**3. ADALINE**

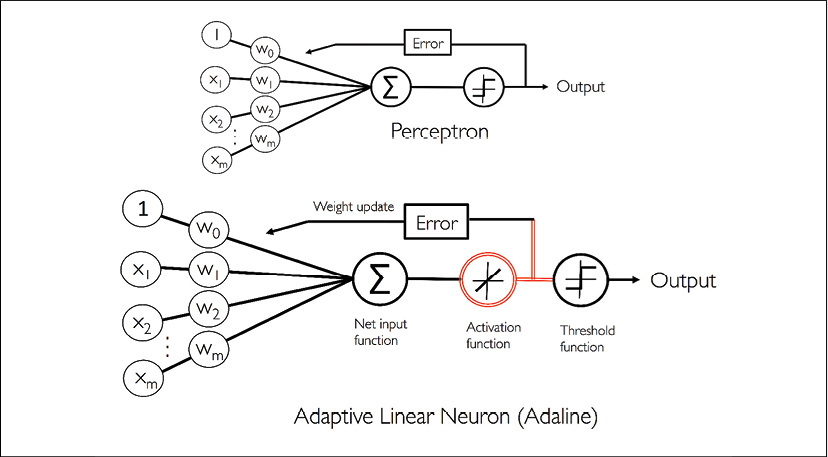
**3.1 Inspirace a historický kontext**

Adaptivní lineární neuron (ADALINE) je podstatně vylepšená verze perceptronu. Stále se jedná o jednovrstvou síť, avšak proces učení je daleko efektivnější a přesnější. ADALINE poprvé představil prof. Bernard Widrow a jeho student Ted Hoff ze Stanfordské univerzity. Jako podklad sloužil právě model McCulloch-Pitts neuronu, na základě kterého navrhl Rosenblatt perceptronový algoritmus.

V této kapitole opět důsledně využíváme maticový zápis formulí. Tedy zvýrazněný font znamená že pracujeme s vektorovou, případně maticovou (zejména matice vah ) veličinou.

**3.2 Lineární aktivační funkce**

Hlavním rozdílem mezi ADALINE a perceptronem (viz obr. 3.1) je způsob, jakým probíhá učení (optimalizace vah). Zatímco u perceptronu jsou váhové koeficienty aktualizovány podle rozdílu mezi cílovou a predikovanou třídou, ADALINE nejdříve zmenší jistou ztrátovou funkci úpravou vah pomocí vhodného algoritmu, a teprve poté vytváří predikci.6 Kromě skokové funkce kvůli predikci využívá navíc *lineární aktivační funkci*, která je prostou identickou funkcí lineárního vstupu1:



Obr. 3.1 Porovnání schémat perceptronu a ADALINE

**3.3 Ztrátová funkce**

Ztrátová nebo *účelová funkce* (také *loss* nebo *cost*) je jednou z hlavních komponent všech neuronových sítí. Zjednodušeně se jedná o funkci určující ztrátové skóre*,* které představuje momentální míru chybovosti sítě na základě hodnot váhových koeficientů. Během učení se snažíme ztrátové skóre minimalizovat a docílit tak co nejlepších predikcí. V případě ADALINE definujeme ztrátovou funkci jako *sumu kvadratických odchylek* (*sum of squared errors*) SSE, kde je cílová třída příkladu a aplikace lineární aktivační funkce na lineární vstup :

Použití lineární aktivační funkce namísto skokové je výhodné hlavně z toho důvodu, že ztrátová funkce je nyní diferencovatelná (má v každém bodě derivaci2)a konvexní (má globální minimum)1, což je důležité pro použití matematických metod při hledání minim ztrátových funkcí (viz gradientní sestup níže) – rozdíl mezi konvexní a konkávní funkcí je vidět na obrázku 3.2.

Chart, line chart

Description automatically generated

Obr. 3.2 Příklady konvexní a konkávní funkce

**3.4 Gradientní sestup**

Algoritmus *gradientního sestupu* (*gradient descent*) GDje velmi efektivní a populární způsob optimalizace váhových koeficientů. Používá se nejen v oblasti hlubokého učení, ale také v běžných algoritmech strojového učení (například v lineární regresi). Jeho cílem je minimalizovat ztrátové skóre a nalézt tak co nejvýhodnější hodnoty vah. Abychom mohli na funkci aplikovat gradientní sestup, musí být diferencovatelná a konvexní (výše definovaná SSE obě podmínky splňuje).7 Optimalizační algoritmuspoté upravuje váhové koeficienty v opačném směru gradientu ztrátové funkce (viz obr. 3.4), protože gradient určuje směr nejrychlejšího růstu ztrátové funkce – viz dále 3.5. Snaží se tak co nejvíce přiblížit jejímu globálnímu minimu (ztrátové skóre zde bude velmi nízké – hodnoty vah budou nejoptimálnější). Často používané vysvětlení principu gradientního sestupu je představa o klesání z kopce do jeho nejnižšího bodu (viz obr. 3.3).

Chart, surface chart

Description automatically generated

Obr. 3.4 Znázornění gradientního sestupu ve 2D

Obr. 3.3 Model znázorňující gradientní sestup ve 3D

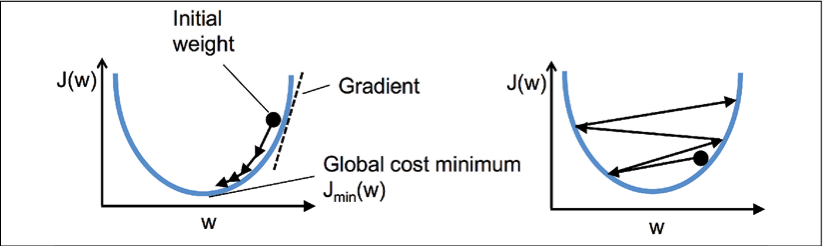
**3.5 Optimalizace váhových koeficientů**

Gradient ztrátové funkce získáme její derivací. Jelikož pracujeme s více proměnnými, výsledkem bude vektor parciálních derivací podle každého váhového koeficientu :

Tímto jsme zjistili, jak vypadá výpočet gradientu naší ztrátové funkce:

Z uvedené rovnice vyplývá, že u gradientního způsobu optimalizace, který ADALINE využívá, se všechny váhové koeficienty aktualizují najednou:

Jak jsem již vysvětloval, pro efektivní optimalizaci vah se musíme pohybovat v opačném směru oproti gradientu v daném bodě, kde značí vektor příznaků daného příkladu:

Záporný gradient je zároveň násoben rychlostí učení . Ta v tomto případě určuje, o jak velký krok se váhové koeficienty posunou ve směru záporného gradientu . Pokud je rychlost příliš veliká, může se stát, že optimalizační algoritmus globální minimum ztrátové funkce doslova *přestřelí* (viz obr. 3.5) a nikdy ho nedosáhne.

Obr. 3.5 Porovnání gradientního sestupu při optimální a příliš veliké rychlosti učení (přestřelení)

Pokud je však rychlost učení příliš malá, algoritmus se může zaseknout v lokálním minimu ztrátové funkce (viz obr. 3.6 a 3.7), ve kterém nemusí být váhové koeficienty optimální.7 Zároveň se snižuje rychlost konvergence (algoritmus je pomalejší).

Chart, surface chart

Description automatically generatedDiagram

Description automatically generated

Obr. 3.7 Krajina obsahující globální a lokální minimum

Obr. 3.6 Znázornění záseku gradientního sestupu v lokálním minimu ztrátové funkce

**3.6 Stochastický gradientní sestup**

*Stochastický gradientní sestup* (*stochastic gradient descent*) je populární (dost možná i častěji používanější) alternativou ke klasickému gradientnímu sestupu. Jedním z hlavních problémů GD je jeho konvergenční rychlost (rychlost posunu směrem k minimu). Očividným řešením tohoto problému je určení větší rychlosti učení , avšak jak jsem již vysvětloval, kvůli přestřelování je při příliš velikých hodnotách téměř nemožné dosáhnout minima. Stochastický gradientní sestup redukuje problém s konvergenční rychlostí tak, že namísto aktualizace na základě součtu všech chyb v celém trénovacím souboru aktualizuje váhové koeficienty inkrementálně zvlášť pro každý trénovací příklad souboru (viz obr. 3.8):

Může se zdát, že SGD je pouze aproximací gradientního sestupu (protože v původním odvození metody GD pracujeme s celým trénovacím souborem), avšak díky častějším aktualizacím váhových koeficientů nachází obvykle minimum funkce mnohem rychleji. Důležité je, aby bylo v každé epoše určeno pořadí trénovacích příkladů náhodně, jinak může jednoduše dojít k přetrénování. 1

Diagram

Description automatically generated

Obr. 3.8 Stochastický gradientní sestup s inkrementální aktualizací

**3.7 Souhrn**

Adaptivní lineární neuron je jednovrstvá neuronová síť, která vychází z Rosenblattova perceptronu. Na lineární vstup aplikuje lineární aktivační funkci (ta je identická, hodnota lineárního vstupu se tedy nemění). Dále pomocí definované ztrátové funkce (v tomto případě SSE) počítá ztrátové skóre, které udává míru chybovosti sítě při použití aktuálních váhových koeficientů . Pro aktualizaci vah aplikuje efektivní algoritmus gradientního sestupu, který funguje tak, že pomocí derivace ztrátové funkce (ta musí být konvexní a diferencovatelná) zjistí její gradient, a následně učiní krok v opačném směrem podle rychlosti učení . Jelikož pracujeme s více než jednou proměnnou, výsledkem je vektor parciálních derivací podle každé váhy . Pro správné fungování gradientního sestupu musí být data v každé epoše zamíchána a rychlost učení nesmí být příliš velká ani malá (parametr se většinou určuje experimentálně), aby nedocházelo k přestřelování a rychlost konvergence byla optimální. Populární alternativou ke gradientnímu sestupu je stochastický gradientní sestup, který pracuje s jednotlivými příklady namísto celého datového souboru, většinou tedy konverguje o dost rychleji.

**3.8 Implementace**

Implementace ADALINE v Pythonu je podobná jako u perceptronu. Celý algoritmus je opět zabalen ve třídě *ADALINE():*



Kód 3 – ADALINE.py

Parametry třídy *ADALINE()* jsou naprosto shodné s nastavením perceptronu (viz kapitola 2). Jediným rozdílem je atribut *cost\_,* který uchovává průměrnou hodnotu SSE v každé epoše. O trénování modelu se opět stará metoda *fit(X, y)*:



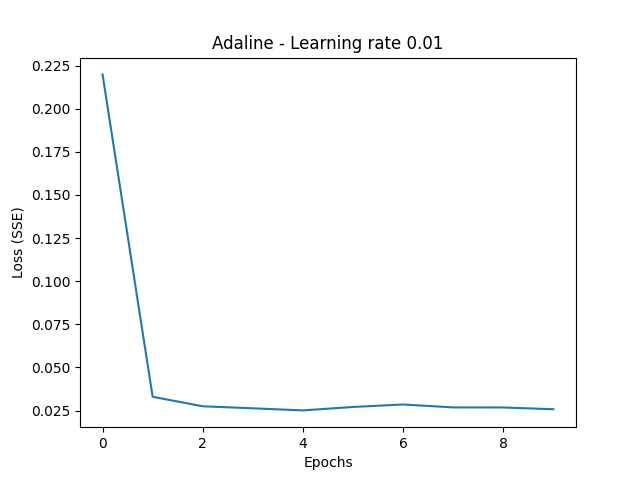
Kód 3 – ADALINE.py

Jak můžeme vidět, aktualizace vah není prováděna přímo metodou *fit(X, y)*, nýbrž pomocnou metodou *\_update\_weights(self, xi, gt)*. Kvůli zjednodušení jsem jako optimalizační algoritmus zvolil stochastický gradientní sestup, takže tato metoda vyžaduje pouze jeden vektor příznaků (jeden trénovací příklad) **xi** a hodnotu skutečné třídy **gt**:



Kód 3 – ADALINE.py

Následující graf ukazuje, jakým způsobem se během trénování snižovalo průměrné ztrátové skóre funkce SSE během deseti epoch při rychlosti učení 0,01:



Obr. 3.9 Graf zobrazující průměrné hodnoty ztrátové funkce SSE během 10 epoch

Opět jsem pracoval se dvěma příznaky z datového souboru Iris (viz kapitola 2) a rozdělil příklady na trénovací a testovací množinu. Jelikož ADALINE je stejně jako perceptron algoritmus binární klasifikace, bral jsem v úvahu pouze druhy setosa a versicolor:



Kód 4 – adaline\_iris\_binary.py

Natrénovaný model opět správně klasifikoval všech 10 příkladů z testovací množiny dat. Jak je vidět z grafu na obrázku 3.9, stochastický gradientní sestup fungoval v tomto případě skvěle, jelikož ztrátová funkce byla minimalizována již během první epochy. Je však třeba mít na paměti, že v porovnání s reálnými situacemi se jedná o síť s velmi málo parametry a data v Iris jsou jednoduše lineárně separovatelná, takže podobný výsledek byl předvídatelný. I tak je však dobře vidět síla a efektivita gradientního přístupu k optimalizaci.