Sieci neuronowe w R i Python

Plan prezentacji:

1. Implementacja sieci neuronowych w R: Omówienie pakietów R, które umożliwiają implementację sieci neuronowych, takich jak nnet, neuralnet, caret itp. Przedstawienie sposobów tworzenia sieci neuronowych w R i wizualizacji wyników.
2. Implementacja sieci neuronowych w Python: Przedstawienie popularnych bibliotek Python, które umożliwiają tworzenie sieci neuronowych, takich jak TensorFlow, Keras, PyTorch itp. Omówienie sposobów tworzenia sieci neuronowych w Python i wizualizacji wyników.
3. Porównanie implementacji w R i Python: Porównanie możliwości, wydajności, składni i narzędzi dostępnych w pakietach R i bibliotekach Python do implementacji sieci neuronowych.
4. Optymalizacja i strojenie sieci neuronowych: Przedstawienie technik optymalizacji sieci neuronowych, takich jak regularyzacja, normalizacja wsadowa, dropout, inicjalizacja wag itp., które pozwalają na poprawę wydajności i zmniejszenie ryzyka przeuczenia.
5. Modele uczenia z nadzorem i bez nadzoru: Przedstawienie różnic między modelem uczenia z nadzorem i bez nadzoru, omówienie popularnych algorytmów uczenia bez nadzoru, takich jak autoenkoder i sieci GAN.

Ad.1

R to język programowania i środowisko obliczeniowe, które oferuje wiele pakietów i bibliotek do uczenia maszynowego i analizy danych. Istnieje kilka pakietów w R, które umożliwiają implementację sieci neuronowych, takich jak:

1. nnet: Pakiet nnet jest jednym z najstarszych i najpopularniejszych pakietów R do uczenia sieci neuronowych. Oferuje on funkcje do tworzenia sieci neuronowych wielowarstwowych i jednowarstwowych z różnymi funkcjami aktywacji, propagacją wsteczną i kontrolą przeuczenia.
2. neuralnet: Pakiet neuralnet oferuje bardziej zaawansowane funkcje do uczenia sieci neuronowych, takie jak nieliniowe funkcje aktywacji, regularyzacja, normalizacja wsadowa, algorytm Levenberga-Marquardta itp. Ten pakiet umożliwia również wizualizację sieci neuronowych.
3. caret: Pakiet caret oferuje narzędzia do budowania różnych modeli uczenia maszynowego, w tym sieci neuronowych. Ten pakiet umożliwia automatyczne strojenie hiperparametrów, selekcję cech i tworzenie modeli ensemble.

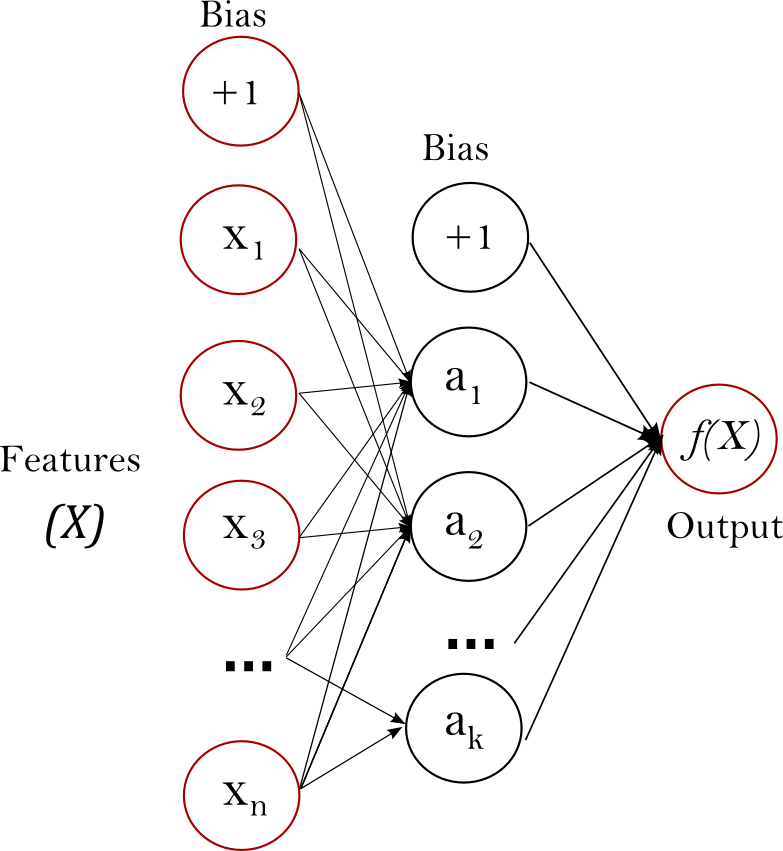
Aby zacząć pracować z sieciami neuronowymi w R, należy najpierw zainstalować wybrany pakiet. Można to zrobić za pomocą funkcji install.packages(). Następnie, po instalacji, należy zaimportować pakiet do środowiska R za pomocą funkcji library(). Po zaimportowaniu pakietu można utworzyć sieć neuronową, określić liczbę warstw, funkcje aktywacji, wagi i uczyć sieć za pomocą funkcji nnet() lub neuralnet().

Sieci neuronowe w języku Python

1. Multi-layer Perceptron

Multi-layer Perceptron (MLP) to algorytm uczenia nadzorowanego, który uczy się funkcji  
 poprzez trenowanie na zbiorze danych, gdzie jest liczbą wymiarów dla wejścia, a jest liczbą wymiarów dla wyjścia. Dla danego zestawu cech i celu , może nauczyć się aproksymatora funkcji nieliniowej, zarówno dla klasyfikacji, jak i regresji. Różni się on od regresji logistycznej tym, że pomiędzy warstwą wejściową a wyjściową może wystąpić jedna lub więcej warstw nieliniowych, nazywanych warstwami ukrytymi.

Rysunek 1 przedstawia MLP z jedną warstwą ukrytą i skalarnym wyjściem.



Lewa warstwa, zwana warstwą wejściową, składa się z zestawu neuronów {} reprezentujących cechy wejściowe. Każdy neuron w warstwie ukrytej przetwarza wartości z poprzedniej warstwy za pomocą ważonej sumy liniowej a następnie funkcji aktywacji nieliniowej , takiej jak funkcja tangens hiperboliczny. Warstwa wyjściowa otrzymuje wartości z ostatniej warstwy ukrytej i przetwarza je na wartości wyjściowe.

Moduł zawiera publiczne atrybuty coefs\_ i intercepts\_.  
coefs\_ to lista macierzy wag, gdzie macierz wag pod indeksem reprezentuje wagi między warstwą oraz warstwą .  
intercepts\_ to lista wektorów obciążenia, gdzie wektor pod indeksem reprezentuje wartości obciążenia dodawane do warstwy.

Zalety MLP:

* Możliwość nauki nieliniowych modeli.
* Możliwość nauki modeli w czasie rzeczywistym (uczenie online) za pomocą partial\_fit.

Wady MLP:

* MLP z warstwami ukrytymi ma nieliniową funkcję straty, gdzie istnieje więcej niż jedno lokalne minimum. Dlatego różne losowe inicjalizacje wag mogą prowadzić do różnych dokładności walidacji.
* MLP wymaga dopasowania wielu hiperparametrów, takich jak liczba neuronów ukrytych, warstwy i iteracje.
* MLP jest wrażliwy na skalowanie cech.

Utworzenie MLP za pomocą sklearn.neural\_network