BACKPROPAGATION

September 9, 2020

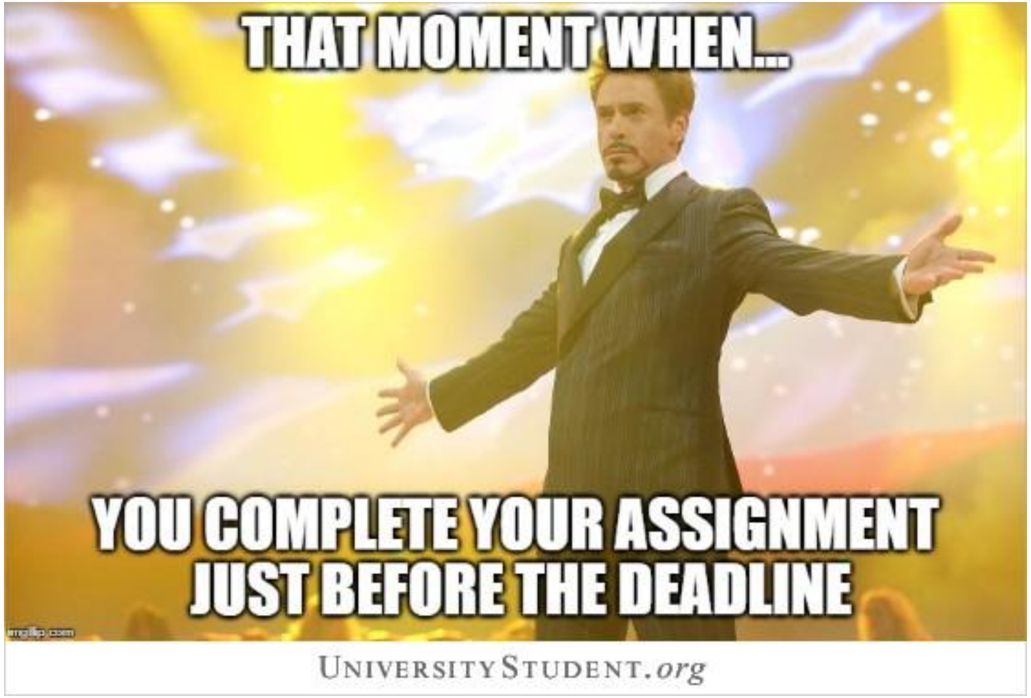
Nikitin Filipp

Введение

Оценивание

~5 практических заданий:

- Любой плагиат - обнуление
- Задания принимаются строго до дедлайна
- ~50% баллов - 3/10
- ~90% на 10/10



[Страница курса]

Семинары

Цель:

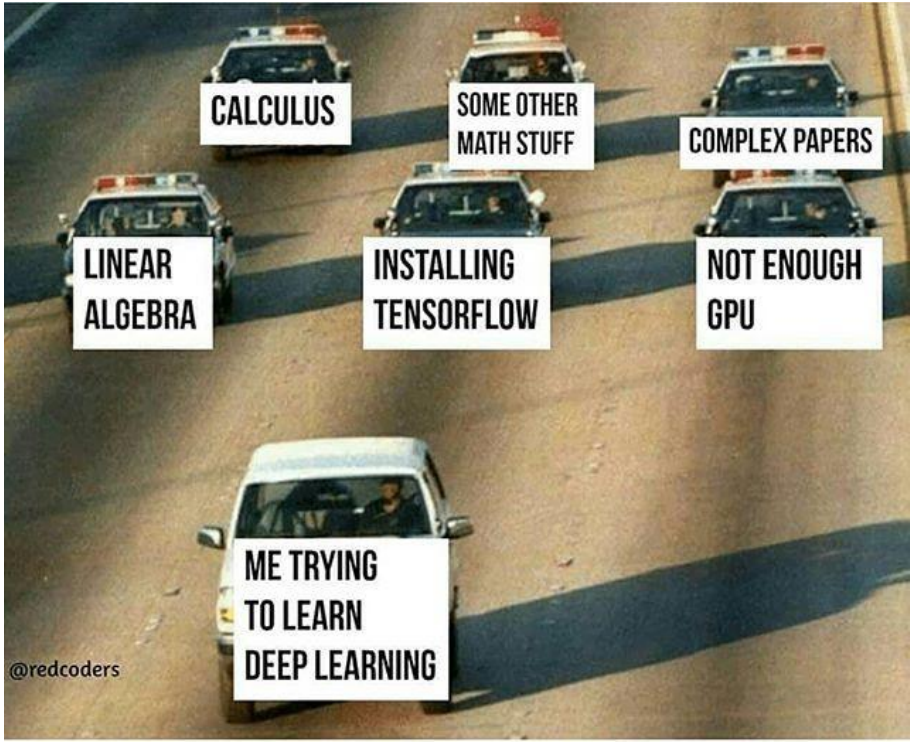
- Освоить DL на практике
- Обсудить подводные камни, хаки
- Консультация по имеющимся вопросам

Формат:

- Оффлайн (надеемся и ждём)
- Пока онлайн

Требуется:

- Ноутбук, зарядка, желание программировать



Градиентный спуск

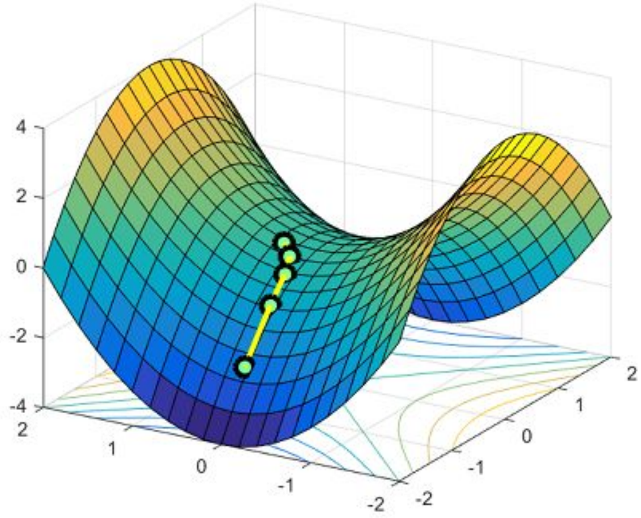
Методы оптимизации

Методы оптимизации:

- Случайный поиск
- Градиентные методы
- Методы 2-го, 3-го порядка
- Методы дискретной оптимизации

Вопросы:

Какие методы вам знакомы?
Почему в глубинном обучение проблематично использовать методы высоких порядков?
Известны ли вам модификации градиентного спуска?
Как можно вычислить градиент функции?



Численное дифференцирование

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Преимущества:

- Простота
- Варьируемая точность

Недостатки:

- Медленно
- Неточно

Аналитическое дифференцирование

$\mathbf{y}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$
$\mathbf{Ax}$	$\mathbf{A}^T$
$\mathbf{x}^T\mathbf{A}$	$\mathbf{A}$
$\mathbf{x}^T\mathbf{x}$	$2\mathbf{x}$
$\mathbf{x}^T\mathbf{Ax}$	$\mathbf{Ax} + \mathbf{A}^T\mathbf{x}$

Преимущества:

- Точность
- Скорость

Недостатки:

- Легко допустить ошибку
- Неприменимо для больших сетей (CNN, Transformer, Neural Turing Machine)

Автоматическое дифференцирование

Идея

Пусть функция есть суперпозиция нескольких функций:

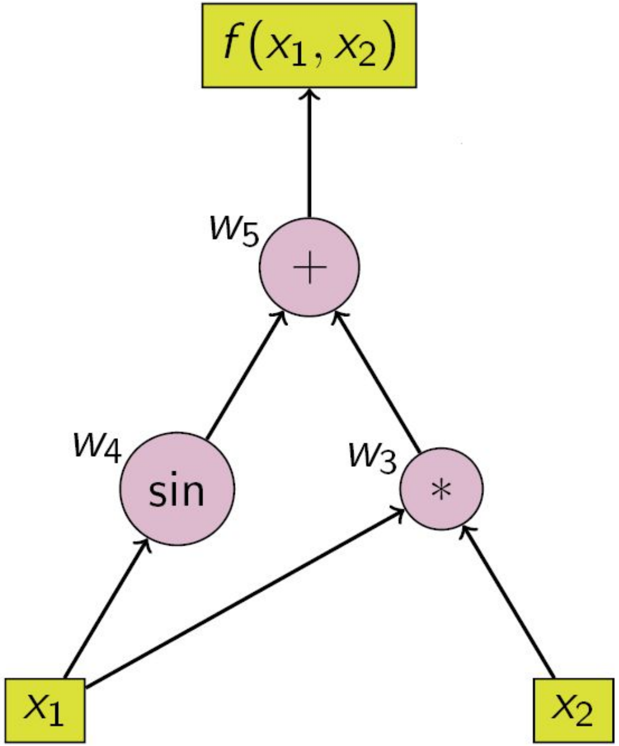
$$y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$$

$$\begin{aligned} w_0 &= x \\ w_1 &= h(w_0) \\ w_2 &= g(w_1) \\ w_3 &= f(w_2) = y \end{aligned}$$

Можем записать цепное правило:

$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx} = \frac{df(w_2)}{dw_2}\frac{dg(w_1)}{dw_1}\frac{dh(w_0)}{dx}$$

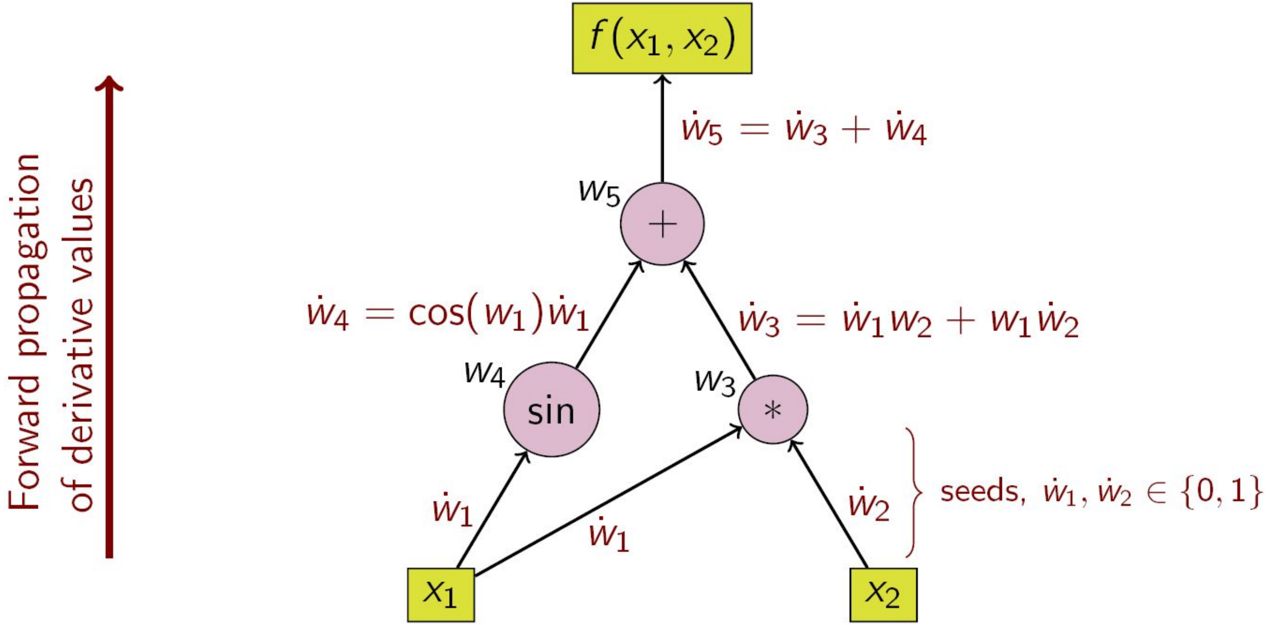
Вычислительный граф



Вершины соответствуют вычислительным операциям,
листья - переменным,
ребра задают определенную последовательность операций

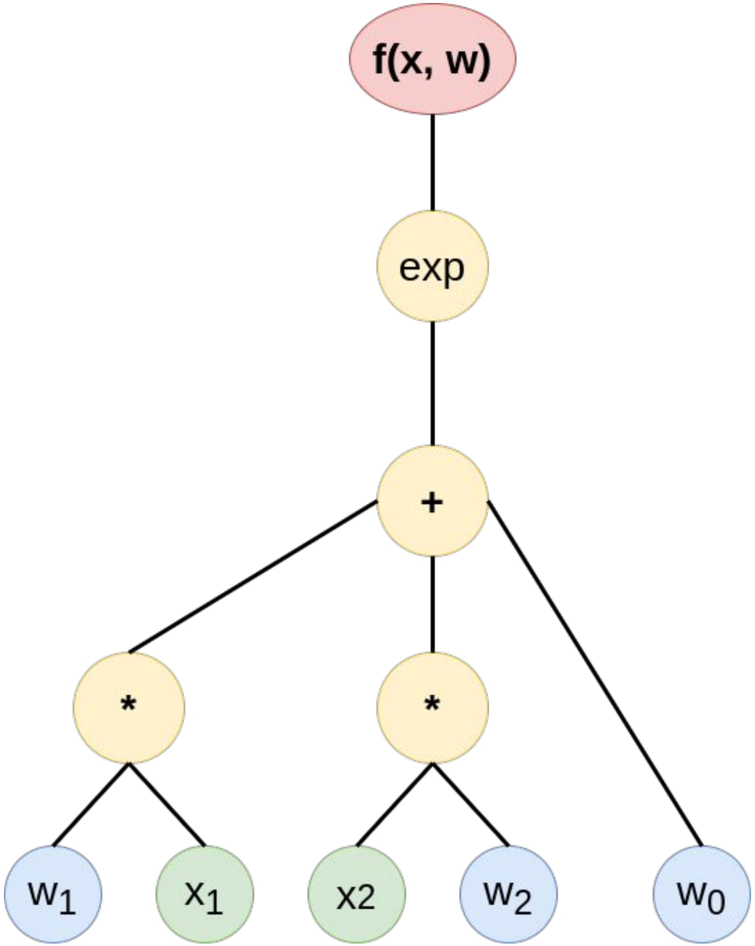
Forward mode

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_{n-1}} \frac{\partial w_{n-1}}{\partial x} = \frac{\partial y}{\partial w_{n-1}} \left(\frac{\partial w_{n-1}}{\partial w_{n-2}} \frac{\partial w_{n-2}}{\partial x} \right) = \frac{\partial y}{\partial w_{n-1}} \left(\frac{\partial w_{n-1}}{\partial w_{n-2}} \left(\frac{\partial w_{n-2}}{\partial w_{n-3}} \frac{\partial w_{n-3}}{\partial x} \right) \right) = \cdots$$



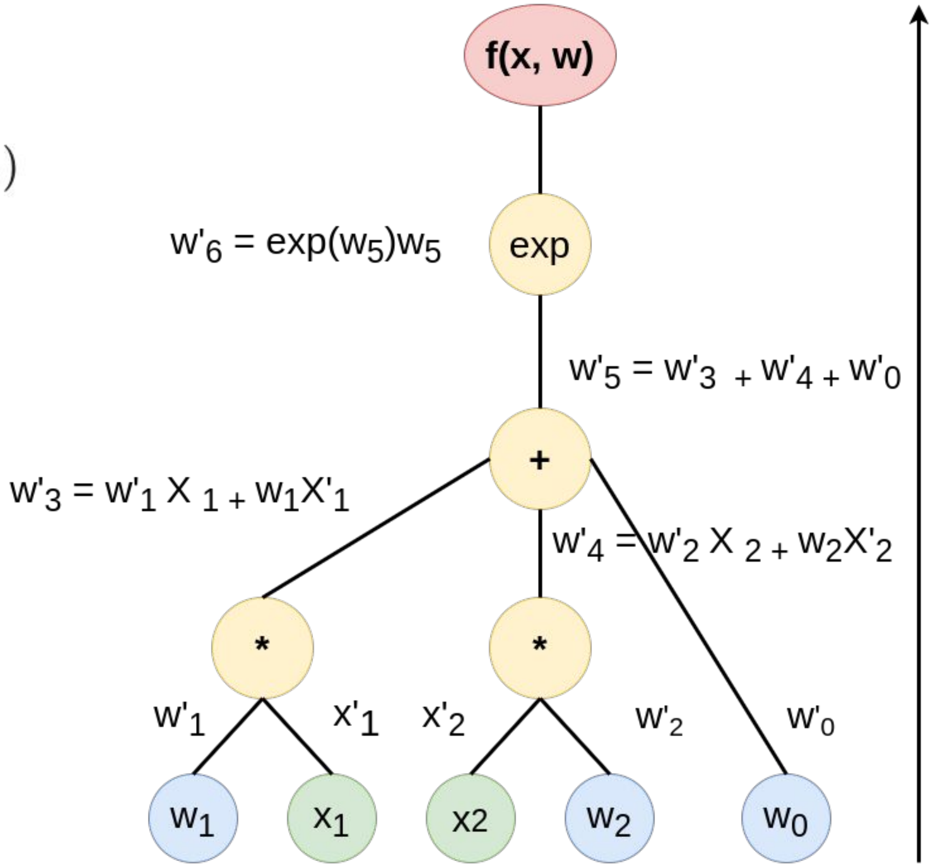
Forward mode Задача

$$f(\mathbf{x}, \mathbf{w}) = e^{(w_1 x_1 + w_2 x_2 + w_0)}$$



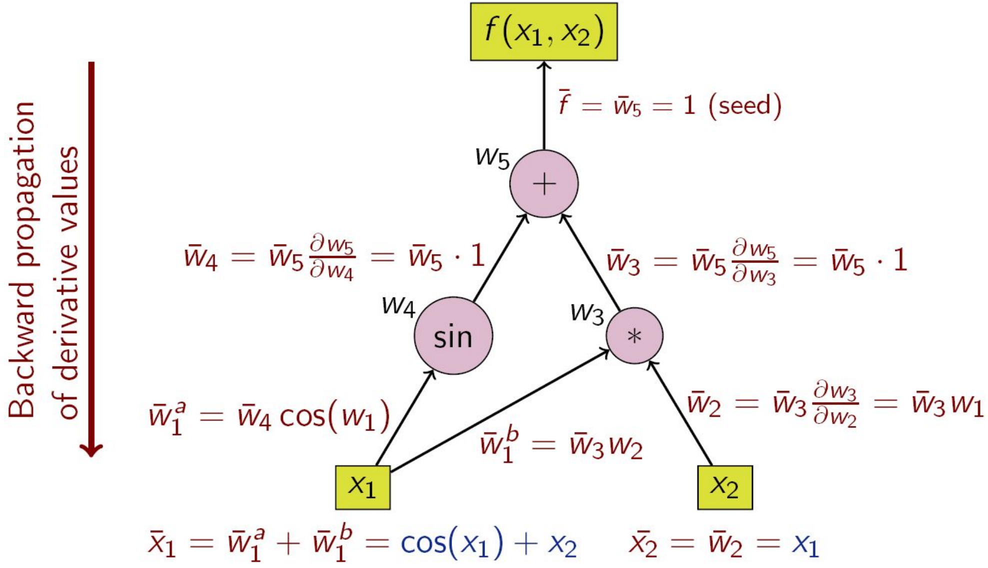
Forward mode Решение

$$f(\mathbf{x}, \mathbf{w}) = e^{(w_1 x_1 + w_2 x_2 + w_0)}$$



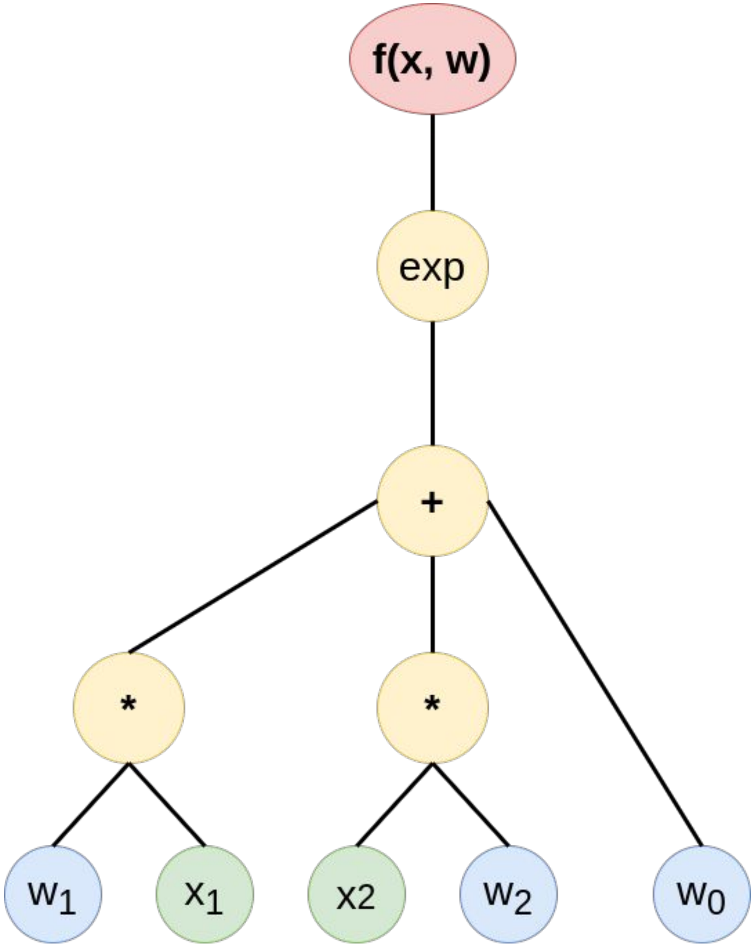
Reverse mode

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \left(\left(\frac{\partial y}{\partial w_3}\frac{\partial w_3}{\partial w_2}\right)\frac{\partial w_2}{\partial w_1}\right)\frac{\partial w_1}{\partial x} = \cdots$$



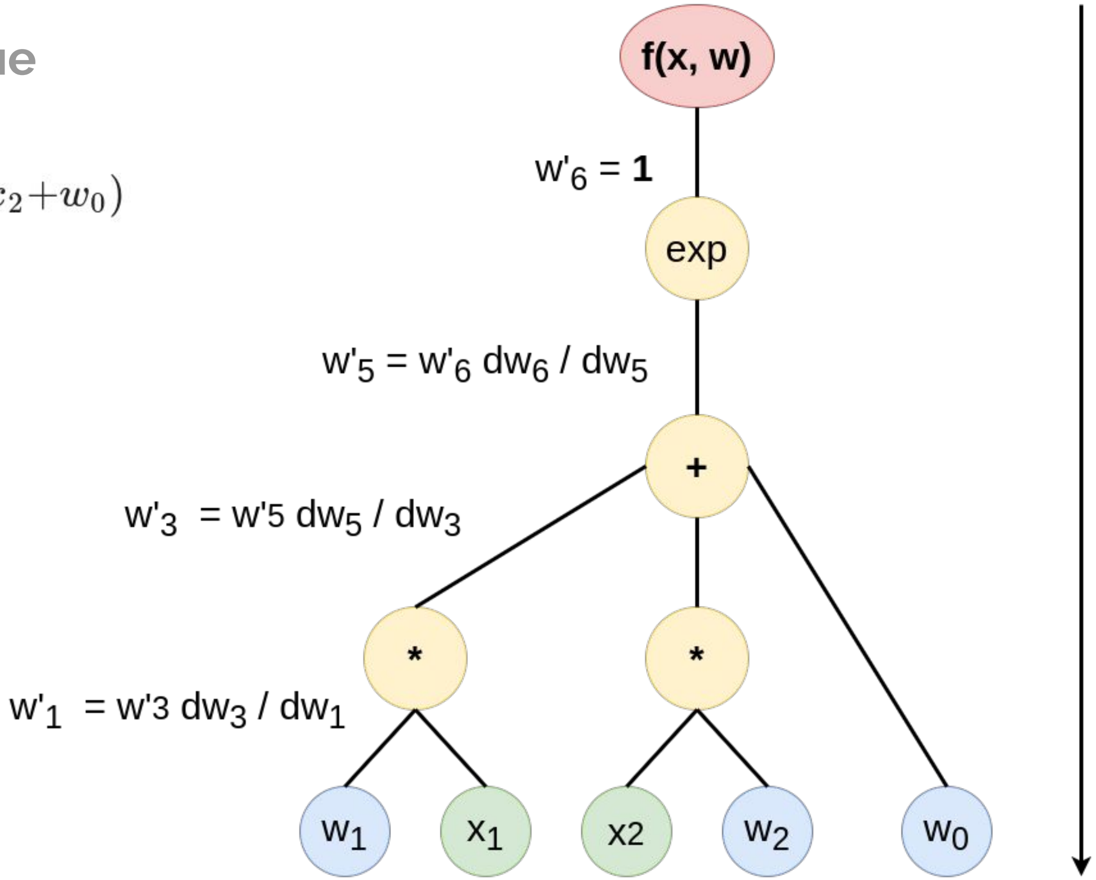
Reverse mode Задача

$$f(\mathbf{x}, \mathbf{w}) = e^{(w_1 x_1 + w_2 x_2 + w_0)}$$

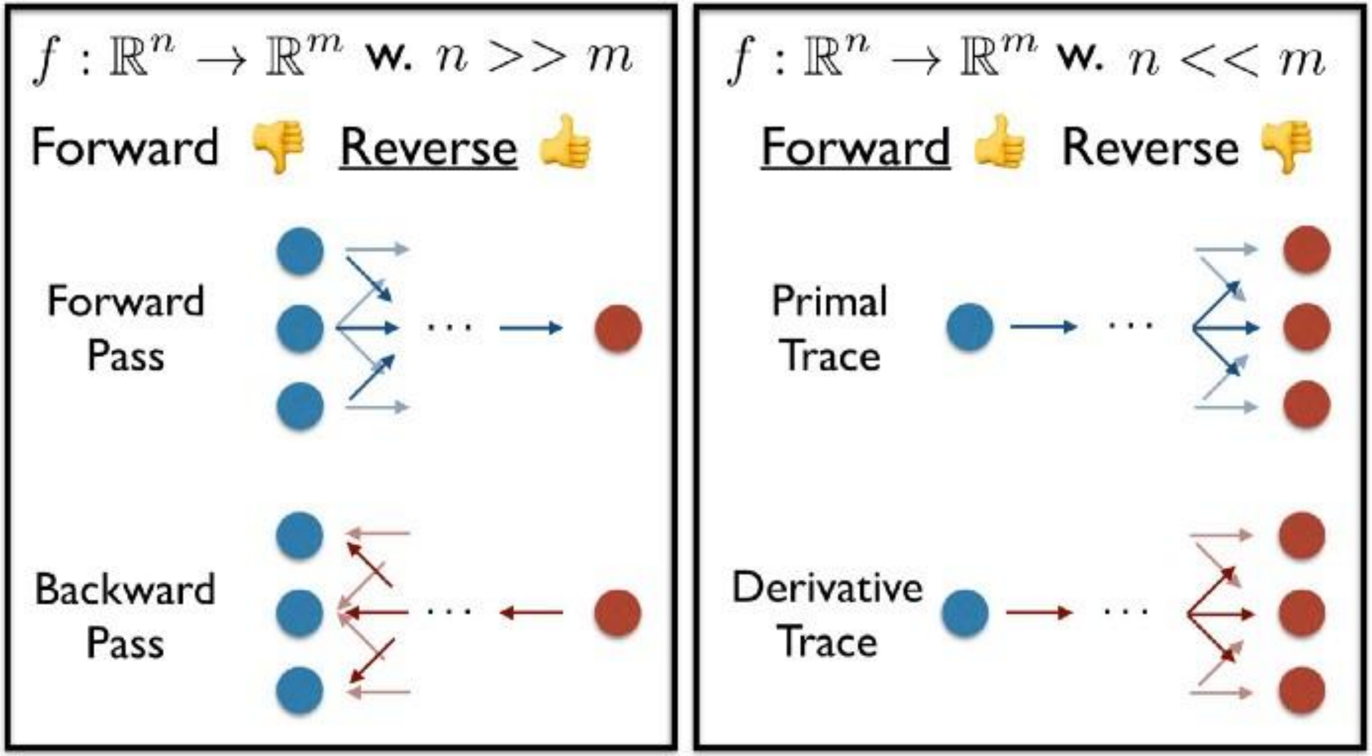


Reverse mode Решение

$$f(\mathbf{x}, \mathbf{w}) = e^{(w_1 x_1 + w_2 x_2 + w_0)}$$



Сравнение



$f: \mathbb{R}^n \to \mathbb{R}^m$ w. $n >> m$
Forward
Reverse
Forward Pass
Backward Pass
$f: \mathbb{R}^n \to \mathbb{R}^m$ w. $n << m$
Forward
Reverse
Primal Trace
Derivative Trace

Задачи

Тензорное дифференцирование

$$
\begin{aligned}
\partial \mathbf{A} &= 0 \\
\partial(\alpha \mathbf{X}) &= \alpha \partial \mathbf{X} \\
\partial(\mathbf{X}+\mathbf{Y}) &= \partial \mathbf{X} + \partial \mathbf{Y} \\
\partial(\mathrm{Tr}(\mathbf{X})) &= \mathrm{Tr}(\partial \mathbf{X}) \\
\partial(\mathbf{X}\mathbf{Y}) &= (\partial \mathbf{X})\mathbf{Y} + \mathbf{X}(\partial \mathbf{Y}) \\
\partial(\mathbf{X} \circ \mathbf{Y}) &= (\partial \mathbf{X}) \circ \mathbf{Y} + \mathbf{X} \circ (\partial \mathbf{Y}) \\
\partial(\mathbf{X} \otimes \mathbf{Y}) &= (\partial \mathbf{X}) \otimes \mathbf{Y} + \mathbf{X} \otimes (\partial \mathbf{Y}) \\
\partial(\mathbf{X}^{-1}) &= -\mathbf{X}^{-1}(\partial \mathbf{X})\mathbf{X}^{-1} \\
\partial(\det(\mathbf{X})) &= \mathrm{Tr}(\mathrm{adj}(\mathbf{X})\partial \mathbf{X}) \\
\partial(\det(\mathbf{X})) &= \det(\mathbf{X})\mathrm{Tr}(\mathbf{X}^{-1}\partial \mathbf{X}) \\
\partial(\ln(\det(\mathbf{X}))) &= \mathrm{Tr}(\mathbf{X}^{-1}\partial \mathbf{X}) \\
\partial \mathbf{X}^T &= (\partial \mathbf{X})^T \\
\partial \mathbf{X}^H &= (\partial \mathbf{X})^H
\end{aligned}
$$

Почему плохо использовать поэлементное взятие производной?

Что такое “градиент” вектор-функции?

Размерность производной

1. Градиент (производная скалярной функции):

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} \cdots \frac{\partial y}{\partial x_n} \end{bmatrix}$$

2. Матрица Якоби (производная векторной функции):

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

3. Производная скаляра по матрице:

$$\frac{\partial y}{\partial A} = \begin{bmatrix} \frac{\partial y}{\partial A_{11}} & \frac{\partial y}{\partial A_{12}} & \cdots & \frac{\partial y}{\partial A_{1n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial A_{m1}} & \frac{\partial y}{\partial A_{m2}} & \cdots & \frac{\partial y}{\partial A_{mn}} \end{bmatrix}$$

Размерность производной векторной функции от матрицы?

Размерность производной

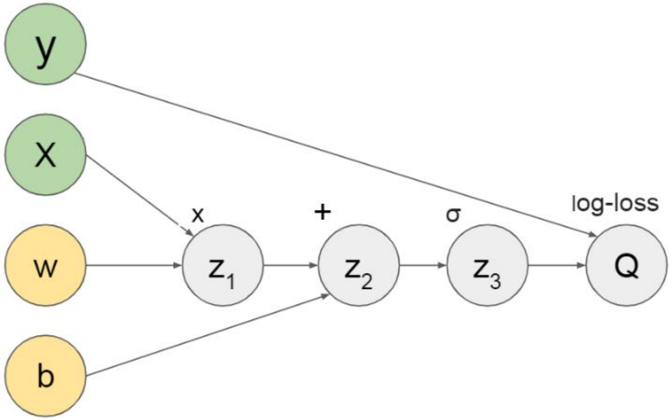
Утверждение

:

Пусть $\mathbf{z} = \mathbf{Ax}$, тогда $\dfrac{\partial \mathbf{z}}{\partial A_{ij}}$ есть вектор с одним ненулевым элементом на позиции i со значением x_j !

Предлагается доказать данное утверждение :)

Логистическая регрессия



$$z_1 = Xw$$

$$z_2 = z_1 + b\bar{1}$$

$$z_3 = \sigma(z_2)$$

$$Q = -\sum_{i=1}^{N}(y_i \log z_{3,i} + (1 - y_i)\log(1 - z_{3,i}))$$

Найти: $\dfrac{\partial Q}{\partial w}$ $\dfrac{\partial Q}{\partial b}$?

Логистическая регрессия Решение

$$\frac{\partial Q}{\partial z_3} = -\frac{y}{z_3} + \frac{1-y}{1-z_3}$$

$$\frac{\partial Q}{\partial z_2} = \frac{\partial Q}{\partial z_3} \odot \sigma(z_2) \odot (1-\sigma(z_2)))$$

$$\frac{\partial Q}{\partial z_1} = \frac{\partial Q}{\partial z_2}$$

$$\frac{\partial Q}{\partial b} = \frac{\partial Q}{\partial z_2}\bar{1}$$

$$\frac{\partial Q}{\partial w} = \frac{\partial Q}{\partial z_1}X$$

Линейное преобразование

y = Wx,
R^n -> R^m

dy/dW, dy/dx?

Нелинейность

y = sigmoid(x),
R^n -> R^n

y = ReLU(x),
R^n -> R^n

dy/dx?

Бонус: переопределение backward в Pytorch

torch.nn.Module

```
class Linear(nn.Module):
    def __init__(self, input_features, output_features, bias=True):
        super(Linear, self).__init__()
        self.weight = nn.Parameter(torch.Tensor(output_features, input_features))
        self.bias = nn.Parameter(torch.Tensor(output_features))

    def forward(self, input):
        return self.weight.mm(input) + self.bias
```

torch.autograd.Function

```
class LinearFunction(Function):
    @staticmethod
    def forward(ctx, input, weight, bias):
        ctx.save_for_backward(input, weight, bias)
        output = input.mm(weight.t())
        output += bias.unsqueeze(0).expand_as(output)
        return output

    @staticmethod
    def backward(ctx, grad_output):
        input, weight, bias = ctx.saved_tensors
        grad_input = grad_weight = grad_bias = None
        Z
        grad_input = grad_output.mm(weight)
        grad_weight = grad_output.t().mm(input)
        grad_bias = grad_output.sum(0)

        return grad_input, grad_weight, grad_bias


class Linear(nn.Module):
    def __init__(self, input_features, output_features, bias=True):
        super(Linear, self).__init__()
        self.weight = nn.Parameter(torch.Tensor(output_features, input_features))
        self.bias = nn.Parameter(torch.Tensor(output_features))

    def forward(self, input):
        return LinearFunction.apply(input, self.weight, self.bias)
```

Литература

Литература

1. cs231n.stanford.edu/vecDerivs.pdf
2. www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf
3. http://cs231n.stanford.edu/slides/2018/cs231n_2018_ds02.pdf
4. https://cs231n.github.io/optimization-2/
5. https://arxiv.org/pdf/1502.05767.pdf
6. https://en.wikipedia.org/wiki/Automatic_differentiation
7. https://towardsdatascience.com/forward-mode-automatic-differentiation-dual-numbers-8f47351064bf

Спасибо!