Reminder: Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

\[
\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)
\]

- If the activation is:
  - Positive, output +1
  - Negative, output -1
How to get probabilistic decisions?

- **Activation:** \( z = w \cdot f(x) \)
- If \( z = w \cdot f(x) \) very positive: want probability going to 1
- If \( z = w \cdot f(x) \) very negative: want probability going to 0

- **Sigmoid function**

\[
\phi(z) = \frac{1}{1 + e^{-z}}
\]
Best w?

- Maximum likelihood estimation:

\[
\max_w \ ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)
\]

with:

\[
P(y^{(i)} = +1 | x^{(i)}; w) = \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}
\]

\[
P(y^{(i)} = -1 | x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}
\]

= Logistic Regression
Multiclass Logistic Regression

- Multi-class linear classification
  - A weight vector for each class: \( w_y \)
  - Score (activation) of a class \( y \): \( w_y \cdot f(x) \)
  - Prediction w/highest score wins: \( y = \text{arg max}_y w_y \cdot f(x) \)

- How to make the scores into probabilities?

\[
\begin{align*}
Z_1, Z_2, Z_3 & \rightarrow \frac{e^{Z_1}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}, \quad \frac{e^{Z_2}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}, \quad \frac{e^{Z_3}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}
\end{align*}
\]

original activations softmax activations
Best w?

- Maximum likelihood estimation:

$$\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

with:

$$P(y^{(i)}|x^{(i)}; w) = \frac{e^{w y^{(i)} \cdot f(x^{(i)})}}{\sum_y e^{w y \cdot f(x^{(i)})}}$$

= Multi-Class Logistic Regression
This Lecture

- Optimization
  - i.e., how do we solve:

\[
\max_w \ ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)
\]
Hill Climbing Diagram

- Objective function
- Global maximum
- Shoulder
- Local maximum
- "Flat" local maximum
- Current state
- State space
Required Mathematics Background

- **Linear algebra:**
  - Definition and properties of dot products
  - Composition of linear transformations is linear

- **Vector calculus:**
  - How to take partial derivatives (incl. chain rule, vector derivatives)
  - Solving optimization problems using derivatives (e.g., deriving MLE)
  - Taylor expansion (used in lecture; non-examinable)

- **Probability:** definition of a probability distribution, random variables, joint and marginal distributions, conditional probabilities, Bayes’ rule, normalization
Hill Climbing

- Recall from CSPs lecture: simple, general idea
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

- What’s particularly tricky when hill-climbing for multiclass logistic regression?
  - Optimization over a continuous space
    - Infinitely many neighbors!
    - How to do this efficiently?
1-D Optimization

- Could evaluate $g(w_0 + h)$ and $g(w_0 - h)$
  - Then step in best direction

- Or, evaluate derivative:

  $$\frac{\partial g(w_0)}{\partial w} = \lim_{h \to 0} \frac{g(w_0 + h) - g(w_0 - h)}{2h}$$
  - Tells which direction to step into
2-D Optimization

Source: offconvex.org
Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider: $g(w_1, w_2)$

  - Updates:
    
    $w_1 \leftarrow w_1 + \alpha \cdot \frac{\partial g}{\partial w_1}(w_1, w_2)$
    
    $w_2 \leftarrow w_2 + \alpha \cdot \frac{\partial g}{\partial w_2}(w_1, w_2)$

  - Updates in vector notation:
    
    $w \leftarrow w + \alpha \cdot \nabla_w g(w)$

    with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix} = \text{gradient}$
Gradient Ascent

- Idea:
  - Start somewhere
  - Repeat: Take a step in the gradient direction

Figure source: Mathworks
What is the Steepest Direction?

First-Order Taylor Expansion:
\[
g(w + \Delta) \approx g(w) + \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2
\]

Steepest Direction:
\[
\max_{\Delta: \Delta^2 + \Delta^2 \leq \varepsilon} g(w + \Delta)
\]

Recall:
\[
\max_{\Delta: \|\Delta\| \leq \varepsilon} \Delta^T a
\]

Hence, solution:
\[
\Delta = \varepsilon \frac{a}{\|a\|}
\]
\[
\nabla g = \begin{bmatrix}
\frac{\partial g}{\partial w_1} \\
\frac{\partial g}{\partial w_2}
\end{bmatrix}
\]

Gradient direction = steepest direction!
Gradient in n dimensions

\[ \nabla g = \begin{bmatrix}
\frac{\partial g}{\partial w_1} \\
\frac{\partial g}{\partial w_2} \\
\vdots \\
\frac{\partial g}{\partial w_n}
\end{bmatrix} \]
Optimization Procedure: Gradient Ascent

- init $w$
- for iter = 1, 2, ...

\[
w \leftarrow w + \alpha \cdot \nabla g(w)
\]

- $\alpha$: learning rate --- tweaking parameter that needs to be chosen carefully
- How? Try multiple choices
  - Crude rule of thumb: update changes $\mathcal{W}$ about 0.1 – 1%
Batch Gradient Ascent on the Log Likelihood Objective

\[
\max_w \ ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)
\]

\[g(w)\]

- \text{init } w)
- \text{for iter } = 1, 2, \ldots

\[
w \leftarrow w + \alpha \sum_i \nabla \log P(y^{(i)}|x^{(i)}; w)
\]
Stochastic Gradient Ascent on the Log Likelihood Objective

\[
\max_w \ ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)
\]

**Observation:** once gradient on one training example has been computed, might as well incorporate before computing next one

- \textbf{init} \( w \)
- \textbf{for} \ iter = 1, 2, ...
  - \textbf{pick} random \( j \)
    
    \[
    w \leftarrow w + \alpha \times \nabla \log P(y^{(j)}|x^{(j)}; w)
    \]
Mini-Batch Gradient Ascent on the Log Likelihood Objective

\[
\max_w \; ll(w) = \max_w \; \sum_i \log P(y^{(i)}|x^{(i)}; w)
\]

**Observation:** gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- **init** \( w \)
- **for** iter = 1, 2, ...
  - pick random subset of training examples \( J \)
  \[
  w \leftarrow w + \alpha \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)
  \]
How about computing all the derivatives?

- We’ll talk about that once we covered neural networks, which are a generalization of logistic regression
Mid-Semester Survey

- Add 2 points of extra credit to your midterm score!
Key ideas:
- Second-order optimization methods
- Momentum
- Adaptive learning rates

Example optimizers:
- Newton’s method
- Nesterov accelerated gradient
- Adagrad, Adam, RMSProp, etc.
Neural Networks
Multi-class Logistic Regression

- = special case of neural network

\[
P(y_1|x; w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}
\]

\[
P(y_2|x; w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}
\]

\[
P(y_3|x; w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}
\]
Deep Neural Network = Also learn the features!

\[ P(y_1|x; w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}} \]

\[ P(y_2|x; w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}} \]

\[ P(y_3|x; w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}} \]
Deep Neural Network = Also learn the features!

\[ z_i^{(k)} = g \left( \sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)} \right) \]
Deep Neural Network = Also learn the features!

\[
\begin{align*}
  z_i^{(k)} &= g \left( \sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)} \right)
\end{align*}
\]
Importance of Nonlinear Activation Functions

- What happens if we add more layers?
  
  \[ z_2 = W_2 (W_1 x + b_1) + b_2 \]
  
  \[ z_2 = W_2 (W_1 x + b_1) + b_2 = W_2 W_1 x + W_2 b_1 + b_2 = W_{new} x + b_{new} \]

- No gain to adding more linear layers!
- Idea: add nonlinearities to capture more complex relationships
Deep Neural Network = Also learn the features!

\[ x_1 \to z_1^{(1)} \to z_2^{(1)} \to z_3^{(1)} \to \ldots \to z_{K^{(1)}}^{(1)} \]
\[ x_2 \to z_1^{(2)} \to z_2^{(2)} \to z_3^{(2)} \to \ldots \to z_{K^{(2)}}^{(2)} \]
\[ x_3 \]
\[ \ldots \]
\[ x_L \to z_1^{(L)} \to z_2^{(L)} \to z_3^{(L)} \to \ldots \to z_{K^{(L)}}^{(L)} \]

\[ z_i^{(k)} = g \left( \sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)} \right) \]

\[ g = \text{nonlinear activation function} \]