Recap: Bayesian learning

- **Prior** $P(H)$, training data $X = x_1, ..., x_N$
- Given the data so far, each hypothesis has a posterior probability:
  - $P(h_k | X) = \alpha P(X | h_k)P(h_k) = \alpha \times \text{Likelihood} \times \text{Prior}$
- Predictions use a likelihood-weighted average over the hypotheses:
  - $P(x_{N+1} | X) = \sum_k P(x_{N+1} | X, h_k)P(h_k | X) = \sum_k P(x_{N+1} | h_k)P(h_k | X)$
- No need to pick one best-guess hypothesis!
  - Drawback: $\sum_k$ may be expensive/impossible for large/infinite $H$
Recap: Logistic Regression

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

- Sigmoid function

\[
\phi(z) = \frac{1}{1 + e^{-z}}
\]
Recap: Maximum Likelihood Estimation for Logistic Regression

- Maximum likelihood estimation:

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

with:

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

\[
P(y^{(i)} = -1 | x^{(i)}; w) = 1 - \frac{1}{1 + e^{-w \cdot f(x^{(i)})}}
\]
Recap: 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: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ = gradient
Recap: Neural Networks

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

\( g = \text{nonlinear activation function} \)
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
Universal Function Approximation Theorem*

- In words: Given any continuous function \( f(x) \), if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate \( f(x) \).

Hornik theorem 1: Whenever the activation function is bounded and nonconstant, then, for any finite measure \( \mu \), standard multilayer feedforward networks can approximate any function in \( L^p(\mu) \) (the space of all functions on \( R^k \) such that \( \int_{R^k} |f(x)|^p d\mu(x) < \infty \)) arbitrarily well, provided that sufficiently many hidden units are available.

Hornik theorem 2: Whenever the activation function is continuous, bounded and non-constant, then, for arbitrary compact subsets \( X \subseteq R^k \), standard multilayer feedforward networks can approximate any continuous function on \( X \) arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

Cybenko (1989) “Approximations by superpositions of sigmoidal functions”
Hornik (1991) “Approximation Capabilities of Multilayer Feedforward Networks”
Leshno and Schocken (1991) “Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function”
Universal Function Approximation Theorem*

The Universal Function Approximation Theorem states that a feedforward neural network with a single hidden layer and a non-polynomial activation function can approximate any continuous function on a compact subset of the input space. This theorem is fundamental to the understanding of the capabilities of neural networks in function approximation.

1. Introduction

The approximation capabilities of neural networks have been extensively investigated by many authors, including Cybenko (1989), Hornik (1991), and Leshno and Schocken (1991). These studies have shown that neural networks can approximate a wide range of functions, including complex mappings and non-polynomial functions.

1.1. Approximation Capabilities of Multilayer Feedforward Networks

Cybenko (1989) "Approximations by superpositions of sigmoids functions"
Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks"
Leshno and Schocken (1991) "Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function"
How about computing all the derivatives?

- Derivatives tables:

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

[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:

$$f(x) = g(h(x))$$

Then

$$f'(x) = g'(h(x))h'(x)$$

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 $w$
  - This is typically done by caching info during forward computation pass of $f$, and then doing a backward pass = “backpropagation”
  - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass

- Need to know this exists
- How this is done? -- outside of scope of CS188
Training a Network (setting weights)

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

\( g \) = nonlinear activation function
Training a Network

Key words:
• Forward
• Backwards
• Gradient
• Backprop

\[ g = \text{nonlinear activation function} \]
Back Propagation: \( g(w) = w_1^3 w_2 + 3w_1 \)

Suppose we have \( g(w) = w_1^3 w_2 + 3w_1 \) and want the gradient at \( w = [2, 3] \).

- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute \( \frac{\partial g}{\partial w_1} \) and \( \frac{\partial g}{\partial w_2} \).
    - \( \frac{\partial g}{\partial w_1} = \)
    - \( \frac{\partial g}{\partial w_2} = \)

Computation Graph
Back Propagation: $g(w) = w_1^3w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3w_2 + 3w_1$ and want the gradient at $w = [2, 3]$.
  - Think of the function as a composition of many functions.
    - Can use derivative chain rule to compute $\frac{\partial g}{\partial w_1}$ and $\frac{\partial g}{\partial w_2}$.
  - $g = b + c$
    - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
Back Propagation: $g(w) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3 w_2 + 3w_1$ and want the gradient at $w = [2, 3]$.
  - Think of the function as a composition of many functions.
    - Can use derivative chain rule to compute $\frac{\partial g}{\partial w_1}$ and $\frac{\partial g}{\partial w_2}$.
  - $g = b + c$
    - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
  - $b = a \times w_2$
    - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a}$

![Diagram showing the computation of gradients](image)
Suppose we have \( g(w) = w_1^3 w_2 + 3w_1 \) and want the gradient at \( w = [2, 3] \).

- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute \( \frac{\partial g}{\partial w_1} \) and \( \frac{\partial g}{\partial w_2} \).

\[
\begin{align*}
g &= b + c \\
\frac{\partial g}{\partial b} &= 1, \quad \frac{\partial g}{\partial c} = 1 \\
b &= a \times w_2 \\
\frac{\partial g}{\partial a} &= \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = ?
\end{align*}
\]
Back Propagation: $g(w) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3 w_2 + 3w_1$ and want the gradient at $w = [2, 3]$.
  - Think of the function as a composition of many functions.
    - Can use derivative chain rule to compute $\frac{\partial g}{\partial w_1}$ and $\frac{\partial g}{\partial w_2}$.
  - $g = b + c$
    - $\frac{\partial g}{\partial b} = 1$, $\frac{\partial g}{\partial c} = 1$
  - $b = a \times w_2$
    - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

![Diagram showing the calculation of the gradient using the chain rule.](image)
Back Propagation: $g(w) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3 w_2 + 3w_1$ and want the gradient at $w = [2, 3]$
  - Think of the function as a composition of many functions.
    - Can use derivative chain rule to compute $\frac{\partial g}{\partial w_1}$ and $\frac{\partial g}{\partial w_2}$.

- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = ???$
Back Propagation: $g(w) = w_1^3w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3w_2 + 3w_1$ and want the gradient at $w = [2, 3]$.
  - Think of the function as a composition of many functions.
    - Can use derivative chain rule to compute $\frac{\partial g}{\partial w_1}$ and $\frac{\partial g}{\partial w_2}$.

- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$

Interpretation: A tiny increase in $w_1$ will result in an approximately 36 times increase in $g$ due to this computation path.
Suppose we have $g(w) = w_1^3 w_2 + 3w_1$ and want the gradient at $w = [2, 3]$

- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute $\frac{\partial g}{\partial w_1}$ and $\frac{\partial g}{\partial w_2}$.

- $g = b + c$
  - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$

- $b = a \times w_2$
  - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$

- $a = w_1^3$
  - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$

- $\frac{\partial g}{\partial w_2}$

**Hint:** $b = a \times 3$ may be useful.
Back Propagation: \( g(\mathbf{w}) = w_1^3 w_2 + 3w_1 \)

- Suppose we have \( g(\mathbf{w}) = w_1^3 w_2 + 3w_1 \) and want the gradient at \( \mathbf{w} = [2, 3] \)
- Think of the function as a composition of many functions.
  - Can use derivative chain rule to compute \( \partial g / \partial w_1 \) and \( \partial g / \partial w_2 \).
- \( g = b + c \)
  - \( \frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1 \)
- \( b = a \times w_2 \)
  - \( \frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3 \)
  - \( \frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial w_2} = 1 \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8 \)
- \( a = w_1^3 \)
  - \( \frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36 \)
Back Propagation: $g(w) = w_1^3w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3w_2 + 3w_1$ and want the gradient at $w = [2, 3]$
  - Think of the function as a composition of many functions, use chain rule.
  - $g = b + c$
    - $\frac{\partial g}{\partial b} = 1$, $\frac{\partial g}{\partial c} = 1$
  - $b = a \times w_2$
    - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
    - $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial w_2} = 1 \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$
  - $a = w_1^3$
    - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$
  - $c = 3w_1$
    - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c} \frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$

Adding the changes to $g$ contributed by change in $w_1$ together
Back Propagation: $g(w) = w_1^3 w_2 + 3w_1$

- Suppose we have $g(w) = w_1^3 w_2 + 3w_1$ and want the gradient at $w = [2, 3]$
- Think of the function as a composition of many functions, use chain rule.
  - $g = b + c$
    - $\frac{\partial g}{\partial b} = 1, \frac{\partial g}{\partial c} = 1$
  - $b = a \times w_2$
    - $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial a} = 1 \frac{\partial b}{\partial a} = 1 \cdot 3 = 3$
    - $\frac{\partial g}{\partial w_2} = \frac{\partial g}{\partial b} \frac{\partial b}{\partial w_2} = 1 \frac{\partial b}{\partial w_2} = 1 \cdot 8 = 8$
  - $a = w_1^3$
    - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial a} \frac{\partial a}{\partial w_1} = 3 \cdot 3w_1^2 = 36$
  - $c = 3w_1$
    - $\frac{\partial g}{\partial w_1} = \frac{\partial g}{\partial c} \frac{\partial c}{\partial w_1} = 1 \cdot 3 = 3$

\[
\nabla g = \left[ \frac{\partial g}{\partial w_1}, \frac{\partial g}{\partial w_2} \right] = [39, 8]
\]
The diagram illustrates the relationship between variables and gradients. It shows how gradients, both downstream and upstream, are calculated and used in computations. The local gradient is highlighted, and the terms for downstream and upstream gradients are defined. The variables $x$, $y$, and $z$ are input to the function $f$, and the gradients are calculated as follows:

- Downstream gradients:
  - $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}$
  - $\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}$

- Upstream gradient:
  - $\frac{\partial L}{\partial z}$
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

- Deep neural nets
  - Layered computation graph
    - Last layer = often logistic regression
    - Now also many more layers before this last layer
      - = computing the features
        - the features are learned rather than hand-designed
    - Different neural network architectures: CNN, RNN, LSTM, Transformer

- Universal function approximation theorem
  - If neural net is large enough
  - Then neural net can represent any continuous mapping from input to output with arbitrary accuracy
  - But remember: need to avoid overfitting / memorizing the training data; early stopping!

- Automatic differentiation gives the derivatives efficiently
Different Neural Network Architectures

Convolutional network (AlexNet)

- Input image
- Weights
- Loss

Neural network as General computation graph

Krizhevsky, Sutskever, Hinton, 2012
Different Neural Network Architectures

- Exploration of different neural network architectures
  - ResNet: residual networks
  - Networks with attention
  - Transformer networks
- Neural network architecture search
- Really large models
  - GPT2, GPT3
  - CLIP
In a shocking finding, a scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. “By the time we reached the top of one peak, the water looked blue, with some crystals on top,” said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them— they were so close they could touch their horns.

While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, “We can see, for example, that they have a common ‘language,’ something like a dialect or dialectic.”

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, “In South America, such incidents seem to be quite common.”

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. “But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization,” said the scientist.
A college student used GPT-3 to write fake blog posts and ended up at the top of Hacker News

He says he wanted to prove the AI could pass as a human writer

By Kim Lyons | Aug 16, 2020, 1:55pm EDT