Kunskapskontroll – Teoretiska Frågor

Besvara nedanstående frågor kort och koncist.

1. Hur är AI, Maskininlärning och Deep Learning relaterat?

Deep Learning är en delmängd av Maskininlärning, som i sin tur är en delmängd av AI. AI är ett övergripande koncept som strävar efter att skapa intelligenta system. Maskininlärning är en del av AI som syftar till att skapa modeller som genom identifikation av mönster i data, kan lära att utföra uppgifter som tex. prediktionsuppgifter. I traditionell AI, var det programmerare som specificerade reglarna för hur datorn skulle utför uppgifterna, medan det i maskininlärning är datorn som genom data skapar regelverket för hur uppgiften ska uppföras.

Deep Learning är en specifik teknik inom maskininlärning som är byggd på konceptet om artificiella neurala nätverk, som är inspirerad av neurala närverk i hjärnan. Inspirationen från de biologiska neurala nätverken avspeglas i Deep Learning-modellernas arkitektur som är byggd av en mängd lager bestående av neuroner (noder). En arkitektur som kan vara omfattande och komplex.

2. Hur är Tensorflow och Keras relaterat?

Keras är ett high-level API (Application Programming Interface) utvecklad för att göra det enkelt att skapa, träna och utvärdera deep learning modeller. Att den är high-level innebär att den är utformad för att vara användarvänlig, till skillnad från en low-level API där användaren har mer flexibilitet, men API:et är mer komplext att arbeta med och kräver djupare kunskaper om systemet och programmering.

För att använda Keras måste den köras genom ett backend-system, där det finns flera att välja mellan, och Tensorflow är ett av dessa. Tensorflow, som är utvecklat av Google, är ett open source-bibliotek som möjliggör körning av deep learning modeller.

3. Vad är en parameter? Vad är en hyperparameter?

En modells parametrar består av antalet vikter i modellen plus antalet bias termer. Till exempel, i ett Dense-lager med 10 noder och en input på 784 (28x28 pixlar som i MNIST), skulle det finnas 10x784 vikter plus en bias-term för varje nod. Det totala antalet parametrar i det lagret blir därför (10x784) + 10 = 7850. Detta beror på att varje nod i lagret är kopplad till varje input via en vikt. Dessa vikter, tillsammans med bias-termerna, utgör lagrets parametrar.

Genom att köra model.summary() får man en översikt över antalet parametrar i de olika lagren och det totala antalet för hela modellen. Parametrarna justeras under modellens träningsprocess, bland annat genom backpropagation, och deras värde styrs av de hyperparametrar som finns i modellen.

Hyperparametrar i Deep Learning-modeller är de variabler som kan finjusteras för att förbättra modellens prestanda. Dessa inkluderar till exempel antalet lager i modellen, antalet neuroner i varje lager, aktiveringsfunktioner, learning rate, optimzers, batch_size, epoch med mera. Genom att justera dessa hyperparametrar kan modellen optimeras. Den optimala learning rate baseras på värdena av de andra hyperparametrarna, varför learning rate bör justeras om de andra hyperparametrarna ändras.

4. När man skall göra modellval och modellutvärdering så kan man använda ett tränings, validerings och test data. Förklara hur de olika delarna kan användas.

Träningsdatan används för att träna modellen, det vill säga att hitta mönster i datan som gör att modellen kan göra prediktioner på nya data. Valideringsdatan används för att utvärdera modellens prediktionsförmåga och är en viktig del av processen för att finjustera hyperparametrarna. På så sätt kan man testa modellen på valideringsdatan, justera hyperparametrarna och göra nya tester tills man är nöjd med hur modellen prediktera på både validerings- och träningsdatan. När man har hittat den slutgiltiga modellen används testdatan för att göra den sista utvärdering av modellen. Detta steg är viktigt då det visar modellens prediktionsförmåga på nya data.

5. Förklara vad nedanstående kod gör:

```
n_cols = X_train.shape[1]

nn_model = Sequential()
nn_model.add(Dense(100, activation = 'relu', input_shape = (n_cols, )))
nn_model.add(Dropout(rate=0.2))
nn_model.add(Dense(50, activation = 'relu'))
nn_model.add(Dense(1, activation = 'sigmoid'))

nn_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

early_stopping_monitor = EarlyStopping(patience = 5)
nn_model.fit(X_train, y_train, validation_split = 0.2, epochs = 100, callbacks = [early_stopping_monitor])
```

Funktionen shape returnerar antalet rader och kolumner i träningsdatan, och genom att använda X_train.shape[1] får vi antalet kolumner (features) som lagras i n_cols.

Nästa steg är att bygga modellen, som är en Sequential model. Det innebär att modellen har lager som är staplade på varandra och är kopplade sekventiellt. Det första lagret är ett Dense-lager, där varje nod är kopplad till varje input nod, vilket skapar en tätt sammanvävd struktur i modellen. Det finns 100 noder i detta lager, och aktiveringsfunktionen är ReLU. Input_shape brukar specificeras i det första lagret så att modellen vet vilken form inputen har, detta görs för att modellen ska veta hur många vikter det första lagret ska ha. I detta fall används n_cols, som är antalet kolumner i datan.

Det nästa lagret är ett Dropout-lager med en dropout-rate på 0.2. Det innebär att 20% av outputen från det föregående lagret sorteras bort och inte går vidare till nästa lager. Detta görs för att förhindra överanpassning.

Därefter följer ett Dense-lager med 50 noder och ReLU som aktiveringsfunktion. Slutligen finns ett Dense-output lager med 1 output och sigmoid som aktiveringsfunktion. Sigmoid-funktionen används när man vill ha en eller flera binära klassificeringar som output.

När modellen är byggd kompileras den genom att specificera vilken optimerare (i detta fall "adam"), vilken loss-funktion och vilken metric som ska användas för att utvärdera modellen. Eftersom sigmoid används i outputlagret, används "binary_crossentropy" som loss-funktion och "accuracy" som metric.

En "early_stopping_monitor" har initialiserats med patience på 5. Monitor argumentet är inte initierat, så funktionen ser på default måttet, som är valideringsfelet. Det innebär att träningen stoppas om valideringsfelet inte minskar under 5 epoker.

Slutligen tränas modellen med .fit metoden på träningsdatan, där 20% av datamängden reserveras som valideringsset. Modellen tränas i högst 100 epoker, där varje epok innebär en iteration över hela datamängden. Callbacks-argumentet är inställt på "early stopping", vilket definierades tidigare.

6. Vad är syftet med att regularisera en modell?

Syftet är att hindra överanpassning av modellen på träningsdatan. Det kan vara väldigt många parametrar i en deep learning modell, vilket kan leda till överanpassning. Om modellen passar för bra på träningsdatan, kommer den inte prestera bra på nya data. Då syftet är att hitta en modell som kan generalisera och prediktera bra på nya data, då är det essentiellt att hindra överanpassning.

7. ”Dropout” är en regulariseringsteknik, vad är det för något?

Dropout används för att hindra överanpassning. Intentionen är att regularisera modellen genom att den sorterar bort en andel av noder från ett lager genom att sätta nodernas output till 0. En nods output selekteras bort efter ett probabilitets gränsvärde som sätt när man skapa drop out lagret, detta kallas dropout rate.

Dropout utförs efter varje viktuppdatering dvs. varje träningstillfälle. Konsekvensen av att ta bort vissa noders output är att modellen tvingas att hitta mönster i den resterande data. Regulariseringen fungerar genom att göra modellen mindre sensitiv för ändringar i få noder, och därmed hindrar överanpassning och förhoppningsvis förbättra prediktionerna på nya data.

8. ”Early stopping” är en regulariseringsteknik, vad är det för något?

Early Stopping används för att stoppa träningen när modellen inte längre registrerar en förbättring av prediktionerna på valideringssetet. Hur många epoker som ska köras där ingen förbättring syns, bestäms av argumentet patience som man kan definiera när man initierar en early stoping callback. Early stopping hindrar både överanpassning på träningsdatan och det sparar tid och datorkraft genom att stanna träningen när det inte syns någon förbättring.

9. Din kollega frågar dig vilken typ av neuralt nätverk som är populärt för bildanalys, vad svarar du?

Det är Conventionel Neural Networks (CNNs). En CNN modell består av minst ett convolution lager. Et convolutional lager är gjort för att kunna hantera input i flera dimensioner motsatt ett Dense lager som tar en flatten array som input. Detta gör att ett convolutional lager kan processa bilder där pixlarna finns bredvid varandra och kan därmed förstås i relation till varandra. Denna relation beräknas genom att inputen körs genom en kernel, som kan ha olik astorlekar (tex. 3x3), och som på detta sätt kan ”se” inputen i relation till varandra.

10. Förklara översiktligt hur ett ”Convolutional Neural Network” fungerar.

CNN används speciellt för bildigenkänning och hantering av visuella data. Bilderna blir ”convoluted” genom en kernal i de dolda convolutional lagrar i modellen. Denna kernel kan bidra till att dra ut mönster i datan, som vertikala och horisontala linjer. Typiskt hittar de första lager grova och simpla mönster, medan de senare lagar i en CNN modell, kan hitta mera komplexa mönster i datan.

De convolutional lagrar är ofta följd av ett pooling lager, där max pooling är mest använd. Här kollar man tex. på ett pooling lager i storlek 2x2 och tar det högsta värde i denna fyrkant. Man kan också använda AvgPooling, som då tar ett genomsnitt av värdarna. Genom att använda ett Pooling lager då minimeras det input som förs vidare till nästa convolution lager. Det bidrar därmed både till att minimera inputen och till att lyfta fram de mönster som det convolutional lager tog fram.

Jo fler convoluted lager som körs, ju mera komplexa mönster kan modellen identifiera. Till sist i modellen är det ofta att man lägger till ett flatten lager som plattar ut datan tilll en array, som kan köras som input i ett vanligt Dense lager innan outputlagret.

11. Din vän har ett album med 100 olika bilder som innehåller t.ex. tennisbollar och zebror. Hur hade han/hon kunnat klassificera de bilderna trots att han/hon inte har någon mer data att träna en modell på?

Man kan använda transfer leraning som är en metod där man tar en modell som redan är tränad på liknande bilder, som tex. ResNet50 modellen. Det finns manga pretrained modeller som man med fördel kan använda istället för att träna sin egen modell från början. Fördelen är att de ofta har tränads på en stor mängd data och har en komplex arkitektur som har tagits fram genom mycket testande.

Man kan ladda ner en pretrained modell och frysa vikterna, men ta bort outputlagret. Man kan på så sätt använda lärdomen från modellen, men själv bestämma vilket output (tennisbollar, zebror) som modellen ska kunna klassificera.

12. Vad gör nedanstående kod?

```
1  model.save('model_file.h5')
```

```
1  my_model = load_model('model_file.h5')
```

Den första kod används när man ska spara en deep learning modell, medan den andra kodsnippen används när man vill ladda upp den sparade modell. Detta är väldigt viktigt när man jobbar med deep learning modeller som tar lång tid att träna. Genom att spara modellerna kan de används utan att man behöver träna modellen igen.

13. Deep Learning modeller kan ta lång tid att träna, då kan GPU via t.ex. Google Colab skynda på träningen avsevärt. Läs följande artikel: https://blog.purestorage.com/purely-informational/cpu-vs-gpu-for-machine-learning/ och skriv mycket kortfattat vad CPU och GPU är.

CPU står för Central Processing Unit och är datorns processor som hanterar både hardware- och software-instruktioner. Den är ofta multicore, vilket betyder att den kan hantera flera instruktioner samtidigt på ett antal processorkärnor. Instruktioner och beräkningar i en CPU körs sekventiellt.

GPU står för Graphics Processing Unit. Det är också en processor, men den är speciellt anpassad för att hantera visuella data. Numera används den även för att hantera stora och komplexa uppgifter inom analys och maskininlärning. En GPU delar upp uppgifter i mindre delar och distribuerar dem till en stor mängd kärnor, ofta tusentals. Till skillnad från CPUs sekventiella hantering kan en GPU hantera uppgifter parallellt. Den parallella hanteringen och den stora mängden processorkärnor i en GPU bidrar till ökad beräkningshastighet och gör den väl anpassad för träning av deep learning-modeller, som både tränas på stora mängder data och utför komplexa matematiska beräkningar.