www.math-stockholm.se/cirkel

21 april 2022

Välkomna!

Idag:

- Jobbar med **Keras** i **TensorFlow** för att approximera funktionen $f(x) = |x - \frac{1}{2}|^2$ med en funktion $\alpha_{\boldsymbol{\theta}}$ sådant att

$$\begin{aligned} \alpha_{\boldsymbol{\theta}}(x) &= \sum_{k=1}^{K} \tilde{w}_{1,k}^{(0)} \sigma\left(w_{1,k}^{(0)} x + b_k^{(0)}\right) \\ \boldsymbol{\theta} &= [\tilde{w}_{1,k}^{(0)}, w_{1,k}^{(0)}, b_k^{(0)}], k = 1, \ldots, K \end{aligned}$$

- $\boldsymbol{\theta} \in \mathbb{R}^{K \times 3}$
- x är ingångsnoden
- σ är aktiveringsfunktion
- $\boldsymbol{\theta}$ representerar vikter och bias
- $K = 10$ eftersom vi har 10 noder i det dolda lagret

Keras och TensorFlow

- Vi gör inga beräkningarna (vikterna, bias och noderna) själva
- Använder befintlig programvara
- TensorFlow: ett programvarubibliotek av öppen källkod för maskininlärning
- Det har utvecklats och används av Google
- Keras är ett djupt lärande **API** (eng: *application programming interface*) skrivet i Python
- API:er gör det möjligt att återanvända redan utvecklad och kvalitetssäkrad mjukvara som är i form av kodbibliotek
- Keras är enkelt, flexibelt och kraftfullt

Kod

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

- Importera nödvändiga bibliotek

Kod

- `N` är det totala antalet datapunkter (träningsdata och testmängden tillsammans)
- `K` är hur många noder vi har i det första dolda lagret

```
N=1000
K=10
```

- Vi måste ta tillräckligt många steg av stokastisk gradientnedstigning så att algoritmen konvergerar
- `M` är hur många steg

```
M=40000
```

Kod

- `dt` är inlärningshastighet (parametern γ i Algoritm 2)
- Vi tar ett litet värde här (steglängd)
- Stokastisk gradientnedstigning försiktigt tar små steg mot minimipunkten

```
dt=0.008
```

Kod

- Stokastisk gradientnedstigning approximerar gradienten i Algoritm 2 med **en punkt** x_i där i väljs slumpmässigt
- Ett alternativ är att approximera gradienten på mer än en punkt och därefter beräknar genomsnittet

Välj slumpmässigt k parametrar i mängden $\{1, \ldots, N\}$

$$\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j - \gamma \left(\frac{1}{k} \sum_{i=1}^{k} \nabla_{\boldsymbol{\theta}} \left(|\alpha_{\boldsymbol{\theta}_j}(x_i) - f_i| \right)^2 \right)$$

- Vi tar bara en punkt som beskrivs av variabeln `Nbatch`

```
Nbatch=1
```

Kod

- De allra flesta datapunkter används som träningdata
- Följande parameter anger kvoten mellan träningsdatan och testmängden

```
validation_split_ratio=0.2
```

Kod

- En epok (eng. *epoch*) är en hyperparameter som refererar till hur många gånger algoritmen kommer att arbeta genom hela träningsdatasetet

```
M*Nbatch=(N*(1-validation_split_ratio))*EPOCHS
```

- Vänstra sidan motsvarar det totala antalet datapunkter stokastisk gradientnedstigning kommer att hantera
- Då löser vi ekvationen för `EPOCHS`
- Notera att `np.int` förvandlar ett decimaltal till ett heltal

```
EPOCHS=np.int64(M/(N*(1-validation_split_ratio)/Nbatch
    ))
```

Kod

- Vi ställer in slumptalsgeneratorn så att vi får samma datapunkter varje gång vi kör koden
- Detta gör det möjligt att återge våra resultat

```
np.random.seed(123)
tf.random.set_seed(124)
```

- Vi tar ett stickprov av N punkter från en likformig fördelning på intervallet $(-4, 4)$

```
xs = np.random.uniform(-4,4,size=(N,1))
```

Kod

- Definierar funktionen f

```
def f(x):
    return np.square(np.abs(x-0.5))
```

- Vi evaluerar funktionen f på träningsdatan och testmängden

```
ys=f(xs)
```

Kod

- Definiera inputlagret med en ingångsnod
- Definiera det dolda lagret med `K` noder
- Sigmoid aktiveringsfunktionen
- Initialisera vikterna enligt en normalfördelning
- Bias satt till noll

```
Input_layer=tf.keras.Input(shape=(1,))
Hidden_layer= keras.layers.Dense(units=K,
                    activation="sigmoid",
                    use_bias=True,
                    kernel_initializer='random_normal',
                    bias_initializer='zeros')
```

Kod

- Outputlagret med en utgångsnod och inga bias

```
Output_layer=keras.layers.Dense(units=1,use_bias=False
    )
```

Kod

- Ange ordningen på lagren
- Välj **stokastisk gradientnedstigning** som den iterativa metoden med inlärningshastighet `dt`
- Förlustfunktionen är inställd på medelkvadratfel

```
model=keras.Sequential([Input_layer,Hidden_layer,
    Output_layer])
optimizer=tf.keras.optimizers.SGD(learning_rate=dt)
model.compile(optimizer=optimizer,loss='
    mean_squared_error')
```

Kod

- Träna modellen på de angivna parametrarna

```
history=model.fit(x=xs,
                  y=ys,
                  batch_size=Nbatch,
                  epochs=EPOCHS,
                  validation_split=validation_split_ratio,
                  verbose=1)
```

Kod

- Skapa en lista med ekvidistanta (jämnt utspridda) punkter mellan -4 och 4
- Evaluera funktionen f på dem

```
pts = np.linspace(-4,4,300).reshape(-1,1)
target_fcn_vals=f(pts)
```

Kod

- Att ställa in `verbose=1` visar oss en utskrift av förlustfunktionen på vår skärm när algoritmen körs
- Noterar att utskriften visar
  - `loss` (förlustfunktionen evaluerad på träningsdata)
  - `val_loss` (värdet av förlustfunktionen evaluerad på testmängden)

Kod

- Evaluera modellen som algoritmen har skapat på samma punkter

```
alpha_vals=model(pts)
```

Kod

- Skapa en graf av funktionen f och modellens approximation $\alpha_{\boldsymbol{\theta}}$

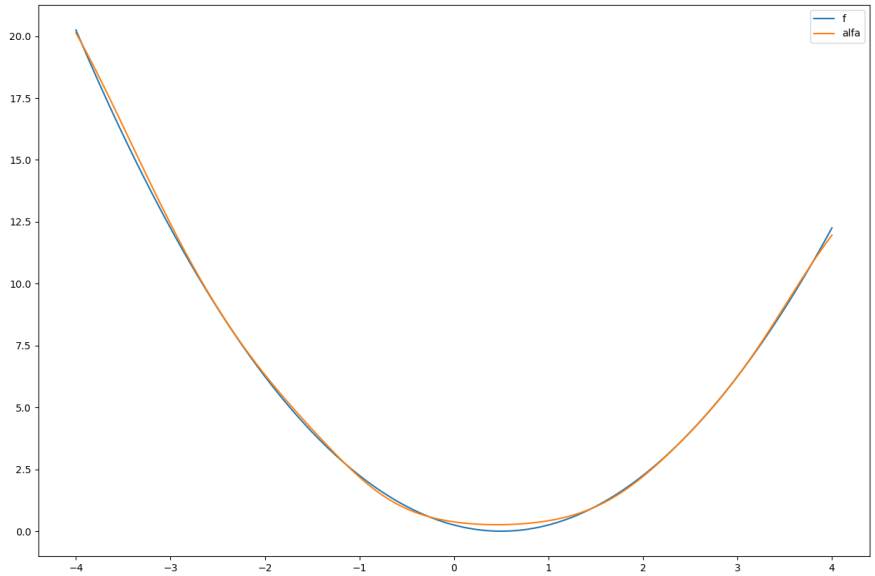
```
plt.figure('Alfa',figsize=(15,10))
plt.plot(pts,target_fcn_vals,label='f')
plt.plot(pts,alpha_vals,label='alfa')
plt.legend(['f','alfa'])
plt.show()
```

Kod

- Skapa graf av träningsfelet och generaliseringsfelet

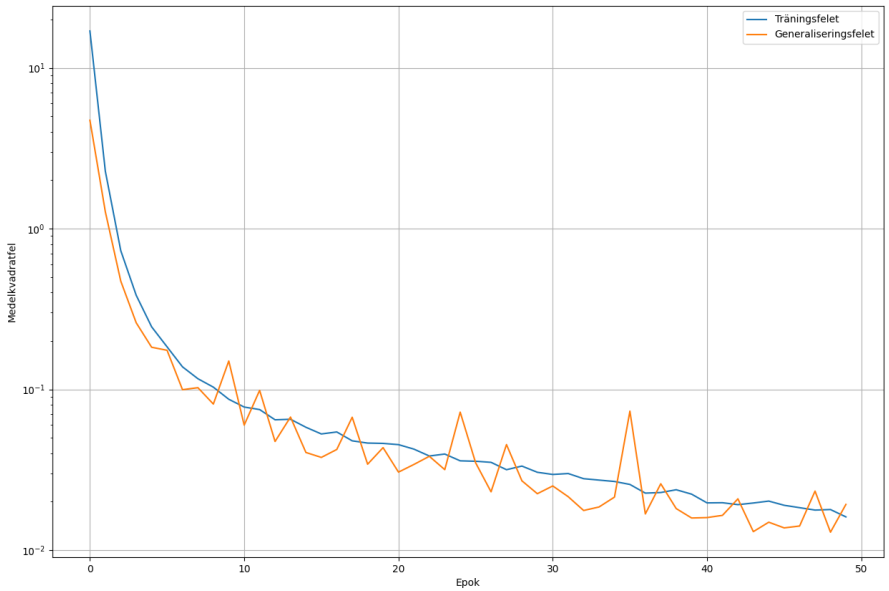
```
plt.figure('Fel',figsize=(15,10))
plt.semilogy(history.history['loss'],label='
    Traningsfelet')
plt.semilogy(history.history['val_loss'],
     label='Generaliseringsfelet')
plt.xlabel('Epok')
plt.ylabel('Medelkvadratfel')
plt.legend()
plt.grid(True)
plt.show()
```

Resultat



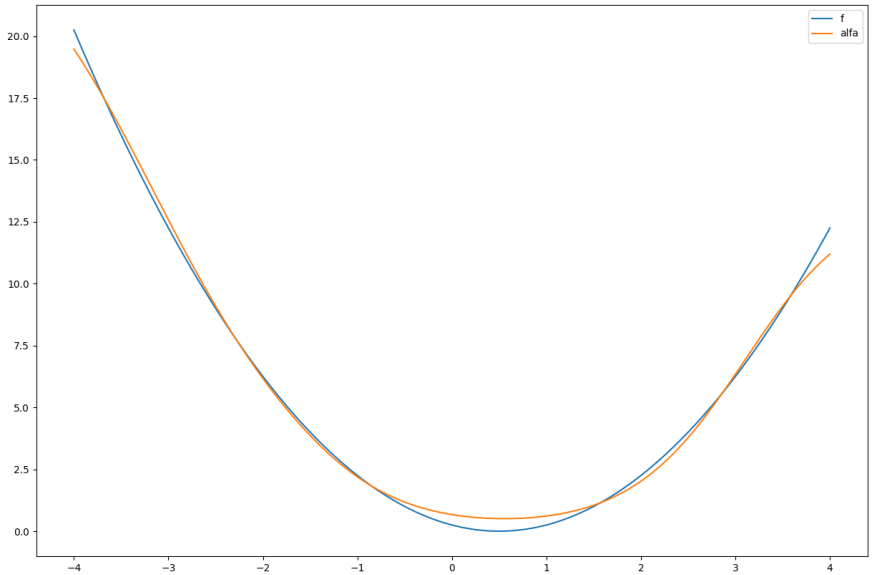
- Graf av funktionen f och algoritmens approximation

Resultat



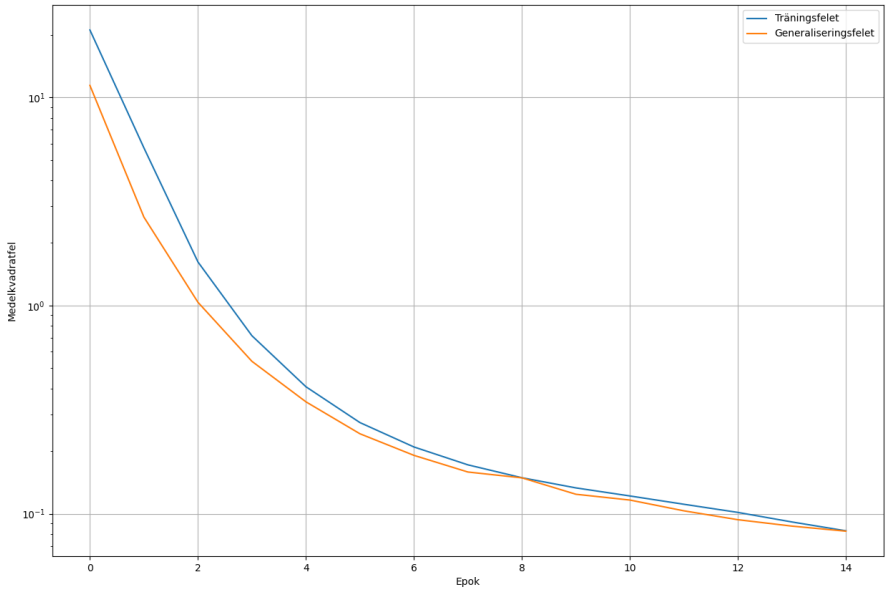
- Träningsfelet och generaliseringsfelet

Resultat



- Graf av funktionen f och algoritmens approximation
- Fler punkter
- Fler steg med stokastisk gradientnedstigning
- Långsammare inlärningshastighet

Resultat



- Träningsfelet och generaliseringsfelet
- Fler punkter
- Fler steg med stokastisk gradientnedstigning
- Långsammare inlärningshastighet

Sammanfattning

- Bara två exempel utförda med en **mycket enkel** modell
- I verkligheten finns det väldigt många mer komplicerade modeller som kan användas för att approximera mycket mer komplicerade fenomen
- Den här kursen fungerar endast som en introduktion men vi uppmuntrar er att fortsätta experimentera och fortsätta att söka efter ny information om detta ämne

Nästa gången

- Jag är i Tyskland
- Vi har gått igenom allt i kompendiet nu!
- Frågor? Zoom rum 5 maj, kl 16:00-17:15
- Enkät: `https://forms.gle/9m2MgqUHAviKkER48`
- Hjälper oss att skapa en bra kurs till nästa år

Tack för kursen!