# Neural Networks 201

Deep Learning for Natural Language Processing

Vladislav Lialin, Text Machine Lab

# Administrative

- Python code style quiz is due Monday Feb 13 before the class
- Homework 3 posted today and due on Thursday Feb 16 before the class
    - Significantly harder than previous homeworks
    - It will likely take more than two days to solve it
    - Any cheating will be prosecuted
    - Homework 3 will be peer-review graded
        - You will receive grading sheet
        - Each of you will grade two homeworks of your fellow students selected randomly and write a short feedback paragraph

# Administrative: peer review

- On Feb 16 you will receive a grading sheet for homework 3
- Each of you will grade two homeworks of your fellow students selected randomly and write a short homework feedback paragraph
- HW3 peer reviews are due on Monday Feb 20 until the end of the day

for this homework you can get up to 23 raw points that are rescaled to 10

Inline question 4.1: we add BOS token to the beginning of each sequence so that the network would learn that every single translation starts with BOS and during test we knew what to input to the decoder at the very first decoding step when we don't have any words translated. EOS serves as an indicator of the end of translation so that at test time we could stop generating tokens for this sentence when we see this token.
-0.5 point
Inline question 4.2: we need to shift labels by one token to the left so that decoder predicts future tokens and not current tokens. E.g., for non-shifted decoder inputs the task would look like this: given tokens A B C predict token C. For shifted tokens now it looks like for given tokens A B predict token C. Without shifting the task becomes trivial — it it not "predict the next translated word" it is "copy the last word from the decoder input". -0.5 point
Inline question 4.3: .forward is the method implementing forward pass of the network. During training we use it to compute loss and input all decoder input ids into it (teacher forcing). During test time, it is not possible to know decoder_input_ids in advance as we don't know the true translation and instead we generate translation one by one. -1 point
+1 point as we allow for one inline question mistake

Total: 20 / 23 -> 8.7

# Administrative

- Python code style quiz is due on **Monday Feb 13** before the class
- Homework 3 to be posted today and due on **Thursday Feb 16** before the class
- Feb 11 office hours are extended:
  - 11am - 1pm, DAN 415
  - Please come early
  - Note that office hours end one hour before the class
- Homework 3 will be peer-reviewed
- HW1 and HW2 grades will be posted next week

# What you should remember after this lecture

- Backpropagation
- General definition of a neural network

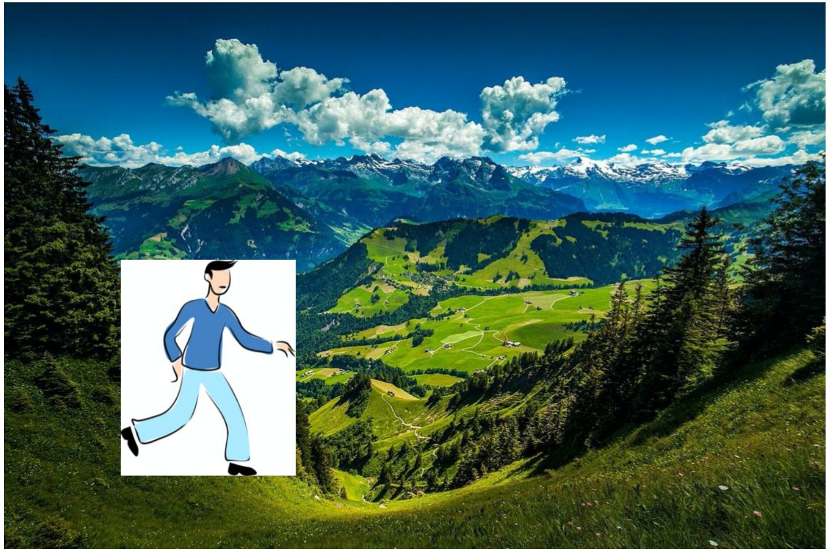
# Neural networks 101 recap

# Linear model

$$s = Wx + b$$

Given training data $\{x_i, y_i^{true}\}^N_{i=0}$
how to find the best W and b?

# Gradient descent



$$W_i = W_{i-1} - \eta \frac{\partial L(y, \hat{y})}{\partial W}$$

$$b_i = b_{i-1} - \eta \frac{\partial L(y, \hat{y})}{\partial b}$$

```
while True:
    weights_grad = compute_gradient(loss_fn, data, weights)
    weights -= step_size * weights_grad
```

Slide credit: Stanford CS231n
Image credit: Landscape image, walking man image

# Solution: Multilayer Neural Network

(Other names: multilayer perceptron / feedforward NN / fully-connected NN)

$$y_1 = \underline{f}(x_1 W_1 + b_1)$$

$$y_2 = \underline{f}(y_1 W_2 + b_2)$$

$$\cdots$$

$$y_n = \underline{f}(y_{n-1} W_n + b_n)$$

# Problem: how to compute gradients

$$y_1 = max\{xW_1 + b_1, 0\}$$
$$y_2 = max\{y_1W_2 + b_2, 0\}$$
$$\cdots$$
$$y_{n-1} = max\{y_2W_{n-2} + b_{n-2}, 0\}$$
$$p = softmax(yW_{n-1} + b_{n-1})$$
$$L = -\sum \hat{p} \log p$$

$$\frac{\partial L(y, \hat{y})}{\partial W_1} = ?$$
$$\cdots$$
$$\frac{\partial L(y, \hat{y})}{\partial W_{n-1}} = ?$$

By hand?

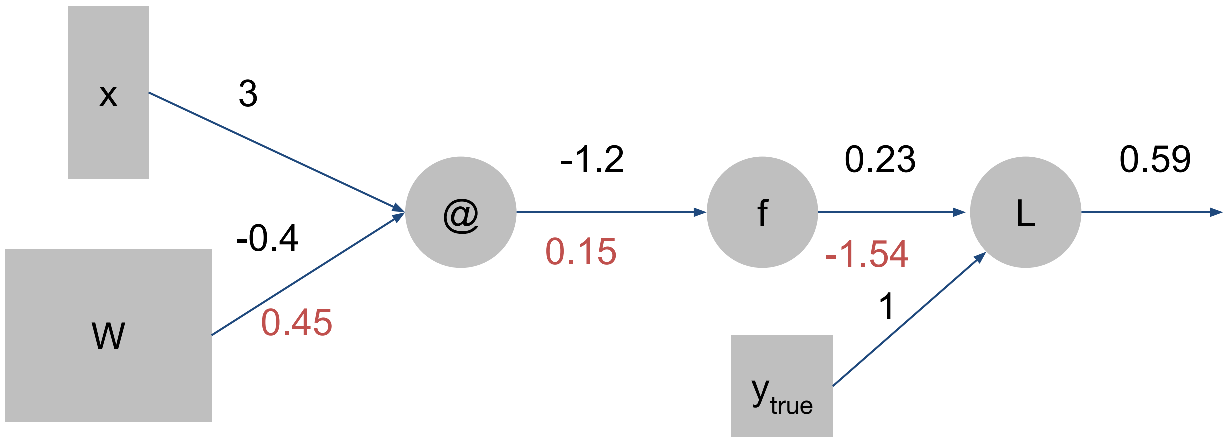
# Solution: backpropagation

Two main ideas:

- Chain rule
- Memorizing intermediate values

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \quad + \quad \text{memorization}$$

# Backward pass (computing the gradients)




f - sigmoid

$L(y, y_{true}) = (y - y_{true})^2$

# Real neural network backpropagation example

# Two layer neural network

$$h_1 = Linear(x) = W_1 x$$

$$h_2 = ReLu(h1) = \max(h_1, 0)$$

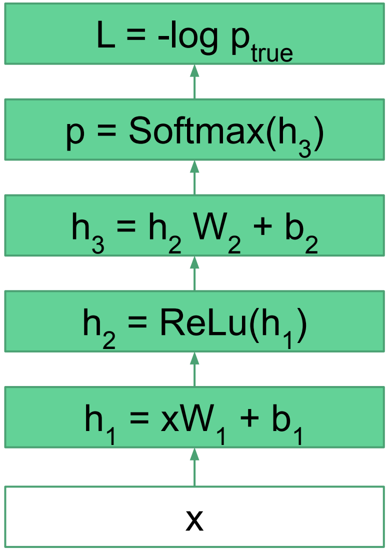
$$h_3 = Linear(h_2) = W_2 h_2$$

$$s = e^{h_3}$$

$$L = -\log \frac{s_{y_{true}}}{\sum_j s_j}$$

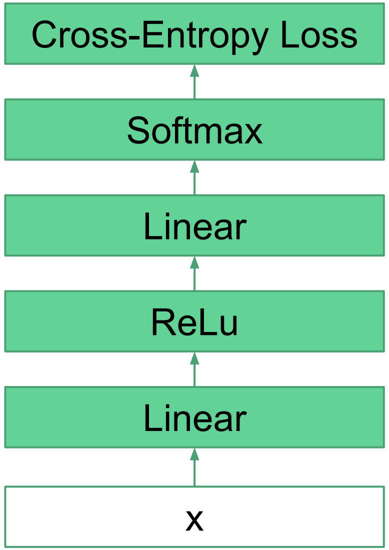
$$\frac{dL}{dW_1} = ?$$

# Two layer neural network




$$\frac{dL}{dW_1} = ?$$

# Two layer neural network




$$\frac{dL}{dW_1} = ?$$

# Two layer neural network

$$h_1 = Linear(x) = W_1 x$$

$$h_2 = ReLu(h1) = \max(h_1, 0)$$

$$h_3 = Linear(h_2) = W_2 h_2$$

$$s = e^{h_3}$$

$$L = -\log \frac{s_{y_{true}}}{\sum_j s_j}$$

$$\frac{dL}{dW_1} = \frac{dL}{ds} \frac{ds}{dh_3} \frac{dh_3}{dh_2} \frac{dh_2}{dh_2} \frac{dh_2}{dh_1} \frac{dh_1}{dW_1}$$

# Two layer neural network

$$\frac{dL}{ds} = \frac{e^s}{\sum_j e^{s_j}} - I_{y_{true}}$$

$$\frac{ds}{dh_3} = e^{h_3}$$

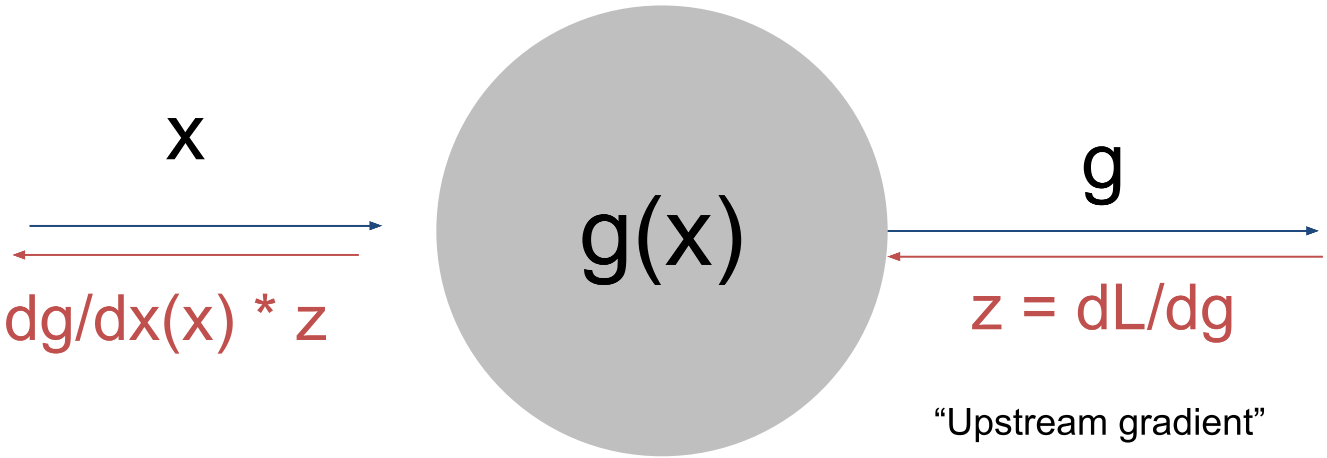
$$\frac{dh_3}{dh_2} = h_2^T$$

$$\frac{dh_2}{dh_1} = I_{h_1>0} \circ h_1^2$$

$$\frac{dh_1}{dW_1} = x^T$$

$$\frac{dL}{dW_1} = \frac{dL}{ds}\frac{ds}{dh_3}\frac{dh_3}{dh_2}\frac{dh_2}{dh_2}\frac{dh_2}{dh_1}\frac{dh_1}{dW_1}$$

# Backpropagation




Slide credit: Stanford CS231n

# Backpropagation
(we don't need the chain rule for the first step)

s, $y_{true}$

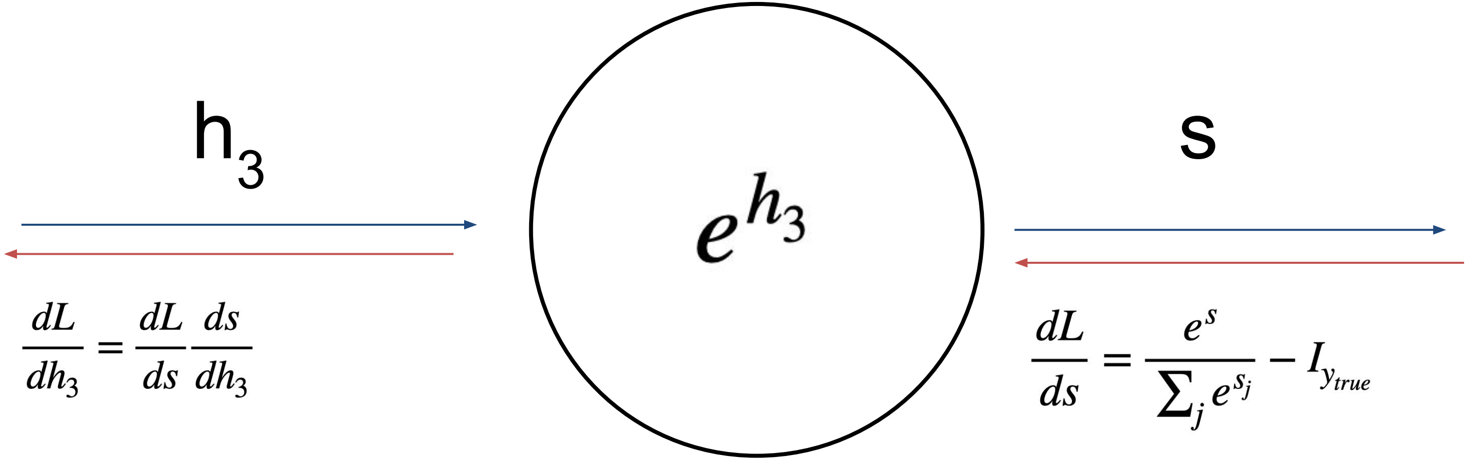
$$-\log \frac{s_{y_{true}}}{\sum_j s_j}$$

L

$$\frac{dL}{ds} = \frac{e^s}{\sum_j e^{s_j}} - I_{y_{true}}$$

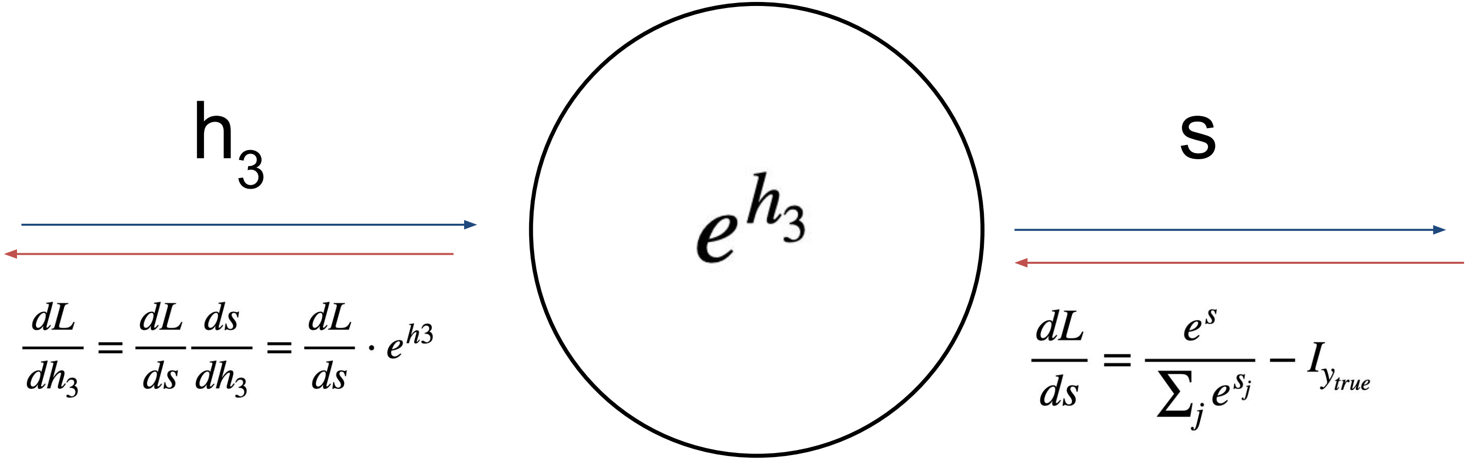
Slide credit: Stanford CS231n

# Backpropagation
## the gradient from the previous step is our downstream gradient




Slide credit: Stanford CS231n

# Backpropagation
the gradient from the previous step is our downstream gradient

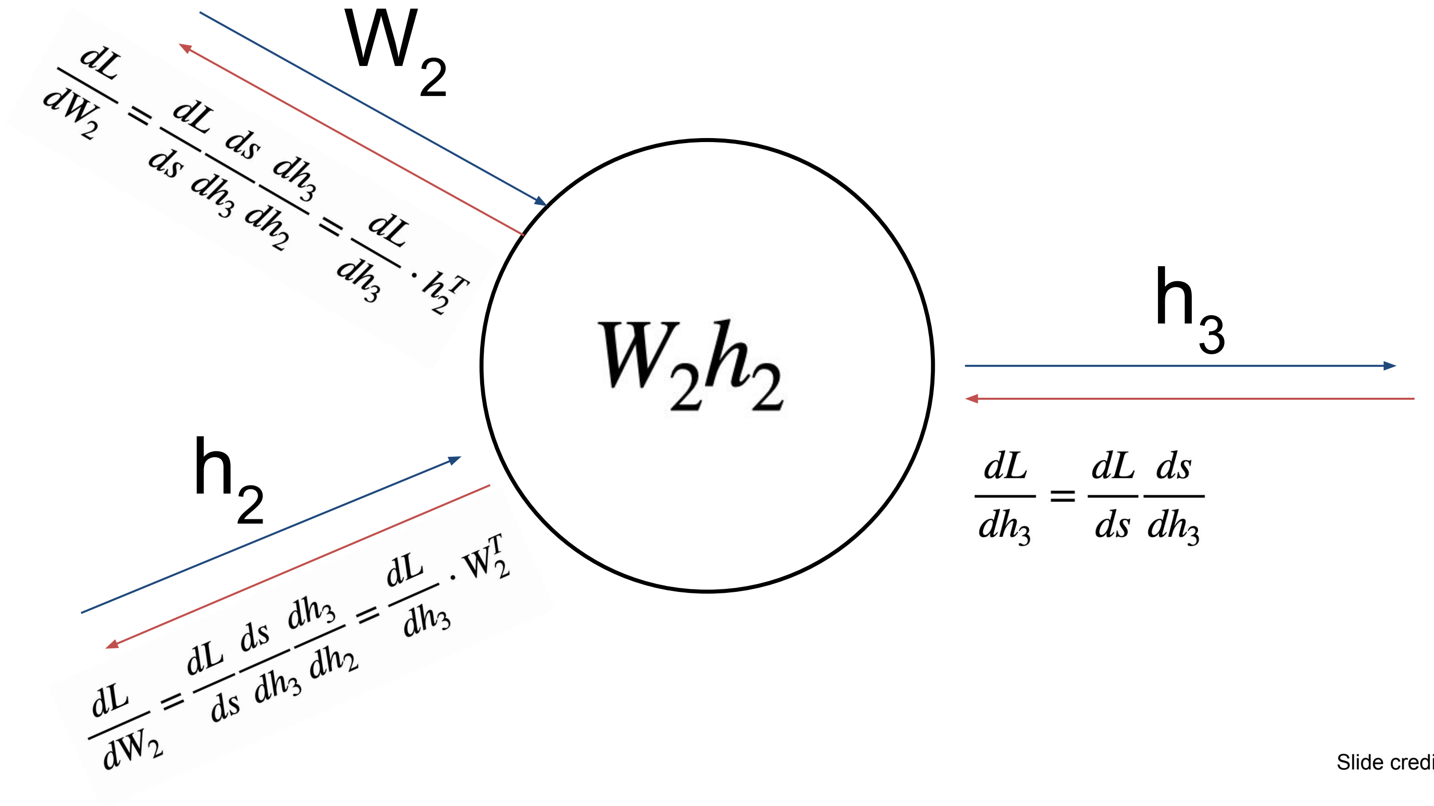



$$\frac{dL}{dh_3} = \frac{dL}{ds}\frac{ds}{dh_3} = \frac{dL}{ds} \cdot e^{h3}$$

$$\frac{dL}{ds} = \frac{e^s}{\sum_j e^{s_j}} - I_{y_{true}}$$

Slide credit: Stanford CS231n

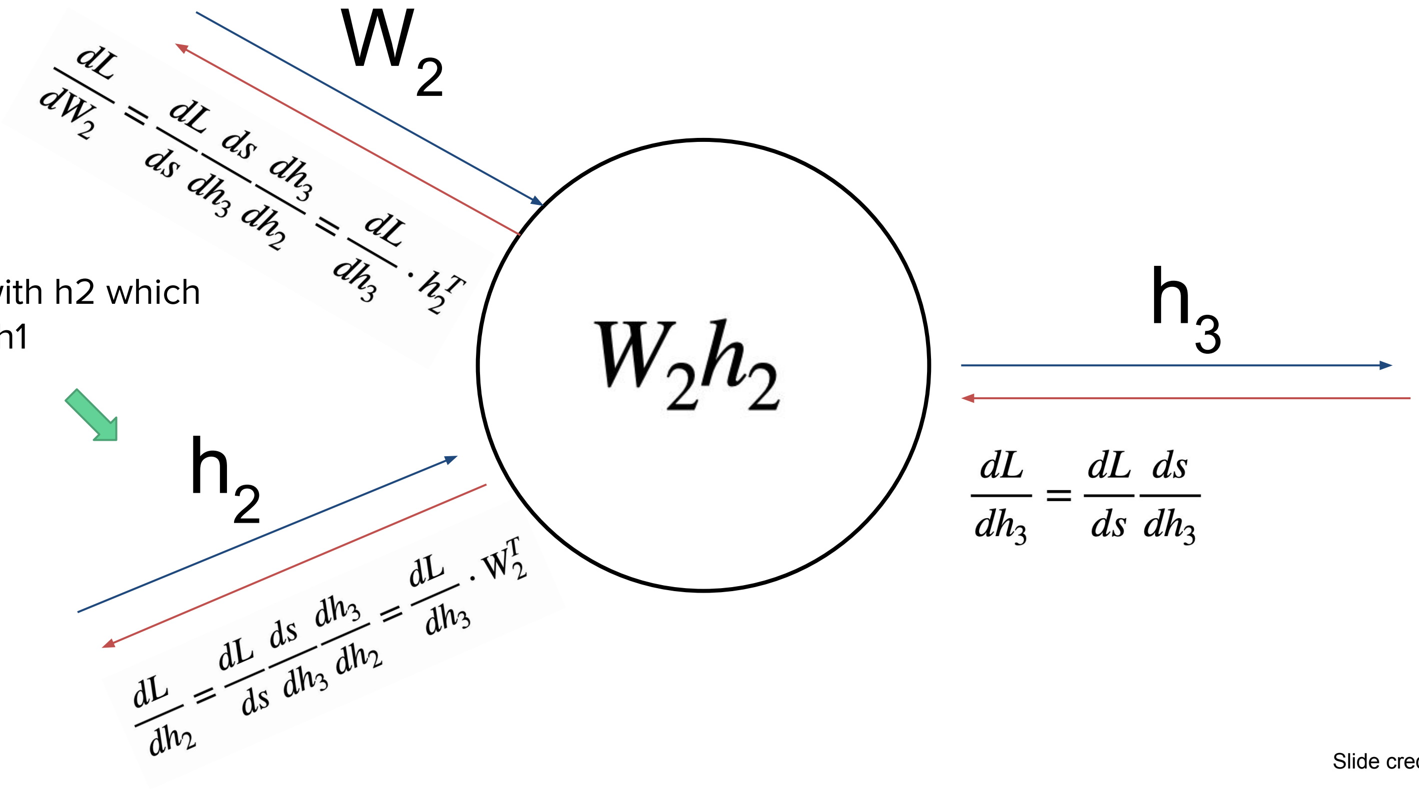
# Backpropagation. Function with two inputs




Slide credit: Stanford CS231n

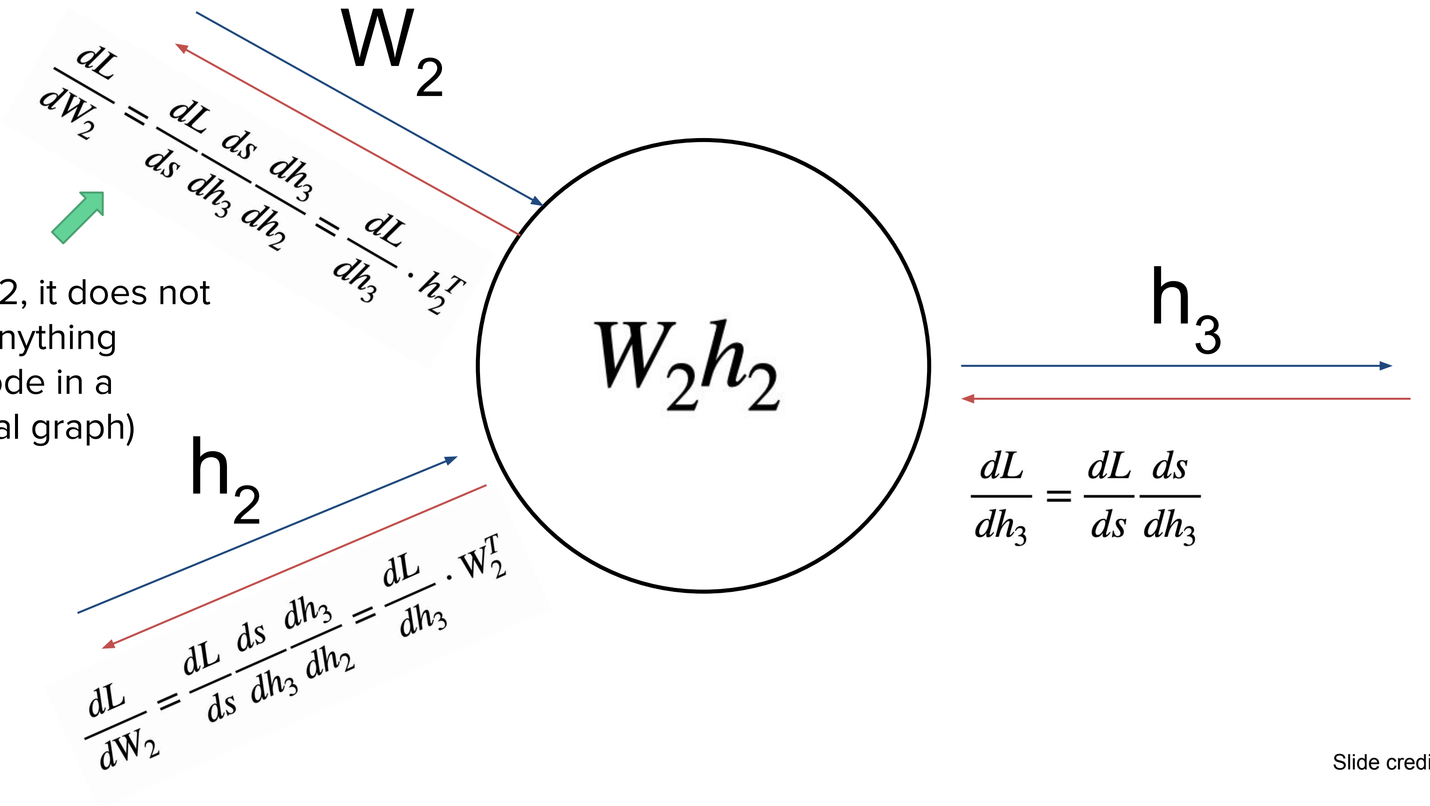
# Backpropagation. Function with two inputs

Continuing with h2 which depends on h1




Slide credit: Stanford CS231n

# Backpropagation. Function with two inputs




Done with W2, it does not depend on anything (a.k.a. leaf node in a computational graph)

Slide credit: Stanford CS231n

# Backpropagation

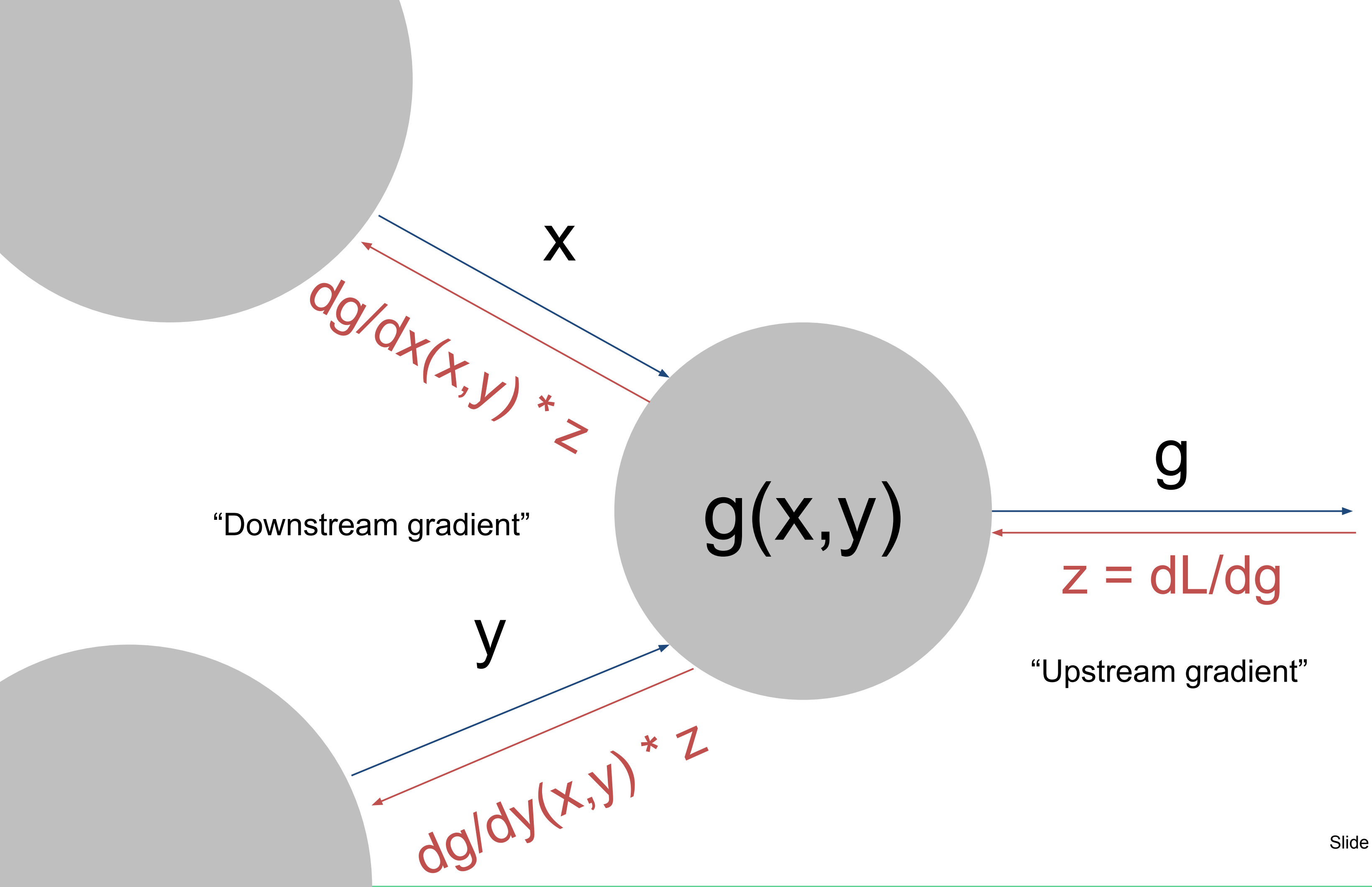
$h_1$

$\max(h_1, 0)$

$h_2$

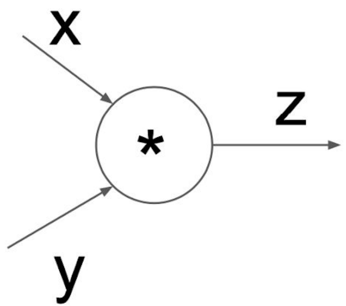
$$\frac{dL}{dh_1} = \frac{dL}{ds}\frac{ds}{dh_3}\frac{dh_3}{dh_2}\frac{dh_2}{dh_1} = \frac{dL}{dh_2} \cdot I_{h_1>0}$$

$$\frac{dL}{dh_2} = \frac{dL}{ds}\frac{ds}{dh_3}\frac{dh_3}{dh_2} = \frac{dL}{dh_3} \cdot W_2^T$$




Slide credit: Stanford CS231n

# How it is actually implemented in PyTorch




(x,y,z are scalars)

```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz
    return grad_x, grad_y
```

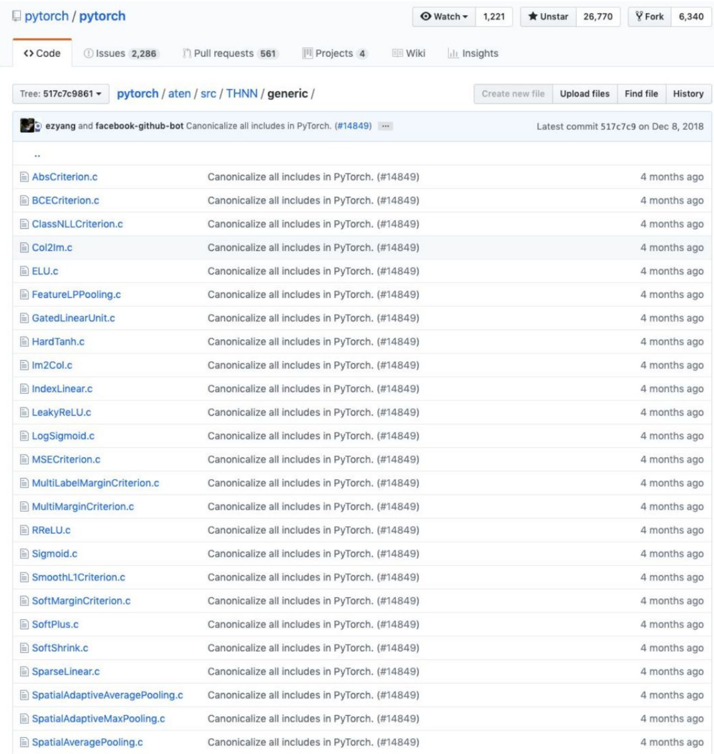
Need to stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

Slide credit: Stanford CS231n

# How it is actually implemented in PyTorch



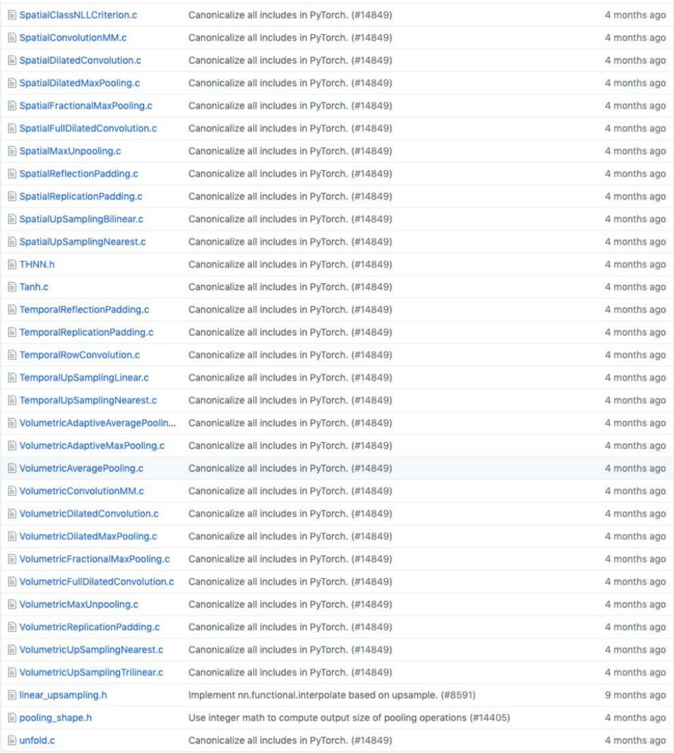
pytorch / pytorch — Watch 1,221 — Unstar 26,770 — Fork 6,340

Code — Issues 2,286 — Pull requests 561 — Projects 4 — Wiki — Insights

Tree: 517c7c9861 — pytorch / aten / src / THNN / generic / — Create new file — Upload files — Find file — History

ezyang and facebook-github-bot Canonicalize all includes in PyTorch. (#14849) ... — Latest commit 517c7c9 on Dec 8, 2018

| | | |
|---|---|---|
| .. | | |
| AbsCriterion.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| BCECriterion.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| ClassNLLCriterion.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| Col2Im.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| ELU.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| FeatureLPPooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| GatedLinearUnit.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| HardTanh.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| Im2Col.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| IndexLinear.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| LeakyReLU.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| LogSigmoid.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| MSECriterion.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| MultiLabelMarginCriterion.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| MultiMarginCriterion.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| RReLU.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| Sigmoid.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SmoothL1Criterion.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SoftMarginCriterion.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SoftPlus.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SoftShrink.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SparseLinear.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialAdaptiveAveragePooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialAdaptiveMaxPooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialAveragePooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |



| | | |
|---|---|---|
| SpatialClassNLLCriterion.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialConvolutionMM.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialDilatedConvolution.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialDilatedMaxPooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialFractionalMaxPooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialFullDilatedConvolution.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialMaxUnpooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialReflectionPadding.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialReplicationPadding.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialUpSamplingBilinear.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| SpatialUpSamplingNearest.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| THNN.h | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| Tanh.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| TemporalReflectionPadding.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| TemporalReplicationPadding.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| TemporalRowConvolution.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| TemporalUpSamplingLinear.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| TemporalUpSamplingNearest.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricAdaptiveAveragePoolin... | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricAdaptiveMaxPooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricAveragePooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricConvolutionMM.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricDilatedConvolution.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricDilatedMaxPooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricFractionalMaxPooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricFullDilatedConvolution.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricMaxUnpooling.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricReplicationPadding.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricUpSamplingNearest.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| VolumetricUpSamplingTrilinear.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |
| linear_upsampling.h | Implement nn.functional.interpolate based on upsample. (#8591) | 9 months ago |
| pooling_shape.h | Use integer math to compute output size of pooling operations (#14405) | 4 months ago |
| unfold.c | Canonicalize all includes in PyTorch. (#14849) | 4 months ago |

Slide credit: Stanford CS231n

# We talk about a computational graph, but how is it represented?

TensorFlow 1.X, Theano, Cafee: explicit static graphs

PyTorch 1.X: dynamic graphs

A graph is created on the fly


```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

# We talk about a computational graph, but how is it represented?

TensorFlow 1.X, Theano, Cafee: explicit static graphs

PyTorch 1.X: dynamic graphs

# We talk about a computational graph, but how is it represented?

TensorFlow 1.X, Theano, Cafee: explicit static graphs

PyTorch 1.X: dynamic graphs

PyTorch 2.X: implicit static graphs with just-in-time compilation

- More about that when we talk about hardware and tooling

# Backpropagation

- Just a way of computing the gradients for arbitrary computational graph
- Can be applied not just to FCN, you can use if for any (almost) differentiable function

- Forward pass for values
- Backward pass for gradients

# General definition of neural network

Neural network is any **differentiable* computational graph**

# General definition of neural network

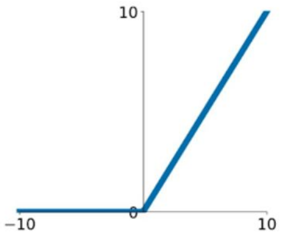
Neural network is any **differentiable* computational graph**

usually trained with a gradient optimization method
and use backpropagation to compute these gradients

# Note on differentiability

The class of functions that can be used in neural networks is (a bit) broader than differentiable functions[1]

- ReLU = max(0, x) is not differentiable at x=0, but it is widely used




Solution: ReLU is only non-differentiable in a single point (x=0), we can pretend that the derivative there is 0

1: More technical answer would be that we can use functions that are differentiable *almost everywhere*
https://en.wikipedia.org/wiki/Almost_everywhere

# Vector derivatives

## Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

## Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will y change?

## Vector to Vector

$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will each element of y change?

Slide credit: Stanford CS231n

# Vector derivatives: a trick that works 99% of the time

1. Pretend you work with a scalar function. Compute derivative of it, do not change move the order of multiplications.
2. Now remember that these are actually matrices and we can only multiply matrices that have matching shapes. Add transpositions where necessary.
3. The gradient of any parameter W should have the exact same shape as the parameter itself (remember that we will be subtracting them from each other in gradient descent).

# TL;DR

- Neural network is a differentiable(-ish) computational graph
- NNs are optimized using gradient methods
- Gradients are computed via backpropagation (chain rule + memorization)
- Backprop computes the value of analytical gradient, it does not perform numerical approximations
- Fully-connected neural network is:

$$
\begin{aligned}
y_1 &= f(x_1 W_1 + b_1) \\
y_2 &= f(y_1 W_2 + b_2) \\
&\dots \\
y_n &= f(y_{n-1} W_{n-1} + b_{n-1})
\end{aligned}
$$

# Homework

# Homework

- This homework is not about NLP, it is about neural networks.
- No deep learning frameworks
- Just linear algebra (numpy)
- Part 1: Write your own linear model from scratch
- Part 2: Write your own neural network from scratch
- **Significantly** harder than the first homeworks
    - Plan that it will take **more than two days**
    - Ask questions in Discord
    - Come to office hours on Monday

The catch: no for-loops allowed, unless specified otherwise.
The tip: use numpy broadcasting.

# Homework. Extra materials

- How to run the homework in Colab: https://cs231n.github.io/assignments2021/assignment1/
- Linear classifier math: https://cs231n.github.io/linear-classify/
- Backpropagation math: https://cs231n.github.io/optimization-2/