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
Recap: How to get probabilistic decisions?

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

- **Sigmoid function**

\[
\phi(z) = \frac{1}{1 + e^{-z}}
\]
Recap: 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: \( y = \arg \max_y w_y \cdot f(x) \)

- How to make the scores into probabilities?

\[
\begin{align*}
Z_1, Z_2, Z_3 & \rightarrow \frac{e^{Z_1}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}, \quad \frac{e^{Z_2}}{e^{Z_1} + e^{Z_2} + e^{Z_3}}, \quad \frac{e^{Z_3}}{e^{Z_1} + e^{Z_2} + e^{Z_3}} \\
\text{original activations} & \quad \text{softmax activations}
\end{align*}
\]
Best $w$?

- Maximum likelihood estimation:

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

with:

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

= Multi-Class Logistic Regression
Optimization

i.e., how do we solve:

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

- 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?
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 $w$ about 0.1 – 1%
How about computing all the derivatives?

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

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

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

\[
\begin{align*}
\mathbf{z}_i^{(k)} &= g\left( \sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)} \right) \\
g &= \text{nonlinear activation function}
\end{align*}
\]
Common Activation Functions

Sigmoid Function

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

\[ g'(z) = g(z)(1 - g(z)) \]

Hyperbolic Tangent

\[ g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \]

\[ g'(z) = 1 - g(z)^2 \]

Rectified Linear Unit (ReLU)

\[ g(z) = \max(0, z) \]

\[ g'(z) = \begin{cases} 
1, & z > 0 \\
0, & \text{otherwise} 
\end{cases} \]
Training the deep neural network is just like logistic regression:

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

just \( w \) tends to be a much, much larger vector 😊

→ just run gradient ascent
+ stop when log likelihood of hold-out data starts to decrease
Neural Networks Properties

- **Theorem (Universal Function Approximators).** A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

- **Practical considerations**
  - Can be seen as learning the features
  - Large number of neurons
    - Danger for overfitting
    - (hence early stopping!)
Fun Neural Net Demo Site

- Demo-site:
  - http://playground.tensorflow.org/
How about computing all the derivatives?

Derivatives tables:

\[
\begin{align*}
\frac{d}{dx} (a) &= 0 \\
\frac{d}{dx} (x) &= 1 \\
\frac{d}{dx} (au) &= a \frac{du}{dx} \\
\frac{d}{dx} (u+v-w) &= \frac{du}{dx} + \frac{dv}{dx} - \frac{dw}{dx} \\
\frac{d}{dx} (uv) &= u \frac{dv}{dx} + v \frac{du}{dx} \\
\frac{d}{dx} (u/v) &= \frac{1}{v} \frac{du}{dx} - \frac{u}{v^2} \frac{dv}{dx} \\
\frac{d}{dx} (u^n) &= nu^{n-1} \frac{du}{dx} \\
\frac{d}{dx} (\sqrt{u}) &= \frac{1}{2\sqrt{u}} \frac{du}{dx} \\
\frac{d}{dx} (1/u) &= -\frac{1}{u^2} \frac{du}{dx} \\
\frac{d}{dx} (1/u^n) &= -\frac{n}{u^{n+1}} \frac{du}{dx} \\
\frac{d}{dx} [f(u)] &= \frac{df}{du} \frac{du}{dx} \\
\frac{d}{dx} [\ln u] &= \frac{1}{u} \frac{du}{dx} \\
\frac{d}{dx} [\log_e u] &= \frac{1}{u} \frac{du}{dx} \\
\frac{d}{dx} [e^u] &= e^u \frac{du}{dx} \\
\frac{d}{dx} [a^u] &= a^u \ln a \frac{du}{dx} \\
\frac{d}{dx} (u^r) &= r u^{r-1} \frac{du}{dx} + u^r \frac{dv}{dx} \\
\frac{d}{dx} [\sin u] &= \cos u \frac{du}{dx} \\
\frac{d}{dx} [\cos u] &= -\sin u \frac{du}{dx} \\
\frac{d}{dx} [\tan u] &= \sec^2 u \frac{du}{dx} \\
\frac{d}{dx} [\cot u] &= -\csc^2 u \frac{du}{dx} \\
\frac{d}{dx} [\sec u] &= \sec u \tan u \frac{du}{dx} \\
\frac{d}{dx} [\csc u] &= -\csc u \cot u \frac{du}{dx}
\end{align*}
\]

[source: http://hyperphysics.phy-astr.gsu.edu/hbase/Math/derfunc.html]
How about computing all the derivatives?

- But neural net \( f \) is never one of those?
  - No problem: CHAIN RULE:

\[
\text{If } f(x) = g(h(x))
\]

\[
\text{Then } f'(x) = g'(h(x))h'(x)
\]

\( \rightarrow \) Derivatives can be computed by following well-defined procedures
Automatic Differentiation

- **Automatic differentiation software**
  - e.g. Theano, TensorFlow, PyTorch, Chainer
  - Only need to program the function $g(x, y, w)$
  - Can automatically compute all derivatives w.r.t. all entries in $w$

- Need to know this exists
- How is this done? -- outside of scope of CSE573
Summary of Key Ideas

- Optimize probability of label given input
  \[ \max_w \ ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w) \]

- Continuous optimization
  - Gradient ascent:
    - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
    - Take step in the gradient direction
    - Repeat (until held-out data accuracy starts to drop = “early stopping”)

- Deep neural nets
  - Last layer = still logistic regression
  - Now also many more layers before this last layer
    - = computing the features
    - \( \rightarrow \) the features are learned rather than hand-designed
  - Automatic differentiation gives the derivatives efficiently (how? = outside of scope of 573)
Deep Reinforcement Learning