Spis treści

1 Fizyka

1.1 Plazma kwarkowo-gluonowa

Plazma kwarkowo–gluonowa (ang. *quark-gluon plasma – QGP*) to stan materii, który istniał w pierwszych ułamkach sekund po Wielkim Wybuchu. Przewiduje się, że materia w takim stanie obecna jest także w jądrach gwiazd neutronowych [1].

Obecnie aby uzyskać dostęp do materii w stanie plazmy kwarkowo-gluonowej potrzebne są wysokoenergetyczne zderzenia cząstek. Powszechnie mówi się o niej w kontekście zderzeń ciężkich jonów, chociaż istnieją także prace doszukujące się obecności QGP w mniejszych systemach np. w zderzeniach proton-proton [2], [3].

QGP jest stanem o ekstremalnej gęstości i temperaturze. Jego cechą charakterystyczną jest obecność wolnych kwarków i gluonów (zbiorczo nazywanych partonami). W każdym innym stanie materii są one zawsze związane i tworzą hadrony. Zjawisko to, zwane uwięzieniem koloru (ang. *color confinement*) uniemożliwia obserwację cząstek obdarzonych ładunkiem kolorowym (a takimi są kwarki i gluony) w stanie wolnym. Wolne kwarki i gluony powstające w zderzeniach muszą zatem przejść przez proces hadronizacji, w którym rekombinują one ze spontanicznie wytwarzanymi nowymi partonami, tworząc hadrony. W wyniku tego procesu, z każdego partonu obecnego w początkowym etapie zderzenia może powstać wiele cząstek poruszających się podobnym kierunku, tworząc stożek z wierzchołkiem blisko punktu interakcji wiązek. Taki stożek skolimowanych cząstek nazywany jest dżetem cząstek.

1.2 Dżety

Przedstawiona powyżej definicja dżetu nie jest precyzyjna z punktu widzenia pracy eksperymentalnej. W detektorze obserwuje się tylko cząstki w stanie końcowym, nie jest znana natomiast ich historia (tj. parton, z którego powstały), nie jest zatem możliwe przyporządkowanie cząstki według jej pochodzenia. W związku z tym, konieczne jest użycie algorytmu klasteryzującego, dostającego na wejściu tylko obserwowalne eksperymentalnie cząstki. To jakie dżety zostaną zaobserwowane w danym zdarzeniu zależy od użytego algorytmu. Oznacza to, że precyzyjną definicję dżetu stanowi algorytm klasteryzujący wraz z zestawem parametrów. Obecnie najpowszechniej stosowanym algorytmem jest algorytm *anti-kt* [**Cacciari:2008gp**].

Eksperymentalne ograniczenia związane z obserwacją tylko końcowego stanu oddziaływań nie występują w analizie danych z symulacji Monte Carlo (MC), gdzie ma się dostęp do pełnej informacji na temat historii każdej cząstki. Można pomyśleć, że daje to możliwość lepszej klasteryzacji dżetów. Rekonstrukcja przy użyciu informacji o partonach - matkach sprawia jednak, że definicja dżetu wykorzystana w badaniach danych symulacyjnych jest inna od tej wykorzystywanej w eksperymencie. Tracona jest przez to cecha odpowiedniości między obiektami nazywanymi dżetami w symulacji i w eksperymencie, która to cecha jest niewątpliwie jedną z podstawowych wymagań stawianych przed dobrą symulacją.

Dżety stanowią ważny element w badaniach plazmy kwarkowo-gluonowej. Dają one pośredni wgląd we właściwości QGP na podstawie jej wpływu na oddziałujące z nią partony. Przykładową obserwablą mierzoną w zderzeniach ciężkich jonów jest czynnik modyfikacji jądrowej (ang. *nuclear modification factor*), który jest miarą strat energii przez parton przechodzący przez medium.

Oprócz globalnego wpływu medium na dżety, analizuje się także różnice między dżetami pochodzącymi z gluonów oraz kwarków o różnych zapachach (ang. *flavours*). Modele teoretyczne przewidują między innymi większe straty energii w wyniku interakcji z QGP dla dżetów gluonowych niż kwarkowych [4] oraz zależność strat energii od masy partonu [5] – w tym przypadku

precyzyjne pomiary rozróżniające typy dżetów pozwalają lepiej zrozumieć mechanizm odpowiadający za straty energii przez partony. Zagadnienie rozpoznania z jakiego rodzaju partonu powstał dany dżet, nazywane jest identyfikacją lub tagowaniem dżetu.

1.3 Identyfikacja dżetów *b*

Poza badaniami właściwości *QGP*, szczególne znaczenie ma identyfikacja dżetów pochodzących z ciężkich kwarków: *b* i *c*. Są one ważnym elementem w poszukiwaniu łamania symetrii *CP* w rozpadach hadronów B i D oraz innych sygnatur tzw. *Nowej Fizyki* wykraczającej poza ramy Modelu Standardowego. Kwarki *piękne* pojawiają się także często w kanałach rozpadu cząstek takich jak bozon Higgsa i kwark *t*.

Identyfikacja dżetów *b* jest sporym wyzwaniem ze względu na zdecydowanie częściej występujące dżety lekkie, tj. powstałe z hadronizacji kwarków *u,d,s* oraz gluonów. Rozpoznawanie dżetów *b* bazuje na charakterystycznych właściwościach hadronów zawierających kwark piękny: relatywnie długim czasie życia oraz (w mniejszym stopniu) na ich pół-leptonowym rozpadach o względnej częstości rozpadu w tym kanale (ang. *branching ratio*) na poziomie 10%.

przegląd algorytmów używanych w identyfikacji b-jetów: TBD

1.4 Eksperyment ALICE

TBD

2 Uczenie maszynowe

Uczenie maszynowe jest bardzo szerokim i obecnie dynamicznie się rozwijającym obszarem wiedzy. Występuje w wielu odmianach łącząc w sobie w zależności od wariantu wiele dziedzin takich jak matematyka (statystyka, algebra) informatyka (algorytmika, teoria informacji) a także elementy robotyki i sterowania. Dziedzinami, w których jest najczęściej wykorzystywane są min. widzenie maszynowe, przetwarzanie języka naturalnego, autonomiczne roboty i pojazdy, systemy decyzyjno - eksperckie, optymalizacyjne oraz rekomendacyjne.

W tej pracy wykorzystywana jest gałąź uczenia maszynowego nazywana uczeniem nadzorowanym lub "uczeniem z nauczycielem" (ang. *supervised learning*), gdzie uczenie występuje na podstawie poprawnie oznaczonych przykładów. Terminami bliskoznacznymi dla tak rozumianego uczenia maszynowego są uczenie statystyczne (ang. *statistical learning*) i rozpoznawanie wzorców (ang. *patterm recognition*).

Problem identyfikacji dżetów jest klasycznym przykładem zagadnienia klasyfikacji, gdzie poprawna odpowiedź jest jedną ze skończonej ilości opcji (klas) w przeciwieństwie do regresji, gdzie szukana odpowiedź algorytmu ma charakter ciągły.

Występuje wiele algorytmów uczenia maszynowego takich jak regresja liniowa i logistyczna, drzewa decyzyjne i ich wariacje, maszyny wektorów wspierających, sztuczne sieci neuronowe oraz wiele innych. Uczenie polega na znalezieniu pewnej funkcji dopasowującej do przyjmowanego na wejściu zestawu (wektora) cech (zmiennych, kolumn) pewną odpowiedź (predykcję), która minimalizuje zadaną funkcję straty. Jej rolę w przypadku regresji często pełni błąd średniokwadratowy a w przypadku klasyfikacji np. entropia krzyżowa (ang. *cross entropy*) [1]. Różne algorytmy szukają przy tym funkcji dopasowującej należącej do różnych klas funkcji: przykładowo klasyczne drzewa decyzyjne przeszukują tylko przestrzeń funkcji dających się opisać skończonym zbiorem reguł "jeśli – to" (ang. *if – else*).

W pracy wykorzystane zostały dwa rodzaje algorytmów: wzmacniane drzewa decyzyjne oraz sieci neuronowe.

2.1 Wzmacniane drzewa decyzyjne

Wzmacniane drzewa decyzyjne są jednym z rozwinięć klasycznego algorytmu drzewa decyzyjnego. Pojedyncze drzewo decyzyjne dzieli przestrzeń cech uczących przy pomocy prostopadłych cięć, na mniejsze/większe niż zadana wartość w przypadku zmiennej ciągłej lub na należące/nie należące do danej klasy w przypadku zmiennej kategorycznej. Każdy podział, nazywany węzłem, daje dwie gałęzie, które można dalej niezależnie dzielić aż do ostatniego poziomu (liści). Kolejne podziały wybierane są tak, aby zbiory przykładów wpadające do poszczególnych gałęzi były jak najbardziej jednorodne. Stosuje się różne miary nieporządku takie tak: *indeks Gini* lub entropia.

Drzewa decyzyjne są często łączone w komitety klasyfikatorów (ang. *ensemble methods*). Wiele "słabych" klasyfikatorów jest łączonych w jeden "silny" na dwa sposoby: *bagging* oraz *boosting* (wzmacnianie), które są często ze sobą porównywane.

Bagging – w zastosowaniu dla drzew decyzyjnych nazywany algorytmem lasów losowych (ang. *random forest*) - polega na wytrenowaniu wielu drzew, każdego na podstawie N przykładów losowo wylosowanych z powtórzeniami spośród N-licznego zbioru treningowego. Dodatkowo, do uczenia każdego drzewa używa się tylko podzbioru wszystkich cech uczących. Końcową predykcję algorytmu otrzymuje się poprzez "głosowanie" wszystkich drzew z odpowiednimi

[1] $J = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)]$, gdzie y_i to prawidłowa klasa i-tego przykładu a $\hat{y}_i$ to predykcja algorytmu

wagami.

Boosting – wzmacniane drzewa decyzyjne (ang. *boosted decision trees*) – jest metodą podobną do *baggingu*. Główną różnicą jest zwiększanie wag przykładom uczącym, które przez poprzednie drzewo zostały źle zaklasyfikowane – każde kolejne drzewo koncentruje się bardziej na poprawie błędów poprzednich drzew. Widać tu kolejną ważną cechę odróżniającą obie metody: *boosting* jest algorytmem sekwencyjnym podczas gdy *bagging* daje się trywialnie zrównoleglić (każde drzewo trenowane jest w osobnym wątku).

Parametry i sposób trenowania drzew decyzyjnych na analizowanych danych

W niniejszej pracy wykorzystano wzmacniane drzewa decyzyjne zaimplementowane w wydajnej bibliotece `XGBoost` [6]. Szybkość obliczeń jest bardzo ważna, gdyż oprócz komfortu pracy z algorytmem, przekłada się na jakość otrzymanych wyników – krótszy czas obliczeń oznacza możliwość przeprowadzenia większej ilości eksperymentów i lepsze dobranie parametrów oraz danych. Implementacja wzmacnianych drzew decyzyjnych w `XGBoost` wykorzystuje wszystkie rdzenie procesora, pomimo że sam algorytm ma charakter sekwencyjny – jest to możliwe dzięki paralelizacji procesu tworzenia każdego drzewa (przed każdym podziałem konieczne jest sprawdzenie pewnej ilości możliwych zmiennych i wartości progowych i ten proces jest wykonywany równolegle).

Dzięki szybkiemu uczeniu się algorytmu, możliwe było użycie kosztownego obliczeniowo automatycznego przeszukiwania przestrzeni parametrów przy pomocy przeszukiwania losowego (ang. *random search*), które jest zwykle preferowane nad przeszukiwanie sieciowe [7]. W tym celu cały zbiór danych dzielony był na dwie części: trenującą oraz testową (80/20%). Następnie algorytm był trenowany i oceniany z użyciem trzy- lub pięciokrotnej walidacji krzyżowej (ang. *cross-validation*) na zbiorze trenującym dla różnych zestawów parametrów. Model z najlepszym wynikiem uzyskanym w walidacji krzyżowej był sprawdzany na zbiorze testowym.

Parametry optymalizowane w opisanym procesie to:

- *max_ depth* – maksymalna głębokość każdego drzewa (niekoniecznie osiągana)
- *n_ estimators* – liczba drzew
- *learning_ rate* – parametr szybkości uczenia, komplementarny do *n_ estimators*, w praktyce można ustalić liczbę drzew i szukać optymalnej szybkości uczenia
- *subsample, colsample_ bytree, colsample_ bylevel* – parametry regularyzacyjne określające ułamek kolejno: danych użytych do trenowania każdego drzewa, kolumn użytych w każdym drzewie (cechy losowane raz dla danego drzewa), kolumn użytych przy każdym podziale (cechy losowane przy każdym podziale)
- γ – minimalny zysk w postaci zmniejszenia wartości funkcji straty konieczny do wykonania podziału

2.2 Sieci neuronowe

Sieci neuronowe (ang. *neural networks – NN*) są szczególnym algorytmem uczenia maszynowego. Występują w bardzo wielu odmianach i są wykorzystywane w rozwiązywaniu szerokiej gamy problemów. Nawet bardzo pobieżny opis sieci neuronowych wymaga dużo więcej miejsca niż może być temu poświęcone w tej pracy.

Wprowadzenia do sieci neuronowych na różnym poziomie zaawansowania można znaleźć min w [#REF]x100. Tu przedstawione zostaną wyłącznie wybrane zagadnienia mające ściślejszy

związek z pracą, używane mogą być terminy, których znaczenie wyjaśniane jest w podanych źródłach.

W niniejszej pracy, wykorzystane zostały dwa rodzaje sieci neuronowych: sieci w pełni połączone (ang. *fully connected NN – FC NN*), nazywane także wielowarstwowymi perceptronami (ang. *multi-layer perceptron – MLP*) oraz sieci konwolucyjne (ang. *convolutional NN – ConvNets, CNN*).

Sieci w pełni połączone

W nierekurencyjnych sieciach neuronowych (tylko takie są używane w tej pracy), informacja jest przekazywana kolejno od warstw wejściowych, poprzez warstwy ukryte aż do wyjściowej. W sieciach typu *FC* wszystkie warstwy składają się z identycznych neuronów – każdy neuron dostaje na wejściu wektor, natomiast zwraca skalar – wartość pewnej zadanej, nieliniowej funkcji, jako argument podając średnią ważoną z elementów wektora wejściowego. Wartości zwracane przez neurony w danej warstwie składają się na wektor wejściowy dla neuronów kolejnej warstwy. Wejściem dla pierwszej warstwy są natomiast kolejne przykłady ze zbioru uczącego. Trenowanie sieci neuronowych polega na zmienianiu wag (parametrów) w liczonej w każdym neuronie średniej, każdy neuron posiada własny, niezależny zestaw wag.

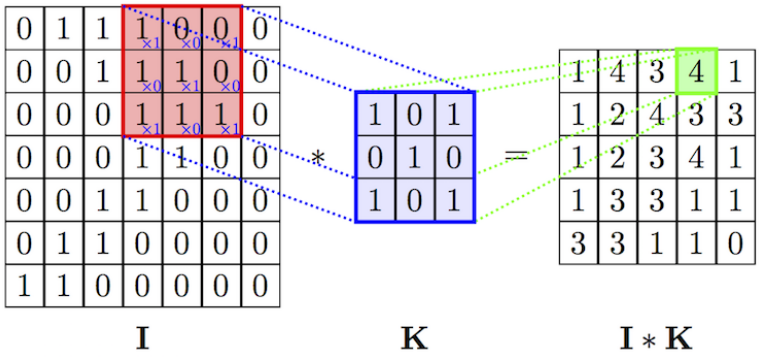
Istnieje twierdzenie o sieciach neuronowych jako uniwersalnych aproksymatorach funkcji (ang. *Universal approximation theorem*) [8], mówiące, że już sieć neuronowa o jednej warstwie ukrytej jest zdolna do przybliżenia dowolnej funkcji z dowolną dokładnością. Twierdzenie to milczy niestety na temat liczby potrzebnych neuronów a przede wszystkim sposobu ich trenowania. Trenowanie jest prostsze w przypadku zastosowania wielu warstw, które odpowiadają kolejnym poziomą abstrakcji jednak nadal jest dużym wyzwaniem ze względu na fakt, że nawet stosunkowo niewielka sieć może posiadać bardzo dużą liczbę parametrów, przykładowo sieć o czterech warstwach, w każdej po 128 neuronów ma ich ponad 65 tysięcy.

Sieci konwolucyjne

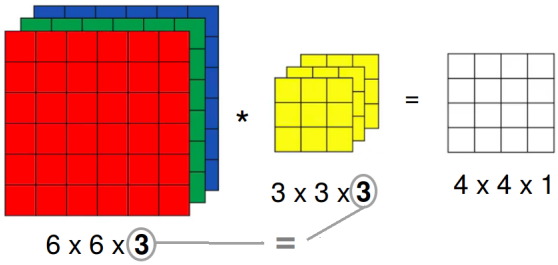
Jednym ze sposobów na ograniczenie liczby trenowanych parametrów jest użycie konwolucyjnych sieci neuronowych (bardziej poprawną choć rzadko używaną nazwą w języku polskim jest sieć splotowa). Są one inspirowane połączeniami w korze wzrokowej zwierząt i wywodzą się z badań w obszarze wizji komputerowej, gdzie liczby parametrów są szczególnie duże (wektor wejściowy ma wymiar równy liczbie pikseli w obrazku), na takim przykładzie również najłatwiej zrozumieć ich działanie.

Sieci konwolucyjne różnią się od sieci typu *FC*, tym że część wag połączeń między warstwami jest dzielona. Występuje w nich nowy rodzaj warstwy, nazywany warstwą konwolucyjną. Każda jednostka w warstwie konwolucyjnej (filtr) ma pewną stałą (niewielką) liczbę wag. Połączenie z dużym wejściem realizowane jest przez powielanie tych samych wag w połączeniach z kolejnymi fragmentami wektora wejściowego (por. Rys. 1). Rezultatem działania filtra na macierz jest wynik operacji splotu. Liczba parametrów przypadająca na każdy filtr jest równa jego rozmiarowi i nie zależy od wielkości wektora wejściowego.

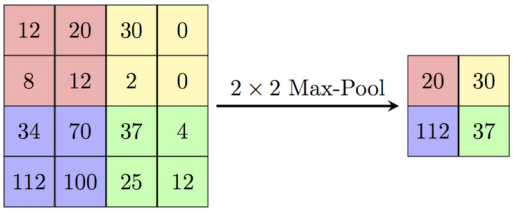
W przypadku gdy zamiast wejścia dwuwymiarowego (jak np. obraz czarno-biały), mamy do czynienia z wejściem trójwymiarowym (np. trzeci wymiar to kolejne kolory w kodowaniu RGB), filtry również muszą mieć trzy wymiary, przy czym rozmiar w ostatnim wymiarze musi być równy rozmiarowi w tym kierunku wektora wejściowego. Wynik operacji splotu jest ponownie dwuwymiarowy, gdyż filtr przesuwany jest tylko w dwóch pierwszych wymiarach. Trzeci wymiar powstaje przez składanie kolejnych filtrów. Widać zatem, że również w przypadku gdy na wejścia podawany jest obraz czarno-biały, filtry w kolejnych warstwach konwolucyjnych (oprócz pierwszej) mają po trzy wymiary.



Rysunek 1: Schemat działania pojedynczego filtra z warstwy konwolucyjnej (operacja splotu).



Rysunek 2: Działanie pojedynczego filtra (3D) na wejście o trzech wymiarach.

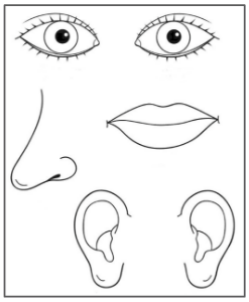


Rysunek 3: Działanie warstwy typu *max-pool*

Oprócz warstw konwolucyjnych, w sieciach tego typu stosowane są także tzw. warstwy typu *max-pooling*. Zasada jej działania jest bardzo prosta: wykonuje funkcję *maksimum* na zadanym fragmencie obrazu (por. Rys. 3). Ich rolą jest zmniejszanie rozmiaru przekazywanej w sieci informacji.

Typowa architektura stosowana w przypadku sieci konwolucyjnych jest następująca: najpierw warstwy konwolucyjne (pomiędzy nimi czasem warstwy typu *max-pool*), następnie wszystkie filtry są rozwijane i składane w długi jednowymiarowy wektor, który przekazywany jest do warstw typu *FC*. W przypadku problemu klasyfikacji, na końcu znajduje się jeszcze warstwa typu *softmax* normalizująca wyjście z sieci do jedynki.

Sieci konwolucyjne posiadają dwie właściwości odróżniające je od *MLP*:

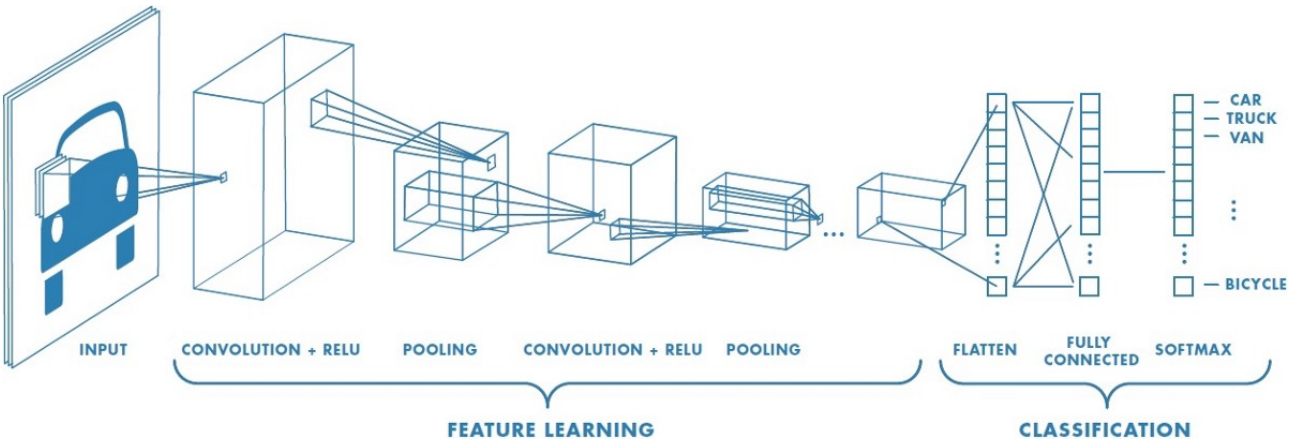


- niezmienniczość względem przesunięcia (ang. *translation invariance*) – głównie za sprawą dzielenia wag oraz obecności warstw typu *max-pool*, położenie danej cechy na obrazie jest niemal bez znaczenia (obraz po prawej stronie byłby rozpoznany jako twarz)
- lokalność połączeń – filtry obejmują tylko kilka sąsiednich pikseli (tam zwykle występują najsilniejsze zależności) nie są w stanie dostrzec cechy rozciągniętej na obszar większy od rozmiaru filtra

Hiperparametry i trenowanie sieci neuronowych na analizowanych danych

Parametry sieci, których wartości są określane przez projektanta sieci, takie jak liczba warstw ukrytych są nazywane hiperparametrami (dla odróżnienia od parametrów - wag połączeń).

Testowane były trzy architektury: sieci w pełni połączone, sieci konwolucyjne oraz sieć złożona z dwóch gałęzi, osobnych dla wtórnych wierzchołków i cząstek tworzących dżet (por.



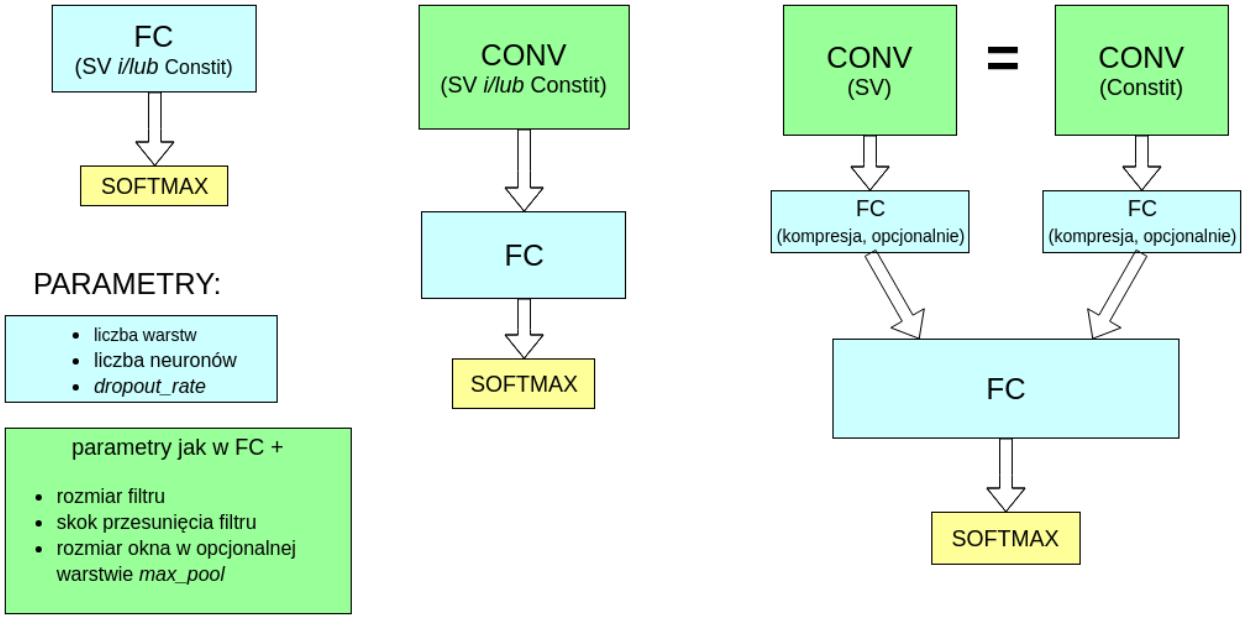
Rysunek 4: Typowa struktura stosowana w sieciach konwolucyjnych. Warstwy konwolucyjne mają za zadanie wydobywać cechy, na podstawie których późniejsze warstwy dokonują klasyfikacji. Widoczna jest charakterystyczna stopniowa zmiana rozmiaru przekazywanej macierzy: rozmiar poprzeczny maleje kosztem głębokości, co odpowiada rosnącej liczbie filtrów i malejącemu rozmiarowi obszaru po jakim są one przesuwane.

Rozdz. 3) przedstawione schematycznie na Rys. 5. Do trenowania sieci wykorzystano wysokopoziomową bibliotekę `Keras` [9] korzystającą z silnika obliczeniowego zaimplementowanego w `TensorFlow` [10].

Zestaw hiperparametrów definiujący działanie sieci w pełni połączonej:

- liczba warstw ukrytych
- liczba neuronów w każdej warstwie
- funkcja aktywacji – nieliniowa funkcja aplikowana przed zwróceniem wartości w każdym neuronie, najpopularniejsze to *tanh*, $ReLU$ oraz funkcja sigmoidalna
- algorytm optymalizacyjny – spadek gradientowy lub jego wariacje
- parametr szybkości uczenia i jego modyfikacje w trakcie uczenia
- liczba przykładów trenujących przetwarzanych w jednym kroku uczenia (ang. *batch_size*) – im większy tym szybsze jest trenowanie sieci (dzięki wydajnym operacjom macierzowym), natomiast może się to odbywać kosztem precyzji
- liczba epok uczenia – ile razy będzie pokazywany sieci każdy przykład
- opcjonalnie: warunki stopu (ang. *early stopping*) w razie osiągnięcia *plateau*
- opcjonalnie: regularyzacja przy pomocy różnych technik (zwykle konieczna)
- ponadto dla sieci konwolucyjnych:
  - liczba warstw konwolucyjnych i liczba filtrów w tychże
  - obecność lub brak warstw *max-pool* i rozmiar ich okna
  - rozmiar filtrów i długość skoku przy ich przesuwaniu

Same dwie pierwsze wielkości dają nieograniczoną liczbę konfiguracji. Czas trenowania sieci neuronowych jest rzędy wielkości większy niż drzew decyzyjnych, dlatego przyjęto szereg kroków mających na celu zmniejszenie przeszukiwanej przestrzeni hiperparametrów. Na podstawie wstępnych testów oraz różnych wskazówek dostępnych w literaturze przyjęto:



Rysunek 5: Schematyczne przedstawienie trzech testowanych rodzin architektur sieci. Każdy blok odpowiada kilku warstwom danego typu. Bloki warstw typu *FC* i opisane jako "kompresja" składały się z kilku neuronów i miały za zadanie zredukować całą informację z danej gałęzi do wektora kilku liczb. Obie gałęzie miały taką samą strukturę.

- *batch_ size* zawsze równy 64 (inne testowane wartości: 16, 32, 128)
- za algorytm optymalizacyjny przyjęto algorytm o nazwie *Nadam*, tj. rozwinięcie algorytmu *Adam* o parametr Nesterova (inne testowane to zwykły spadek gradientowy oraz *Adam*)
- funkcję aktywacji: $ReLU$
- liczba epok równa 100 lub 200 (lub mniej do testów), zrezygnowano z *early stopping*
- stałe w trakcie treningu wartości parametru szybkości uczenia
- spośród technik regularyzacyjnych testowano wyłącznie *dropout* [11] z parametrami 0.1, 0.2, 0.5
- kilka wybranych kombinacji dla zestawu parametrów: rozmiar filtra, długość skoku i rozmiar okna w warstwach *max-pool* – takie same w kolejnych warstwach
- liczby neuronów/filtrów w warstwach będące zawsze potęgami dwójki oraz stałą liczbę w kolejnych warstwach lub zmieniającą się o stały czynnik, np. 256-128-64, 128-128-128 lub 16-32-64
- liczba warstw *FC*: 2-8, konwolucyjnych 2-6

Nawet po przyjęciu powyższych uproszczeń nie sposób sprawdzić wszystkich możliwych zestawów hiperparametrów, dlatego sposób ich dobierania w kolejnych testach był mocno empiryczny. Dostępne dane dzielone były na trzy zbiory: trenujący, walidacyjny i testowy. Wobec braku warunków stopu, zbiór walidacyjny użyty był wyłącznie do porównywania różnych zestawów parametrów, tak aby wynik testowy pozostał nieobciążony.

Zgodnie z zasadą ortogonalizacji działań, proces doboru hiperparametrów dzielono na dwie części: najpierw starano się uzyskać jak najlepsze wyniki na zbiorze uczącym, a dopiero później

zmusić algorytm do lepszej generalizacji na zbiorze testowym przez zwiększoną regularyzację i modyfikację parametru szybkości uczenia.

2.3 Dyskusja użycia dwóch algorytmów

Użycie więcej niż jednego algorytmu ma wiele zalet. Po pierwsze daje możliwość porównania wyników. Pozwala to na pewne oszacowanie błędu *Bayesowskiego* (najniższego możliwego do osiągnięcia przez jakikolwiek algorytm błędu). Jest to bardzo ważne w sytuacji, gdy nie dysponuje się innym oszacowaniem tego błędu (w wielu problemach naturalnych dla człowieka jak rozpoznawanie obiektów na obrazkach jest nim błąd ludzki lub też błąd popełniany przez zespół ekspertów w bardziej zaawansowanych zastosowaniach).

Po drugie, wykorzystane zostały dwa algorytmy mocno różniące się w swojej naturze, co pozwala wykorzystać cechy każdego z nich w analizie: przykładowo sieci neuronowe dobrze radzą sobie z nieustrukturyzowanymi danymi – potrafią tworzyć wysoko poziomowe cechy na podstawie niskopoziomowego wejścia (np. położenia oka na zdjęciu twarzy na podstawie pixeli). Są natomiast trudne w interpretacji i często traktowane są jako tzw. "czarne skrzynki" (ang. *black box*). Oprócz tego, liczba możliwych konfiguracji sieci jest ogromna i przez to niemożliwe jest stwierdzenie czy wykorzystane zostały pełne ich możliwości.

Z kolei drzewa decyzyjne posiadają stosunkowo niewielką liczbę parametrów, a ich trenowanie jest bardzo szybkie co pozwala na ich ekstensywne przeszukiwanie i otrzymanie wyników, które można uznać za optymalne dla tego algorytmu. Ponadto, w przypadku drzew istnieją niewymagające dodatkowych obliczeń miary użyteczności poszczególnych zmiennych, co daje wgląd w działanie algorytmu i poprawia intuicyjne zrozumienie jego predykcji.

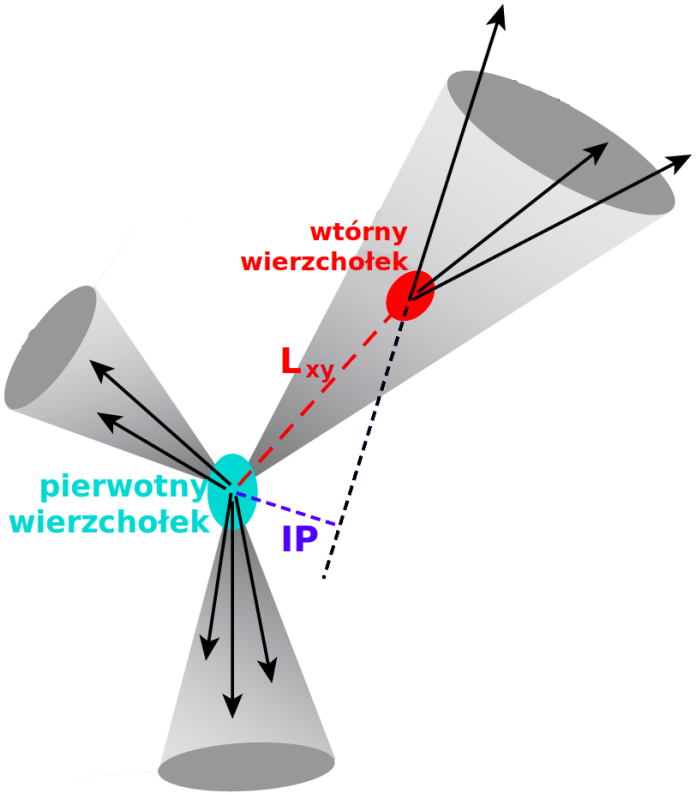
3 Dane

Dane użyte w analizie pochodzą z symulacji Monte Carlo zderzeń proton-proton przy energii w układzie środka masy równej $\sqrt{s} = 13$ TeV dostępnych na serwerach eksperymentu ALICE. W symulacjach zderzeń użyty był symulator `Pythia8` (*tune: Pythia8Jets_Monash2013*), następnie przy pomocy `Geant3` symulowany był transport cząstek i odpowiedź detektora ALICE.

Do rekonstrukcji dżetów wykorzystany został algorytm *anti-kt* z parametrem $R = 0.4$ zaimplementowany w pakiecie `FASTJET`. Dżetów poszukiwano tylko wyłącznie wśród cząstek naładowanych (ang. *charged jets*).

Do analizy wybrano dżety o p_T większym niż 15 GeV i mieszczących się w całości w akceptancji detektora *TPC*, tj. $|\eta| < 0.9$, co przy użytym parametrze rozmiaru dżetu $R = 0.4$, daje ograniczenie na pseudopospieszność $|\eta| < 0.5$ dla osi dżetu.

Dla każdego dżetu obliczony został szereg wielkości, które można podzielić na zmienne na poziomie dżetu, związane z wtórnymi wierzchołkami oraz cząstkami tworzącymi dżet. Za potencjalne wtórne wierzchołki uznaje się wszystkie kombinacje trzech cząstek spełniających pewne dosyć luźne wymagania jak $p_T > 1GeV$ (rozważane są wyłącznie trzy-cząstkowe wtórne wierzchołki), stąd ich liczba może być dużo większa od liczby cząstek tworzących dżet.



Rysunek 6: Rysunek ilustrujący znaczenie używanych wielkości: L_{xy} oraz parametru zderzenia IP.

Lista używanych zmiennych:

- Zmienne na poziomie dżetu:
  - η, ϕ – pseudopospieszność (ang. *pseudorapidity*) i kąt azymutalny
  - p_T – pęd poprzeczny dżetu

- masa dżetu
- powierzchnia dżetu – liczona w płaszczyźnie (η, ϕ), do powierzchni dżetu zaliczany jest element w powierzchni w którym dodanie cząstki o nieskończenie małym pędzie poprzecznym sprawi, że zostanie ona zaliczona do tego dżetu [#REF https://arxiv.org/pdf/0707
- gęstość tła (w danym zdarzeniu)
- N_{SV} – liczba wtórych wierzchołków
- $N_{Constit}$ – liczba cząstek tworzących dżet

- Zmienne opisujące cząstki tworzące dżet:
  - η, ϕ – pseudopospieszność i kąt azymutalny cząstki względem osi dżetu
  - p_T – pęd poprzeczny cząstki
  - IP_D – rzut na kierunek poprzeczny wektora parametru zderzenia
  - IP_Z – rzut na oś z wektora parametru zderzenia

- Zmienne opisujące wtórne wierzchołki:
  - L_{xy} – odległość między pierwotnym a wtórnym wierzchołkiem (ang. *decay length*)
  - σ_{Lxy} – niepewność wyznaczenia L_{xy}
  - $\sigma_{vertex} = \sqrt{d_1^2 + d_2^2 + d_3^2}$ – rozrzut śladów (ang. *tracks*) wokół wtórnego wierzchołka, gdzie d_i to odległość najbliższego zbliżenia śladu / odległość najbliższego przelotu do wtórnego wierzchołka (ang. *distance of closest approach – DCA*)
  - $M_{inv} = \sqrt{(E_1 + E_2 + E_3)^2 - (\vec{p_1} + \vec{p_2} + \vec{p_3})^2}$ – masa niezmiennicza wierzchołka, gdzie E_i, p_i to energia i pęd i-tej cząstki tworzącej wierzchołek
  - χ^2/Ndf dopasowania wtórnego wierzchołka

TBD: tabelaryczna postać, liczba SV i Nconstit, jak posortowane, wypleniania zerami korespondencja miedzy uzytymi danymi a algorytmami z sec. "identyfikacja dżetów b"

4 Analiza

TBD

References

[1] Anton Andronic. "An overview of the experimental study of quark-gluon matter in high-energy nucleus-nucleus collisions". In: *Int. J. Mod. Phys.* A29 (2014), p. 1430047. DOI: 10.1142/S0217751X14300476. arXiv: 1407.5003 [nucl-ex].

[2] Vardan Khachatryan et al. "Evidence for collectivity in pp collisions at the LHC". In: *Phys. Lett.* B765 (2017), pp. 193–220. DOI: 10.1016/j.physletb.2016.12.009. arXiv: 1606.06198 [nucl-ex].

[3] Jaroslav Adam et al. "Enhanced production of multi-strange hadrons in high-multiplicity proton-proton collisions". In: *Nature Phys.* 13 (2017), pp. 535–539. DOI: 10.1038/nphys4111. arXiv: 1606.07424 [nucl-ex].

[4] Carlos A. Salgado and Urs Achim Wiedemann. "Calculating quenching weights". In: *Phys. Rev.* D68 (2003), p. 014008. DOI: 10.1103/PhysRevD.68.014008. arXiv: hep-ph/0302184 [hep-ph].

[5] Yuri L. Dokshitzer and D. E. Kharzeev. "Heavy quark colorimetry of QCD matter". In: *Phys. Lett.* B519 (2001), pp. 199–206. DOI: 10.1016/S0370-2693(01)01130-3. arXiv: hep-ph/0106202 [hep-ph].

[6] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[7] James Bergstra and Yoshua Bengio. "Random Search for Hyper-parameter Optimization". In: *J. Mach. Learn. Res.* 13 (Feb. 2012), pp. 281–305. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2188385.2188395.

[8] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4.2 (1991), pp. 251 –257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T.

[9] François Chollet et al. *Keras*. https://keras.io. 2015.

[10] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[11] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2627435.2670313.