

Datorlaboration 5

Josef Wilzén

19 september 2024

Allmänt

Datorlaborationerna kräver att ni har R och Rstudio installerat.

- Exempelkod för neurala nätverk: [här](#)
- Kodmanual: [länk](#)
- **ISL**: An Introduction to Statistical Learning,
 - Boken: [länk](#)
 - R-kod till labbar: [länk](#)
 - Dataset: [länk](#) och [länk](#)
- **IDM**: Introduction to Data Mining
 - Kod till boken finns [här](#)
 - Sample chapters
- Dataset till vissa uppgifter finns [här](#). mnist-data finns på [Lisam](#).

Notera att ni inte behöver göra alla delar på alla uppgifter. Det viktiga är att ni får en förståelse för de olika principerna och modellerna som avhandlats. Dessa uppgifter ska inte lämnas in, utan är till för er övning.

Datauppdelning

För att motverka överanpassning bör ni dela upp data till träning-, validering-, (och testmängd). Detta kan göras med `createDataPartition()` från `caret`-paketet. Argument till den funktionen som är av vikt här är p som hur stor andel av observationerna som ska användas till träningsmängden. Ni kan också använda `subset()` för att göra detta också, men det blir svårare att tydligt ange de observationer som ska tilldelas till valideringsmängden. Denna uppdelning ska ske slumpmässigt. Notera att om en testmängd ska skapas måste uppdelningen ske en gång till från valideringsmängden.

Keras

- Se CHEAT SHEET för Keras. Dokumentation för keras finns [här](#):
 - R interface to Keras
 - TensorFlow/Keras
 - Lista över funktioner i Keras
- Utvärdering vid klassificering kan göras med funktionen `class_evaluation_keras()`.

Del 1: Optimiering av neurala nätverk

Här kommer vi utgå från Fashion MNIST-data, kolla [här](#). Läs in datamaterialet och se till att ha koll på vad det är för sorts data. Nu ska ni testa olika inställningar på optimeringen.

1. Definera följande modell i keras:

```
> model <- keras_model_sequential()
> model %>%
+   layer_flatten(input_shape = c(28, 28)) %>%
+   layer_dense(units = 128, activation = 'relu') %>%
+   layer_dense(units = 128, activation = 'relu') %>%
+   layer_dense(units = 10, activation = 'softmax')
```

2. Fixera antal epoker till 30. Sätt den globala learning rate till 0.01. Sätt batchstorlek till 128.
3. Hur många parametrar har modellen?
4. Testa nu följande inställningar. Undersök tränings- och valideringsdata under träning och utvärdera på testdata. **Tips:** titta [här](#) vid behov.
 - (a) Skatta modellen med vanlig SGD
 - (b) Skatta modellen med Adam algoritmen. Använd defaultvärden på övriga hyperparametrar.
 - (c) Skatta modellen med RMSPROP algoritmen. Använd defaultvärden på övriga hyperparametrar.
 - (d) Vilken metod är ni mest nöjd med?
5. Gör om 4) men med batchstorlek 64 och 256. Hur påverkar det optimeringen?
6. Välj batchstorlek 128 och SGD. Ändra learning rate till 1 och 0.0001, vad händer?
7. Upprepa 6) men med Adam eller RMSPROP
8. Välj batchstorlek 128 och SGD. Testa att ändra startvärdena. Detta görs i lagerfunktionerna. Kolla i dokumentationen hur ni ska göra. Testa att låta startvärdena vara väldigt små (men ej exakt noll) och rätt stora. Hur blir resultatet?
9. Batch normalization: Definiera modellen nedan. Skatta modellen med Adam, learning_rate=0.01, epochs = 30, batch_size=128. Hur blir resultatet? Undersök tränings- och valideringsdata under träning och utvärdera på testdata.

```
> model_batch_norm <- keras_model_sequential()
> model_batch_norm %>%
+ layer_flatten(input_shape = c(28, 28)) %>%
+ layer_dense(units = 128, activation = 'relu') %>%
+ layer_batch_normalization() %>%
+ layer_dense(units = 128, activation = 'relu') %>%
+ layer_batch_normalization() %>%
+ layer_dense(units = 10, activation = 'softmax')
```

10. Välj en metod från 4) och skatta modellen med 50 epoker.
11. Av alla metoder för optimering ni testat, vad tycker ni har funkat bäst? Har det varit stor skillnad?

Del 2: Neurala nätverk: regularisering

Kör denna tutorial: [länk](#).

Del 3: Faltade nätverk och MNIST data

Kör denna tutorial: [länk](#). Jämför med er bästa MLP-modell (förra labben) på samma dataset.

Del 4: Faltade nätverk och CIFAR10 data

Kör denna tutorial: [länk](#). Ni ska här jobba med CIFAR10 datasetet, som är ett klassiskt dataset inom maskininlärning. För mer info, se [här](#) och [här](#).

Notera att detta är färgbilder som har tre kanaler (grön, röd, blå) och att beräkningarna blir mycket tyngre. **Läs steg 1 och 2 nedan innan börjar skatta modeller.**

1. Beroende på hårdvaran i er dator så kan det vara bra att minska ner storleken på datasetet. Testa att börja med hälften av observationerna. Sedan kan ni använda fler om er dator klarar av det.

```

> cifar0 <- dataset_cifar10()
> no_obs<-dim(cifar0$train$y)[1]
> no_obs_test<-dim(cifar0$test$y)[1]
> set.seed(34)
> index<-base::sample(no_obs,size = no_obs/2)
> index2<-base::sample(no_obs_test,size = no_obs_test/2)
> cifar<-cifar0
> cifar$train$x<-cifar0$train$x[index,,]
> cifar$train$y<-cifar0$train$y[index,]
> cifar$test$x<-cifar0$test$x[index2,,]
> cifar$test$y<-cifar0$test$y[index2,]
> rm(cifar0)
> # kolla så att klasserna är hyfsat balanserade:
> table(cifar$train$y)
> table(cifar$test$y)

```

2. Börja med att träna i max 5 epoker när ni skattar modellen. Sen kan ni öka antalet när ni ser hur lång tid beräkningarna tar.
3. Vad är det för sorts data ni jobbar med här? Hur många obs? Hur stora bilder? Vilka klasser finns?
4. Skatta den föreslagna modellen (OBS kan ta lång tid). Hur många parameterar har den?
5. Hur presterar modellen på testdata? Beräkna förväxlingsmatrisen för testdata.
 - (a) Vilken klass var lättast att klassificera?
 - (b) Vilken klass var svårast?
6. Testa att ändra arkitekturen på modellen.
 - (a) Ändra de faltade lagren:
 - i. Antal lager, varning: ta ej för många
 - ii. Ändra antal och storlek på **filters** i faltade lagren, varning: ta ej för många
 - iii. Ändra **pool_size** i pooling-lagren.
 - iv. Testa average pooling
 - (b) Testa att modifiera de fullt kopplade lagren.
7. Testa att ändra optimeringen
8. Hur presterar den bästa modellen som ni lyckas hitta?
9. Gå in [här](#) och se hur de bästa modellerna presterar på CIFAR10 och MNIST.

Del 5: Autoencoders och Functional API

I denna del ska ni testa att skatta autoencoders. Ni även testa att använda functional API i Keras (istället för Sequential model som ni testat innan). Functional API är ett gränssnitt för konstruera modeller som är mer avancerade och där allt inte bara är kopplat från lagret precis innan.

1. Autoencoder och spambase data: Målet är nu att testa att träna en autoencoder datasetet spambase. Vi kommer att använda Keras functional API, för mer detaljer, se steg 3) nedan. Datasetet har 57 förklarande variabler och en kategorisk responsvariabel (Spam). För detaljer kring data, se denna [länk](#).
 - (a) Läs data och transformera data enligt koden: [här](#) (finns kod för skattning av modeller mm)
 - (b) Skatta följande modeller som grund för vidare analys. Använd alla förklarande variabler och ha Spam som responsvariabel. Använd all data som träningsdata. Ta fram träffsäkerhet på träningsdata.
 - i. Vanlig linjär logistik regression

- ii. Random forest, använd 100 träd.
 - iii. MLP, ha lager: 100 noder + 100 noder och sen en klassificeringsnod. Ni väljer övriga inställningar själva.
- (c) Vi ska nu utgå från de 57 förklarande variablerna. Målet är att få fram en transformation/kod av dessa som har en låg dimension, men som ändå innehåller relevant information för klassificering av spam från de ursprungliga variablerna. Skatta den föreslagna autoencoder-modellen. Beskriv hur encoder och decoder ser ut. Vårt flaskhalslager har här 2 noder¹. Så vi gör en icke-linjär variabelreduktion från 57 variabler till 2 variabler.
- (d) Plotta data i de två nya variablerna (baserat på flaskhalslagret), färglägg punkterna baserat på Spam-variabeln. Detta får vi genom att göra prediktioner med encoder-delen av modellen. Verkar de nya variablerna vara meningsfulla? Notera att Spam inte har använts för att göra variabelreduktionen.
- (e) Testa nu att skatta några modeller där ni har de två nya variablerna som förklarande variabler och Spam som respons. Ta fram träffsäkerhet för träningsdata. Hur bra blir modellerna? Jämför med b) ovan. Verkar de nya variablerna vara användbara eller ej?
- i. Vanlig logistik regression
 - ii. KNN, där ni väljer K med korsvalidering från en lista av värden (5,7, 9, ... ,21).
 - iii. MLP, ha lager: 50 noder + 50 noder. Ni väljer övriga inställningar själva.
- (f) Bonus: Gör PCA på de 57 förklarande variablerna, och spara de två första komponenterna i ett dataset. Hur mycket av datas varians förklarar de två första variablerna tillsammans? Skatta modellerna i e) med de två första komponenterna som förklarande variabler. Jämför med modellerna i b) och e).
- (g) Bonus: Gör om c), men flaskhalslagret ha 3 noder och testa även att ha 5 noder. Notera att det är mycket svårare att analysera det nya variabelrummet är bra/meningsfullt när vi har mer än 2 noder. Vi kan kolla på parvisa punktdiagram, men det kommer inte att ge hela bilden av de nya variablerna, men kan lära oss nåt.
2. Denoising autoencoder: Kör denna tutorial: [länk](#), minska ev antal epoker om det tar lång tid att träna modellerna. Målet är att återskapa rena bilder baserat på data med brusiga bilder.
3. Functional API: kolla på den tutorial.
- (a) Gå igenom "Setup" till "All models are callable, just like layers"
 - (b) Kolla på "Manipulate complex graph topologies" till "Shared layers". Vissa delar här ligger utanför kursen (tex hantering av text, embeddings, LSTM etc). Kolla mer på koncepten på hur mer avancerade arkitekturer kan skapas. Testa att återkskapa modellen under "A toy ResNet model" och försök skatta den på CIFAR10 eller fashion MNIST (obs kan ta lång tid).

Del 6: Mer faltade nätverk

Återvänd till Fashion MNIST-data, kolla [här](#). Läs in datamaterialet och se till att ha koll på vad det är för sorts data.

1. Skapa följande modeller:
 - (a) Faltat lager + pooling + två fullt kopplade lager
 - (b) Faltat lager + pooling + faltat lager + pooling + två fullt kopplade lager
 - (c) En egen kombination av lager, men med minst ett faltat lager.
2. Skatta de föreslagna modellerna. Hur många parameterar har de?
3. Hur presterar modellerna på testdata? Beräkna förväxlingsmatrisen för testdata för de olika modellerna.
 - (a) Vilken klass var lättast att klassificera?

¹Dvs de minsta lagret som ligger i mitten av modellen. Två noder gör det lätt att visualisera de nya transformerade variablerna i ett spridningsdiagram.

- (b) Vilken klass var svårast?
- Om ni vill: testa att lägga till någon regularisering till den bästa modellen i 1). Hur blir prediktionen på testdata?
 - Jämför med er bästa modell med bara fullt kopplade lager på detta dataset. Vilken presterar bäst? Hur stor är skillnaden? Hur skiljer sig antalet parametrar åt?
 - Utgå från MNIST och försök skapa en model som kopplingar som hoppar över vissa lager. Använd Functional API.