**1. Základy neuronových sítí a hlubokého učení**

**Jak neuronová síť skutečně funguje**

* **Vstupy a váhy**: Každý neuron přijímá vstupy, které mají určité váhy. Váhy řídí důležitost každého vstupu, a jak ovlivní výstup.
* **Bias** (posun): Další parametr přidávaný k výpočtu, který umožňuje neuronům lépe modelovat vztahy mezi daty.
* **Výstup a chybová funkce**: Výstup sítě se porovnává se skutečným výsledkem pomocí chybové (ztrátové) funkce, která určí, jak moc se výstup modelu liší od skutečných hodnot.

**Backpropagation (zpětná propagace chyby)**

* **Gradientní sestup (Gradient Descent)**: Algoritmus, který minimalizuje chybovou funkci iterativním upravováním vah. Vyhledává směrem dolů ve funkci chyb, aby našel minimum.
* **Zpětná propagace**: Proces, při kterém se chyba vypočítaná na výstupu zpětně propaguje přes síť. Gradienty se počítají pro každou váhu tak, aby se minimalizovala chyba.

**Optimalizátory**

* **SGD (Stochastic Gradient Descent)**: Často používaný, ale u menších dávkových velikostí je nestabilní.
* **Adam** (Adaptive Moment Estimation): Kombinuje výhody RMSProp a momentum, což znamená, že upravuje rychlost učení na základě změny gradientu.
* **Momentum**: Udržuje pohyb váh v předchozích směrech, což pomáhá překonat lokální minima.

**2. Detailní popis Keras API**

**Nastavení a trénování modelu**

* **Model.compile()**: Definuje chybovou funkci, optimalizátor a metriky pro hodnocení výkonu. Například:
* python
* Copy code
* model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])
* **Model.fit()**: Spouští trénování modelu nad daty. Parametry epochs, batch\_size, validation\_data atd. řídí, kolikrát model projde daty a jaké části si vyčlení na validaci.

**Přizpůsobení modelů pomocí Callbacků**

* **Callbacky** jako EarlyStopping, ModelCheckpoint nebo TensorBoard umožňují sledování tréninkového procesu, automatické ukládání modelu při dosažení lepších výsledků nebo přerušení tréninku, pokud se výkon modelu přestane zlepšovat.
* **EarlyStopping** sleduje metriky jako val\_loss a ukončí trénink, pokud se metrika přestane zlepšovat, čímž zabrání přetrénování.

**3. Konvoluční neuronové sítě (CNN)**

**Konvoluční vrstvy: Podrobný popis**

* **Filtry (kernels)**: Každý filtr má vlastní váhy, které se během trénování upravují, aby dokázal detekovat specifické vzory. Například jeden filtr může detekovat hrany, jiný textury.
* **Vlastnosti prostorové invariance**: Díky konvolucím se model naučí detekovat stejný vzor bez ohledu na jeho polohu v obraze.

**Typy vrstev CNN**

* **Convolutional Layer**: Aplikuje se filtry na obraz, aby extrahovala různé vlastnosti.
* **Pooling Layer**: Obvykle následuje po konvoluci a zmenšuje rozměry, což redukuje parametry a zvyšuje robustnost.
* **Flatten Layer**: V CNN prochází obraz řadou konvolučních a pooling vrstev, až se výstupní obraz „rozloží“ do jednorozměrného pole, které se dá zpracovat v plně propojených (dense) vrstvách.

**Struktura CNN: Populární architektury**

* **VGGNet**: Používá malý počet větších vrstev, což zjednodušuje architekturu a často zlepšuje výkon pro specifické úlohy.
* **ResNet** (Residual Networks): Obsahuje speciální „skip“ (přeskočené) vrstvy, které obcházejí jednu nebo více vrstev. Pomáhá řešit problémy s mizejícím gradientem v hlubokých sítích.

**4. Rekurentní neuronové sítě (RNN): Detailní přehled**

**BPTT (Backpropagation Through Time)**

* Trénink RNN probíhá pomocí algoritmu zpětné propagace skrze čas. Model využívá informace z předchozích časových kroků, čímž rozpoznává dlouhodobé závislosti v datech.

**LSTM a GRU v praxi**

* **Long Short-Term Memory (LSTM)** používá pro každý časový krok tři brány (vstupní, zapomínací a výstupní) a interní paměťový blok, který uchovává informace. Pomáhá předcházet problémům s gradientem, které běžně vznikají u RNN.
* **Gated Recurrent Unit (GRU)** je podobný LSTM, ale má méně parametrů a může být efektivnější.

**5. Optimalizace a regularizace v detailu**

**Typy regularizací**

* **Dropout Layer**: Náhodně vypíná neurony, čímž modelu brání v přeučení na tréninkových datech.
* **Batch Normalization**: Normalizuje výstupy z každé vrstvy, což stabilizuje a urychluje trénink.

**Hyperparametry a jejich význam**

* **Velikost dávky (batch size)**: Má vliv na rychlost tréninku a stabilitu gradientu. Malé dávky generují hlučnější gradienty, což může mít pozitivní vliv na generalizaci.
* **Learning rate scheduler**: Dynamicky mění rychlost učení během tréninku, což umožňuje rychlejší konvergenci nebo se vyhnout lokálním minimům.

**6. Pokročilé techniky s TensorFlow**

**Vlastní tréninkové smyčky**

* TensorFlow umožňuje vytvořit vlastní tréninkovou smyčku, kde lze přímo ovlivňovat proces trénování, například ruční implementaci mini-batch gradient descent nebo metody řízení rychlosti učení.

**Distribuované učení**

* **Data parallelism**: Data jsou rozdělena mezi více procesorů (GPU/TPU), každý zpracovává část dat a aktualizuje váhy na základě výsledků.
* **Model parallelism**: Rozdělí se samotný model mezi různé výpočetní jednotky, což umožňuje trénink extrémně velkých modelů.

**7. Případové studie a aplikace**

**Praktická implementace v Keras**

* **Obrazová klasifikace**: Příklady zahrnují trénink CNN na datasetu MNIST, což je jednoduchý dataset ručně psaných číslic. Naučí tě vytvořit CNN od nuly, definovat vrstvy, trénovat model a hodnotit výkon.
* **Sekvenční modelování v NLP (Natural Language Processing)**: Například sekvenční klasifikace, jako je sentimentová analýza nebo strojový překlad. Tyto modely ukazují, jak pracovat s textovými daty a jaký dopad má volba hyperparametrů na jejich výkon.

**GANs (Generativní Adversariální Sítě)**

* GANs se skládají ze dvou sítí – generátor a diskriminátor – které proti sobě soupeří. Generátor se učí vytvářet realistická data, zatímco diskriminátor hodnotí, zda jsou tato data skutečná, nebo generovaná.
* **Trénink GANs**: Je náročný, protože vyžaduje balancování mezi generátorem a diskriminátorem tak, aby dosáhly co nejlepší spolupráce bez kolapsu tréninkového procesu.