Deep Learning and Backpropagation

Boaz Barak

Neuro 140/240 guest lecture 2/8/2022

Recommended reading: (links on website)
Learning

Unknown \( f: X \rightarrow Y \)  
Observe \( (x_1, f(x_1)), \ldots, (x_n, f(x_n)) \)

Goal: Find \( h \) s.t. \( h(x) \approx f(x) \)

Hypothesis

With high prob over \( x \), \( h(x) \) close to \( f(x) \)

Examples:

1. \( f: \{\text{pixels}\} \rightarrow \{\text{"cat","dog"}\} \), \( f(\quad) = \text{"cat"} \)

\( f: \{\text{pixels}\} \rightarrow \{\text{"benign","malignant"}\} \), \( f(\quad) = \text{"benign"} \)
Learning

Unknown \( f: X \rightarrow Y \) 

Observe \((x_1, f(x_1)), \ldots, (x_n, f(x_n))\)

**Goal:** Find \( h \) s.t. \( h(x) \approx f(x) \)

**Hypothesis**

With high prob over \( x \), 
\( h(x) \) close to \( f(x) \)

Examples:

2. \( f: \{ \text{chess positions} \} \rightarrow \mathbb{N}, \ f(\text{position}) = 157 \)

Quality score of position
Learning

Uknown $f: \mathcal{X} \to \mathcal{Y}$

Observe $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$

$f(\text{"I hate MIT"}) = 2^{-32.4}$

$f(\text{"I love MIT"}) = 2^{-30.4}$

Both schools loved at 4:1 ratio but Harvard talked about twice as much

Goal: Find $h$ s.t. $h(x) \approx f(x)$

With high prob over $x$, $h(x)$ close to $f(x)$

Examples:

3. $f: \{\text{sentences}\} \to [0,1], f(\text{"I hate Harvard"}) = 2^{-31.2}$

$f(\text{"I love Harvard"}) = 2^{-29.3}$

* GPT-3 provided probabilities of the Pr[ "love/hate X" | "I"] multiplied by $2^{-12.8}$ for the frequency of "I".
Learning is a tool

Data → Function Approximation → Higher level algorithm → Decisions

Approximation
Learning is a tool

Data → Function Approximation → Higher level algorithm → Decisions

Examples:

\[ f: \{\text{pixels}\} \rightarrow \{\text{"benign","malignant"}\}, \quad f(\text{image}) = \text{"benign"} \]

refer to human doctor
Learning is a tool

**Data**

Function Approximation

Higher level algorithm

**Decisions**

Examples:

\[ f: \{ \text{chess positions} \} \rightarrow \mathbb{N}, f(\text{position}) = 157 \]

Search for move maximizing score, continue virtual playing to update score
Learning is a tool

Data \rightarrow \text{Function Approximation} \rightarrow \text{Higher level algorithm} \rightarrow \text{Decisions}

Examples:

\[ f: \{\text{sentences}\} \rightarrow [0,1], \quad f(\text{"I hate Harvard"}) = 2^{-31.23} \]

Generate continuation \( w_k \) to \( w_1 \ldots w_{k-1} \) by maximizing \( f(w_1 \ldots w_k) \)
I hate Harvard and my degree."

“Whoa,” I said. “That’s a big one.”

“I’m serious.”

“Why?”

“I don’t want to talk about it"

I love Harvard. I love the people in my life that I’ve met here,” she said.

“I’m really excited to go back to my school and my family and my friends ... but I also feel really at home here,” she added.
Deep Learning in One Slide

```python
f = random_program()
for x,y in training_data:
    loss = abs(f(x)-y)
    f_pert =  # perturbation of f with smaller loss
    f = f_pert
```
Why dial is important

Deep Learning performance
“[the bitter lesson] is the great power of general-purpose methods ... that continue to scale with increased computation even as the available computation becomes very great”

*The Bitter Lesson, Rich Sutton*

“Every sufficiently advanced technology is indistinguishable from potatoes.”
Deep Learning in One Slide

```python
f = random_program()
for x,y in training_data:
    loss = abs(f(x)-y)
    f_pert =  # perturbation of f with smaller loss
    f = f_pert
```

**Data**
(# samples)

**Memory**
(size of f)

**Compute**
(# steps)

**Needed:**
- Represent program $f$
- Find perturbation
What is a program?

A program is a circuit

\[ \text{Output} = f(\vec{x}) \]
What is a Neural Network? A circuit.

often \( f(x, w) = \sigma(w_0 + \sum w_i x_i) \)

Output = \( N(\mathbf{\hat{x}}, \mathbf{\hat{w}}) \)
What is a Neural Network?

A circuit.

often $f(x,w) = \sigma(w_0 + \sum w_i x_i)$

$\mathbf{N}(\mathbf{x}, \mathbf{w})$

Want: If $\mathbf{w}_1 \approx \mathbf{w}_2$ then $\mathbf{N}(\mathbf{x}, \mathbf{w}_1) \approx \mathbf{N}(\mathbf{x}, \mathbf{w}_2)$

Cor: Perturb $f(\mathbf{x}) = \mathbf{N}(\mathbf{x}, \mathbf{w})$ by $\mathbf{w} \leftarrow \mathbf{w} + \nabla$
Training: perturb weights until net fits the training data.
Training a Neural Net

This is the output from one neuron. Hover to see it.

The outputs are mixed with varying weights, shown by the thickness of the lines.
Training a Neural Net
Math starts now
Learning

Uknown \( f: X \to Y \)

Observe \((x_1, f(x_1)), ..., (x_n, f(x_n))\)

**Hypothesis**

Goal: Find \( h \) s.t. \( h(x) \approx f(x) \)

Define \( \mathcal{H} \) - class of hypotheses

\( L: Y \times Y \to \mathbb{R} \) - loss function

Output: \( \hat{h} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(f(x_i), h(x_i)) \)

Hope: \( \hat{h} \approx \arg\min_{h \in \mathcal{H}} \mathbb{E}_{x,y} L(f(x), h(y)) \)
Learning

Unknown \( f: \mathcal{X} \to \mathcal{Y} \)

Observe \((x_1, f(x_1)), \ldots, (x_n, f(x_n))\)

General approach: Define \( \mathcal{H} \) - \textit{class of hypotheses}

\[ L: \mathcal{Y} \times \mathcal{Y} \to \mathbb{R} \] - \textit{loss function}

Output:

\[ \hat{h} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(f(x_i), h(x_i)) \]

Hope:

\[ \hat{h} \approx \arg\min_{h \in \mathcal{H}} \mathbb{E}_{x,y} L(f(x), h(y)) \]
Gradient Descent

**Input:** \((x_1, y_1), \ldots, (x_n, y_n)\)

**Goal:** Compute 
\[
\hat{h} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(h(x_i), y_i)
\]

**Assume:**
Every \(h \in \mathcal{H}\) specified by parameter \(\theta \in \mathbb{R}\)

\(\theta\) is variable
\(x_i, y_i\) are constant

**Define:** 
\[
L(\theta) := \sum_{i=1}^{n} L(h_\theta(x_i), y_i)
\]

**Goal:** Compute 
\[
\hat{\theta} = \arg\min_{\theta \in \mathbb{R}} L(\theta)
\]

\[
\theta_1 = \theta_0 - \eta L'(\theta_0)
\]
Gradient Descent

Input: \((x_1, y_1), \ldots, (x_n, y_n)\)

Goal: Compute \(\hat{h} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(h(x_i), y_i)\)

Assume:
Every \(h \in \mathcal{H}\) specified by parameter \(\theta \in \mathbb{R}\)

Define: \(L(\theta) := \sum_{i=1}^{n} L(h_\theta(x_i), y_i)\)

Goal: Compute \(\hat{\theta} = \arg\min_{\theta \in \mathbb{R}} L(\theta)\)

\[
\theta_1 = \theta_0 - \eta L'(\theta_0) \\
\theta_2 = \theta_1 - \eta L'(\theta_1)
\]
Gradient Descent

**Input:** \((x_1, y_1), \ldots, (x_n, y_n)\)

**Goal:** Compute

\[
\hat{h} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(h(x_i), y_i)
\]

**Assume:**
Every \(h \in \mathcal{H}\) specified by parameter \(\theta \in \mathbb{R}\)

**Define:**
\[
L(\theta) := \sum_{i=1}^{n} L(h_\theta(x_i), y_i)
\]

**Goal:** Compute

\[
\hat{\theta} = \arg\min_{\theta \in \mathbb{R}} L(\theta)
\]

slope = \(L'(\theta_2)\)

\[
\begin{align*}
\theta_1 &= \theta_0 - \eta L'(\theta_0) \\
\theta_2 &= \theta_1 - \eta L'(\theta_1) \\
\theta_3 &= \theta_2 - \eta L'(\theta_2)
\end{align*}
\]
Gradient Descent

Input: \((x_1, y_1), \ldots, (x_n, y_n)\)

Goal: Compute \(\hat{h} = \arg \min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(h(x_i), y_i)\)

Assume:
Every \(h \in \mathcal{H}\) specified by parameter \(\theta \in \mathbb{R}\)

Define: \(L(\theta) := \sum_{i=1}^{n} L(h_\theta(x_i), y_i)\)

Goal: Compute \(\hat{\theta} = \arg \min_{\theta \in \mathbb{R}} L(\theta)\)

\[
\begin{align*}
\theta_1 &= \theta_0 - \eta L'(\theta_0) \\
\theta_2 &= \theta_1 - \eta L'(\theta_1) \\
\theta_3 &= \theta_2 - \eta L'(\theta_2)
\end{align*}
\]
**Stochastic Gradient Descent**

**Input:** \((x_1, y_1), \ldots, (x_n, y_n)\)

**Goal:** Compute

\[
\hat{h} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(h(x_i), y_i)
\]

**Assume:**
Every \(h \in \mathcal{H}\) specified by parameter \(\theta \in \mathbb{R}\)

**Define:**

\[
L(\theta) := \sum_{i=1}^{n} L(h_\theta(x_i), y_i)
\]

**Observation:**

\[
L'(\theta) = \sum_{i=1}^{n} L'(h_\theta(x_i), y_i) \propto \mathbb{E}_{i \sim [n]}[L'(h_\theta(x_i), y_i)]
\]

\[
\theta_1 = \theta_0 - \eta L'(\theta_0)
\]

\[
\theta_2 = \theta_1 - \eta L'(\theta_1)
\]

\[
\theta_3 = \theta_2 - \eta L'(\theta_2)
\]

\[
\min L(\theta)
\]
**Stochastic Gradient Descent**

**Input:** \((x_1, y_1), ..., (x_n, y_n)\)

**Goal:** Compute  
\[ \hat{h} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} L(h(x_i), y_i) \]

**Assume:**  
Every \(h \in \mathcal{H}\) specified by parameter \(\theta \in \mathbb{R}\)

**Observation:**  
\[ L'(\theta) = \sum_{i=1}^{n} L'(h_\theta(x_i), y_i) \propto \mathbb{E}_{i \sim [n]} [ L'(h_\theta(x_i), y_i) ] \]

**Algorithm:**
1. \(\theta \leftarrow \$\)
2. for \(t = 1, ..., \#\text{epochs}\):
   for \(i \in \text{shuffle}(1, ..., n)\):
      \[ \theta \leftarrow \theta - \eta \cdot \frac{dL(h_\theta(x_i), y_i)}{d\theta} \]
Training a Neural Network

**SGD:**

1. $\vec{w} \leftarrow \$ 
2. for $t = 1, \ldots, \# \text{epochs}$:
   
   for $i \in \text{shuffle}(1, \ldots, n)$:
   
   $\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla_{\vec{w}} L_i(\vec{w})$

\[
L_i(\vec{w}) = L(N(x_i, \vec{w}), y_i)
\]

\[
\nabla_{\vec{w}} L_i(\vec{w}) = \left( \frac{\partial L(N(x_i, \vec{w}), y_i)}{\partial w_1}, \ldots, \frac{\partial L(N(x_i, \vec{w}), y_i)}{\partial w_k} \right)
\]
Neural Networks Gradient

\[ \nabla L(w) = \left( \frac{\partial L}{\partial w_1}, \ldots, \frac{\partial L}{\partial w_k} \right) \approx \left( \frac{L(w + \epsilon b_1) - L(w)}{\epsilon}, \ldots, \frac{L(w + \epsilon b_k)}{\epsilon} \right) \]

Naïve Approach: Take \( v_1, \ldots, v_k \) basis for \( \mathbb{R}^k \), measure \( \langle \nabla, v_i \rangle \approx \frac{L(w + \epsilon v_i) - L(w)}{\epsilon} \)

Cost: \( k + 1 \) evaluations of network

Signal: One global scalar per evaluation, “neurologically plausible”
Neural Networks Gradient 

Backpropagation: Recursive algorithm to compute $\frac{\partial L}{\partial g}$ for every gate $g$ in net

Cost: Two evaluations of network

Signal: Propagate gradient vector backward, “neurologically implausible?”
Multivariate chain rule

\[
\frac{\partial z}{\partial u} = \left( \frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w} \right)
\]
If you know 
\[
\frac{\partial z}{\partial v_1}, \frac{\partial z}{\partial v_2}, \frac{\partial z}{\partial v_3}
\]

You can compute \( \frac{\partial z}{\partial u} \)

\[
\frac{\partial z}{\partial u} = \left( \frac{\partial v_1}{\partial u} \cdot \frac{\partial z}{\partial v_1} + \frac{\partial v_2}{\partial u} \cdot \frac{\partial z}{\partial v_2} + \frac{\partial v_2}{\partial u} \cdot \frac{\partial z}{\partial v_3} \right)
\]
Example

\[ v = u^3 \]

\[ z = v \cdot w \]

\[ w = u + u \]

\[
\frac{\partial z}{\partial u} = \left( \frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w} \right) = 3 \cdot u^2 \cdot w + 2 \cdot v
\]

\[ u = 5 \Rightarrow \frac{\partial z}{\partial u} = 3 \cdot 25 \cdot 10 + 250 = 1000 \]
\[
\begin{align*}
\frac{\partial z}{\partial u} &= \left( \frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w} \right)
= 3 \cdot u^2 \cdot w + 2 \cdot v \\
\end{align*}
\]
\[
\begin{align*}
u &= u^3 \\
w &= u + u \\
z &= v \cdot w \\
\end{align*}
\]
\[
\begin{align*}
u &= 5 \Rightarrow \frac{\partial z}{\partial u} = 3 \cdot 25 \cdot 10 + 250 = 1000
\end{align*}
\]
\[
\begin{align*}
\frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} &= 10 \cdot 3 \cdot u^2 = 750 \\
\frac{\partial z}{\partial v} &= 10 \\
\frac{\partial z}{\partial w} &= 125 \\
\frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w} &= 2 \cdot 125 = 250 \\
\frac{\partial z}{\partial u} &= (\frac{\partial v}{\partial u} \cdot \frac{\partial z}{\partial v} + \frac{\partial w}{\partial u} \cdot \frac{\partial z}{\partial w}) = 3 \cdot u^2 \cdot w + 2 \cdot v \\
\frac{\partial z}{\partial u} &= 3 \cdot 25 \cdot 10 + 250 = 1000
\end{align*}
\]
Summary

• "Neural networks" are not about neurons – architecture details not as important.

Some method for parameterized functions such that

• Increase complexity by increasing parameters

• Efficiently find small perturbations improving objective

• Back propagation: Method to find perturbation using constant number of evaluations.

Cost: Vector-valued feedback from reward layer to computation layer