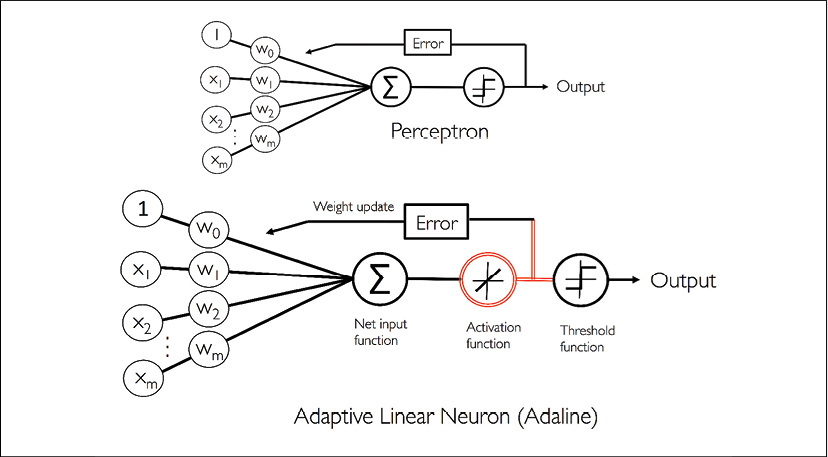
**3. ADALINE**

**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 profesor 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.

**3.2 Lineární aktivační funkce**

Hlavním rozdílem mezi ADALINE a perceptronem je způsob, jakým probíhá učení (optimalizace vah). Zatímco u perceptronu jsou váhové koeficienty aktualizovány na základě rozdílu mezi skutečnou a predikovanou třídou, ADALINE nejdříve počítá skóre ztrátové funkce, které se snaží minimalizovat, a teprve poté vytváří predikci.6 Kromě skokové funkce kvůli predikci využívá navíc lineárníaktivační funkci, která je prostě funkcí identity lineárního vstupu1:



Obr. 3. Porovnání schémat perceptronu a ADALINE

**3.3 Ztrátová funkce**

*Ztrátová* nebo *objektivní 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:

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

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 nelézt 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 oproti gradientu ztrátové funkce a snaží se dostat co nejblíže 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 nejnižšího bodu.

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 :

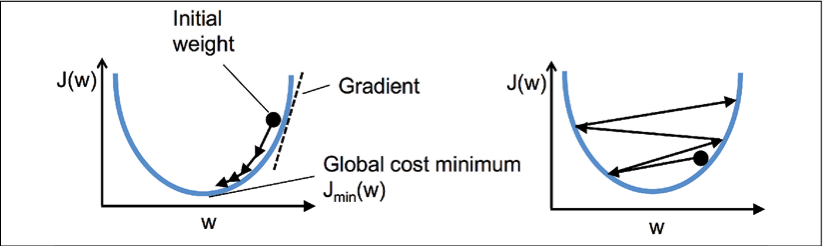
Tímto jsme zjistili, jak vypadá výpočet gradientu naší ztrátové funkce **:**

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

Již jsem vysvětloval, že pro efektivní optimalizaci vah se musíme pohybovat v opačném směru oproti gradientu ztrátové funkce v daném bodě:

Záporný gradient je zároveň násoben rychlostí učení . Ta v tomto případě určuje, o jak velký krok se váhové koeficienty posunou směrem od gradientu. Pokud je rychlost příliš veliká, může se stát, že optimalizační algoritmus globální minimum ztrátové funkce doslova *přestřelí* 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, GD se může zaseknout v lokálním minimu ztrátové funkce, ve kterém nemusí být váhové koeficienty optimální.7 Zároveň se snižuje konvergenční rychlost.

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 alternativou ke klasickému algoritmu gradientního sestupu. Jedním z hlavních problémů GD je jeho konvergenční rychlost. Očividným řešením tohoto problému je určení větší rychlosti učení, avšak jak jsem již vysvětloval, kvůli přestřelování je při příliš velikých hodnotách téměř nemožné dosáhnout globálního minima. Stochastický gradientní sestup redukuje problém s konvergenční rychlostí tak, že namísto aktualizace vah na základě součtu všech chyb aktualizuje váhové koeficienty inkrementálně podle každého tréninkového příkladu:

Může se zdát, že SGD je pouze aproximací gradientního sestupu, avšak díky častějším aktualizacím váhových koeficientů nalézá 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

**3.8 Souhrn**

Adaptivní lineární neuron je jednovrstvá neuronová síť, která vychází z Rosenblattova perceptronu. Na lineární vstup aplikuje lineární aktivační funkci (která je pouze funkcí identity, hodnotu lineárního vstupu 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 opačným směrem. 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 rychlostí učení nesmí být příliš velká ani malá.

**3.7 Implementace**