DD1418:
8a: More on classification

Johan Boye, KTH

(Text) classification

Given a datapoint x we want to find the class y that maximizes $P(y|x)$.

$$\arg\max_y P(y|x)$$

If x is represented by the features $w_1, \ldots, w_n$, then we want to find

$$\arg\max_y P(y|w_1, \ldots, w_n)$$

Generative classifiers

One of the classification methods we've seen is *Naive Bayes*.

Naive Bayes is modeling $P(y|w_1 \dots w_n)$ in terms of

$$\arg\max_y P(y|w_1 \dots w_n) = \arg\max_y P(w_1 \dots w_n|y)P(y)$$

$P(w_1 \dots w_n|y)P(y) = P(w_1 \dots w_n, y)$ is the *joint* probability of $w_1 \dots w_n$ and y.

Classifiers that model the joint probability of the input and the class label are called *generative classifiers*.

Generative classifiers

Hidden Markov models (HMMs) are also generative:

$$\arg\max_{t_1 \ldots t_n} P(t_1 \ldots t_n | w_1 \ldots w_n) =$$
$$\arg\max_{t_1 \ldots t_n} P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n)$$

$P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n) = P(w_1 \ldots w_n, t_1 \ldots t_n)$, the joint probability of $w_1 \ldots w_n, t_1 \ldots t_n$.

Discriminative classifiers

By constrast, a *discriminative classifier* estimates $P(y|x)$ without estimating $P(x, y)$.

(where x is the data point, and y is the class label.)

The predicted class is $\arg\max_y P(y|x)$.

Spam detection example again

Consider the spam detection problem again. Some possible features:

- f_1 The mail contains a clickable button (Yes=1, No=0)
- f_2 The mail mentions sums of money (like \$1,000).
- f_3 The sender has domain name `kth.se`
- f_4 Proportion of spelling errors

A datapoint (a mail) can now be represented by a vector, e.g. $x = (0, 1, 0, 0.02) = (x_1, x_2, x_3, x_4)$.

Spam detection example, cont.

$x = (0, 1, 0, 0.02)$

Now we'd like to find *weights* (real numbers) $\theta_0, \theta_1, \theta_2, \theta_3, \theta_4$, such that when the weighted sum

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

results in a high number, it is likely that x is spam.
Low number $\rightarrow$ unlikely that x is a spam.

But what is a high number? Or a low number?

Vector notation

More convenient vector notation: Add an extra "dummy" feature to each data point, whose value is always 1:

$$x = (0, 1, 0, 0.02) \rightarrow x = (1, 0, 1, 0, 0.02)$$

Now

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

can be expressed simply as:

$$\theta^T x$$

DD1418:
8b: Logistic regression

Johan Boye, KTH

Logistic regression

Logistic Regression is a **discriminative binary** classification method (in spite of having "regression" in its name), returning a number between 0 and 1, which we interpret as the probability

$$P(y = 1|x)$$

i.e. the probability that x belongs to the "positive" class 1.

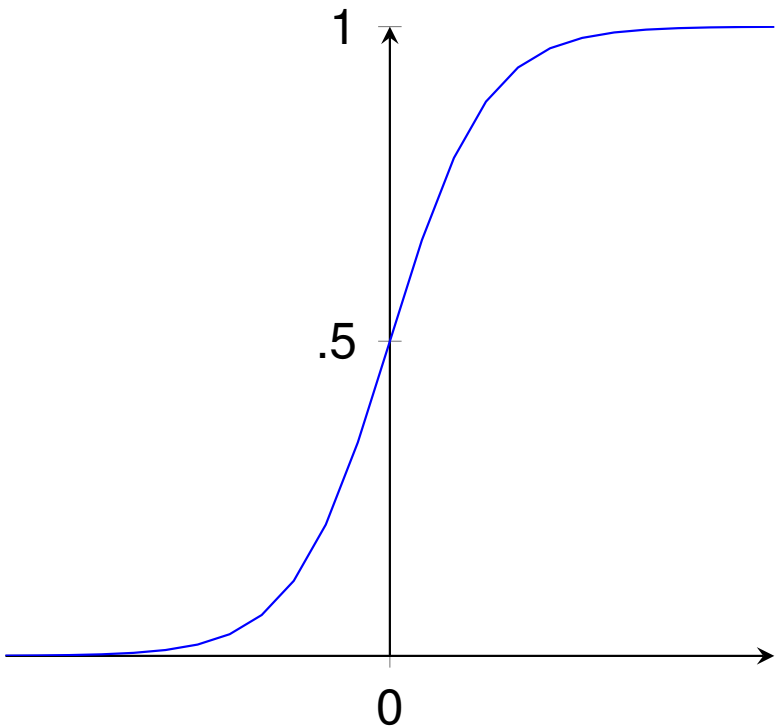
Problem with $\theta^T x$:

- It might be negative
- It might be greater than 1

Solution: Use the *logistic function* $\sigma(z) = \frac{e^z}{e^z+1} = \frac{1}{1+e^{-z}}$.

Logistic function

$\sigma(z)$ is called the *logistic function*, and is a *sigmoid* (s-shaped) function.



$$\sigma(z) = \frac{1}{1+e^{-z}}$$

It is always the case that $0 < \sigma(z) < 1$.

Logistic regression

Compute the weighted sum $\theta^T x$, and then apply the logistic function:

$$\begin{aligned} P(y=1|x) &= \sigma(\theta^T x) \\ &= \frac{1}{1+e^{-\theta^T x}} \end{aligned}$$

and

$$\begin{aligned} P(y=0|x) &= 1 - P(y=1|x) \\ &= \frac{e^{-\theta^T x}}{1+e^{-\theta^T x}} \end{aligned}$$

Often we write $h_\theta(x)$ for $P(y=1|x)$ to emphasize that θ are the parameters of the model.

Quiz: Features

We want to find documents that express a positive or a negative opinion (about films). We define the following features:

- f_0 "Dummy" feature, always=1
- f_1 Document contains positive words
- f_2 Document contains negative words

How would you encode the following documents as feature vectors?

"*This is a great film!*"
"*Worst crap I've ever seen.*"
"*Great plot but bad acting.*"
"*The film was made in 2008.*"

Quiz: Logistic regression

"*This is a great film!*"	$x = (1, 1, 0)$
"*Worst crap I've ever seen.*"	$x = (1, 0, 1)$
"*Great plot but bad acting.*"	$x = (1, 1, 1)$
"*The film was made in 2008.*"	$x = (1, 0, 0)$

Let $\theta = (-1, 2, -3)$. Which documents would belong to the positive class?

Logistic regression

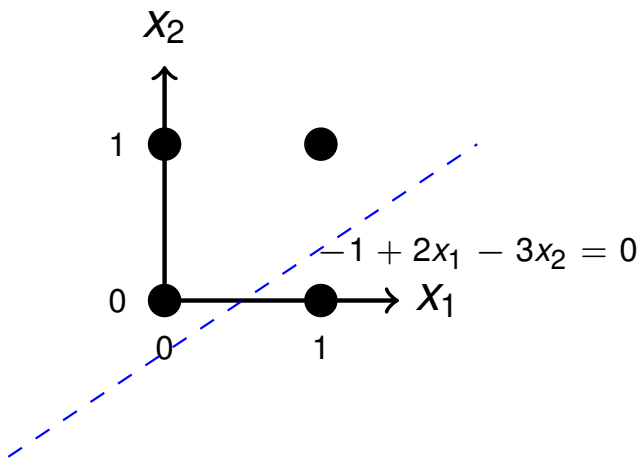
"*This is a great film!*"	$x = (1, 1, 0)$	Pos
"*Worst crap I've ever seen.*"	$x = (1, 0, 1)$	Neg
"*Great plot but bad acting.*"	$x = (1, 1, 1)$	Neg
"*The film was made in 2008.*"	$x = (1, 0, 0)$	Neg

For instance:

$$P(y = 1|(1, 1, 1)) = \sigma(-1 \cdot 1 + 2 \cdot 1 - 3 \cdot 1) = \sigma(-2) = 0.12$$

Decision boundary

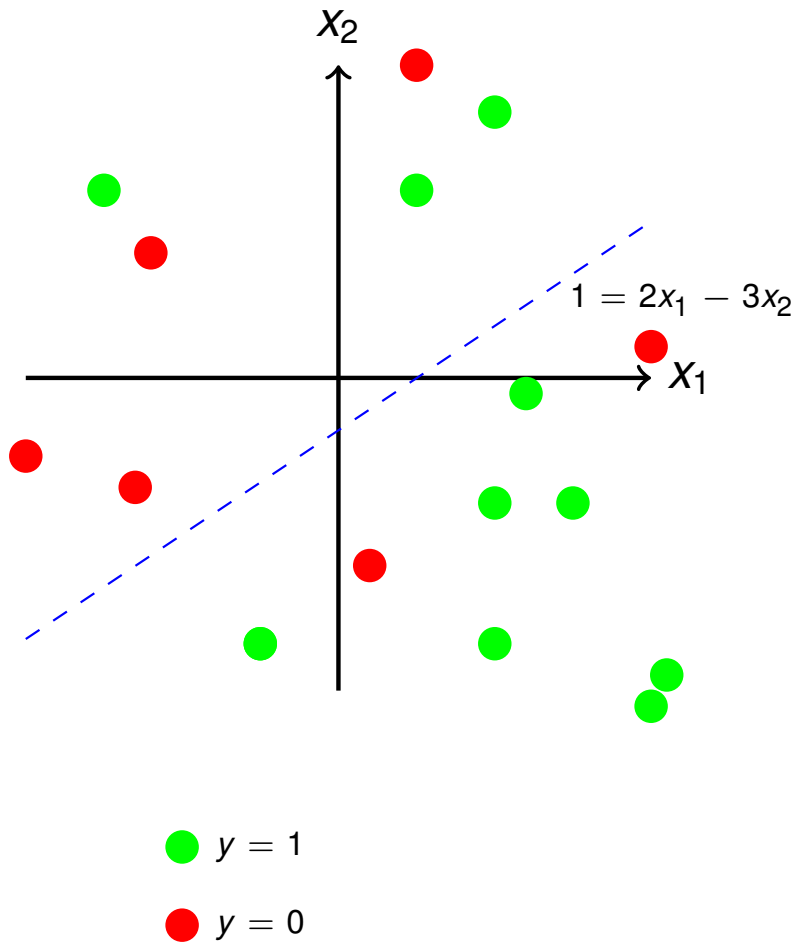
"*This is a great film!*"	$x = (1, 1, 0)$	Pos
"*Worst crap I've ever seen.*"	$x = (1, 0, 1)$	Neg
"*Great plot but bad acting.*"	$x = (1, 1, 1)$	Neg
"*The film was made in 2008.*"	$x = (1, 0, 0)$	Neg



Decision boundary

If $\theta = (-1, 2, -3)$, then the line $-1 + 2x_1 - 3x_2 = 0$ separates the positive and negative predictions.

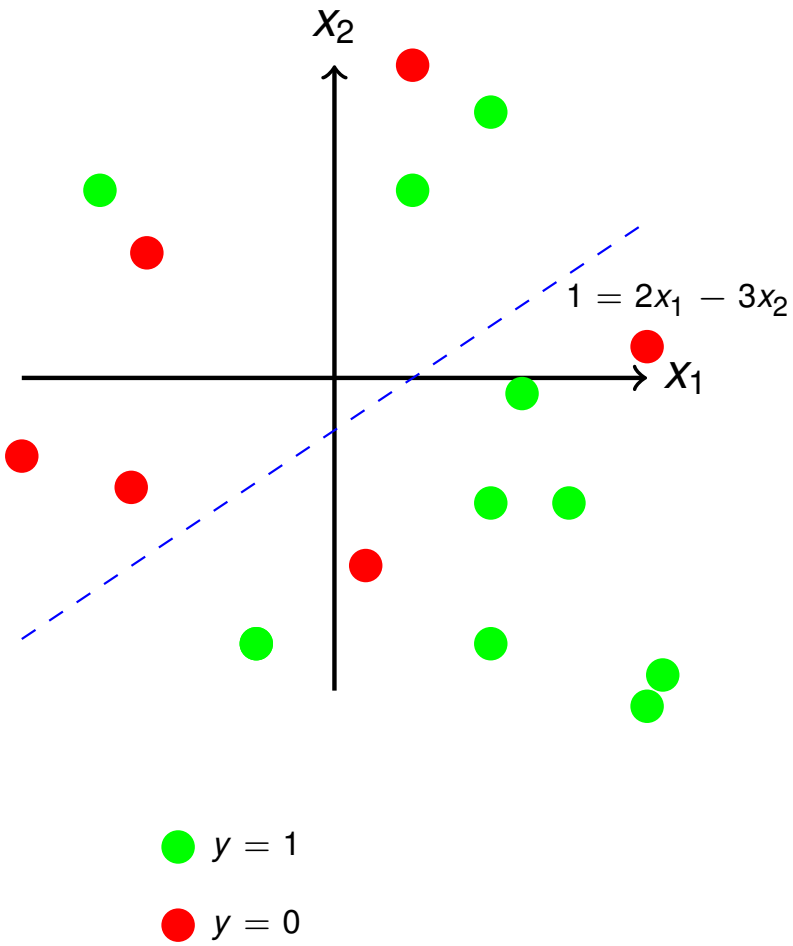
- In general:
- All data points x for which $\theta^T x > 0$ are tagged as positive ($h_\theta(x) > 0.5$)
- If $\theta^T x < 0$, the data point is tagged as negative ($h_\theta(x) < 0.5$)



Decision boundary

If $\theta = (-1, 2, -3)$, then the line $-1 + 2x_1 - 3x_2 = 0$ separates the positive and negative predictions.

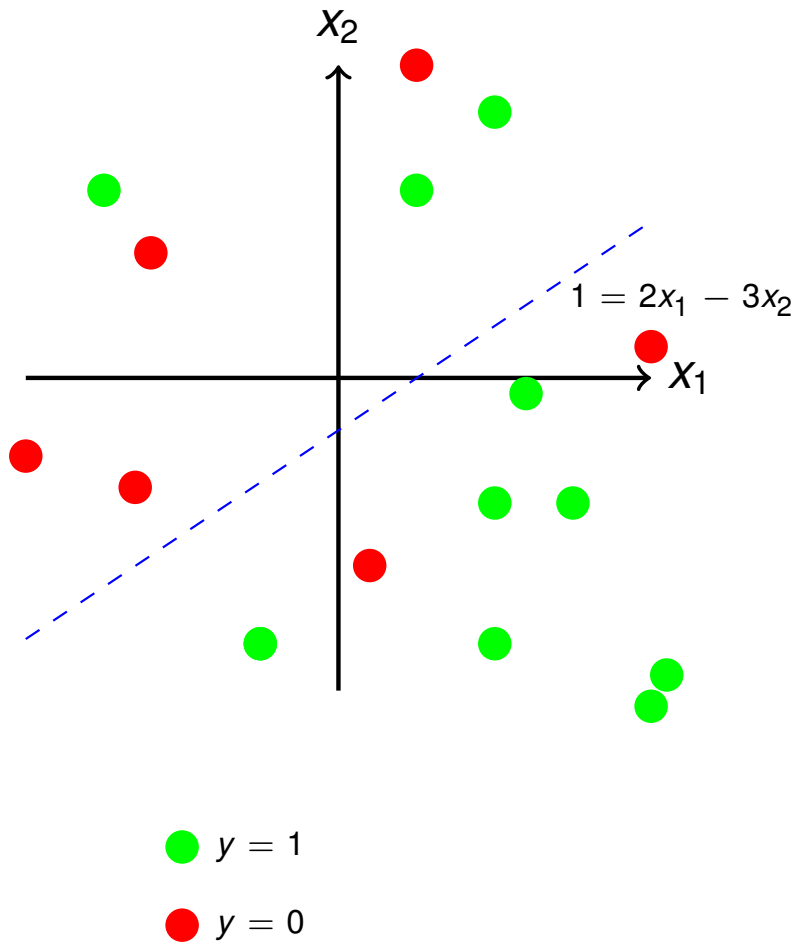
- Accuracy = ?
- Precision (pos) = ?
- Recall (pos) = ?



Decision boundary

If $\theta = (-1, 2, -3)$, then the line $-1 + 2x_1 - 3x_2 = 0$ separates the positive and negative predictions.

- Accuracy = 11/16
- Precision (pos) = 7/9
- Recall (pos) = 7/10



DD1418
8c: Learning a logistic regression model

Johan Boye, KTH

Learning a logistic regression model

Input: a matrix x of m data points, each having n features, and a vector y of m class labels (either 1 or 0).

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ & \ddots & & \\ & & \ddots & \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{pmatrix}$$

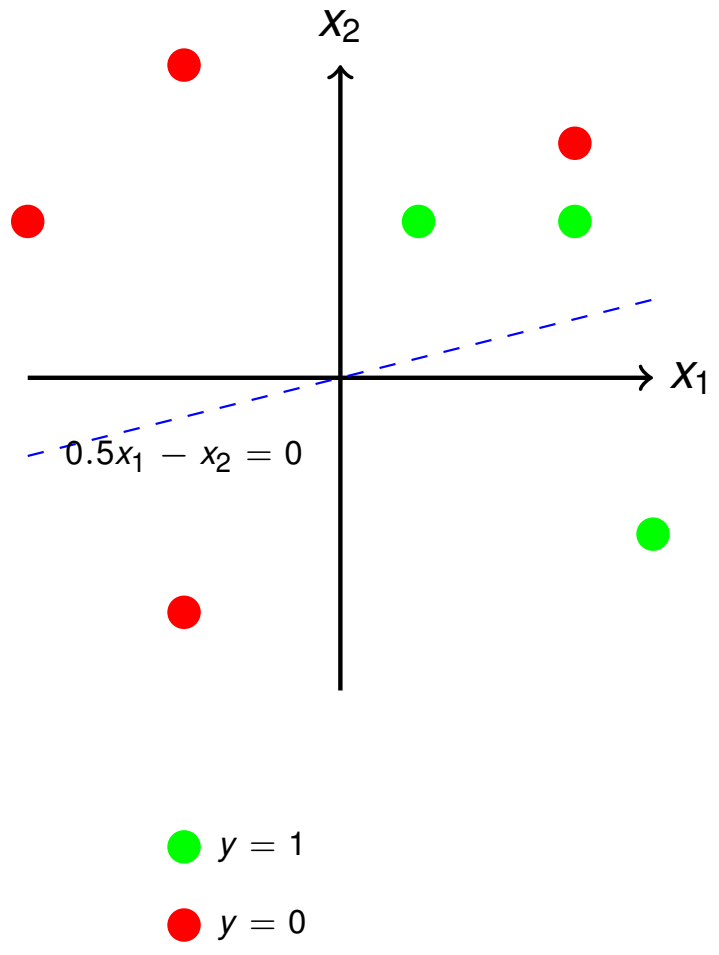
$$y = (y_1, \dots y_m)$$

The objective is to learn a vector of parameters

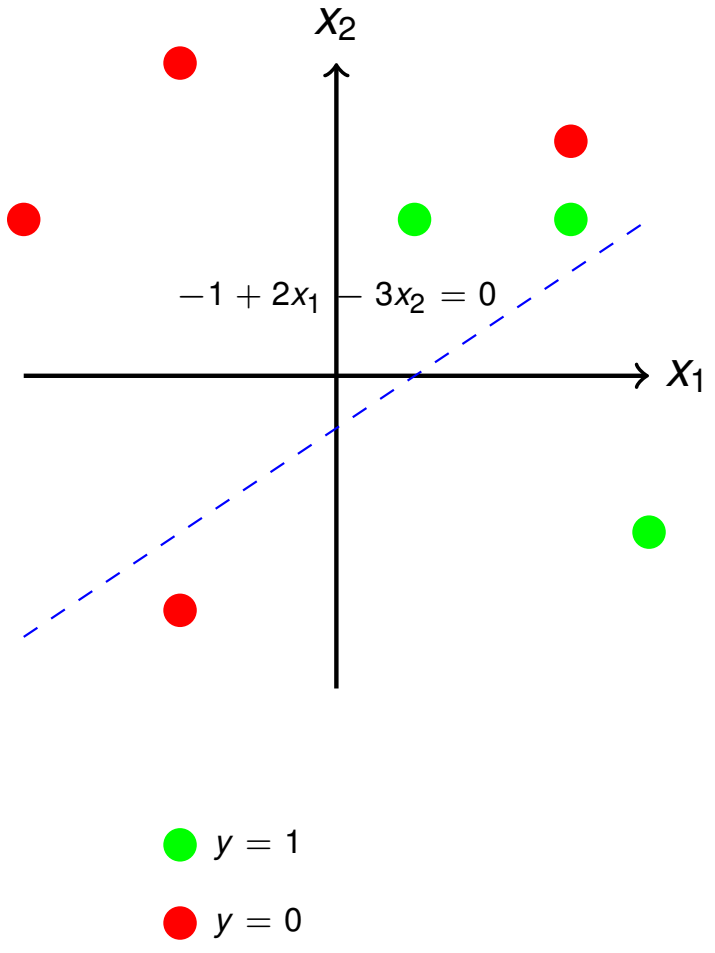
$$\theta = (\theta_0, \theta_1, \dots, \theta_n)$$

which predict the correct class as often as possible.

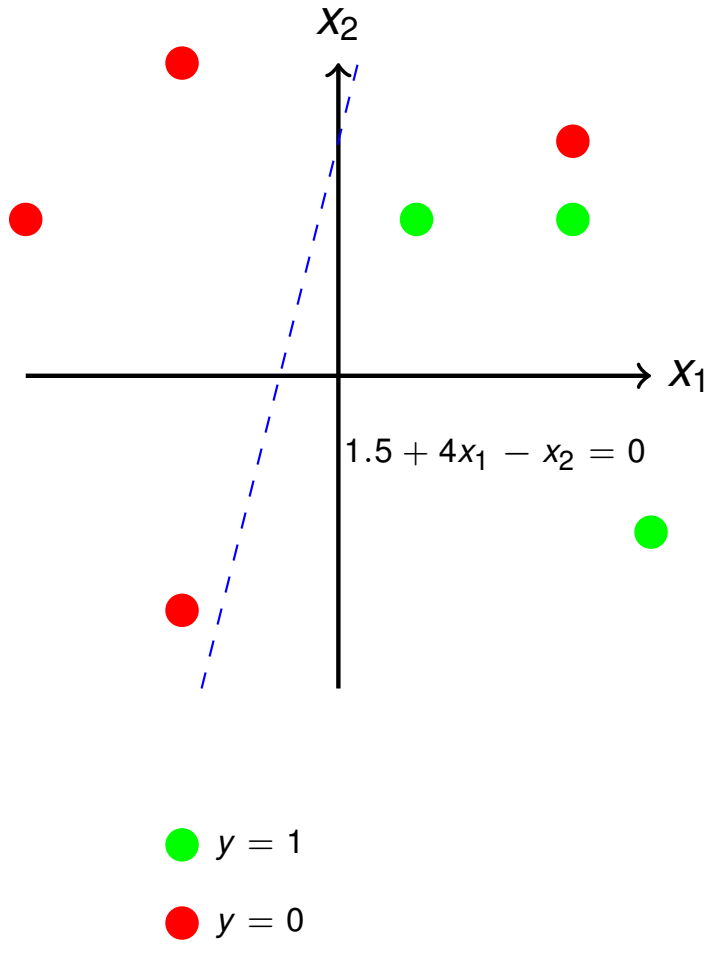
A decision boundary



A better decision boundary?



A better decision boundary



Maximum likelihood estimation of parameters

Recall that:

$$P(y = 1|x) = \sigma(\theta^T x)$$
$$P(y = 0|x) = 1 - \sigma(\theta^T x)$$

If we know that the correct class is $y = 1$, then we would like $\sigma(\theta^T x)$ to be as close to 1 as possible.

Conversely, if the correct class is $y = 0$, we would like $\sigma(\theta^T x)$ to be as close to 0 as possible.

Maximum likelihood estimation of parameters

$$P(y = 1|x) = \sigma(\theta^T x)$$
$$P(y = 0|x) = 1 - \sigma(\theta^T x)$$

This can also be written as

$$P(y|x) = \sigma(\theta^T x)^y (1 - \sigma(\theta^T x))^{1-y}$$

Assuming independence of training examples, we have:

$$\begin{aligned} L(x, y, \theta) &= \Pi_{i=1}^m P(y^{(i)}|x^{(i)}) = \\ &= \Pi_{i=1}^m \sigma(\theta^T x^{(i)})^{y^{(i)}} (1 - \sigma(\theta^T x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

We want to find θ that maximizes this function.

Cross-entropy loss function

$$\begin{aligned} L(x, y, \theta) &= \Pi_{i=1}^{m} P(y^{(i)}|x^{(i)}) = \\ &= \Pi_{i=1}^{m} \sigma(\theta^T x^{(i)})^{y^{(i)}} (1 - \sigma(\theta^T x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

Maximizing L is the same as maximizing $\log(L)$, which is the same as minimizing $-\frac{1}{m}\log(L)$:

$$\ell(x, y, \theta) = \frac{1}{m} \sum_{i=0}^{m} [-y^{(i)} \log(\sigma(\theta^T x^{(i)})) - (1 - y^{(i)}) \log(1 - (\sigma(\theta^T x^{(i)})))]$$

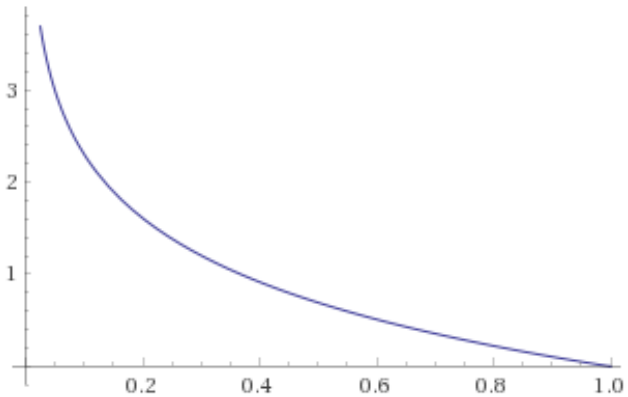
ℓ is the cross-entropy loss function.

We want to find the parameters θ that minimize the loss $\ell(x, y, \theta)$.

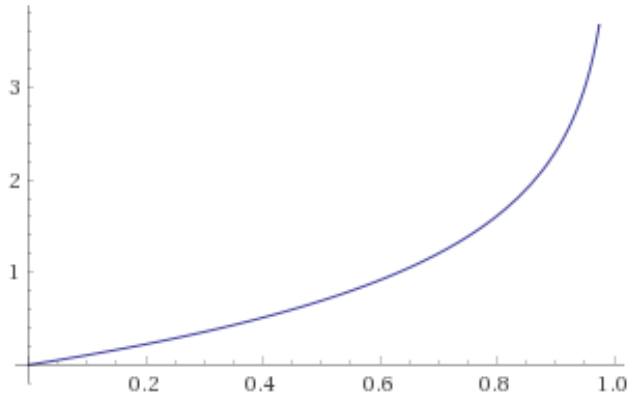
Cross-entropy loss function

$$\ell(x^{(i)}, y^{(i)}, \theta) = -y^{(i)} \log(\sigma(\theta^T x^{(i)})) - (1 - y^{(i)}) \log(1 - (\sigma(\theta^T x^{(i)})))$$

$$\ell(x^{(i)}, y^{(i)}, \theta) = \begin{cases} -\log(\sigma(\theta^T x^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - \sigma(\theta^T x^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$



$-\log(\sigma(\theta^T x^{(i)}))$



$-\log(1 - \sigma(\theta^T x^{(i)}))$

Finding minimum loss

We want to find the parameters θ that minimize the loss $\ell(x, y, \theta)$ for a given training set x, y.

This is done by computing the gradient of ℓ w.r.t. θ, and by then finding the zero of the gradient.

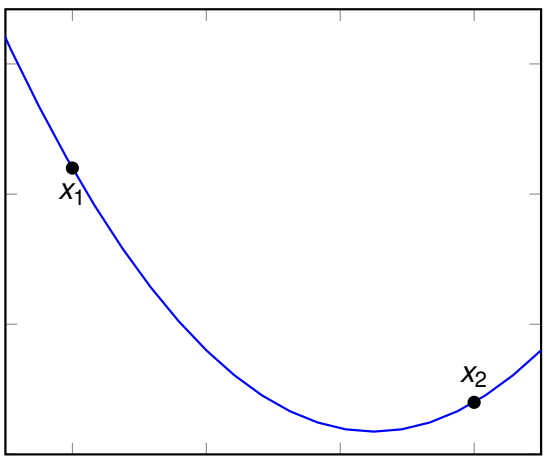
DD1418
8d: Gradient descent

Johan Boye, KTH

Gradient descent

Gradient descent is a greedy local search method.

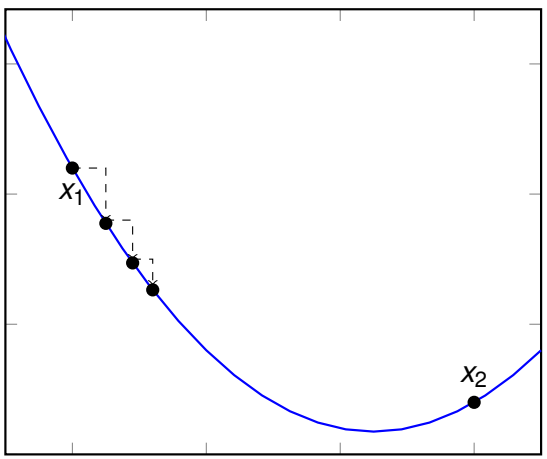
Start at some x. If the derivative of $f(x)$ is negative, take a small step to the right. If the derivative is positive, take a step to the left.



Gradient descent

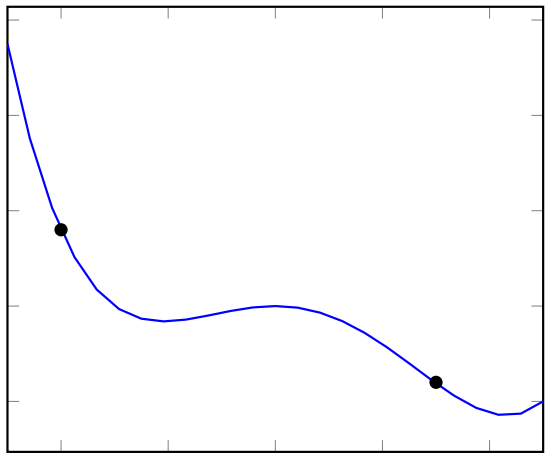
Gradient descent is a greedy local search method.

Start at some x. If the derivative of $f(x)$ is negative, take a small step to the right. If the derivative is positive, take a step to the left.



Gradient descent

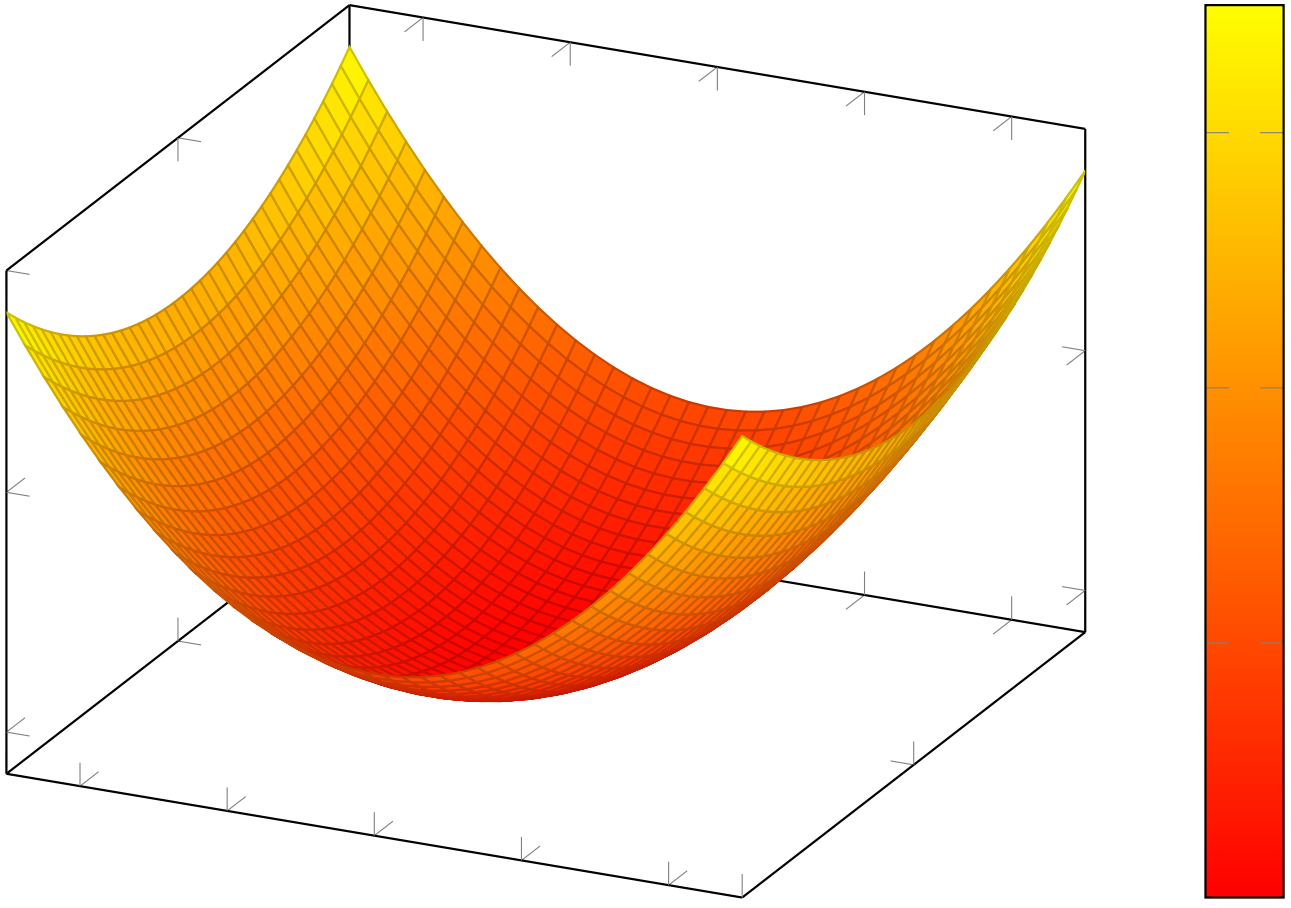
However, if the function is *not convex*, then we are not guaranteed to find a global minimum.



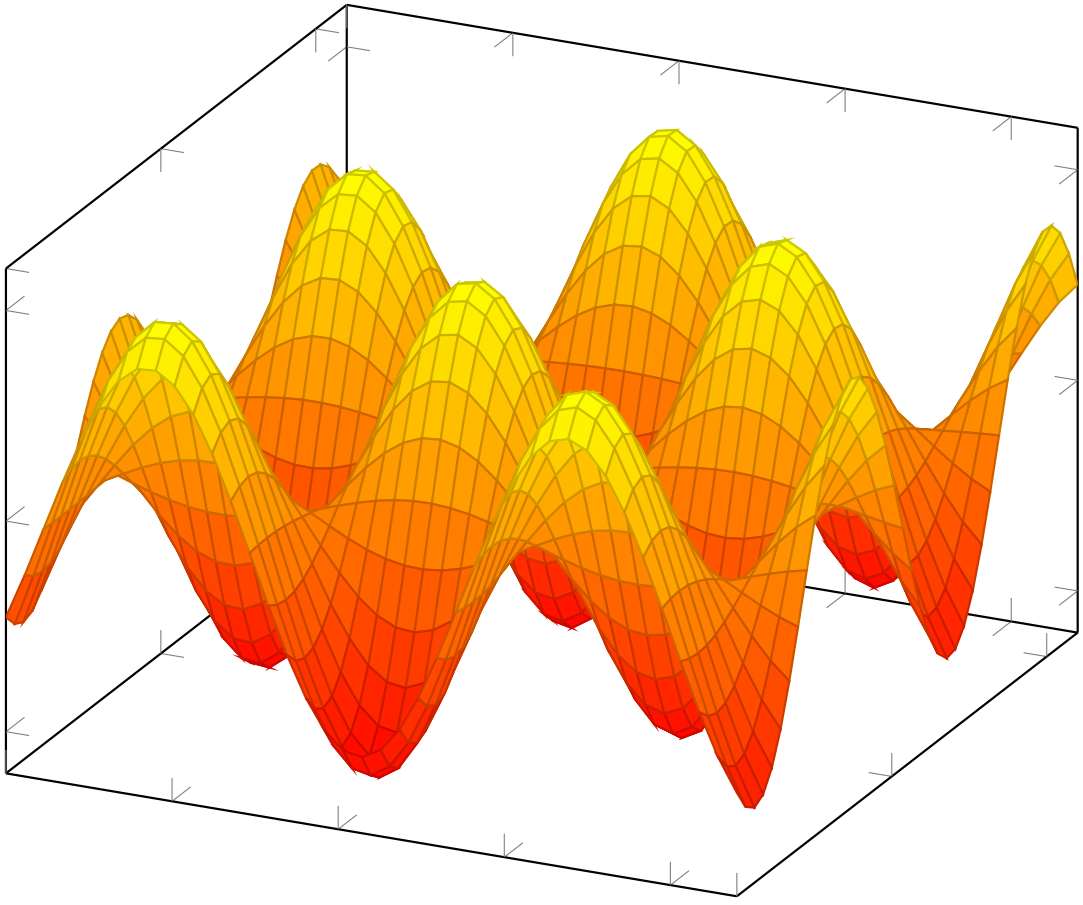
A convex function f satisfies $f(\frac{a+b}{2}) \leq \frac{f(a)+f(b)}{2}$

The cross-entropy loss function for logistic regression is convex.

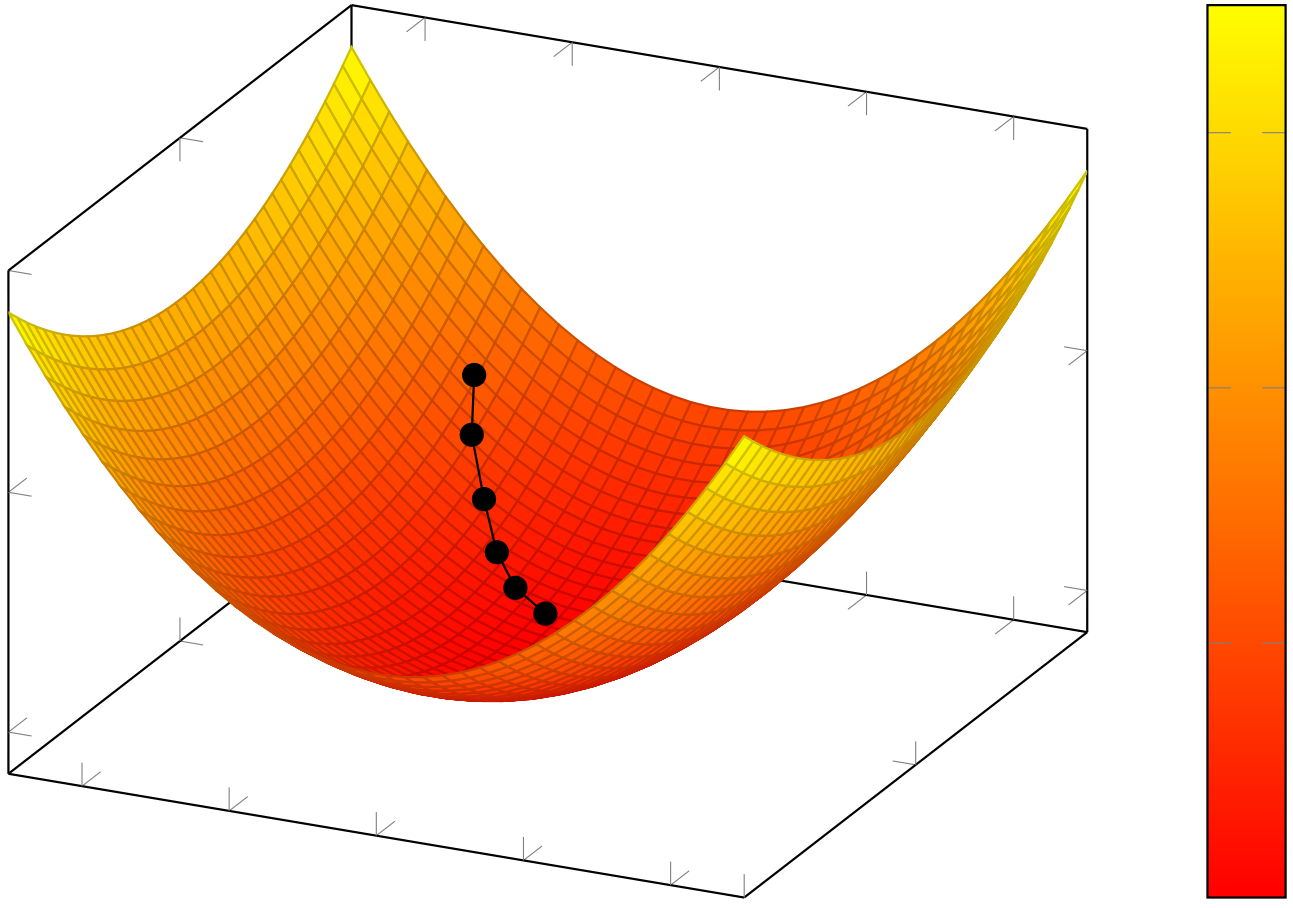
A convex function in two variables



A non-convex function in two variables



Gradient descent, two variables



Gradient

In general, the *gradient* of a multivariate function $f(z_1, \ldots z_n)$ is the vector of all its partial derivatives.

$$\nabla f = (\frac{\partial f}{\partial z_1}, \ldots, \frac{\partial f}{\partial z_n})$$

The vector ∇f points in the direction of the *steepest ascent* of f.
(Hence $-\nabla f$ points in the direction of *steepest descent* of f.
This is the direction where we want to go!)

Gradient descent

The gradient descent method: Initialize $x = x_0$ (randomly). Then update

$$x_{i+1} = x_i - \alpha \nabla(f)$$

until $|\nabla f(x)| < \epsilon$.

α is a hyperparameter called the *learning rate*, and is typically small (0.01)

Gradient descent always converges to a local minimum (in the general case), or a global minimum (if f is convex).

DD1418
8e: More on gradient descent

Johan Boye, KTH

Gradient descent

The gradient descent method: Initialize $x = x_0$ (randomly). Then update

$$x_{i+1} = x_i - \alpha \nabla(f)$$

until $|\nabla f(x)| < \epsilon$.

α is a hyperparameter called the *learning rate*, and is typically small (0.01)

Gradient descent always converges to a local minimum (in the general case), or a global minimum (if f is convex).

Gradient of the cross-entropy loss function (1)

We now need to compute the partial derivative of the loss function w.r.t. θ_k:

$$\frac{\partial}{\partial \theta_k} \frac{1}{m} \sum_{i=0}^{m} [-y^{(i)} \log(\sigma(\theta^T x^{(i)})) - (1 - y^{(i)}) \log(1 - (\sigma(\theta^T x^{(i)})))] =$$

$$\frac{1}{m} \sum_{i=0}^{m} \frac{\partial}{\partial \theta_k} [-y^{(i)} \log(\sigma(\theta^T x^{(i)})) - (1 - y^{(i)}) \log(1 - (\sigma(\theta^T x^{(i)})))]$$

(Derivative of a sum is a sum of the derivatives)

Let's compute the partial derivative for

$$-y^{(i)} \log(\sigma(\theta^T x^{(i)})) - (1 - y^{(i)}) \log(1 - (\sigma(\theta^T x^{(i)})))$$

Gradient of the cross-entropy loss function (2)

First rewrite:

$$-y^{(i)} \log(\sigma(\theta^T x^{(i)})) - (1 - y^{(i)}) \log(1 - (\sigma(\theta^T x^{(i)}))) =$$

$$-y^{(i)} \log(\frac{1}{1+e^{-\theta^T x^{(i)}}}) - (1 - y^{(i)}) \log(1 - \frac{1}{1+e^{-\theta^T x^{(i)}}}) =$$

$$y^{(i)} \log(1 + e^{-\theta^T x^{(i)}}) - (1 - y^{(i)}) \log(\frac{e^{-\theta^T x^{(i)}}}{1+e^{-\theta^T x^{(i)}}}) =$$

$$y^{(i)} \log(1 + e^{-\theta^T x^{(i)}}) - (1 - y^{(i)})(\log(e^{-\theta^T x^{(i)}}) - \log(1 + e^{-\theta^T x^{(i)}})) =$$

$$y^{(i)} \log(1 + e^{-\theta^T x^{(i)}}) - (1 - y^{(i)})(-\theta^T x^{(i)} - \log(1 + e^{-\theta^T x^{(i)}})) =$$

$$\theta^T x^{(i)} + \log(1 + e^{-\theta^T x^{(i)}}) - y^{(i)} \theta^T x^{(i)}$$

Now differentiate!

Gradient of the cross-entropy loss function (3)

Differentiate w.r.t. θ_k

$$\frac{\partial}{\partial \theta_k}(\theta^T x^{(i)} + \log(1 + e^{-\theta^T x^{(i)}}) - y^{(i)}\theta^T x^{(i)}) =$$

$$x_k^{(i)} + \frac{1}{1+e^{-\theta^T x^{(i)}}} \cdot e^{-\theta^T x^{(i)}} \cdot (-x_k) - y^{(i)} x_k =$$

$$x_k^{(i)}(1 - \frac{e^{-\theta^T x^{(i)}}}{1+e^{-\theta^T x^{(i)}}} - y^{(i)}) =$$

$$x_k^{(i)}(1 - (1 - \frac{1}{1+e^{-\theta^T x^{(i)}}}) - y^{(i)}) =$$

$$x_k^{(i)}(\frac{1}{1+e^{-\theta^T x^{(i)}}}) - y^{(i)}) =$$

$$x_k^{(i)}(\sigma(\theta^T x^{(i)}) - y^{(i)})$$

Gradient of the cross-entropy loss function (4)

$$\frac{\partial}{\partial \theta_k} \frac{1}{m} \sum_{i=0}^{m} [-y^{(i)} \log(\sigma(\theta^T x^{(i)})) - (1 - y^{(i)}) \log(1 - (\sigma(\theta^T x^{(i)})))] =$$

$$\frac{1}{m} \sum_{i=0}^{m} x_k^{(i)} (\sigma(\theta^T x^{(i)}) - y^{(i)})$$

Batch gradient descent

Parameter updating, pseudocode:

```
Repeat until convergence:
    for k = 0 to n:
        gradient[k] = (1/m) Σ_{i=1}^{m} x_k^(i) (σ(θ^T x^(i)) − y^(i))
    for k = 0 to n:
        θ[k] = θ[k] − α*gradient[k]
```

Convergence = the gradient is close to the zero vector = the sum of squares of the `gradient[k]` is smaller than some ϵ.

Simultaneous updating

Simultaneous updating of all the θs is necessary for a correct result!

```
Repeat until convergence:
    for k = 0 to n:
        gradient[k] = 1/m Σ_{i=1}^{m} x_k^(i) (σ(θ^T x^(i)) − y^(i))
    for k = 0 to n:
        θ[k] = θ[k] − α*gradient[k]
```

⇓ This ⇓ is the wrong approach!! ⇓

```
Repeat until convergence:
    for k = 0 to n:
        θ[k] = θ[k] − α 1/m Σ_{i=1}^{m} x_k^(i) (h_θ(x^(i)) − y^(i))
```

Variants of gradient descent

Batch gradient descent usually requires a long time to reach convergence.

In every mini-update to θ, we compute a sum over **all** data points.

Two faster alternatives:

- Stochastic gradient descent
- Minibatch gradient descent

However, batch gradient descent guarantees that we come closer to the minimum **in every step**.

The alternatives don't.

Stochastic gradient descent

Parameter updating, Pseudocode:

```
Repeat until stopping_criterion:
    Select i randomly, 0 ≤ i ≤ m:
    for k = 0 to n:
        gradient[k] = x_k^(i) (σ(θ^T x^(i)) − y^(i))
    for k = 0 to n:
        θ[k] = θ[k] − α*gradient[k]
```

Stochastic gradient descent will (usually) produce a solution faster than batch gradient descent.

Quiz: Stochastic gradient descent, logistic regression

Suppose we have the following dataset

$$x = \begin{pmatrix} 1 \\ 0.82 \end{pmatrix} \quad y = 1$$

and the following parameter values:

$$\theta = \begin{pmatrix} 0.04 \\ 0.5 \end{pmatrix}$$

and learning rate $\alpha = 0.01$. Compute 1 step of stochastic gradient descent.

Quiz: Gradient descent, logistic regression

Gradienten:

$$x(\sigma(\theta^T x) - y) =$$

$$\begin{pmatrix} 1 \\ 0.82 \end{pmatrix} \left(\frac{1}{1+e^{-(0.04 \cdot 1 + 0.5 \cdot 0.82)}} - 1\right) =$$

$$\begin{pmatrix} -0.39 \\ -0.32 \end{pmatrix}$$

Gradient descent =

$$\theta = \begin{pmatrix} 0.04 \\ 0.5 \end{pmatrix} - 0.01 \begin{pmatrix} -0.39 \\ -0.32 \end{pmatrix} = \begin{pmatrix} 0.0439 \\ 0.5032 \end{pmatrix}$$

Early stopping

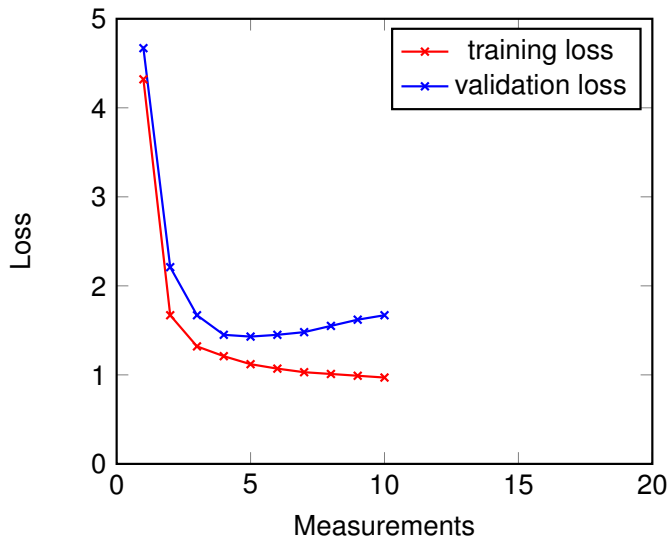
Split the data into 3 sets: **training**, **validation** (development), and **test** (usually 80-10-10).

At regular intervals, we compute the **training loss** and the **validation loss**.

If both decrease, everything is fine.

If training loss decreases, but **validation loss increases** for more than P consecutive measurements, this indicates **overfitting**, so stop training!

The hyperparameter P is called **patience**.



Minibatch gradient descent

Minibatch gradient descent is a trade-off between batch gradient descent (precise but slow) and stochastic gradient descent.

Select a number of data points (e.g. 100 data points), and do batch gradient descent on this subset. Then select another 100 data points, etc. Continue until convergence.

Learning in logistic regression

Summary:

- Represent data as n-ary vectors of features $x^{(i)}$.
- Represent correct labels as an m-ary vector y.
- The cross-entropy loss function computes the "badness" (loss) of a model (= a choice of parameters θ).
- Use an algorithm (like gradient descent) to find the minimum of the loss function.
- Minimal loss = optimal choice of parameters θ.

Regularization

The weights computed can sometimes be very large. It is an example of *overfitting*, where the model might generalize badly.

To avoid this, a *regularization term* can be added to the loss function:

$$\hat{L}(\theta) = L(\theta) + \lambda \sum_{i=1}^{n} \theta_i^2$$

This regularization term prevents the parameters θ from becoming too large.

DD1418
8f: Multinomial logistic regression

Johan Boye, KTH

Softmax function

What if we have more than two classes?

Suppose we have a set of numbers $z_1, ..., z_n$.

The *softmax* function normalizes those numbers into a probability distribution:

$$\text{softmax}(z_k) = \frac{e^{z_k}}{\sum_{i=1}^{n} e^{z_i}}$$

So

$$\sum_{i=1}^{n} \text{softmax}(z_i) = 1$$

E.g.

$\text{softmax}((1.45, 0.33, -0.12, 0.01)) = (0.5647, 0.1842, 0.1175, 0.1338)$

Non-binary classification

A multinomial logistic regression classifier (or maximum entropy classifier) estimates a $(n+1)$-ary vector of parameters for each class j.

$$\theta_j = (\theta_{j,0}, \ldots, \theta_{j,n})$$

where n is the number of features.

We then estimate $P(y = k|x)$ as softmax$(\theta_k x)$. E.g. in the case of three classes:

$$P(y = 2|x) = \frac{e^{\theta_2 x}}{e^{\theta_1 x} + e^{\theta_2 x} + e^{\theta_3 x}}$$

Quiz: Multinomial logistic regression

We have the following parameters:

$$\theta = \begin{pmatrix} 0.45 & -0.12 & 0.14 & 1.3 \\ -0.7 & 0.9 & 0.68 & -0.31 \\ -0.26 & 0.05 & 0.12 & 0.51 \end{pmatrix}$$

and the following datapoint

$$x = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

How would the datapoint x be classified using the model θ?

Quiz: Multinomial logistic regression

First compute:

$$\theta x = \begin{pmatrix} 0.47 \\ 0.88 \\ -0.09 \end{pmatrix}$$

Applying softmax to these numbers yields

$$\text{softmax}(\theta x) = \begin{pmatrix} 0.3249 \\ 0.4895 \\ 0.1856 \end{pmatrix}$$

Thus the model would predict class 2 (since 0.4895 is the highest probability).

Multinomial cross-entropy loss function

In the multinomial case, the cross-entropy function is extended as follows:

$$Loss(x^{(i)}, y^{(i)}) = \begin{cases} -\log(P(y^{(i)} = k|x^{(i)}) & \text{if } y^{(i)} = k \\ 0 & \text{otherwise} \end{cases}$$

that is,

$$Loss(x^{(i)}, y^{(i)}) = \begin{cases} -\log(\text{softmax}(\theta_k x^{(i)})) & \text{if } y^{(i)} = k \\ 0 & \text{otherwise} \end{cases}$$

Total loss is obtained by averaging $Loss(x^{(i)}, y^{(i)})$ over all datapoints $(x^{(i)}, y^{(i)})$.

Gradient of the loss function (1)

Let us compute the gradient of the loss function w.r.t. every parameter vector θ_j. First rewrite:

$$-\log(\text{softmax}(\theta_k x)) = -\log \frac{e^{\theta_k x}}{\sum_m e^{\theta_m x}} = -(\log e^{\theta_k x} - \log \sum_m e^{\theta_m x}) =$$

$$= \log \sum_m e^{\theta_m x} - \log e^{\theta_k x} = \log \sum_m e^{\theta_m x} - \theta_k x$$

(We write x rather than $x^{(i)}$ to remove some clutter).

First consider the first term. We have:

$$\frac{\partial}{\partial \theta_j} \log \sum_m e^{\theta_m x} = \frac{1}{\sum_m e^{\theta_m x}} e^{\theta_j x} x = \text{softmax}(\theta_j x) x$$

Gradient of the loss function (2)

Now consider the second term. In the case where **j** $\neq$ **k**, then

$$\frac{\partial}{\partial \theta_j} \theta_k x = 0$$

because $\theta_k x$ is just a constant.

In the case where **j** $=$ **k**, we have:

$$\frac{\partial}{\partial \theta_k} \theta_k x = x$$

Gradient of the loss function (3)

To sum up, the gradient of $Loss(x^{(i)}, y^{(i)})$ w.r.t. θ_j is:

$$\frac{\partial Loss(x^{(i)}, y^{(i)})}{\partial \theta_j} = \begin{cases} x^{(i)}(\text{softmax}(\theta_j x^{(i)}) - 1) & \text{if } j = y^{(i)} \\ x^{(i)}\text{softmax}(\theta_j x^{(i)}) & \text{otherwise} \end{cases}$$

The gradient of total loss is obtained by averaging $\frac{\partial Loss(x^{(i)}, y^{(i)})}{\partial \theta_j}$ over all datapoints $(x^{(i)}, y^{(i)})$.

Note that the gradient above is a *vector* and not a scalar for each class j.

Gradient of the loss function (4)

The parameters θ form a matrix

$$\theta = \begin{pmatrix} \theta_{0,1} & \cdots & \theta_{n,1} \\ & \ddots & \\ \theta_{0,k} & \cdots & \theta_{n,k} \end{pmatrix}$$

The gradient of the loss function *Loss* w.r.t. θ is a full (Jacobian) matrix

$$\begin{pmatrix} \frac{\partial Loss}{\partial \theta_{0,1}} & \cdots & \frac{\partial Loss}{\partial \theta_{n,1}} \\ & \ddots & \\ \frac{\partial Loss}{\partial \theta_{0,k}} & \cdots & \frac{\partial Loss}{\partial \theta_{n,k}} \end{pmatrix}$$