Automatic differentiation

JJV

Oct 7, 2022

Optimization

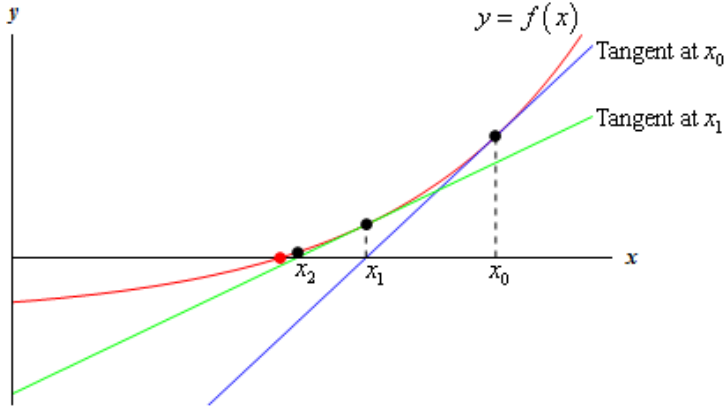
Find “the best” parameters to reach a goal

Usually, minimize a differentiable loss function

Find the zeroes of another function (its derivative)

How to find the zeroes of a function?

Newton's method: find x such that $f(x) = 0$



$f : \mathbf{R} \to \mathbf{R}$ differentiable, $\nexists x, f'(x) = 0$

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

Quadratic convergence

$$\exists C > 0, |x_{t+1} - \ell| \leq C|x_t - \ell|^2$$

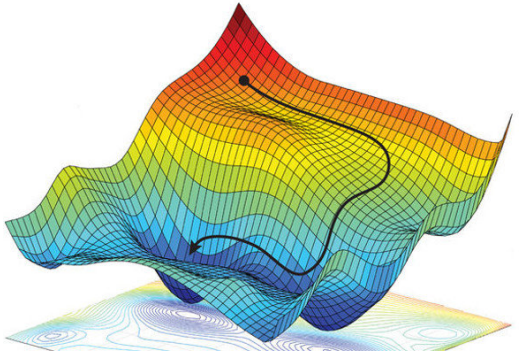
In higher dimension

Let $g : \mathbf{R}^n \to \mathbf{R}$ twice differentiable, with $n >> 1$

What is the size of $g'(\boldsymbol{x})$? Usually noted $\frac{\partial g}{\partial \boldsymbol{x}}$ or $\nabla_{\boldsymbol{x}} g$.

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \underbrace{g''(\boldsymbol{x}_t)^{-1}}_{\in \mathbf{R}^{n \times n},\ O(n^3)} \underbrace{g'(\boldsymbol{x}_t)}_{\in \mathbf{R}^n}$$

Gradient descent



$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}_t)$$

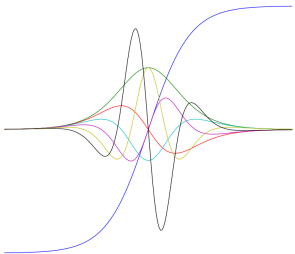
```
for each epoch:
    for x, y in dataset:
        compute gradients ∂L/∂θ (f_θ(x), y)   # also noted ∇_θ L
        θ ← θ − γ ∇_θ L   # γ is the learning rate
```

SGD in PyTorch

```
import torch

# define model, n_epochs, trainloader
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)

for _ in range(n_epochs):
    for batch_inputs, batch_labels in trainloader:
        outputs = model(batch_inputs)
        loss = criterion(outputs, batch_labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

How to compute gradients automatically?

```
>>> from autograd import elementwise_grad as egrad  # vectorize over inputs
>>> import matplotlib.pyplot as plt
>>> x = np.linspace(-7, 7, 200)
>>> plt.plot(x, tanh(x),
...          x, egrad(tanh)(x),                                  # first  derivative
...          x, egrad(egrad(tanh))(x),                           # second derivative
...          x, egrad(egrad(egrad(tanh)))(x),                    # third  derivative
...          x, egrad(egrad(egrad(egrad(tanh))))(x),             # fourth derivative
...          x, egrad(egrad(egrad(egrad(egrad(tanh)))))(x),      # fifth  derivative
...          x, egrad(egrad(egrad(egrad(egrad(egrad(tanh))))))(x))  # sixth  derivative
>>> plt.show()
```



Existing methods and their limitations

Numerical differentiation

$$\frac{f(x+h) - f(x)}{h}$$

Round-off errors

Symbolic differentiation

Have to keep symbolic expressions at each step of the process

Automatic differentiation

Chain rule

$$(f \circ g)' = g' \cdot (f' \circ g)$$

Generalized chain rule

$$\frac{df_1}{dx} = \frac{df_1}{df_2}\frac{df_2}{df_3} \cdots \frac{df_n}{dx}$$

Reminder: Jacobians

Consider differentiable $f : \mathbf{R}^n \to \mathbf{R}^m$, its Jacobian contains its first-order partial derivatives $(J_f)_{ij} = \frac{\partial f_i}{\partial x_j}$:

$$J_f = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Example: linear layer

W is a $m \times n$ matrix
If $f(\boldsymbol{x}) = W\boldsymbol{x}$
$f_i = W_i\boldsymbol{x} = \sum_j W_{ij}x_j$ where W_i is ith row of W
$(J_f)_{ij} = \frac{\partial f_i}{\partial x_j} = W_{ij}$ so $J_f = W$

Generalized multivariate chain rule

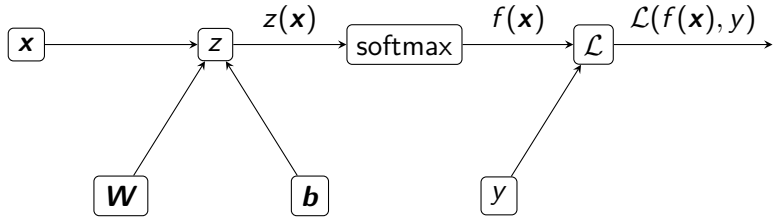
Consider differentiable $f : \mathbf{R}^m \to \mathbf{R}^k, g : \mathbf{R}^n \to \mathbf{R}^m$ and $\boldsymbol{a} \in \mathbf{R}^n$.

$$D_{\boldsymbol{a}}(f \circ g) = D_{g(\boldsymbol{a})} f \circ D_{\boldsymbol{a}} g$$

So the Jacobians verify:

$$J_{f \circ g} = (J_f \circ g) J_g$$

Computation graph



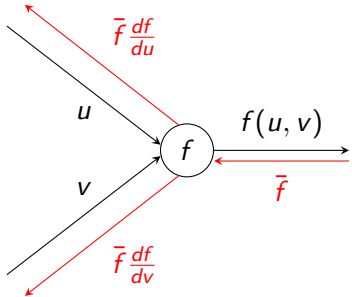
$$\frac{d\mathcal{L}}{db_1} = \frac{d\mathcal{L}}{df}\frac{df}{db_1} = \frac{d\mathcal{L}}{df}\left(\frac{df}{dz}\frac{dz}{db_1}\right) \text{ (forward)}$$
$$= \frac{d\mathcal{L}}{dz}\frac{dz}{db_1} = \left(\frac{d\mathcal{L}}{df}\frac{df}{dz}\right)\frac{dz}{db_1} \text{ (backward)}$$

Properly written: $J_{\mathcal{L}\circ\text{softmax}\circ z} = (J_{\mathcal{L}} \circ f)(J_{\text{softmax}} \circ z)J_z$

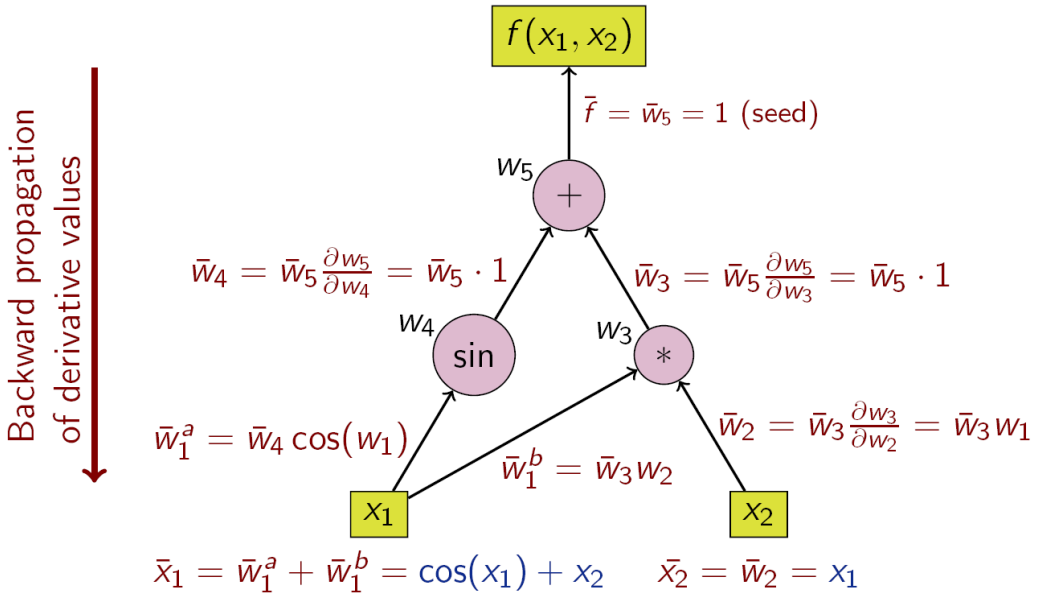
Given that $\boldsymbol{x} \in \mathbf{R}^d, z(\boldsymbol{x}), f(\boldsymbol{x}) \in \mathbf{R}^{d_2}, \mathcal{L}(f(\boldsymbol{x}), y) \in \mathbf{R}$,
which order is better?

Reverse accumulation (in $\mathbf{R}$)

Let us note the adjoint $\overline{f} \triangleq \frac{d\mathcal{L}}{df}$.

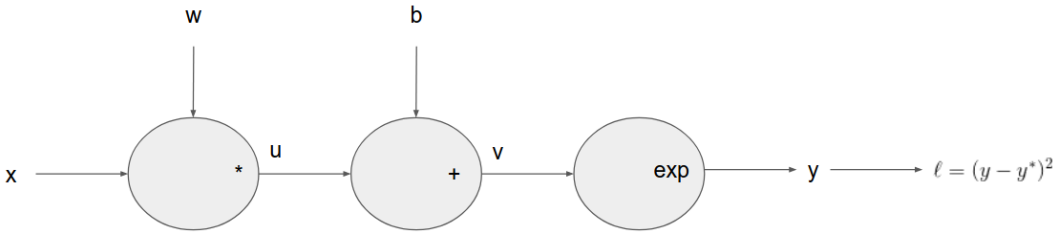


A complete example (Wikipedia)



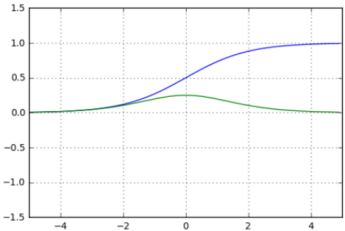
Please compute gradients

FORWARD PASS



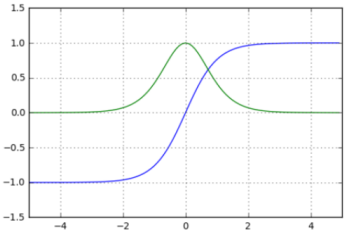
$u = wx$
$v = u + b$
$y = e^v$
$\ell = (y - y^*)^2$

Examples of link functions (Ollion & Grisel)



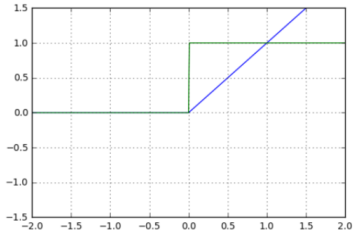
$$\text{sigm}(x) = \frac{1}{1+e^{-x}}$$

$$\text{sigm}'(x) = \text{sigm}(x)(1-\text{sigm}(x))$$



$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

$$\tanh'(x) = 1 - \tanh(x)^2$$



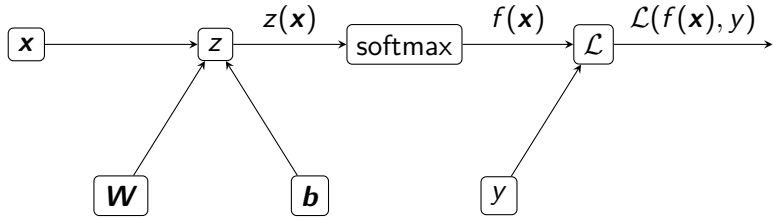
$$\text{relu}(x) = \max(0, x)$$

$$\text{relu}'(x) = 1_{x>0}$$

More interesting link functions

	Binary	Multiclass
f	softplus : $x \mapsto \log(1 + \exp(x))$	$\text{logsumexp}^+ : \boldsymbol{x} \mapsto \log(1 + \sum_c \exp(x_c))$
f'	sigmoid : $x \mapsto 1/(1 + \exp(-x))$	$\text{softmax} : \boldsymbol{x} \mapsto \exp(\boldsymbol{x}) / \sum_c \exp(x_c)$
f''	$x \mapsto \text{sigmoid}(x)(1 - \text{sigmoid}(x))$	$?\boldsymbol{x} \mapsto \text{diag}(s(\boldsymbol{x})) - s(\boldsymbol{x})s(\boldsymbol{x})^T$

Computation graph: classifier



Here we define $\mathcal{L}$ as cross-entropy:

$$\mathcal{L}(f(\boldsymbol{x}), y) = -\sum_{c=1}^{K} \mathbf{1}_{y=c} \log f(\boldsymbol{x})_c = -\log f(\boldsymbol{x})_y$$

Compute $\frac{d\mathcal{L}}{dz_c}$