Classification

Numerical Methods for Deep Learning

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 $f(\mathbf{y}, \theta)$ is supposed to be probability, use logistic sigmoid

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

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

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

from now on consider linear models for simplicity

Multinomial Logistic Regression

Suppose data falls into $n_c \geq 2$ categories and the components of $\mathbf{c}_{\text{obs}}(\mathbf{y}) \in [0,1]^{n_c}$ contain probabilities for each class.

Applying the logistic sigmoid to each component of $f(\mathbf{y}, \mathbf{W})$ not enough (probabilities must sum to one). Use

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

where $\mathbf{e} = (1, 1, \ldots, 1)^\top \in \mathbb{R}^{n_c}$.

Note: Division and exp are done element-wise!

Logistic Regression - Loss Function

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

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

$$\frac{1}{2n} \|\mathbf{c}(\mathbf{Y}, \mathbf{W}) - \mathbf{c}_{\mathrm{obs}}\|_F^2$$

Problems

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

Need to be careful to treat $\mathbf{c}$ appropriately.

Example: Designing a Code

Goal: Communicate using minimal number of bits.

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?

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
dog	00
cat	01
fish	10
bird	11

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

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?

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 common we use it, the shorter the word should be.

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 common 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!

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

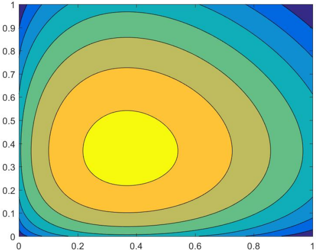


Figure: The entropy of a vector $\mathbf{c} = [c_1, c_2]$

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

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

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

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

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.

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

$$\begin{aligned}\mathbf{c}(\mathbf{y}, \mathbf{w}) &= \left[\frac{1}{1+\exp(\mathbf{y}^\top \mathbf{w})}, 1 - \frac{1}{1+\exp(\mathbf{y}^\top \mathbf{w})}\right] \\ &= [h(\mathbf{y}^\top \mathbf{w}), 1 - h(\mathbf{y}^\top \mathbf{w})]\end{aligned}$$

Assume that we have the observation $\mathbf{C}_{\mathrm{obs}} = [\mathbf{c}_{\mathrm{obs}}, 1 - \mathbf{c}_{\mathrm{obs}}]$ then

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

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

$$\mathbf{C}(\mathbf{Y}, \mathbf{w}) = \left[\frac{1}{1 + \exp(\mathbf{Y}\mathbf{w})}, 1 - \frac{1}{1 + \exp(\mathbf{Y}\mathbf{w})}\right]$$
$$= [h(\mathbf{Y}\mathbf{w}), 1 - h(\mathbf{Y}\mathbf{w})]$$

A little abuse of notation

$$\mathbf{C}_{\mathrm{obs}} = [\mathbf{c}_{\mathrm{obs}}, 1 - \mathbf{c}_{\mathrm{obs}}] \in \mathbb{R}^{n \times 2}$$

$$E(\mathbf{c}_{\mathrm{obs}}, \mathbf{c}(\mathbf{Y}, \mathbf{w})) = \mathbf{c}_{\mathrm{obs}}^{\top} \log(h(\mathbf{Y}\mathbf{w})) + (1 - \mathbf{c}_{\mathrm{obs}})^{\top} \log(1 - h(\mathbf{Y}\mathbf{w}))).$$

Cross Entropy for Multinomial Logistic Regression

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

$$\mathbf{C}(\mathbf{Y}, \mathbf{W}) = \mathrm{diag}\left(\frac{1}{\exp(\mathbf{YW})\mathbf{e}}\right) \exp(\mathbf{YW})$$

Get cross entropy by summing over all examples

$$E(\mathbf{C}_{\mathrm{obs}}, \mathbf{C}(\mathbf{Y}, \mathbf{W})) = -\mathrm{trace}(\mathbf{C}_{\mathrm{obs}}^\top \log(\mathbf{C}(\mathbf{Y}, \mathbf{W}))).$$

This is also called *softmax* function.

Simplifying the Softmax Function

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

Verify that this is equal to

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

Here: $\odot$ is Hadamard product (component-wise)

Simplifying the Softmax Function

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

Verify that this is equal to

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

Here: $\odot$ is Hadamard product (component-wise)

If $\mathbf{C}_{\mathrm{obs}}$ as a unit row sum (why?) then

$$\mathbf{e}_{n_c}^\top \mathbf{C}_{\mathrm{obs}}^\top = \mathbf{e}_n^\top$$

and therefore

$$E(\mathbf{C}_{\mathrm{obs}}, \mathbf{YW}) = -\mathbf{e}_n^\top \left(\mathbf{C}_{\mathrm{obs}} \odot (\mathbf{YW})\right)\mathbf{e}_{n_c} + \mathbf{e}_n^\top \log(\exp(\mathbf{YW})\mathbf{e}_{n_c})$$

Numerical Considerations

Scale to prevent overflow. Note that for an arbitrary s we have

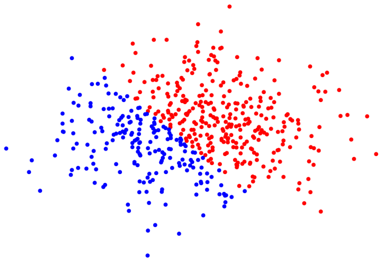
$$E(\mathbf{YW} - s, \mathbf{C}_{\mathrm{obs}}) = E(\mathbf{YW}, \mathbf{C}_{\mathrm{obs}})$$

Therefore we can choose $\mathbf{s} = \max(\mathbf{YW}, [], 2)$ to avoid overflow and potential scaling issues

Note $\mathbf{s}$ is a vector

Test Problem: Linear Classification

Generate data that is linearly separable:



```
a = 3; b = 2;

Y = randn(500,2);
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 **Y**, its class **C**, and a matrix **W**.

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

% Your code here


end
```

Linear Classification

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

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} -\mathbf{e}_n^\top \left(\mathbf{C}_{\rm obs} \odot (\mathbf{YW})\right) \mathbf{e}_{n_c} + \mathbf{e}_n^\top \log(\exp(\mathbf{YW})\mathbf{e}_{n_c})$$

Linear Classification

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

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} -\mathbf{e}_n^\top \left(\mathbf{C}_{\text{obs}} \odot (\mathbf{YW})\right) \mathbf{e}_{n_c} + \mathbf{e}_n^\top \log(\exp(\mathbf{YW})\mathbf{e}_{n_c})$$

This is a convex optimization problem $\rightsquigarrow$ use standard optimization techniques

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

Differentiating the Softmax Regression Function

We need to compute the derivative of the softmax function with respect to $\mathbf{W}$ and b.
Three hints

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

To differentiate we will use the chain rule and test our results at each step.

Differentiating the Softmax Function

Do it in two stages (chain rule)

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

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

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

Second term requires a bit more care

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

Differentiating the Softmax Function

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

Second term requires a bit more care

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

and

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

Differentiating the Softmax Function

$$\mathbf{J_S} \exp(\mathbf{S})\mathbf{e}_{n_c} = \mathbf{J_S}(\mathbf{e}_{n_c}^\top \otimes \mathbf{I})\mathrm{vec}(\exp(\mathbf{S})) = (\mathbf{e}_{n_c}^\top \otimes \mathbf{I})\mathrm{diag}(\mathrm{vec}(\exp(\mathbf{S})))$$

Differentiating the Softmax Function

$$\mathbf{J}_{\mathbf{S}} \exp(\mathbf{S})\mathbf{e}_{n_c} = \mathbf{J}_{\mathbf{S}}(\mathbf{e}_{n_c}^\top \otimes \mathbf{I})\mathrm{vec}(\exp(\mathbf{S})) = (\mathbf{e}_{n_c}^\top \otimes \mathbf{I})\mathrm{diag}(\mathrm{vec}(\exp(\mathbf{S})))$$

Putting it together

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

Differentiating the Softmax Function

$$\mathbf{J}_{\mathbf{S}} \exp(\mathbf{S})\mathbf{e}_{n_c} = \mathbf{J}_{\mathbf{S}}(\mathbf{e}_{n_c}^\top \otimes \mathbf{I})\mathrm{vec}(\exp(\mathbf{S})) = (\mathbf{e}_{n_c}^\top \otimes \mathbf{I})\mathrm{diag}(\mathrm{vec}(\exp(\mathbf{S})))$$

Putting it together

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

Remember to take the transpose ...

Differentiating the Softmax Function

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

Differentiating the Softmax Function

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

simplifying

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

Differentiating the Softmax Function

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

simplifying

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

And even more ... matrix representation

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

Differentiating the Softmax Function

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

simplifying

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

And even more ... matrix representation

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

Finally (almost)

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

Differentiating the Softmax Function

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

Finally (really!)

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

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

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

Testing your Derivatives

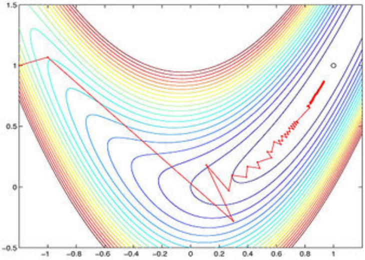
Your derivatives are assumed to be wrong unless you prove otherwise.
Test based on Taylor theorem

h	$E(\mathbf{W}+h\mathbf{S})-E(\mathbf{W})$	$E(\mathbf{W}+h\mathbf{S})-E(\mathbf{W})-h\,\mathrm{tr}(\mathbf{S}^\top\mathbf{W})$
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”



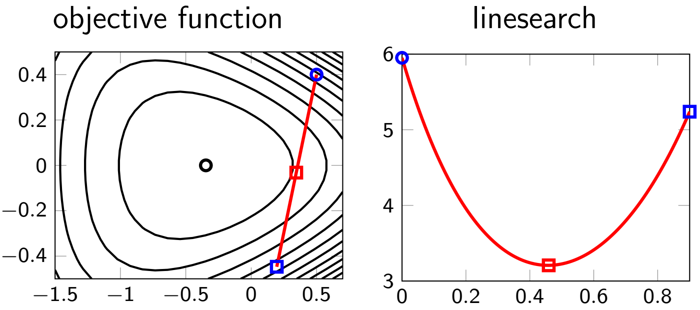
Iterate:

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

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

$$E(\mathbf{W}_k + \mu\mathbf{S}) < E(\mathbf{W}_k)$$

Line Search Problem



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

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

Armijo Line Search

A method for inexact line search

- Start with $\mu = \mu_0$
- Test $E(\mathbf{W} + \mu\mathbf{S}) < E(\mathbf{W})$
- If fail $\mu \leftarrow \frac{1}{2}\mu$

Armijo Line Search

A method for inexact line search

- Start with $\mu = \mu_0$
- Test $E(\mathbf{W} + \mu\mathbf{S}) < E(\mathbf{W})$
- If fail $\mu \leftarrow \frac{1}{2}\mu$

A few (small) but helpful tricks

- Choose your μ_0 based on the problem
- If line search is needed ($\mu \neq \mu_0$) at iteration k set $\mu_0 = \mu_k$ in next iteration.
- If no line search is needed ($\mu = \mu_0$) at iteration k set $\mu_0 = \gamma\mu_k$, $\gamma > 1$ in next iteration.

Coding: Steepest Descent

Write a code for steepest descent.

```
function W = steepestDescent(E,W,param)

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

for i=1:maxIter

% Your code here



end

end
```

Newton flavored methods

Use higher information about the function

$$E(\mathbf{W}+\mathbf{S}) = E(\mathbf{W}) + (\mathbf{S}, \nabla E) + \frac{1}{2}(\mathbf{S}, \nabla^2 E\,\mathbf{S}) + \mathrm{hot}$$

where $\nabla^2 E$ is the Hessian
Minimizing with respect to $\mathbf{S}$ obtain

$$\mathbf{S} = -(\nabla^2 E)^{-1}\nabla E.$$

Vanilla Newton

- Compute the Hessian
- Solve the linear system

Quadratic convergence

Newton flavored methods

Use higher information about the function

$$E(\mathbf{YW} + \mathbf{YS}) = E(\mathbf{YW}) + (\mathbf{YS}, \nabla E) + \frac{1}{2}(\mathbf{YS}, \nabla^2 E\, \mathbf{YS}) + \mathrm{hot}$$

where $\nabla^2 E$ is the Hessian with respect to $\mathbf{YW}$
Minimizing with respect to $\mathbf{S}$ obtain

$$\mathbf{S} = -(\mathbf{Y}^\top \nabla^2 E \mathbf{Y})^{-1} \nabla E.$$

Newton in practice

- Compute Hessian mat-vecs (or approximations)
- **Approximately** Solve the linear system

Quadratic/superlinear/good linear convergence

Newton for softMax function

The gradient

$$\nabla E = \left(-\mathbf{C} + \exp(\mathbf{S}) \odot \frac{1}{\exp(\mathbf{S})\mathbf{e}\mathbf{e}^\top} \right)$$

Vectorizing $\mathbf{s} = \mathrm{vec}(\mathbf{S})$

$$\nabla E = -\mathbf{C} + \exp(\mathbf{s}) \odot \frac{1}{(\mathbf{e}\mathbf{e}^\top \otimes \mathbf{I}) \exp(\mathbf{s})}$$

Use product rule

$$\begin{aligned} \nabla^2 E \quad &= \quad \mathrm{diag}\left(\frac{1}{(\mathbf{e}\mathbf{e}^\top \otimes \mathbf{I}) \exp(\mathbf{s})} \right) \nabla \left(\exp(\mathbf{s}) \right) + \\ &\quad\quad \mathrm{diag}(\exp \mathbf{s}) \nabla \left(\frac{1}{(\mathbf{e}\mathbf{e}^\top \otimes \mathbf{I}) \exp(\mathbf{s})} \right) \end{aligned}$$

Newton for softMax function

First term easy

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

Need only mat-vec

$$
\mathbf{H}\mathbf{V} \approx \mathbf{Y}^\top \left(\left(\frac{\exp(\mathbf{S})}{\exp(\mathbf{S})\mathbf{e}}\right) \odot (\mathbf{Y}\mathbf{V})\right)
$$

Newton for softMax function

Second term mat-vec

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

A little bit longer do derive

May not want to use the second term in Newton

Newton for softMax function

Use the mat-vec in Newton-CG algorithm

Newton for softMax function