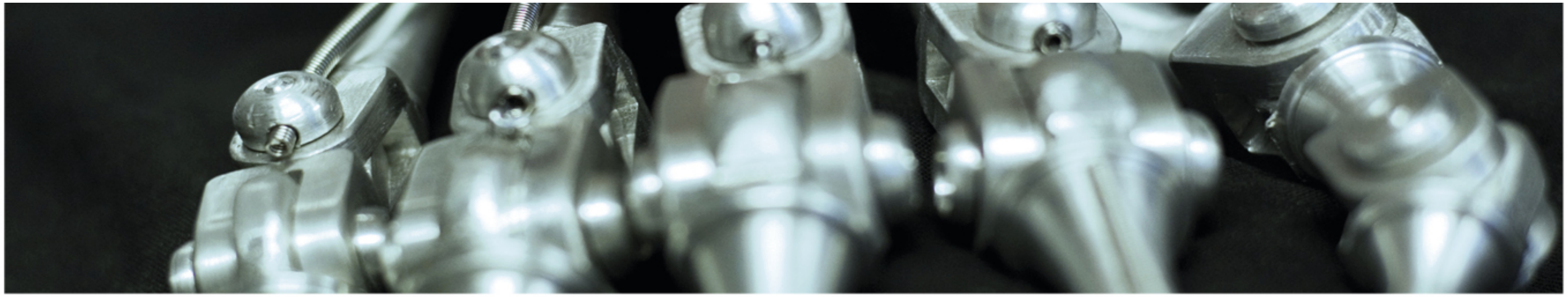


UiO : Department of Informatics
University of Oslo

INF 5860 Machine learning for image classification

Lecture 3 : Image classification and regression – part II

Anne Solberg

January 31, 2018

Today’s topics

- Multiclass logistic regression and softmax
- Regularization
- Image classification using a linear classifier.
- Link to probabilistic classifiers and SVM

Relevant additional video links:

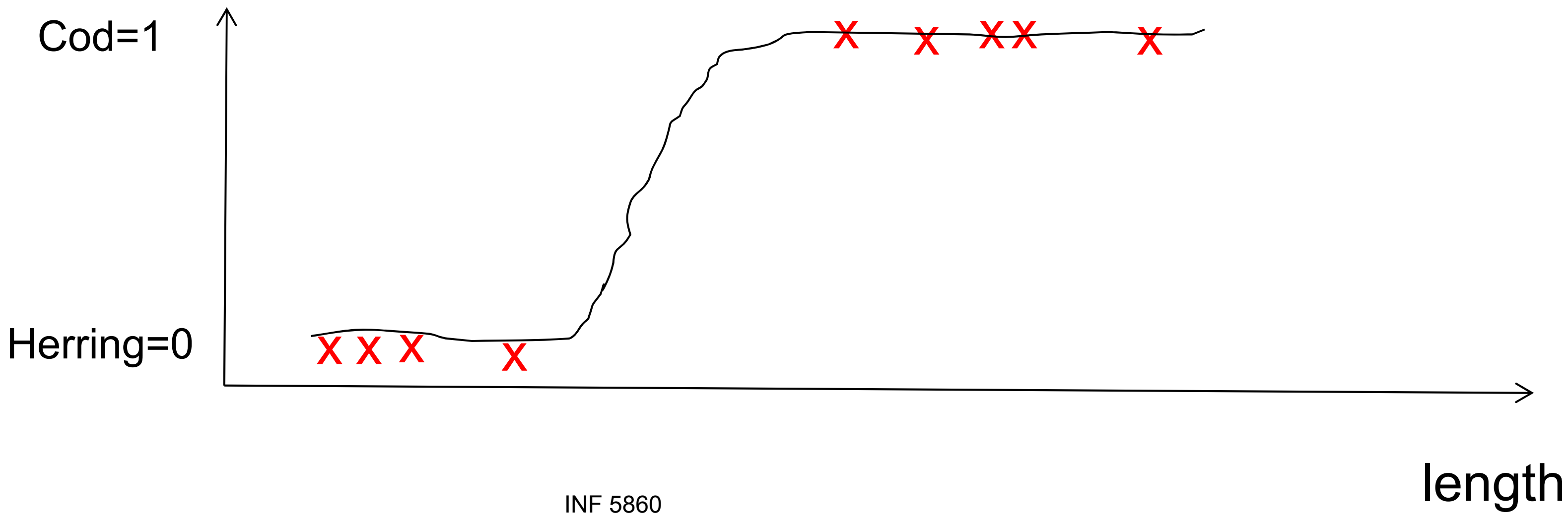
- https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv
  - Lecture 2 and 3
  - Remark: they do not cover regresion.

From last week: Introduction to logistic regression

- Let us show how a regression problem can be transformed into a binary (2-class) classification problem using a nonlinear loss function.
- Then generalize to multiple classes (next week).

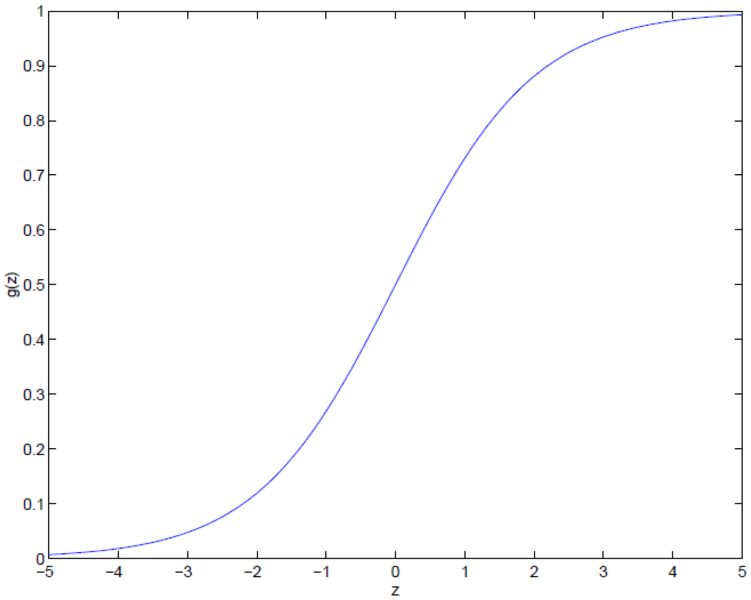
From last week: What if we fitted it to a function f(x) that is close to either 0 or 1?

- Hypothesis $h_\theta(x)$ is now a non-linear function of x
  Classification: y=0 or 1
  Threshold $h_\theta(x)$: if $h_\theta(x)>0.5$: set y=1, otherwise set y=0
- Desirable to have $h_\theta(x) \leq 1$



Logistic regression model

- Want $0 \leq h_\theta(x) \leq 1$ (binary problem)
- Let
- $h_\theta(x) = g(wx + b)$
- $g(z) = \frac{1}{1+e^{-z}}$
- $h_\theta(x) = \frac{1}{1+e^{-(wx+b)}}$
- g(z) is called the sigmoid function



Decisions for logistic regression

- Decide y=1 if $h_\theta(x)>$ 0.5, and y=0 otherwise

$$h_\theta(x) = g(wx + b)$$

$$h_\theta(X) = g\left(\theta^T x\right)$$

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

$$h_\theta(X) = \frac{1}{1+e^{-\theta^T x}}$$

- g(z)>0.5 if z>0
  - wx+b>0

g(z)<0 if z<0

wx+b<0

Here the compact notation θ means the vector of parameters [w,b]

Loss function for logistic regreesion

- We have two classes, 1 and 0.
- Let us use a probabilistic model
  Let the parameters be $\theta=[w_1,\ldots.w_{nk},b]$ if we have nk features.
- $P(y=1|x,) = h_\theta(x)$
- $P(y=0,x)= (1- h_\theta(x))$
- This can be written more compactly as
  $p(y|x, \theta) = h_\theta(x)^y(1- h_\theta(x))^{1-y}$

Loss function for logistic regreesion

- The likelihood of the parameter values is

$$
\begin{aligned}
L(\theta) &= p(\vec{y} \mid X; \theta) \\
&= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\
&= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}
\end{aligned}
$$

- It is easier to maximize the log-likelihood

$$
\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))
\end{aligned}
$$

- We will use gradient descent to maximize this, taking a step in the positive direction since we are maximizing, not minimizing

Computing the gradient of the likelihood function

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1-g(\theta^T x)} \right) g(\theta^T x)(1-g(\theta^T x) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= \left(y(1-g(\theta^T x)) - (1-y) g(\theta^T x) \right) x_j \\
&= \left(y - h_\theta(x) \right) x_j
\end{aligned}
$$

Here, we used the fact the g'(z)=g(z)(1-g(z))

Gradient descent of J(θ)=-L(θ)

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y(i)\log h_\theta(X(i,:) + (1-y(i))\log(1-h_\theta(X(i,:))\right]$$

To find θ: find θ that minimize J(θ) using gradient descent

Repeat :

$$\theta_j = \theta_j - \varepsilon \frac{\partial}{\partial \theta_j}$$

$$\theta_j - \varepsilon \frac{1}{m}\sum_{i=1}^{m}((h_\theta(X(i,:)) - y(i))X(i,j))$$

This algorithm looks similar to linear regression, but now

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

Overfitting and regularization

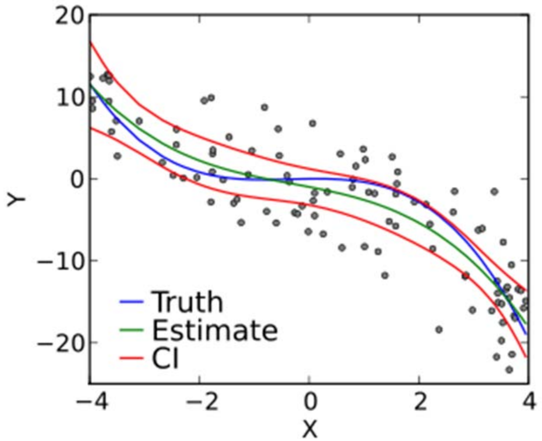
- For any classifier, it is a risk of overfitting to the training data.
- Overfitting:
  - High accuracy on training data
  - Lower accuracy on validation data.
- This risk is higher the more parameters the classifier can use.
-

Example: polynomial regression

- If a linear model is not sufficient, we can extend to allow higher-order terms or cross-terms between the variables by changing our hypothesis $h_\theta(x)$

$$h_\theta(x) = \theta^0 + \theta^1 x^1 + \theta^2 (x^1)^2 + \theta^3 (x^1)^3 \ldots$$

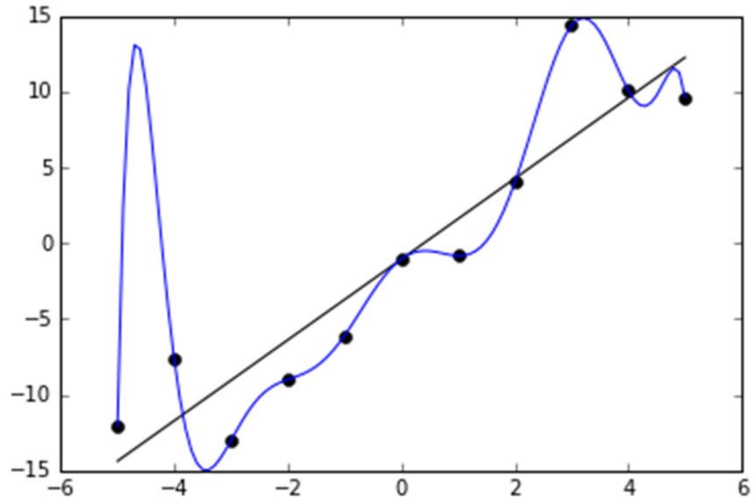
$$h_\theta(x) = \theta^0 + \theta^1 x^1 + \theta^2 \sqrt{x^1}$$



The danger of overfitting

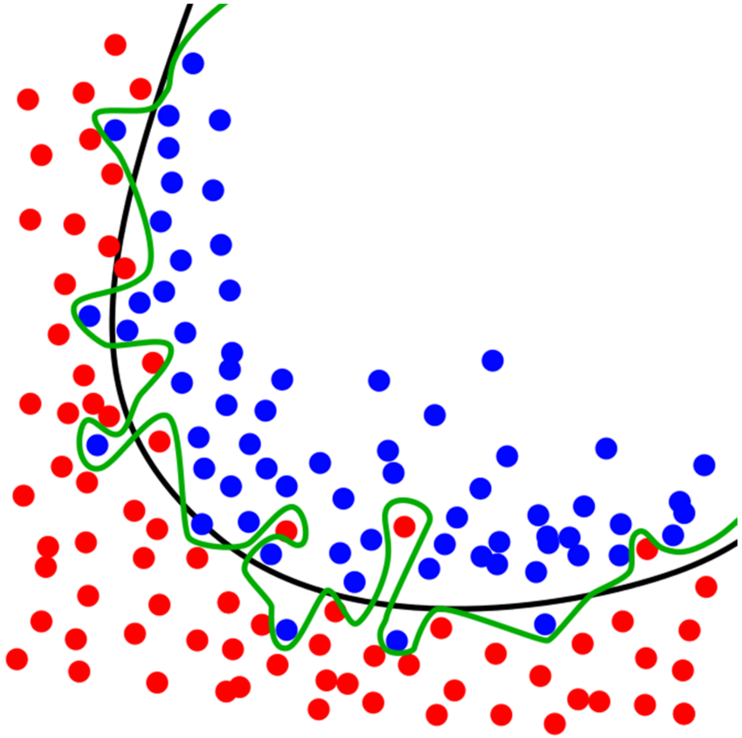
A higher-order model can easily overfit the training data
For the higher order terms:

- The higher the value of the coefficients, the more the curve can fluctuate
- This is not valid for the first two coefficients
- Restricting only the value of higher-order terms is difficult in general (e.g. for neural nets)
- But we can restrict the magnitude of the coefficients (except θ_0).

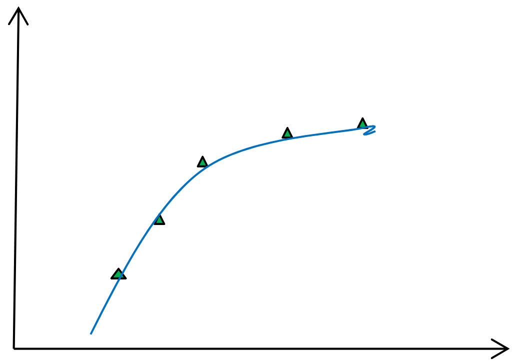


Overfitting for classification

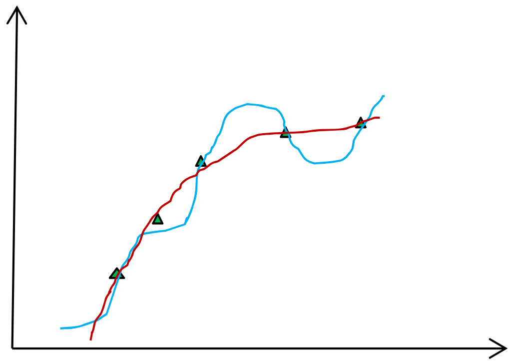
- Overfitting must be avoided for classifiation also – this is partly why we start with simple linear models



Regularization - intuition



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Suppose we add a penalty to restrict θ_3 and θ_4

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(X(i,:)) - y(i)\right)^2 + 100\theta_3 + +100\theta_4$$

To minimize, θ_3 and θ_4 must be small

Regularized cost function

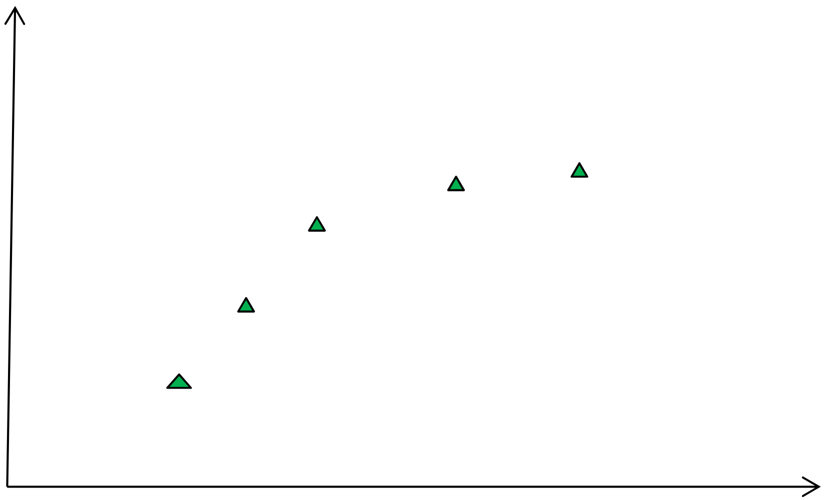
- Simplify the hypothesis by having small values for $\theta_1, \ldots \theta_n$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(X(i,:)) - y(i) \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n} \theta_j^{\,2}$$

- λ is the regularization parameter
- This is L2-regularization, later we will see
  - Dropout, max norm…
- Remark: we do not regularize the offset b (also called θ_0

What if λ is very large?

- Will we get overfit or underfit?



Gradient descent with regularization: linear regression

To find θ: find θ that minimize J(θ) using gradient descent

Note : NO penalty on θ_0

Repeat :

$$\theta_j = \theta_j - \varepsilon \frac{\partial}{\partial \theta_j}$$

$$\theta_0 = \theta_0 - \varepsilon \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,j))$$

$$\theta_j = \theta_j - \varepsilon \left[\frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,j)) + \frac{\lambda}{m} \theta_j \right]$$

$$= \theta_j \left(1 - \varepsilon \frac{\lambda}{m} \right) - \varepsilon \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,j))$$

Regularized logistic regression: gradient descent

Repeat :

$$\theta_0 = \theta_0 - \varepsilon \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,0))$$

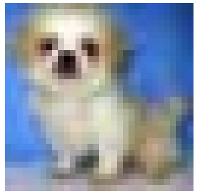
$$\theta_j = \theta_j - \varepsilon \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(X(i,:)) - y(i))X(i,j)) + \frac{\lambda}{m}\theta_j$$

$$j = 1,....n$$

$$h_\theta(X) = \frac{1}{1+e^{-\theta^T X}}$$

Introducing classifying CIFAR images

- CIFAR-10 images: 32x32x3 pixels



- Stack one image into a vector x of length 32x32x3=3072
- Classification will be to find a mapping f(W,x,b) from image space to a set of C classes.
- For CIFAR:

$$x = \begin{bmatrix} \text{pixel 1} \\ \\ \\ \text{pixel 3072} \end{bmatrix} \quad \mathrm{W} = \begin{bmatrix} \text{weight for pixel 1 for class 1} & \mathrm{K} & \text{weight for pixel 3072 for class 1} \\ \\ \\ \text{weight for pixel 1 for class 10} & & \text{weight for pixel 3072 for class 10} \end{bmatrix} \quad b = \begin{bmatrix} \text{b1} \\ \\ \\ \text{b10} \end{bmatrix}$$

Small example 2 classes

$$\text{Graylevel image} = \begin{bmatrix} 40 & 36 \\ 16 & 12 \end{bmatrix} \quad x = \begin{bmatrix} 40 \\ 36 \\ 16 \\ 12 \end{bmatrix} \quad W = \begin{bmatrix} 0.5 & -1.2 & 0.1 & 2.0 \\ 1.0 & 0.2 & -0.5 & 0.3 \end{bmatrix} \quad \mathrm{b} = \begin{bmatrix} 2.1 \\ 0.3 \end{bmatrix}$$

$$\begin{bmatrix} \text{Score for class 1} \\ \text{Score for class 2} \end{bmatrix} = \begin{bmatrix} 0.5 & -1.2 & 0.1 & 2.0 \\ 1.0 & 0.2 & -0.5 & 0.3 \end{bmatrix} \begin{bmatrix} 40 \\ 36 \\ 16 \\ 12 \end{bmatrix} + \begin{bmatrix} 2.1 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 4.5 \\ 43.1 \end{bmatrix}$$

W: 2x4

One weight w(i,j) for pixel j for class i

- If color image, append the r,g,b bands into one long vector x.
- Note: no spatial information concerning pixel neighbors is used here.
  - Convolutional nets use spatial information.
- All images are standarized to the same size!
  - For CIFAR-10 it is 32x32.
  - If a classifier is trained on CIFAR and we have a new image to classify, resize to 32x32.

W for multiclass image classification

- W is a Cx(n+1)-matrix (C classes, n pixels in the image plus 1 for b)
- We train one linear model pr. class, so each class has a different W(c,:)-vector
- If W(c,:) is a vector of length (n+1)

$$x = \begin{bmatrix} 1 \\ pixel1 \\ \\ \text{pixel } 3072 \end{bmatrix} \qquad \mathrm{W} = \begin{bmatrix} \mathrm{b}_1 & \text{weight for pixel 1 for class 1} & \ldots & \text{weight for pixel 3072 for class 1} \\ & & & \\ \mathrm{b}_C & \text{weight for pixel 1 for class C} & & \text{weight for pixel 3072 for class C} \end{bmatrix}$$

Let the score for class s_c be f(W,x)=W(c,:)x (b is included in W and x)

From 2 to C classes: alternative 1

- One vs. all classification:
  - Train a logistic classifier $h_{\theta,c}(x)$ for each class c to predict the probability for y=c.
  - Classify new sample x by picking the class c that maximize

$$\max_{c} h_{\theta,c}(x)$$

From 2 to multiple classes: Softmax

- The common generalization to multiple clasess is the **softmax classifier**.
- We want to predict the class label y_i={1,…C} for sample X(i,:), y can take one of C discrete values, so it follows a multinomial probability distribution.
- This is derived from an assumption that the probability/score of class y=k is

$$h_\theta(x) = p(y = k \mid x, \theta) = \frac{e^{\theta_k^T x}}{\sum_{j=1}^{C} e^{\theta_j^T x}}$$

Softmax prediction/classification

- Assign each sample to the class that maximize the score:

$$h_\theta(x) = p(y = k \mid x, \theta) = \frac{e^{\theta_k^T x}}{\sum_{j=1}^{C} e^{\theta_j^T x}}$$

Cross-entropy

- From information theory, the cross entropy between a true distribution p and an estimated distribution q is:

$$H(p,q) = -\sum_x p(x)\log q(x)$$

- Softmax minimize the cross-entropy between the estimated class probabilities and the 'true' distribution (the distribution where all the mass is in the correct class).

Softmax

- From a training data set with m samples, we formulate the log-likelihood function that the model fits the data:

$$l(\theta) = \sum_{i=1}^{m} \log(p(y_i \mid X(i,:), \theta))$$

- We can now find θ that maximize the likelihood using e.g. gradient ascent of the log-likelihood function.
  - Or we can minimize –l(θ) using gradient descent

- More details on deriving softmax next week (Ole-Johan)

Cross-entropy loss function for softmax

- The loss function for softmax, including regularization:

$x_i = X(i,:)^T$, the n pixel values for image i , let $\theta_{\mathrm{j}} = W(j,:)$, the row for class j

$$J(\theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{j=1}^{C} I(y_i = j)\log\left(\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{C} e^{\theta_l^T x_i}}\right)\right] + \frac{\lambda}{2}\sum_{i=1}^{C}\sum_{j=0}^{n} W(i,j)^2$$

- I(y=j) is the indicator function that is 1 if y=j and zero otherwise.
- See http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression

Softmax prediction example

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

exp →

normalize →

→ L_i = -log(0.13)
= **0.89**

unnormalized log probabilities

probabilities

Gradients of the cross entropy loss , including regularization

$x_i = X(i,:)^T$, the n pixel values for image i , let $\theta_j = W(j,:)$, the row for class j

$$\nabla J_{\theta_j} = -\frac{1}{n}\sum_{i=1}^{n} x_i (I(y_i = j) - p(y_i = j \mid x_i, W)) + \lambda\theta_j$$

For those who want calculus..

- Computing the derivative of the softmax function: see all details at
- https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/

Link to Gaussian classifiers

- In INF 4300, we used a traditional Gaussian classifier
  - This type of models is called generative models, where a specific distribution is assumed.

FROM INF 4300:Discriminant functions for the Gaussian density

- When finding the class with the highest probability, these functions are equivalent:

$$g_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})}$$

$$g_i(\mathbf{x}) = p(\mathbf{x} \mid \omega_i)P(\omega_i)$$

$$g_i(\mathbf{x}) = \ln p(\mathbf{x} \mid \omega_i) + \ln P(\omega_i)$$

- With a multivariate Gaussian we get:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^t \Sigma_i^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln\left|\Sigma_i\right| + \ln P(\omega_i)$$

- If we assume all classes have equal diagonal covariance matrix, the discriminant function is a linear function of x:

$$\frac{1}{(\sigma^2)}\boldsymbol{\mu}_j^T \mathbf{x} - \frac{1}{2(\sigma^2)}\boldsymbol{\mu}_j^T \boldsymbol{\mu}_j + \ln P(\omega_j)$$

Gaussian classifier vs. logistic regression

- These Gaussian with diagonal covariance and the logistic regression/softmax classifier will result in different linear decision boundaries.

- If the Gaussian assumption is correct, we will expect that this classifier has the lowest error rate.
- The logistic regresion might be better if the data is not entirely Gaussian.
- NOTE: SOFTMAX reduces to logistic regression if we have 2 classes.

Support Vector Machine classifiers

- Another popular classifier is the Support Vector Machine (SVM) formulation, which also can be formulated in terms of loss functions
- The following foils are for completeness, only a basic understand of the SVM as a maximum-margin classifier is expected in this course.

Hyperplanes and margins

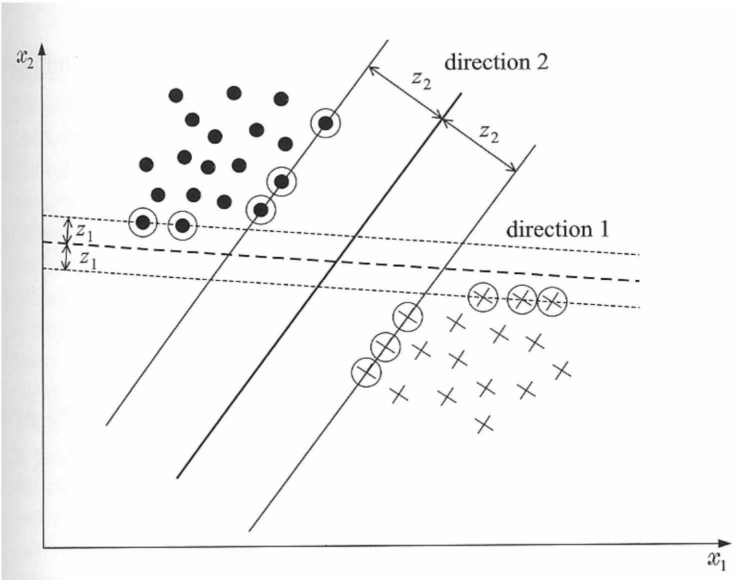
Background SVM

1. Have a margin of $\frac{1}{\|w\|}+\frac{1}{\|w\|}=\frac{2}{\|w\|}$

2. Require that all pixels are correctly classified:

$$w^T x + w_0 \geq 1, \quad \forall x \in \omega_1$$
$$w^T x + w_0 \leq -1, \quad \forall x \in \omega_2$$

- **Goal: find w and w_0**



Support Vector Machine loss

- A SVM loss function can be formulated by having as large margin as possible.
- This is generalized to multiple classes so the SVM 'wants' the correct class to have a score higher than the scores for the incorrect classes by som margin Δ
- If s_i is the score for class i, the loss function for SVM is

$$L_i = \sum_{j \neq i} \max(0, s_j - s_{y_i} + \Delta)$$

This is called the **hinge** loss

SVM and gradient descent

- We can also solve the SVM using gradient descent also, we will not cover this, but see http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf

FROM INF 4300:Discriminant functions for the Gaussian density

- When finding the class with the highest probability, these functions are equivalent:

$$g_i(\mathbf{x}) = P(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i)P(\omega_i)}{p(\mathbf{x})}$$

$$g_i(\mathbf{x}) = p(\mathbf{x} \mid \omega_i)P(\omega_i)$$

$$g_i(\mathbf{x}) = \ln p(\mathbf{x} \mid \omega_i) + \ln P(\omega_i)$$

- With a multivariate Gaussian we get:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^t \Sigma_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_i| + \ln P(\omega_i)$$

- If we assume all classes have equal diagonal covariance matrix, the discriminant function is a linear function of x:

$$\frac{1}{(\sigma^2)}\boldsymbol{\mu}_j^T\mathbf{x} - \frac{1}{2(\sigma^2)}\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j + \ln P(\omega_j)$$

Next week:

- Feed forward nets and learning by backpropagation
  - Reading material:
    - http://cs231n.github.io/neural-networks-1/
    - http://cs231n.github.io/neural-networks-2/
    - http://cs231n.github.io/optimization-2/
    - Deep learning Chapter 6