Backpropagation

Karl Stratos

June 25, 2018

Review/Setup

- A **model** is a function f_θ defined by a set of **parameters** θ that receives an input $\boldsymbol{x}$ and outputs some value.
- For example, a logistic regressor is parameterized by a single vector $\theta = \{\boldsymbol{w}\}$ and defines

$$f_{\boldsymbol{w}}(\boldsymbol{x}) := \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})} \in [0, 1]$$

which represents the probability of "on" for the given input $\boldsymbol{x}$.

Review/Setup

- A **model** is a function f_θ defined by a set of **parameters** θ that receives an input $\boldsymbol{x}$ and outputs some value.

- For example, a logistic regressor is parameterized by a single vector $\theta = \{\boldsymbol{w}\}$ and defines

$$f_{\boldsymbol{w}}(\boldsymbol{x}) := \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})} \in [0, 1]$$

  which represents the probability of "on" for the given input $\boldsymbol{x}$.

- The model is **trained** by minimizing some average **loss** $J_S(\theta)$ on training data S (e.g., the log loss for logistic regression).

Review/Setup

- A **model** is a function f_θ defined by a set of **parameters** θ that receives an input $\boldsymbol{x}$ and outputs some value.

- For example, a logistic regressor is parameterized by a single vector $\theta = \{\boldsymbol{w}\}$ and defines

$$f_{\boldsymbol{w}}(\boldsymbol{x}) := \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x})} \in [0, 1]$$

  which represents the probability of "on" for the given input $\boldsymbol{x}$.

- The model is **trained** by minimizing some average **loss** $J_S(\theta)$ on training data S (e.g., the log loss for logistic regression).

- If $J_S(\theta)$ is differentiable, we can use stochastic gradient descent (SGD) to efficiently minimize the loss.

Sketch of SGD

Initialize model parameters θ and repeat the following:

1. Choose a random "mini-batch" $B \subset S$ of your training data.

Sketch of SGD

Initialize model parameters θ and repeat the following:

1. Choose a random "mini-batch" $B \subset S$ of your training data.

2. Define a "partial" loss function $J_B(\theta)$ on this mini-batch.

Sketch of SGD

Initialize model parameters θ and repeat the following:

1. Choose a random "mini-batch" $B \subset S$ of your training data.

2. Define a "partial" loss function $J_B(\theta)$ on this mini-batch.

3. Calculate the **gradient of** $J_B(\theta)$ (with respect to θ)

$$\nabla J_B(\theta)$$

Sketch of SGD

Initialize model parameters θ and repeat the following:

1. Choose a random "mini-batch" $B \subset S$ of your training data.

2. Define a "partial" loss function $J_B(\theta)$ on this mini-batch.

3. Calculate the **gradient of** $J_B(\theta)$ (with respect to θ)

$$\nabla J_B(\theta)$$

4. Update the parameter value

$$\theta \leftarrow \theta - \eta \nabla J_B(\theta)$$

Calculating the Gradient

- Implication: we can optimize *any* (differentiable) average loss function by SGD if we can calculate the gradient of the scalar-valued loss function $J_B(\theta) \in \mathbb{R}$ on any batch B with respect to parameter θ.

Calculating the Gradient

- Implication: we can optimize *any* (differentiable) average loss function by SGD if we can calculate the gradient of the scalar-valued loss function $J_B(\theta) \in \mathbb{R}$ on any batch B with respect to parameter θ.

- For simple models, we can manually specify the gradient. For example, we *derived* the gradient of the log loss

$$\nabla J_B^{\mathrm{LOG}}(\boldsymbol{w}) = \frac{1}{|B|} \sum_{(\boldsymbol{x},y) \in B} (y - f_{\boldsymbol{w}}(\boldsymbol{x}))\, \boldsymbol{x} \in \mathbb{R}^d$$

  and calculated this vector on batch B to update the parameter $\boldsymbol{w} \in \mathbb{R}^d$.

Problems with Manually Deriving Gradient Formula?

Problems with Manually Deriving Gradient Formula?

- It is specific to a particular loss function.
  - For a new loss function, you have to derive its gradient again.

Problems with Manually Deriving Gradient Formula?

- It is specific to a particular loss function.
  - For a new loss function, you have to derive its gradient again.
- What if loss $J_B(\theta)$ is an *extremely* complicated function of θ?
  - It is technically possible to manually derive a gradient formula, but it is tedious/difficult/error-prone.

Backpropagation: Input and Output

- A technique to automatically calculate $\nabla J_B(\theta)$ for any definition of scalar-valued loss function $J_B(\theta) \in \mathbb{R}$.

  Input: loss function $J_B(\theta) \in \mathbb{R}$, parameter value $\hat{\theta}$
  Output: $\nabla J_B(\hat{\theta})$, the gradient of $J_B(\theta)$ at $\theta = \hat{\theta}$

Backpropagation: Input and Output

- A technique to automatically calculate $\nabla J_B(\theta)$ for any definition of scalar-valued loss function $J_B(\theta) \in \mathbb{R}$.

  Input: loss function $J_B(\theta) \in \mathbb{R}$, parameter value $\hat{\theta}$
  Output: $\nabla J_B(\hat{\theta})$, the gradient of $J_B(\theta)$ at $\theta = \hat{\theta}$

- For example, when applied to the log loss $J_B^{\mathrm{LOG}}(\hat{\boldsymbol{w}}) \in \mathbb{R}$ at some parameter $\hat{\boldsymbol{w}} \in \mathbb{R}^d$, it calculates $\nabla J_B^{\mathrm{LOG}}(\hat{\boldsymbol{w}}) \in \mathbb{R}^d$ without needing an explicit gradient formula.

Backpropagation: Input and Output

- A technique to automatically calculate $\nabla J_B(\theta)$ for any definition of scalar-valued loss function $J_B(\theta) \in \mathbb{R}$.

  Input: loss function $J_B(\theta) \in \mathbb{R}$, parameter value $\hat{\theta}$

  Output: $\nabla J_B(\hat{\theta})$, the gradient of $J_B(\theta)$ at $\theta = \hat{\theta}$

- For example, when applied to the log loss $J_B^{\mathrm{LOG}}(\hat{\boldsymbol{w}}) \in \mathbb{R}$ at some parameter $\hat{\boldsymbol{w}} \in \mathbb{R}^d$, it calculates $\nabla J_B^{\mathrm{LOG}}(\hat{\boldsymbol{w}}) \in \mathbb{R}^d$ without needing an explicit gradient formula.

- More generally, it can calculate the gradient of an *arbitrarily complicated* (differentiable) function of parameter θ.

  Including neural networks

Overview

Calculus Warm-Up

Directed Acyclic Graph (DAG)

Backpropagation

- Computation Graph, Forward Pass
- Backpropagation

Notation

- For the most part, we will consider (differentiable) function $f : \mathbb{R} \to \mathbb{R}$ with a single 1-dimensional parameter $x \in \mathbb{R}$.

- The gradient/derivative of f is a *function* of x and written as

$$\frac{\partial f(x)}{\partial x} : \mathbb{R} \to \mathbb{R}$$

- The *value* of the gradient of f with respect to x at $x = a$ is written as

$$\left.\frac{\partial f(x)}{\partial x}\right|_{x=a} \in \mathbb{R}$$

Chain Rule

- Given any differentiable functions f, g from $\mathbb{R}$ to $\mathbb{R}$,

$$\frac{\partial g(f(x))}{\partial x} = \frac{\partial g(f(x))}{\partial f(x)} \times \underbrace{\frac{\partial f(x)}{\partial x}}_{\text{easy to calculate}}$$

Exercises

At $x = 42$,

- What is the value of the gradient of $f(x) := 7$?
- What is the value of the gradient of $f(x) := 2x$?
- What is the value of the gradient of $f(x) := 2x + 99999$?
- What is the value of the gradient of $f(x) := x^3$?
- What is the value of the gradient of $f(x) := \exp(x)$?
- What is the value of the gradient of $f(x) := \exp(2x^3 + 10)$?
- What is the value of the gradient of

$$f(x) := \log(\exp(2x^3 + 10))$$

Chain Rule for a Function of Multiple Input Variables

- Let $f_1 \dots f_m$ denote any differentiable functions from $\mathbb{R}$ to $\mathbb{R}$.
- If $g : \mathbb{R}^m \to \mathbb{R}$ is a differentiable function from $\mathbb{R}^m$ to $\mathbb{R}$,

$$\frac{\partial g(f_1(x), \dots, f_m(x))}{\partial x}$$

$$= \sum_{i=1}^{m} \frac{\partial g(f_1(x), \dots, f_m(x))}{\partial f_i(x)} \times \underbrace{\frac{\partial f_i(x)}{\partial x}}_{\text{easy to calculate}}$$

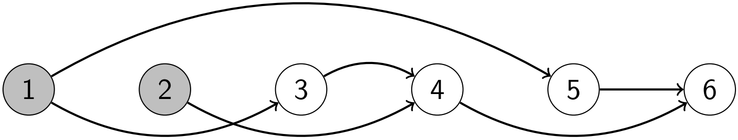
- Calculate the gradient of $x + x^2 + yx$ with respect to x using the chain rule.

Overview

- Calculus Warm-Up
- Directed Acyclic Graph (DAG)
- Backpropagation
  - Computation Graph, Forward Pass
  - Backpropagation

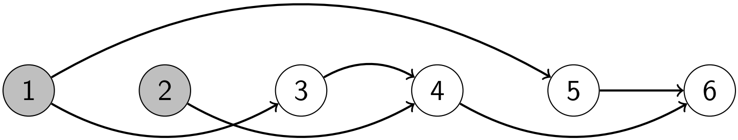
DAG

A **directed acylic graph (DAG)** is a directed graph $G = (V, A)$ with a **topological ordering**: a sequence π of V such that for every arc $(i, j) \in A$, i comes before j in π.



For backpropagation: usually assume have many roots and 1 leaf

Notation



$$
\begin{aligned}
V &= \{1, 2, 3, 4, 5, 6\} \\
V_I &= \{1, 2\} \\
V_N &= \{3, 4, 5, 6\} \\
A &= \{(1,3), (1,5), (2,4), (3,4), (4,6), (5,6)\} \\
\textbf{pa}(4) &= \{2, 3\} \\
\textbf{ch}(1) &= \{3, 5\} \\
\Pi_G &= \{(1,2,3,4,5,6),\ (2,1,3,4,5,6)\}
\end{aligned}
$$

Overview

- Calculus Warm-Up
- Directed Acyclic Graph (DAG)
- Backpropagation
  - Computation Graph, Forward Pass
  - Backpropagation

Computation Graph

- DAG $G = (V, E)$ with a single output node $\omega \in V$.
- Every node $i \in V$ is equipped with a **value** $x^i \in \mathbb{R}$:
  1. For input node $i \in V_I$, we assume $x^i = a^i$ is given.
  2. For non-input node $i \in V_N$, we assume a differentiable **function** $f^i : \mathbb{R}^{|\mathbf{pa}(i)|} \to \mathbb{R}$ and compute

$$x^i = f^i((x^j)_{j \in \mathbf{pa}(i)})$$

Computation Graph

- DAG $G = (V, E)$ with a single output node $\omega \in V$.
- Every node $i \in V$ is equipped with a **value** $x^i \in \mathbb{R}$:
  1. For input node $i \in V_I$, we assume $x^i = a^i$ is given.
  2. For non-input node $i \in V_N$, we assume a differentiable **function** $f^i : \mathbb{R}^{|\mathbf{pa}(i)|} \to \mathbb{R}$ and compute

$$x^i = f^i((x^j)_{j \in \mathbf{pa}(i)})$$

- Thus G represents a *function*: it receives multiple values $x^i = a^i$ for $i \in V_I$ and calculates a scalar $x^\omega \in \mathbb{R}$.
  - We can calculate x^ω by a **forward pass**.

Forward Pass

Input: computation graph $G = (V, A)$ with output node $\omega \in V$
Result: populates $x^i = a^i$ for every $i \in V$

1. Pick some topological ordering π of V.
2. For i **in order** of π, if $i \in V_N$ is a non-input node, set

$$x^i \leftarrow a^i := f^i((a^j)_{j \in \mathbf{pa}(i)})$$

Why do we need a topological ordering?

Exercise

Construct the computation graph associated with the function

$$f(x, y) := (x + y)xy^2$$

Compute its output value at $x = 1$ and $y = 2$ by performing a forward pass.

Overview

- Calculus Warm-Up
- Directed Acyclic Graph (DAG)
- Backpropagation
  - Computation Graph, Forward Pass
  - Backpropagation

For Notational Convenience...

- Collectively refer to **all input slots** by $x_I = (x^i)_{i \in V_I}$.
- Collectively refer to **all input values** by $a_I = (a^i)_{i \in V_I}$.

For Notational Convenience...

- Collectively refer to **all input slots** by $x_I = (x^i)_{i \in V_I}$.
- Collectively refer to **all input values** by $a_I = (a^i)_{i \in V_I}$.

- At $i \in V$:

  Refer to its **parental slots** by $x_I^i = (x^j)_{j \in \mathbf{pa}(i)}$.
  Refer to its **parental values** by $a_I^i = (a^j)_{j \in \mathbf{pa}(i)}$.

For Notational Convenience...

- Collectively refer to **all input slots** by $x_I = (x^i)_{i \in V_I}$.
- Collectively refer to **all input values** by $a_I = (a^i)_{i \in V_I}$.

- At $i \in V$:

  Refer to its **parental slots** by $x_I^i = (x^j)_{j \in \mathbf{pa}(i)}$.
  Refer to its **parental values** by $a_I^i = (a^j)_{j \in \mathbf{pa}(i)}$.

Two equally valid ways of viewing any $a^i \in \mathbb{R}$ as a function:

For Notational Convenience...

- Collectively refer to **all input slots** by $x_I = (x^i)_{i \in V_I}$.
- Collectively refer to **all input values** by $a_I = (a^i)_{i \in V_I}$.

- At $i \in V$:

  Refer to its **parental slots** by $x_I^i = (x^j)_{j \in \mathbf{pa}(i)}$.
  Refer to its **parental values** by $a_I^i = (a^j)_{j \in \mathbf{pa}(i)}$.

Two equally valid ways of viewing any $a^i \in \mathbb{R}$ as a function:

- A "global" function of x_I evaluated at a_I.

For Notational Convenience...

- Collectively refer to **all input slots** by $x_I = (x^i)_{i \in V_I}$.
- Collectively refer to **all input values** by $a_I = (a^i)_{i \in V_I}$.

- At $i \in V$:

  Refer to its **parental slots** by $x_I^i = (x^j)_{j \in \mathbf{pa}(i)}$.
  Refer to its **parental values** by $a_I^i = (a^j)_{j \in \mathbf{pa}(i)}$.

Two equally valid ways of viewing any $a^i \in \mathbb{R}$ as a function:

- A "global" function of x_I evaluated at a_I.
- A "local" function of x_I^i evaluated at a_I^i.

Computation Graph: Gradients

- Now for every node $i \in V$, we introduce an additional slot $z^i \in \mathbb{R}$ defined as

$$z^i := \left.\frac{\partial x^{\omega}}{\partial x^i}\right|_{x_I = a_I}$$

- **The goal of backpropagation** is to calculate z^i for every $i \in V$.
  - Why are we done if we achieve this goal?

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$z^i := \left.\frac{\partial x^{\omega}}{\partial x^i}\right|_{x_I = a_I}$$

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$z^i := \left.\frac{\partial x^{\omega}}{\partial x^i}\right|_{x_I=a_I} = \sum_{j \in \textbf{ch}(i)} \left.\frac{\partial x^{\omega}}{\partial x^j}\right|_{x_I=a_I} \times \left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j=a_I^j}$$

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$
\begin{aligned}
z^i := \left.\frac{\partial x^{\omega}}{\partial x^i}\right|_{x_I = a_I} &= \sum_{j \in \textbf{ch}(i)} \left.\frac{\partial x^{\omega}}{\partial x^j}\right|_{x_I = a_I} \times \left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j = a_I^j} \\
&= \sum_{j \in \textbf{ch}(i)} z^j \times \underbrace{\left.\frac{\partial f^j(x_I^j)}{\partial x^i}\right|_{x_I^j = a_I^j}}_{\text{easy to calculate}}
\end{aligned}
$$

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$
\begin{aligned}
z^i := \left.\frac{\partial x^{\omega}}{\partial x^i}\right|_{x_I = a_I} &= \sum_{j \in \mathbf{ch}(i)} \left.\frac{\partial x^{\omega}}{\partial x^j}\right|_{x_I = a_I} \times \left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j = a_I^j} \\
&= \sum_{j \in \mathbf{ch}(i)} z^j \times \underbrace{\left.\frac{\partial f^j(x_I^j)}{\partial x^i}\right|_{x_I^j = a_I^j}}_{\text{easy to calculate}}
\end{aligned}
$$

- If we compute z^i in a **reverse topological ordering**, then we will have already computed z^j for all $j \in \mathbf{ch}(i)$.

Key Ideas of Backpropagation

- Chain rule on the DAG structure

$$z^i := \left.\frac{\partial x^\omega}{\partial x^i}\right|_{x_I = a_I} = \sum_{j \in \textbf{ch}(i)} \left.\frac{\partial x^\omega}{\partial x^j}\right|_{x_I = a_I} \times \left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j = a_I^j}$$

$$= \sum_{j \in \textbf{ch}(i)} z^j \times \underbrace{\left.\frac{\partial f^j(x_I^j)}{\partial x^i}\right|_{x_I^j = a_I^j}}_{\text{easy to calculate}}$$

- If we compute z^i in a **reverse topological ordering**, then we will have already computed z^j for all $j \in \textbf{ch}(i)$.
  - What's the base case z^ω?

Backpropagation

Input: computation graph $G = (V, A)$ with output node $\omega \in V$ whose value slots $x^i = a^i$ are already populated for every $i \in V$
Result: populates z^i for every $i \in V$

1. Set $z^\omega \leftarrow 1$.
2. Pick some topological ordering π of V.
3. For i **in reverse order** of π, set

$$z^i \leftarrow \sum_{j \in \textbf{ch}(i)} z^j \times \left. \frac{\partial f^j(x_I^j)}{\partial x^i} \right|_{x_I^j = a_I^j}$$

Exercise

Calculate the gradient of

$$f(x, y) := (x + y)xy^2$$

with respect to x at $x = 1$ and $y = 2$ by performing backpropagation. That is, calculate the scalar

$$\left.\frac{\partial f(x, y)}{\partial x}\right|_{(x,y)=(1,2)}$$

Implementation

- Each type of function f creates a child node from parent nodes and initializes its gradient to zero.
  - "Add" function creates a child node c with two parents (a, b) and sets $c.z \leftarrow 0$.

Implementation

- Each type of function f creates a child node from parent nodes and initializes its gradient to zero.
  - "Add" function creates a child node c with two parents (a, b) and sets $c.z \leftarrow 0$.
- Each node has an associated **forward** function.
  - Calling forward at c populates $c.x = a.x + b.x$ (assumes parents have their values).

Implementation

- Each type of function f creates a child node from parent nodes and initializes its gradient to zero.
  - "Add" function creates a child node c with two parents (a, b) and sets $c.z \leftarrow 0$.
- Each node has an associated **forward** function.
  - Calling forward at c populates $c.x = a.x + b.x$ (assumes parents have their values).
- Each node also has an associated **backward** function.
  - Calling backward at c "broadcasts" its gradient $c.z$ (assumes it's already calculated) to its parents

$$a.z \leftarrow a.z + c.z$$
$$b.z \leftarrow b.z + c.z$$

Implementation (Cont.)

- Express your loss $J_B(\theta)$ on minibatch B at $\theta = \hat{\theta}$ as a computation graph.

Implementation (Cont.)

- Express your loss $J_B(\theta)$ on minibatch B at $\theta = \hat{\theta}$ as a computation graph.

- **Forward pass.** For each node a in a topological ordering,

$$a.\textbf{forward}()$$

Implementation (Cont.)

- Express your loss $J_B(\theta)$ on minibatch B at $\theta = \hat{\theta}$ as a computation graph.

- **Forward pass.** For each node a in a topological ordering,

$$a.\textbf{forward}()$$

- **Backward pass.** For each node a in a reverse topological ordering,

$$a.\textbf{backward}()$$

Implementation (Cont.)

- Express your loss $J_B(\theta)$ on minibatch B at $\theta = \hat{\theta}$ as a computation graph.

- **Forward pass.** For each node a in a topological ordering,

$$a.\textbf{forward}()$$

- **Backward pass.** For each node a in a reverse topological ordering,

$$a.\textbf{backward}()$$

- The gradient of $J_B(\theta)$ at $\theta = \hat{\theta}$ is stored in the input nodes of the computation graph.

General Backpropagation

- Computation graph in which input values that are vectors

$$x^i \in \mathbb{R}^{d^i} \qquad \forall i \in V$$

But the output value $x^\omega \in \mathbb{R}$ is always a scalar!

General Backpropagation

- Computation graph in which input values that are vectors

$$x^i \in \mathbb{R}^{d^i} \qquad \forall i \in V$$

But the output value $x^\omega \in \mathbb{R}$ is always a scalar!

- The corresponding gradients are also vectors of the same size

$$z^i \in \mathbb{R}^{d^i} \qquad \forall i \in V$$

General Backpropagation

- Computation graph in which input values that are vectors

$$x^i \in \mathbb{R}^{d^i} \qquad \forall i \in V$$

But the output value $x^\omega \in \mathbb{R}$ is always a scalar!

- The corresponding gradients are also vectors of the same size

$$z^i \in \mathbb{R}^{d^i} \qquad \forall i \in V$$

- Backpropagation has exactly the same structure using the generalized chain rule

$$z^i = \sum_{j \in \mathbf{ch}(i)} \underbrace{\left.\frac{\partial x^\omega}{\partial x^j}\right|_{x_I = a_I}}_{1 \times d^j} \times \underbrace{\left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j = a_I^j}}_{d^j \times d^i}$$

General Backpropagation

- Computation graph in which input values that are vectors

$$x^i \in \mathbb{R}^{d^i} \qquad \forall i \in V$$

  But the output value $x^\omega \in \mathbb{R}$ is always a scalar!

- The corresponding gradients are also vectors of the same size

$$z^i \in \mathbb{R}^{d^i} \qquad \forall i \in V$$

- Backpropagation has exactly the same structure using the generalized chain rule

$$z^i = \sum_{j \in \mathbf{ch}(i)} \underbrace{\left.\frac{\partial x^\omega}{\partial x^j}\right|_{x_I = a_I}}_{1 \times d^j} \times \underbrace{\left.\frac{\partial x^j}{\partial x^i}\right|_{x_I^j = a_I^j}}_{d^j \times d^i}$$

- For detail, read the note at: `http://karlstratos.com/notes/backprop.pdf`

Vector-Valued Functions and Jacobian

- View $f : \mathbb{R}^n \to \mathbb{R}^m$ simply as m scalar-valued functions $f_1 \dots f_m : \mathbb{R}^n \to \mathbb{R}$.

$$f(x) = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix} \qquad\qquad \forall x \in \mathbb{R}^n$$

- The **Jacobian** of $f : \mathbb{R}^n \to \mathbb{R}^m$ at $x = a$ is an $m \times n$ matrix

$$\left.\frac{\partial f(x)}{\partial x}\right|_{x=a} \in \mathbb{R}^{m \times n}$$

whose i-th row is $\nabla f_i(a) \in \mathbb{R}^n$

- Equivalently,

$$\left[\left.\frac{\partial f(x)}{\partial x}\right|_{x=a}\right]_{i,j} = \left.\frac{\partial f_j(x)}{\partial x_i}\right|_{x=a}$$