Fusklapp för Python & Neurala Nätverk

Denna fusklapp är designad för att hjälpa dig navigera i de Python-koncept och de implementationer av neurala nätverk som används i denna kurs.

1. Grundläggande Python

Variabler och typer

Variabler sparar data. Inom maskininlärning använder vi mestadels siffror (floats/integers) och listor.

```
# Siffror
learning_rate = 0.1   # Float (decimaltal)
iterations = 100      # Integer (heltal)

# Strängar (Text)
name = "Perceptron"

# Booleans (Sant/Falskt)
is_training = True
```

Listor (Arrays)

Listor innehåller flera element. Vi använder dem för inputs, vikter och dataset.

```
# En lista med input-värden
inputs = [0.5, 1.0, -0.2]

# Komma åt element (Index börjar på 0!)
first_input = inputs[0]  # 0.5
last_input = inputs[-1]  # -0.2 (sista elementet)

# Lägga till i en lista
errors = []
errors.append(0.5)

# Längden på en lista
num_weights = len(inputs) # 3
```

Loopar

Loopar tillåter oss att upprepa instruktioner, till exempel för att gå igenom träningsdata eller vikter.

Loopa med index (`range`): Användbart när du behöver komma åt element på samma position i två olika listor (t.ex. input och vikt).

```
weights = [0.1, -0.5]
inputs = [1.0, 0.5]

for i in range(len(weights)):
    # i kommer vara 0, sen 1
    w = weights[i]
    x = inputs[i]
    print(w * x)
```

Loopa med `zip`: Det smidigaste sättet i Python att loopa över två listor samtidigt.

```
# input_example är en lista med inputs, target är det korrekta svaret
training_data = [[0, 0], [0, 1]]
targets = [0, 1]

for inputs, target in zip(training_data, targets):
    print(f"Tränar på {inputs}, förväntar mig {target}")
```

Funktioner

Kodblock som kan återanvändas.

```
def beräkna_summa(a, b):
    resultat = a + b
    return resultat

# Anropa funktionen
total = beräkna_summa(5, 10)
```

---

2. Klasser & Objektorienterad Programmering (OOP)

Vi använder klasser för att bygga våra neuroner och nätverk. En klass är en ritning för att skapa objekt.

Definiera en klass

```
class Neuron:
    # __init__ är "konstruktorn". Den körs automatiskt när du skapar ett
nytt objekt.
    # 'self' refererar till det specifika objektet som skapas/används.
    def __init__(self, num_inputs):
        self.weights = [] # Skapar en egenskap 'weights' specifik för
denna neuron
        self.bias = 0.0   # Skapar en egenskap 'bias'
```

Använda `self`

`self` tillåter funktioner inuti klassen att komma åt objektets data (som vikter).

```
    def predict(self, inputs):
        # Vi kan komma åt self.weights här eftersom de sparades i __init__
        return self.weights[0] * inputs[0]
```

Arv (Inheritance)

Skapa en specialiserad version av en klass.

```
# Perceptron ärver från Neuron (den får alla Neurons metoder)
class Perceptron(Neuron):
    def __init__(self):
        super().__init__() # Anropa förälderns setup först
        self.learning_rate = 0.1
```

---

3. Matematiska koncept för Neurala Nätverk

Viktad summa (Skalärprodukt)

Kärnan i en neurons beräkning: $\sum (input \cdot vikt) + bias$

Manuellt (Loop):

```
total = 0
for i in range(len(inputs)):
    total += inputs[i] * self.weights[i]
total += self.bias
```

Med Numpy (Snabbt & rent):

```
import numpy as np
# np.dot multiplicerar elementen och summerar dem automatiskt
total = np.dot(self.weights, inputs) + self.bias
```

Aktiveringsfunktioner

Bestämmer neuronens output.

Stegfunktion (Perceptron): Hårt avklipp. Output är 0 eller 1.

```
if total > 0:
    return 1
else:
    return 0
```

Sigmoid-funktion: Mjuk kurva. Output är mellan 0.0 och 1.0.

```
import math
def sigmoid(x):
    # Skydd mot overflow (math.exp kraschar på för stora tal)
    if x < -700: return 0
    if x > 700: return 1

    return 1 / (1 + math.exp(-x))
```

---

4. Vanliga uppgifter & mönster

Mönster för träningsloopen

Hur ett neuralt nätverk lär sig:

1. **Iterera** kontinuerligt (Epoker/Epochs).
2. **Prediktera** (Gissa) baserat på nuvarande vikter (Forward Pass).
3. **Jämför** gissning med facit (Fel/Error).
4. **Uppdatera** vikter för att minska felet (Backward Pass).

```
for epoch in range(100): # Upprepa 100 gånger
    for inputs, target in zip(data, targets):

        # 1. Gissa (Predict)
        prediction = model.predict(inputs)

        # 2. Beräkna fel (Error)
        error = target - prediction

        # 3. Uppdatera (Perceptron-regeln)
        # ny_vikt = gammal_vikt + (inlärningsfaktor * fel * input)
        for i in range(len(model.weights)):
            model.weights[i] += learning_rate * error * inputs[i]

        model.bias += learning_rate * error
```

Ladda data (Pandas)

Vanligt mönster som syns i notebooks.

```
import pandas as pd

# Ladda CSV-fil
df = pd.read_csv("data/penguins.csv")

# Visa de första raderna
print(df.head())

# Välj specifika kolumner
X = df[["bill_length_mm", "bill_depth_mm"]].values # .values ger oss en
numpy array/lista
y = df["species"].values
```

Slumptal

Används för att initialisera vikter.

```
import random

# Slumpmässigt decimaltal mellan -1.0 och 1.0
w = random.uniform(-1.0, 1.0)
```