenciem os pontos de cada reta, podemos usar algoritmos de clusterização (*machine learning* não supervisionado). Devemos escolher algoritmos que sejam muito rápidos pois a ideia é substituir o uso do RANSAC. O algoritmo usado é o *Hierarchical Density-Based Spatial Clustering of Applications with Noise* (HDBSCAN)[65, 66], pois é muito eficiente em tempo e consegue realizar *clustering* mesmo em dados muito complexos. O algoritmo consegue também nos dar os *outliers* da *pointcloud*, porém para eliminação de *outliers* será usado novamente o *outlier removal* do Open3D. A figura 5.3 mostra resultados do uso do algoritmo, já com as retas ajustadas. Percebe-se casos em que a houve falha na clusterização.

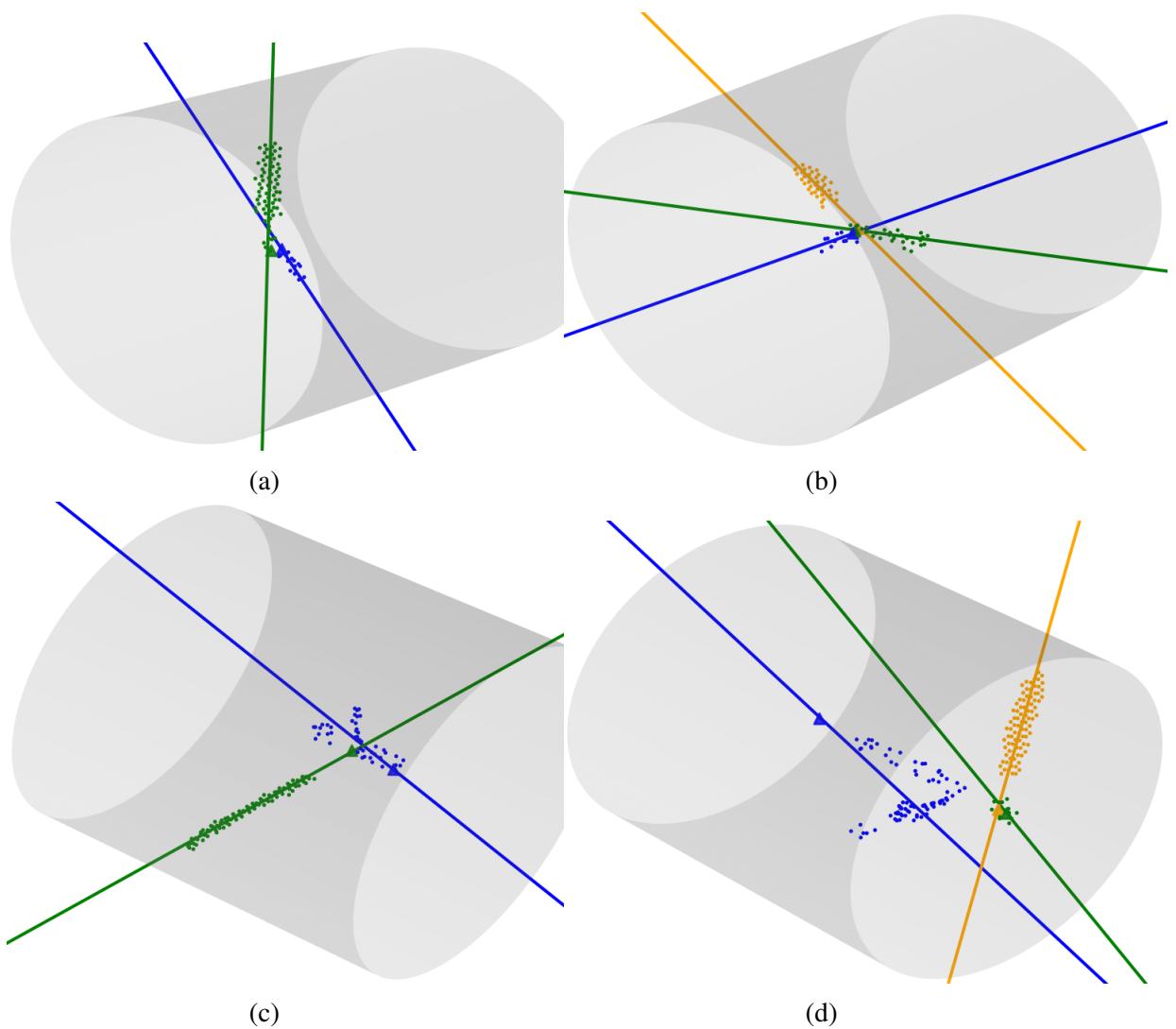


Figura 5.3: Exemplos dos resultados para o HDBSCAN. Percebe-se que em 5.3c e 5.3d o algoritmo falhou, juntando diferentes *clusters* ou simplesmente detectando ruído junto da *track*.

A eficiência em relação ao RANSAC é menor, pois os *clusters* são pouco densos, o que dificulta a seleção de *clusters* pelo algoritmo. Além disso, o algoritmo tem dificuldade em separar *clusters* que possuem pontos sobrepostos, ou seja, onde há vértice de reação (vide figuras 5.3c e 5.3d). A tabela 5.1 mostra a comparação entre a taxa de acerto e a eficiência em tempo entre o HDBSCAN e do RANSAC. A taxa de acertos mostra quantos foram corretamente solucionados (ou seja, todos os *clusters* foram detectados corretamente), e a eficiência mostra a capacidade de processamento de eventos pelo algoritmo, medida em eventos por segundo. O *benchmark* foi feito usando o processador Ryzen 5 3600X.

Table 5.1: Comparação entre algoritmos usados para identificar tracks em eventos. O HDBSCAN acerta menos vezes em comparação com o RANSAC (cerca de 14% menos), porém é quase 4 vezes mais rápido.

Método	Taxa de acertos (%)	Eficiência (eventos/s)
RANSAC	78.2	57
HDBSCAN	64.3	208

A clusterização se mostrou melhor em eventos que tinham uma separação clara entre os *clusters*, sem pontos que coincidem duas *tracks* diferentes[67], e também nos casos em que a densidade de pontos era muito significativo. Apesar da queda na taxa de acertos a velocidade de execução sobe significativamente, sendo uma possível escolha no lugar do RANSAC.

Capítulo 6

Resultados

Após identificar todos os *clusters* de cada evento devemos identificar quais são as partículas que originaram cada uma das trajetórias detectadas. Temos o comprimento de cada trajetória e sua energia, portanto o objetivo é, dada essas duas informações, identificar qual a partícula. A figura 6.1 mostra um histograma bidimensional do comprimento de cada *track* em função do ângulo de espalhamento no referencial do laboratório, usando apenas eventos que possuíam duas *tracks* com o mesmo vértice de reação. É possível perceber as medidas coincidentes do ^{16}O e do próton.

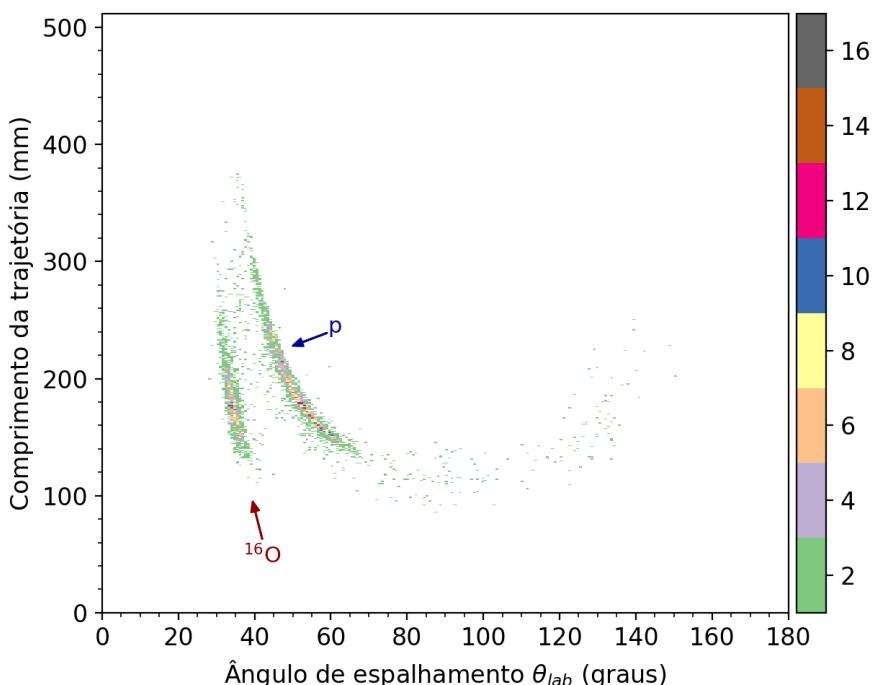


Figura 6.1: Histograma de comprimento de *track* no eixo y e ângulo de espalhamento no eixo x. O histograma foi feito coletando eventos que possuíam duas trajetórias com o mesmo vértice de reação, indicando a detecção simultânea do ^{16}O e do próton.

Para determinar qual é a partícula de cada *track* podemos usar o LISE++[68] para calcular o alcance (*range*) das possíveis partículas (^{17}F , ^{16}O e próton) dada as propriedades do alvo (^4He à uma pressão de 350 Torr). A figura ?? mostra o alcance em mm das partículas em função da energia em MeV.

Capítulo 7

Conclusão

Referências

- [1] F.D. Becchetti et al. “The TwinSol low-energy radioactive nuclear beam apparatus: status and recent results”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 505.1 (2003). Proceedings of the tenth Symposium on Radiation Measurements and Applications, pp. 377 –380. ISSN: 0168-9002. DOI: [https://doi.org/10.1016/S0168-9002\(03\)01101-X](https://doi.org/10.1016/S0168-9002(03)01101-X). URL: <http://www.sciencedirect.com/science/article/pii/S016890020301101X>.
- [2] D. Suzuki et al. “Prototype AT-TPC: Toward a new generation active target time projection chamber for radioactive beam experiments”. In: *Nuclear Instruments and Methods in Physics Research Section A Accelerators Spectrometers Detectors and Associated Equipment* 691 (Nov. 2012), p. 39. DOI: [10.1016/j.nima.2012.06.050](https://doi.org/10.1016/j.nima.2012.06.050).
- [3] J.J. Kolata et al. “A radioactive beam facility using a large superconducting solenoid”. In: *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms* 40-41 (1989), pp. 503–506. ISSN: 0168-583X. DOI: [https://doi.org/10.1016/0168-583X\(89\)91032-X](https://doi.org/10.1016/0168-583X(89)91032-X). URL: <https://www.sciencedirect.com/science/article/pii/0168583X8991032X>.
- [4] Panagiotis Gastis et al. “Improved phenomenological description of equilibrium charge state distributions for Ni, Co, and Cu ions in Mo based on new experimental data at 2 MeV/u”. In: (Sept. 2015).
- [5] L. E. Tamayose et al. *Simulation of the RIBRAS Facility with GEANT4*. 2022. arXiv: 2202.07180 [physics.acc-ph].
- [6] J.C. Zamora. “Estudo do espalhamento elástico dos isótopos ${}^7\text{Be}$, ${}^9\text{Be}$ e ${}^{10}\text{Be}$ em alvo de ${}^{12}\text{C}$ ”. MA thesis. University of Sao Paulo, Brazil, 2011. DOI: [10.11606/D.43.2011.tde-30092011-132427](https://doi.org/10.11606/D.43.2011.tde-30092011-132427).

- [7] S. Agostinelli et al. “Geant4—a simulation toolkit”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (2003), pp. 250–303. ISSN: 0168-9002. DOI: [https://doi.org/10.1016/S0168-9002\(03\)01368-8](https://doi.org/10.1016/S0168-9002(03)01368-8). URL: <https://www.sciencedirect.com/science/article/pii/S0168900203013688>.
- [8] R. Lichtenthäler et al. “Radioactive ion beams in Brasil (RIBRAS)”. en. In: *Brazilian Journal of Physics* 33 (June 2003), pp. 294 –296. ISSN: 0103-9733. URL: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-97332003000200025&nrm=iso.
- [9] Sergio Muniz, Mishkat Bhattacharya, and V. Bagnato. “Simple analysis of off-axis solenoid fields using the scalar-magnetostatic potential: application to a Zeeman-slower for cold atoms”. In: (Mar. 2010).
- [10] Jaspreet Singh Randhawa et al. “Beam induced space-charge effects in Time Projection Chambers in low-energy nuclear physics experiments”. In: *Nucl. Instr. and Meth. A* (July 2019).
- [11] Y. Giomataris et al. “MICROMEGAS: a high-granularity position-sensitive gaseous detector for high particle-flux environments”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 376.1 (1996), pp. 29–35. ISSN: 0168-9002. DOI: [https://doi.org/10.1016/0168-9002\(96\)00175-1](https://doi.org/10.1016/0168-9002(96)00175-1). URL: <https://www.sciencedirect.com/science/article/pii/0168900296001751>.
- [12] J. Bradt et al. “Commissioning of the Active-Target Time Projection Chamber”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 875 (2017), pp. 65 –79. ISSN: 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2017.09.013>. URL: <https://www.sciencedirect.com/science/article/pii/S0168900217309683>.
- [13] J. W. Bradt. “Measurement of isobaric analogue resonances of ^{47}Ar with the Active-Target Time Projection Chamber”. PhD thesis. Michigan State University, USA, 2017. URL: https://publications.nscl.msu.edu/thesis/%20Brandt_2017_5279.pdf.

- [14] J. Giovinazzo et al. “GET electronics samples data analysis”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 840 (2016), pp. 15–27. ISSN: 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2016.09.018>. URL: <https://www.sciencedirect.com/science/article/pii/S0168900216309408>.
- [15] Thomas Lohse and Werner Witzeling. “The Time Projection Chamber”. In: *Instrumentation in High Energy Physics*, pp. 81–155. DOI: 10.1142/9789814360333_0002. eprint: https://www.worldscientific.com/doi/pdf/10.1142/9789814360333_0002. URL: https://www.worldscientific.com/doi/abs/10.1142/9789814360333_0002.
- [16] A. Geron. *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems*. 2nd ed. O’Reilly, 2019. ISBN: 9781492032649.
- [17] Maryam M. Najafabadi et al. “Deep learning applications and challenges in big data analytics”. In: *Journal of Big Data* 2.1 (2015), p. 1. ISSN: 2196-1115. DOI: 10.1186/s40537-014-0007-7. URL: <https://doi.org/10.1186/s40537-014-0007-7>.
- [18] Morten Hjorth-Jensen. *Data Analysis and Machine Learning: Neural networks, from the simple perceptron to deep learning*. 2020. URL: <https://nucleartalent.github.io/MachineLearningECT/doc/pub/Day4/pdf/Day4.pdf> (visited on 08/2020).
- [19] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York, 2016. ISBN: 9781493938438. URL: <https://books.google.com.br/books?id=kOXDtAEACAAJ>.
- [20] Steven B. Damelin and Willard Miller Jr. *The Mathematics of Signal Processing*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2011. DOI: 10.1017/CBO9781139003896.
- [21] K.-L Du and M.N.S Swamy. “Radial Basis Function Networks”. In: Dec. 2014, pp. 299–335. ISBN: 978-1-4471-5570-6. DOI: 10.1007/978-1-4471-5571-3_10.

- [22] Alex Sherstinsky. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network”. In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306. ISSN: 0167-2789. DOI: <https://doi.org/10.1016/j.physd.2019.132306>. URL: <https://www.sciencedirect.com/science/article/pii/S0167278919305974>.
- [23] Abien Fred Agarap. “Deep Learning using Rectified Linear Units (ReLU)”. In: (Mar. 2018).
- [24] Moraga C. Han J. “The influence of the sigmoid function parameters on the speed of backpropagation learning”. In: *Lecture Notes in Computer Science* 930 (1995). DOI: https://doi.org/10.1007/3-540-59497-3_175.
- [25] Tomasz Szandała. *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks*. Oct. 2020. DOI: https://doi.org/10.1007/978-981-15-5495-7_11.
- [26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [27] Hecht-Nielsen. “Theory of the backpropagation neural network”. In: *International 1989 Joint Conference on Neural Networks*. 1989, 593–605 vol.1. DOI: [10.1109/IJCNN.1989.118638](https://doi.org/10.1109/IJCNN.1989.118638).
- [28] Douglas Kline and Victor Berardi. “Revisiting squared-error and cross-entropy functions for training neural network classifiers”. In: *Neural Computing and Applications* 14 (Dec. 2005), pp. 310–318. DOI: [10.1007/s00521-005-0467-y](https://doi.org/10.1007/s00521-005-0467-y).
- [29] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).
- [30] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *ArXiv* abs/1609.04747 (2016).
- [31] Mohammad Hossin and Sulaiman M.N. “A Review on Evaluation Metrics for Data Classification Evaluations”. In: *International Journal of Data Mining & Knowledge Management Process* 5 (Mar. 2015), pp. 01–11. DOI: [10.5121/ijdkp.2015.5201](https://doi.org/10.5121/ijdkp.2015.5201).

- [32] N. S. Altman. “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression”. In: *The American Statistician* 46.3 (1992), pp. 175–185. DOI: 10.1080/00031305.1992.10475879. eprint: <https://www.tandfonline.com/doi/pdf/10.1080/00031305.1992.10475879>. URL: <https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879>.
- [33] Angela Serra and Roberto Tagliaferri. “Unsupervised Learning: Clustering”. In: Jan. 2018. ISBN: 9780128096338. DOI: 10.1016/B978-0-12-809633-8.20487-1.
- [34] Kristina Sinaga and Miin-Shen Yang. “Unsupervised K-Means Clustering Algorithm”. In: *IEEE Access* PP (Apr. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.2988796.
- [35] Y Reddy, Viswanath Pulabaigari, and Eswara B. “Semi-supervised learning: a brief review”. In: *International Journal of Engineering & Technology* 7 (Feb. 2018), p. 81. DOI: 10.14419/ijet.v7i1.8.9977.
- [36] Michael L Littman Leslie Pack Kaelbling and Andrew W Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research* (1996), 237–285. DOI: <https://doi.org/10.1613/jair.301>.
- [37] Kim D et al. “Review of machine learning methods in soft robotics”. In: *PLoS ONE* (2021). DOI: <https://doi.org/10.1371/journal.pone.0246102>.
- [38] Stefano Carboni et al. “Particle identification using the (DELT)A-E-E technique and pulse shape discrimination with the silicon detectors of the FAZIA project”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 664 (Feb. 2012), 251–263. DOI: 10.1016/j.nima.2011.10.061.
- [39] Erich Schubert et al. “DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN”. In: *ACM Trans. Database Syst.* 42.3 (2017). ISSN: 0362-5915. DOI: 10.1145/3068335. URL: <https://doi.org/10.1145/3068335>.
- [40] Guolin Ke et al. “LightGBM: A Highly Efficient Gradient Boosting Decision Tree”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, 3149–3157. ISBN: 9781510860964.

- [41] Ze-Peng Gao et al. “Machine learning the nuclear mass”. In: *Nuclear Science and Techniques* 32.10 (2021), p. 109. ISSN: 2210-3147. DOI: 10.1007/s41365-021-00956-1. URL: <https://doi.org/10.1007/s41365-021-00956-1>.
- [42] R Utama, Wei-Chia Chen, and J Piekarewicz. “Nuclear charge radii: density functional theory meets Bayesian neural networks”. In: *Journal of Physics G: Nuclear and Particle Physics* 43.11 (2016), p. 114002. DOI: 10.1088/0954-3899/43/11/114002. URL: <https://doi.org/10.1088/0954-3899/43/11/114002>.
- [43] Niu Zhongming et al. “Predictions of nuclear β -decay half-lives with machine learning and their impact on r-process nucleosynthesis”. In: *Physical Review C* 99 (June 2019). DOI: 10.1103/PhysRevC.99.064307.
- [44] M.P. Kuchera et al. “Machine learning methods for track classification in the AT-TPC”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 940 (2019), pp. 156–167. ISSN: 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2019.05.097>. URL: <https://www.sciencedirect.com/science/article/pii/S0168900219308046>.
- [45] Christoph Dalitz, Tilman Schramke, and Manuel Jeltsch. “Iterative Hough Transform for Line Detection in 3D Point Clouds”. In: *Image Processing On Line* 7 (2017). <https://doi.org/10.5201/ipol.2017.208>, pp. 184–196.
- [46] Edmundo Capelas de Oliveira and José Emilío Maiorino. *Introdução aos métodos da matemática aplicada*. 3rd ed. Editora UNICAMP, 2010. ISBN: 9788526809062.
- [47] *ROOT Data Analysis Framework*. 2021. URL: <https://root.cern.ch/>.
- [48] C.G. Ryan et al. “SNIP, a statistics-sensitive background treatment for the quantitative analysis of PIXE spectra in geoscience applications”. In: *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms* 34.3 (1988), pp. 396–402. ISSN: 0168-583X. DOI: [https://doi.org/10.1016/0168-583X\(88\)90063-8](https://doi.org/10.1016/0168-583X(88)90063-8). URL: <https://www.sciencedirect.com/science/article/pii/0168583X88900638>.
- [49] Miroslav Morháč et al. “Background elimination methods for multidimensional coincidence γ -ray spectra”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 401.1 (1997), pp. 113–132. ISSN: 0168-9002. DOI: [https://doi.org/10.1016/0168-9002\(97\)90001-0](https://doi.org/10.1016/0168-9002(97)90001-0).

10.1016/S0168-9002(97)01023-1. URL: <https://www.sciencedirect.com/science/article/pii/S0168900297010231>.

- [50] Donald D. Burgess and Richard J. Tervo. “Background estimation for gamma-ray spectrometry”. In: *Nuclear Instruments and Methods in Physics Research* 214.2 (1983), pp. 431–434. ISSN: 0167-5087. DOI: [https://doi.org/10.1016/0167-5087\(83\)90612-9](https://doi.org/10.1016/0167-5087(83)90612-9). URL: <https://www.sciencedirect.com/science/article/pii/0167508783906129>.
- [51] Miroslav Morháč et al. “Identification of peaks in multidimensional coincidence γ -ray spectra”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 443.1 (2000), pp. 108–125. ISSN: 0168-9002. DOI: [https://doi.org/10.1016/S0168-9002\(99\)01005-0](https://doi.org/10.1016/S0168-9002(99)01005-0). URL: <https://www.sciencedirect.com/science/article/pii/S0168900299010050>.
- [52] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: [10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
- [53] Matt Newville et al. *lmfit/lmfit-py 1.0.2*. Version 1.0.2. Feb. 2021. DOI: [10.5281/zenodo.4516651](https://doi.org/10.5281/zenodo.4516651). URL: <https://doi.org/10.5281/zenodo.4516651>.
- [54] Xavier Glorot and Y. Bengio. “Understanding the difficulty of training deep feed-forward neural networks”. In: *Journal of Machine Learning Research - Proceedings Track 9* (Jan. 2010), pp. 249–256.
- [55] Ekaba Bisong. “Google Colaboratory”. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 59–64. ISBN: 978-1-4842-4470-8. DOI: [10.1007/978-1-4842-4470-8_7](https://doi.org/10.1007/978-1-4842-4470-8_7). URL: https://doi.org/10.1007/978-1-4842-4470-8_7.
- [56] Andrea Dal Pozzolo et al. “Calibrating Probability with Undersampling for Unbalanced Classification”. In: Dec. 2015. DOI: [10.1109/SSCI.2015.733](https://doi.org/10.1109/SSCI.2015.733).
- [57] Ashraf A Aly, Safaai Bin Deris, and Nazar Zaki. “Research review for digital image segmentation techniques”. In: *Int. J. Comput. Sci. Inf. Technol. Res.* 3.5 (2011), p. 99.

- [58] Jim Pivarski et al. *scikit-hep/uproot4: 4.0.11*. Version 4.0.11. June 2021. DOI: 10.5281/zenodo.5047600. URL: <https://doi.org/10.5281/zenodo.5047600>.
- [59] Yassid Ayyad et al. “Novel particle tracking algorithm based on the Random Sample Consensus Model for the Active Target Time Projection Chamber (AT-TPC)”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 880 (2018), pp. 166 – 173. ISSN: 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2017.10.090>. URL: <http://www.sciencedirect.com/science/article/pii/S0168900217311798>.
- [60] P.H.S. Torr and A. Zisserman. “MLESAC: A New Robust Estimator with Application to Estimating Image Geometry”. In: *Computer Vision and Image Understanding* 78.1 (2000), pp. 138–156. ISSN: 1077-3142. DOI: <https://doi.org/10.1006/cviu.1999.0832>. URL: <http://www.sciencedirect.com/science/article/pii/S1077314299908329>.
- [61] *Point Cloud Library*. 2020. URL: <https://pointclouds.org/>.
- [62] J.C. Zamora and G.F. Fortino. “Tracking algorithms for TPCs using consensus-based robust estimators”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* (2020), p. 164899. ISSN: 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2020.164899>. URL: <http://www.sciencedirect.com/science/article/pii/S0168900220312961>.
- [63] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “Open3D: A Modern Library for 3D Data Processing”. In: *arXiv:1801.09847* (2018).
- [64] Peter J. Rousseeuw. “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53 –65. ISSN: 0377-0427. DOI: [https://doi.org/10.1016/0377-0427\(87\)90125-7](https://doi.org/10.1016/0377-0427(87)90125-7). URL: <http://www.sciencedirect.com/science/article/pii/0377042787901257>.
- [65] Leland McInnes, John Healy, and Steve Astels. “hdbscan: Hierarchical density based clustering”. In: *The Journal of Open Source Software* 2.11 (2017), p. 205.
- [66] Leland McInnes and John Healy. “Accelerated Hierarchical Density Based Clustering”. In: *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*. IEEE. 2017, pp. 33–42.

- [67] Christoph Dalitz, Jens Wilberg, and Lukas Aymans. “TriplClust: An Algorithm for Curve Detection in 3D Point Clouds”. In: *Image Processing On Line* 9 (2019). <https://doi.org/10.5201/ipol.2019.234>, pp. 26–46.
- [68] MP Kuchera et al. “LISE++ Software Updates and Future Plans”. In: *Journal of Physics: Conference Series* 664.7 (2015), p. 072029. DOI: 10.1088/1742-6596/664/7/072029. URL: <https://doi.org/10.1088/1742-6596/664/7/072029>.