INTRODUCTION

IN 5400— Linear models for regression and classification

Anne Solberg

23.01.2019

University of Oslo

Plan for today

- Main focus: linear models for regression and classification
- Linear regression
- Logistic classification
- Softmax classification
- Loss functions
- Gradient descent optimization

Reading material and relevant video links:

- Note on linear models for classification and regression is linked here (pages 1-7 and 16-19)
- Optimization note: http://cs231n.github.io/optimization-1
- Relevant video links: Lecture 2 and 3 from CS 231n at Stanford, link here
  Note: they do not cover regression, but we do!

- Given a training set with input x and desired output y

$$\Omega_{\text{train}} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$$

- Create a function f that “approximates” this mapping

$$f(x) \approx y, \quad \forall (x, y) \in \Omega_{\text{train}}$$

- Hope that this generalises well to unseen examples, such that

$$f(x) = \hat{y} \approx y, \quad \forall (x, y) \in \Omega_{\text{test}}$$

  where Ω_{test} is a set of relevant unseen examples.
- Hope that this is also true for all unseen relevant examples.
- Today we approximate f based on **linear regression, logistic regression and softmax classification**.

Notation

- n_x: Input dimension
- n_y: Output dimension (number of classes)
- x, X, $\mathcal{X}$: Arrays representing input
- y, Y, $\mathcal{Y}$: Arrays representing *true* output
- $\tilde{y}$, $\tilde{Y}$, $\tilde{\mathcal{Y}}$: Arrays representing *one-hot encoded true* output.
- $\hat{y}$, $\hat{Y}$: Arrays representing *predicted* output
- Loss function: measures the discrepancy betwen the predicted and true output for one sample.
- Cost function: aggregated loss over all training samples.
- Subscript j or jk: Element in vector, or matrix
- Superscript with parenthesis (i): data example (i)
- Ω_{dataset}: A collection of examples $\{(x^{(i)}, y^{(i)})\}$ constituting a dataset.
- m: Number of examples

LINEAR REGRESSION

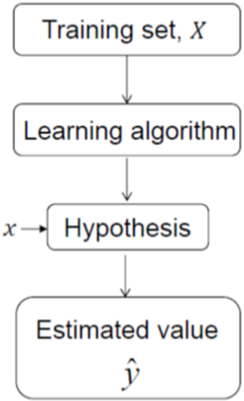
Introduction

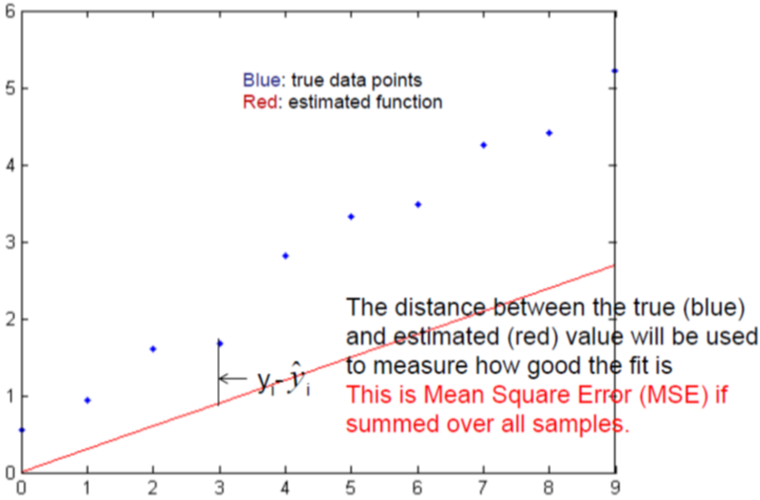
- Linear regression gives a nice introduction to neural nets
- Linear regression: predict a continuous value
- Logistic regression: binary classification, predict between two classes
- Softmax regression: a generalization to classification with multiple classes

Linear regression: training data

- Want to estimate y based on data x
- The data set Ω_{dataset} has m training samples $x^{(i)}$ with true values $y^{(i)}, 1 \leq i \leq m$

- Predict the y-values based on data x in the training data set.
- $\hat{y}$ denotes the predicted value.
- y is a continuous number.
- Linear hypotesis: $\hat{y} = wx + b$
- w and b are the unknown values that regression will estimate
- w has the same dimension as $x^{(i)}$, and b is a scalar in this case.
- Learning will be based on comparing y and $\hat{y}$, so we need a measure of how well the model fits the data.




Blue: true data points
Red: estimated function
$y_i - \hat{y}_i$
The distance between the true (blue) and estimated (red) value will be used to measure how good the fit is
This is Mean Square Error (MSE) if summed over all samples.
0
1
2
3
4
5
6
7
8
9

Error measure (cost function) for linear regression: Mean square error (MSE)

- Mean square error between the true and predicted value of y summed over the m samples in the training set.

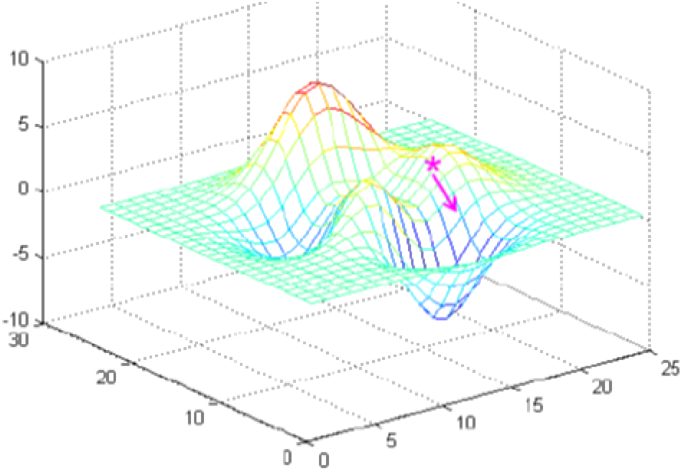
$$J(m,b) = MSE = \frac{1}{2m}\sum_{i=1}^{m}[\hat{y}^{(i)} - y^{(i)}]^2$$

- In vector form:

$$J(m,b) = \frac{1}{2m}||\hat{y} - y||_2^2, \ \textbf{L2-norm}$$

OPTIMIZATION

Start from a point and take a step downhill in the steepest possible direction.
Repeat this until we end up in a local minimum.
If we start from a neighboring point, we should end up in the same minimum.



Iterative minimization outline

- Have a function $J(w, b)$ (can be generalized to more than two parameters)
- Want to find the value of w and b that minimize $J(w, b)$
- Outline
  1. Start with some value of w, b, e.g. $w = 0, b = 0$.
  2. Compute $J(w, b)$ for the given value of w and b
  3. Change w and b in a manner that will decrease $J(w, d)$
  4. Repeat step 2-3 until we hopefully end up in a minimum

Gradient descent principle

- Given a function $J(w, b)$
- The directional derivative of $J(w, b)$ in a given direction is the slope of $J(w, b)$ in that direction
- To iteratively minimize $J(w, b)$, we want to find the direction in which $J(w, b)$ decreases fastest.
- This can be shown to be in the **the opposite direction** of the gradient
- So we can minimize $J(w, b)$ by taking a sted in the **direction of the negative gradient**
- Gradient descent propose a new point

$$w = w - \lambda \nabla_w J(w, b),$$

$$b = b - \lambda \nabla_b J(w, b)$$

- λ is the learning rate, if λ is too large, the algorithm may diverge, if λ is too small, the algorithm converges very slow

Gradient descent for linear regression, single (univariate) feature

- Let w and$b$ be the two unknown parameters in the linear model $y = wx + b$
- We want to minimize the mean square error between the true values and the predictions, $J(w, b)$

$$J(w,b) = \frac{1}{2m}\sum_i(\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m}\sum_i(wx^{(i)} + b - y^{(i)})^2$$

- Let us find the partial derivatives of $J(w, b)$ with respect to w and b

$$\frac{\partial}{\partial w}J(w,b) = \frac{\partial}{\partial w}\frac{1}{2m}\sum_i(wx^{(i)} + b - y^{(i)})^2 = \frac{2}{2m}\sum_i(wx^{(i)} + b - y^{(i)})x^{(i)}$$

$$\frac{\partial}{\partial b}J(w,b) = \frac{\partial}{\partial b}\frac{1}{2m}\sum_i(wx^{(i)} + b - y^{(i)})^2 = \frac{2}{2m}\sum_i(wx^{(i)} + b - y^{(i)})$$

- Here, we sum the gradient over **all samples in the training data set**. This is called **batch gradient descent**.
- Remark: This simple problem is quadratic and could be solved analytically, but we will seek an iterative solution

Gradient descent updates for linear regression

- Linear regression model $y = wx + b$
- Gradient descent: repeat until convergence

$$w = w - \lambda \frac{\partial J}{\partial w} = w - \lambda \frac{1}{m} \sum_i [wx^{(i)} + b - y^{(i)}] x^{(i)}$$

$$b = b - \lambda \frac{\partial J}{\partial b} = b - \lambda \frac{1}{m} \sum_i [wx^{(i)} + b - y^{(i)}]$$

Checkpoint: verify that you can derive these equations!

Implementing gradient descent

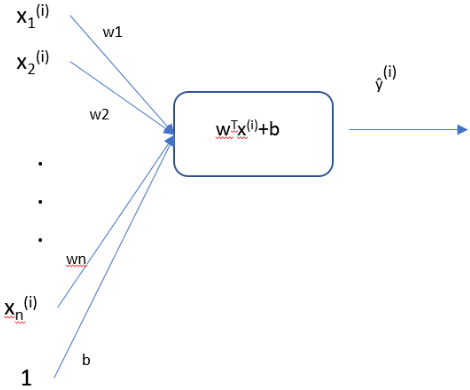
- The sum over all samples $x^{(i)}$ can be done on vectors using np.sum()

$$w = w - \lambda \frac{\partial J}{\partial w} = w - \lambda \frac{1}{m} \sum_i [wx^{(i)} + b - y^{(i)}] x^{(i)}$$

$$b = b - \lambda \frac{\partial J}{\partial b} = b - \lambda \frac{1}{m} \sum_i [wx^{(i)} + b - y^{(i)}]$$

Computational graph for prediction using linear regression

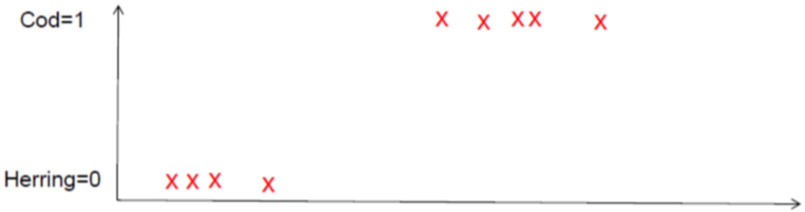
- The graph shows how to predict new samples



- Corresponing graphs can be drawn for loss function also
- Computational graphs are useful for gradient computation also, more next week

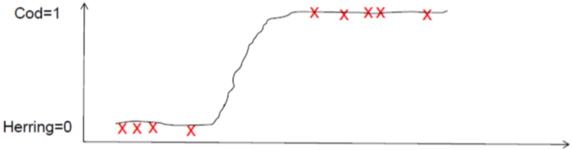
LOGISTIC REGRESSION

- Let us see how a regression problem can be transformed into a binary (2-class) classification problem using a nonlinear loss function.
- Below is an example:
- In this example deciding either class 1 or 0 could be done by introducing a threshold.
- An alternative way is to do a nonlinear transform that squeeze the data to either 0 or 1 (next)

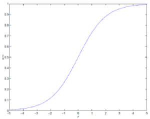


Fitting with a sigmoid function

- Now introduce $g(wx+b)$, a nonlinear function of x.
- In classification we want $y=1$ or $y=0$
- In this example deciding either class 1 or 0 could be done by introducing a threshold
- An alternative is to do a nonlinear transform that squeeze the data to either 0 or 1



- Let $g(z) = \frac{1}{1+e^{-(wx+b)}}$
- $g(z)$ is called the sigmoid function



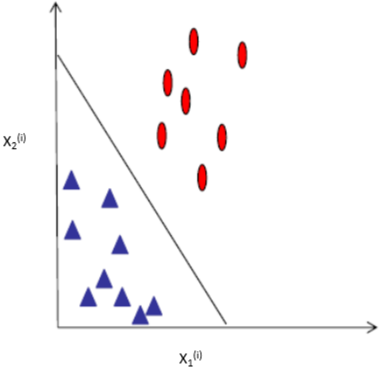
Logistic regression model with multiple features

- We can have multiple (n_x) features for each samples $x = x_j^{(i)}, j \in 1, n_x$.
- Decide and assign class $y = 1$ if

$$g(\sum_j w_j x_j^{(i)} + b) \geq 0.5$$

  otherwise decide $y = 0$.
- This gives a linear decision boundary for the classification
- If we want a non-linear boundary, we could add higher order combinations of the features
- In classification we want $y = 1$ or $y = 0$

A 2D EXAMPLE



- $g(w_1x_1 + w_2x_2 + b)$
- Predict $y = 1$ if $g(w_1x_1 + w_2x_2 + b) \geq 0.5$, 0 otherwize
- Decision boundary: $w_1x_1 + w_2x_2 + b = 0$ and $g(w_1x_1 + w_2x_2 + b) = 0.5$
- If we know w_1, w_2, b, decision is based on which side of the boundary we are.

Introducing a logistic cost function

- Assume we have a set of m 2D training images $F^{(i)}(b, j, k)$
- Each image has size $C \times N_J \times N_K$, where C is the number of bands in the image (e.g. 3 for a RGB-image).
- For convenience, reshape these into a 1D vector $x^{(i)}$ of length $1 \times (C \times N_J \times N_K)$
- The true class labels for each image in the training set is $y^{(i)}$.

Logistic loss per sample

- We need a function to measure the classification accuracy of the model, but using the accuracy as a binary variable does not work so well.
- Instead, let the cost function for one sample be such that:
- Loss = 0 if $y^{(i)} = 1$ and $g\left(\sum_j w_j x_j^{(i)} + b\right) = 1$
- Loss = 0 if $y^{(i)} = 0$ and $g\left(\sum_j w_j x_j^{(i)} + b\right) = 0$
- Loss $\rightarrow \infty$ if $y^{(i)} = 1$ and $g\left(\sum_j w_j x_j^{(i)} + b\right) \rightarrow 0$
- Loss $\rightarrow \infty$ if $y^{(i)} = 0$ and $g\left(\sum_j w_j x_j^{(i)} + b\right) \rightarrow 1$

Logistic loss per sample

- The sigmoid $g\left(\sum_j w_j x_j^{(i)} + b\right)$ will mimic a probability and give a number between 0 and 1.
- Let us assume that

$$P\left(y^{(i)} = 1 | x^{(i)}, w, b\right) = g\left(\sum_j w_j x_j^{(i)} + b\right)$$

$$P\left(y^{(i)} = 0 | x^{(i)}, w, b\right) = 1 - g\left(\sum_j w_j x_j^{(i)} + b\right)$$

- This can be written more compactly as:

$$p\left(y^{(i)} | x^{(i)}, w, b\right) = g\left(\sum_j w_j x_j^{(i)} + b\right)^{y^{(i)}} \left[1 - g\left(\sum_j w_j x_j^{(i)} + b\right)\right]^{1-y^{(i)}}$$

Checkpoint: verify that this is the same as the equation above

Logistic cost function

- The cost function $L(w,b)$ considers the loss over all samples
- Assume that the samples are independent, we want to maximize the likelihood of the parameters:

$$
\begin{aligned}
L\left(w,b\right) &= \prod_{i=1}^{m} p\left(y^{(i)}|x^{(i)},w,b\right) \\
&= \prod_{i=1}^{m} g\left(\sum_j w_j x_j^{(i)} + b\right)^{y^{(i)}} \left[1 - g\left(\sum_j w_j x_j^{(i)} + b\right)\right]^{1-y^{(i)}}
\end{aligned}
$$

- It is easier to **maximize the log likelihood**:

$$
\begin{aligned}
l\left(w,b\right) &= \log L\left(w,b\right) \\
&= \sum_{i=1}^{m} y^{(i)} \log\left(g\left(\sum_j w_j x_j^{(i)} + b\right)\right) + (1 - y^{(i)}) \log\left[1 - g\left(\sum_j w_j x_j^{(i)} + b\right)\right]
\end{aligned}
$$

- We will use **gradient descent to minimize** $-l\left(w,b\right)$.

Derivative of the sigmoid function

For gradient descent of the log likelihood, we need the derivative of the sigmoid function

$$g^{'}(z) = \frac{d}{dz}\frac{1}{1+e^{-z}} = \frac{1}{(1+e^{-z})^2}(e^{-z}) = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = g(z)(1-g(z))$$

Note that we can actually compute the derivative by using the function itself. This will come in handy later.

Derivative of logistic loss for a single sample

- Consider one sample and a univariate model with a single feature.
- The loss for one sample with this model is

$$l\left(w,b\right)^{(i)}=y^{(i)}\log\left(g(wx^{(i)}+b)\right)+(1-y^{(i)})\log\left[1-g(wx^{(i)}+b)\right]$$

- We need the derivative with respect to w and b.

$$\frac{\partial}{\partial w}l(w,b)=\left(y^{(i)}\frac{1}{g(wx^{(i)}+b)}-[1-y^{(i)}]\frac{1}{1-g(wx^{(i)}+b)}\right)\frac{\partial}{\partial w}g(wx^{(i)}+b)$$

$$=\left(y^{(i)}\frac{1}{g(wx^{(i)}+b)}-[1-y^{(i)}]\frac{1}{1-g(wx^{(i)}+b)}\right)g(wx^{(i)}+b)(1-g(wx^{(i)}+b))\frac{\partial}{\partial w}(wx^{(i)}+b)$$

$$=[y^{(i)}\left(1-g(wx^{(i)}+b)\right)-(1-y^{(i)})(g(wx^{(i)}+b)]x^{(i)}$$

$$=[y^{(i)}-g(wx^{(i)}+b)]x^{(i)}$$

- **Verify this!**

Derivative of logistic loss for a single sample

- Correspondingly we find the derivative with respect to b:

$$\frac{\partial}{\partial b} l(w,b) = \left(y^{(i)} \frac{1}{g(wx^{(i)}+b)} - [1-y^{(i)}] \frac{1}{1-g(wx^{(i)}+b)} \right) \frac{\partial}{\partial b} g(wx^{(i)}+b)$$

$$= \left(y^{(i)} \frac{1}{g(wx^{(i)}+b)} - [1-y^{(i)}] \frac{1}{1-g(wx^{(i)}+b)} \right) g(wx^{(i)}+b)(1-g(wx^{(i)}+b)) \frac{\partial}{\partial b}(wx^{(i)}+b)$$

$$= [y^{(i)} \left(1-g(wx^{(i)}+b)\right) - (1-y^{(i)})(g(wx^{(i)}+b)]$$

$$= [y^{(i)} - g(wx^{(i)}+b)]$$

- **Verify this!**

Gradient descent for logistic classifier

- Given the cost function for all samples:

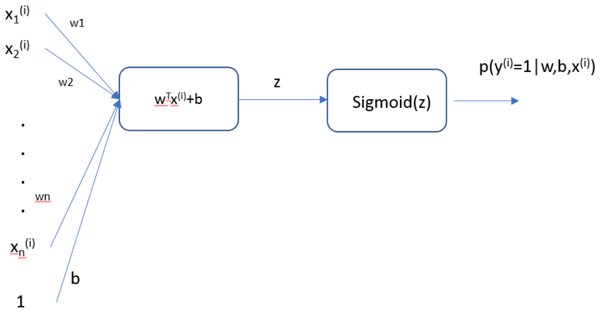
$$l\left(w,b\right) = -\sum_{i=1}^{m} y^{(i)} \log\left(g(wx^{(i)}+b)\right) + (1-y^{(i)}) \log\left[1-g(wx^{(i)}+b)\right]$$

- Find w and b using gradient descent to maximize the likelihood:
- Repeat

$$w = w - \lambda \frac{1}{m} \sum_{i=1}^{m} [y^{(i)} - g(wx^{(i)}+b)]x^{(i)}$$

$$b = b - \lambda \frac{1}{m} \sum_{i=1}^{m} [y^{(i)} - g(wx^{(i)}+b)]$$

- The graph shows how to predict new samples



- Choose class 1 if $p(y^{(i)} = 1|w, b, x^{(i)}) \geq 0.5$, otherwise class 0

SOFTMAX CLASSIFICATION AND CROSS ENTROPY COST

From 2 to n_y classes using Softmax

- In a logistic classifier, we apply a sigmoid function to $z = wx + b$, to predict output close to 0 for class 0 and output close to 1 for class 1.
- In the generalization to multiple classes, we will approximate a probability between 0 and 1 for each class, and choose the class with the highest probability.
- We use the softmax function for this.
- Given sample $x^{(i)}$, we want to predict the class label $y^{(i)} \in [1, ..., n_y]$ as one of n_y predefined classes.
- The true class labels for the training data set is known.
- $y(i)$ can take of of n_y discrete values and follows a multinomial distrubtion.
- We assume that the probability (or score) that $y(i) = k$ is

$$p(y^{(i)} = k | x^{(i)}, W[:, k], b[k]) = \frac{e^{W[:,k]^T x(i) + b[k]}}{\sum_{c=1}^{n_y} e^{W[:,c]^T x(i) + b[k]}}$$

- Note that we will fit one set of weights $W[:, k], b[k]$ to each class.

Prediction/classification with Softmax

- Given a trained model with weights W, b
- W is a matrix of size $[n_x, n_y]$ with one row for each input dimension, and one column per class, and b is a vector of length n_y.
- $W[:, k]$ and $b[k]$ corresponds to the weights for class k.
- The probability or score for class k is:

$$\hat{y}_k^{(i)} = p(y^{(i)} = k | x^{(i)}, W[: k], b[k]) = \frac{e^{(W[:,k]^T x(i) + b[k])}}{\sum_{c=1}^{n_y} e^{(W[:,c]^T x(i) + b[c])}}$$

- Compute $p(y^{(i)} = k | x^{(i)}, W[:, k], b[k])$ for all classes and choose the class that maximize it

Implementing the softmax function

- Numerical instability can be a problem, because of the exponential function, and division.
- Two common “tricks” that can help this follows
- Shift exponential arguments to max zero

$$s_k(z) = \frac{e^{z_k}}{\sum_{i=1}^{n} e^{z_i}}$$
$$= \frac{e^{z_k - \max(z)}}{\sum_{i=1}^{n} e^{z_i - \max(z)}}$$

- Take logarithm and exponentiate it to get rid of division

$$t(z)_k = \log s(z)_k$$
$$= z_k - \log \sum_{i=1}^{n} e^{z_i}$$
$$s(z)_k = e^{t(z)_k}$$

- The above can be combined

One-hot encoding y

- Since we will compare the predicted output for each class, $\hat{y_k^{(i)}}$ to the true label $y^{(i)}$, it is conventient to use the *one-hot encoded* vector $\tilde{y}^{(i)}$:

$$\tilde{y}_k(i) = \begin{cases} 1, & \text{if} \quad y^{(i)} = k \\ 0, & \text{else} \end{cases} \tag{1}$$

- A compact notation for the predicted score/probability for all classes is then:

$$p_{\mathcal{Y}}(y|\mathcal{X} = x; \Theta) = \prod_{k=1}^{n_y} \hat{y}(x; \Theta)_k^{\tilde{y}_k} \tag{2}$$

Cross-entropy cost function for learning $\Theta = [W, b]$

- We will use gradient descent to fit W and b to the training data set.
- The cost function will be the cross entropy between the predicted and the true class labels (this will be derived next lecture):

$$\hat{\Theta} = \arg\min_{\Theta} \left\{ -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{n_y} \tilde{y}_k^{(i)} \log \hat{y}_k^{(i)} \right\}.$$

- The optimization objective function will therefore be the *cross entropy cost*

$$\mathcal{C}(\Theta, \Omega^y_{\text{train}}, \Omega^x_{\text{train}}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{n_y} \tilde{y}_k^{(i)} \log \hat{y}_k^{(i)}. \quad (3)$$

- We will reserve loss function to the discrepancy between the predicted and true output for a
- Our *cross entropy loss* is then (for a single sample we sum over alll classes k)

$$\mathcal{L}(y^{(i)}, \hat{y}^{(i)}) = -\sum_{k=1}^{n_y} \tilde{y}_k^{(i)} \log \hat{y}_k^{(i)}. \quad (4)$$

Working on a batch of samples

- In gradient descent, updating the weights for each sample is time consuming.
- We organize the traning data in batches, and update the cost function summed over each batch of m_b samples

$$\mathcal{C}_b = \frac{1}{m_b} \sum_{i=1}^{m_b} \sum_{k=1}^{n_y} \tilde{y}_k^{(i)} \log \hat{y}_k^{(i)}$$

- We can predict a batch of samples $X = x^{(i)}, i \in [1, m_b]$ using a dot product between X and W then add b.

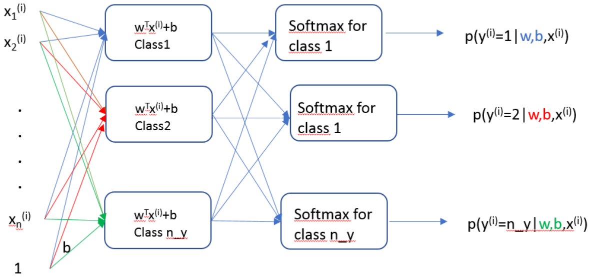
Gradient descent softmax updates

- Gradient descent updates are very similiar to logistic classfication
- Next we we derive the gradients for batches and look at vectorized implementations. Here is the per sample gradient updates:

$$
\begin{aligned}
W^{(i)} &= W^{(i)} - \lambda \nabla_{W^{(i)}} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) \\
&= W^{(i)} - \lambda \left(\hat{y}^{(i)} - \tilde{y}_k^{(i)} \right) x^{(i)}
\end{aligned}
$$

$$
\begin{aligned}
b^{(i)} &= b^{(i)} - \lambda \nabla_{b_{(i)}} \mathcal{L}(y^{(i)}, \hat{y}^{(i)}) \\
&= b^{(i)} - \lambda \left(\hat{y}^{(i)} - \tilde{y}_k^{(i)} \right)
\end{aligned}
$$

- The graph shows how to predict new samples



- Choose the class k that has the highest probability if $p(y^{(i)}k1|w,b,x^{(i)}) \geq 0.5$

Comment about image classification

- Reshape the image into a long 1D-vector.
- If color image, append the (r,g,b)-channels also into the vector.
- Note: no spatial information concering pixel neighbors is used here.
- All images must be standardized to the same size!
- If a classifier is trained on CIFAR-10 images of size 32×32 rgb-images, all testimages must also be resized to 32×32 rgb-images

Learning goals

- Understand linear regression and the loss function
- Be able to compute by hand and implement the gradient descent updates
- Understand logistic regression and the loss function
- Be able to compute by hand and implement the logistic gradient descent updates
- Understand softmax classification
- Cross-entropy loss will be derived in detail next week
- Implement softmax and gradient descents for cross-entropy loss
    - This will come in handy for Mandatory 1
- Theory exercises relevant for exam

Next week

- Feed forward nets and learning by backpropagation including vectorization
- Reading material
  - Note on backpropagation: http://cs231n.github.io/optimization-2/
  - Neural networks introduction: http://cs231n.github.io/neural-networks-1/

Questions?