I have a Dream!

That one day, we will

Backpropagate Easily!

ES654: Machine Learning

Varun Gohil

Remember Autograd?

Autograd

build passing benchmarked by asv

Autograd can automatically differentiate native Python and Numpy code. It can handle a large subset of Python's features, including loops, ifs, recursion and closures, and it can even take derivatives of derivatives of derivatives. It supports reverse-mode differentiation (a.k.a. backpropagation), which means it can efficiently take gradients of scalar-valued functions with respect to array-valued arguments, as well as forward-mode differentiation, and the two can be composed arbitrarily. The main intended application of Autograd is gradient-based optimization. For more information, check out the tutorial and the examples directory.

Example use:

```
>>> import autograd.numpy as np  # Thinly-wrapped numpy
>>> from autograd import grad    # The only autograd function you may ever need
>>>
>>> def tanh(x):                 # Define a function
...     y = np.exp(-2.0 * x)
...     return (1.0 - y) / (1.0 + y)
...
>>> grad_tanh = grad(tanh)       # Obtain its gradient function
>>> grad_tanh(1.0)               # Evaluate the gradient at x = 1.0
0.41997434161402603
>>> (tanh(1.0001) - tanh(0.9999)) / 0.0002  # Compare to finite differences
0.41997434264973155
```

Remember Autograd?

Autograd

build passing benchmarked by asv

Autograd can automatically differentiate native Python and Numpy code. It can handle a large subset of P… including loops, ifs, recursion and closures, and it can even take derivatives of derivati… of … mode differentiation (a.k.a. backpropagation), which means it can efficiently take … of … functions with respect to array-valued arguments, as well as forward-mode differenti… be composed arbitrarily. The main intended application of Autograd is gradient-based … rmation, check out the tutorial and the examples directory.

Example use:

```
... ad… … np  # Thinly-wrapped numpy
...     import grad    # The only autograd function you may ever need
...
... def tanh(x):                # Define a function
...     y = np.exp(-2.0 * x)
...     return (1.0 - y) / (1.0 + y)
...
>>> grad_tanh = grad(tanh)       # Obtain its gradient function
>>> grad_tanh(1.0)               # Evaluate the gradient at x = 1.0
0.41997434161402603
>>> (tanh(1.0001) - tanh(0.9999)) / 0.0002  # Compare to finite differences
0.41997434264973155
```

How does Autograd work?

After this lecture

- You will have an idea of how Autograd works!
- You will know how backpropagation is implemented in popular ML frameworks like TensorFlow, PyTorch etc.
- You won’t bang your head against the wall while implementing backprop. 

Computational Graphs

- Nodes ⟶ operations
- Edges ⟶ variables/tensors

Edges also represent data dependencies between operations.

Computational Graphs

- Nodes ⟶ operations
- Edges ⟶ tensors/variables.

Edges also represent data dependencies between operations.

Example: (x * y) + z

Computational Graphs

- Nodes ⟶ operations
- Edges ⟶ tensors/variables.

Edges also represent data dependencies between operations.

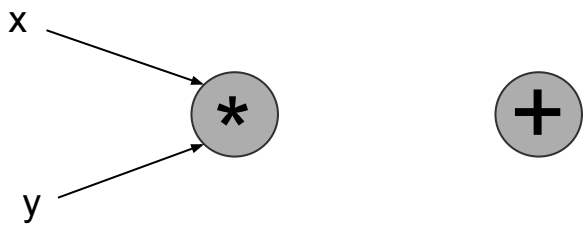
Example: (x * y) + z

Computational Graphs

- Nodes ⟶ operations
- Edges ⟶ tensors/variables.

Edges also represent data dependencies between operations.

Example: (x * y) + z

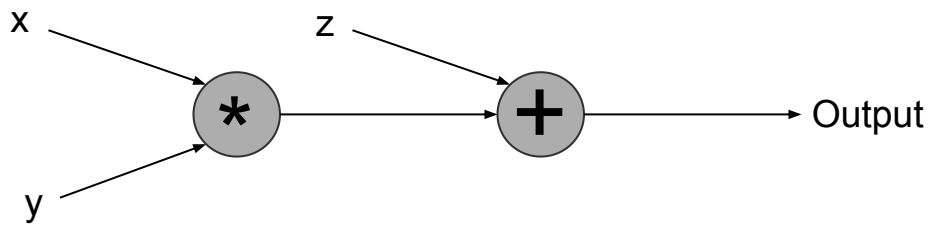


Computational Graphs

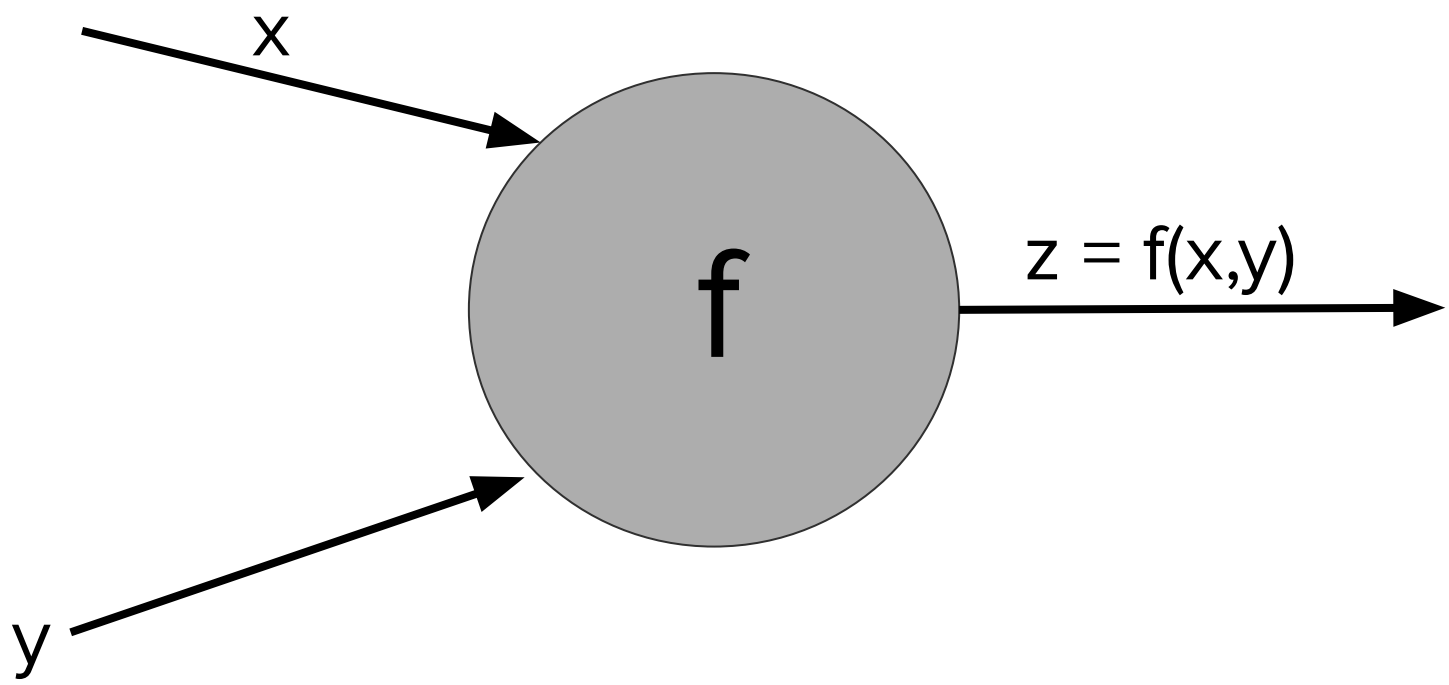
- Nodes ⟶ operations
- Edges ⟶ tensors/variables.

Edges also represent data dependencies between operations.

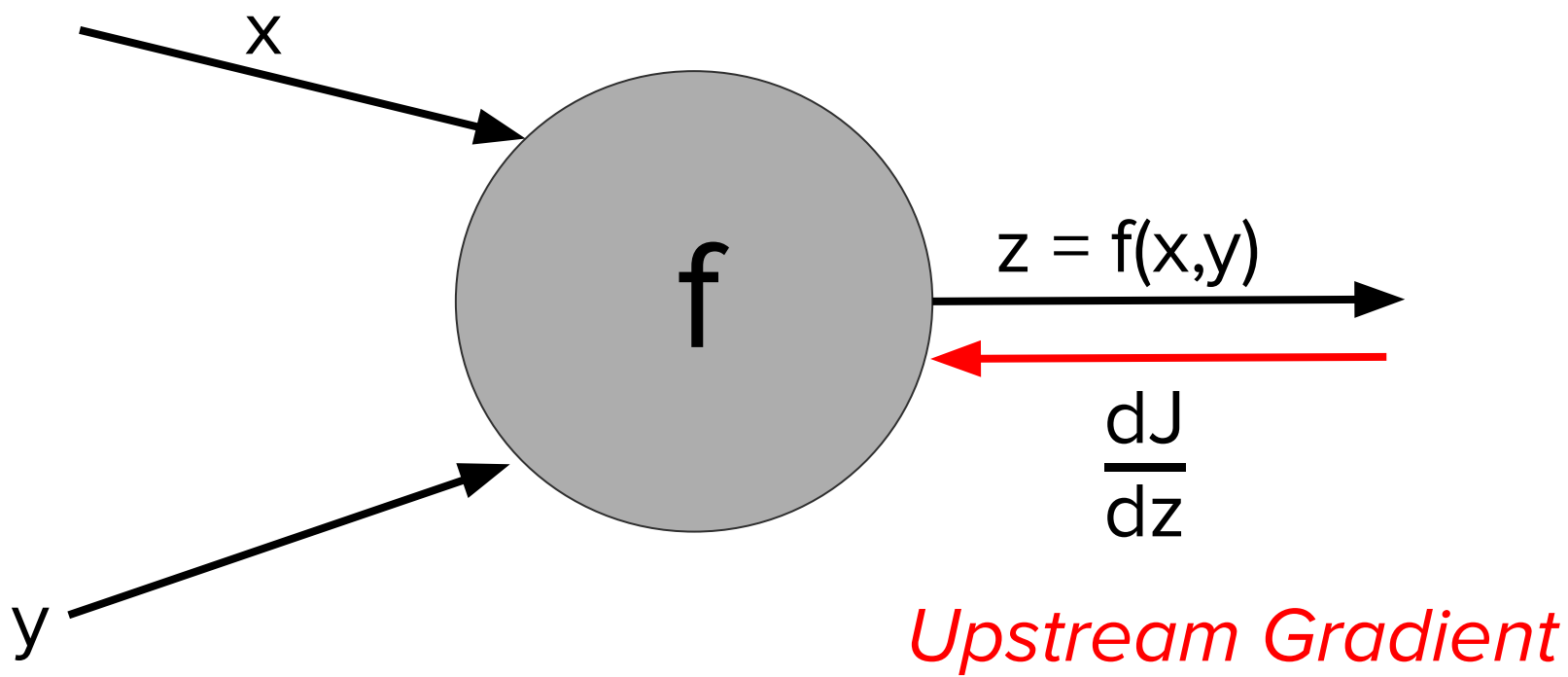
Example: (x * y) + z



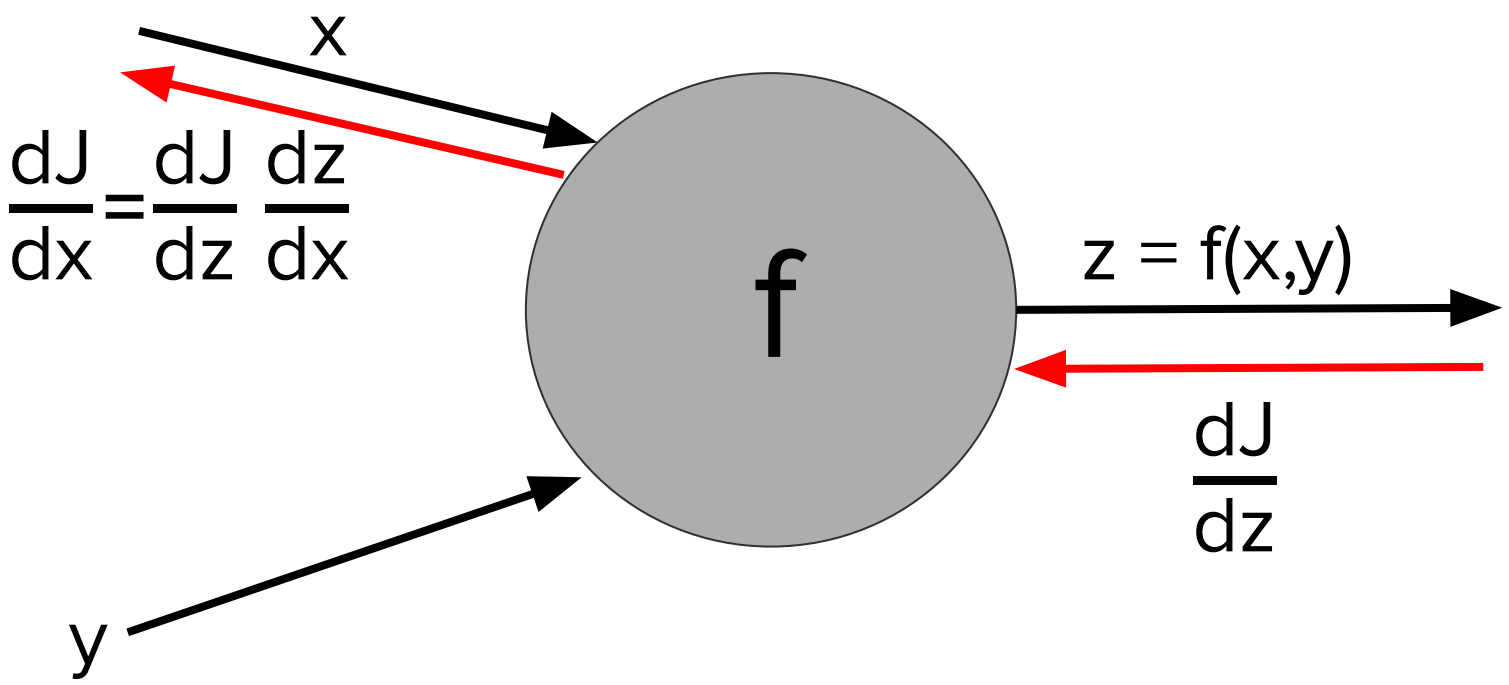
Backprop through a node in Computational graph



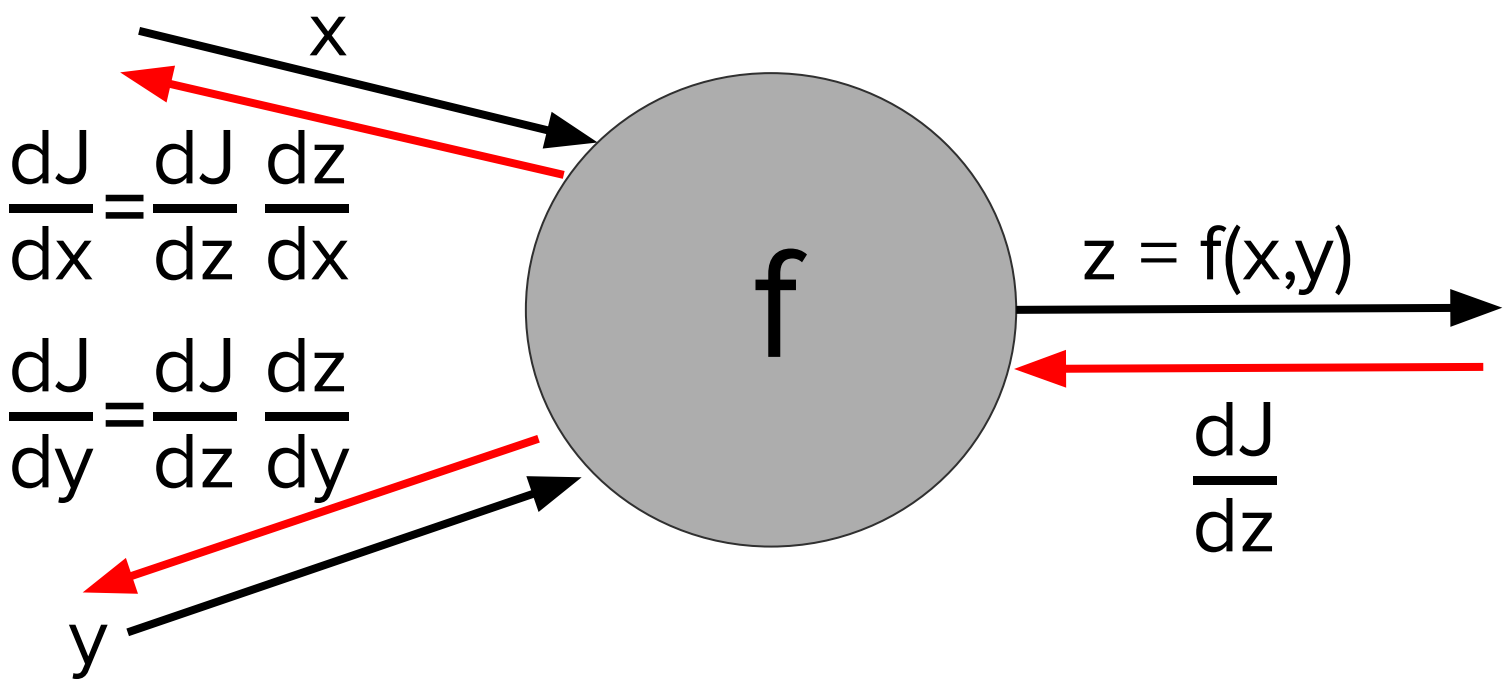
Backprop through a node in Computational graph



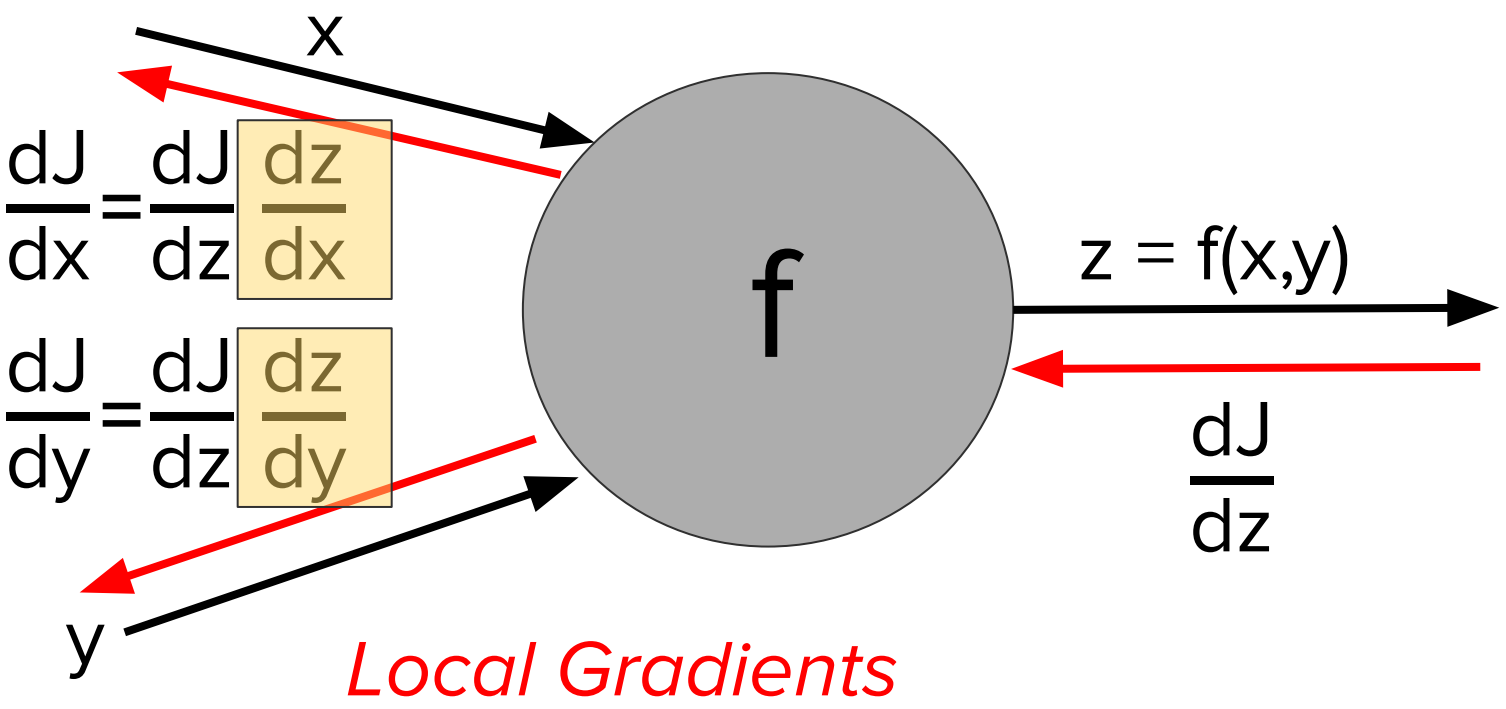
Backprop through a node in Computational graph



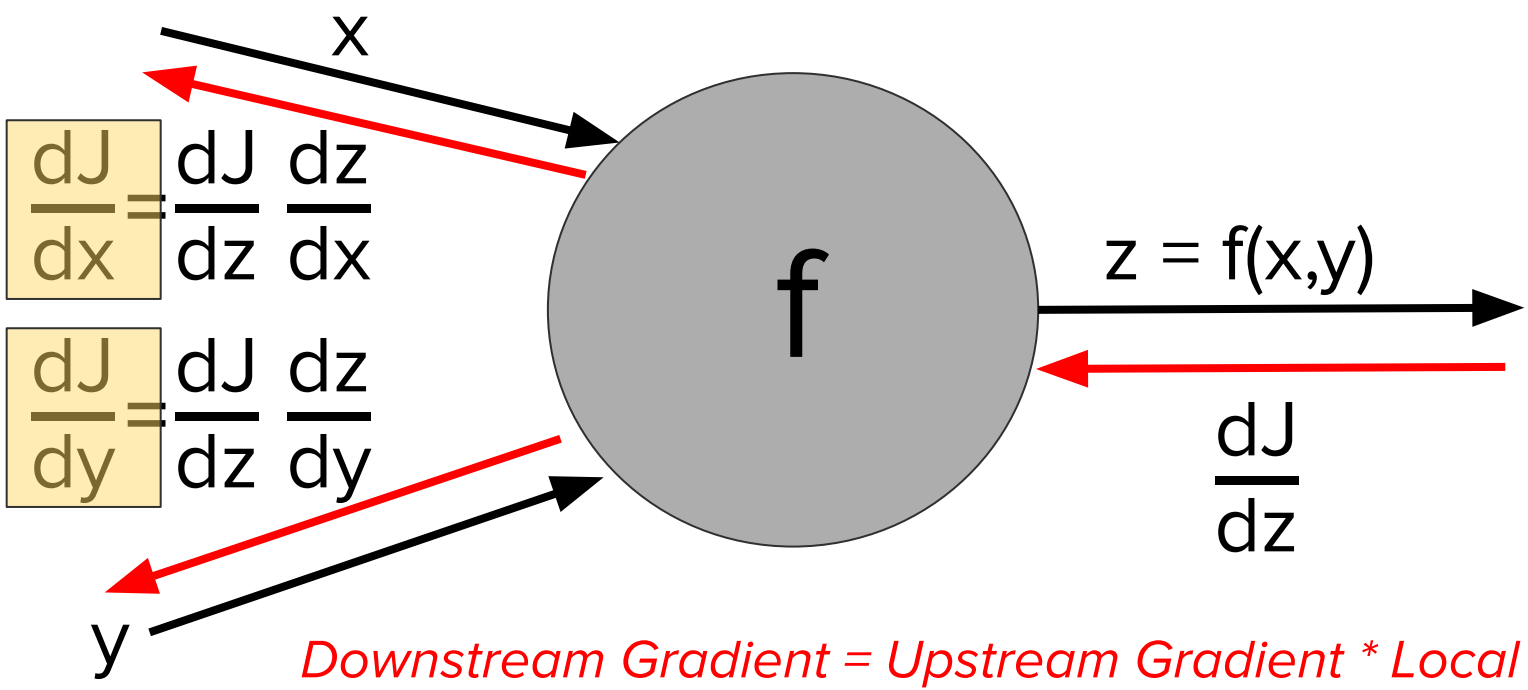
Backprop through a node in Computational graph



Backprop through a node in Computational graph



Backprop through a node in Computational graph



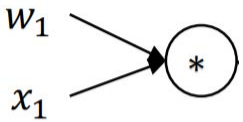
*Downstream Gradient = Upstream Gradient * Local Gradient*

An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

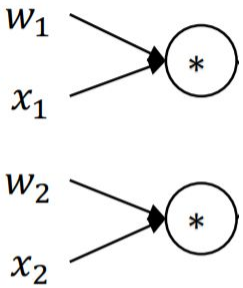
An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



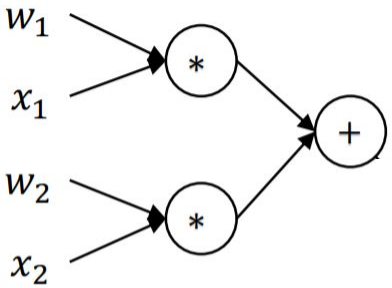
An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



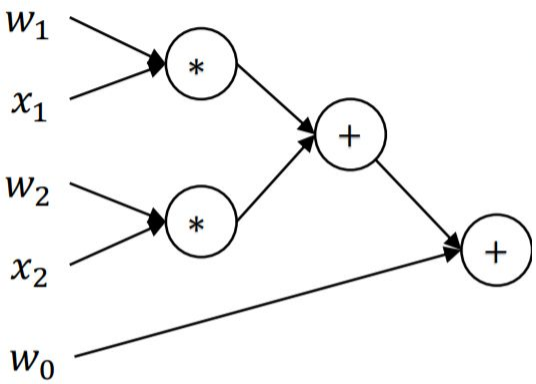
An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



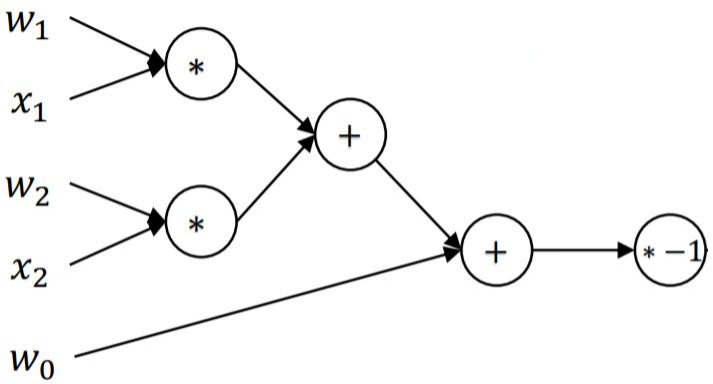
An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



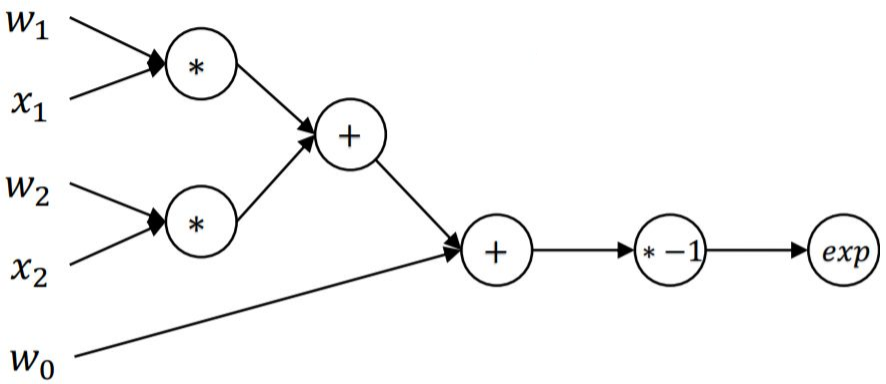
An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



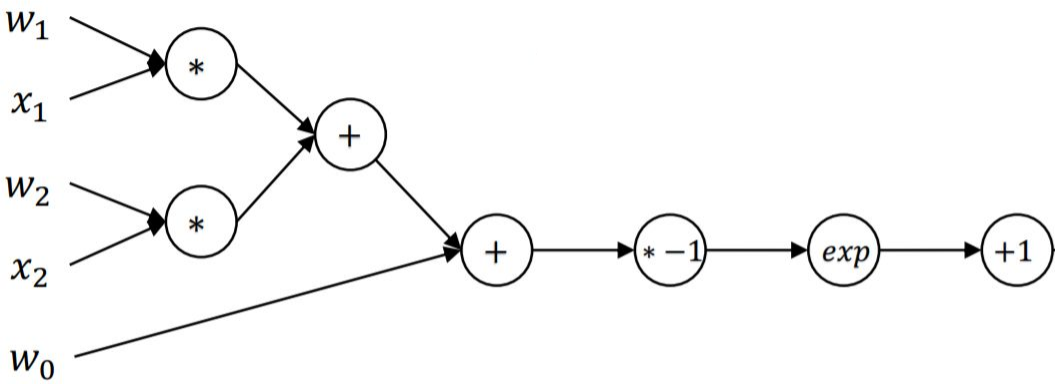
An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



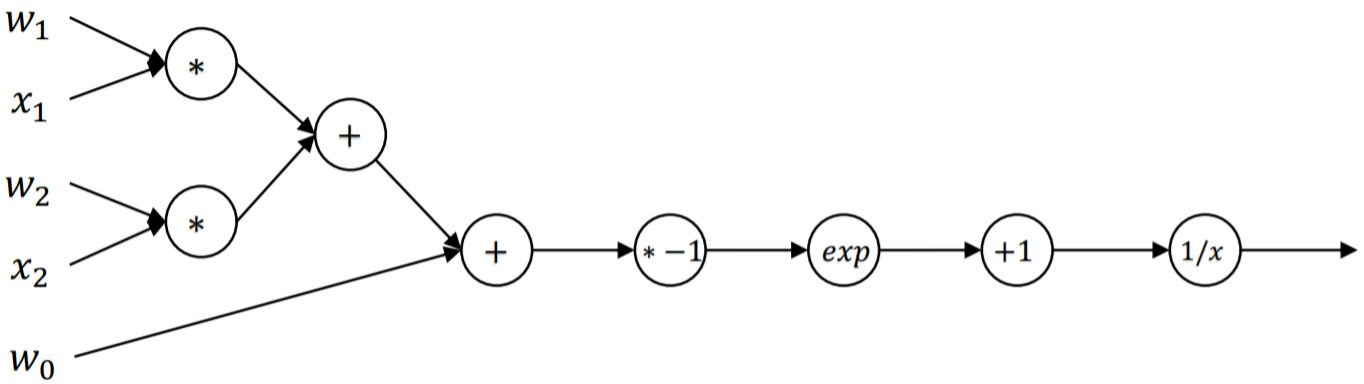
An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



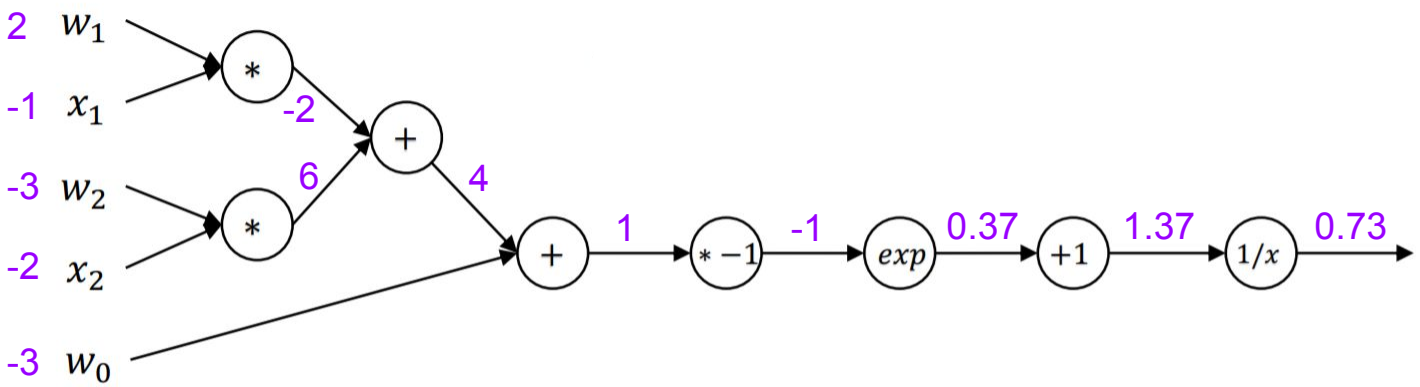
An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



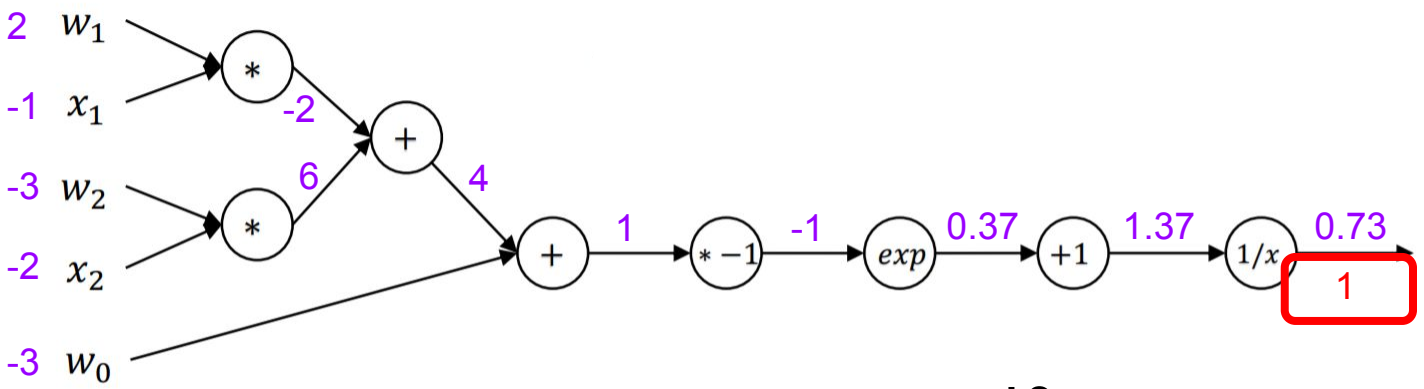
An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



An Example!

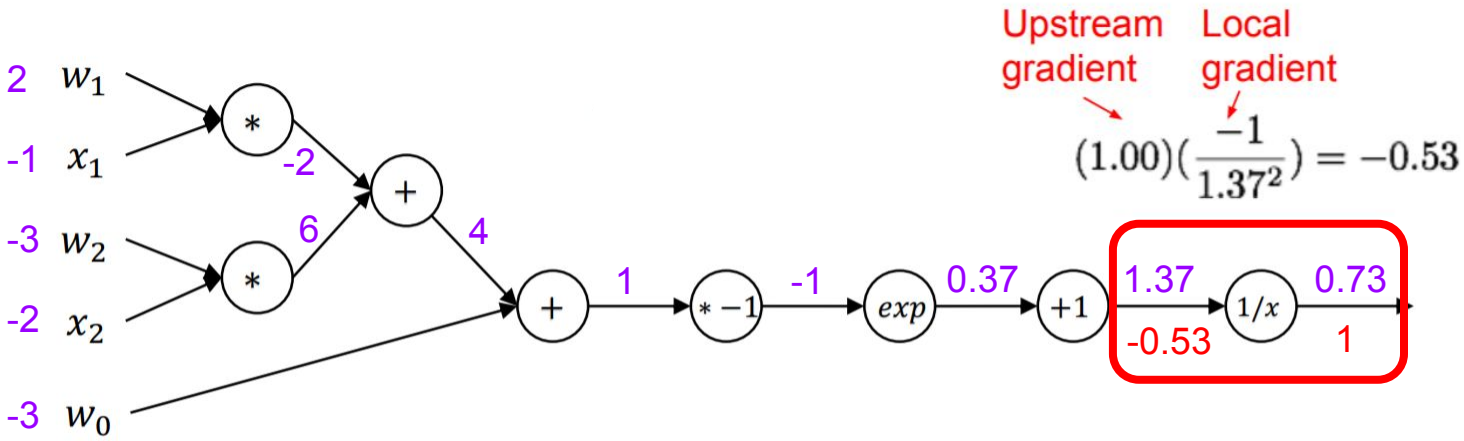
$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



$$\frac{df}{df} = 1$$

An Example!

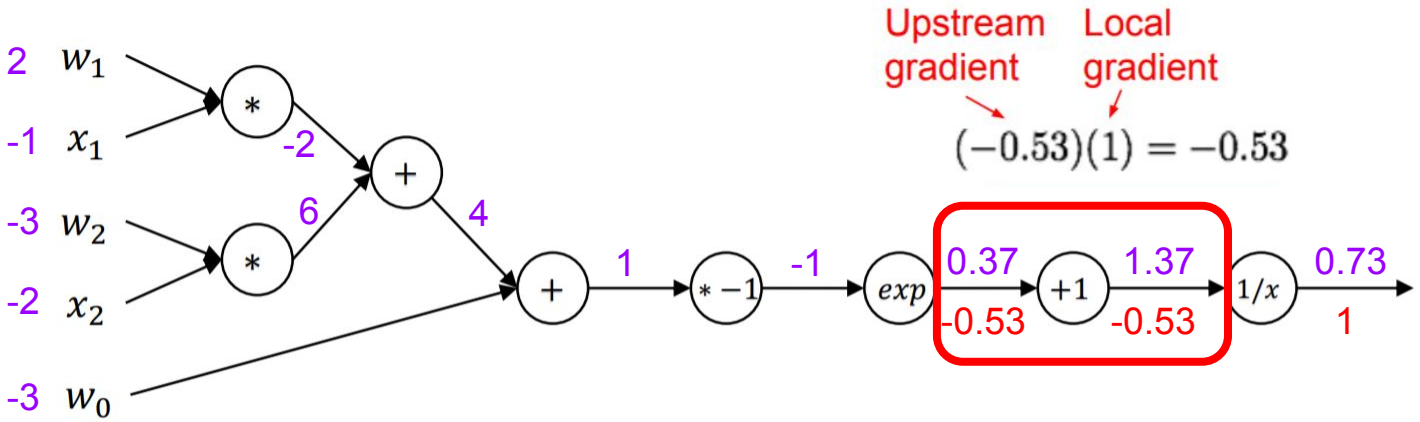
$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



$$f(x) = 1/x \quad \rightarrow \quad \frac{\partial f}{\partial x} = -1/x^2$$

An Example!

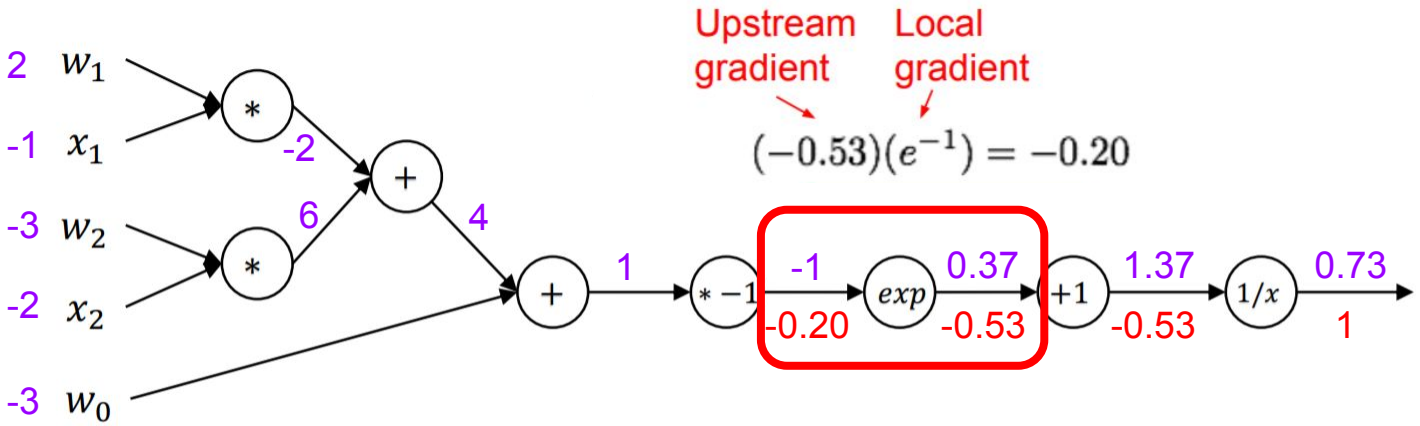
$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



$$f(x) = x + 1 \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1$$

An Example!

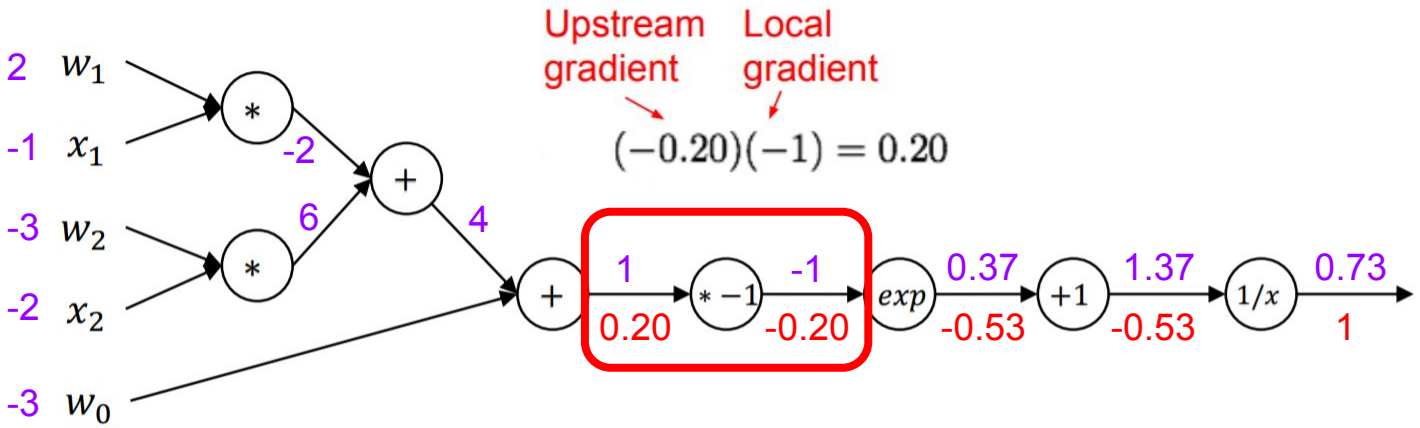
$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{\partial f}{\partial x} = e^x$$

An Example!

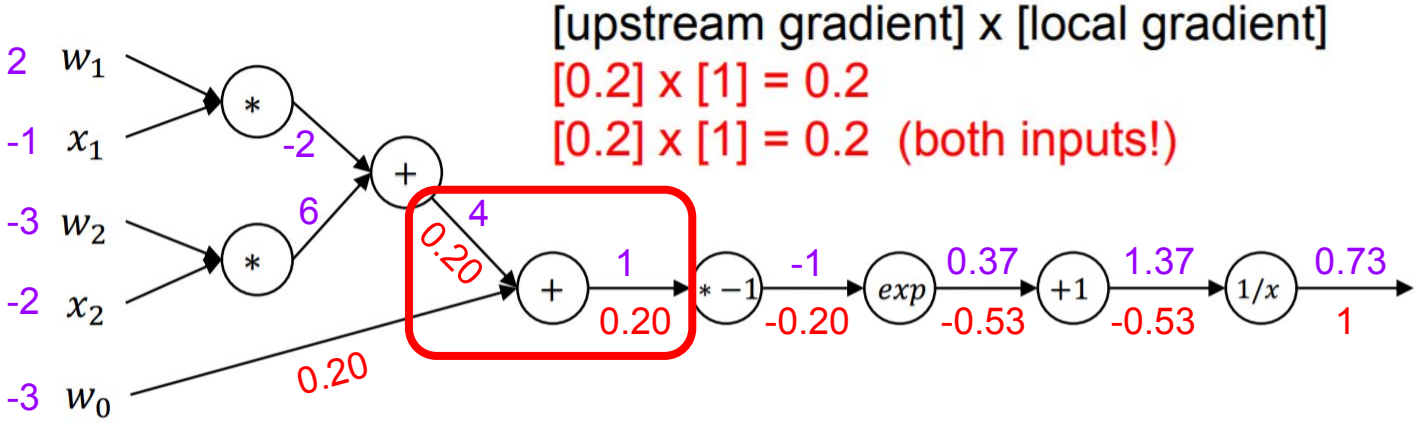
$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



$$f(x) = -x \rightarrow \quad \frac{\partial f}{\partial x} = -1$$

An Example!

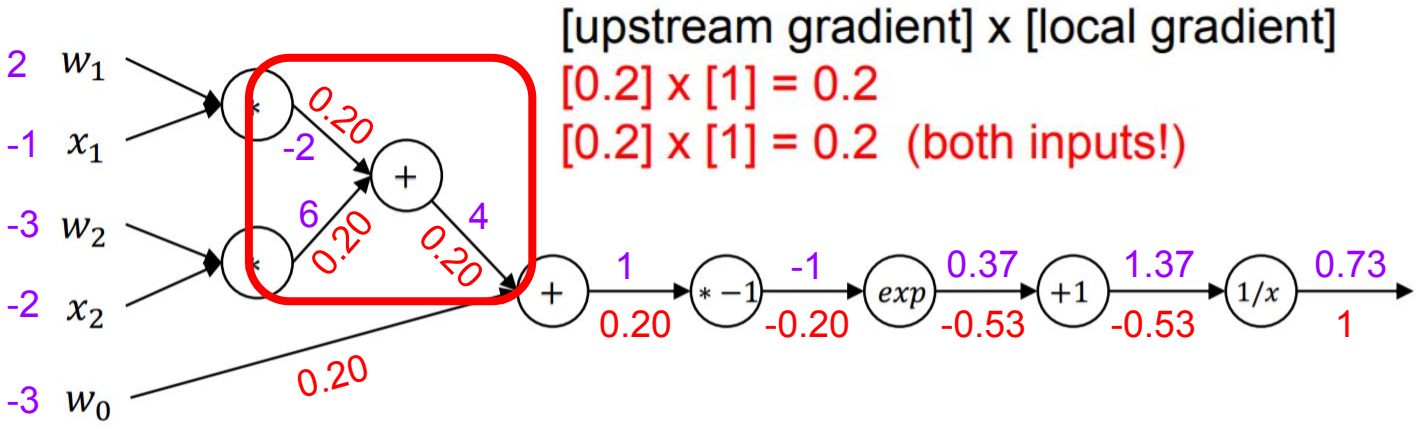
$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



$$f(x) = x + w \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial w} = 1$$

An Example!

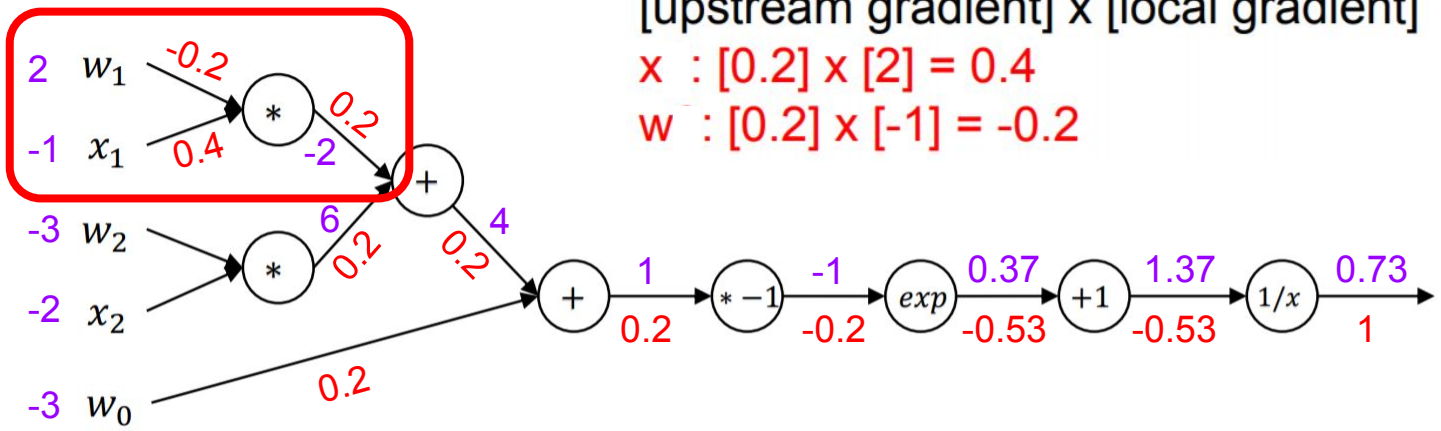
$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



$$f(x) = x + w \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1 \quad \frac{\partial f}{\partial w} = 1$$

An Example!

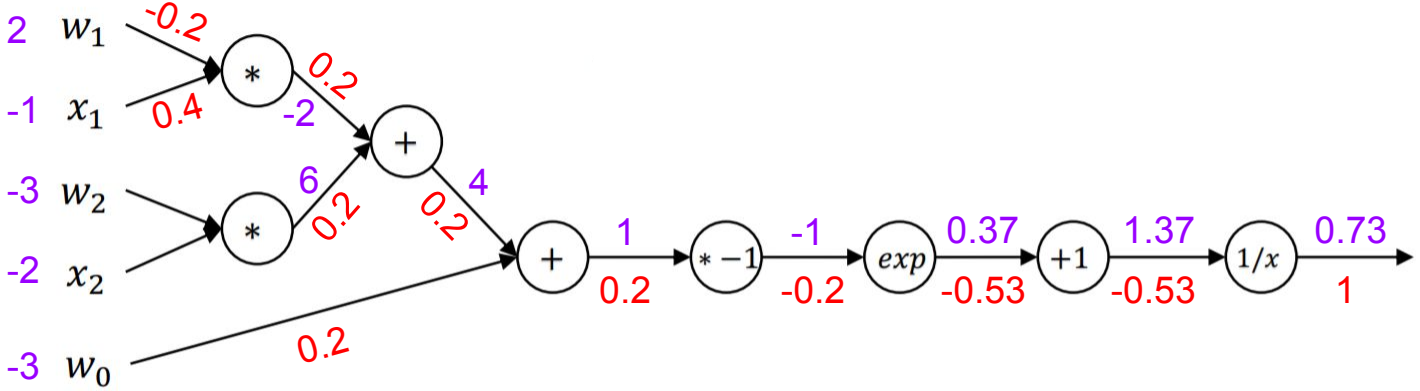
$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



$$f(x, w) = xw \quad \rightarrow \quad \frac{\partial f}{\partial x} = w, \ \frac{\partial f}{\partial w} = x$$

An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

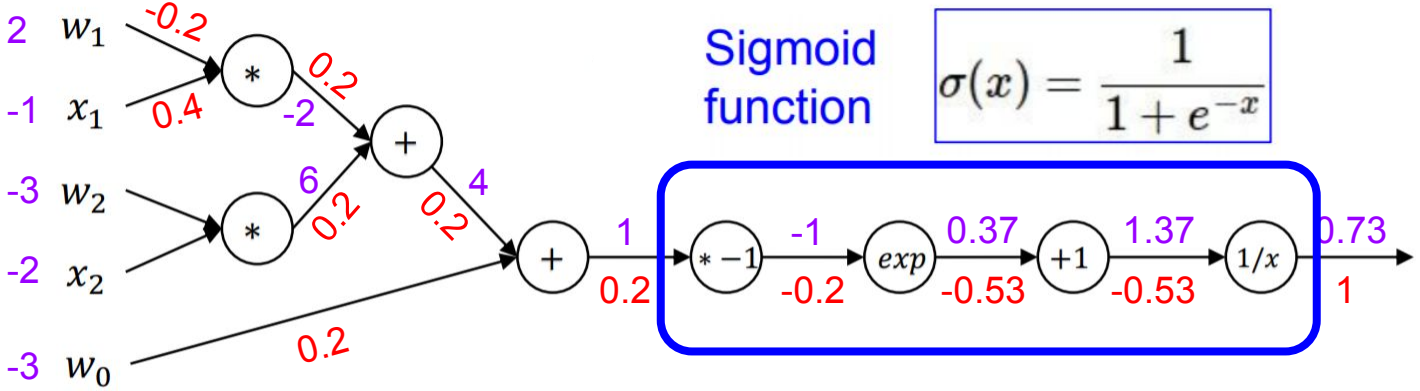


There might be multiple computational graphs for the same expression.

Choose one where local gradients of each node can be easily expressed!

An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

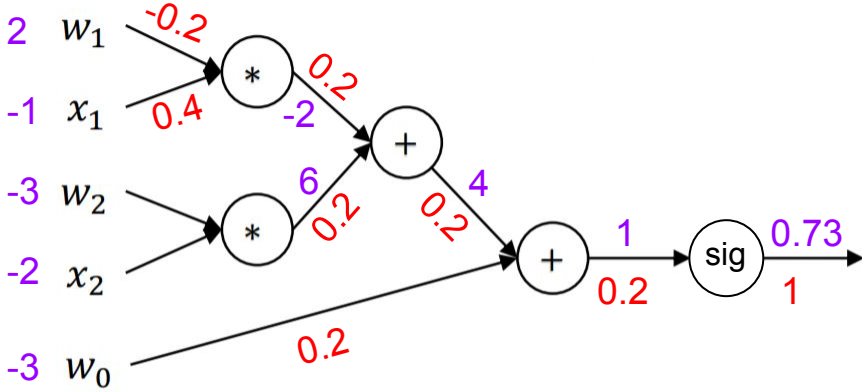


There might be multiple computational graphs for the same expression.

Choose one where local gradients of each node can be easily expressed!

An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

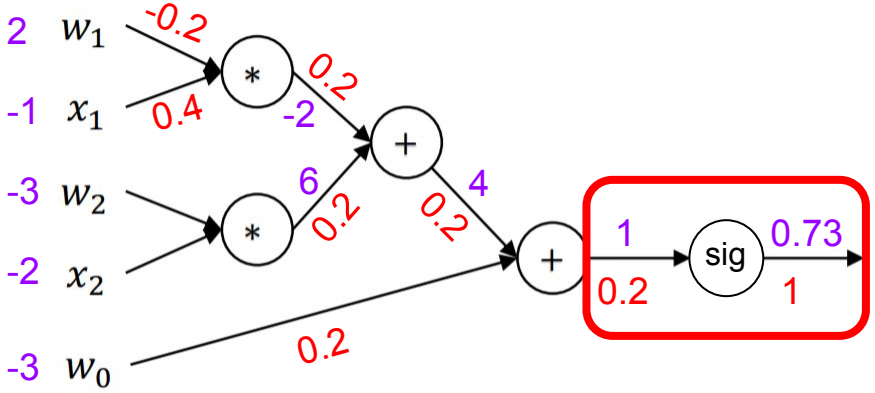


There might be multiple computational graphs for the same expression.

Choose one where local gradients of each node can be easily expressed!

An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

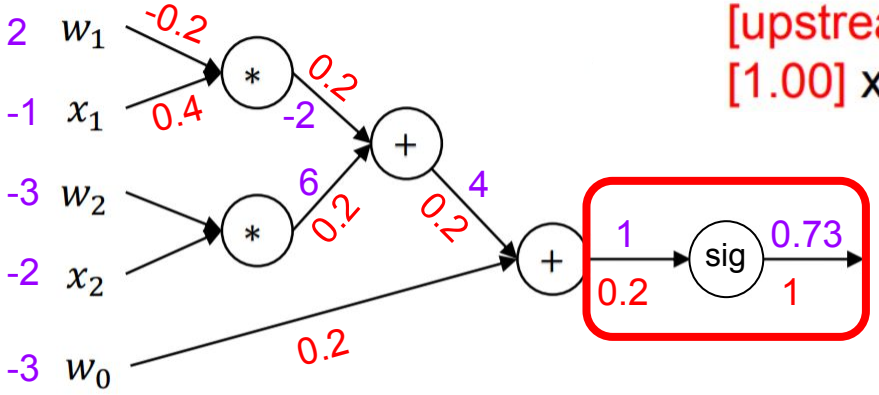


Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

An Example!

$$f = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$



[upstream gradient] x [local gradient]
[1.00] x [(1 - 0.73) (0.73)] = 0.2

Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

Does this work for vector-valued functions?

Does this work for vector-valued functions?

YES!

Use vector derivatives / Jacobians !

Implementation

Implementation

- Create a class for all unique operations in the nodes of computational graph.

Implementation

- Create a class for all unique operations in the nodes of computational graph.

- Define the following methods for each operation class:
  - Forward: Computes result of operation. Saves any data needed for gradient calculation.
  - Backward: Computes local gradient and multiplies it with the upstream gradient.

Example Implementation of Node/Gate

```
class ReLU:
    def __init__(self):
        self.type = "relu"
        self.input = None

    def forward(self, inputs):
        self.input = inputs
        return np.clip(inputs, a_min = 0, a_max = np.inf)

    def backward(self, upstream_grad):
        return (self.input>0).astype(int)*upstream_grad
```

Example Implementation of Node/Gate

```
class ReLU:
    def __init__(self):
        self.type = "relu"
        self.input = None

    def forward(self, inputs):
        self.input = inputs
        return np.clip(inputs, a_min = 0, a_max = np.inf)

    def backward(self, upstream_grad):
        return (self.input>0).astype(int)*upstream_grad
```

Example Implementation of Node/Gate

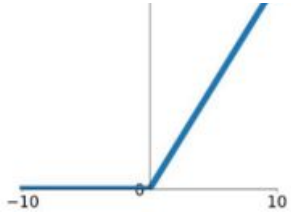
```
class ReLU:
    def __init__(self):
        self.type = "relu"
        self.input = None

    def forward(self, inputs):
        self.input = inputs
        return np.clip(inputs, a_min = 0, a_max = np.inf)

    def backward(self, upstream_grad):
        return (self.input>0).astype(int)*upstream_grad
```

ReLU

$$\max(0, x)$$



Example Implementation of Node/Gate

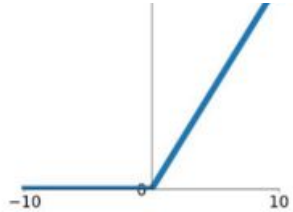
```
class ReLU:
    def __init__(self):
        self.type = "relu"
        self.input = None

    def forward(self, inputs):
        self.input = inputs
        return np.clip(inputs, a_min = 0, a_max = np.inf)

    def backward(self, upstream_grad):
        return (self.input>0).astype(int)*upstream_grad
```

ReLU

$$\max(0, x)$$



Tensorflow/PyTorch have similar implementations

Forward

```
// Functor used by ReluOp to do the computations.
template <typename Device, typename T>
struct Relu {
  // Computes Relu activation.
  //
  // features: any shape.
  // activations: same shape as "features".
  void operator()(const Device& d, typename TTypes<T>::ConstTensor features,
                  typename TTypes<T>::Tensor activations) {
    activations.device(d) = features.cwiseMax(static_cast<T>(0));
  }
};
```

Backward

```
// Functor used by ReluGradOp to do the computations.
template <typename Device, typename T>
struct ReluGrad {
  // Computes ReluGrad backprops.
  //
  // gradients: gradients backpropagated to the Relu op.
  // features: either the inputs that were passed to the Relu or, or its
  //           outputs (using either one yields the same result here).
  // backprops: gradients to backpropagate to the Relu inputs.
  void operator()(const Device& d, typename TTypes<T>::ConstTensor gradients,
                  typename TTypes<T>::ConstTensor features,
                  typename TTypes<T>::Tensor backprops) {
    // NOTE: When the activation is exactly zero, we do not propagate the
    // associated gradient value. This allows the output of the Relu to be used,
    // as well as its input.
    backprops.device(d) =
        gradients * (features > static_cast<T>(0)).template cast<T>();
  }
};
```

Tensorflow Source Code File : https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/kernels/relu_op_functor.h

Tensorflow/PyTorch have similar implementations

Forward

```
// Functor used by ReluOp to do the computations.
template <typename Device, typename T>
struct Relu {
  // Computes Relu activation.
  //
  // features: any shape.
  // activations: same shape as "features".
  void operator()(const Device& d, typename TTypes<T>::ConstTensor features,
                  typename TTypes<T>::Tensor activations) {
    activations.device(d) = features.cwiseMax(static_cast<T>(0));
  }
};
```

features.cwiseMax(0) compares each element of features with 0 and takes the max value.

Backward

```
// Functor used by ReluGradOp to do the computations.
template <typename Device, typename T>
struct ReluGrad {
  // Computes ReluGrad backprops.
  //
  // gradients: gradients backpropagated to the Relu op.
  // features: either the inputs that were passed to the Relu or, or its
  //           outputs (using either one yields the same result here).
  // backprops: gradients to backpropagate to the Relu inputs.
  void operator()(const Device& d, typename TTypes<T>::ConstTensor gradients,
                  typename TTypes<T>::ConstTensor features,
                  typename TTypes<T>::Tensor backprops) {
    // NOTE: When the activation is exactly zero, we do not propagate the
    // associated gradient value. This allows the output of the Relu to be used,
    // as well as its input.
    backprops.device(d) =
        gradients * (features > static_cast<T>(0)).template cast<T>();
  }
};
```

Tensorflow Source Code File : https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/kernels/relu_op_functor.h

Tensorflow/PyTorch have similar implementations

Forward

```
// Functor used by ReluOp to do the computations.
template <typename Device, typename T>
struct Relu {
  // Computes Relu activation.
  //
  // features: any shape.
  // activations: same shape as "features".
  void operator()(const Device& d, typename TTypes<T>::ConstTensor features,
                  typename TTypes<T>::Tensor activations) {
    activations.device(d) = features.cwiseMax(static_cast<T>(0));
  }
};
```

Backward

```
// Functor used by ReluGradOp to do the computations.
template <typename Device, typename T>
struct ReluGrad {
  // Computes ReluGrad backprops.
  //
  // gradients: gradients backpropagated to the Relu op.
  // features: either the inputs that were passed to the Relu or, or its
  //           outputs (using either one yields the same result here).
  // backprops: gradients to backpropagate to the Relu inputs.
  void operator()(const Device& d, typename TTypes<T>::ConstTensor gradients,
                  typename TTypes<T>::ConstTensor features,
                  typename TTypes<T>::Tensor backprops) {
    // NOTE: When the activation is exactly zero, we do not propagate the
    // associated gradient value. This allows the output of the Relu to be used,
    // as well as its input.
    backprops.device(d) =
        gradients * (features > static_cast<T>(0)).template cast<T>();
  }
};
```

Local Gradient

Upstream Gradient

Tensorflow Source Code File : https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/kernels/relu_op_functor.h

Implementation of Graph (Pseudocode)

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

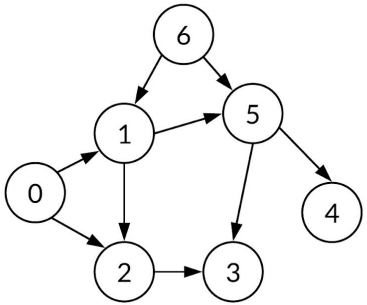
Implementation of Graph (Pseudocode)

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

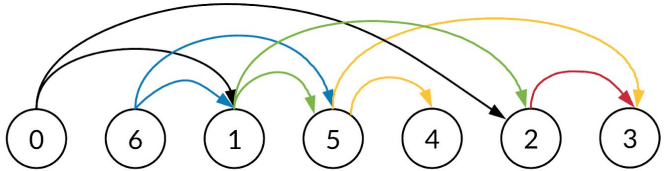
Implementation of Graph (Pseudocode)

Topological Sort gives is a linear ordering of nodes such that for every directed edge **uv** (edge starting from node u pointing to node v), node u comes before v in the ordering.

Unsorted graph



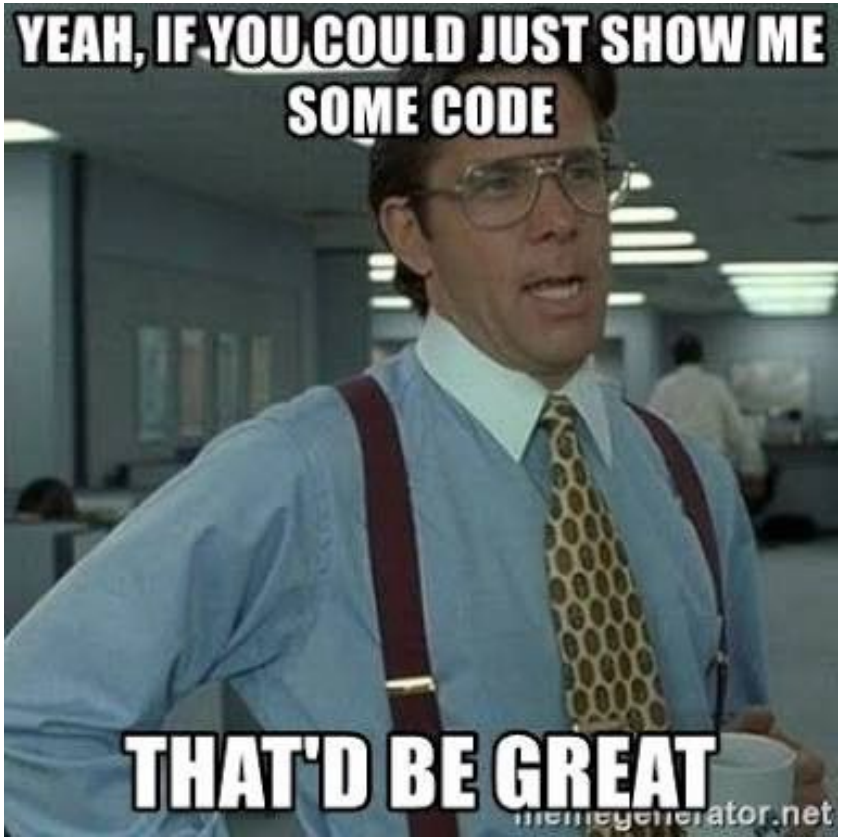
Topologically sorted graph

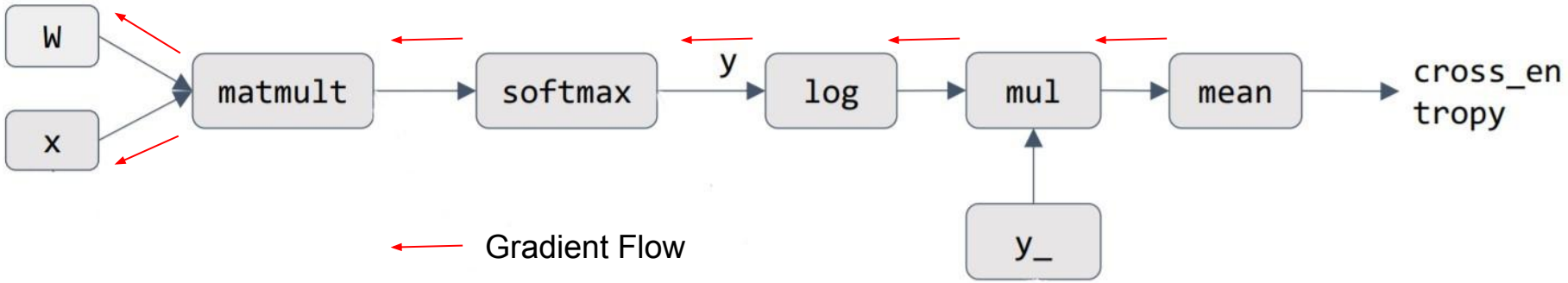


Topological Sort, Codepath : https://guides.codepath.com/compsci/Topological-Sort

Implementation of Graph (Pseudocode)

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```


YEAH, IF YOU COULD JUST SHOW ME SOME CODE
THAT'D BE GREAT



Any Problem with this?

Can we do better?

Problems with backpropagation through graph

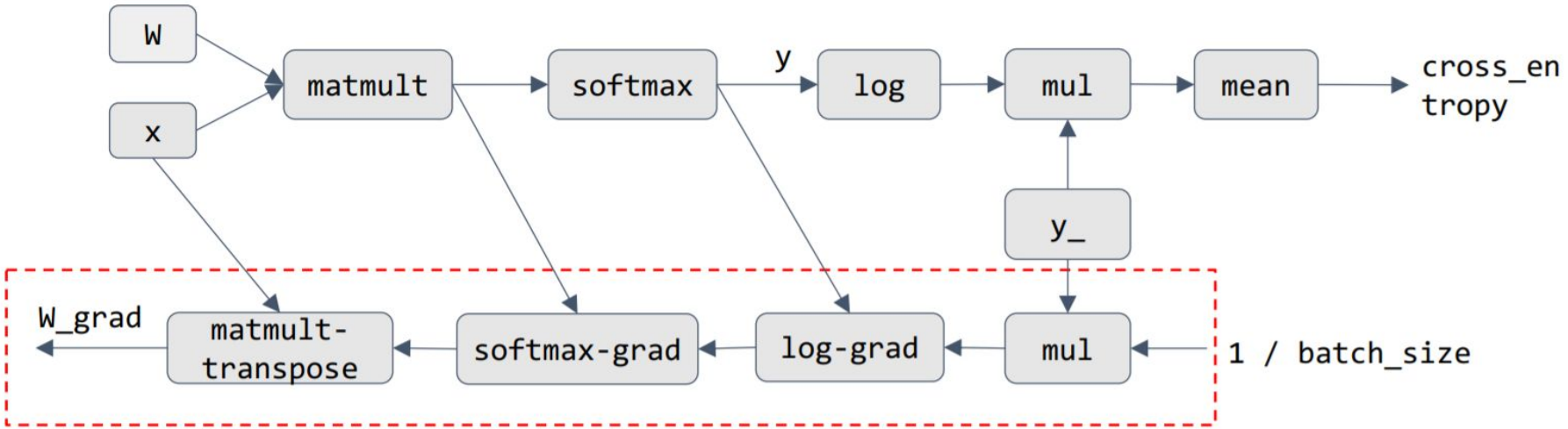
- Always need to keep intermediate data in memory for computing local gradients.

Problems with backpropagation through graph

- Always need to keep intermediate data in memory for computing local gradients.

- Lack of flexibility i.e.compute gradient of gradient?

Automatic Differentiation



- Augment computation graph with nodes for gradient computation.
- Better for memory use and schedule optimization

The technique we saw till now is a
gift from the vast field of
Automatic Differentiation

It is called Reverse-Mode Differentiation

Want to know more?

Automatic Differentiation in Machine Learning: a Survey

Atılım Güneş Baydin GUNES@ROBOTS.OX.AC.UK
Department of Engineering Science
University of Oxford
Oxford OX1 3PJ, United Kingdom

Barak A. Pearlmutter BARAK@PEARLMUTTER.NET
Department of Computer Science
National University of Ireland Maynooth
Maynooth, Co. Kildare, Ireland

Alexey Andreyevich Radul AXCH@MIT.EDU
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139, United States

Jeffrey Mark Siskind QOBI@PURDUE.EDU
School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47907, United States