Instruktioner - Datortentamen TDDD73 Funktionell och imperativ programmering i Python

Hjälpmedel

Följande hjälpmedel är tillåtna:

- Exakt en valfri bok, t.ex. den rekommenderade kursboken. Boken får ha anteckningar, men inga lösa lappar.
- Exakt ett A4-papper med egna anteckningar om precis vad som helst. Det kan vara hand- eller maskinskrivet, på ena eller båda sidorna.
- Pennor, radergummin och liknande för att kunna skissa lösningar på papper.

Icke tillåtna hjälpmedel inkluderar bl.a. alla former av elektronisk utrustning samt böcker och anteckningar utöver det som listats ovan.

Lösa tomma papper för att göra anteckningar tillhandahålls av tentamensvakterna.

Genomförande

Under datortentan kommer du att arbeta i en begränsad och övervakad miljö. Med hjälp av en särskild tentaklient skickar du in dina svar. Du kan även använda tentaklienten för att ställa frågor till examinator. Besök i skrivsalen kommer endast ske vid allvarligare tekniska problem.

Varje uppgift kan skickas in en enda gång och kommer därefter poängsättas. Det finns alltså inga möjligheter till komplettering.

Betygsättning

Datortentan består av totalt sex uppgifter som vardera kan ge maximalt fem poäng. Uppgifterna är utvalda för att ha olika svårighetsgrad och är ordnade så att den lättaste oftast kommer först. Observera att vad som är lätt eller svårt naturligtvis skiljer sig från person till person, så det är alltid en bra idé att snabbt läsa igenom alla uppgifterna för att bättre kunna prioritera arbetet. Betygssättningen sker enligt följande tabell:

- För betyg 3 krävs minst 12 poäng.
- För betyg 4 krävs minst 19 poäng.
- För betyg 5 krävs minst 25 poäng.

Rättningskriterier

Följande allmänna kriterier används vid poängsättning av uppgifter. Lösningar som bryter mot dessa kommer att få poängavdrag.

- Funktioner ska ha exakt samma namn som i uppgiften. Detta underlättar rättningen.
- Namn på parametrar, variabler och hjälpfunktioner som inte specificerats explicit i uppgiften ska vara beskrivande och följa namnstandarden.
- Lösningen ska vara välstrukturerad och väldokumenterad på samma sätt som kursens laborationer.
- Lösningen ska följa de regler som uppgiften har satt upp. Det kan t.ex. röra sig om att vissa funktioner eller metoder ej får användas, eller att lösningen ska göras enligt en särskild modell.
- Lösningen ska fungera exakt som i körexemplen i uppgiften, om inte texten indikerar något annat.
- Lösningen ska vara generell, d.v.s. den ska inte enbart fungera för de körexempel som finns i uppgiften, utan även för alla andra möjliga indata på samma form. Felkontroller av indata behöver dock normalt inte göras, om inte uppgiften särskilt uttrycker det.

Uppgifter - Datortentamen TDDD73 Funktionell och imperativ programmering i Python

Tisdag 10 januari 2017 kl 14-19

Uppgift 1

Skriv en funktion `find_closest(seq1, seq2)` som tar två listor och returnerar en ny lista som för varje tal i `seq1` innehåller det tal i `seq2` som har närmsta värdet. Du kan anta att `seq2` innehåller minst ett tal. Om det finns både ett större och ett mindre tal som är närmast så ta det största.

```
>>> find_closest([1, 2, 4], [1, 3])
[1, 3, 3]
>>> find_closest([1, 2, 4], [-5, 15, -2])
[-2, -2, -2]
```

Uppgift 2

Skriv en funktion som tar en lista och ett tal `threshold` och delar upp talen i tre listor, en med alla tal mindre än `threshold`, en med alla tal lika med `threshold` och en med alla tal större än `threshold`. Funktionen ska finnas i två varianter: en som arbetar enligt en rekursiv modell (kallad `split_r`) och en som arbetar enligt en iterativ modell (kallad `split_i`). Du får inte använda inbyggda funktioner eller metoder som behandlar hela listor. Du får inte heller använda listbyggare (eng. *list comprehensions*). Lösningarna ska vara funktionella i bemärkelsen att de inte får modifiera indata. Funktionerna ska fungera exakt likadant, bortsett från att de ska ha olika lösningsmodeller. Här är några exempel:

```
>>> split_r([5, 2, 1, 7, 2, 5], 5)
([2, 1, 2], [5, 5], [7])
>>> split_i([5, 2, 1, 7, 2, 5], 7)
([5, 2, 1, 2, 5], [7], [])
```

De båda funktionerna `split_r` och `split_i` ger vardera 2.5 poäng, och det är möjligt att få poäng på dem oberoende av varandra.

Uppgift 3

Skriv en funktion `sum_nth` som tar en lista som kan innehålla godtyckligt nästlade listor och ett tal n och summerar vart n:te tal i listan. Funktionen ska vara icke-destruktiv. Det går bra att lösa uppgiften rekursivt eller iterativt. Tänk som vanligt på att dokumentera funktionen på lämpligt sätt.

```
>>> sum_nth([1, 2, 3, 4, 5], 1)
15
>>> sum_nth([1, [2, 3], 4, [5]], 2)
6
>>> sum_nth([1, [[2, [3]], 4], [5]], 3)
3
```

Uppgift 4

Deluppgift 4a (3p)

Att derivera funktioner är en viktig del av den matematiska analysen. En funktion kan antingen deriveras symboliskt eller numeriskt. Du har huvudsakligen lärt dig att derivera funktioner symboliskt. Ett sätt att numeriskt uppskatta derivatan f'(x) av en funktion f(x) är att räkna ut lutningen på en linje mellan två punkter kring x enligt (f(x+h) - f(x-h)) / 2h. Skriv en högre ordningens funktion `derivate(f, h)` som tar en funktion `f(x)` och ett tal `h` och returnerar en funktion som tar ett argument och beräknar derivatan `f'(x)` numeriskt enligt ovan.

Deluppgift 4b (2p)

Skriv ett uttryck som använder `derivate` för att räkna ut derivatan av $2x^3$ vid för x=2.

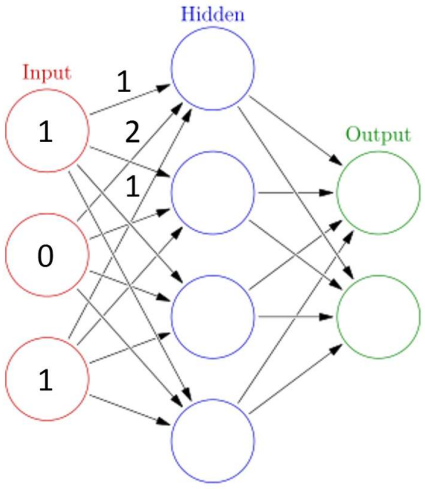
Uppgift 5

Ett vanligt förekommande problem in datavetenskapen är att hitta den längsta växande delsekvensen (longest increasing subsequence) utifrån en sekvens med tal. En växande delsekvens är en sekvens där varje tal är minst lika stort som föregående. Givet sekvensen 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15 är tre exempel på växande delsekvenser 1, 3,7; 0, 8, 14, 15 och 0, 2, 6, 9, 11, 15. Den sista är ett exempel på en längsta växande delsekvens. Ett sätt att hitta dessa är att givet en växande delsekvens och nästa tal i indata som är minst lika stort som det sista i delsekvensen jämföra om delsekvensen över hela indata blir längre med och utan detta tal. Din uppgift är att skriva en funktion `lis(seq)` som returnerar längden på den längsta växande sekvensen. Funktionen behöver inte vara minnes- eller tidseffektiv.

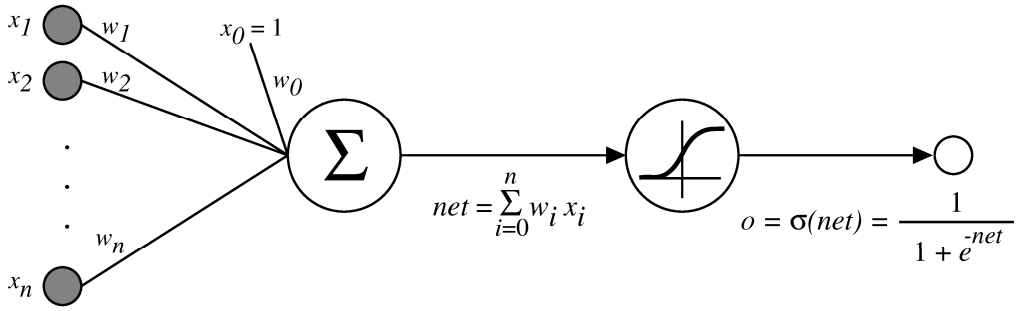
```
>>> lis([0, 2, 3, 4])
4
>>> lis([0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15])
6
```

Uppgift 6

Neurala nätverk är väldigt populära just nu. Trots sin enkelhet har de rönt väldigt stora framgångar inom maskininlärning. Ett neuralt nätverk uppskattar en funktion genom ett antal sammankopplade lager av neuroner. Det första lagret kallas för input och består av ett neuron för varje indata. Det sista lagret kallas för output och består av ett neuron för varje utdata. Mellan dessa två lager finns vanligtvis minst ett ”gömt” lager. I ett fullständigt sammankopplat nätverk så är varje nod i ett lager kopplat med varje nod i föregående lager. Varje koppling har en vikt. För att räkna ut en nods värde så summeras de viktade input-värdena. I exemplet bredvid så är värdet för den översta gömda noden 1*1+2*0+1*1=2.



Vanligtvis så appliceras en icke-linjär överföringsfunktion på värdet för att få ett värde mellan 0 och 1, t.ex. den så kallade Sigmoid-funktionen $1/(1+e^{-x})$. Se bilden nedan.



Din uppgift är att implementera ett enkelt framåtmatande nätverk enligt följande beskrivning. Lägg gärna till ytterligare relevanta primitiver. Alla funktioner ska vara väl dokumenterade.

- `create_network(nodes, transfer_func)` – Givet en lista med antalet noder i respektive lager, från input till output, samt en överföringsfunktion skapa ett nytt nätverk och returnera det. Alla vikter ska ha vikten 0. (1p)
- `init_weights(nn)` – Initialisera vikterna till godtyckliga värden mellan -1 och 1 med medelvärde 0, t.ex. genom att växla mellan 0.1 och -0.1. (1p)
- `feed_forward(nn, input)` – Returnera en lista med output för varje lager i `nn` förutom input-lagret givet `input`. Varje output blir en lista med ett värde för varje nod i lagret. (3p)

```
>>> # Create the network in the picture above
>>> nn = create_network([3, 4, 2], lambda x: 1/(1+math.exp(-x)))
>>> feed_forward(nn, [1, 0, 1])
[[0.5, 0.5, 0.5, 0.5], [0.5, 0.5]]
>>> init_weights(nn)
>>> feed_forward(nn, [1, 0, 1])
[[0.549833997312478, 0.45016600268752216, 0.549833997312478,
0.45016600268752216], [0.5049832347256592, 0. 5049832347256592]]
```