

Kapitola 2

Perceptron a úvod do DL

V této kapitole se seznámíte s perceptronem – jedním z nejjednodušších modelů neuronových sítí, který historicky položil základy moderního strojového učení. Perceptron je lineární klasifikátor, jenž rozhoduje na základě váženého součtu vstupů a následné aplikace aktivační funkce. Přestože má určitá omezení, je klíčovým stavebním kamenem pro porozumění složitějším neuronovým architekturám.

Cíle kapitoly (po přečtení této kapitoly budete schopni)

- C1.1. Popsat strukturu perceptronu (neuronu) a jeho základní princip fungování.
- C1.2. Vysvětlit roli vah, prahové hodnoty (biasu) a aktivační funkce.
- C1.3. Pochopit mechanismus trénování perceptronu (neuronu) na základě učících dat.
- C1.4. Rozlišit úlohy, které je perceptron (neuron) schopen řešit, a ty, kde jeho použití selhává.

Klíčové myšlenky (po přečtení této kapitoly byste si měli pamatovat)

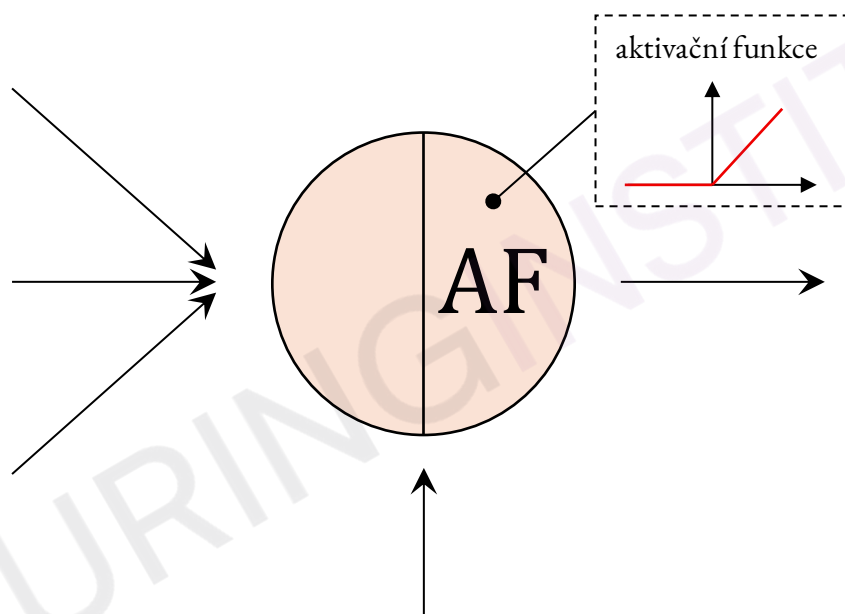
- KM1.1. Perceptron je základní jednotkou neuronových sítí – zpracovává vstupy, váží je a rozhoduje podle výsledné hodnoty.
- KM1.2. Váhy určují význam jednotlivých vstupů, bias posouvá rozhodovací hranici a aktivační funkce určuje výstup.
- KM1.3. Učení perceptronu (neuronu) probíhá úpravou vah podle chyby mezi očekávaným a skutečným výstupem.
- KM1.4. Perceptron (neuron) dokáže řešit pouze lineárně separovatelné úlohy, což je jeho zásadní omezení.
- KM1.5. Přestože je perceptron (neuron) jednoduchý, jeho principy jsou klíčové pro vícevrstvé neuronové sítě, které tato omezení překonávají.

Nové příkazy (po přečtení kapitoly se seznámím s těmito novými příkazy)

1.1 Co je to perceptron (neuron)?

Neuron je jeden z nejjednodušších modelů umělé neuronové sítě. Lze si jej představit jako „rozhodovací jednotku“, která dostane několik vstupních hodnot a na základě nich určí, do které skupiny daný vstup patří. Každý vstup má přiřazenou **váhu**, která vyjadřuje, jak důležitý tento vstup je. Všechny vstupy se vynásobí svými váhami, sečtou se a přičte se k nim tzv. **bias** (posun, který umožňuje měnit rozhodnutí). Výsledkem je jedno číslo, které se následně vyhodnotí pomocí tzv. **aktivační funkce** – například tak, že pokud je toto číslo větší než určitý práh, výstupem je číslo 1 (dá se chápat také jako hodnota „Ano“), pokud je menší, výstupem je číslo 0 (dá se chápat také jako hodnota „Ne“). Trénování *neuronu* obecně představuje postupné nastavování vah a biasu tak, aby výstupy co nejvíce odpovídaly správným odpovědím v trénovacích datech. *Neuron* tedy umí řešit jen jednodušší úlohy, ale jeho princip je základním stavebním kamenem složitějších neuronových sítí, které už dokáží pracovat i se složitými problémy.

1.2 Podoba neuronu



Jak tedy výpočet neuronu vypadá?

1.3 Jak si představit neuron?

Neuron si můžeme představit jako rozhodovací jednotku, která zpracovává informace a převádí je do jednoduchého výsledku. Jeho úkolem není dělat složité závěry, ale z množství vstupů určit, jestli výsledek směřuje spíše k odpovědi „ano“, nebo spíše k odpovědi „ne“.

Představme si, že neuron dostane několik vstupních hodnot. Každá z těchto hodnot něco vypovídá o situaci, ale ne všechny mají stejný význam. Proto neuron přiřazuje každému vstupu určitou váhu – tím říká, jak moc tuto informaci bere v úvahu. Důležité vstupy dostanou vysokou váhu, méně důležité naopak váhu malou. Potom neuron všechny takto „zvážené“ vstupy sečte. Aby bylo rozhodování ještě pružnější, přidává se k součtu tzv. bias. Ten funguje jako posun hranice rozhodnutí. Bias můžeme chápat jako určitou přednastavenou tendenci neuronu – například, že má obecně spíše sklon odpovídat „ano“ nebo naopak „ne“. Díky biasu není neuron vázán jen na to, co přichází ze vstupů, ale může své rozhodování jemně posouvat. Výsledný součet všech vstupů a biasu sám o sobě ještě není odpověď. Aby neuron mohl vydat jasné rozhodnutí, musí tento výsledek projít tzv. aktivační funkcí. Ta funguje jako spínač nebo filtr: pokud je hodnota dostatečně vysoká, neuron se „aktivuje“ a vydá výstup 1, což můžeme chápat jako odpověď „ano“. Pokud hodnota naopak nestačí, neuron zůstane neaktivní a vydá výstup 0, tedy „ne“.

Neuron tedy funguje jako rozhodovací mechanismus, který zváží všechny argumenty, zohlední jejich důležitost, přidá svůj vlastní „posun“ a nakonec vyřkne jednoznačný verdikt. Takto jednoduchá jednotka sama o sobě umí řešit jen základní problémy, ale právě díky své jednoduchosti je ideálním stavebním kamenem pro složitější neuronové sítě. V kombinaci s dalšími neurony se totiž tato jednoduchá logika skládá do mnohem komplexnějších rozhodovacích struktur, které už dokáží rozpoznávat obrazy, text nebo zvuk.

1.4 Co je to aktivační funkce?

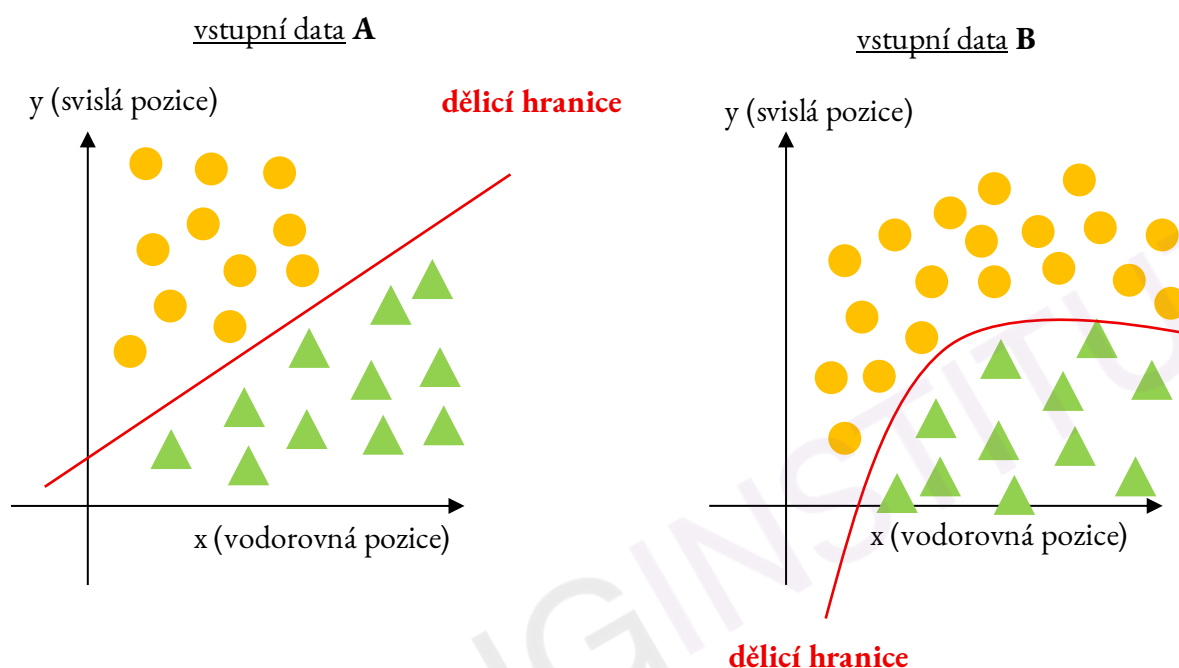
Když neuron zpracuje všechny vstupy, vynásobí je jejich vahami a přičte bias, dostaneme jedno číslo. Toto číslo samo o sobě ale ještě nic neříká – může být kladné, záporné, velké i malé. Abychom z něj získali jasný a použitelný výstup, musí projít přes **aktivační funkci**. Aktivační funkce funguje jako rozhodovací brána. Kontroluje, zda je vypočtená hodnota dostatečně velká na to, aby se neuron „aktivoval“. Pokud ano, neuron vyšle signál (například hodnotu 1, tedy odpověď „ano“). Pokud ne, zůstane neaktivní (hodnota 0, odpověď „ne“).

Ne vždy ale musí výsledek znamenat jen ostré „ano“ nebo „ne“. Některé aktivační funkce umožňují, aby výstup neuronu ležel i mezi těmito dvěma krajními hodnotami. V takovém případě lze říci, že neuron ukazuje, zda se jeho výpočet spíše přiklání k odpovědi „ano“, nebo spíše k „ne“. Aktivační funkce je proto nezbytná: dává neuronu možnost převést surový výsledek výpočtu do konkrétní podoby, se kterou se dá dál pracovat. Bez ní by neuron byl jen jednoduchým sčítačem čísel a nedokázal by vydávat rozhodnutí.

1.5 Stačí neuron i na složitější úlohy?

$$výstup = AF\left(\left(\sum_{i=1}^3 váha_i \cdot vstup_i\right) + bias_1\right) \quad (1.1)$$

Neuron je velmi jednoduchý model – odděluje třídy pomocí rovné hranice v datech: v rovině jde o přímku, v prostoru o rovinu a ve vícerozměrném prostoru o tzv. hyperrovinu. Pokud tedy lze dvě skupiny bodů rozdělit jednou rovnou čarou (viz obrázek níže), *neuron* je schopen úlohu vyřešit. Jestliže je k rozdělení vstupních dat potřeba zakřivená nebo „zlomená“ hranice, *neuron* na takovou úlohu nestačí.



Na obrázku č. 1 výše jsou zobrazeny dvě sady vstupních dat, vstupní data **A** a vstupní data **B**. Jak lze vidět z obrázku, vstupní data **A** lze oddělit rovnou čarou, jsou tedy jednoduše oddělitelná, což znamená, že pro jejich klasifikaci (určení, do které skupiny daný tvar patří) lze použít jednoduchý *neuron*, vstupní data **B** však jednoduše oddělitelná nejsou, jelikož na jejich oddělení je potřeba složitější křivka, jejich klasifikaci tedy jednoduchým *neuronem* provést nelze.

1.6 Trénovací proces neuronu

Aby se neuron dokázal naučit správně rozhodovat, potřebuje projít trénovacím procesem. Ten probíhá v několika krocích, které se stále opakují, dokud se model nezlepší natolik, že je jeho výkon dostatečný.

Cílem trénování *neuronu* je nastavit jeho parametry – tedy **váhy** a **bias** – tak, aby rovná rozhodovací hranice co nejlépe oddělovala dvě třídy v datech. Model postupně prochází trénovací příklady, porovnává svou predikci $\hat{y}$ se správnou odpovědí y_{true} a při chybě upraví tyto parametry tak, aby příště rozhodl lépe. Níže je celý proces vysvětlen stručně a krok za krokem.

Základní pojmy

vzorek dat („sample“) – trénovací příklad (x, y_{true}) , kde x jsou vstupní hodnoty a y_{true} , které může nabývat hodnoty od 0 do 1, je hodnota reprezentující správnou třídu;

váhy a bias („weights“, „bias“) – parametry w a b , které určují sklon a posun přímkové rozhodovací hranice;

predikce („prediction“) – rozhodnutí modelu pro vstup x – nabývá hodnoty 0, nebo 1 – podle strany hranice;

rychlost učení („learning rate“) η – malé kladné číslo určující velikost kroků při úpravě parametrů během trénovacího procesu;

iterace, krok („iteration“) – zpracování jednoho vzorku dat a případná aktualizace parametrů neuronu;

epocha („epoch“) – jedno úplné projití celé trénovací sady (všech vzorků) neuronem při jeho trénování,

ztrátová funkce („loss function“) – funkce, která vyjadřuje, jak moc se model mýlí – kvantifikuje rozdíl mezi správným výstupem neuronu y_{true} a jeho predikcí $\hat{y}$.

Popis trénovacího procesu

1. Rozdělení dat na:

- trénovací sadu – používá se k samotnému učení vah a biasu, tedy k tomu, aby se neuron přizpůsoboval příkladům,
- validační sadu – slouží k ladění různých hyperparametrů učení, například toho, jak velké kroky neuron při učení dělá nebo kolikrát projde celou trénovací sadou,
- testovací sadu – využívá se až na konci, kdy ověřujeme, jak dobře se neuron naučil řešit úlohu i na nových, jemu dosud neznámých datech.

2. Příprava dat:

Než se data použijí, často se ještě předzpracují – například se jejich hodnoty převedou do určitého rozsahu nebo se promíchají, aby se neuron neučil jen na jedné části dat, ale měl vyvážený přehled.

3. Inicializace modelu: nastavení:

Samotné učení začíná nastavením počátečních hodnot vah a biasu. Tyto hodnoty bývají zvoleny jako malé náhodné číslo. Dále se určí parametry učení, například jak velký krok se bude dělat při opravě vah (tzv. „learning rate“) a kolikrát se má projít celá trénovací sada (tzv. „počet epoch“).

4. Učení modelu prostřednictvím trénovací sady:

Trénování probíhá opakovaně a skládá se z těchto kroků:

- průchod daty – neuron zpracuje jeden vzorek z trénovací sady a vytvoří svůj výstup (predikci) podle aktuálního nastavení vah a biasu,
- porovnání se správnou odpovědí – výstup neuronu se porovná s tím, co má být správně,
- výpočet chyby – rozdíl mezi správným výsledkem a odpovědí neuronu se vyjádří pomocí ztrátové funkce, která ukáže, jak moc se neuron spletl,
- úprava vah a biasu – váhy a bias se změní o malý krok směrem k hodnotám, které by vedly k přesnějšímu výsledku, velikost tohoto kroku je dána hyperparametrem „learning rate“,
- opakování – celý postup se znovu provádí pro každý další vzorek.

Poslední bod trénovacího procesu představuje opakující se proces, což znamená, že se činnosti v tomto bodě neustále opakují, dokud není hodnota ztráty, čili rozdíl mezi predikovanou hodnotou a hodnotou skutečnou, **dostatečně malá** (při vyhodnocení se využívají metriky uvedené v učebním textu „Úvod do strojového učení“), nebo dokud není dosaženo maximálního počtu možných iterací, což je parametr definován uživatelem – v obou případech však v tomto bodě trénovací proces končí.

1.7 Optimalizační algoritmus neuronu

Když se neuron učí, potřebuje nějaký způsob, jak se rozhodnout, o kolik a jakým směrem má změnit své váhy a bias. Nestačí jen vědět, že se spletl – je třeba vědět také, **jak opravit svou chybu co nejefektivněji**. A právě to zajišťuje tzv. optimalizační algoritmus.

Jeho úkolem je, aby neuron při učení vždy našel vhodný krok směrem k lepšímu výsledku. Lze si to představit jako cestu z kopce do údolí: chyba neuronu je jako výška terénu. Neuron stojí někde na svahu a snaží se dostat co nejniž – tam, kde je chyba minimální. Optimalizační algoritmus mu říká, kterým směrem se má vydat a jak velký krok má udělat.

Nejjednodušší a nejčastěji používanou metodou je **gradientní sestup (Gradient Descent)**. Princip je tento:

- pokud je chyba velká, algoritmus zjistí, kterým směrem se hodnota vah musí změnit, aby se chyba zmenšila,
- udělá krok tímto směrem – ne příliš velký, aby neskočil úplně jinam, ale ani ne příliš malý, aby se zbytečně nezdržoval,
- tento postup se opakuje pořád dokola.

Díky tomu neuron krok za krokem „sestupuje z kopce“ a jeho rozhodování se stále zpřesňuje. Celý proces končí, když už je chyba dostatečně malá, nebo když uběhne předem stanovený počet kroků.

Optimalizační algoritmus je tedy mozkiem samotného učení: převádí informaci o chybě na konkrétní úpravy uvnitř neuronu a umožňuje mu zlepšovat se s každým dalším příkladem.

1.8 Skládání více neuronů do tzv. neuronové sítě

Samostatný neuron je velmi jednoduchý prvek. Proto se v praxi spojují neurony dohromady. Pokud je jich více vedle sebe, mluvíme o **vrstvě**. A pokud takových vrstev umístíme několik za sebou, vzniká **neuronová síť**. Základní myšlenka je, že výstup jedné vrstvy se stává vstupem vrstvy další. Tím se z jednotlivých, poměrně jednoduchých rozhodovacích jednotek vytváří celek, kde každá vrstva navazuje na předchozí a posouvá zpracování dat o krok dál. Tento princip skládání vrstev tvoří základ všech moderních neuronových sítí – od nejjednodušších až po ty nejkompaktnější.

To, že se vrstvy skládají za sebe, samo o sobě není ještě to nejdůležitější. Podstatné je, že **každá další vrstva přidává novou úroveň zpracování dat**. První vrstva si všímá základních znaků, další vrstva je kombinuje do složitějších obrazců a další vrstva z těchto obrazců skládá ještě komplexnější struktury. Jinými slovy: více vrstev umožňuje vytvářet **hierarchii poznatků**. Síť se tak učí řešit problém postupně – od jednoduchých dílků až po celý složitý obraz. Díky tomuto vrstvení dokážou neuronové sítě zachytit i velmi komplikované souvislosti, které jsou pro jediný neuron nedostupné.

1.9 Rozpoznávání obrazu pomocí neuronových sítí

Obrázek se skládá z mnoha malých čtverečků – pixelů. Každý pixel má svou barvu nebo jas. Pokud bychom každý pixel předali neuronové síti zvlášť, dostali bychom obrovské množství čísel, ve kterých by bylo velmi obtížné poznat, co obrázek vlastně zobrazuje.

Proto se vytvořily tzv. **konvoluční neuronové sítě**. Ty nečtou obraz pixel po pixelu, ale „dívají se“ na něj po menších částech. Než se k tomu dostanou, obrázek se obvykle ještě **zjednoduší (normalizuje)** – například se číselné hodnoty pixelů, které reprezentují jejich barvy, převedou na čísla v rozsahu od 0 do 1. Díky tomu jsou data jednodušší a síť se z nich dokáže učit spolehlivěji.

Samotný název *konvoluční* pochází z operace zvané **konvoluce**. V praxi to znamená, že vezmeme malou mřížku – **kernel (filtr)** – a tuto mřížku postupně posouváme přes celý obrázek. V každém kroku se filtr porovná s malou částí obrázku a vypočítá číslo, které říká, jak moc tato část připomíná sledovaný vzor. Když tento postup zopakujeme pro celý obrázek, dostaneme nový „obrázek“ – tzv. **feature mapu**, kde jsou už zvýrazněny důležité rysy, jako jsou hrany nebo křivky.

Aby se síť nezabývala každým detailem, ale jen tím nejpodstatnějším, používá se často také krok nazývaný **max-pooling**. Ten zmenší obrázek tak, že z několika sousedních pixelů vezme jen ten nejvýraznější. Výsledkem je jednodušší obraz, kde se neztratila podstata, ale síť má méně dat ke zpracování.

Celý proces probíhá postupně ve vrstvách:

- **první vrstvy** odhalují čáry a hrany,
- **další vrstvy** skládají čáry do tvarů (oblouky, křížení, uzavřené části),
- **hlubší vrstvy** tyto tvary spojují do celého objektu (například číslice nebo obličeje).

Na příkladu čísla „5“ to vypadá takto: síť nejprve díky konvolucím a poolingům uvidí krátké tahy a zakřivení, pak z nich složí horní oblouček a spodní čáru, a nakonec z toho vyhodnotí, že se jedná o číslici „5“.

Aby síť na konci opravdu rozhodla, co obrázek představuje, používá ještě **plně propojenou vrstvu neuronů**. Tady je každý neuron spojený se všemi neurony předchozí vrstvy. V praxi to znamená, že závěrečné rozhodnutí už není založené jen na jednotlivých čarách nebo tvarech, ale na kombinaci všech informací, které si síť během předchozích vrstev vybudovala. Plně propojená vrstva pak vydá konečné rozhodnutí – například určí, jestli jde o číslo 3, 5 nebo 7.

Díky tomuto postupnému zpracování – od normalizace přes konvoluce a pooling až po plně propojené vrstvy – dokážou konvoluční neuronové sítě rozpoznávat obrázky velmi spolehlivě. Proto se dnes používají všude tam, kde je potřeba pracovat s vizuálními daty: od čísel a písmen přes dopravní značky až po lékařské snímky.

ZDE OBRÁZEK CNN

Příloha – více matematiky

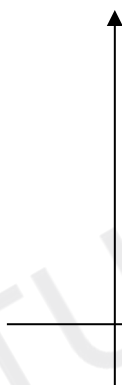
Optimalizační algoritmus neuronu

Při trénování neuronu se používají různé optimalizační algoritmy, jejichž cílem je najít takové hodnoty vah a biasu, aby model co nejlépe řešil zadaný úkol (například správně klasifikoval vstupní data). Existuje více přístupů k trénování neuronu, mezi nejjednodušší patří tzv. „Perceptron Learning Algorithm“, které funguje i při použití nediferencovatelných aktivačních funkcí, jako je například skoková funkce. Pokud je však aktivační funkce diferencovatelná (například sigmoid, tanh nebo ReLU), lze využít metody založené na tzv. gradientním sestupu („gradient descent“) a jeho variantách („stochastic gradient descent“, „mini-batch gradient descent“ aj.). V těchto případech se změny vah odvozují pomocí derivací zvolené chybové funkce (viz minulá kapitola, bod c). Celý proces optimalizačního algoritmu bude vysvětlen v této kapitole s důrazem na jeho pochopení, nikoliv na jeho matematickou stránku.

Co to je derivace?

Derivace je matematický nástroj, kterým popisujeme, jak se něco mění. Představte si třeba, že sledujete teplotu během dne: ráno je chladno, postupně se otepluje, kolem poledne je nejtepleji a večer teplota opět klesá. Kdyby došlo k vykreslení závislosti teploty na části dne, mohl by graf vypadat následovně:

Teplota ve stupních Celsia



Derivace dokáže v každém okamžiku takového grafu říct, zda teplota stoupá či klesá a zároveň jak rychle – na grafu lze vidět, že rychlost stoupání teploty v první polovině dne (stoupající část grafu) je menší než rychlost klesání teploty v druhé polovině dne (klesající část grafu). Když si čáru v grafu určující teplotu v závislosti na tom, kolik je zrovna hodin, představíte jako funkci, její obecný zápis by mohl být následovný:

$$teplota = funkce(čas),$$

který říká, že teplota je funkce, která je závislá na čase – to znamená, že to, kolik je hodin, ovlivňuje to, jaká je teplota. Derivace této funkce by vypadala následovně:

$$derivace = \frac{změna(funkce(čas))}{změna(čas)},$$

a když dojde k malé změně názvů na trochu odbornější, získáte známý tvar:

$$f'(t) = \frac{df(t)}{dt},$$

kde $f'(t)$ představuje označení derivace, $f(t)$ funkci, která je derivována a t proměnnou, podle které se derivuje. V našem případě je tedy $f(t)$ funkce stanovující teplotu v závislosti na proměnné t , která označuje čas, tedy to, kolik je zrovna hodin. Z důvodů, které zde nebudou detailně vysvětleny, platí následující:

když je derivace funkce v dané oblasti <u>kladná</u> , čili když: derivace > 0 :	funkce je v dané oblasti <u>rostoucí</u>
když je derivace funkce v dané oblasti <u>záporná</u> , čili když: derivace < 0 :	funkce je v dané oblasti <u>klesající</u>
když je derivace funkce v dané oblasti <u>nulová</u> , čili když: derivace $= 0$:	funkce v dané oblasti ani neklesá a ani neroste, čili se zde nachází její maximální/minimální hodnota

Využití výše uvedených poznámek:

Když si představíte, že jste slepý panáček, který se nachází v grafu uvedeném na minulé stránce, a že křivka, která je v grafu uvedena, je palouček, derivace by vám díky znalostem uvedeným v tabulce výše dokázala v každé pozici na tomto paloučku říct, jakým směrem byste měli jít, kdybyste chtěli po paloučku stoupat, či naopak klesat, a to následovně:

- když jdete po paloučku směrem doprava a derivace funkce (paloučku) je kladná, znamená to, že stoupáte: když tedy budete chtít dále stoupat, stačí jít ve směru stoupající derivace, když budete chtít klesat, stačí se otočit a jít proti směru derivace.

Myšlenka uvedena v odrážce výše je základním stavebním kamenem optimalizačního algoritmu neuronu, na základě něhož se samotný neuron trénuje a jehož název je Gradient Descent.

Popis trénovacího procesu

1. Rozdělení dat na:

- trénovací sadu – učení vah a biasu,
- validační sadu – ladění hyperparametrů (například „learning rate“, počet epoch apod.),
- testovací sadu – finální ověření přesnosti natrénovaného modelu.

Dobrým rozdělením může být například rozdělení $a : b : c = (60 : 20 : 20) \%$.

2. Příprava dat:

Normalizace dat, volby rysů (tzv. „feature engineering“) či náhodné zamíchání dat.

3. Inicializace modelu: nastavení:

Prvotní nastavení hodnot vah w_i a biasu b na velmi malé náhodné hodnoty a volba „learning rate“ η (typicky na hodnotu kolem 0.01 až 1) a maximálního počtu epoch.

$$\text{výstup} = \frac{1}{1 + e^{-(0.005 \cdot x + 0.002 \cdot y + 0)}}.$$

4. Učení modelu prostřednictvím trénovací sady:

- a. průchod vzorku označených trénovacích dat neuronem a tvorba predikce $\hat{y}$ tohoto vzorku pomocí dříve inicializovaného neuronu:

$$\text{vzorek} = \text{●} \quad \text{pozice vzorku} = (2, 8) \quad \hat{y} = \frac{1}{1 + e^{-((0.005 \cdot 2 + 0.002 \cdot 8) + 0)}} = 0.5065$$

- b. výpočet chyby predikce $\hat{y}$ vůči skutečné hodnotě y_{true} prostřednictvím ztrátové funkce v podobě tzv. „binary cross-entropy“ (BCE):

$$\begin{aligned} ZF_{BCE}(y_{true}, \hat{y}) &= -[y_{true} \cdot \ln \ln(\hat{y}) + (1 - y_{true}) \cdot \ln(1 - \hat{y})] \\ ZF_{BCE}(1, 0.5065) &= -[1 \cdot \ln \ln(0.5065) + (1 - 1) \cdot \ln(1 - 0.5065)] \\ ZF_{BCE}(1, 0.5065) &= -[1 \cdot \ln \ln(0.5065) + (1 - 1) \cdot \ln(1 - 0.5065)] \\ ZF_{BCE}(1, 0.5065) &= -[1 \cdot \ln \ln(0.5065)] \\ ZF_{BCE}(1, 0.5065) &= 0.6802 \end{aligned}$$

- c. úprava hodnot vah a biasu dle hodnoty ztrátové funkce ZF_{BCE} a hodnoty „learning rate“ η s využitím techniky tzv. „gradient descent“ (může se použít v případě diferencovatelné ztrátové funkce – případ, kdy ztrátovou funkci lze derivovat) a využitím pravidla o derivaci složené funkce:

$$\begin{aligned} w_{1, \text{nový}} &= w_{1, \text{starý}} - \eta \cdot \frac{dZF_{BCE}}{dw_1}, \\ w_{1, \text{nový}} &= |\text{pravidlo o derivaci složené funkce}| = w_{1, \text{starý}} - \eta \cdot \frac{dZF_{BCE}}{d\hat{y}} \cdot \frac{d\hat{y}}{dz} \cdot \frac{dz}{dw_1}, \\ w_{1, \text{nový}} &= 0.005 - 0.1 \cdot \frac{d\{-[y_{true} \cdot \ln \ln(\hat{y}) + (1 - y_{true}) \cdot \ln \ln(1 - \hat{y})]\}}{d\hat{y}} \cdot \frac{d\left(\frac{1}{1 + e^{-z}}\right)}{dz} \cdot \frac{d((w_1 \cdot 2 + w_2 \cdot 8) + b_1)}{dw_1}, \\ w_{1, \text{nový}} &= 0.005 - 0.1 \cdot -\left(\frac{y_{true}}{\hat{y}} - \frac{1 - y_{true}}{1 - \hat{y}}\right) \cdot \frac{e^{-z}}{(1 + e^{-z})^2} \cdot 2 \\ w_{1, \text{nový}} &= 0.005 - 0.1 \cdot -\left(\frac{y_{true}}{\hat{y}} - \frac{1 - y_{true}}{1 - \hat{y}}\right) \cdot \frac{e^{-z}}{(1 + e^{-z})^2} \cdot 2 \\ w_{1, \text{nový}} &= 0.005 + 0.2 \cdot (y_{true} - \hat{y}) \\ w_{1, \text{nový}} &= 0.005 + 0.2 \cdot (1 - 0.5065) \\ w_{1, \text{nový}} &= 0.1037; \end{aligned}$$

obdobně by došlo k výpočtu $w_{2, \text{nový}}$ a $b_{1, \text{nový}}$:

$$w_{2, \text{nový}} = w_{2, \text{starý}} - \eta \cdot \frac{dZF_{BCE}}{dw_2}, \quad b_{1, \text{nový}} = b_{1, \text{starý}} - \eta \cdot \frac{dZF_{BCE}}{db_1}.$$

Po výpočtu nových vah a biasu jsou tyto hodnoty v neuronu aktualizovány. Tento bod trénovacího procesu představuje iterativní proces, což znamená, že se činnosti v tomto bodě neustále opakují, dokud není hodnota ztráty, čili rozdíl mezi predikovanou hodnotou a hodnotou skutečnou, **dostatečně malá** (při vyhodnocení se využívají metriky uvedené v učebním textu „Úvod do strojového učení“), nebo dokud není dosaženo maximálního počtu možných iterací, což je parametr definován uživatelem – v obou případech však v tomto bodě trénovací proces končí. Důvod, proč matematické vyjádření algoritmu „gradient descent“ vypadá tak, jak jsme si uvedli výše v bodě c, bude vysvětlen v kapitole s názvem „optimalizační algoritmus neuronu“.

Důkaz linearity perceptronu (neuronu)

Proč však jednoduchý *neuron* umí řešit pouze lineární úlohy? Abychom mohli odpovědět na tuto otázku, pojďme si blíže analyzovat vztah 1.1.

Obecně tedy platí:

$$výstup = AF\left(\left(\sum_{i=1}^3 váha_i \cdot vstup_i\right) + bias_1\right).$$

Když si představíme řešení úlohy vstupních dat A, jedná se o klasifikaci skutečnosti, jestliže je vstupní obrazec „oranžový kruh“, nebo „zelený trojúhelník“, *neuron* by tedy měl dvě vstupní hodnoty, které by reprezentovaly souřadnice obrazce (pozici obrazce vůči vodorovné a svislé ose). Pro 2 vstupní hodnoty do *neuronu* a funkci sigmoid jako aktivací funkci platí:

$$výstup = AF(z) = AF((váha_1 \cdot vstup_1 + váha_2 \cdot vstup_2) + bias_1)$$

$$výstup = \left| sigmoid = \frac{1}{1 + e^{-x}} \right| = sigmoid(z) = \frac{1}{1 + e^{-((váha_1 \cdot vstup_1 + váha_2 \cdot vstup_2) + bias_1)}}$$

Ještě nesmíme zapomenout na označení dat a jejich číselnou reprezentaci:

$výstup = 1$ pro 

$výstup = 0$ pro 

a na vyjádření vstupu pomocí pozice daného obrazce:

$$vstup = (x, y),$$

kde **x** představuje vodorovnou souřadnici popisující polohu obrazce ve vodorovném směru a **y** představuje svislou souřadnici popisující polohu obrazce ve svislém směru.

Předchozí obecnou rovnici tedy můžeme přepsat následovně:

Platí pro dokonale funkční² neuron:



Důležité tvrzení a důkaz linearity:

Aktivační funkce sigmoid může nabývat hodnot od 0 do 1, což znamená, že i celkový výstup *neuronu* bude nabývat hodnot od 0 do 1, a tedy i hranice, která rozhoduje o tom, zda výstup bude oranžový kruh, nebo zelený trojúhelník, tedy musí ležet mezi těmito hodnotami. Platí tedy:

$(výstup \geq hranice) \Leftrightarrow \text{třída}$ 

$(výstup \leq hranice) \Leftrightarrow \text{třída}$ 

Jelikož je aktivační funkce sigmoid monotónní (neexistuje oblast, ve které by funkce byla klesající), lze tvrdit následující (směr nerovnosti $\geq$ se při aplikaci inverzní funkce na obě strany rovnice nemění):



výstup

$$\begin{aligned} \text{sigmoid}(z) \geq hranice &\Leftrightarrow z \\ &\geq \text{sigmoid}^{-1}(hranice) \end{aligned}$$

Množina bodů třídy  a :

$$\begin{aligned} \{(x, y): (váha_1 \cdot x + váha_2 \cdot y) + bias_1 \geq \text{sigmoid}^{-1}(hranice)\}, \\ \{(x, y): (váha_1 \cdot x + váha_2 \cdot y) + bias_1 \leq \text{sigmoid}^{-1}(hranice)\}, \end{aligned}$$

což je matematický zápis pro polovinu, jejíž hranice je **přímka**:

$$(váha_1 \cdot x + váha_2 \cdot y) + bias_1 = \text{sigmoid}^{-1}(hranice)$$

Dle důkazu uvedeného výše vidíme, že hranice oddělující vstupní hodnoty je přímka, což obecně znamená, že neuron je opravdu lineární model (zde odvozeno pro aktivační funkci sigmoid, tvrzení je však obecně platné i pro ostatní monotónní aktivační funkce).