CS 584 Natural Language Processing

Logistic Regression, Gradient Descent,
Neural Networks

Department of Computer Science
Yue Ning
yue.ning@stevens.edu

Late Submission Policy

- **10%** penalty for late submission within **<u>24 hours</u>**.
- **40%** penalty for late submissions within **<u>24-48 hours</u>**.
- After 48 hours, you get **NO** points on the assignment.

Classification

- Supervised vs. Unsupervised learning
- Generally we have a training dataset consisting of samples

$$\{x_i, y_i\}^N_{i=1}$$

- x_i are inputs (vectors), e.g. words (indices or vectors), sentences, documents, etc.
  - dimension d
- y_i are labels (one of C classes) we try to predict
  - classes: sentiment, named entities, buy/sell decision
  - other words
  - later: multi-word sequences

Feature Vectors

- For document classifications, build a feature vector for each document

$$\{x_i, y_i\}^N_{i=1}$$

- x_i can be:
  - word counts
  - TF-IDF
  - ...etc

What is TF-IDF

- Term frequency - inverse document frequency
    - first define a vocabulary with length V
    - for each word in the vocabulary, calculate the vector for current document:

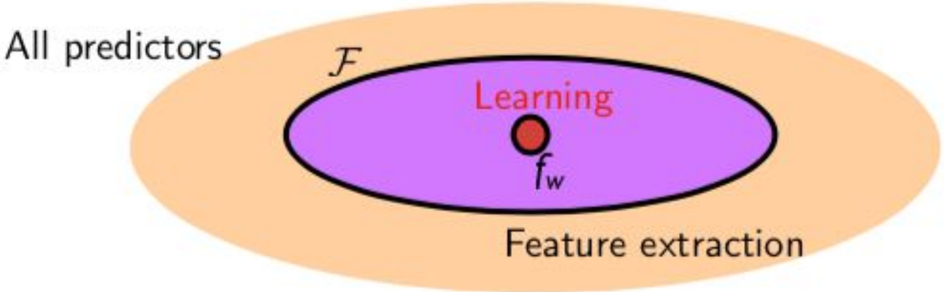
      [TF_1/ DF_1,......, TF_v/ DF_v, ……, TF_V/ DF_V]

    - where **TF** is the frequency of word v appearing in the current document and **DF** is the # of documents where word v appears.

https://en.wikipedia.org/wiki/Tf%E2%80%93idf

The Learning Problem

- **Hypothesis class**: we consider some restricted set F of mappings f : X → Y from input data to output.



- **Estimation**: on the basis of a training set of examples of their labels, $\{x_i, y_i\}^N_{i=1}$ we find an estimate $\hat{f} \in F$

- **Evaluation**: We measure how well the estimate generalize to yet unseen examples.

Training vs Validation vs Testing

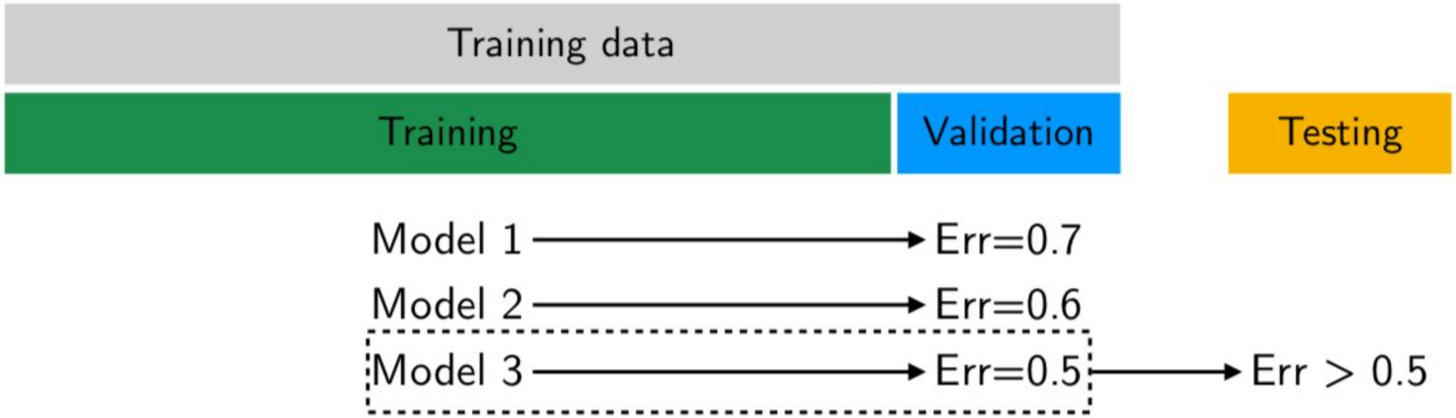
- Training dataset: $\{x_i, y_i\}^N_{i=1}$ The sample of data used to fit the model.

- Validation dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.

- Testing dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

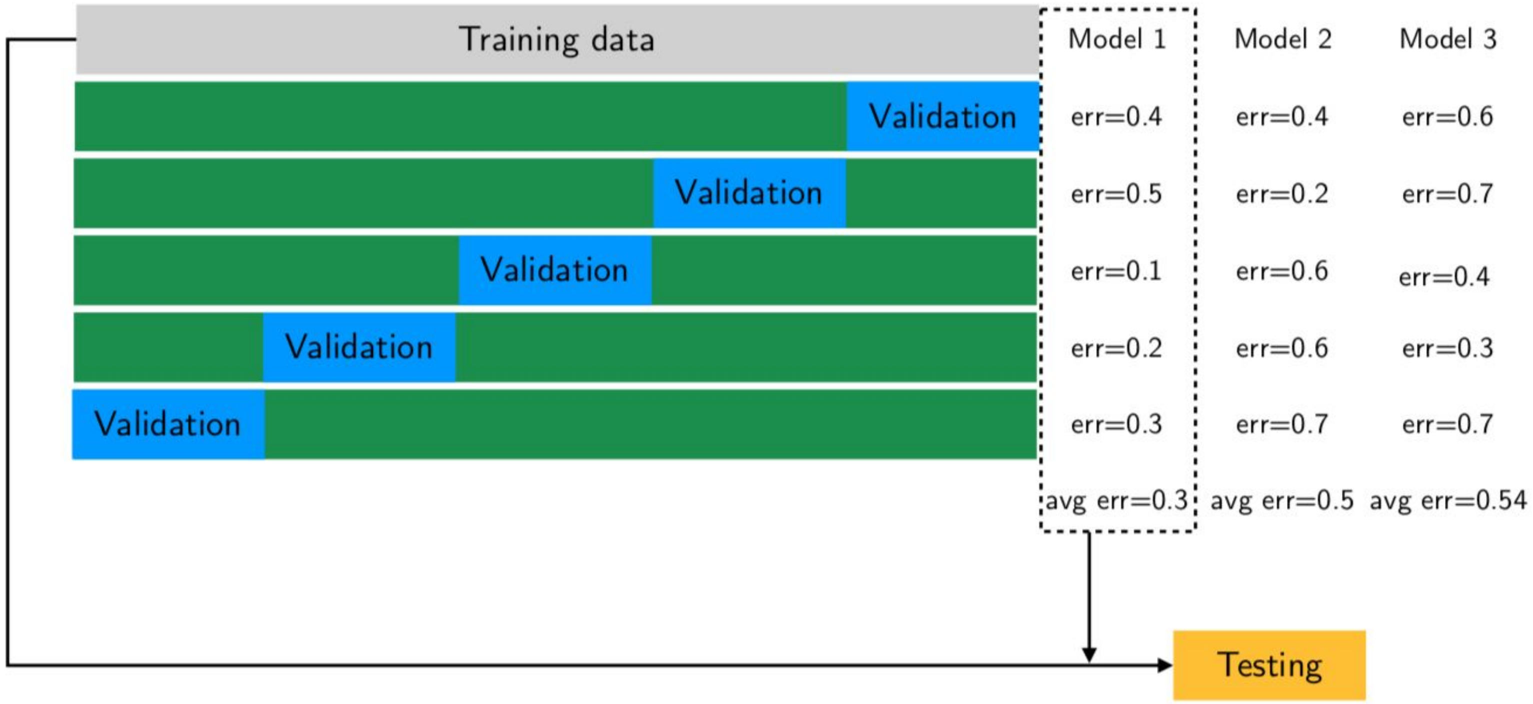
Cross-Validation

- Cross-validation: it allows us to estimate the generalization error based on training examples alone.



K-fold Cross-Validation

- K-fold Cross-validation: the original sample is randomly partitioned into k equal sized subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data.

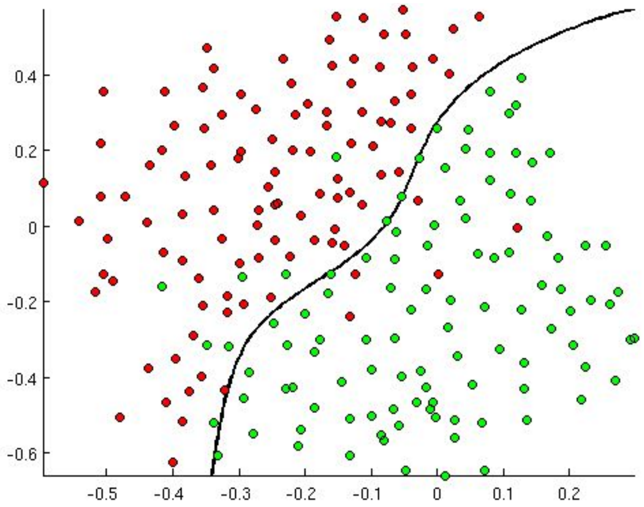


Loss function

- A loss function ***Loss(x,y,w)*** quantifies how wrong you would be if you used w to make a prediction on x when the correct output is y. It is the object we want to minimize.
- Loss is a function of **the parameters w** and we can try to minimize it directly.
- We reduce the estimation problem to a **minimization** problem.

Classification intuition

- Training dataset: $\{x_i, y_i\}^N_{i=1}$
- Traditional ML/Stats approaches: assume x_i are fixed, train a model (with weights w) to determine a decision boundary (hyperplane) as in this picture



Logistic Regression

- Prediction: For each x_i, predict if x_i belongs to class k:

$$p(C_k|\mathbf{x}_i) = \frac{\exp(\mathbf{w}_k \mathbf{x}_i)}{\sum_{c=1}^{C} \exp(\mathbf{w}_c \mathbf{x}_i)}$$

- We can consider this prediction function to be 2 steps:
  1. Take the inner product of w_k and x_i, compute f_c for c =1,...,C

$$\mathbf{w}_k \mathbf{x}_i = \sum_{j=1}^{d} \mathbf{w}_{kj} \mathbf{x}_{ij} = f_k$$

  2. Apply softmax function to get normalized probability:

$$p(C_k|\mathbf{x}_i) = \frac{\exp f_k}{\sum_{c=1}^{C} \exp f_c} = \mathrm{softmax}(f_k)$$

Logistic Regression - Training

- For each training example (x,y), our objective is to maximize the probability of correct class y

$$\prod_{i=1}^{N} \prod_{k=1}^{C} p(C_k|\mathbf{x}_i)^{y_{ik}}$$

- Or we can minimize the negative log probability of that class:

$$-y_{ik} \log p(C_k|\mathbf{x}) = -y_{ik} \log \left(\frac{\exp f_k}{\sum_{c=1}^{C} \exp f_c} \right)$$

What is “cross entropy” loss/error?

- From information theory:
  - let the true probability distribution be p
  - let out computed model probability be q
  - The cross entropy is:

$$H(p, q) = -\sum_{c=1}^{C} p(c) \log q(c)$$

- Assuming a ground truth probability distribution that is 1 at the right class and 0 else: p = [0,...,0,1,...0]
- Because of one-hot p, the only term left is the negative log probability of the true class

Classification on a full dataset

- Cross entropy loss over a full dataset $\{x_i, y_i\}^N_{i=1}$

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log\left(\frac{\exp f_k}{\sum_{c=1}^{C}\exp f_c}\right)$$

- Instead of

$$f_k = f_k(\mathbf{x}) = \mathbf{w}_k\mathbf{x} = \sum_{j=1}^{d} w_{kj}\mathbf{x}_j$$

- We can write f in matrix operation:

$$f = W\mathbf{x}$$

Gradient Descent

- Gradient: the direction that increases the loss the most

$$\nabla_{\mathbf{w}} J$$

- Algorithm: gradient descent
  - Initialize w
  - For t in 1, … , T (epoches):

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \mathrm{J}(\mathbf{w})}_{\text{gradient}}$$

Gradient Descent is slow

Algorithm: gradient descent

- Initialize w
- For t in 1, … , T (epoches):

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \mathrm{J}(\mathbf{w})}_{\text{gradient}}$$

$$\frac{\partial}{\partial \mathbf{w}_j} J = \frac{\partial}{\partial \mathbf{w}_j} \Big[- \sum_{n=1}^{N} \sum_{k=1}^{K} y_{nk} \ln\, P(C_k|\mathbf{x}_n) \Big]$$

$$\frac{\partial}{\partial \mathbf{w}_j} J = \sum_{n=1}^{N} (P(C_j|\mathbf{x}_n) - y_{nj}) \mathbf{x}_n$$

Stochastic Gradient Descent

Algorithm: stochastic gradient descent

- Initialize w
- For t in 1, … , T (epoches):
  - Randomly shuffle the data
  - For (x_n, y_n) in training data:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} J(\mathbf{x}_n, y_n)$$

- Allow for online update with new examples.
- With a high variance that cause the objective function to fluctuate heavily

Mini-batch Gradient Descent

Algorithm: mini-batch gradient descent

- Initialize w
- For t in 1, … , T (epoches):
  - Randomly sample a batch

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \nabla_{\mathbf{w}} J(\text{batch})$$

- Reduces the variance of the parameter updates, which can lead to more stable convergence;
- Can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient.

Traditional ML optimization

- For general machine learning parameter

$$\theta = \begin{bmatrix} W_{\cdot 1} \\ . \\ . \\ . \\ W_{\cdot d} \end{bmatrix} = W(:) \in R^{d \times C}$$

- We only update the decision boundary via:

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla W_{\cdot 1} \\ . \\ . \\ . \\ \nabla W_{\cdot d} \end{bmatrix} \in R^{d \times C}$$

Chain rule

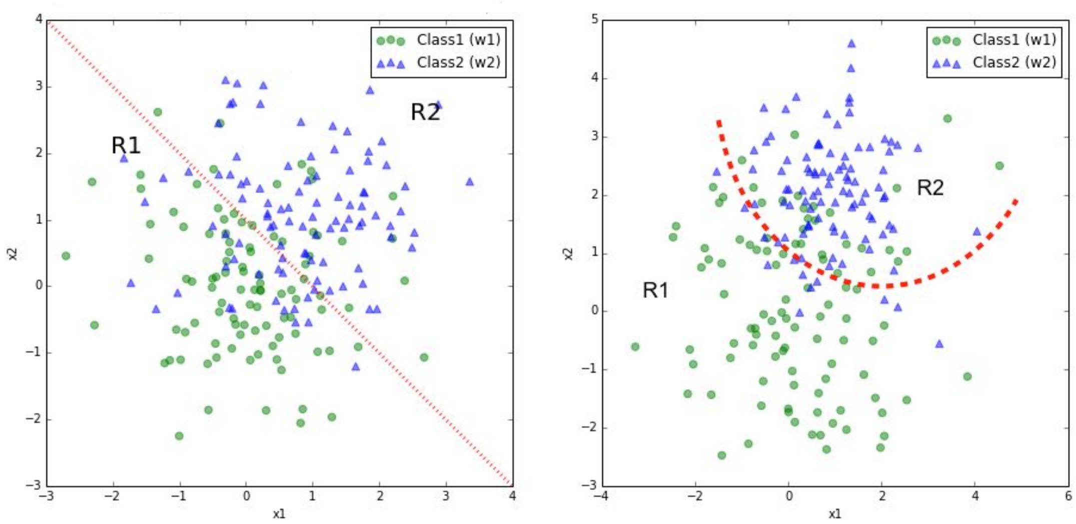
- In general, y = f(u), u = g(x) what is the derivative of y w.r.t x?

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

- Example: $y = e^{x2}$

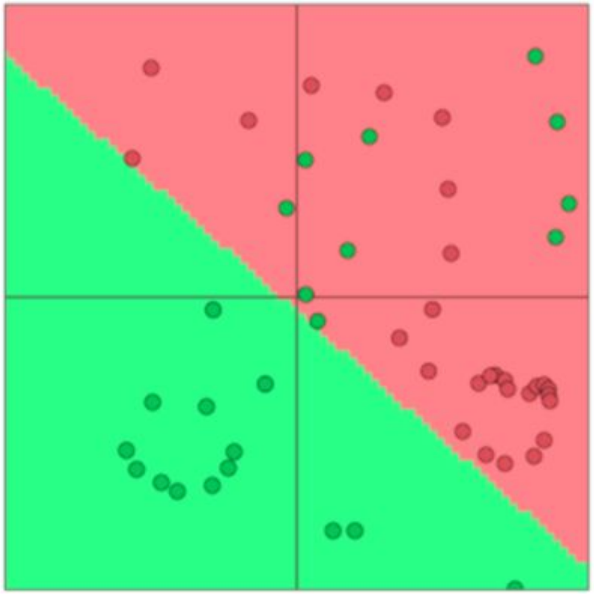
Neural network classifiers

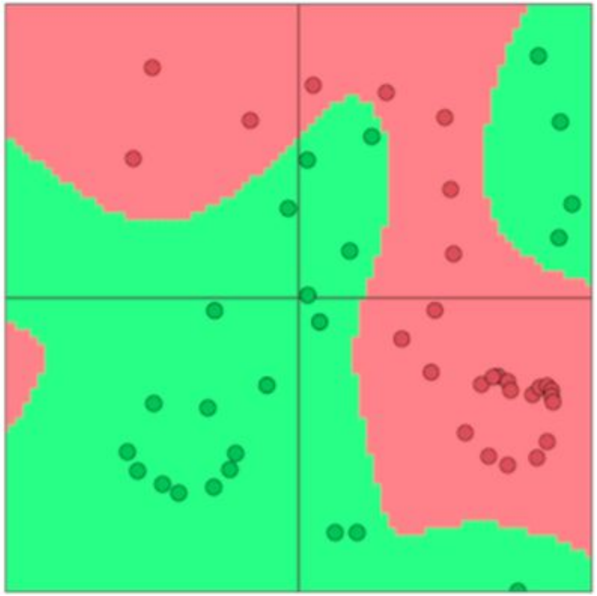
- Softmax (logistic regression~one layer neural nets) alone not very powerful
- Logistic regression gives only linear decision boundaries
  - limited
  - unhelpful when a problem is complex



Neural network classifiers

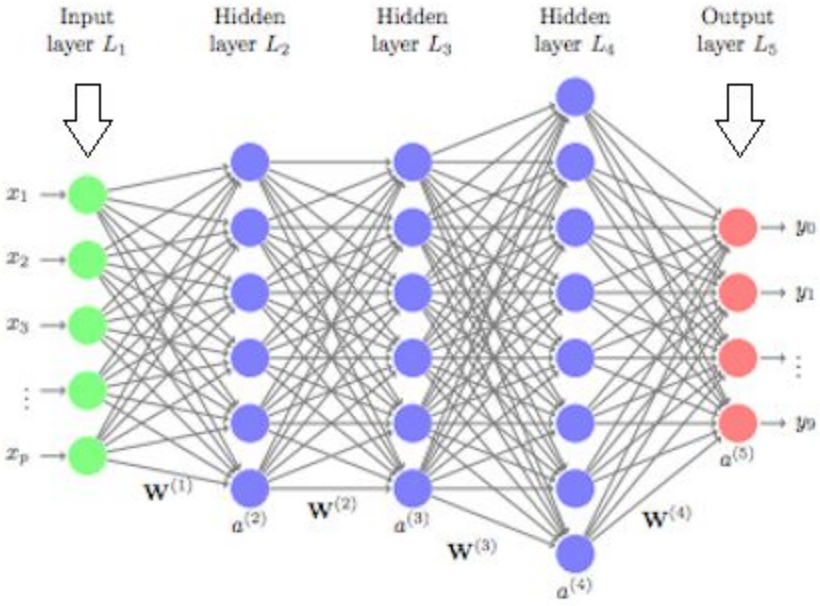
- Neural networks can learn much more complex functions and non linear decision boundaries!





What is a Neural Network?

- Often associated with biological devices (brains), devices, or network diagrams;
- But the best conceptualization for this presentation is none of these: think of a neural network as a mathematical function.



The pros of Neural Networks

- Successfully used on <u>a variety of domains</u>: computer vision, speech recognition, gaming etc.
- Can provide solutions to very complex and nonlinear problems;
- If provided with sufficient amount of data, can solve classification and forecasting problems accurately and easily
- Once trained, prediction is fast;

A simple toy example

- Input: position of two oncoming cars $x = [x_1, x_2]$
- output: whether safe ($y=+1$)or collide ($y=-1$)
- True function: safe if cars are far(at least 1)

$$y = \mathrm{sign}(|x_1-x_2|-1)$$

Decomposing the problem

- Test if car 1 is far right of car 2:
  $$h_1 = I(x_1 - x_2 > 1)$$
- test if car 2 is far right of car 1:
  $$h_2 = I(x_2 - x_1 > 1)$$
- safe if at least one condition is true:
  $$y = \text{sign}(h_1 + h_2)$$

$\mathbf{x}$	h_1	h_2	y
$[1, 3]^T$	0	1	+1
$[3, 1]^T$	1	0	+1
$[1, 0.5]^T$	0	0	-1

Learning strategy

- Define $x=[1,x_1,x_2]$
- Intermediate hidden subproblems:

$$h_1 = I(v_1 x>0) \quad v_1=[-1, +1, -1]$$

$$h_2 = I(v_2 x>0) \quad v_2=[+1, -1, -1]$$

- Final prediction

$$f(x) = \text{sign}(w_1 h_1+w_2 h_2) \quad w=[1,1]$$

Key idea: joint learning, learn both hidden subproblems and combination weights

Gradients

- Problem: gradient of h_1 with respect to v_1 is 0

  $h_1 = I(v_1x>0)$ $v_1=[-1, +1, -1]$

- Use logistic function maps $(-\infty, +\infty)$ to (0,1)

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

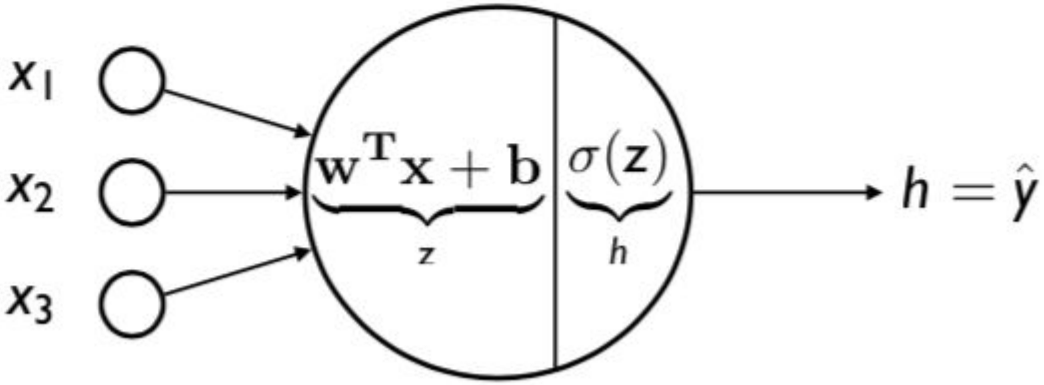
- Derivative of logistiction function

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$

- Solution: $h_j = \sigma(\mathbf{v}_j\mathbf{x})$

No hidden units: logistic regression

- Sigmoid activation function



- Output score: sigmoid/softmax function

Logistic regression

- Binary cross-entropy loss:

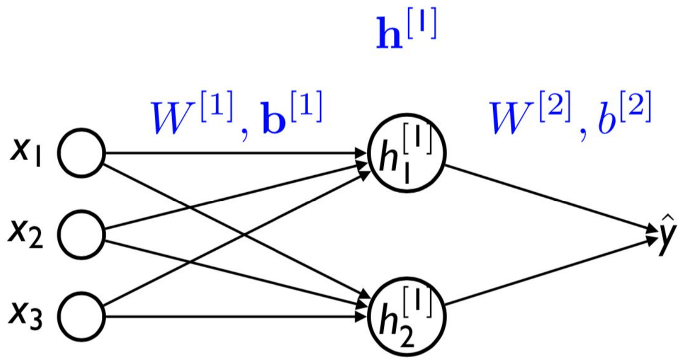
$$J(\theta) = \sum_{i=1}^{N} -y_i \log p_i - (1 - y_i) \log(1 - p_i)$$

- Gradient descent algorithm:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} J$$

Neural networks: one hidden layer

- One hidden layer:



- Hidden layer representation

$$z_1^{[1]} = \mathbf{w}_1^{[1]T}\mathbf{x} + b_1^{[1]}, h_1^{[1]} = \sigma(z_1^{[1]})$$

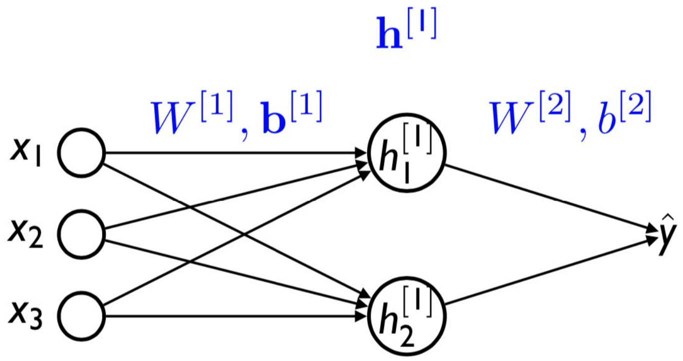
$$z_2^{[1]} = \mathbf{w}_2^{[1]T}\mathbf{x} + b_2^{[1]}, h_2^{[1]} = \sigma(z_2^{[1]})$$

$$\mathbf{z}^{[1]} = \underbrace{W^{[1]}}_{2\times3}\underbrace{\mathbf{x}}_{3\times1} + \underbrace{\mathbf{b}^{[1]}}_{2\times1}$$

$$\hat{y} = \sigma(\mathbf{z}^{[2]})$$

Neural networks: one hidden layer

- One hidden layer:



- output layer

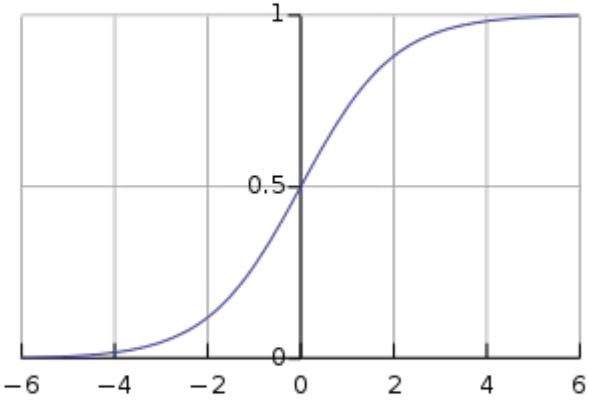
$$z^{[2]} = \underbrace{W^{[2]}}_{1\times 2}\underbrace{\mathbf{h}^{[1]}}_{2\times 1} + b^{[2]}$$

$$\hat{y} = \sigma(z^{[2]})$$

Binary logistic regression neurons

nonlinear activation function (e.g. sigmoid), w = weights, b= bias, h = hidden, x = inputs

$$z_1^{[1]} = \mathbf{w}_1^{[1]^T} \mathbf{x} + b_1^{[1]}, \, h_1^{[1]} = \boxed{\sigma}(z_1^{[1]})$$



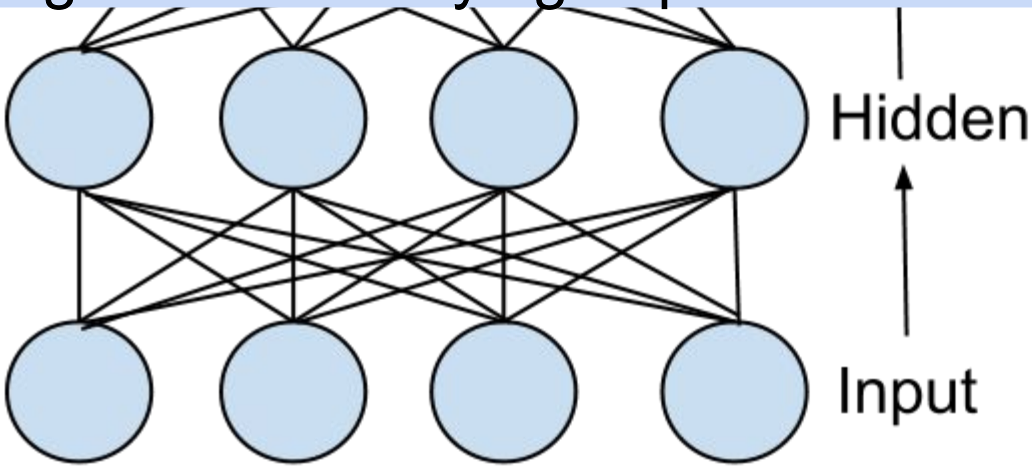
We can have an “always on” features, which gives a class prior, or separate it out, as a bias term.

w_1 and b_1 are the parameters of this neuron

A neural network = running several logistic regression at the same time

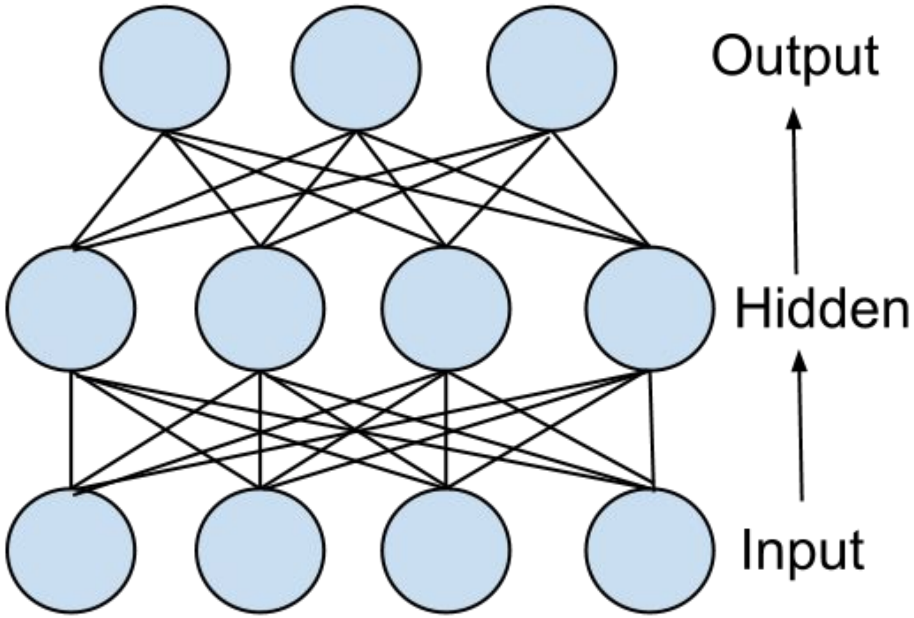
If we feed a vector of inputs through a bunch of logistic functions, then we get a vector of outputs

But we don't have to decide ahead of time what variables these logistic regression are trying to predict



A neural network = running several logistic regression at the same time

The loss function will direct what hidden variables should be, to predict the targets for the output layer



We can feed into another function

Optimization

- Optimization problem:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \text{cross entropy}(\mathbf{x}_i, y_i, \mathbf{W}, \mathbf{b})$$

- Solution: Gradient descent

$$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}, \quad W^{[1]} = W^{[1]} - \eta dW^{[1]}$$

$$d\mathbf{b}^{[1]} = \frac{\partial J}{\partial \mathbf{b}^{[1]}}, \quad \mathbf{b}^{[1]} = \mathbf{b}^{[1]} - \eta d\mathbf{b}^{[1]}$$

$$dW^{[2]} = \frac{\partial J}{\partial W^{[2]}}, \quad W^{[2]} = W^{[2]} - \eta dW^{[2]}$$

$$db^{[2]} = \frac{\partial J}{\partial b^{[2]}}, \quad b^{[2]} = b^{[2]} - \eta db^{[2]}$$

Backpropagation

- Mathematical: just grind through the **chain rule**
- Learn the weights so that the loss is minimized
- It provides an efficient procedure to compute derivatives
    1. Break up equations into simple pieces
    2. Apply the chain rule
    3. Write out the gradients

Computation graphs and backpropagation

- We represent our neural net equations as a graph
  - Source nodes: inputs
  - Interior nodes: operations
  - Edges pass along result of the operation

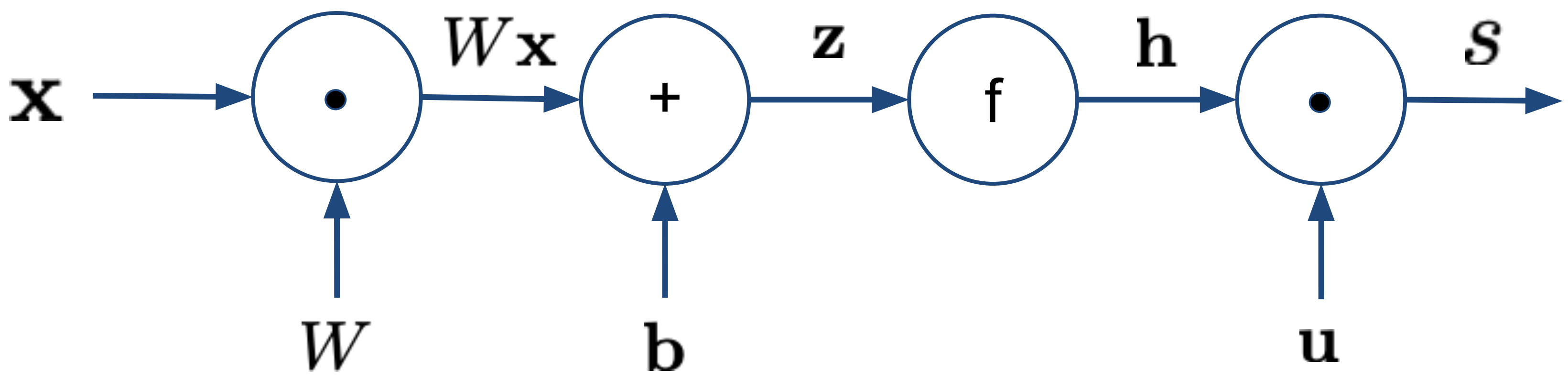
$$s = \mathbf{u}^\top \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{z})$$

$$\mathbf{z} = W\mathbf{x} + \mathbf{b}$$

$$\mathbf{x} \ \text{(input)}$$

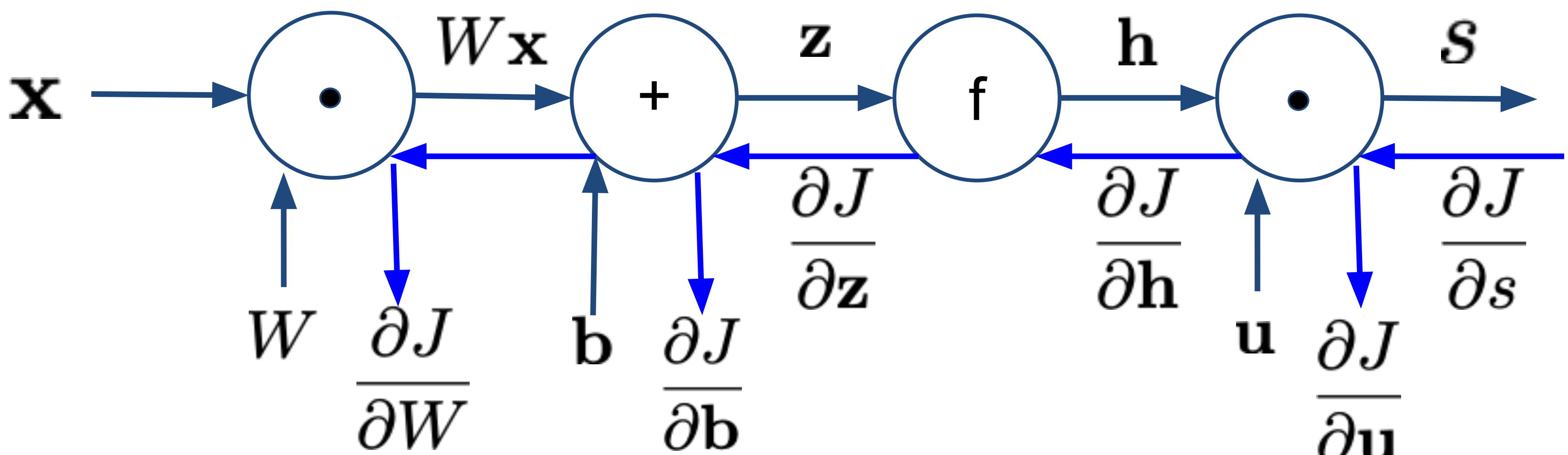
Forward propagation



Backpropagation

- Go backwards along edges
  - Pass along gradients

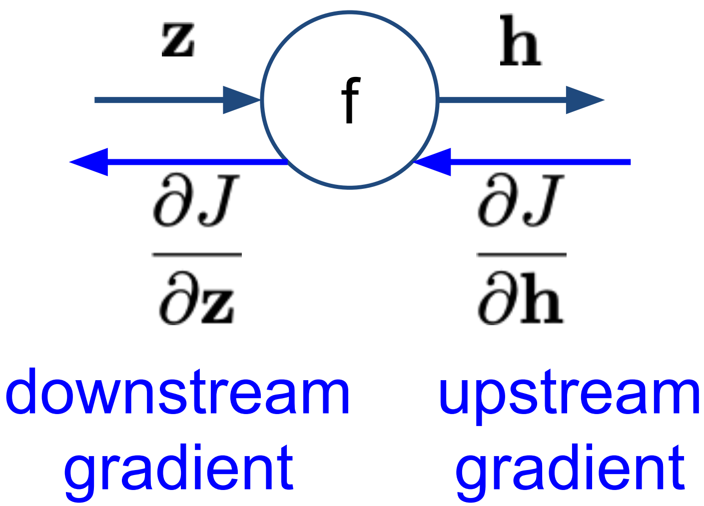
$$s = \mathbf{u}^\top \mathbf{h}$$
$$\mathbf{h} = f(\mathbf{z})$$
$$\mathbf{z} = W\mathbf{x} + \mathbf{b}$$
$$\mathbf{x} \text{ (input)}$$



Backpropagation: single node

- Node receives an “upstream gradient”
- Goal is to pass on the correct “downstream gradient”

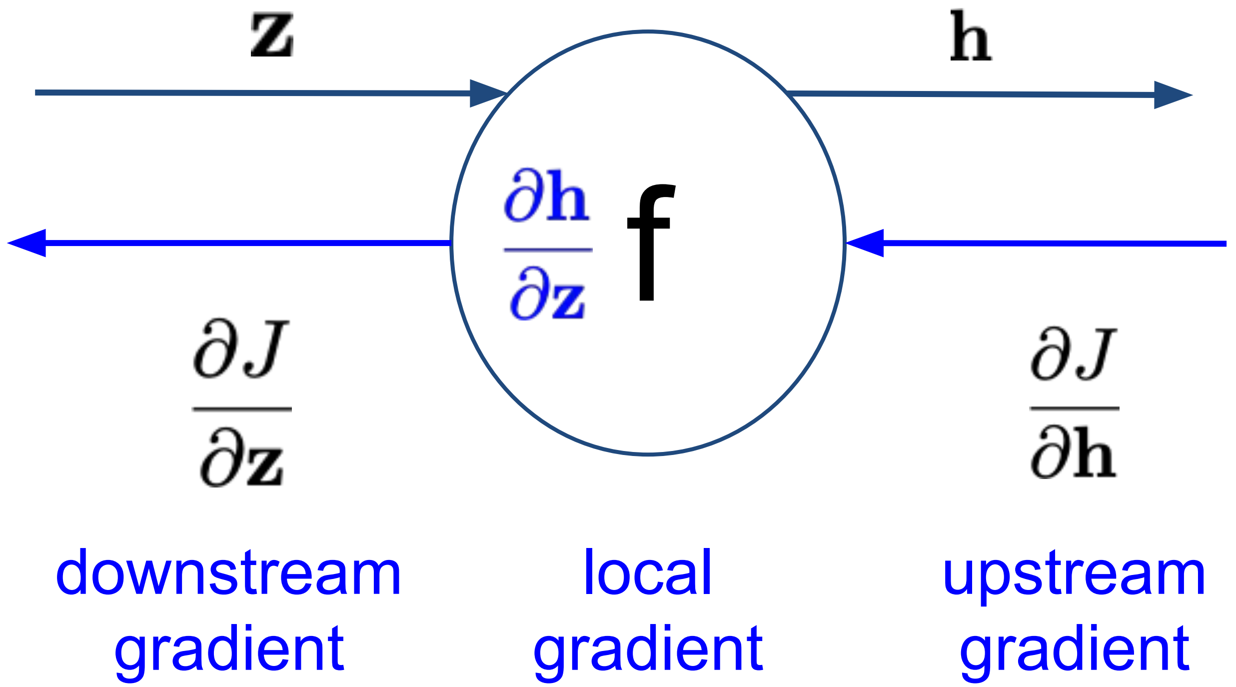
$$\mathbf{h} = f(\mathbf{z})$$



Backpropagation: single node

- Each node has a local gradient
- The gradient of its output with respect to its input

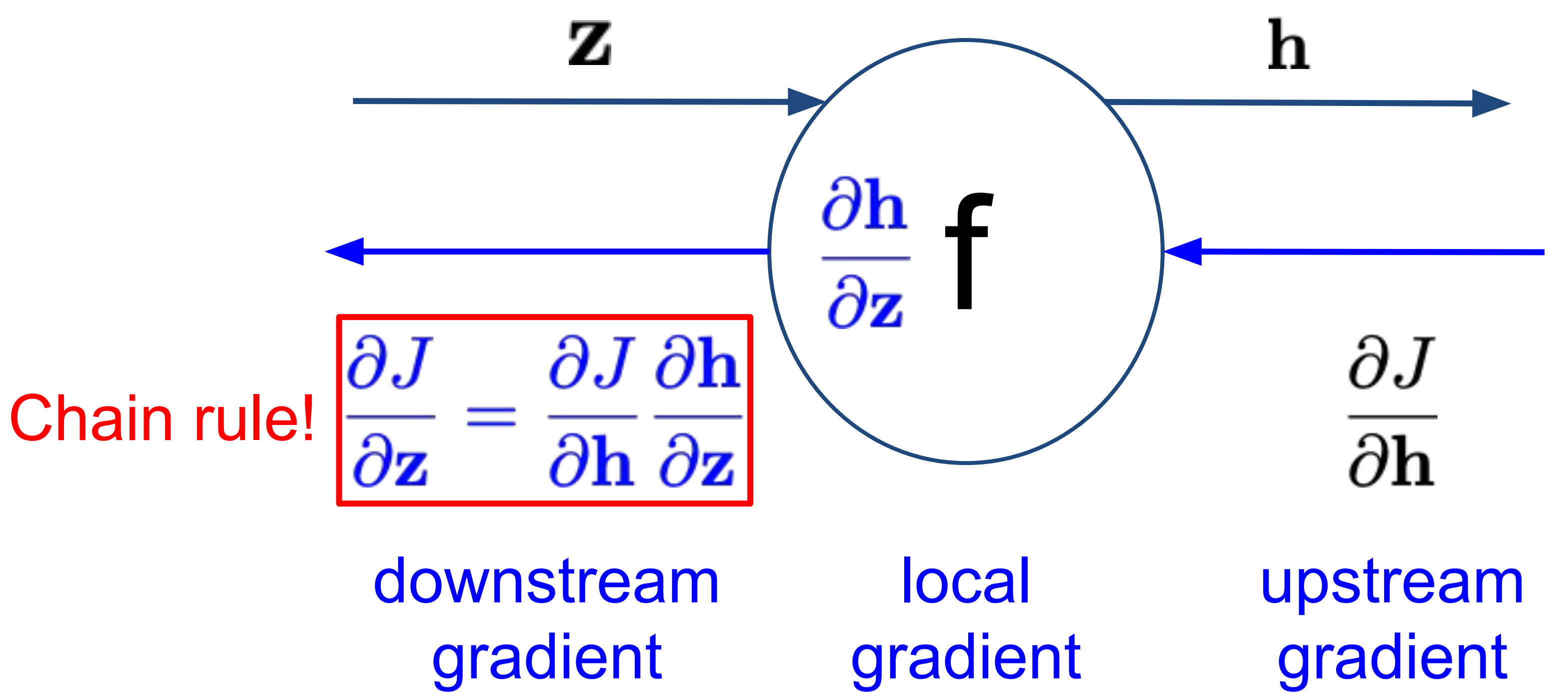
$$\mathbf{h} = f(\mathbf{z})$$



Backpropagation: single node

- Each node has a local gradient
- The gradient of its output with respect to its input

$$\mathbf{h} = f(\mathbf{z})$$

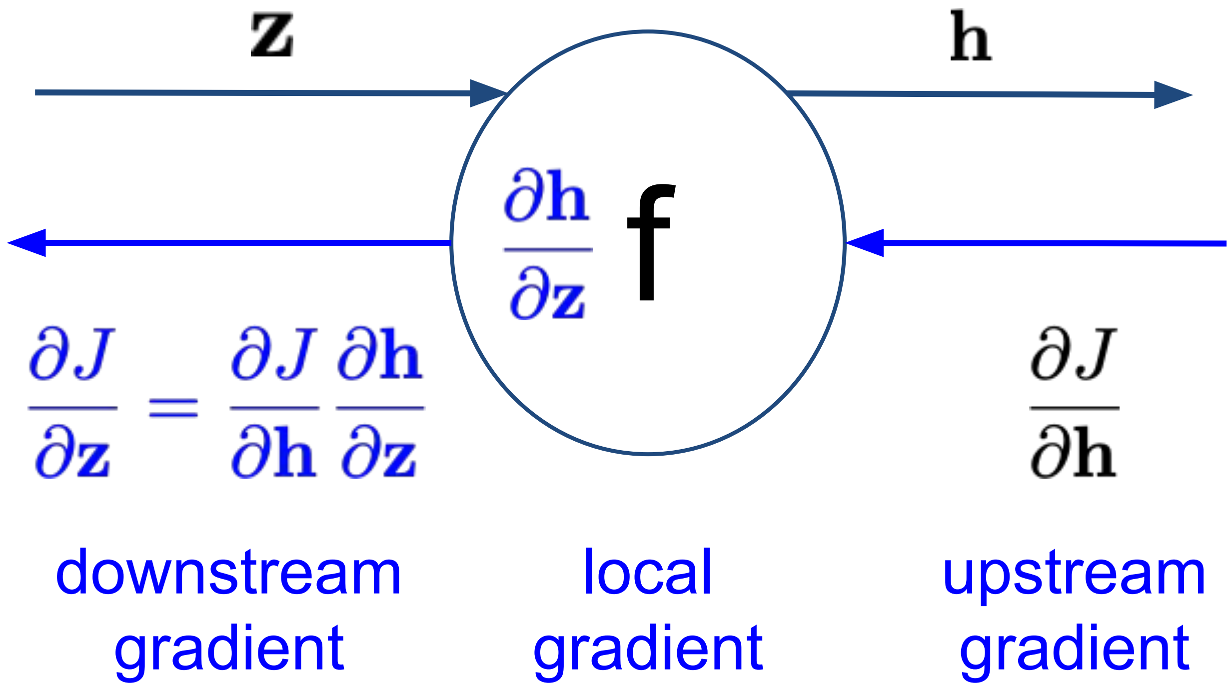


Backpropagation: single node

- Each node has a local gradient
- The gradient of its output with respect to its input

$$\mathbf{h} = f(\mathbf{z})$$

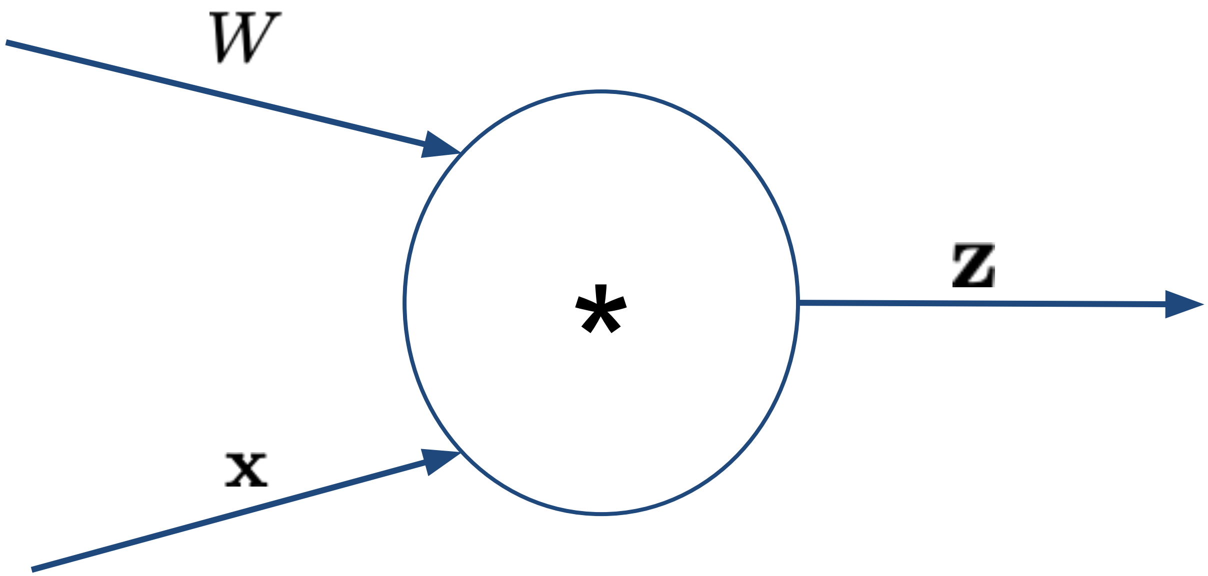
- [downstream gradient] = [upstream gradient]x[local gradient]



Backpropagation: single node

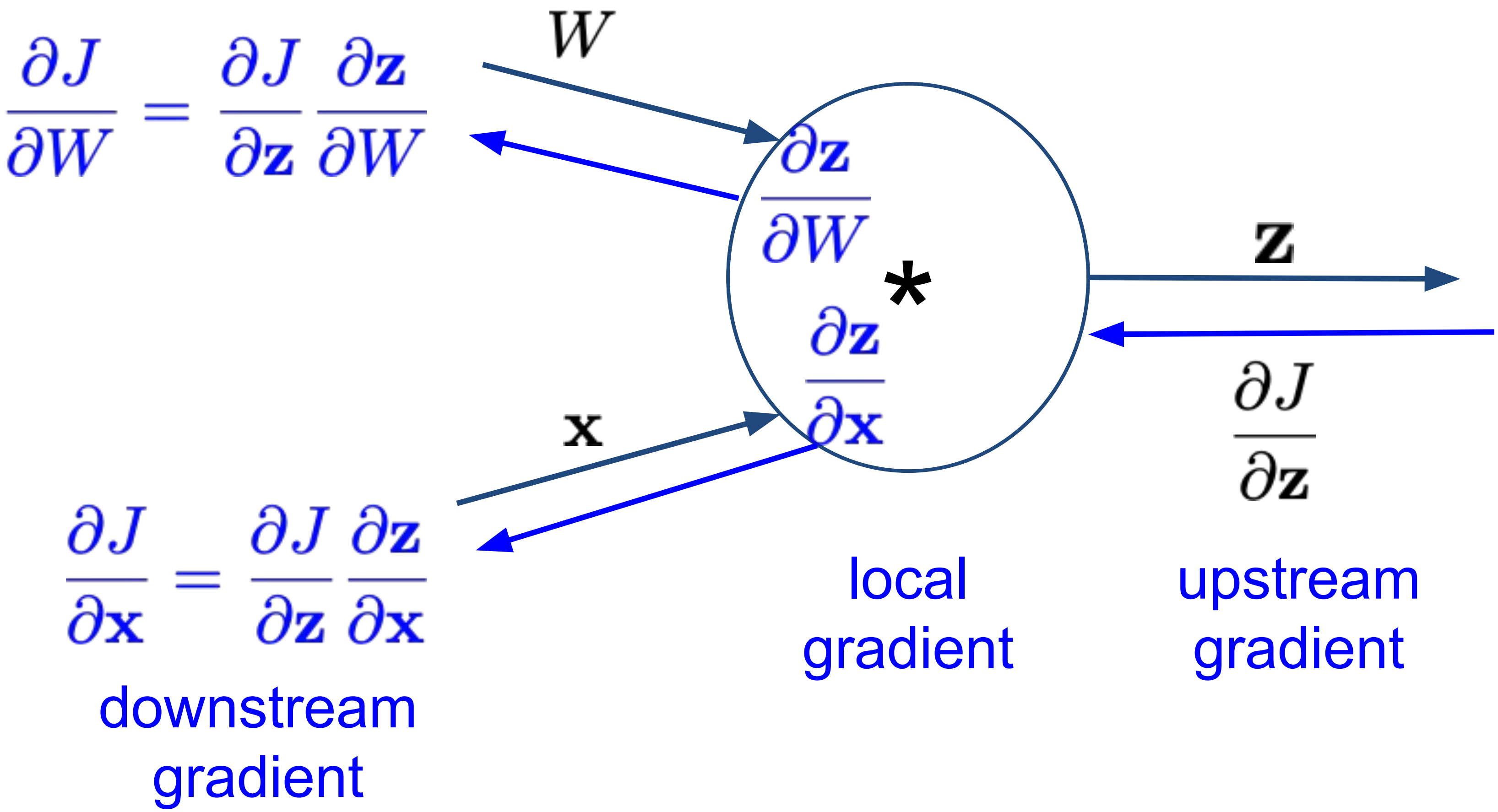
- What about nodes with multiple inputs?

$$\mathbf{z} = W\mathbf{x}$$



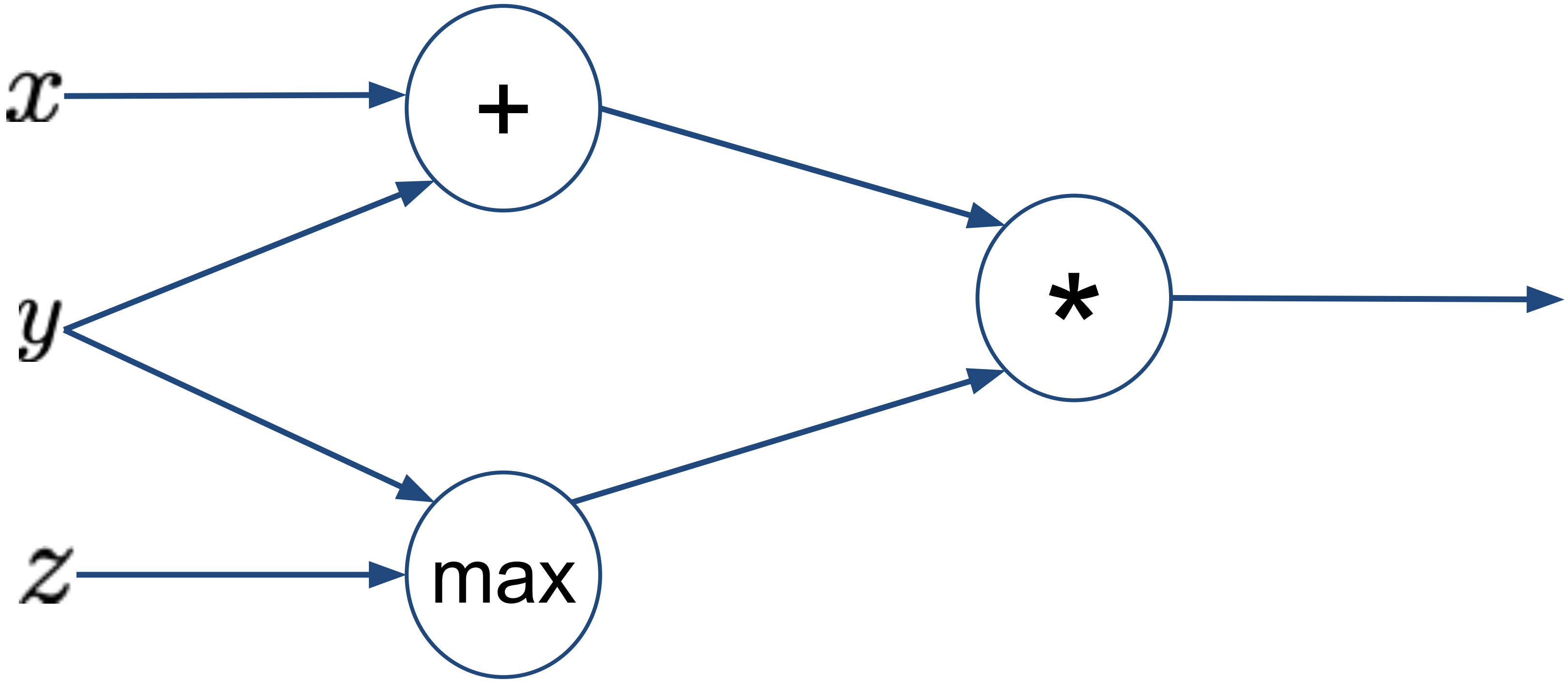
Backpropagation: single node

- multiple inputs -> multiple local gradients $\mathbf{z} = W\mathbf{x}$



An example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

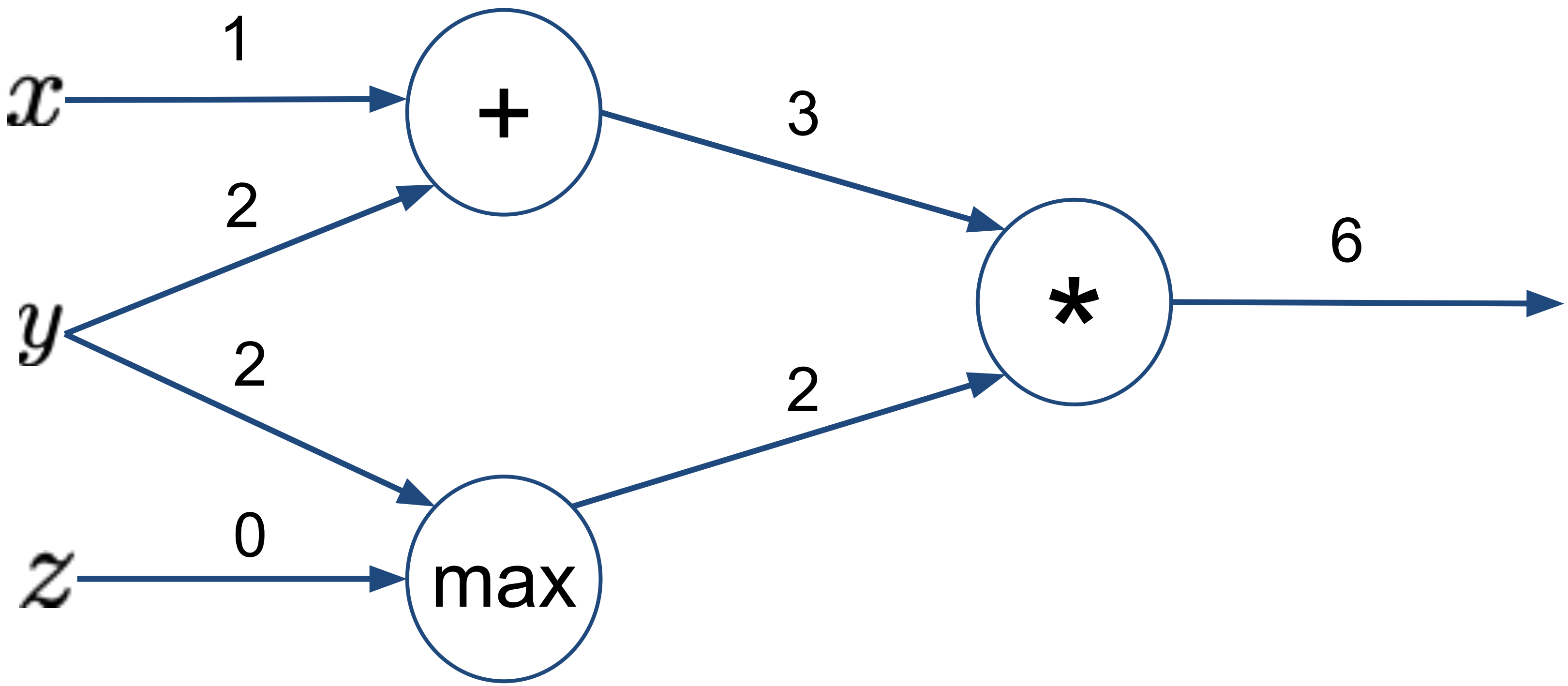


An example

$$f(x, y, z) = (x + y)\max(y, z)$$

$$x = 1, y = 2, z = 0$$

- Forward prop steps:
- a = x+y
- b= max(y,z)
- f = ab



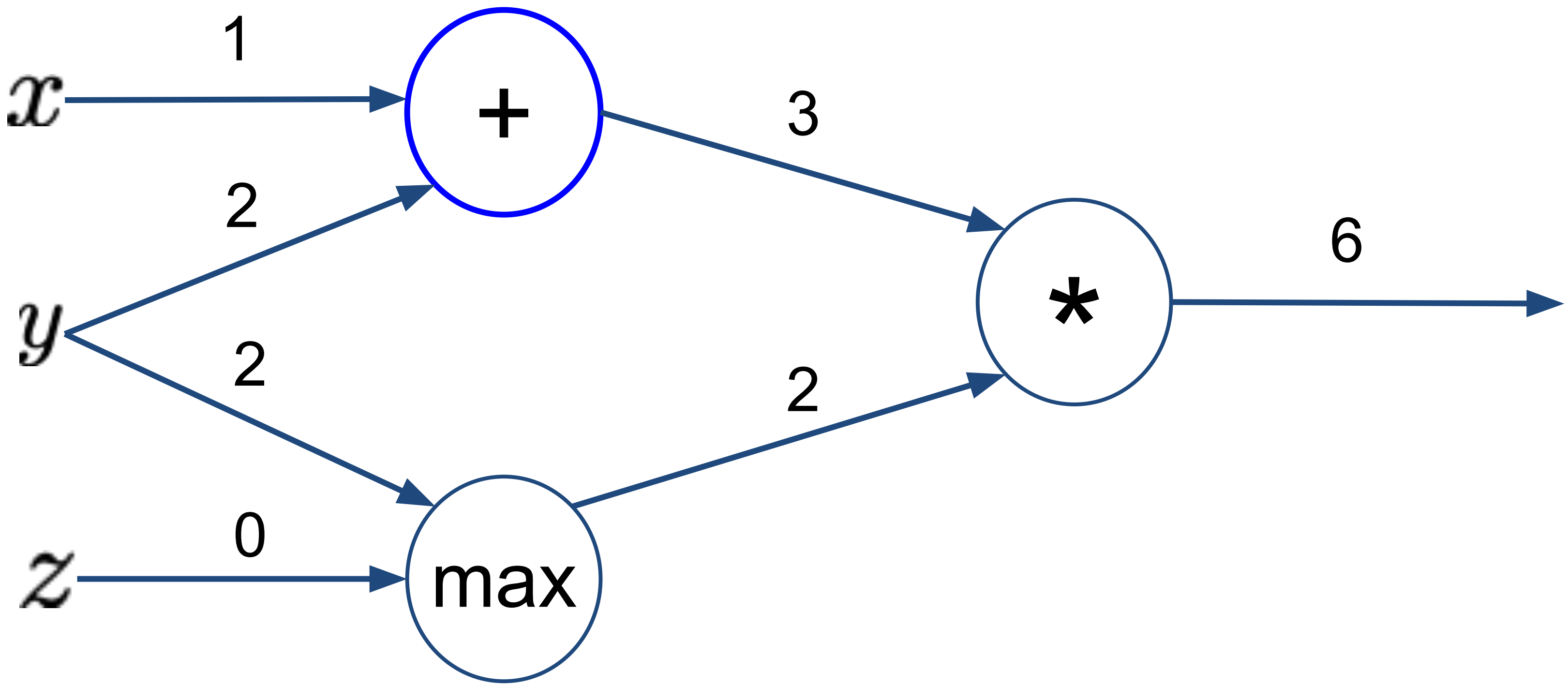
An example

$$f(x,y,z) = (x+y)\max(y,z)$$
$$x = 1, y = 2, z = 0$$

- Forward prop steps:
- a = x+y
- b= max(y,z)
- f = ab

- Local gradients:

$$\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$



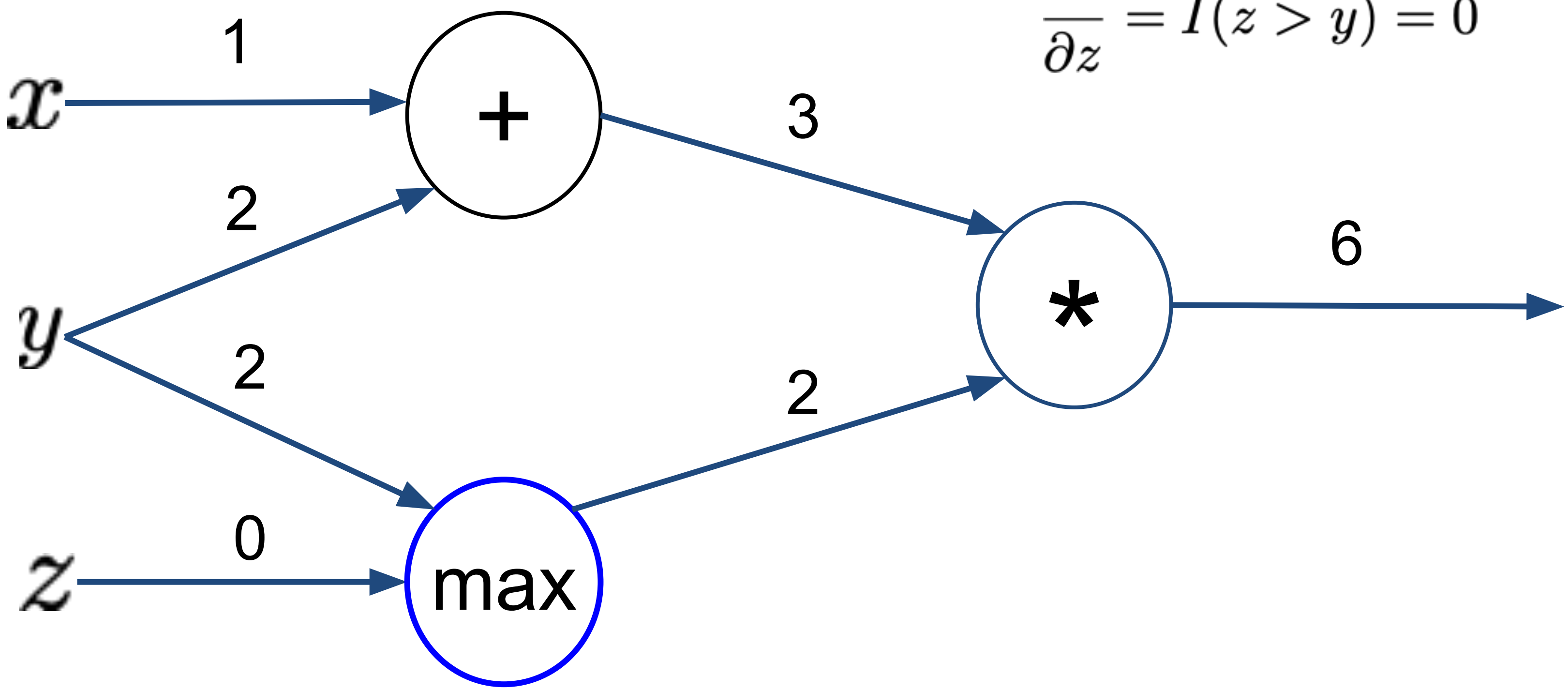
An example

$$f(x,y,z) = (x+y)\max(y,z)$$
$$x = 1, y = 2, z = 0$$

- Forward prop steps:
- a = x+y
- b= max(y,z)
- f = ab

- Local gradients: $\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$

$$\frac{\partial b}{\partial y} = I(y > z) = 1$$
$$\frac{\partial b}{\partial z} = I(z > y) = 0$$



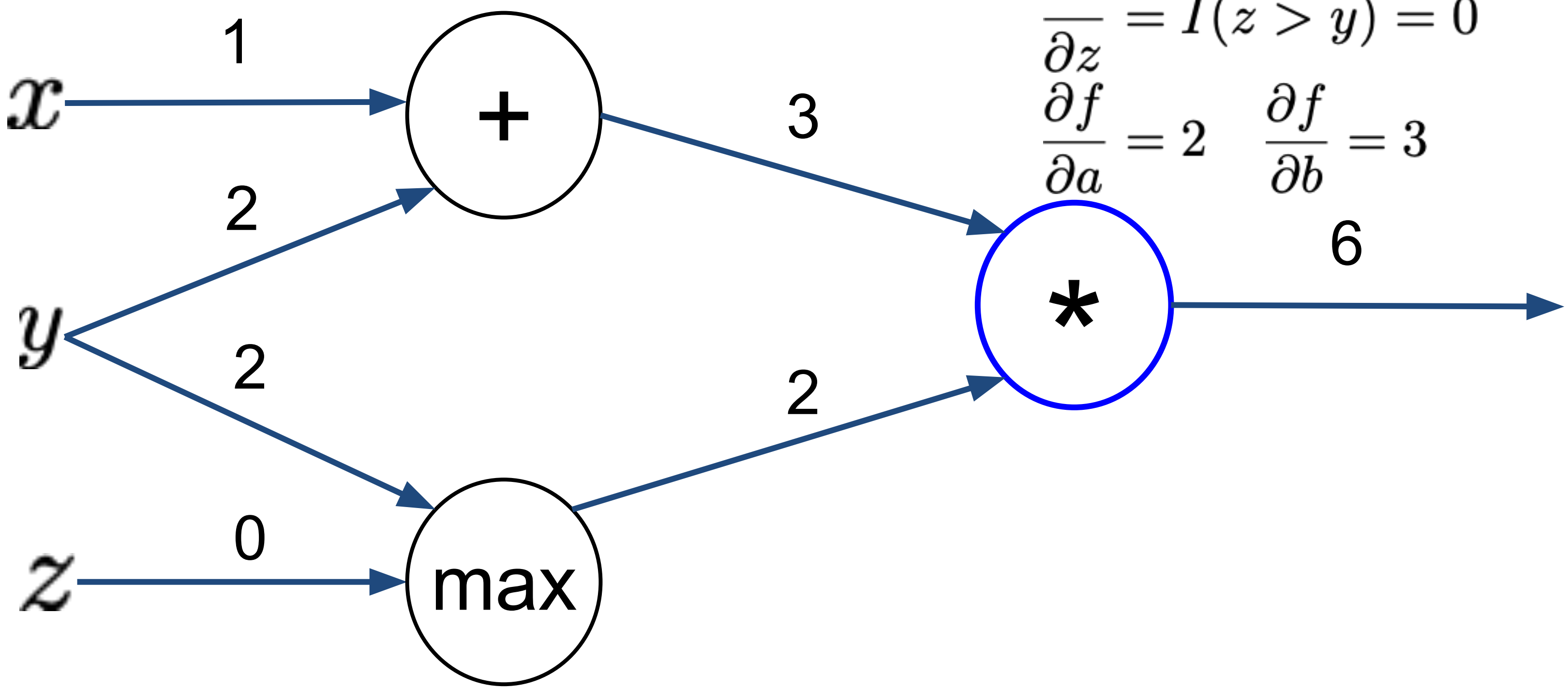
An example

$$f(x, y, z) = (x + y) \max(y, z)$$
$$x = 1, y = 2, z = 0$$

- Forward prop steps:
- a = x+y
- b= max(y,z)
- f = ab

- Local gradients: $\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$

$$\frac{\partial b}{\partial y} = I(y > z) = 1$$
$$\frac{\partial b}{\partial z} = I(z > y) = 0$$
$$\frac{\partial f}{\partial a} = 2 \quad \frac{\partial f}{\partial b} = 3$$

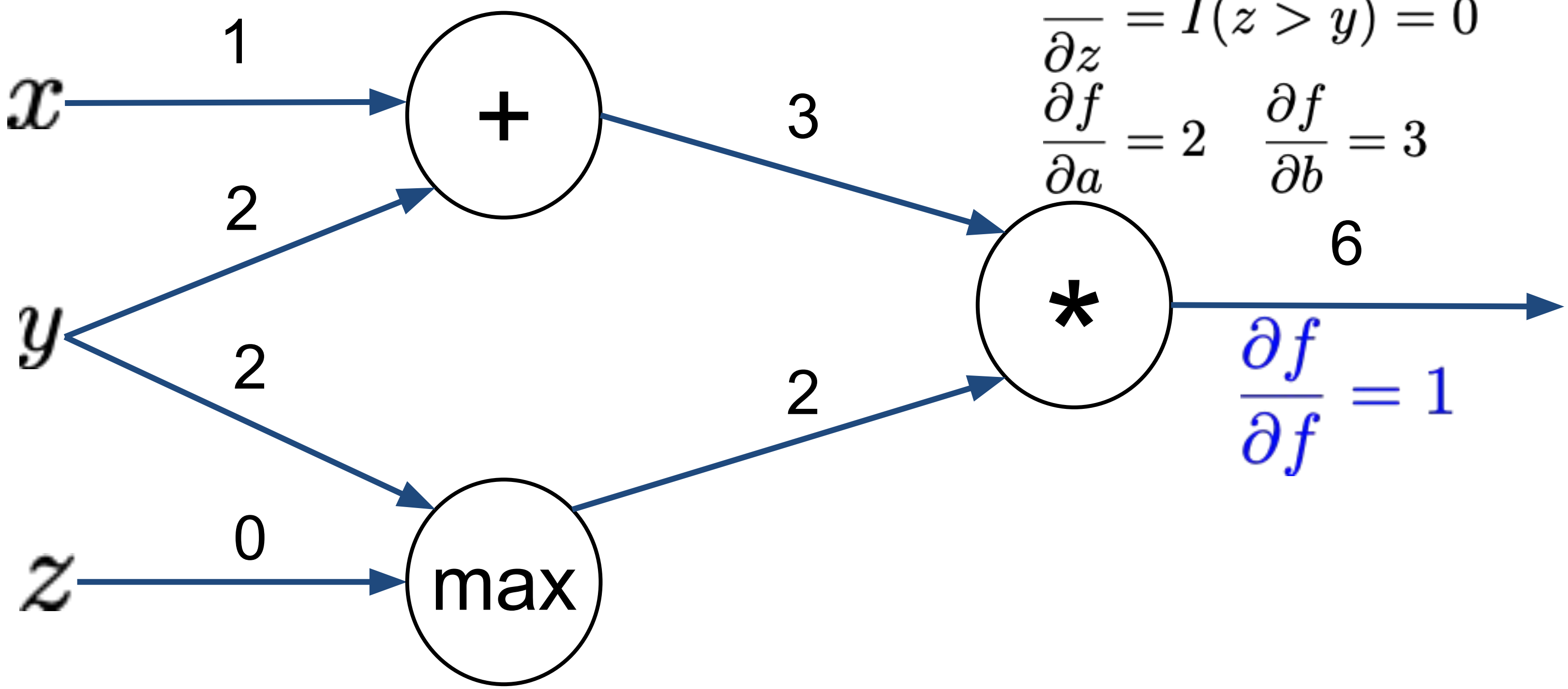


An example

$$f(x,y,z) = (x+y)\max(y,z)$$
$$x=1, y=2, z=0$$

- Backpropagation
- Local gradients: $\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$

$$\frac{\partial b}{\partial y} = I(y>z) = 1$$
$$\frac{\partial b}{\partial z} = I(z>y) = 0$$
$$\frac{\partial f}{\partial a} = 2 \quad \frac{\partial f}{\partial b} = 3$$



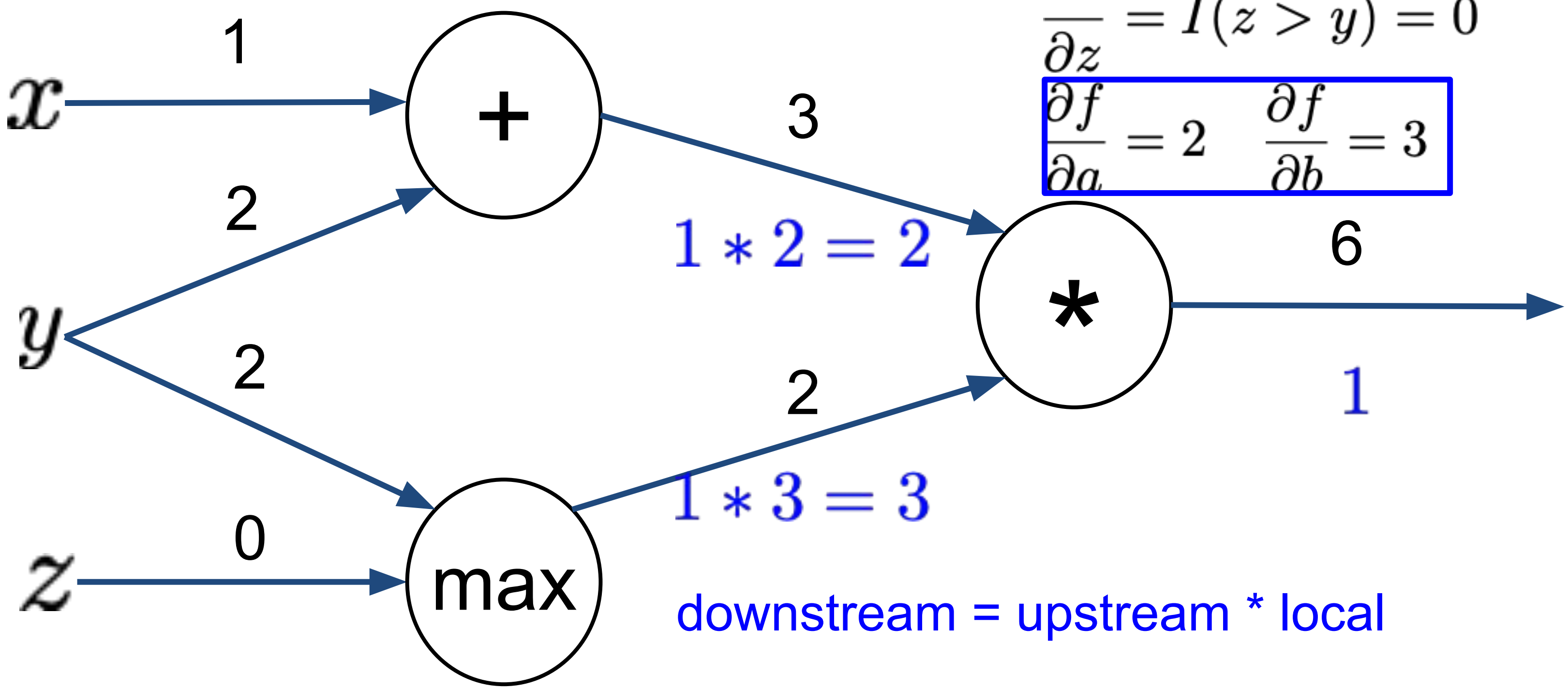
$$\frac{\partial f}{\partial f} = 1$$

An example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

- Backpropagation
- Local gradients: $\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$

$$\frac{\partial b}{\partial y} = I(y > z) = 1$$
$$\frac{\partial b}{\partial z} = I(z > y) = 0$$
$$\boxed{\frac{\partial f}{\partial a} = 2 \quad \frac{\partial f}{\partial b} = 3}$$



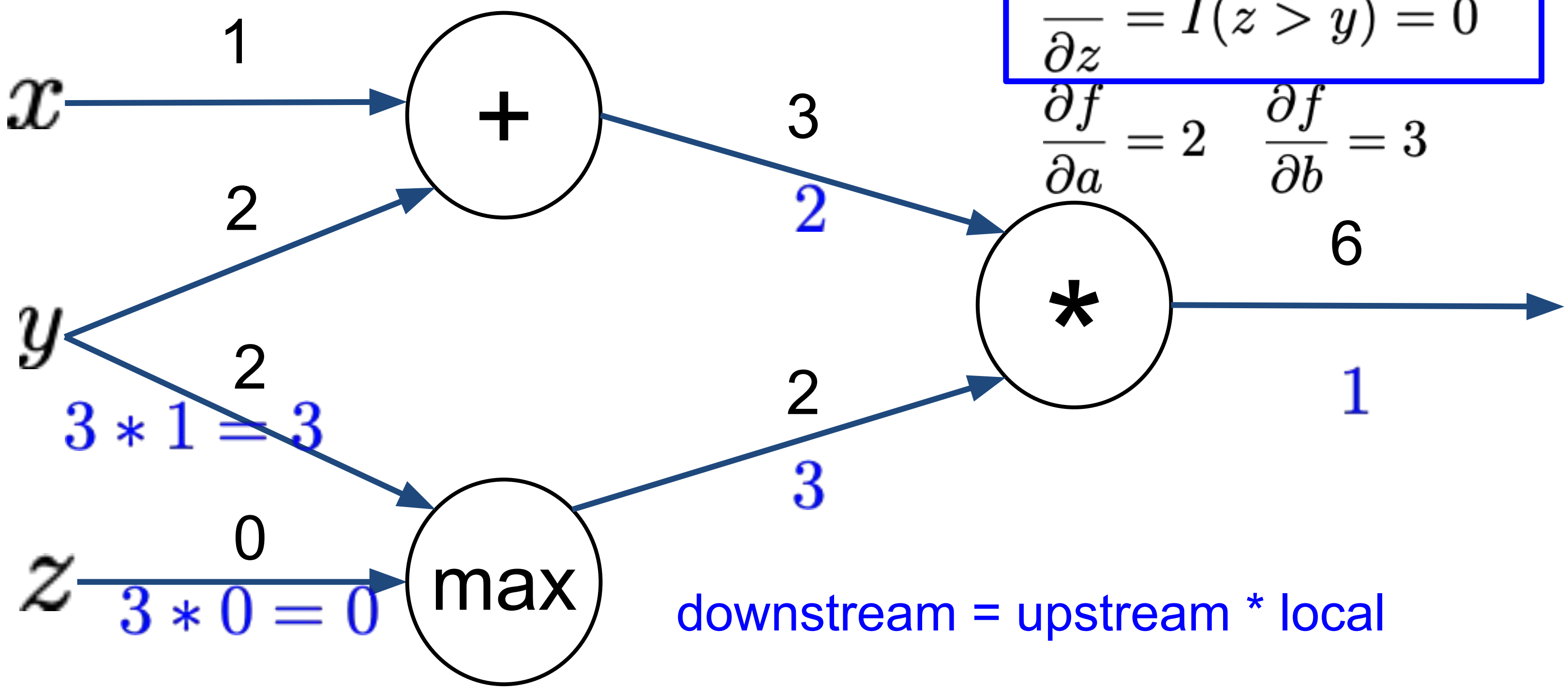
An example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

- Backpropagation
- Local gradients: $\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$

$$\boxed{\begin{aligned} \frac{\partial b}{\partial y} &= I(y > z) = 1 \\ \frac{\partial b}{\partial z} &= I(z > y) = 0 \end{aligned}}$$

$$\frac{\partial f}{\partial a} = 2 \quad \frac{\partial f}{\partial b} = 3$$



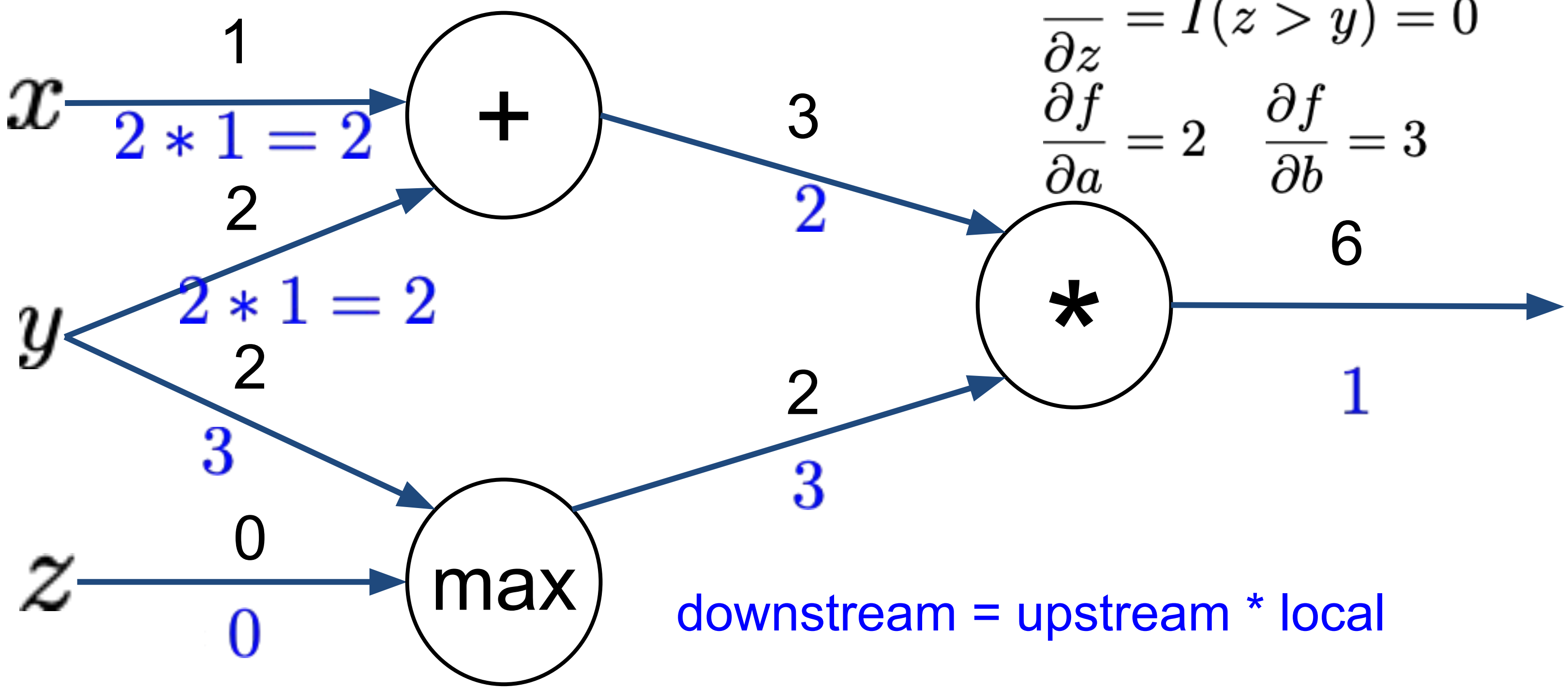
downstream = upstream * local

An example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

- Backpropagation
- Local gradients: $\boxed{\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1}$

$$\frac{\partial b}{\partial y} = I(y > z) = 1$$
$$\frac{\partial b}{\partial z} = I(z > y) = 0$$
$$\frac{\partial f}{\partial a} = 2 \quad \frac{\partial f}{\partial b} = 3$$



downstream = upstream * local

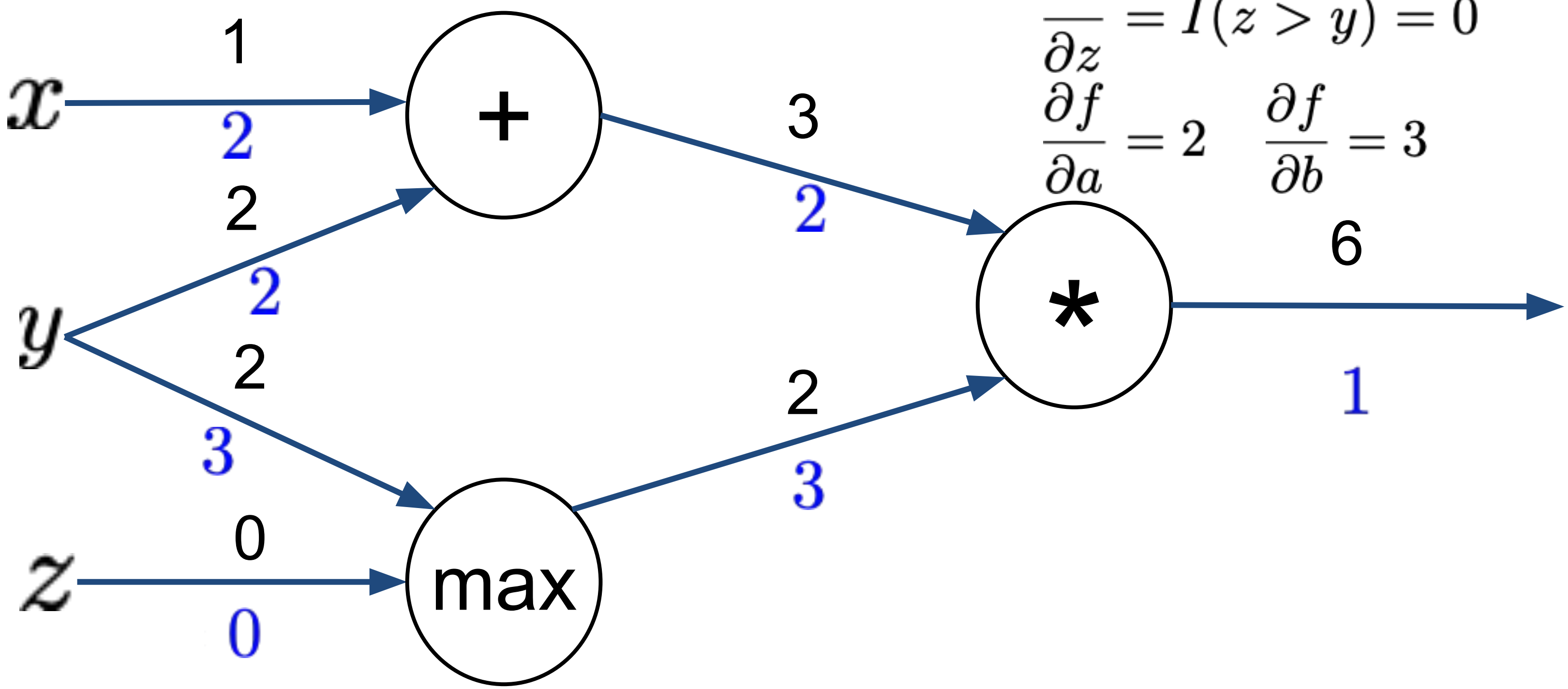
An example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

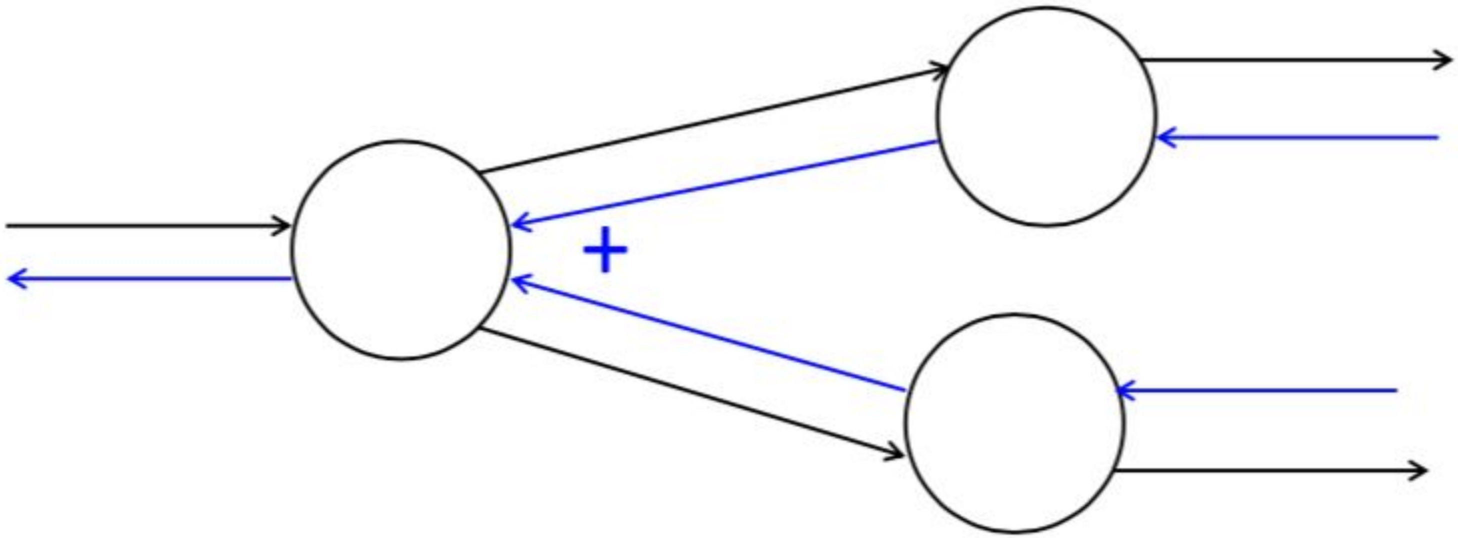
- Backpropagation
- Local gradients: $\frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$

$$\frac{\partial b}{\partial y} = I(y > z) = 1$$
$$\frac{\partial b}{\partial z} = I(z > y) = 0$$
$$\frac{\partial f}{\partial a} = 2 \quad \frac{\partial f}{\partial b} = 3$$

$$\frac{\partial f}{\partial x} = 2 \quad \frac{\partial f}{\partial y} = 2 + 3 = 5 \quad \frac{\partial f}{\partial z} = 0$$



Gradients sum at outward branches

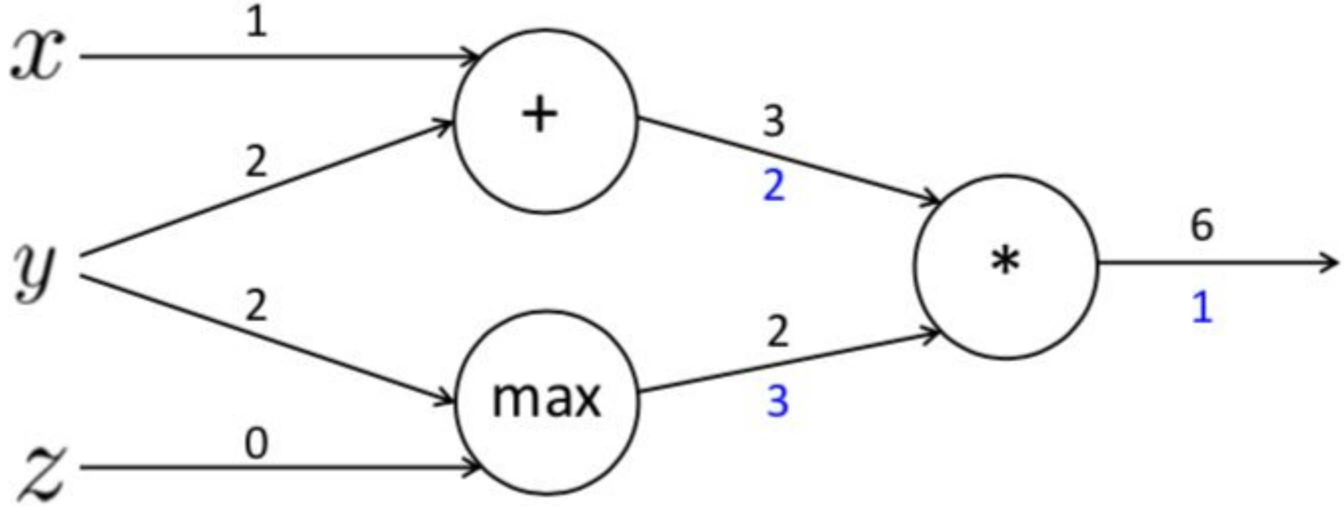


- a = x+y
- b= max(y,z)
- f = ab

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial y} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial y}$$

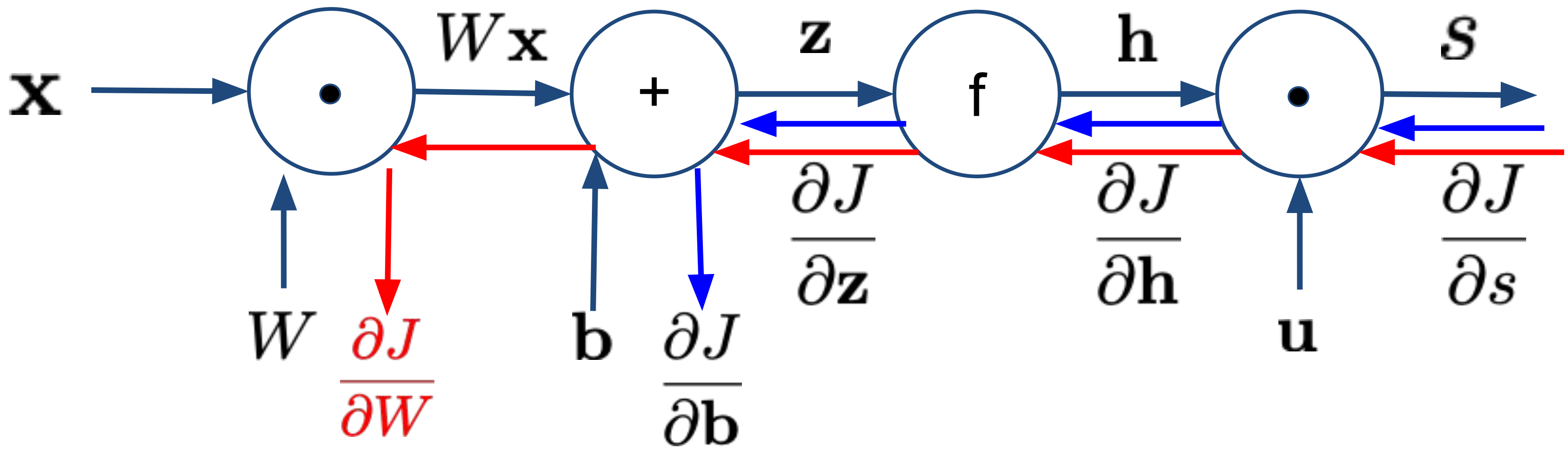
Node intuitions

- + distributes the upstream gradient
- max “routes” the upstream gradient
- * switches the upstream gradient



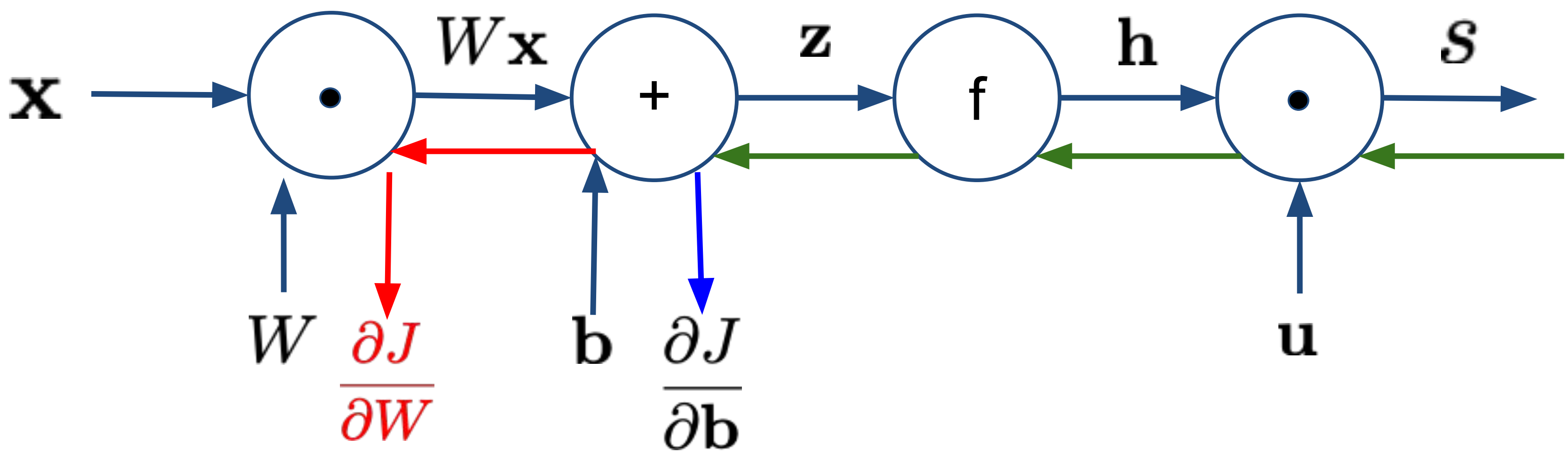
Efficiency: compute all gradients at once

- **Incorrect** way of doing backprop:
  - First compute gradient of b
  - Then independently compute gradient of W
  - Duplicated computation



Efficiency: compute all gradients at once

- **Correct** way of doing backprop:
  - Computer all the gradients at once
  - Analogous to using upstream gradients when we compute gradients by hand



Gradient checking: numeric gradient

- For small h(~1e-4)

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

- Easy to implement correctly
- but approximate and very **slow**
  - hard to recompute f for every parameter of our model
- Useful for checking your implementation
  - in the old days when we hand-wrote everything
  - not much less needed, when throwing together layers

Backpropagation

$$\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \mathbf{x} \rightarrow \underbrace{\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}}_{\mathbf{z}^{[1]}} \; \underbrace{\sigma(\mathbf{z}^{[1]})}_{\mathbf{h}^{[1]}} \rightarrow \underbrace{\mathbf{W}^{[2]}\mathbf{h}^{[1]} + \mathbf{b}^{[2]}}_{\mathbf{z}^{[2]}} \; \underbrace{\sigma(\mathbf{z}^{[2]})}_{\mathbf{h}^{[2]}=\hat{\mathbf{y}}} \rightarrow L(\hat{y}, y)$$

1. Break up equations

$$\hat{y} = \sigma(z^{[2]})$$

$$z^{[2]} = W^{[2]}h^{[1]} + b^{[2]}$$

$$h^{[1]} = \sigma(z^{[1]})$$

$$z^{[1]} = W^{[1]}\mathbf{x} + b^{[1]}$$

Backpropagation

$$\mathbf{W}^{[1]}, \mathbf{b}^{[1]},\ \mathbf{x} \rightarrow \underbrace{\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}}_{\mathbf{z}^{[1]}}\ \underbrace{\sigma(\mathbf{z}^{[1]})}_{\mathbf{h}^{[1]}} \longrightarrow \underbrace{\mathbf{W}^{[2]}\mathbf{h}^{[1]} + \mathbf{b}^{[2]}}_{\mathbf{z}^{[2]}}\ \underbrace{\sigma(\mathbf{z}^{[2]})}_{\mathbf{h}^{[2]} = \hat{\mathbf{y}}} \longrightarrow L(\hat{y}, y)$$

2. Apply Chain Rule

Binary loss for one example: $J(\theta) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$

$$dW^{[2]} = \frac{\partial J}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial W^{[2]}}$$

$$db^{[2]} = \frac{\partial J}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}}$$

Backpropagation

$$\mathbf{W}^{[1]}, \mathbf{b}^{[1]} \rightarrow \underbrace{\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}}_{\mathbf{z}^{[1]}} \; \underbrace{\sigma(\mathbf{z}^{[1]})}_{\mathbf{h}^{[1]}} \rightarrow \underbrace{\mathbf{W}^{[2]}\mathbf{h}^{[1]} + \mathbf{b}^{[2]}}_{\mathbf{z}^{[2]}} \; \underbrace{\sigma(\mathbf{z}^{[2]})}_{\mathbf{h}^{[2]}=\hat{\mathbf{y}}} \rightarrow L(\hat{y}, y)$$

$$\mathbf{x} \rightarrow$$

2. Apply Chain Rule

Binary loss for one example: $J(\theta) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$

$$dW^{[1]} = \frac{\partial J}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial h^{[1]}} \frac{\partial h^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}}$$

$$db^{[1]} = \frac{\partial J}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial h^{[1]}} \frac{\partial h^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial b^{[1]}}$$

The same as last step, avoid duplicated computation

Backpropagation

$$\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \mathbf{x} \rightarrow \underbrace{\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}}_{\mathbf{z}^{[1]}} \; \underbrace{\sigma(\mathbf{z}^{[1]})}_{\mathbf{h}^{[1]}} \rightarrow \underbrace{\mathbf{W}^{[2]}\mathbf{h}^{[1]} + \mathbf{b}^{[2]}}_{\mathbf{z}^{[2]}} \; \underbrace{\sigma(\mathbf{z}^{[2]})}_{\mathbf{h}^{[2]}=\hat{\mathbf{y}}} \rightarrow L(\hat{y}, y)$$

3. Write out the gradients

Binary loss for one example: $J(\theta) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$

$$dW^{[2]} = \frac{\partial J}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial W^{[2]}} = \frac{h^{[2]} - y}{h^{[2]}(1 - h^{[2]})} h^{[2]}(1 - h^{[2]})(\mathbf{h}^{[1]})^\top = (h^{[2]} - y)(\mathbf{h}^{[1]})^\top$$

$$db^{[2]} = \frac{\partial J}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial b^{[2]}} = (h^{[2]} - y)$$

Backpropagation

$$\mathbf{W}^{[1]}, \mathbf{b}^{[1]} \rightarrow \underbrace{\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}}_{\mathbf{z}^{[1]}} \underbrace{\sigma(\mathbf{z}^{[1]})}_{\mathbf{h}^{[1]}} \rightarrow \underbrace{\mathbf{W}^{[2]}\mathbf{h}^{[1]} + \mathbf{b}^{[2]}}_{\mathbf{z}^{[2]}} \underbrace{\sigma(\mathbf{z}^{[2]})}_{\mathbf{h}^{[2]} = \hat{\mathbf{y}}} \rightarrow L(\hat{y}, y)$$

$$\mathbf{x} \rightarrow$$

3. Write out the gradients

Binary loss for one example: $J(\theta) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$

$$dW^{[1]} = \frac{\partial J}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial h^{[1]}} \frac{\partial h^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial W^{[1]}} = (h^{[2]} - y)(W^{[2]})^\top \circ h^{[1]} \circ (1 - h^{[1]})\mathbf{x}^\top$$

$$db^{[1]} = \frac{\partial J}{\partial h^{[2]}} \frac{\partial h^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial h^{[1]}} \frac{\partial h^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial b^{[1]}} = (h^{[2]} - y)(W^{[2]})^\top \circ h^{[1]} \circ (1 - h^{[1]})$$

Backpropagation

$$\begin{matrix} W^{[1]}, \mathbf{b}^{[1]} \rightarrow \\ \mathbf{x} \rightarrow \end{matrix} \underbrace{W^{[1]}\mathbf{x} + \mathbf{b}^{[1]}}_{\mathbf{z}^{[1]}} \; \underbrace{\sigma(\mathbf{z}^{[1]})}_{\mathbf{h}^{[1]}} \longrightarrow \underbrace{W^{[2]}\mathbf{h}^{[1]} + \mathbf{b}^{[2]}}_{\mathbf{z}^{[2]}} \; \underbrace{\sigma(\mathbf{z}^{[2]})}_{\mathbf{h}^{[2]}=\hat{\mathbf{y}}} \longrightarrow L(\hat{y}, y)$$

Binary loss for one example: $J(\theta) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$

$$dz^{[2]} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{[2]}} = \hat{y} - y$$

$$dW^{[2]} = dz^{[2]} \mathbf{h}^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$d\mathbf{z}^{[1]} = \frac{\partial L}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial \mathbf{z}^{[1]}} = W^{[2]T} dz^{[2]} \circ \mathbf{h}^{[1]} \circ (1 - \mathbf{h}^{[1]})$$

$$dW^{[1]} = d\mathbf{z}^{[1]} \mathbf{x}^T$$

$$d\mathbf{b}^{[1]} = d\mathbf{z}^{[1]}$$

Optimization - summary

Feedforward

$$\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + b^{[1]}$$

$$\mathbf{h}^{[1]} = \sigma(\mathbf{z}^{[1]})$$

$$z^{[2]} = W^{[2]}\mathbf{h}^{[1]} + b^{[2]}$$

$$\hat{y} = h^{[2]} = \sigma(z^{[2]})$$

Backpropagation

$$dz^{[2]} = h^{[2]} - y$$

$$dW^{[2]} = dz^{[2]}\mathbf{h}^{[1]\,T}$$

$$db^{[2]} = dz^{[2]}$$

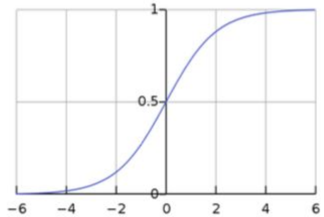
$$d\mathbf{z}^{[1]} = W^{[2]\,T} dz^{[2]} \circ \mathbf{h}^{[1]} \circ (1 - \mathbf{h}^{[1]})$$

$$dW^{[1]} = d\mathbf{z}^{[1]}\mathbf{x}^{T}$$

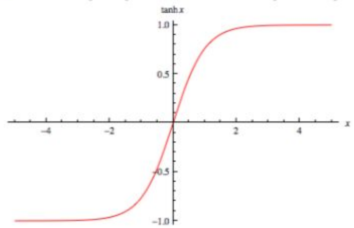
$$d\mathbf{b}^{[1]} = d\mathbf{z}^{[1]}$$

Activation Functions

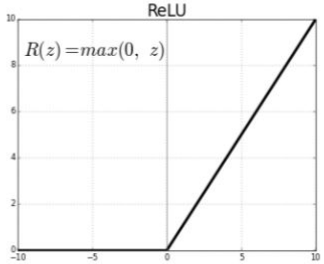
Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$



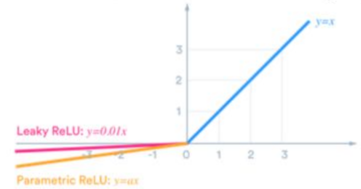
tanh: $f(x) = 2\sigma(2x) - 1$



ReLU: $f(x) = \max(0, x)$



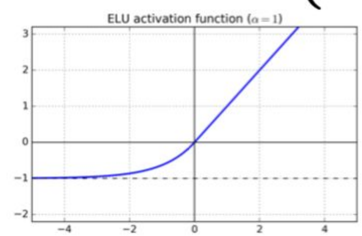
Leaky ReLU: $f(x) = \max(\alpha x, x)$



Maxout

$$\max(\mathbf{w}_1^T\mathbf{x} + b_1, \mathbf{w}_2^T\mathbf{x} + b_2)$$

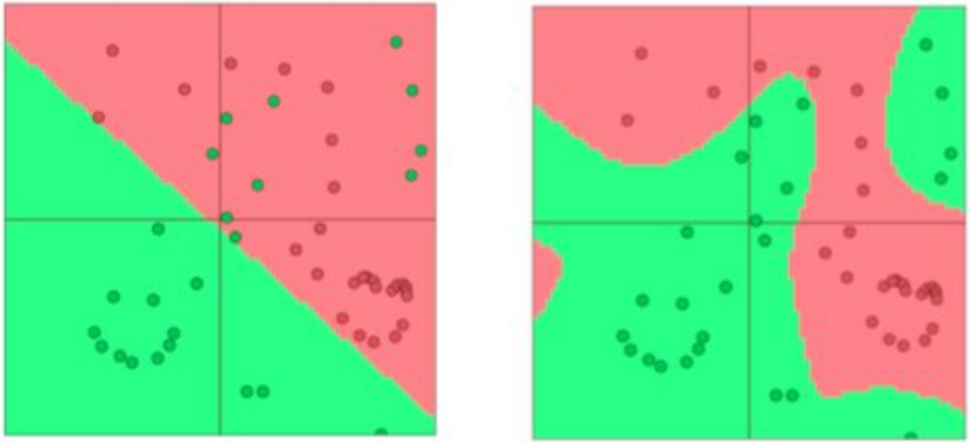
ELU: $f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

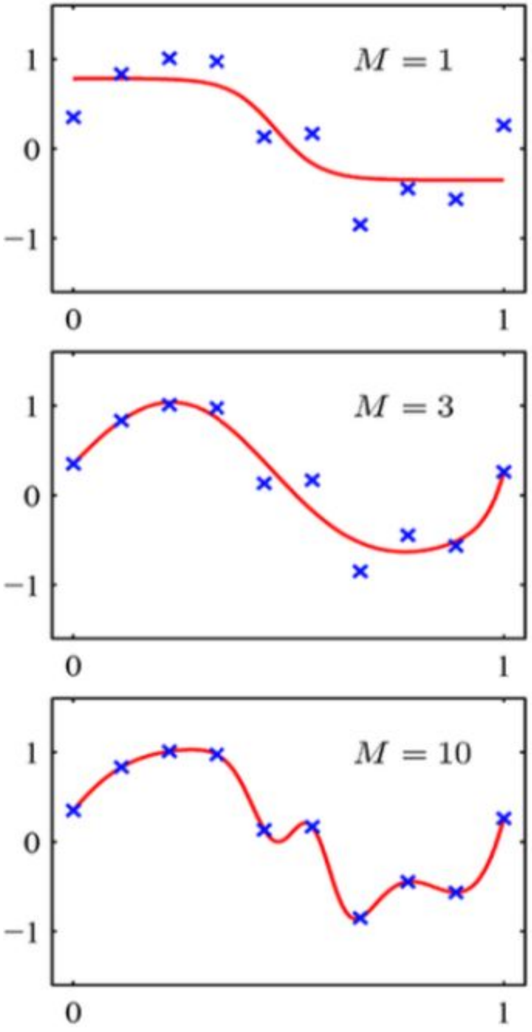


For building a feed-forward deep network, the first thing you should try is ReLU — it trains quickly and performs well due to good gradient backflow

Non-linearities

- Example: function approximation, e.g., regression or classification
  - Without non-linearities, deep neural networks can't do anything more than a linear transform. Extra layers could just be compiled into a single linear transform
  - With more layers, they can approximate more complex functions.





Summary

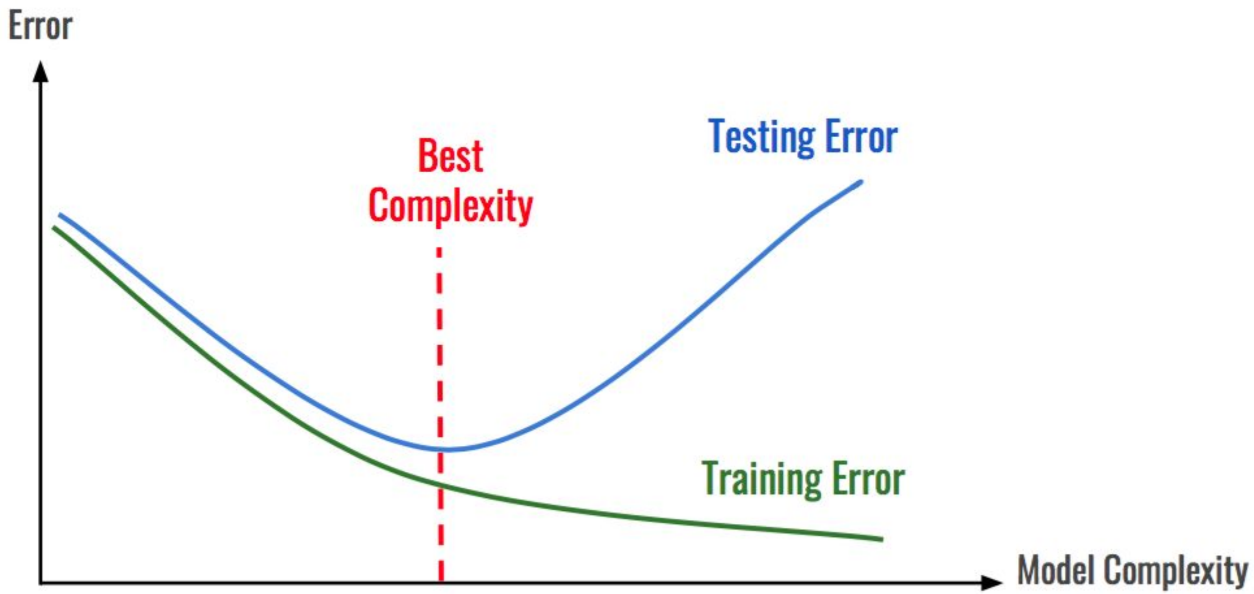
- We’ve mastered the core technology of neural nets
- Backpropagation: recursively apply the chain rule along computation graph
    - [downstream gradients] = [upstream gradients] x [local gradients]
- Forward pass: compute results of operations and save intermediate values
- backward pass: apply chain rule to compute gradients

Why learn all these details about gradients

- Modern deep learning frameworks compute gradients for you
- But why take a class on compilers or systems when they are implemented for you?
  - Understanding what is going on under the hood is useful!
- Backpropagation doesn’t always work perfectly.
  - Understanding why is crucial for debugging and improving models
  - See Karpathy article (in syllabus):
    - https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b
  - Example in future lecture: exploding and vanishing gradients

Regularization

- Really a full loss function in practice includes regularization over all parameters, e.g. L2 regularization
- Regularization (largely) prevents overfitting when we have a lot of features (or later a very powerful/deep model)



Vectorization

Looping over word vectors v.s. concatenating them all into one matrix and then multiplying the softmax weights with that matrix

```
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

1000 loops, best of 3: 639 μs per loop
10000 loops, best of 3: 53.8 μs per loop

Vectorization

```
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

- The (10x) faster method is using a CxN matrix
- Always try to use vectors and matrices rather than for loops!
- You should speed-test your code a lot

Parameter Initialization

- You normally must initialize weights to small random values.
  - To avoid symmetries that prevent learning/specialization
- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target)
- Initialize **all other weights** ~ Uniform(–r, r), with r chosen so numbers get neither too big or too small
- Xavier initialization has variance inversely proportional to fan-in n_{in} (previous layer size) and fan-out n_{out} (next layer size)

$$\mathrm{Var}(W_i) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

Optimizers

- Usually, plain SGD will work just fine
  - However, getting good results will often require hand-tuning the learning rate (next slide)
- For more complex nets and situations, or just to avoid worry, you often do better with one of a family of more sophisticated “adaptive” optimizers that scale the parameter adjustment by an accumulated gradient.
  - These models give per-parameter learning rates
    - Adagrad
    - RMSprop
    - Adam (A fairly good,safe place to begin in many cases)
    - SparseAdam

Learning Rates

- You can just use a constant learning rate. Start around lr = 0.001?
  - It must be order of magnitude right - try powers of 10
    - Too big: model may diverge or not converge
    - Too small: your model may not have trained by the deadline
- Better results can generally be obtained by allowing learning rates to decrease as you train
  - By hand: halve the learning rate every k epochs
    - An epoch = a pass through the data (shuffled or sampled)
    - By a formula: $lr=lr_0 e^{-kt}$ for epoch t
    - There are fancier methods like cyclic learning rates (q.v.)
- Fancier optimizers still use a learning rate but it may be an initial rate that the optimizer shrinks – so may be able to start high

STEVENS
INSTITUTE *of* TECHNOLOGY
THE INNOVATION UNIVERSITY®

1870

stevens.edu

Thank You

Named Entity Recognition (NER)

- The task: find and classify names in text, for example:

The European Commission [ORG] said on Thursday it disagreed with disagreed with German [MISC] advice.
Only France [LOC] and Britain [LOC] backed Fischler [PER]'s proposal .
"What we have to be extremely careful of is how other countries are going to take Germany 's lead", Welsh National Farmers' Union [ORG] (NFU [ORG]) chairman John Lloyd Jones [PER] said on BBC [ORG] radio.

- Possible purposes:
  - Tracking mentions of particular entities in documents
  - For question answering, answers are usually named entities
  - A lot of wanted information is really associations between named entities
  - The same techniques can be extended to other slot-filling classifications

Named Entity Recognition on word sequences

We predict entities by classifying words in context and then extracting entities as word subsequences

Foreign	ORG	B-ORG
Ministry	ORG	I-ORG
spokesman	O	O
Shen	PER	B-PER
Guofang	PER	I-PER
told	O	O
Reuters	ORG	B-ORG
that	O	O
...	...	B(egin) I(nside) O(outside) encoding

Why NER is hard?

- Boundaries of entity
  - “First National bank donates 2 vans to future school of Fort Smith”
  - Is the first entity “First National Bank” or “National Bank”?

- Hard to know if something is an entity
  - Is there a school called “Future School” or is it a future school?

- Hard to know class of unknown/novel entity:

- Entity class is ambiguous and depends on context

Binary word window classification

In general, classifying single words is rarely done

Inter