CS-568 Deep Learning

Nazar Khan

PUCIT

Automatic Differentiation

Automatic Differentiation (AD)

- Set of techniques to numerically evaluate the derivative of a function *specified by a computer program*.
- Analytic or symbolic differentiation evaluates the derivative of a function *specified by a math expression*.
- AD Also called *algorithmic differentiation* or *computational differentiation*.
- Backpropagation is a special case of AD.

> Modern machine learning frameworks (TensorFlow, Theano, PyTorch) employ AD. The programmer only needs to implement the forward pass up to the loss function. Derivatives are handled automatically!

Automatic Differentiation

AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, derivatives of arbitrary order can be computed automatically, accurately to working precision, and using
k(Op) *at most a small constant factor more arithmetic operations than the original program.*

https://en.wikipedia.org/wiki/Automatic_differentiation

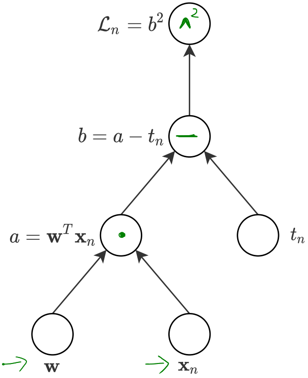
Linear Regression via Automatic Differentiation

$y_n = w^T x_n$

- Consider the squared loss function for linear regression.

$$L_n(\mathbf{w}) = \left(\overbrace{\mathbf{w}^T\mathbf{x}_n}^{y_n} - t_n\right)^2$$

- Can be represented as a computational graph consisting of *elementary operations*.



Linear Regression via Automatic Differentiation

$L = (w^T x_n - t_n)^2$ $\boxed{\nabla_w L = 2(w^T x_n - t_n)\, x_n^T}$

No need anymore

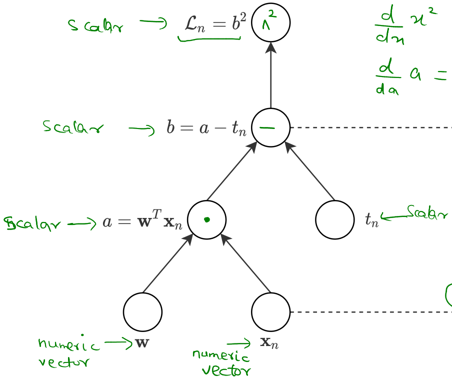
$\left[\frac{\partial L}{\partial w_1} \; \frac{\partial L}{\partial w_2} \; \cdots \; \frac{\partial L}{\partial w_D}\right]$

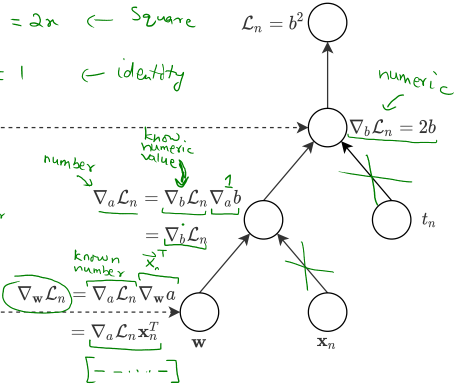
- For training, we are interested in the gradient $\nabla_{\mathbf{w}} L_n$.
- After the forward pass for a particular $\mathbf{w}$ and $\mathbf{x}_n$, gradients can be evaluated numerically.

$\frac{d}{d\vec{w}} \vec{w}^T \vec{x}_n = \vec{x}_n^T$ ← Dot-prod.

$\frac{d}{dx} x^2 = 2x$ ← Square

$\frac{d}{da} a = 1$ ← identity





AD in Python

- A Python package called *Autograd* implements *reverse mode* automatic differentiation.
- Elementary operations such as $+, \sin, x^k$ *etc.* are *overloaded* by also computing their derivates $1, \cos, kx^{k-1}$ *etc.*.
- If required, more sophisticated user-defined functions and their derivative implementations can be *registered* with Autograd.

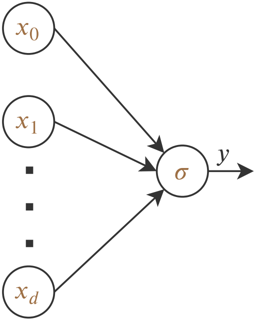
elementary functions $\begin{cases} \sigma(a) = \dfrac{1}{1+e^{-a}} \\ \sigma'(a) = \sigma(1-\sigma) \end{cases}$

Logistic Regression via Automatic Differentiation

Binary classifier with no hidden layer

Just a perceptron with logistic sigmoid activation function. Models probability of class 1 instead of decision.

$$y = p(\mathcal{C}_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$$
$$1 - y = p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x})$$



Binary cross-entropy loss

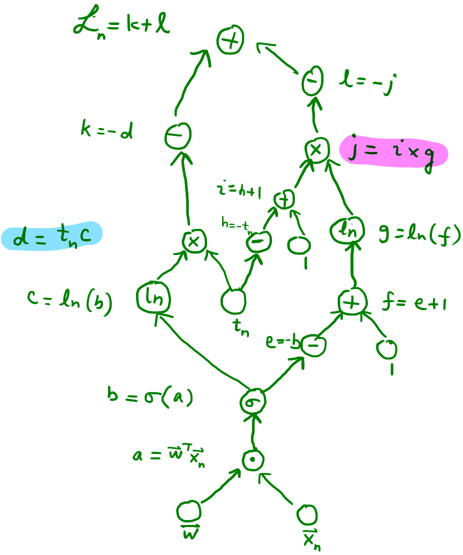
$$\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N} t_n \ln y_n + (1-t_n)\ln(1-y_n)$$

$\mathcal{L}_n(w) = -t_n \ln y_n - (1-t_n)\ln(1-y_n)$ where $y_n = \sigma(\vec{w}^T\vec{x}_n)$

Logistic Regression via Automatic Differentiation

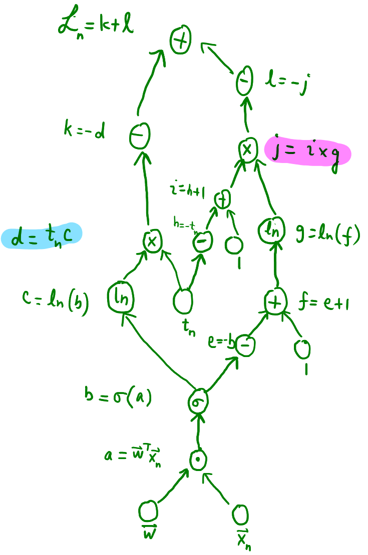
Step 1: Computational Graph for $\mathcal{L}_n$

$\mathcal{L}_n(w) = -t_n \ln y_n - (1-t_n)\ln(1-y_n)$ where $y_n = \sigma(\vec{w}^T \vec{x}_n)$



Logistic Regression via Automatic Differentiation

Step 2: AD till $\nabla_{\mathbf{w}}\mathcal{L}_n$



$\nabla_{\vec{w}} \mathcal{L}_n$?