ANTONIS ANASTASOPOULOS

CS499 INTRODUCTION TO NLP

PRELIMINARIES

https://cs.gmu.edu/~antonis/course/cs499-spring21/
STRUCTURE OF THIS LECTURE

1. Probability Refresher
2. Working with text
3. Regular Expressions
4. Neural Nets Primer
PROBABILITIES
A random variable is a variable with a different random value in each “experiment”

Random Variable: X
P(X) is the distribution of X.

If \( x \in X \), we write \( P(X = x) \) for the probability that X has value x

\[
\sum_{x \in X} P(X = x) = 1
\]

If \( P(W) \) is the distribution of English words, we might have:

\( P(W = \text{the}) = 0.1 \)

\( P(W = \text{syzygy}) = 10^{-10}, \ldots \)
JOINT AND MARGINAL PROBABILITIES

Random Variables $W$ (words) and $S$ (speaker)

Joint distribution $P(S, W)$ such that:

$$\Sigma_{s,w} P(S = s, W = w) = 1$$

$$P(S = Trump, W = bigly) = 0.2$$
$$P(S = Trump, W = huge) = 0.4$$
$$P(S = Biden, W = people) = 0.3$$
$$P(S = Biden, W = fellas) = 0.1$$
Marginal distributions
\[ P(S = s) = \sum_w P(S = s, W = w) \]
\[ P(W = w) = \sum_s P(S = s, W = w) \]

For our made up numbers:
\[ P(S = \text{Trump}) = 0.2 + 0.4 = 0.6 \]
\[ P(S = \text{Biden}) = 0.3 + 0.1 = 0.4 \]
CONDITIONAL DISTRIBUTIONS

$$P(s \mid w) = \frac{P(s, w)}{P(w)}$$

Note that $$\Sigma_s P(s \mid w) = 1.$$  

You know this already, but do not confuse $$p(w \mid s)$$ and $$p(s \mid w):$$  

For our made up numbers:  

$$P(Trump \mid bigly) = \frac{0.2}{0.2} = 1$$  

$$P(bigly \mid Trump) = \frac{0.2}{0.6} \approx 0.33$$
$c_e(w)$: number of occurrences of letter e in a word

The expectation of $c_e(w)$ is

$$E[c_e] = \sum_w P(W = w)c_e(w)$$

For our made up numbers:

$$E[c_e] = 0.2 \cdot 0 + 0.4 \cdot 1 + 0.3 \cdot 2 + 0.1 \cdot 1 = 1.1$$
LOGARITHMS

Some identities that will be useful

\[
\begin{align*}
\log \exp x &= x \\
\log xy &= \log x + \log y \\
\log \prod_i x_i &= \sum_i \log x_i \\
\log x^n &= n \log x \\
\log 1 &= 0
\end{align*}
\]

\[
\begin{align*}
\exp \log x &= x \\
\exp(x + y) &= \exp x \exp y \\
\exp \sum_i x_i &= \prod_i \exp x_i \\
\exp nx &= (\exp x)^n \\
\exp 0 &= 1
\end{align*}
\]
LOGARITHMS

Used to simplify expressions like a product of probabilities:

\[ p(x_1, \ldots, x_n) = \Pi_i p(x_i) \]

Take the log of everything, and now you have a sum:

\[ \log p(x_1, \ldots, x_n) = \Sigma_i \log p(x_i) \]
LOGARITHMS

Used to simplify expressions like a product of probabilities:

\[ p(x_1, \ldots, x_n) = \Pi_i p(x_i) \]

Take the log of everything, and now you have a sum:

\[ \log p(x_1, \ldots, x_n) = \Sigma_i \log p(x_i) \]

For two probabilities \( p, q \) comparing \( \log p \) and \( \log q \) is equivalent.

Instead of multiplying two probabilities \( p \cdot q \) we can just add \( \log p + \log q \)
Let $x = [x_1, x_2, \ldots, x_n]$ be a vector of real numbers

We define $[\text{softmax } x]_i = \frac{\exp x_i}{\sum_{i'=1}^{n} \exp x_{i'}}$. 
WORKING WITH TEXT
A word is an ill-defined concept:

Type: a class of tokens that use the same character sequence

Token: an individual occurrence of a type in speech or writing

Vocabulary: the set of types

---

Lebensversicherungsgesellschaftsangestellter (life insurance company employee)

莎拉波娃现在居住在美国东南部的佛罗里达。 (Sharapova now lives in southeastern Florida)
A rose is a rose is a rose.

#Types: 4

Vocabulary: {a, rose, is, .}

#Tokens: 9
“Don’t think of an elephant!,” says George.

Elephants are not something you should be thinking, according to Lakoff.

Dr. Lakoff asks that you do not think of an elephant.
“Don’t think of an elephant!,” says George.

Elephants are not something you should be thinking, according to Lakoff.

Dr. Lakoff asks that you do not think of an elephant.
“do n’t think of an elephant !,” says George.
elephants are not something you should be thinking, according to Lakoff.
dr. Lakoff asks that you do not think of an elephant.

Tools:
- NLTK (https://www.nltk.org/)
- spacy (https://spacy.io/)
MORE READINGS


“NEURAL” NETS

Original Motivation: The Brain

Current Implementation: Computation Graphs

COMPOSITE FUNCTIONS

We will build computation graphs using an "ordered series of equations".

Each equation is only a function of the preceding equations

\[ f(x, y, z) = z + \sin(x^2 + y \times \exp(z)) \]

We can represent the above equation using intermediate variables:

\[
\begin{align*}
a &= x^2 \\
b &= \exp(z) \\
c &= y \times b \\
d &= a + c \\
e &= \sin(d) \\
f &= e + z
\end{align*}
\]
Main idea behind AD: as long as we have access to the derivatives of a set of primitives, then we can stick these together to get the derivative of any composite function.

Saving the values of intermediate variables (dynamic programming!) allows for low computational complexity (exponential $\rightarrow$ linear).
### GENERAL AUTODIFF FRAMEWORK

<table>
<thead>
<tr>
<th>Primitives</th>
<th>Their Derivatives</th>
<th>1. All edges (hyperedges) are made of primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(y, z) = y + z$</td>
<td>$\frac{\partial}{\partial x} f(y, z) = \frac{\partial y}{\partial x} + \frac{\partial z}{\partial x}$</td>
<td>2. Perform the forward pass, to compute the functions’ value</td>
</tr>
<tr>
<td>$f(y, z) = y \times z$</td>
<td>$\frac{\partial}{\partial x} f(y, z) = y \frac{\partial z}{\partial x} + z \frac{\partial y}{\partial x}$</td>
<td>3. Run back-propagation using the stored forward values, using the derivatives</td>
</tr>
<tr>
<td>$f(y, z) = y^3$</td>
<td>$\frac{\partial}{\partial x} f(y, z) = 3y^2 \frac{\partial y}{\partial x}$</td>
<td></td>
</tr>
<tr>
<td>$f(y, z) = \log(y)$</td>
<td>$\frac{\partial}{\partial x} f(y, z) = \frac{1}{y} \frac{\partial y}{\partial x}$</td>
<td></td>
</tr>
</tbody>
</table>
**EXAMPLE GRADIENT CALCULATION**

\[
f(x, y, z) = z + \sin(x^2 + y \times \exp(z))
\]

\[
a = x^2
\]

\[
\frac{\partial f}{\partial a} = \frac{\partial f}{\partial d} \frac{\partial d}{\partial a} + 1 = \frac{\partial f}{\partial d} \frac{\partial d}{\partial a} = \frac{\partial f}{\partial d} \frac{\partial d}{\partial a}
\]

\[
b = \exp(z)
\]

\[
\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial b} = \frac{\partial f}{\partial c} y
\]

\[
c = y \times b
\]

\[
\frac{\partial f}{\partial c} = \frac{\partial f}{\partial d} \frac{\partial d}{\partial c} = \frac{\partial f}{\partial d} 1 = \frac{\partial f}{\partial d} \frac{\partial d}{\partial a} = \frac{\partial f}{\partial d} 1
\]

\[
d = a + c
\]

\[
\frac{\partial f}{\partial d} = \frac{\partial f}{\partial e} \frac{\partial e}{\partial d} = \frac{\partial f}{\partial e} \cos(d)
\]

\[
e = \sin(d)
\]

\[
\frac{\partial f}{\partial e} = 1
\]

\[
f = e + z
\]

We can easily write the derivatives of individual terms in the graph.

Given all these, we can work backwards to compute the derivative of \(f(x, y, z)\) with respect to each variable.
A node is a \{tensor, matrix, vector, scalar\} value
A **node** with an incoming **edge** is a **function** of that edge’s tail node.

A **node** knows how to compute its value and the value of its derivative w.r.t each argument (edge) times a derivative of an arbitrary input \( \frac{\partial f}{\partial f(u)} \).

\[
\frac{\partial f(u)}{\partial u} \frac{\partial f}{\partial f(u)} = \left( \frac{\partial F}{\partial f(u)} \right)^T
\]

An **edge** represents a function argument (and also an data dependency). They are just pointers to nodes.
COMPUTATION GRAPHS

expression:
\[ x^T A \]

graph:

Functions can be nullary, unary, binary, ... n-ary. Often they are unary or binary.
expression: \[ x^T A x \]

graph:

Computation graphs are directed and acyclic.
expression: 
\[ x^T A x \]

graph:
expression:
\[ x^T A x + b \cdot x + c \]

graph:

\[ f(x_1, x_2, x_3) = \sum_i x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]
expression:
\[ y = x^\top A x + b \cdot x + c \]

graph:

\[ f(x_1, x_2, x_3) = \sum_i x_i \]
\[ f(M, v) = Mv \]
\[ f(U, V) = UV \]
\[ f(u) = u^\top \]
\[ f(u, v) = u \cdot v \]

variable names are just labelings of nodes.
$f(x_1, x_2, x_3) = \sum_i x_i$

$f(M, v) = M v$

$f(U, V) = U V$

$f(u) = u^T$

$f(u, v) = u \cdot v$

$\text{graph:}$
FORWARD PROPAGATION

graph:

\[ f(x_1, x_2, x_3) = \sum_{i} x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]
FORWARD PROPAGATION

graph:

\[ f(x_1, x_2, x_3) = \sum_i x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]

\[ b \]

\[ c \]
graph:

$f(x_1, x_2, x_3) = \sum_i x_i$

$f(M, v) = Mv$

$f(U, V) = UV$

$f(u) = u^T$

$f(u, v) = u \cdot v$
FORWARD PROPAGATION

graph:

\[ f(x_1, x_2, x_3) = \sum_i x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]
\[ f(x_1, x_2, x_3) = \sum x_i \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]

graph:

FORWARD PROPAGATION
FORWARD PROPAGATION

$$f(x_1, x_2, x_3) = \sum_i x_i$$

$$f(M, v) = Mv$$

$$f(U, V) = UV$$

$$f(u) = u^T$$

$$f(u, v) = u \cdot v$$

graph:
FORWARD PROPAGATION

graph:

\[ f(x_1, x_2, x_3) = \sum x_i \]

\[ x^T Ax + b \cdot x + c \]

\[ f(M, v) = Mv \]

\[ f(U, V) = UV \]

\[ f(u) = u^T \]

\[ f(u, v) = u \cdot v \]

\[ b \cdot x \]

\[ x \]

\[ x^T \]

\[ A \]

\[ A \]

\[ b \]

\[ c \]
Back-propagation:

Process examples in reverse topological order

Calculate the derivatives of the parameters with respect to the final value
(This is usually a “loss function”, a value we want to minimize)

Parameter update:

Move the parameters in the direction of this derivative

\[ W = \alpha \times \frac{dl}{dW} \]
Examples in this class will be in DyNet or PyTorch:

- **intuitive**, program like you think (c.f. TensorFlow, Theano)

- **fast for complicated networks** on CPU (c.f. autodiff libraries, Chainer, PyTorch)

- **has nice features** to make efficient implementation easier (automatic batching)
Create a model

For each example

create a graph that represents the computation you want

calculate the result of that computation

if training, perform back propagation and update
The building blocks of words

Lexicons

Edit Distance
ASSIGNMENT 1 OUT!