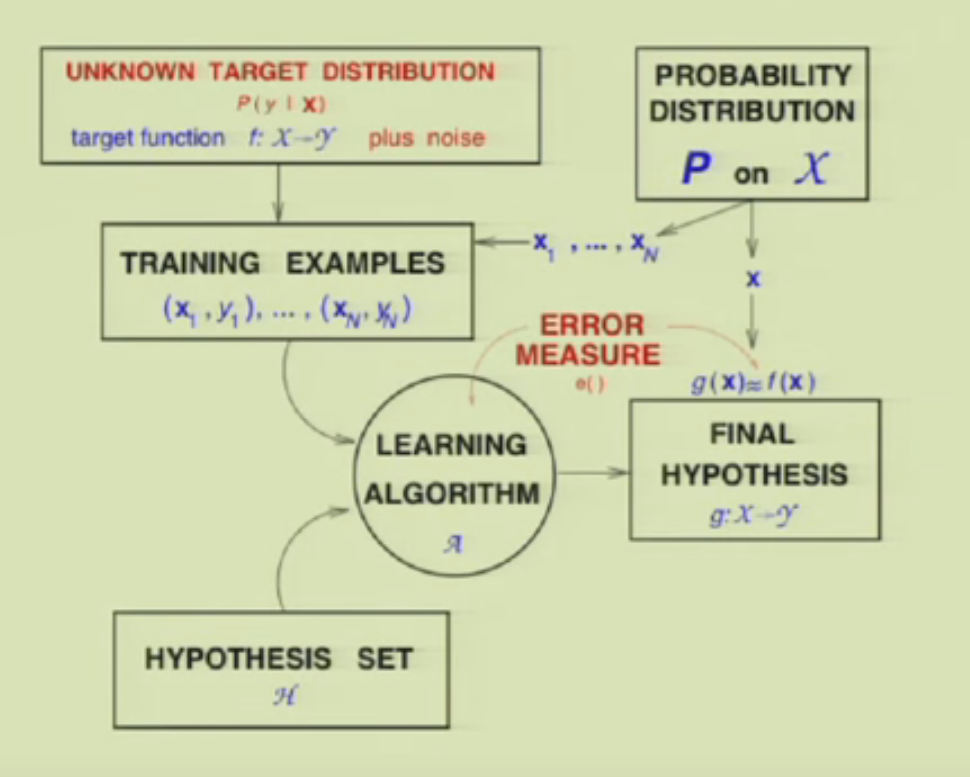
Machine Learning AU 2016

Handin 1

# Introduktion

Denne rapport er udarbejdet den første opgave i kurset machine learning ved Århus universitet 2016. Formålet med opgaven er at få en forståelse for grundlæggende principper i machine learning og derfor implementeres læring af to simple modeller.

I bogen brugt i kurset, Learning From Data[[1]](#footnote-1) bruges følgende visualisering til at anskueliggøre elementerne i machine learning:



I denne opgave beskrives implementationen af læring af to forskellige hypothesis sets:

* **logistic regression**, til binary classification
* **softmax regression**, til multinary classification

For de to modeller defineres error measure ud fra Negativ Log Likelihood (negativ logoritmen af sandsynligheden for data under forudsætning for modellen, NLL), til følgende:

På baggrund af et hypothesis set og et error measure implementeres for både logistic regression og softmax regression, to algoritmer til at finde hypotesen med det laveste error measure, batch gradien descent og mini batch gradient descent.

For at kunne bruge logistic regression til multinary classification implementeres endvidere on-versus-all.

# Logistic Regression

Logistic regression med begge lærings algoritmer er implementeret i vedlagt fil logisticregression.py. Softmax kan findes i softmaxregression.py.

Alle algiritmer er implementeret i python med brug at numpy og jf. opgave beskrivelsen er det vigtigt at implementere algoritmerne med brug at matrix operationer frem for loops.

Derfor er det også kun til iterationerne i gradient descent loops er brugt og til alle matrixs operationer er brugt numpy funktioner.

For at coden skal ligne de matematiske beregninger, så meget som muligt er arrays konverteret til 2d arrays i starten af de fleste funktioner.

På den måde bliver beregningerne i koden de præcist samme, som de matematiske formler, da kolonner og rækker passer. Med brug af broadcasting bliver koden sværere at overskue og sværere at finde fejl i, hvilket derfor er forsøgt undgået. For ikke at numpy funktioner reducerer arrays (som fx np.sum(X, axis=1)) er parametren keepdims (=True) brugt.

## Mulig optimering

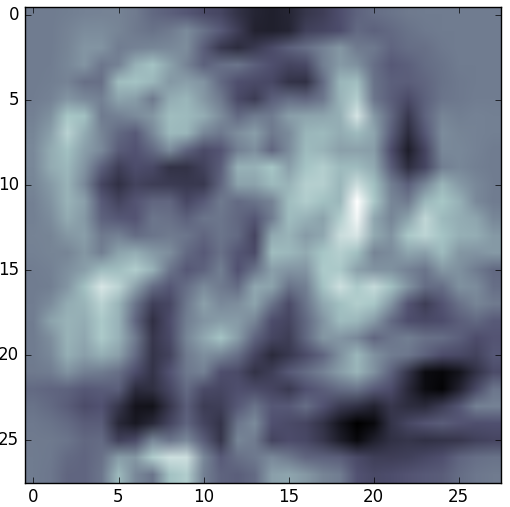
## Resultater

### Modellen

Logistik gradient descent er kørt på tallene 2 og 7 (i AU datasættet) og følgende parametre:

* Gradient descent learning rate: 0.05
* Antal iterationer: 1000

Da logistic regression er en lineær model i tonen på hver pixel, kan vi afbilde vægtene for hver pixel som et billede.



Da 2 kodes som negative kan vi se antydningen af et hvidt total og vi burde så også kunne se antydningen af et mørkt 7-tal, hvilket måske ikke er så tydeligt.

Den samlede tid for alle iterationer er 2.34 sokunder.

Out of samle (på test sættet) misklassificerer algoritmen 9 ud af 516 hvilket svarer til en out of sample error på 1,74%.

De misklassificerer billeder ses herunder:



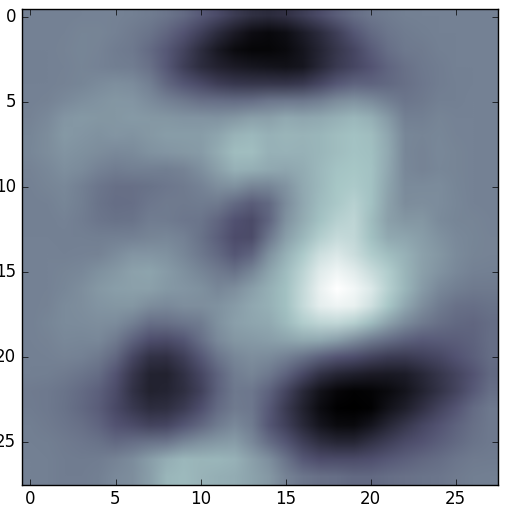
???????????????

#### Regularisering

Ved at tilføje regularisering, straffer vi høje vægte i parameter vektoren.

Jeg har lavet forsøg med regulariserings værdier fra 0.1 til 1. Det har ikke lykkedes mig at få en bedre klassificering. Med 0.1 performer algoritmen ca. lige så godt som uden regularisering.

Ved et forsøg med en regulariserings parameter på 0.3 fås en fejl klassifikation på 23,3% og parameter vektoren visualiseres herunder



Sammenlignes denne med visualiseringen af parameter vektoren for den ikke regulariserede udgave er kontrasten større. Vi kan se et lyst syvtal og de områder hvor totallet vil overlappe og hvor syvtaller ikke umiddelbart skulle overlappe er meget mørke.

Dette skyldes at regularisering presser specielt små værdier ned så klasifikationen kan ske med færre store værdier i vektoren.

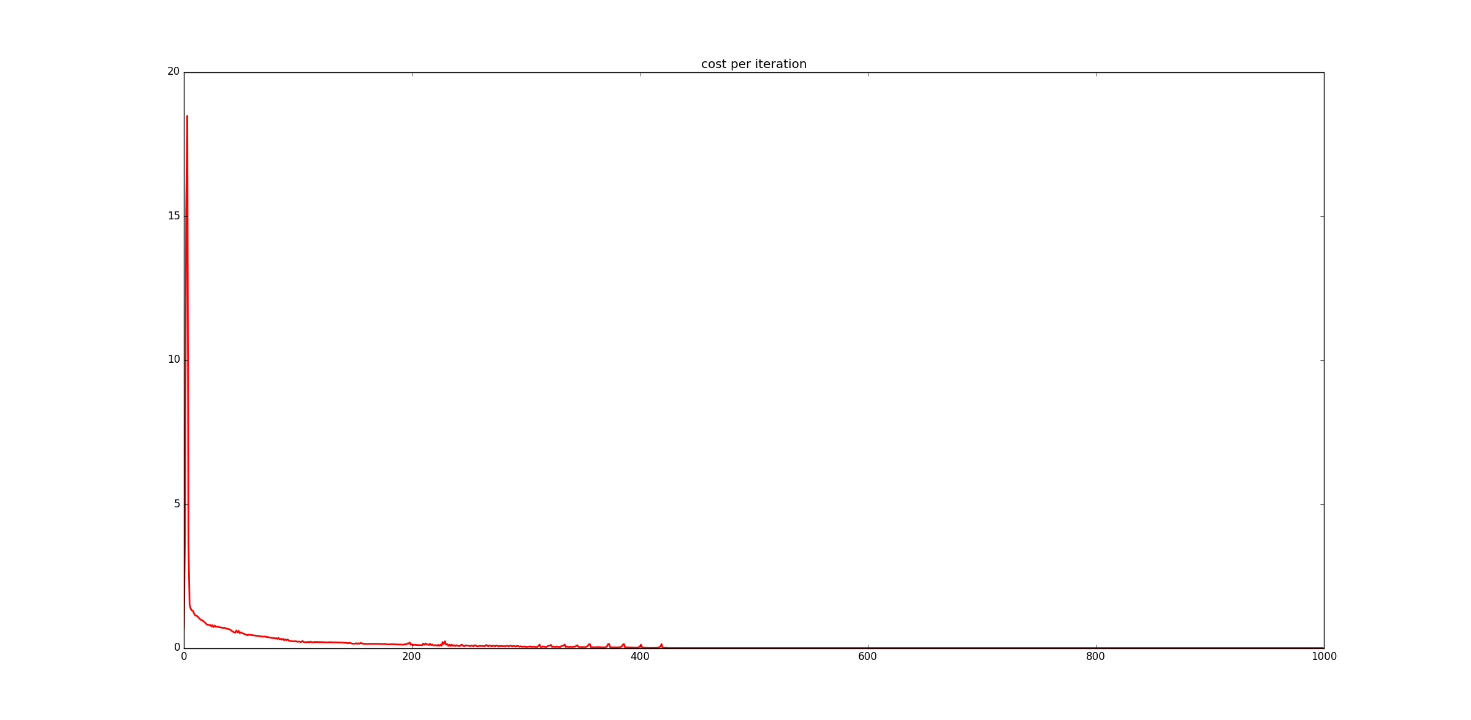
#### MNIST

Til sammenligning er algoritmen med samme parametre kørt på MNIST datasættet.

Fejlen på dette datasæt var 2.08, altså dårligere end for AU datasættet. Dette kan skyldes mindre ensartede tal, men kunne også bare være naturlig variation.

### Lærings algoritmen

I nedenstående graf ses hvordan gradient descent minimerer error measure iteration for iteration. I starten svinger den lidt, hvoretf er der er et hårdt fald, hvorefter den langsomt stabiliserer.

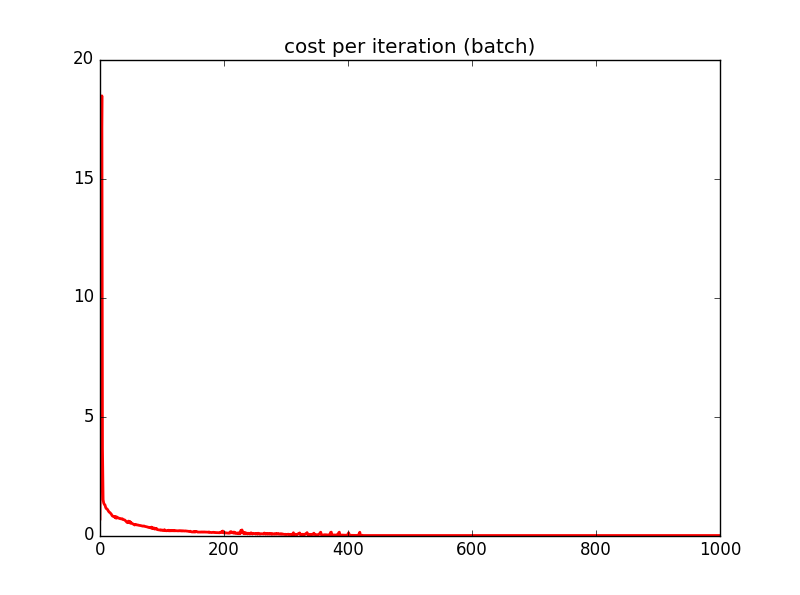
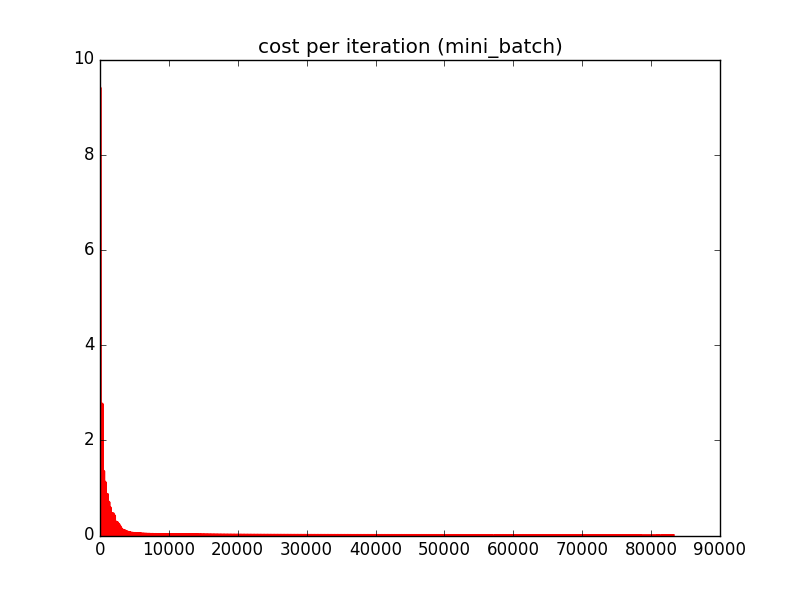


## Sammenligning af batch og mini batch gradien descent

Som beskrevet ovenfor er både gradient descent implementeret både som batch og mini batch hvor hver iteration kun tager en delmængde af data i betragtning. For at kunne sammenligne de to algoritmer blev de to eksekveret på samme data, men med en tidsbegrænsning på hvor længe de måtte køre. Følgende parametre blev brugt:

* Gradient descent learning rate: 0.05
* Batch størrelse (kun for mini batch): 10
* Tidbegrænsning: 10 sekunder

På de ti sekunder batch gradient descent med 1.74% fejl klassifikation hvorimod mini batch kom ned på 1.16%. Udviklingen i cost gennem iterationerne ses nedenfor.

På x-aksen er afbildet iterationer for batch gradient descent og en iteration per batch for mini gradient descent.

Da algoritmerne kører lige lang tid kan vi betragte iterationerne på x aksen relativt som tiden algoritmerne har kørt (også selvom mini batch grafen ikke går helt ud til kanten).

Det kan ses at mini batch gradient descent fluktuerer meget mere selvom den ender med et bedre resultat.

## Fuld OCR med All Versus One

Til slut er en fuld classifier implementeret med All Versus One, hvilket vil sige at de enkelte klasser (tal) trænes mod alle andre tal. Dette giver giver 10 modeller og når et billede skal klassifiseres prøves alle modellerne og den klassifier der giver størst sikkerhed (sandsynlighed for det pågældende tal), afgør udfaldet.

I et eksperiement er algoritmen kørt på AU datasættet med følgende parametre:

* Gradient descent learning rate: 0.05
* Antal iterationer: 100
* Batch størrelse: 10

Udfaldet var at næsten 20 % (19.9%) fejlklassificeres. 10 fejlklassifiserede billeder ses herunder:



Som 2-7 logistic regresion kan parametrene for de 10 modeller i all versus one visualiseres:



Tallene ses tydeligt i parametrene, da modellen belønner områder hvor det pågældende tal differentierer sig andre tal.

## Teoretiske spørgsmål

I opgaven skal følgende teoretiske spørgsmål besvares.

**Sanity Check: What happens if we randomly permute the pixels in each image (with the same permutation) before we train the classifier? Will we get a classifier that is better, worse, or the same? Give a short explanation.**

Permutation af pixels har ingen betydning for vores classifier. Grunden til dette er at modellen ikke antager eller bruger afhængigheder mellem features (her pixels)

**Linear Separable: If the data is linearly separable, what happens to weights when we implement logistic regression with gradient descent? That is, how do the weights that minimize the negative log likelihood look like? Assume that we have full precision (that is, ignore floating point errors). We can run gradient descent on the data set for as long as we want (suppose God helps you). Now what will happen with the weights in the limit? Do they converge to some fixed number (fluctuate around it) or do they keep increasing in magnitude (absolute value)? Give a short explanation for your answer. What happens if we add regularization?**

Når vi minimerer negative log likelihood er det det same som at maksimere produktet at sandsynligheden for de individuelle data punkter. Dette er igen for logistic regression udtrykt ved logistic funktionen. Da denne er voksende vil resultatet af en skalar multiplikation med parameter vektoren være en øget værdi af sandsynligheden for det pågældende data punkt. Dette betyder at der ikke findes noget maksimum da vi ser bort fra numeriske udfordringer og vægtene vil med gradient descent vokse uendeligt.

**Bonus Question: Convexity of negative log likelihood Show that the negative log likelihood function for logistic regression is convex. Is it still convex if we add regularization?**

???????????????????????????

# Softmax Regression

## Results

Det viser sig at for softmax skal lærings hastigheden på gradient descent være noget lavere end for logistic regression. Jeg havde bedst success med en rate på 0.00002. Jo højere værdien var herfra jo mere fluktuerede cost gennem iterationerne ohvilket resulterede i et dårligere resultat. Med følgende parametre:

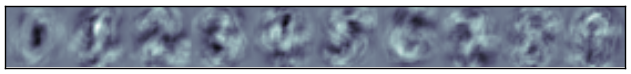
* Gradient descent learning rate: 0.00002
* Antal iterationer: 1000

Fås en out of sample fejl rate på 7.84%. Parametrene for modellen er gemt I filen paramsSoftmax.npz.

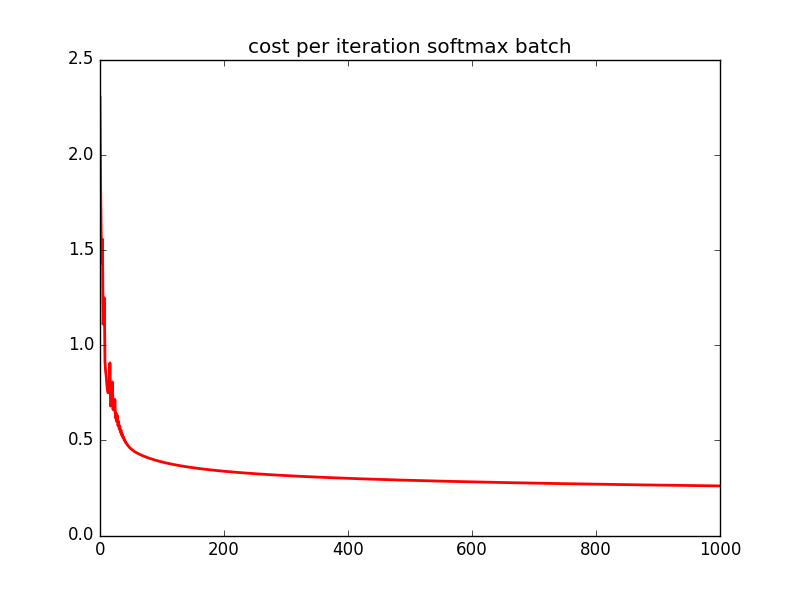
Følgende er eksempler på billeder der ikke klassificeres korrekt:



En visualisering af modellen parametre ses herunder:



Og udviklingen af cost gennem iterationerne ses herunder:



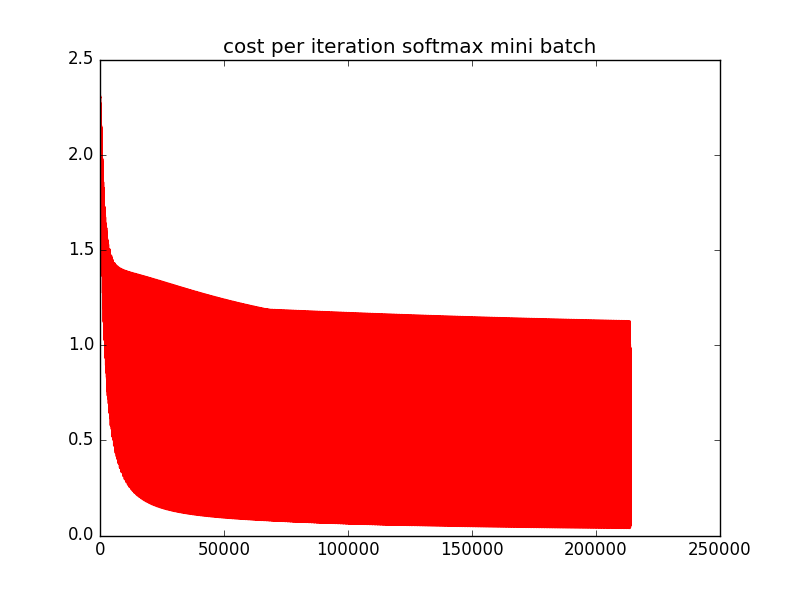
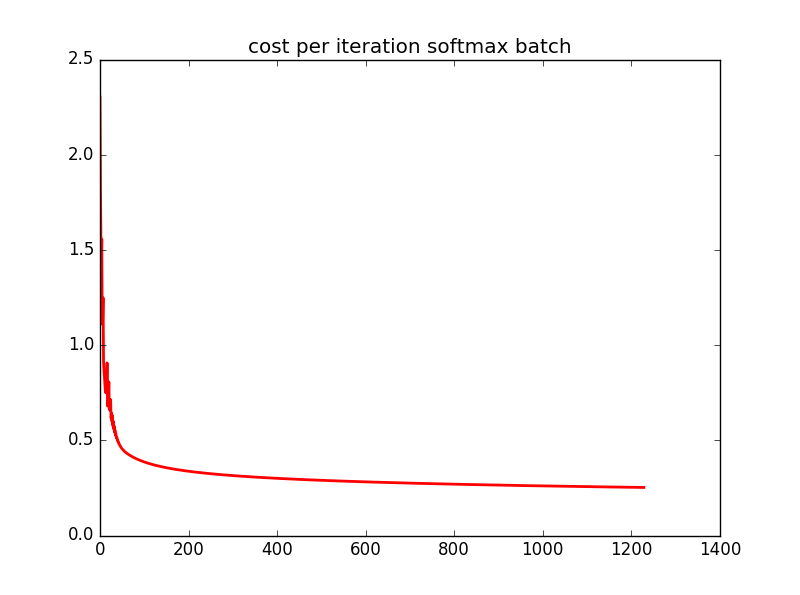
### MNIST

Ovenstående algoritme er kørt på MNIST datasættet hvilket giver en fejl rate på 7.46% ca. det samme som på AU datasættet.

### Comparing Batch and Mini Batch

For at kunne sammenligne batch og mini batch køres begge algoritmer på AU data sættet med samme parameter (læringshastighed 0.00002) og med en tidbegrænsning på 1 minut.

Out of sample fejlen for batch var 7.56% og for mini batch 9.37%. Man kunne forestille sig at dette skyldes at mini batch bevæger sig langsommere mod minimum. Det er derfor interessant at se på costs udvikling gennem iterationerne:

Af ovenstående kan vi ses at mini batch fluktuerer voldsomt helt hen til den sidste iteration. Der er derfor en vis portion tilfældighed (varians) i hvor den ender.

1. Abu-Mostafa, Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*. Vol. 4. Singapore: AMLBook, 2012. [↑](#footnote-ref-1)