Linear Models for Classification

Numerical Methods for Deep Learning

Learning Objective: Linear Classifiers

In this module we review computational methods for linear classification.

Learning tasks:

- binary classification $\rightsquigarrow$ logistic regression
- multiclass problems $\rightsquigarrow$ multinomial logistic regression

Numerical methods:

- convex optimization
- steepest descent
- Newton's method

Logistic Regression

Assume our data falls into two classes. Denote by $\mathbf{c}_{\text{obs}}(\mathbf{y})$ the probability that example $\mathbf{y} \in \mathbb{R}^{n_f}$ belongs to first category.

Since output of our classifier $f(\mathbf{y}, \boldsymbol{\theta})$ is supposed to be a probability, use logistic sigmoid

$$\mathbf{c}(\mathbf{y}, \boldsymbol{\theta}) = \frac{1}{1 + \exp\left(-f(\mathbf{y}, \boldsymbol{\theta})\right)}.$$

Example (Linear Classification): If $f(\mathbf{y}, \boldsymbol{\theta})$ is a linear function (adding bias is easy), $\boldsymbol{\theta} = \mathbf{w} \in \mathbb{R}^{n_f}$ and

$$\mathbf{c}(\mathbf{y}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{y})}.$$

this module: consider linear models and focus on loss

Multinomial Logistic Regression

Suppose data falls into $n_c \geq 2$ categories and the components of $\mathbf{c}_{\text{obs}}(\mathbf{y}) \in \Delta_{n_c}$ contain probabilities for each class.

Δ_{n_c} is the unit simplex in $\mathbb{R}^{n_c}$

$$\Delta_{n_c} = \{\mathbf{c} \in [0,1]^{n_c} \ : \ \mathbf{e}_{n_c}^\top \mathbf{c} = 1\}.$$

Applying the logistic sigmoid to each component of $f(\mathbf{y}, \mathbf{W})$ not enough to be in Δ_{n_c}. Use

$$\mathbf{c}(\mathbf{y}, \mathbf{W}) = \left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{W}\mathbf{y})} \right) \exp(\mathbf{W}\mathbf{y}).$$

Note: Division and exp are applied element-wise!

Logistic Regression - Loss Function

How similar are $\mathbf{c}(\cdot, \mathbf{W})$ and $\mathbf{c}_{\text{obs}}(\cdot)$?

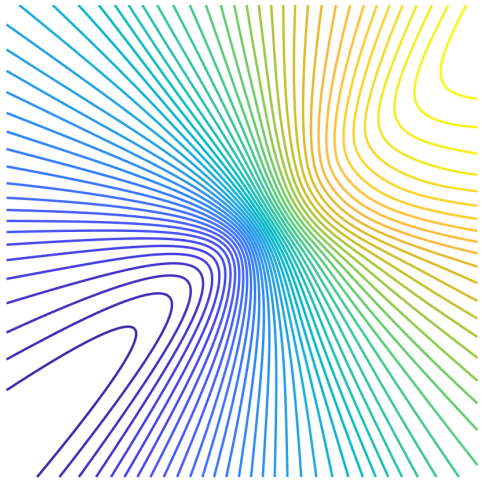
Naive idea: Let $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$ be examples with class probabilities $\mathbf{C}_{\text{obs}} \in [0,1]^{n_c \times n}$, use

$$\phi(\mathbf{W}) = \frac{1}{2n} \sum_{j=1}^{n} \|\mathbf{c}(\mathbf{y}_j, \mathbf{W}) - \mathbf{c}_{j,\text{obs}}\|_F^2$$

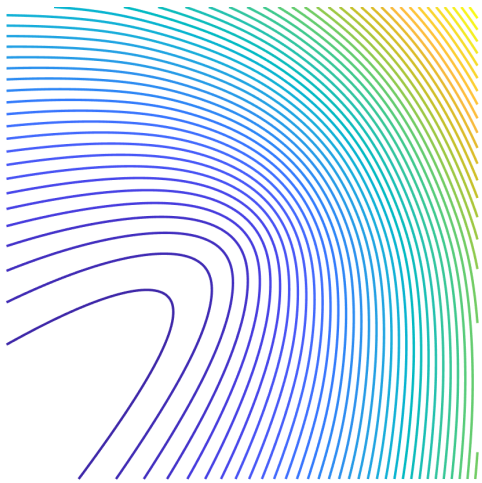
Problems

- ignores that $\mathbf{c}(\cdot, \mathbf{W})$ and $\mathbf{c}_{\text{obs}}(\cdot)$ are distributions.
- leads to non-convex objective function

Frobenius



Cross Entropy



Example: Designing a Code

Goal: Communicate using minimal number of bits.

Example: Bob talks $\mathbf{c} = [1/2, 1/4, 1/8, 1/8]$ of the time about dogs, cats, fish, and birds, respectively.
How many bits need to be transferred on average?

word	naive code	better code
dog	00	0
cat	01	10
fish	10	110
bird	11	111

Idea: Quantify information content in probability distribution using average length.

Entropy

Note: Length of word depends on its probability being used. How long should a word be?

Optimal choice for information for any category

$$I = \log_2(\mathbf{c}_j^{-1}) = -\log_2(\mathbf{c}_j)$$

The larger $\mathbf{c}_j$, the more frequent we use it, the shorter the word should be.

The entropy is the average (expectation) of information over all classes.

$$E(\mathbf{c}) = -\sum \mathbf{c}_j \log_2(\mathbf{c}_j) = -\mathbf{c}^\top \log_2(\mathbf{c})$$

Example: Designing a Code - 2

Entropy for Bob's code is

$$\frac{1}{2}\log(2) + \frac{1}{4}\log(4) + 2\frac{1}{8}\log(8) = 1.75$$

average length of word is 1.75 bits $<$ 2 bits for naive code!

For the complete tutorial on entropy, read
`http://colah.github.io/posts/2015-09-Visual-Information/`

Properties of Entropy

- recall $\lim_{x \to 0} x \log x = 0$
- prefer sparse distributions (why?)
- has been used in compressed sensing type methods

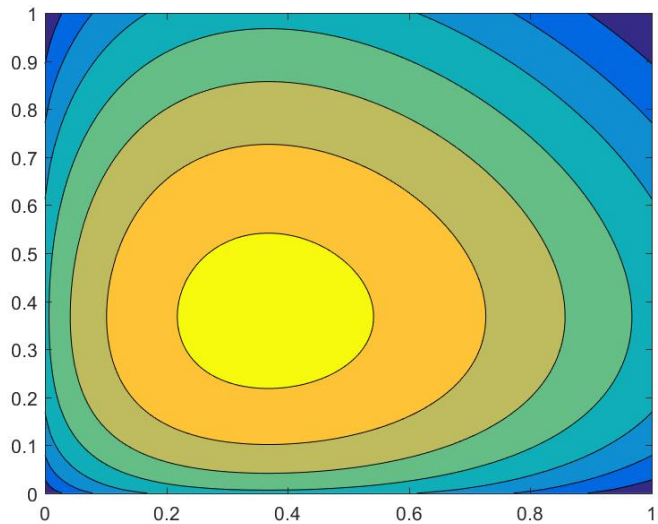


Figure: The entropy of a vector $\mathbf{c} = (c_1, c_2)^\top$

Cross Entropy

Measure the average word length when using code designed for $\mathbf{c}$ for sending information with probability $\widehat{\mathbf{c}}$

$$E(\widehat{\mathbf{c}}, \mathbf{c}) = -\widehat{\mathbf{c}}^\top \log(\mathbf{c}).$$

Clearly

$$E(\widehat{\mathbf{c}}, \mathbf{c}) \geq E(\mathbf{c}, \mathbf{c})$$

Example: Alice talks $\mathbf{c} = [1/8, 1/2, 1/4, 1/8]$ of the time about dogs, cats, fish, and birds, respectively. If she used Bob's code, the average word length would be

$$\frac{1}{8}\log(2) + \frac{1}{2}\log(4) + \frac{1}{4}\log(8) + \frac{1}{8}\log(8) = 2.25 > 1.75$$

E measures how similar the distributions $\mathbf{c}$ and $\widehat{\mathbf{c}}$ are.

One flaw: $E(\mathbf{c}, \widehat{\mathbf{c}}) \neq E(\widehat{\mathbf{c}}, \mathbf{c})$ (verify for our example!)

Cross Entropy for Logistic Regression - 1

Recall: For a single example and two classes we have

$$\mathbf{c}(\mathbf{y}, \mathbf{w}) = \begin{pmatrix} \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{y})} \\ 1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{y})} \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} h(\mathbf{w}^\top \mathbf{y}) \\ 1 - h(\mathbf{w}^\top \mathbf{y}) \end{pmatrix}$$

Assume we have the observation $\mathbf{c}_{\text{obs}} = \begin{pmatrix} \mathbf{c}_{\text{obs}} \\ 1 - \mathbf{c}_{\text{obs}} \end{pmatrix}$ then

$$\begin{aligned} E(\mathbf{c}_{\text{obs}}, \mathbf{c}) &= -\mathbf{c}_{\text{obs}}^\top \log(\mathbf{c}(\mathbf{y}, \mathbf{w})) \\ &= -\mathbf{c}_{\text{obs}} \log(h(\mathbf{w}^\top \mathbf{y})) - (1 - \mathbf{c}_{\text{obs}}) \log(1 - h(\mathbf{w}^\top \mathbf{y})). \end{aligned}$$

where

$$h(z) = \frac{1}{1 + \exp(-z)}$$

Cross Entropy for Logistic Regression - 2

In the case we have many examples need to sum over the data

$$\mathbf{C}(\mathbf{Y}, \mathbf{w}) = \begin{pmatrix} \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{Y})} \\ 1 - \frac{1}{1+\exp(\mathbf{w}^\top \mathbf{Y})} \end{pmatrix} \stackrel{def}{=} \begin{pmatrix} h(\mathbf{w}^\top \mathbf{Y}) \\ 1 - h(\mathbf{w}^\top \mathbf{Y}) \end{pmatrix} \in \mathbb{R}^{2 \times n}$$

Assume we have the observation $\mathbf{c}_{\text{obs}} \in \mathbb{R}^n$. Define

$$\mathbf{C}_{\text{obs}} = \begin{pmatrix} \mathbf{c}_{\text{obs}}^\top \\ 1 - \mathbf{c}_{\text{obs}}^\top \end{pmatrix} \in \mathbb{R}^{2 \times n}.$$

Then the cross entropy is

$$\begin{aligned} E(\mathbf{C}_{\text{obs}}, \mathbf{C}) &= -\frac{1}{n} \text{tr}(\mathbf{C}_{\text{obs}}^\top \mathbf{C}) \\ &= -\frac{1}{n} \mathbf{c}_{\text{obs}}^\top \log(h(\mathbf{w}^\top \mathbf{Y}) \\ &\quad -\frac{1}{n} (1 - \mathbf{c}_{\text{obs}})^\top \log(1 - h(\mathbf{w}^\top \mathbf{Y}))) \end{aligned}$$

Cross Entropy for Multinomial Logistic Regression

Similarly, for general case ($n_c \geq 2$ classes, n examples).
Recall:

$$\mathbf{C}(\mathbf{Y}, \mathbf{W}) = \exp(\mathbf{W}\mathbf{Y}) \operatorname{diag}\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{W}\mathbf{Y})}\right)$$

Get cross entropy by summing over all examples

$$E(\mathbf{C}_{\text{obs}}, \mathbf{C}(\mathbf{Y}, \mathbf{W})) = -\frac{1}{n}\operatorname{tr}(\mathbf{C}_{\text{obs}}^\top \log(\mathbf{C}(\mathbf{Y}, \mathbf{W}))).$$

We will also call this the *softmax* (cross-entropy) function.

Simplifying the Softmax Function

Let $\mathbf{S} = \mathbf{WY}$, then

$$E(\mathbf{C}_{\text{obs}}, \mathbf{S}) = -\frac{1}{n}\text{tr}\left(\mathbf{C}_{\text{obs}}^\top \log\left(\exp(\mathbf{S})\text{diag}\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})}\right)\right)\right).$$

Verify that this is equal to

$$\begin{aligned} E(\mathbf{C}_{\text{obs}}, \mathbf{S}) = &-\frac{1}{n}\mathbf{e}_{n_c}^\top \left(\mathbf{C}_{\text{obs}} \odot \mathbf{S}\right)\mathbf{e}_n \\ &+ \frac{1}{n}\mathbf{e}_{n_c}^\top \mathbf{C}_{\text{obs}} \log\left(\mathbf{e}_{n_c}^\top \exp(\mathbf{S})\right)^\top \end{aligned}$$

($\odot$ is Hadamard product, exp and log component-wise)

If $\mathbf{C}_{\text{obs}}$ has a unit row sum (why?) then $\mathbf{e}_{n_c}^\top \mathbf{C}_{\text{obs}}^\top = \mathbf{e}_n^\top$ and

$$E(\mathbf{C}_{\text{obs}}, \mathbf{S}) = -\frac{1}{n}\mathbf{e}_{n_c}^\top \left(\mathbf{C}_{\text{obs}} \odot \mathbf{S}\right)\mathbf{e}_n + \frac{1}{n}\log(\mathbf{e}_{n_c}^\top \exp(\mathbf{S}))\mathbf{e}_n$$

Numerical Considerations

Scale to prevent overflow. Note that for an arbitrary $s \in \mathbb{R}$ we have

$$E(\mathbf{C}_{\text{obs}}, \mathbf{WY} - s) = E(\mathbf{C}_{\text{obs}}, \mathbf{WY})$$

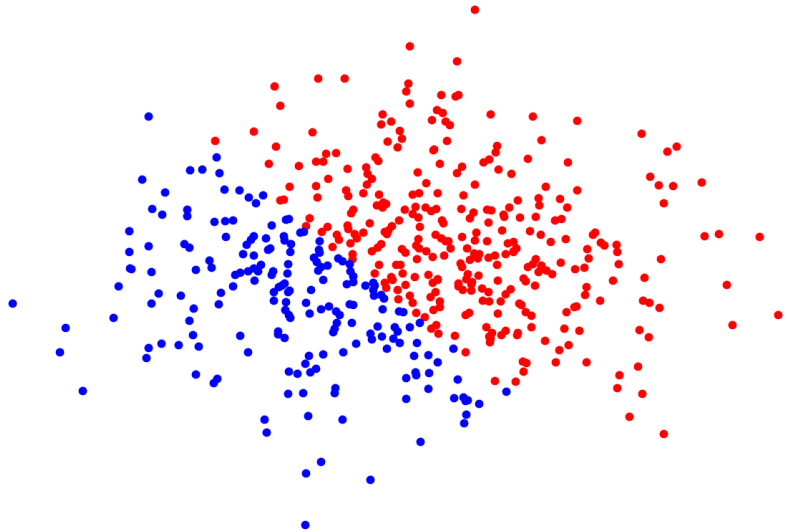
This prevents overflow, but may lead to underflow (and divisions by zero).

Note that s does not need to be the same in each row (example). Hence, we can choose $\mathbf{s} = \max(\mathbf{WY}, [], 1) \in \mathbb{R}^{1 \times n}$ to avoid underflow and overflow.

For stability use $E(\mathbf{C}_{\text{obs}}, \mathbf{S})$ where $\mathbf{S} = \mathbf{WY} - \mathbf{e}_{n_c}\mathbf{s}$.

Test Problem: Linear Classification

Generate data that is linearly separable:



```
a = 3; b = 2;

Y = randn(2,500);
C = a*Y(1,:) + b*Y(2,:) + 1;
C(C>0) = 1; C(C<0) = 0;
C = [C; 1-C]
```

Coding: Softmax Regression Objective Function

Write a function that computes the softmax function given a data matrix $\mathbf{Y}$, its class $\mathbf{C}$, and a matrix $\mathbf{W}$.

```
function[E] = softmaxFun(W,Y,C)
% Your code here
end
```

Here are some tests your code should master

- result should always be scalar
- result should not depend on the ordering of the columns of $\mathbf{Y}$
- for a given $(\mathbf{y}, \mathbf{c})$ the output of $E(\mathbf{y}, \mathbf{c})$ should be the same as $E(\mathbf{y}\mathbf{e}^\top, \mathbf{c}\mathbf{e}^\top)$.
- $\mathbf{W} = \alpha\mathbf{I}$ and $\mathbf{Y} = \mathbf{C}$ not cause overflow for any α

Linear Classification

If $\mathbf{W}$ can separate the classes then the goal is to minimize the cross entropy (with some potential regularization)

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \quad -\frac{1}{n}\mathbf{e}_{n_c}^\top \left(\mathbf{C}_{\text{obs}} \odot \mathbf{S}\right)\mathbf{e}_n + \frac{1}{n}\log(\mathbf{e}_{n_c}^\top \exp(\mathbf{S}))\mathbf{e}_n$$
$$\text{subject to } \mathbf{S} = \mathbf{W}\mathbf{Y} - \mathbf{e}_{n_c}\mathbf{s}$$

This is a smooth convex optimization problem
$\Rightarrow$ many existing optimization techniques will work

For large-scale problems, use derivative-based optimization algorithm. (Examples: Steepest Descent, Newton-like methods, Stochastic Gradient Descent, ADMM, . . .)

References: [4, 2, 1, 3]

Differentiating the Softmax Function - 1

We need to compute the derivative of the cross entropy function with respect to $\mathbf{W}$. Three hints:

- $\sum \mathbf{w} \odot \mathbf{y} = \mathbf{w}^\top \mathbf{y}$
- $\nabla_{\mathbf{w}}(\mathbf{w}^\top \mathbf{y}) = \mathbf{y}$
- $\mathrm{vec}(\mathbf{WY}) = (\mathbf{Y}^\top \otimes \mathbf{I})\mathrm{vec}(\mathbf{W}) = (\mathbf{I} \otimes \mathbf{W})\mathrm{vec}(\mathbf{Y})$

where $\otimes$ is the Kronecker product.

We use the common notation:

- $\mathrm{vec}(\mathbf{A})$ reshapes matrix $\mathbf{A}$ (column-wise) into vector
- $\mathrm{mat}(\mathbf{a})$ reshapes vector $\mathbf{a}$ into matrix, $\mathrm{mat}(\mathrm{vec}(\mathbf{A})) = \mathbf{A}$.
- $\mathrm{diag}(\mathbf{A}) = \mathrm{diag}(\mathrm{vec}(\mathbf{A}))$ builds diagonal matrix with entries given by $\mathbf{A}$.

Differentiating the Softmax Function - 2

Do it in three steps:

1. $\nabla_{\mathbf{S}} E(\mathbf{S})$ with $\mathbf{S} = \mathbf{YW}$ (assume w.l.o.g. no shift)
2. $\nabla_{\mathbf{W}} \mathbf{S} = \nabla_{\mathbf{W}}(\mathbf{WY})$
3. use chain rule to get $\nabla_{\mathbf{W}} E(\mathbf{WY})$.

Break the first step down into two terms

$$E(\mathbf{S}) = \frac{1}{n} \overbrace{-\mathrm{tr}(\mathbf{C}_{\mathrm{obs}}^{\top} \mathbf{S})}^{E1} + \frac{1}{n} \overbrace{\log(\mathbf{e}_{n_c}^{\top} \exp(\mathbf{S}))\mathbf{e}_n}^{E2},$$

First term is linear

$$\nabla_{\mathbf{S}} E_1 = \nabla_{\mathbf{S}} \mathrm{tr}(\mathbf{C}_{\mathrm{obs}}^{\top} \mathbf{S}) = \mathbf{C}_{\mathrm{obs}}.$$

Differentiating the Softmax Function - 3

$$E(\mathbf{S}) = \frac{1}{n}\overbrace{-\mathrm{tr}(\mathbf{C}_{\mathrm{obs}}^{\top}\mathbf{S})}^{E1} + \frac{1}{n}\overbrace{\log(\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S}))\mathbf{e}_n}^{E2}$$

Second term requires a bit more care

$$\mathbf{J}_{\mathbf{S}}E_2 = \mathbf{J}_{\mathbf{S}}\mathbf{e}_n^{\top}\log(\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S})) = \mathbf{e}_n^{\top}\mathbf{J}_{\mathbf{S}}\log(\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S}))$$

and

$$\mathbf{J}_{\mathbf{S}}\log(\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S})) = \mathrm{diag}\left(\frac{1}{\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S})}\right)\mathbf{J}_{\mathbf{S}}\left(\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S})\right)$$

Differentiating the Softmax Function - 4

Recall:

$$\mathbf{J}_{\mathbf{S}} E_2 = \mathbf{e}_n^\top \mathrm{diag}\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})}\right) \mathbf{J}_{\mathbf{S}} \left(\mathbf{e}_{n_c}^\top \exp(\mathbf{S})\right)$$

Focus on last term:

$$\begin{aligned}\mathbf{J}_{\mathbf{S}}(\mathbf{e}_{n_c}^\top \exp(\mathbf{S})) &= \mathbf{J}_{\mathbf{S}} \left(\left(\mathbf{I} \otimes \mathbf{e}_{n_c}^\top\right) \mathrm{vec}(\exp(\mathbf{S}))\right) \\ &= (\mathbf{I} \otimes \mathbf{e}_{n_c}^\top)\mathrm{diag}(\mathrm{vec}(\exp(\mathbf{S})))\end{aligned}$$

Putting it together

$$\mathbf{J}_{\mathbf{S}} E_2 = \mathbf{e}_n^\top \ \mathrm{diag}\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})}\right) \ (\mathbf{I} \otimes \mathbf{e}_{n_c}^\top)\mathrm{diag}(\mathrm{vec}(\exp(\mathbf{S})))$$

Left to do: Take transpose

Differentiating the Softmax Function - 5

$$\nabla_{\mathbf{S}} E_2 = \mathrm{diag}(\mathrm{vec}(\exp(\mathbf{S})))(\mathbf{I} \otimes \mathbf{e}_{n_c})\mathrm{diag}\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})}\right)\mathbf{e}_n$$

simplifying

$$\nabla_{\mathbf{S}} E_2 = \mathrm{diag}(\mathrm{vec}(\exp(\mathbf{S})))(\mathbf{I} \otimes \mathbf{e}_{n_c})\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})}\right)$$

Avoid diag and simplify using matrix representation

$$\nabla_{\mathbf{S}} E_2 = \exp(\mathbf{S}) \odot \left(\mathbf{e}_{n_c}\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})}\right)\right)$$

Finally combine gradients of E_1 and E_2

$$\nabla_{\mathbf{S}} E = -\frac{1}{n}\mathbf{C}_{\mathrm{obs}} + \frac{1}{n}\exp(\mathbf{S}) \odot \left(\mathbf{e}_{n_c}\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})}\right)\right).$$

Differentiating the Softmax Function - 6

$$E(\mathbf{W}) = -\frac{1}{n}\overbrace{\mathrm{tr}(\mathbf{C}_{\mathrm{obs}}^{\top}(\mathbf{WY}))}^{E1} + \frac{1}{n}\overbrace{\mathbf{e}_n^{\top}\log(\mathbf{e}_{n_c}^{\top}\exp(\mathbf{WY}))}^{E2}$$

Note that

$$\nabla_{\mathbf{W}}\mathbf{S} = \nabla_{\mathbf{W}}(\mathbf{WY}) = \nabla_{\mathbf{W}}\left((\mathbf{Y}^{\top}\otimes\mathbf{I})\mathrm{vec}(\mathbf{W})\right) = \mathbf{Y}\otimes\mathbf{I}.$$

Hence, applying the chain rule gives

$$\nabla_{\mathbf{W}}E = \frac{1}{n}\left(-\mathbf{C}_{\mathrm{obs}} + \exp(\mathbf{S})\odot\left(\mathbf{e}_{n_c}\left(\frac{1}{\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S})}\right)\right)\right)\mathbf{Y}^{\top}.$$

Coding: Differentiating the Softmax Function

Extend your softmax function, so that it returns the gradient if needed.

```
function[E,dE] = softmaxFun(W,Y,C)

% Your code from before

if nargout > 1

% Your code for gradient here
end

end
```

Testing your Derivatives

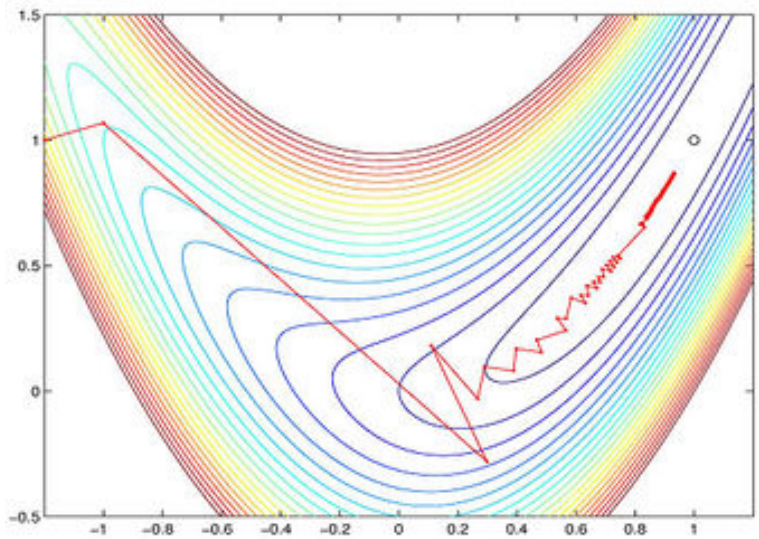
Your derivatives are assumed to be wrong unless you prove otherwise.
Test based on Taylor theorem. Let $\mathbf{W}$ be fixed and $\mathbf{D}$ be a random direction (same size as $\mathbf{W}$):

h	$E(\mathbf{W}+h\mathbf{D})-E(\mathbf{W})$	$E(\mathbf{W}+h\mathbf{D})-E(\mathbf{W})-h\mathrm{tr}(\mathbf{D}^\top\nabla E(\mathbf{W}))$
1		
2^{-1}		
2^{-2}		
2^{-3}		
2^{-4}		
2^{-5}		

First column should decay as $\mathcal{O}(h)$
Second column should decay as $\mathcal{O}(h^2)$

Derivative-Based Optimization: Steepest Descent

To minimize the energy go "down-hill"



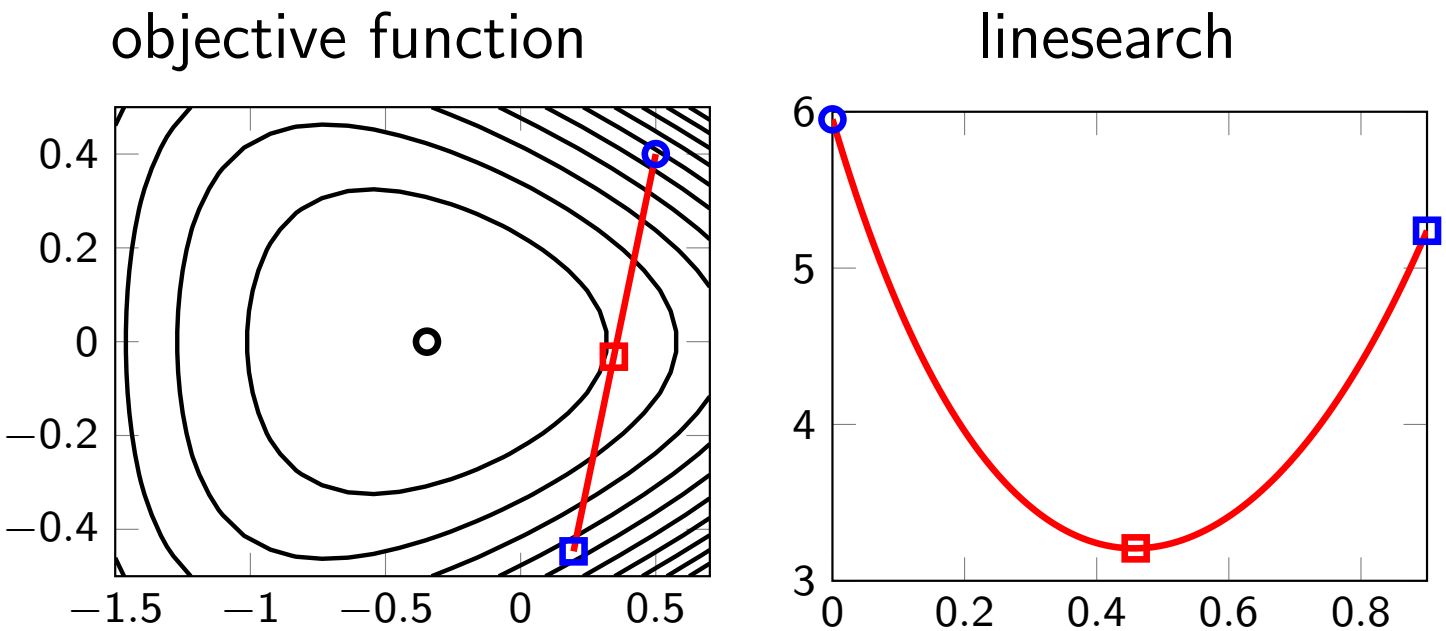
Iterate:

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha \mathbf{D}, \quad \mathbf{D} = -\nabla E(\mathbf{W}_k).$$

Guaranteed to be a descent direction but need to make sure that

$$E(\mathbf{W}_k + \alpha \mathbf{D}) < E(\mathbf{W}_k)$$

Line Search Problem



Let E be the cross entropy, $\mathbf{W}_k$ the current weights, and $\mathbf{D}$ the search direction. The line search problem is:

$$\min_{\alpha>0} \phi(\alpha) \quad \text{where} \quad \phi(\alpha) = E(\mathbf{W}_k + \alpha\mathbf{D}).$$

Armijo Line Search

A method for inexact line search

- Start with $\alpha = \alpha_0$
- Test $E(\mathbf{W} + \alpha\mathbf{D}) < E(\mathbf{W})$
- If fail $\alpha \leftarrow \frac{1}{2}\alpha$

A few (small) but helpful tricks

- Choose your α_0 based on the problem
- If line search is needed ($\alpha \neq \alpha_0$) at iteration k set $\alpha_0 = \alpha_k$ in next iteration.
- If no line search is needed ($\alpha = \alpha_0$) at iteration k set $\alpha_0 = \gamma\alpha_k$, $\gamma > 1$ in next iteration.

Coding: Steepest Descent

Write a code for steepest descent with Armijo linesearch.

```
function W = steepestDescent(E,W,param)
% Inputs:
%     E  - function that provides value and gradient
%     W  - starting guess
%  param - struct with parameters

alpha     = param.maxStep; % max step size
maxIter   = param.maxIter; % max number of iterations

for i=1:maxIter

% Your code here
end
```

Newton-like Methods

Goal: Solve $\min_{\mathbf{W}} E(\mathbf{W})$. Consider kth iteration. Assume E convex.

To find optimal step $\mathbf{D}$, use Taylor's theorem

$$E(\mathbf{W}_k+\mathbf{D}) = E(\mathbf{W}_k)+\nabla E(\mathbf{W}_k)^\top \mathbf{D}+\frac{1}{2}\mathbf{D}^\top \nabla^2 E(\mathbf{W}_k)\mathbf{D}+\mathcal{O}(\|\mathbf{D}\|^3)$$

and differentiate w.r.t $\mathbf{D}$ to obtain

$$\nabla^2 E(\mathbf{W}_k)\mathbf{D} = -\nabla E(\mathbf{W}_k).$$

Practical Newton methods (see, e.g., [4, Ch.7])

- do not compute $\mathbf{D}$ accurately (add line search for safety)
- use, e.g., Conjugate Gradient (CG) methods
- do not generate $\nabla^2 E$ since CG only needs mat-vecs
- give quadratic/superlinear/good linear convergence

Newton-like Methods for Softmax

Need to compute Hessian $\nabla^2 E$. Recall:

$$\begin{aligned}\nabla_{\mathbf{W}} E &= \frac{1}{n}\left(-\mathbf{C}_{\text{obs}} + \exp(\mathbf{S}) \odot \left(\mathbf{e}_{n_c}\left(\frac{1}{\mathbf{e}_{n_c}^\top \exp(\mathbf{S})}\right)\right)\right)\mathbf{Y}^\top \\ &= \nabla_{\mathbf{S}} E(\mathbf{S})\mathbf{Y}^\top,\end{aligned}$$

where $\mathbf{S} = \mathbf{WY}$. For $\mathbf{w} = \text{vec}(\mathbf{W})$ and $\mathbf{s} = \text{vec}(\mathbf{S})$ we know that

$$\nabla^2_{\mathbf{w}} E(\mathbf{w}) = (\mathbf{Y} \otimes \mathbf{I}_{n_c})\, \nabla^2_{\mathbf{s}} E(\mathbf{s})\, (\mathbf{Y}^\top \otimes \mathbf{I}_{n_c})$$

Remarks:

- size of $\nabla^2_{\mathbf{s}} E$ is $n_c n \times n_c n$, typically sparse
- size of $\nabla^2_{\mathbf{w}} E$ is $n_c n_f \times n_c n_f$, typically dense
- building Hessian can be costly (when n is large)
- Hessian is spd since E is convex in $\mathbf{S}$

Hessian of Softmax Function - 1

Recall

$$\nabla_{\mathbf{S}} E = \frac{1}{n}\left(-\mathbf{C}_{\text{obs}} + \exp(\mathbf{S}) \odot \frac{1}{\mathbf{e}_{n_c}\mathbf{e}_{n_c}^{\top}\exp(\mathbf{S})}\right)$$

Let's first vectorize this $\mathbf{s} = \text{vec}(\mathbf{S})$ and $\mathbf{c} = \text{vec}(\mathbf{C}_{\text{obs}})$

$$\nabla_{\mathbf{s}} E = \frac{1}{n}\left(-\mathbf{c} + \exp(\mathbf{s}) \odot \frac{1}{(\mathbf{I} \otimes (\mathbf{e}_{n_c}\mathbf{e}_{n_c}^{\top}))\exp(\mathbf{s})}\right)$$

Use product rule

$$\begin{aligned}
\nabla_{\mathbf{s}}^2 E &= \frac{1}{n}\text{diag}\left(\frac{1}{(\mathbf{I} \otimes (\mathbf{e}_{n_c}\mathbf{e}_{n_c}^{\top}))\exp(\mathbf{s})}\right)\mathbf{J}_{\mathbf{s}}\exp(\mathbf{s}) + \\
&\quad \frac{1}{n}\text{diag}(\exp(\mathbf{s}))\mathbf{J}_{\mathbf{s}}\left(\frac{1}{(\mathbf{I} \otimes (\mathbf{e}_{n_c}\mathbf{e}_{n_c}^{\top}))\exp(\mathbf{s})}\right) \\
&= \nabla_{\mathbf{s}}^2 E_1 + \nabla_{\mathbf{s}}^2 E_2
\end{aligned}$$

Hessian of Softmax Function - 2

First term is easy

$$\begin{aligned}\nabla_{\mathbf{s}}^2 E_1(\mathbf{s}) &= \frac{1}{n}\mathrm{diag}\left(\frac{1}{(\mathbf{I}\otimes(\mathbf{e}_{n_c}\mathbf{e}_{n_c}^\top))\exp(\mathbf{s})}\right)\mathrm{diag}\left(\exp(\mathbf{s})\right)\\ &= \frac{1}{n}\mathrm{diag}\left(\frac{\exp(\mathbf{s})}{(\mathbf{I}\otimes(\mathbf{e}_{n_c}\mathbf{e}_{n_c}^\top))\exp(\mathbf{s})}\right)\end{aligned}$$

Reshaped back, a matrix-vector-product with $\mathbf{V}\in\mathbb{R}^{n_c\times n_f}$ is

$$\nabla_{\mathbf{S}}^2 E_1(\mathbf{S})\mathbf{V} = \frac{1}{n}\left(\left(\frac{\exp(\mathbf{S})}{\mathbf{e}_{n_c}\mathbf{e}_{n_c}^\top\exp(\mathbf{S})}\right)\odot(\mathbf{V}\mathbf{Y})\right)\mathbf{Y}^\top$$

Hessian of Softmax Function - 3

$$E_2 = \frac{1}{n}\mathrm{diag}(\exp(\mathbf{s}))\,\underbrace{\mathbf{J}_{\mathbf{s}}\left(\frac{1}{(\mathbf{I}\otimes(\mathbf{e}_{n_c}\mathbf{e}_{n_c}^\top))\exp(\mathbf{s})}\right)}_{=:\mathbf{T}}.$$

Using chain rule, we get

$$\mathbf{T} = -\mathrm{diag}\left(\frac{1}{\left((\mathbf{I}\otimes(\mathbf{e}_{n_c}\mathbf{e}_{n_c}^\top))\exp(\mathbf{s})\right)^2}\right)(\mathbf{I}\otimes(\mathbf{e}_{n_c}\mathbf{e}_{n_c}^\top))\mathrm{diag}(\exp(\mathbf{s}))$$

After reshape the matrix-vector-product with $\mathbf{V}\in\mathbb{R}^{n_f\times n_c}$ is

$$\nabla^2_{\mathbf{S}}E_2(\mathbf{S})\mathbf{V} = -\frac{1}{n}\left(\frac{(\exp(\mathbf{S}))}{\mathbf{e}_{n_c}(\mathbf{e}_{n_c}^\top\exp(\mathbf{S}))^2}\right)\odot(\mathbf{e}_{n_c}\mathbf{e}_{n_c}^\top(\exp(\mathbf{S})\odot(\mathbf{VY}$$

Newton-CG for Softmax function

Mat-vecs with Hessian can be computed as

$$\begin{aligned}\nabla^2_{\mathbf{W}} E(\mathbf{W})\mathbf{V} &= \frac{1}{n}\left(\left(\frac{\exp(\mathbf{S})}{\mathbf{e}_{n_c}\mathbf{e}_{n_c}^\top \exp(\mathbf{S})}\right) \odot (\mathbf{V}\mathbf{Y})\right)\mathbf{Y}^\top \\ &- \frac{1}{n}\left(\frac{(\exp(\mathbf{S}))}{\mathbf{e}_{n_c}(\mathbf{e}_{n_c}^\top \exp(\mathbf{S}))^2}\right) \odot (\mathbf{e}_{n_c}\mathbf{e}_{n_c}^\top(\exp(\mathbf{S}) \odot (\mathbf{V}\mathbf{Y})))\mathbf{Y}^\top\end{aligned}$$

(possible to further simplify this to reduce operations)

Now, ready to use matrix-free Newton method with Armijo linesearch and CG solver that computes

$$\nabla^2_{\mathbf{W}} E(\mathbf{W})\mathbf{D} \approx -\nabla_{\mathbf{W}} E(\mathbf{W}).$$

Remarks:

- how well to solve? use large tolerance on relative residual
- can accelerate CG with preconditioning $\rightsquigarrow$ PCG
- possible to omit second term in Hessian?

Coding: Hessian of Softmax Function

Extend your softmax function, so that it returns a function handle computing mat-vecs with Hessian if needed.

```
function[E,dE,d2Emv] = softmaxFun(W,Y,C)

% Your code from before
if nargout > 1
% Your code from before
end

if nargout > 2

% Your new code here
d2Emv = @(V) ...;
end
end
```

Don't forget to check your derivatives!

Project: Linear Classification for MNIST

Write a code for solving

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \quad -\frac{1}{n}\mathbf{e}_{n_c}^\top \left(\mathbf{C}_{\text{obs}} \odot \mathbf{S}\right)\mathbf{e}_n + \frac{1}{n}\log(\mathbf{e}_{n_c}^\top \exp(\mathbf{S}))\mathbf{e}_n$$
$$\text{subject to } \mathbf{S} = \mathbf{W}\mathbf{Y} - \mathbf{e}_{n_c}\mathbf{s}$$

and apply it to the MNIST test data set.

1. write a stable code for evaluating the softmax loss
2. implement and test the derivatives we computed in class or use automatic differentiation to compute gradients and Hessians.
3. use optimal weights to predict labels for test data. How well does your solution generalize?
4. visualize the rows of $\mathbf{W}$ as images.
5. compare with the results you got in the last module.

Σ : Linear Classifiers

- classification: use (multinomial) logistic regression
- compare to least-squares
  - advantage: output is in unit simplex
  - disadvantage: problem does not decouple into n_c subproblems, statistical interpretation?
- loss function: cross-entropy preferred (convex)
- can choose between many numerical optimizers: SD, Newton, ℓBFGS, ADMM,...
- useful in supervised and semi-supervised tasks
- work well iff classes can be separated by hyperplanes

References

[1] A. Beck. *Introduction to Nonlinear Optimization*. Theory, Algorithms, and Applications with MATLAB. SIAM, Philadelphia, Oct. 2014.

[2] S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Mar. 2004.

[3] S. W. Fung, S. Tyrväinen, L. Ruthotto, and E. Haber. Large-Scale Classification using Multinomial Regression and ADMM. Jan. 2019.

[4] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer Science & Business Media, New York, Dec. 2006.