STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS 584 Natural Language Processing

Machine Learning Basics

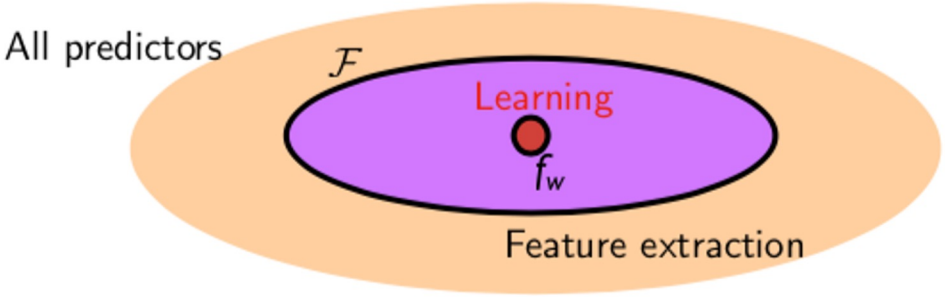
Ping Wang
Department of Computer Science
Stevens Institute of Technology

Outline

- Supervised Learning
- Classification
- Logistic Regression
- Gradient Descent Optimization
- Neural Networks

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, $\{x_i, y_i\}_{i=1}^N$ we find an estimate $\hat{f} \in F$

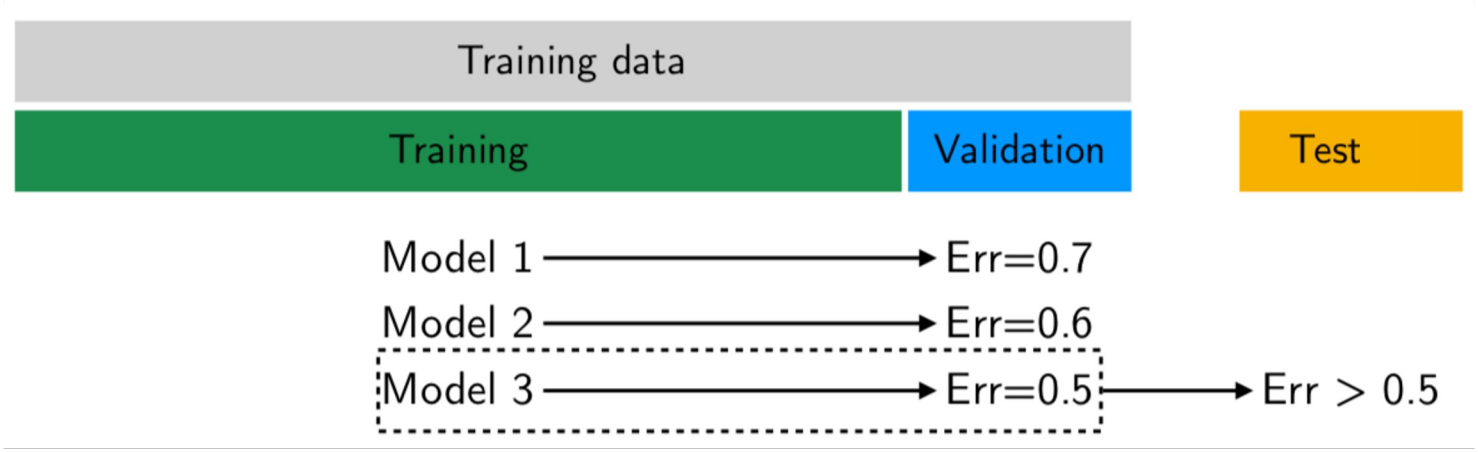
- **Evaluation**: We measure how well the estimate generalize to yet unseen examples.

Training vs Validation vs Testing

- Training dataset: $\{x_i, y_i\}_{i=1}^{N}$ 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.
- Test dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset. Cannot be used for training.

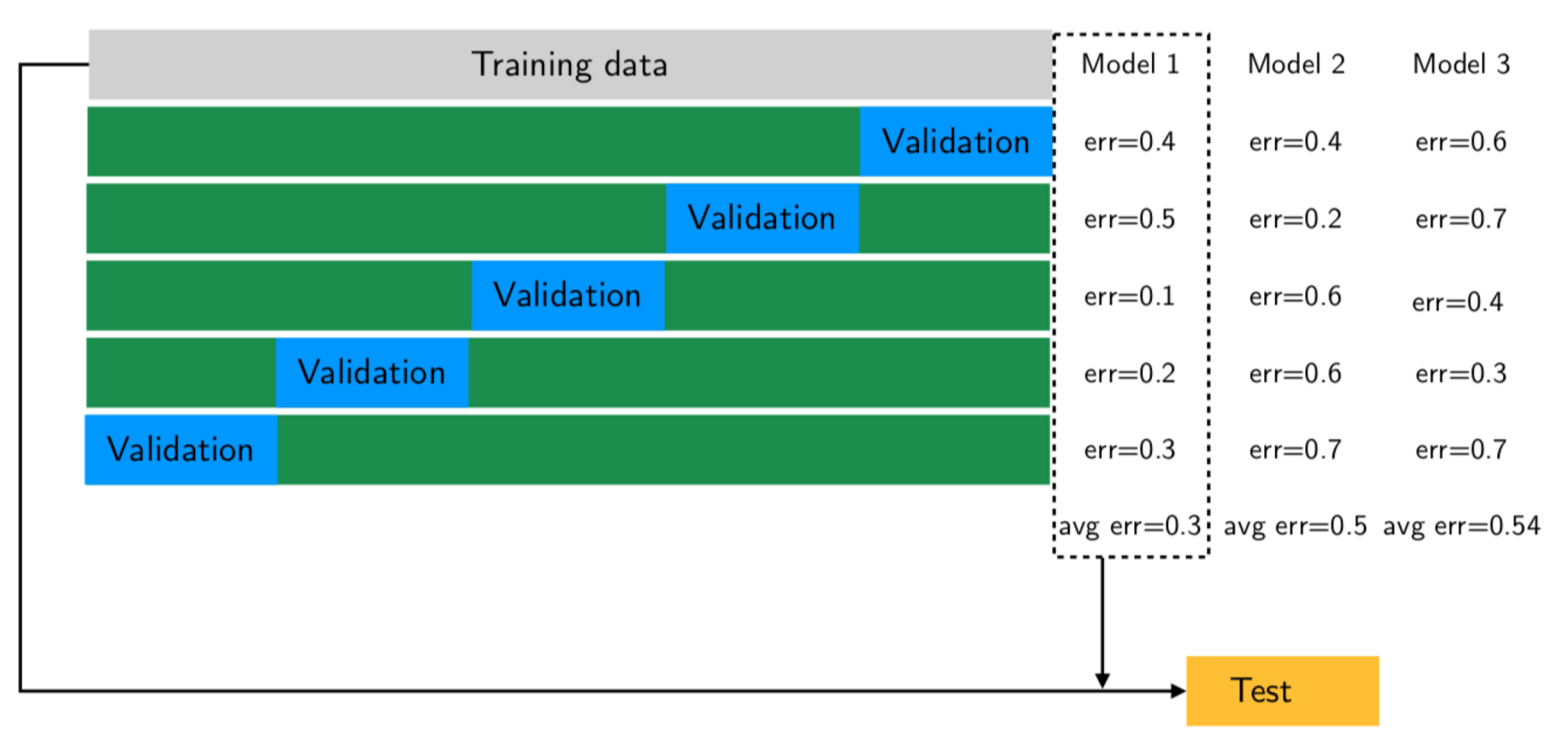
Cross-Validation

- 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.



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

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

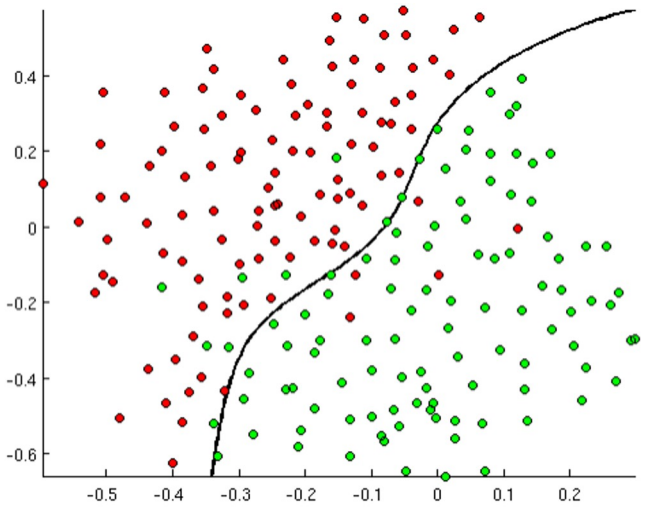
- x_i can be:
  - Word counts
  - TF-IDF
  - Topic distributions
  - …
  - More details will be covered in next lecture.

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: $\{x_i, y_i\}_{i=1}^{N}$
- 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

- Binary: For each x_i, predict if x_i belongs to class +1 using:

$$p(y_i = 1|\mathbf{x}_i) = \sigma(\mathbf{w}^\top \mathbf{x}_i) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}_i}}$$

When x_i belongs to the positive class, y_i=1; otherwise y_i=0.

1. p is the probability of example x_i belonging to the positive class; 1-p is the probability of example x_i belonging to the negative class.
2. $\mathbf{w} \in \mathbb{R}^d$ is the model parameter vector
3. Sigmoid function: $\sigma(a) = \frac{1}{1 + e^{-a}}$

Logistic Regression

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

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

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

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

  2. Apply softmax function to get normalized probability:

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

Logistic Regression - Training

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

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

- Or we can **minimize** the negative log probability of that class (cross-entropy loss):

Multi-class: $$J(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log p(C_k|\mathbf{x}_i)$$

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

Classification on a full dataset

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

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

- Instead of

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

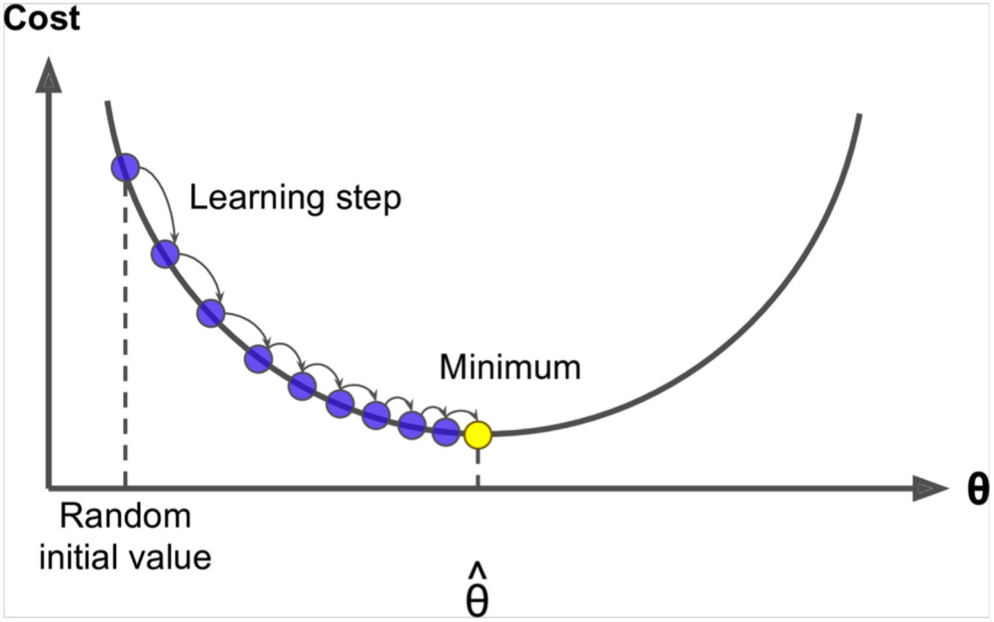
- We can write f in matrix operation:

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

- Always try to use vectors and matrices rather than for loops during implementation.

Optimization: Gradient descent

- We have a cost function J(θ) we want to minimize
- Gradient descent is an algorithm to minimize J(θ)
- Idea: for current value of θ, calculate gradient of J(θ), then take small step in direction of negative gradient. Repeat.



Note: our objectives may not be convex like this.

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 (epochs):

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

Gradient Descent is slow

Algorithm: gradient descent

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

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

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

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

- An epoch: a pass through the data (shuffled or sampled).
- Each iteration requires going over all training examples (Batch gradient descent) – expensive and slow when have lots of data.
- Intractable for datasets that don't fit in memory.
- Guaranteed to converge to the global minimum for convex functions, but may end up at a local minimum for non-convex functions.

Stochastic Gradient Descent

Algorithm: stochastic gradient descent

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

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

- Shuffling: to ensure that each data point creates an “independent” change on the model, without being biased by the same points before them.
- Allow for online update with new examples.
- With a high variance that cause the objective function to fluctuate heavily
- May never reach local minima and oscillate around it due to the fluctuations in each step.

Mini-batch Gradient Descent

Algorithm: mini-batch gradient descent

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

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

- Instead of using the whole data for calculating gradient, we use only a mini-batch of it (batch size is a hyperparameter).
- 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.

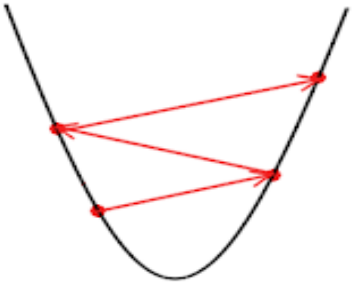
Learning rate

- Learning rate affects the update of gradients as well:

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

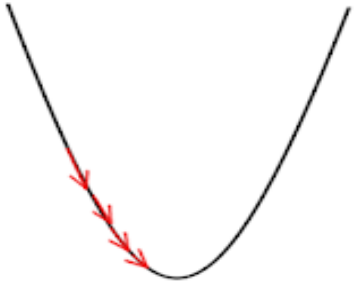
The cost is often highly sensitive to some directions in parameter space

High Learning Rate



Causes drastic updates and leading to divergent behavior

Just Right



Swiftly reaches the minimum point

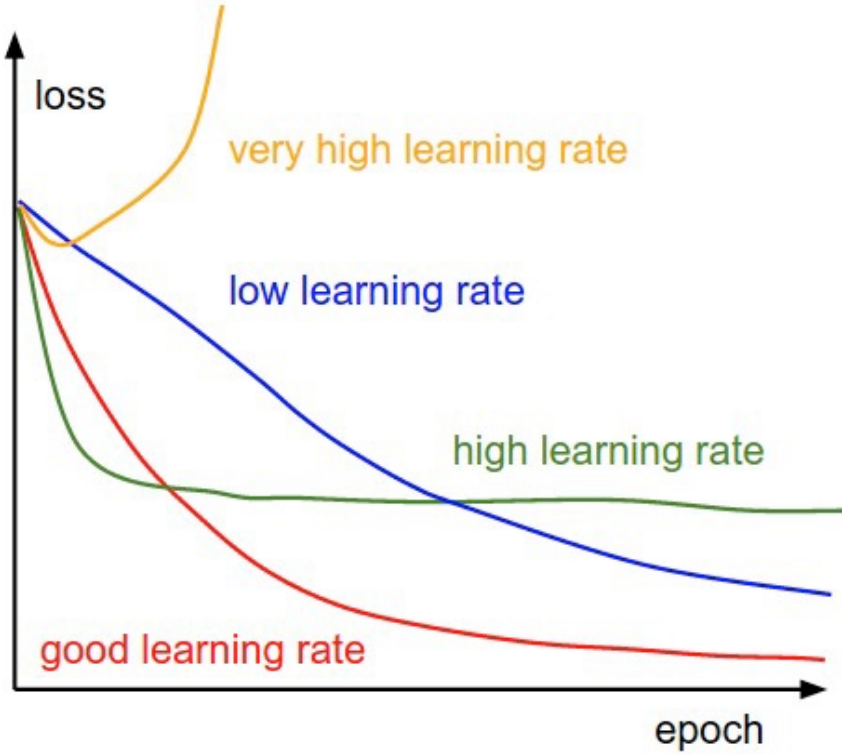
Too small

Requires many updates before reaching the minimum points

Learning Rate

Set the learning rate carefully:

- If there are more than 3 parameters, it is hard to visualize the loss w.r.t the parameters.
- But you can visualize the loss w.r.t the # of parameter updates (epoch).
  - If the loss increases, we need to decrease the value of the learning rate.
  - If the loss is decreasing at a very slow rate, we need to increase the value of the learning rate.



Optimizers

- Usually, plain SGD will work just fine
  - However, getting good results will often require hand-tuning the learning rate

- 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

Adaptive learning rates

- Popular and simple idea: reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate

- Learning rate cannot be one-size-fits-all
  - Giving different parameters different learning rates

Adaptive learning rates

- Divide the learning rate of each parameter by the root mean square of its previous derivatives.

$$\eta^t = \frac{\eta}{\sqrt{t+1}}, g^t = \frac{\partial L(w^t)}{\partial w}$$

- Vanilla gradient descent:

$$w^{t+1} \leftarrow w^t - \eta^t g^t$$

- Adagrad:

$$w^{t+1} \leftarrow w^t - \frac{\eta^t}{\sigma^t} g^t$$

σ^t : root mean square of the previous derivatives of parameter w

AdaGrad

$$
\begin{aligned}
w^{(1)} &\leftarrow w^{(0)} - \frac{\eta^{(0)}}{\sigma^{(0)}} g^{(0)} & \sigma^{(0)} &= \sqrt{(g^{(0)})^2 + \epsilon} \\
w^{(2)} &\leftarrow w^{(1)} - \frac{\eta^{(1)}}{\sigma^{(1)}} g^{(1)} & \sigma^{(1)} &= \sqrt{\tfrac{1}{2}[(g^{(0)})^2 + (g^{(1)})^2] + \epsilon} \\
w^{(3)} &\leftarrow w^{(2)} - \frac{\eta^{(2)}}{\sigma^{(2)}} g^{(2)} & \sigma^{(2)} &= \sqrt{\tfrac{1}{3}[(g^{(0)})^2 + (g^{(1)})^2 + (g^{(2)})^2] + \epsilon} \\
&\dots & &\dots \\
w^{(t+1)} &\leftarrow w^{(t)} - \frac{\eta^{(t)}}{\sigma^{(t)}} g^{(t)} & \sigma^{(t)} &= \sqrt{\tfrac{1}{t+1} \textstyle\sum_{i=1}^{t} (g^{(i)})^2 + \epsilon}
\end{aligned}
$$

ε is a smoothing term that avoids division by zero (usually on the order of 1e − 8)

AdaGrad

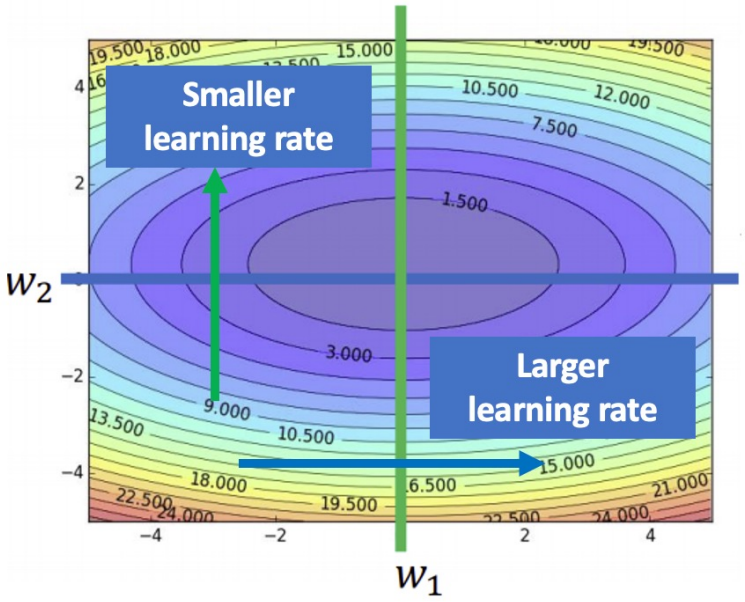
- Divide the learning rate of each parameter by the root mean square of its previous derivatives

$$\eta^{(t)} = \frac{\eta}{\sqrt{t+1}}$$

$$g^{(t)} = \frac{\partial L(x, y, w^{(t)})}{\partial w^{(t)}}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \frac{\eta^{(t)}}{\sigma^{(t)}} g^{(t)}$$

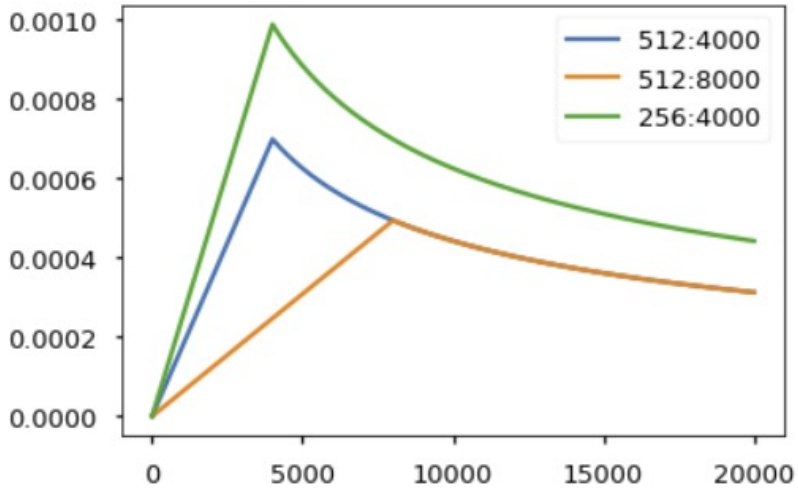
with $\sigma^{(t)} = \sqrt{\frac{1}{t+1}\sum_{i=1}^{t}(g^{(i)})^2 + \varepsilon}$



Small gradient, larger learning rate
Large gradient, smaller learning rate

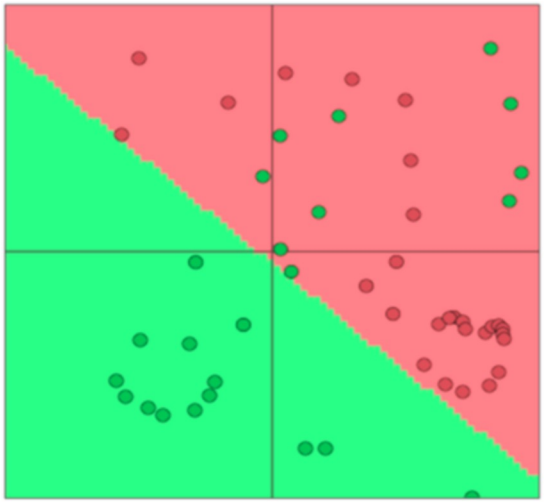
Warm-up Learning Rate

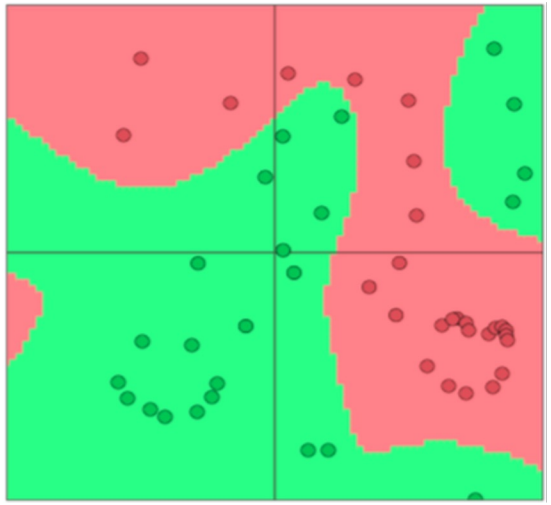
- **Motivation**: At the beginning of training, the weights of the model are randomly initialized. If you choose a larger learning rate, it may lead to instability (oscillation) of the model.
- **Solution**: At the beginning of training, it uses a small learning rate to train some epochs or steps, and then modifies it to the preset learning rate for training.
- The model can gradually become stable, which makes the convergence of the model become faster and the model effect is better.



Neural network classifiers

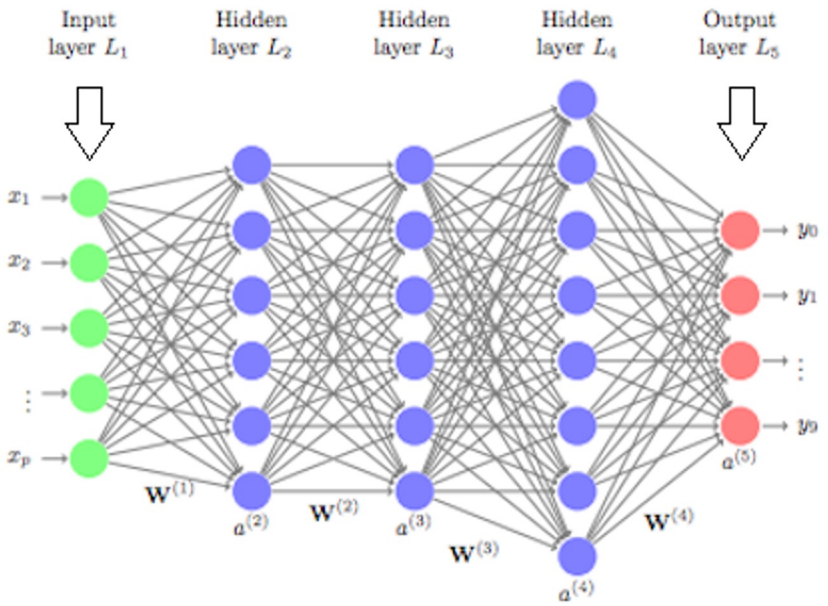
- Softmax (logistic regression~one layer neural nets) alone not very powerful
- Logistic regression gives only linear decision boundaries
  - Limited
  - Unhelpful when a problem is complex
- Neural networks can learn much more complex functions and nonlinear decision boundaries!





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 a variety of domains: 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;

Chain rule

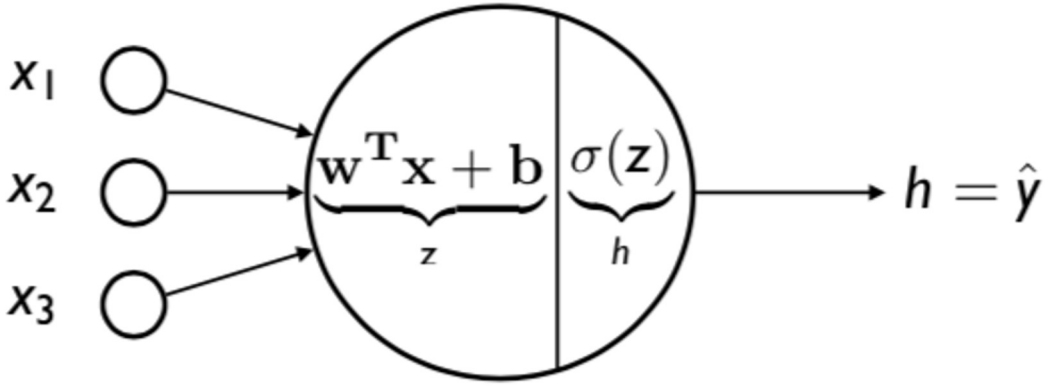
- In general, y = f(u), u = g(x) what is the derivative of y w.r.t x?

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

- Example: $y = e^{x^2}$

No hidden units: logistic regression

- Sigmoid activation function



- Output score: sigmoid/softmax function

Logistic regression

- Binary cross-entropy loss:

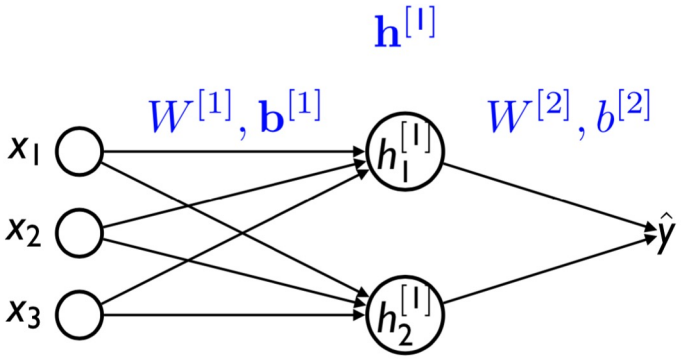
$$J(\theta) = \sum_{i=1}^{N} -y_i \log p_i - (1 - y_i) \log(1 - p_i)$$

- Gradient descent algorithm:

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

Neural networks: one hidden layer

- One hidden layer:



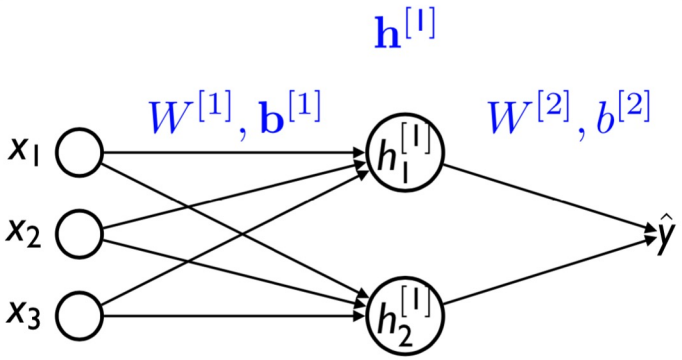
- Hidden layer representation

$$z_1^{[1]} = \mathbf{w}_1^{[1]\,T}\mathbf{x} + b_1^{[1]}, h_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = \mathbf{w}_2^{[1]\,T}\mathbf{x} + b_2^{[1]}, h_2^{[1]} = \sigma(z_2^{[1]})$$

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

Neural networks: one hidden layer

- One hidden layer:



- output layer

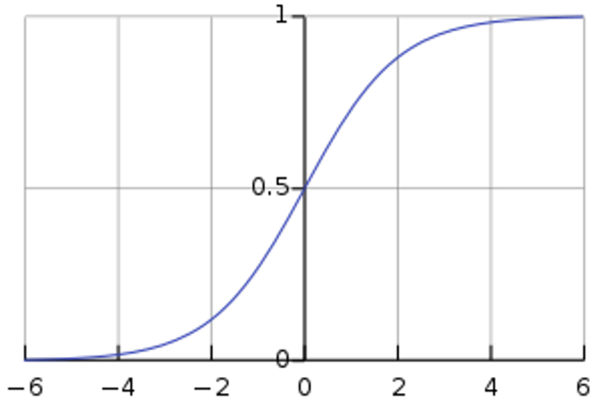
$$z^{[2]} = \underbrace{W^{[2]}}_{1\times 2}\underbrace{\mathbf{h}^{[1]}}_{2\times 1} + b^{[2]}$$

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

Binary logistic regression neurons

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

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



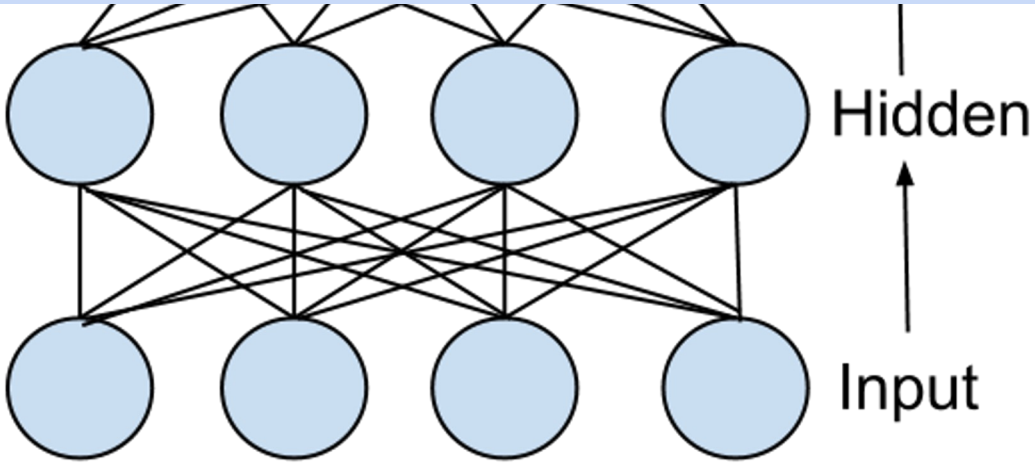
We can have an “always on” features, which gives a class prior, or separate it out, as a bias term.

w_1 and b_1 are the parameters of this neuron

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

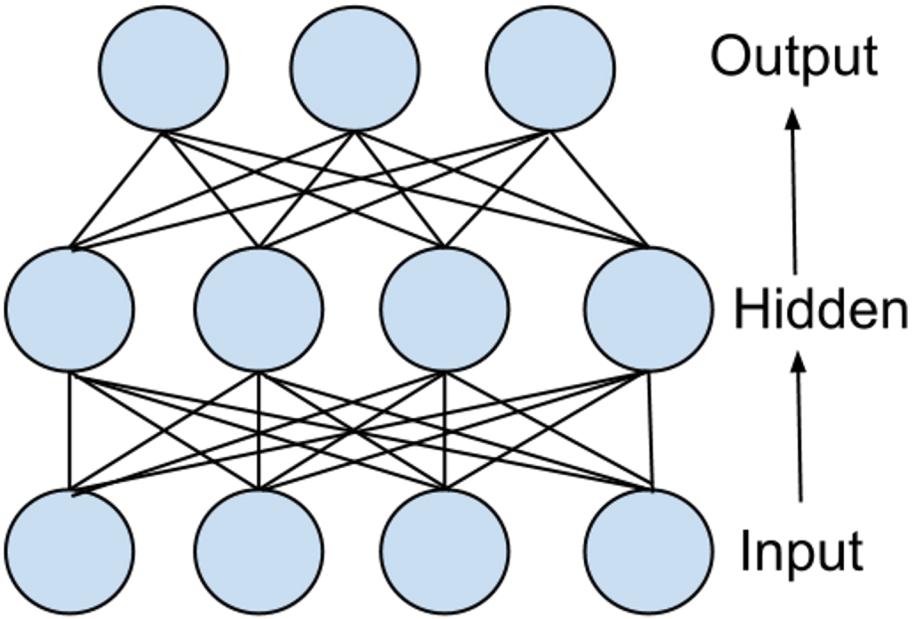
If we feed a vector of inputs through a bunch of logistic functions, then we get a vector of outputs

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

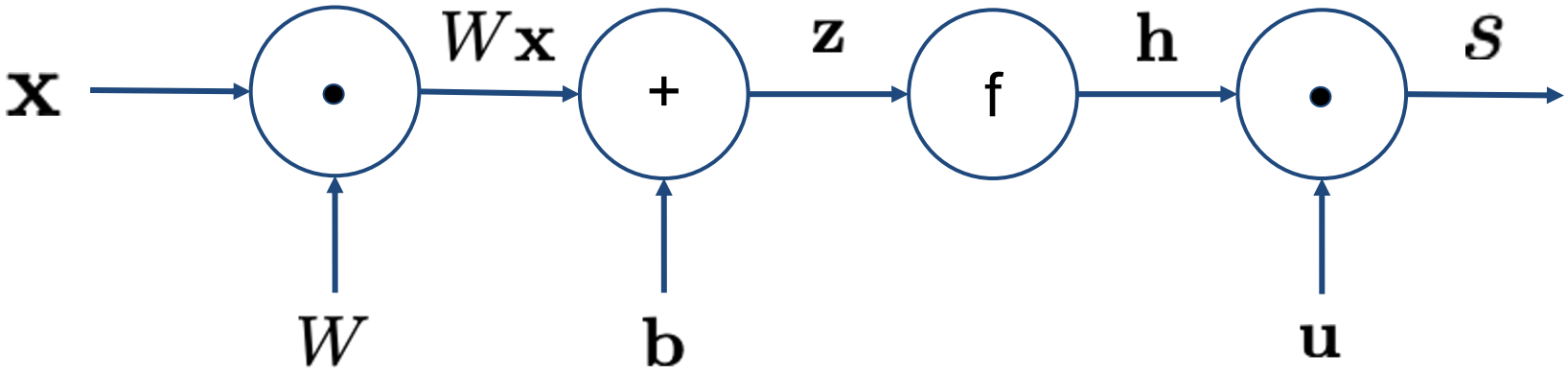
- Mathematical: just grind through the **chain rule**
- Learn the weights so that the loss is minimized
- It provides an efficient procedure to compute derivatives
  1. Break up equations into simple pieces
  2. Apply the chain rule
  3. Write out the gradients

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

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

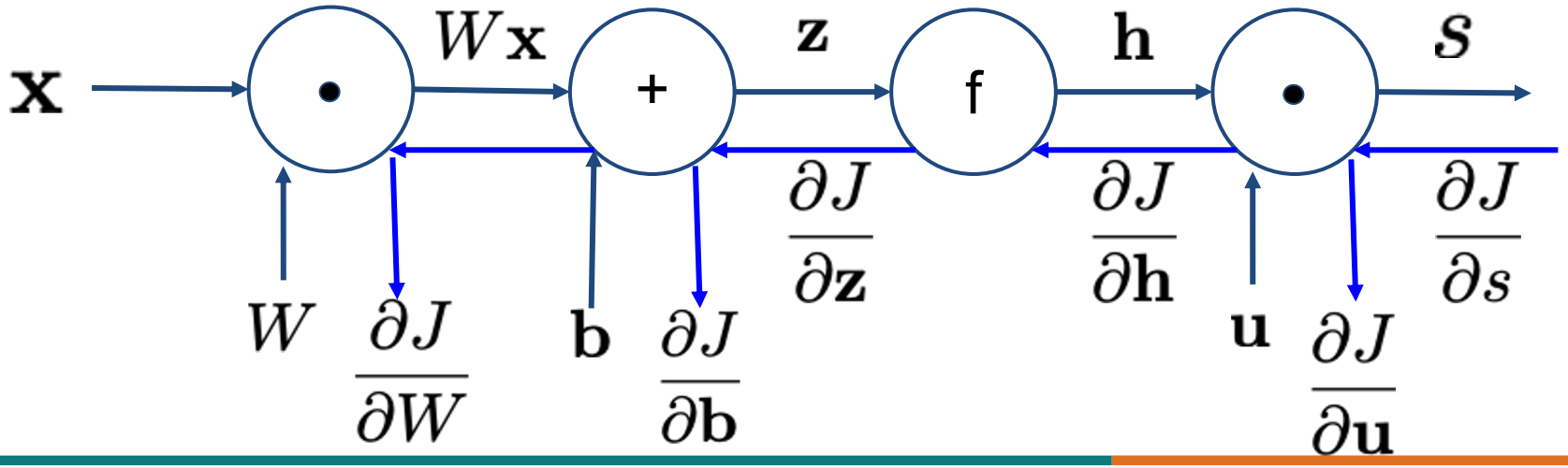
Forward propagation



Backpropagation

- Go backwards along edges
  - Pass along gradients

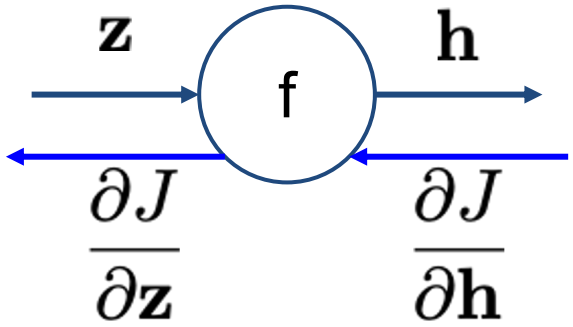
$$s = \mathbf{u}^\top \mathbf{h}$$
$$\mathbf{h} = f(\mathbf{z})$$
$$\mathbf{z} = W\mathbf{x} + \mathbf{b}$$
$$\mathbf{x} \text{ (input)}$$



Backpropagation: single node

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

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

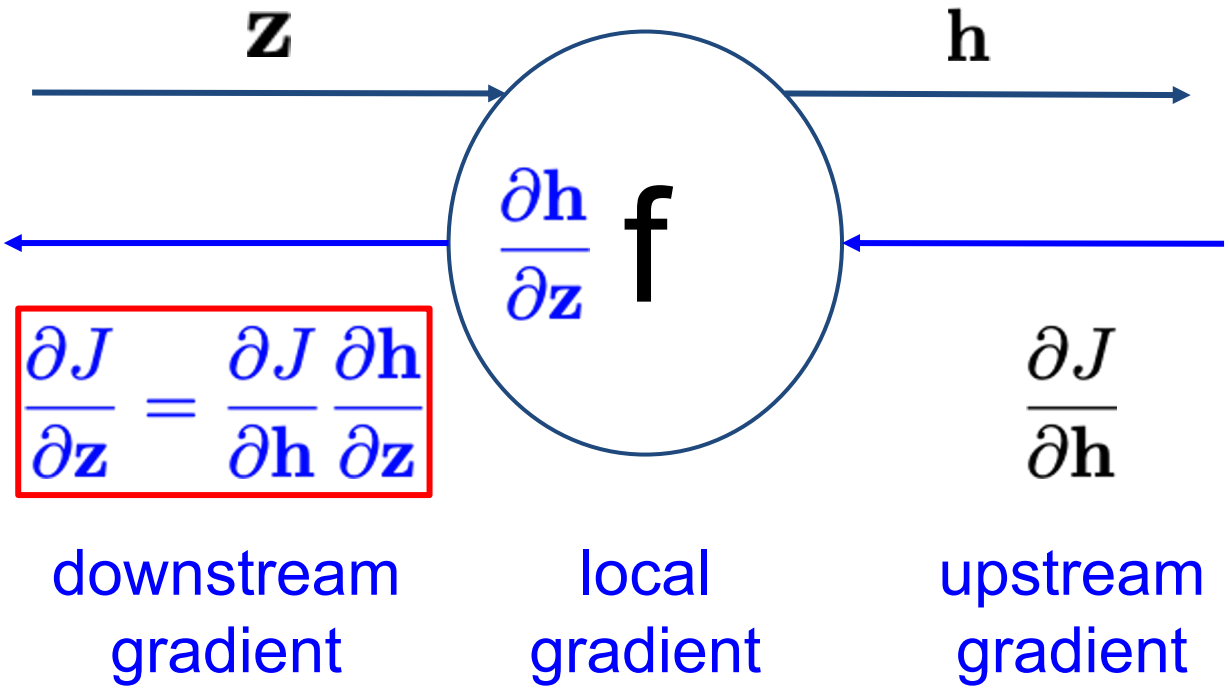


Backpropagation: single node

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

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

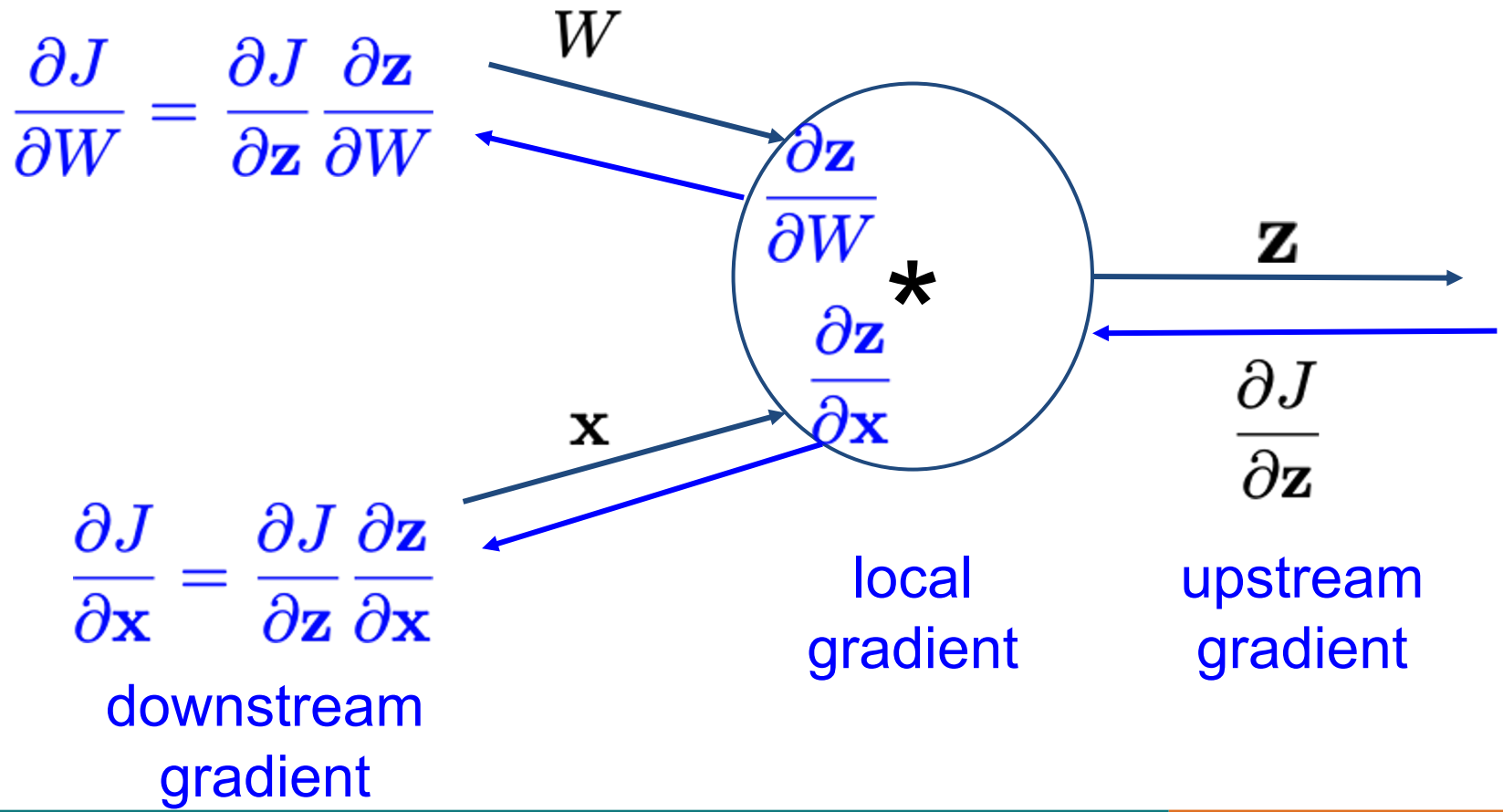
- [downstream gradient] = [upstream gradient]x[local gradient]



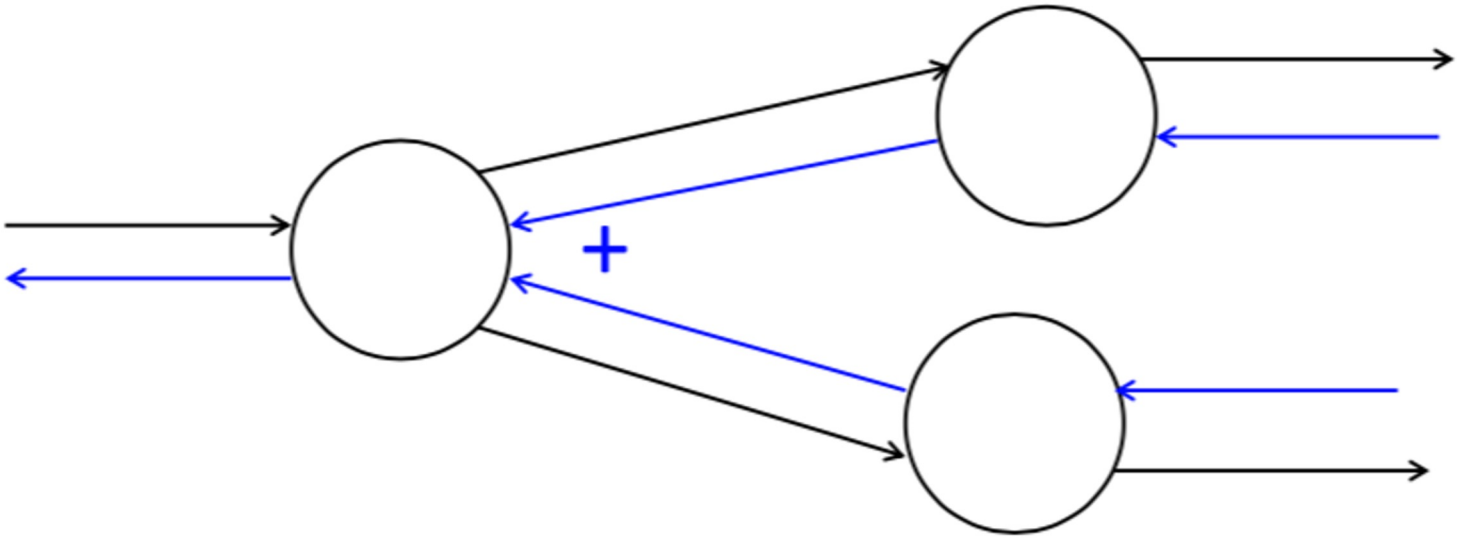
Backpropagation: single node

- What about nodes with multiple inputs?
- multiple inputs -> multiple local gradients

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



Gradients sum at outward branches



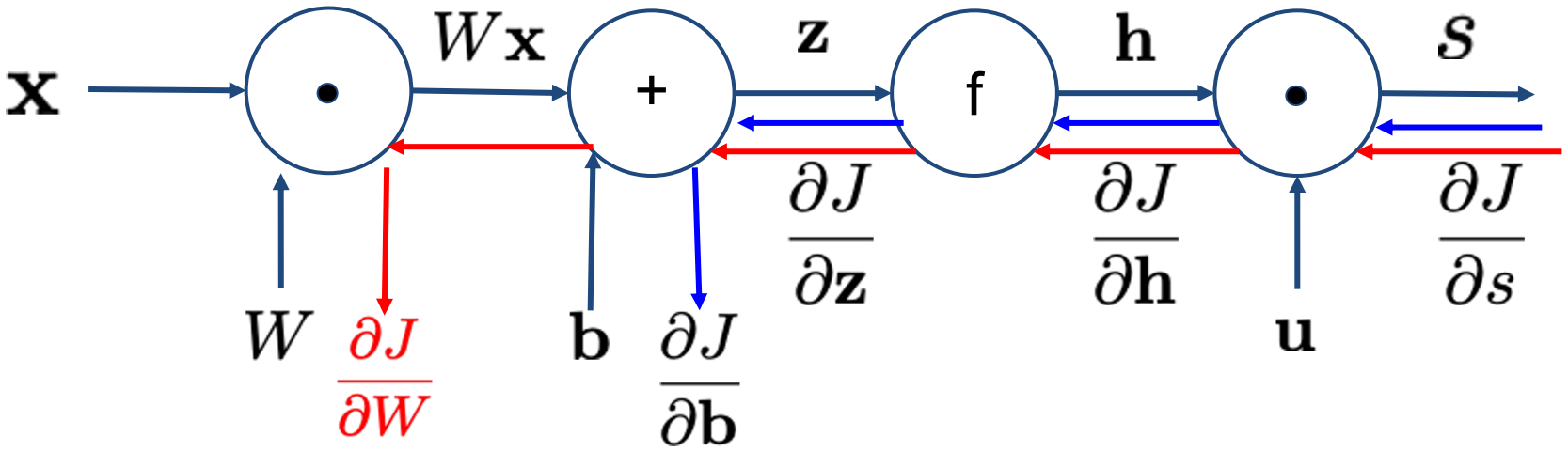
- a = x+y
- b= max(y,z)
- f = ab

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

Practice it. Detailed calculations are in the additional slides.

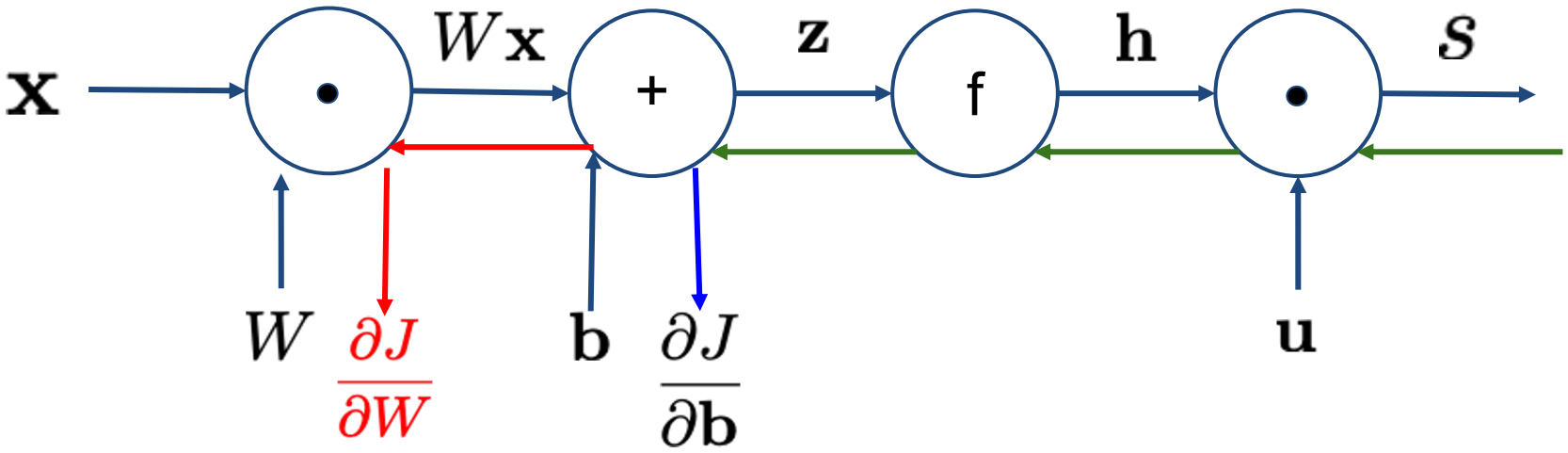
Efficiency: compute all gradients at once

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

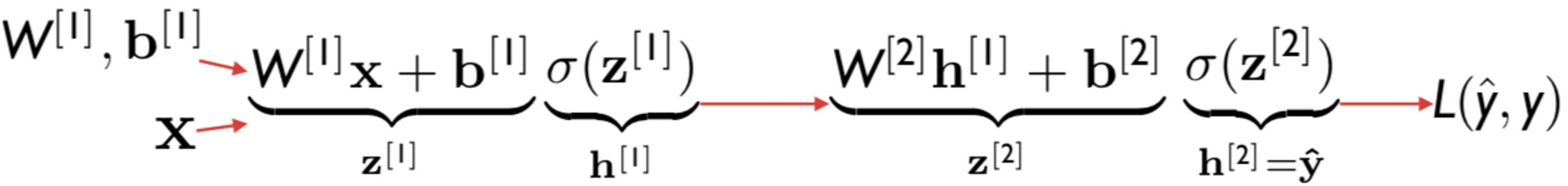


Efficiency: compute all gradients at once

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



Backpropagation



1. Break up equations

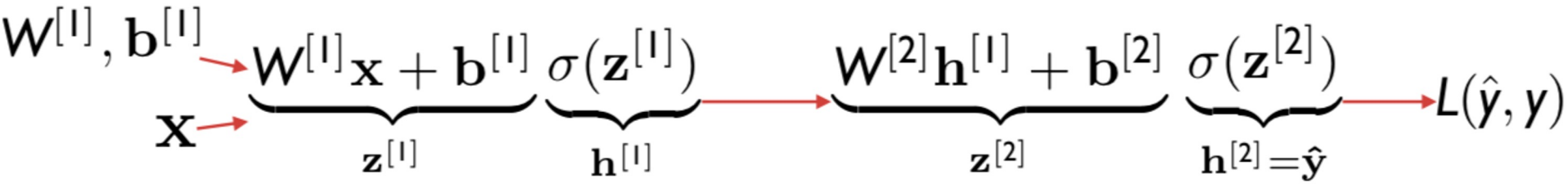
$$\hat{y} = \sigma(z^{[2]})$$

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

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

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

Backpropagation



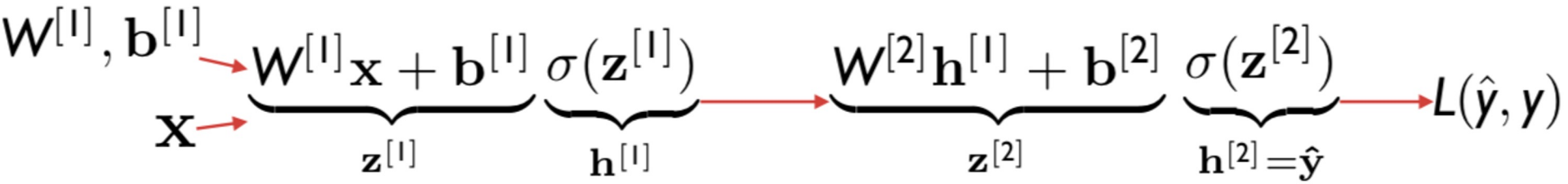
2. Apply Chain Rule

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



2. Apply Chain Rule

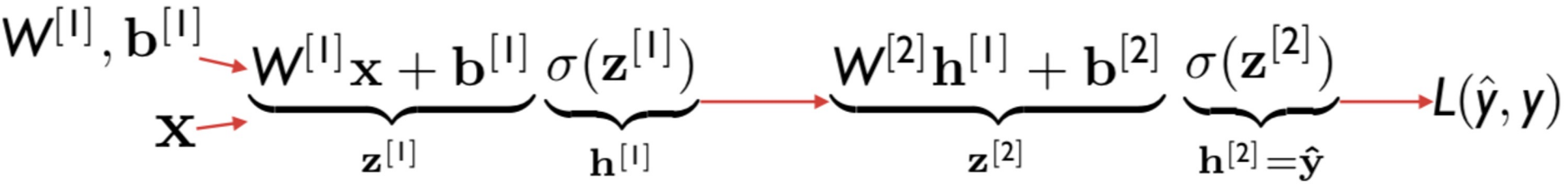
Binary loss for one example: $J(\theta) = -y \log \hat{y} - (1-y) \log(1-\hat{y})$

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

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

The same as last step, avoid duplicated computation

Backpropagation



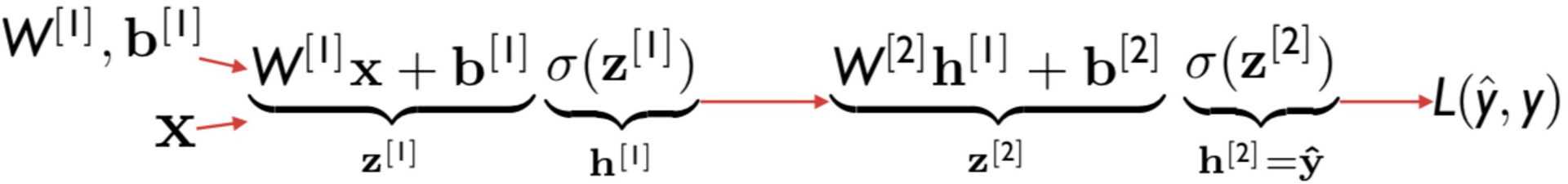
3. Write out the gradients

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



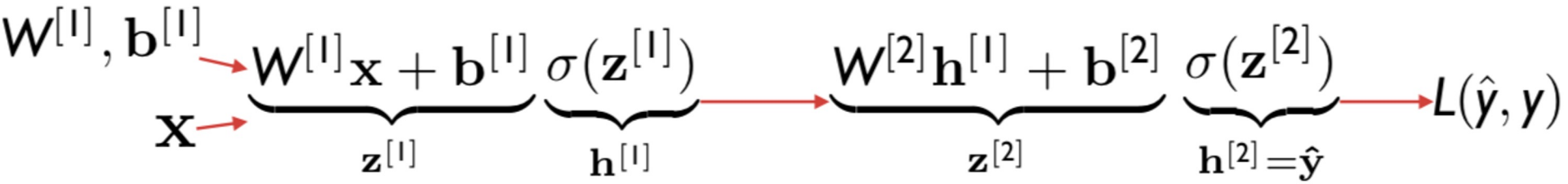
3. Write out the gradients

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



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

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

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

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

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

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

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

Optimization - summary

Feedforward

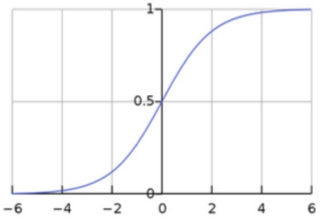
$$\mathbf{z}^{[1]} = W^{[1]}\mathbf{x} + b^{[1]}$$
$$\mathbf{h}^{[1]} = \sigma(\mathbf{z}^{[1]})$$
$$z^{[2]} = W^{[2]}\mathbf{h}^{[1]} + b^{[2]}$$
$$\hat{y} = h^{[2]} = \sigma(z^{[2]})$$

Backpropagation

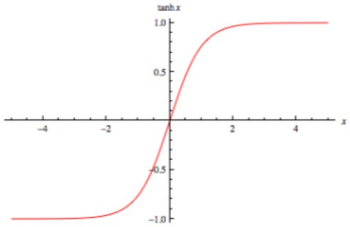
$$dz^{[2]} = h^{[2]} - y$$
$$dW^{[2]} = dz^{[2]}\mathbf{h}^{[1]\,T}$$
$$db^{[2]} = dz^{[2]}$$
$$d\mathbf{z}^{[1]} = W^{[2]\,T} dz^{[2]} \circ \mathbf{h}^{[1]} \circ (1 - \mathbf{h}^{[1]})$$
$$dW^{[1]} = d\mathbf{z}^{[1]}\mathbf{x}^{T}$$
$$d\mathbf{b}^{[1]} = d\mathbf{z}^{[1]}$$

Activation Functions

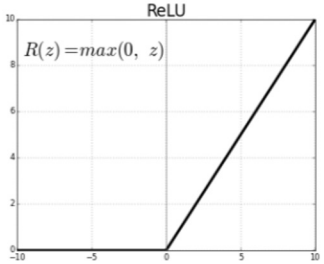
Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$



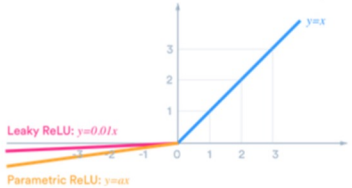
tanh: $f(x) = 2\sigma(2x) - 1$



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



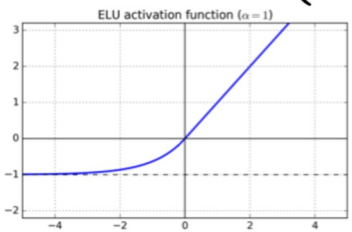
Leaky ReLU: $f(x) = \max(\alpha x, x)$



Maxout

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

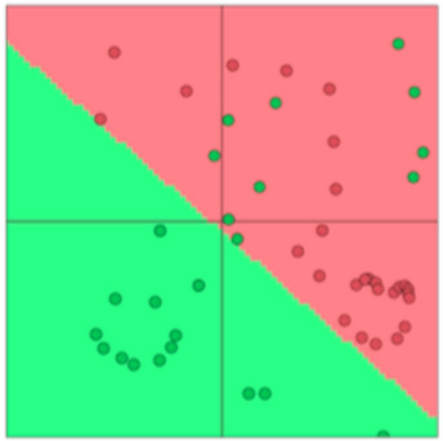
ELU: $f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

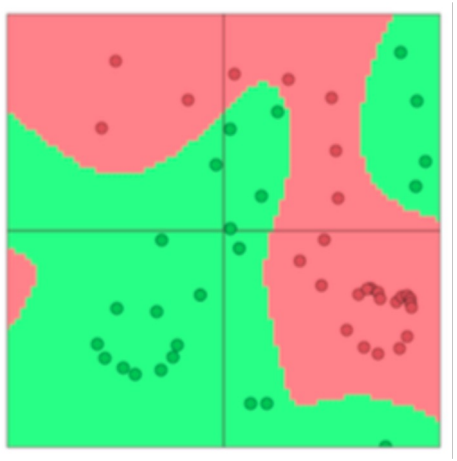


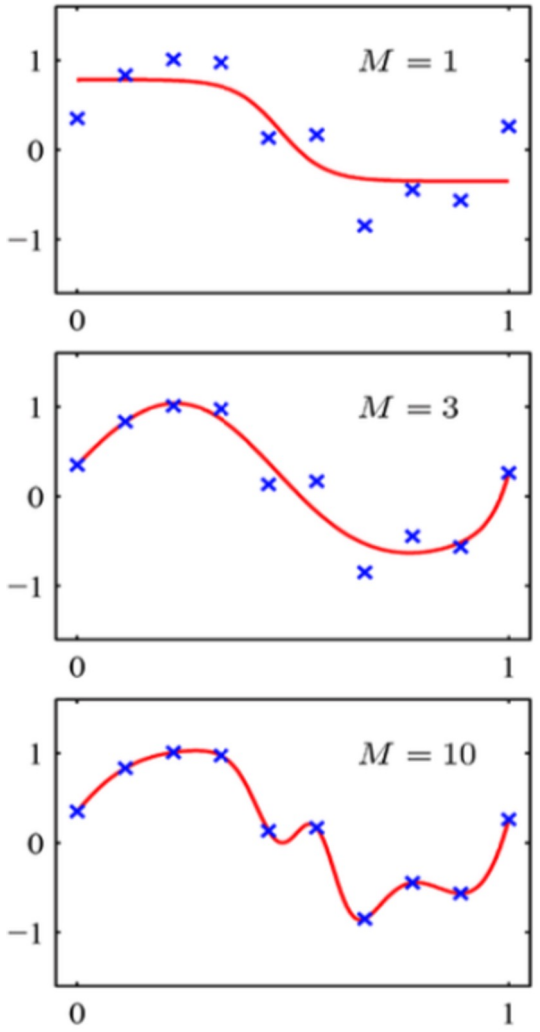
For building a feed-forward deep network, the first thing you should try is ReLU — it trains quickly and performs well due to good gradient backflow

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.







Why learn all these details about gradients

- Modern deep learning frameworks compute gradients for you
- But 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:
    - https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b
  - Example in future lecture: exploding and vanishing gradients

Parameter Initialization

- Initialize weights to small random values.
- 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
- Normal Xavier initialization draw the weight from a normal distribution, with a mean of 0, and variance inversely proportional to fan-in n_{in} (previous layer size) and fan-out n_{out} (next layer size)

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

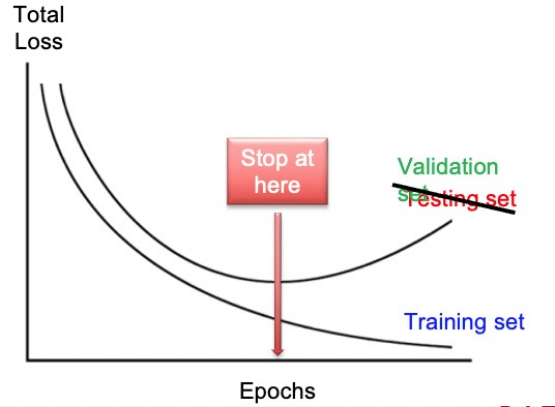
Regularization

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

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log\left(\frac{\exp f_k}{\sum_{c=1}^{K}\exp f_c}\right) + \lambda\sum_{j=1}^{d} w_{kj}^2 \qquad \theta = \{w_1, w_2, \ldots\}$$

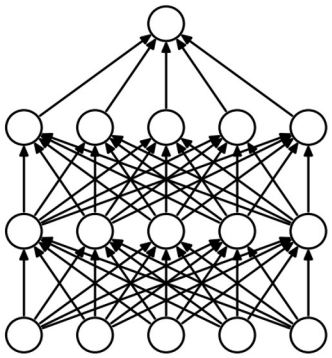
L1 regularization: $\|\theta\|_1 = |w_1| + |w_2| + \ldots$ L2 regularization: $\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \ldots$

- Early stopping: a popular regularization technique. Monitor the performance for every epoch on the validation set during the training. Terminate the training as soon as the validation error reaches a minimum.

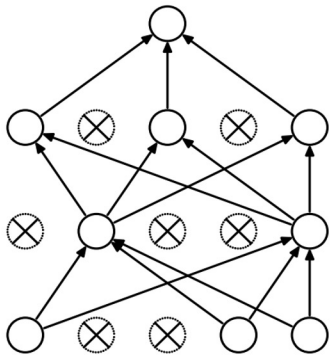


Dropout

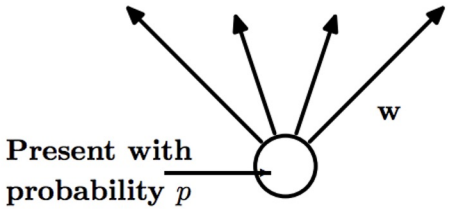
- Dropout (Srivastava et al.,2014) provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- At training (each iteration): forces the network to make decision based on part of the features.
  - each neuron is either retained with probability p or dropped out with probability 1- p. The probability p is a hyperparameter.
  - The weights of the dropped-out neurons are not updated during training.
- At test (prediction): the dropout is turned off.
  - the network is used as a whole.
  - the weights are scaled-down by a factor of p.



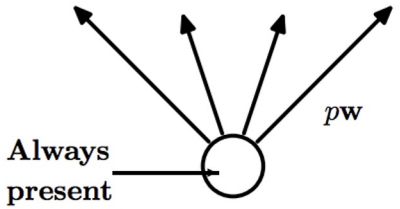
(a) Standard Neural Net



(b) After applying dropout.



(a) At training time



(b) At test time

Wager, Wang, & Liang. Dropout Training as Adaptive Regularization. In NIPS, 2013.

Readings

- CH2-8 DL
- CH2-5 NNLP

- Wager et al. Dropout Training as Adaptive Regularization. 2013
- Ning Qian. On the momentum term in gradient descent learning algorithms
- Ruder et al. An overview of gradient descent optimization algorithms. 2017

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

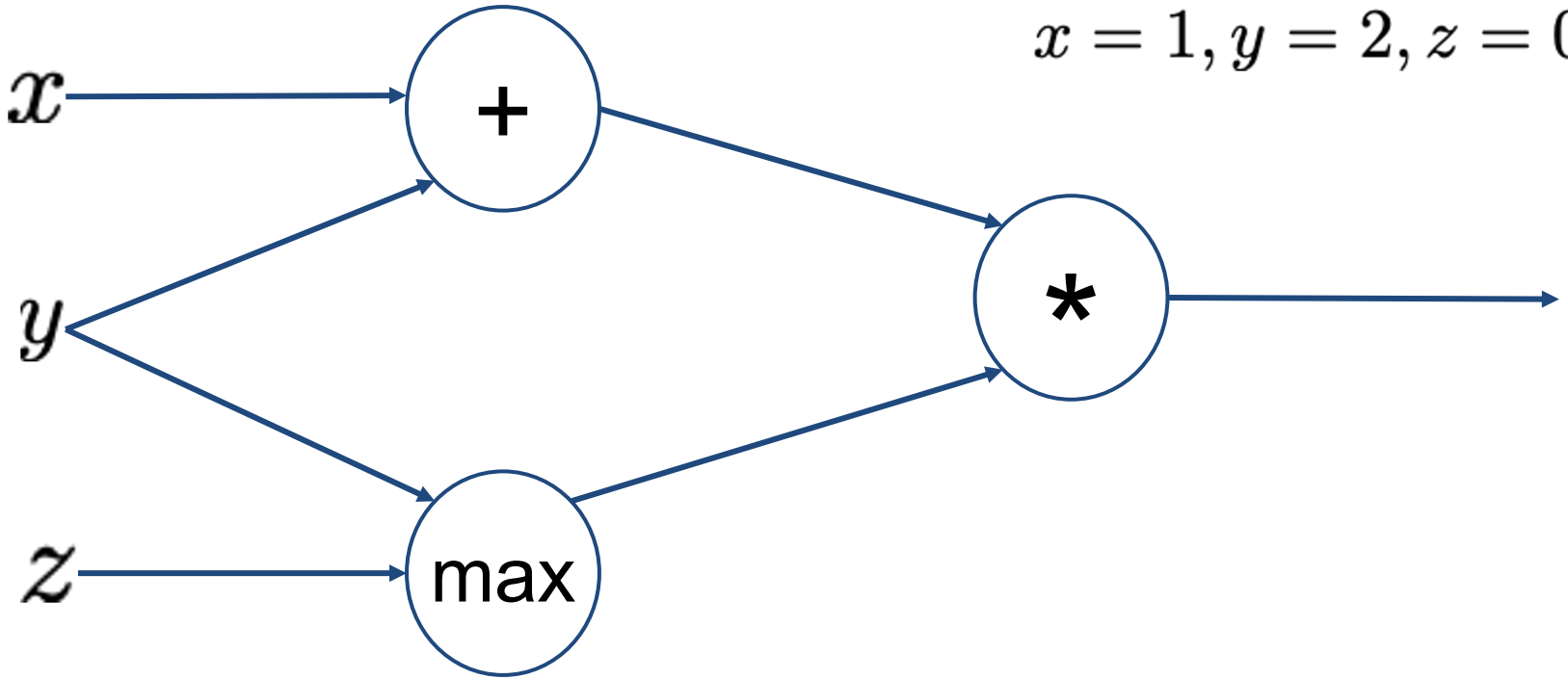
stevens.edu

Thank You

An example of Gradients Calculation

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

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

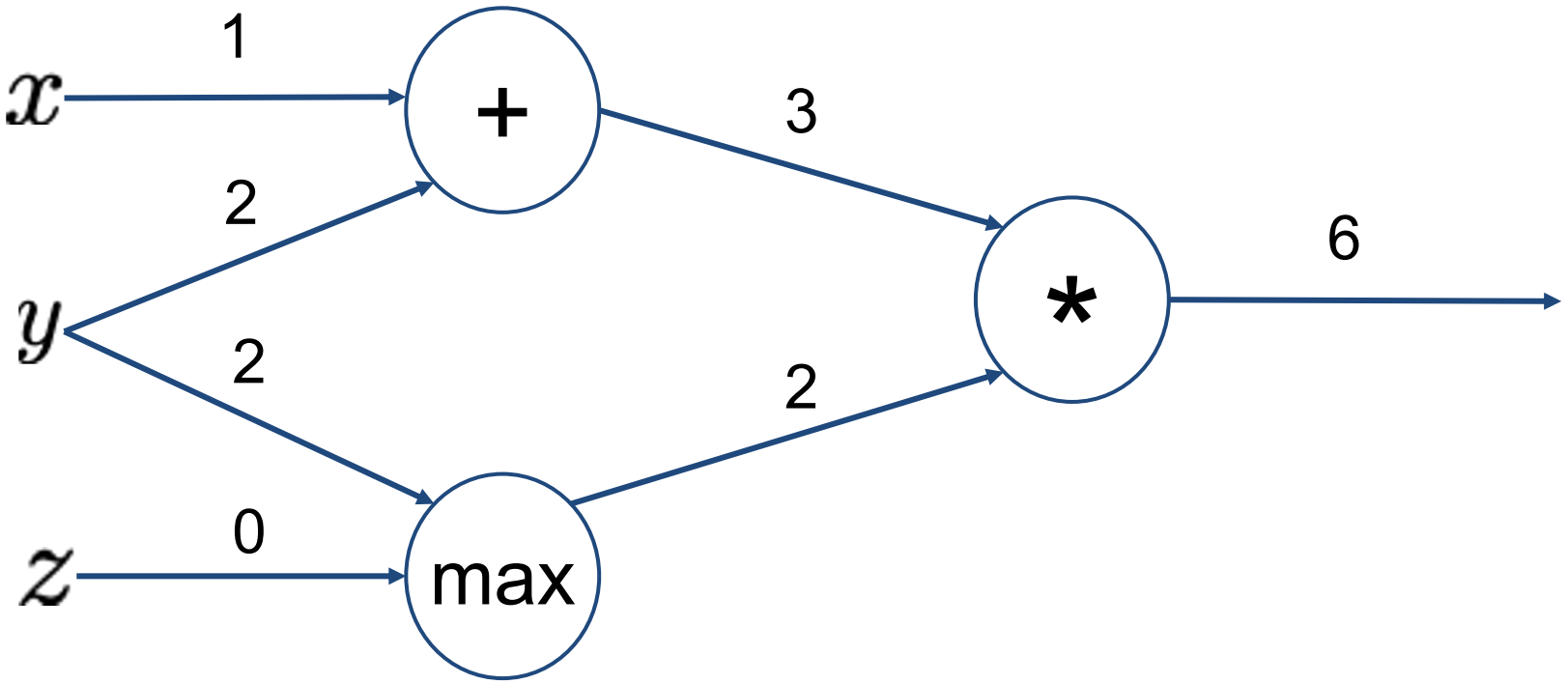


An example

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

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

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



An example

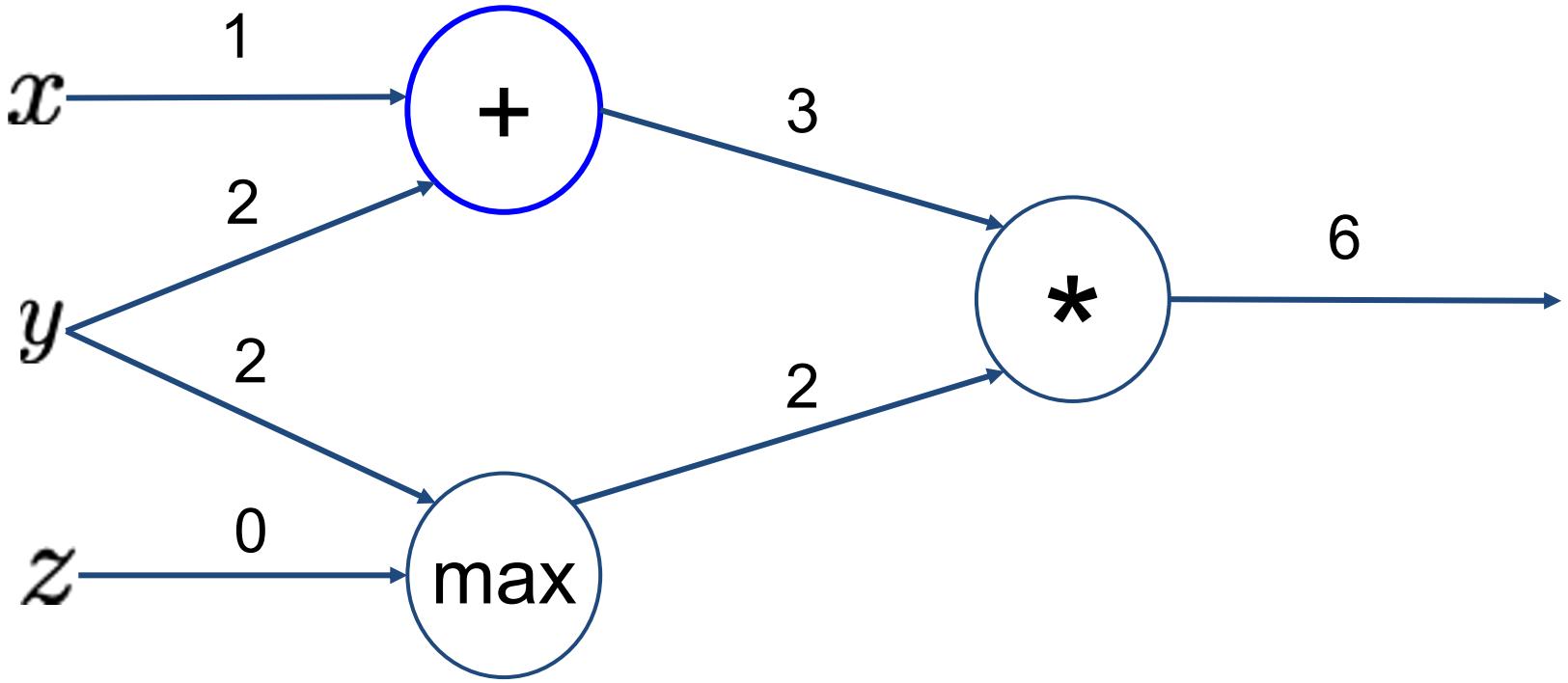
$$f(x, y, z) = (x + y)\max(y, z)$$

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

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

- Local gradients:

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



An example

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

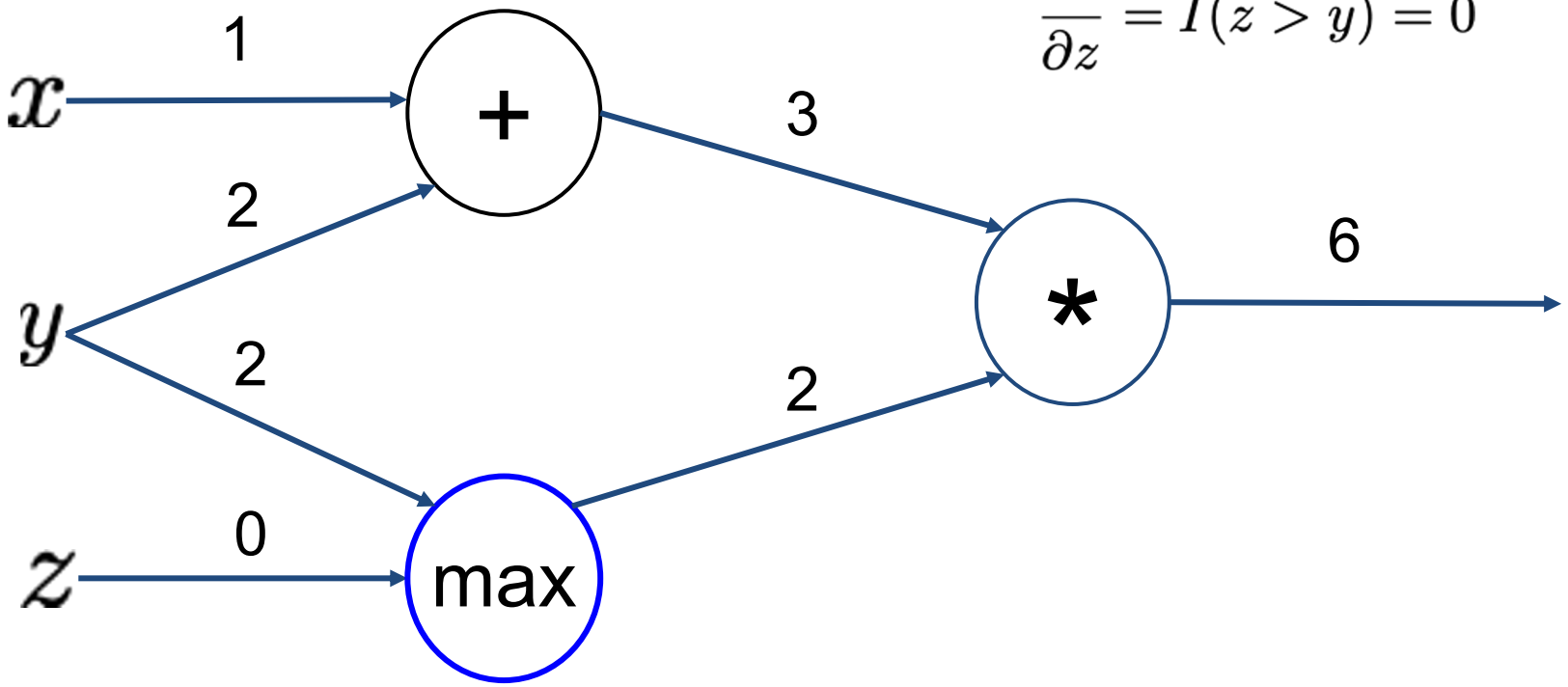
$$x = 1, y = 2, z = 0$$

- 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

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

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

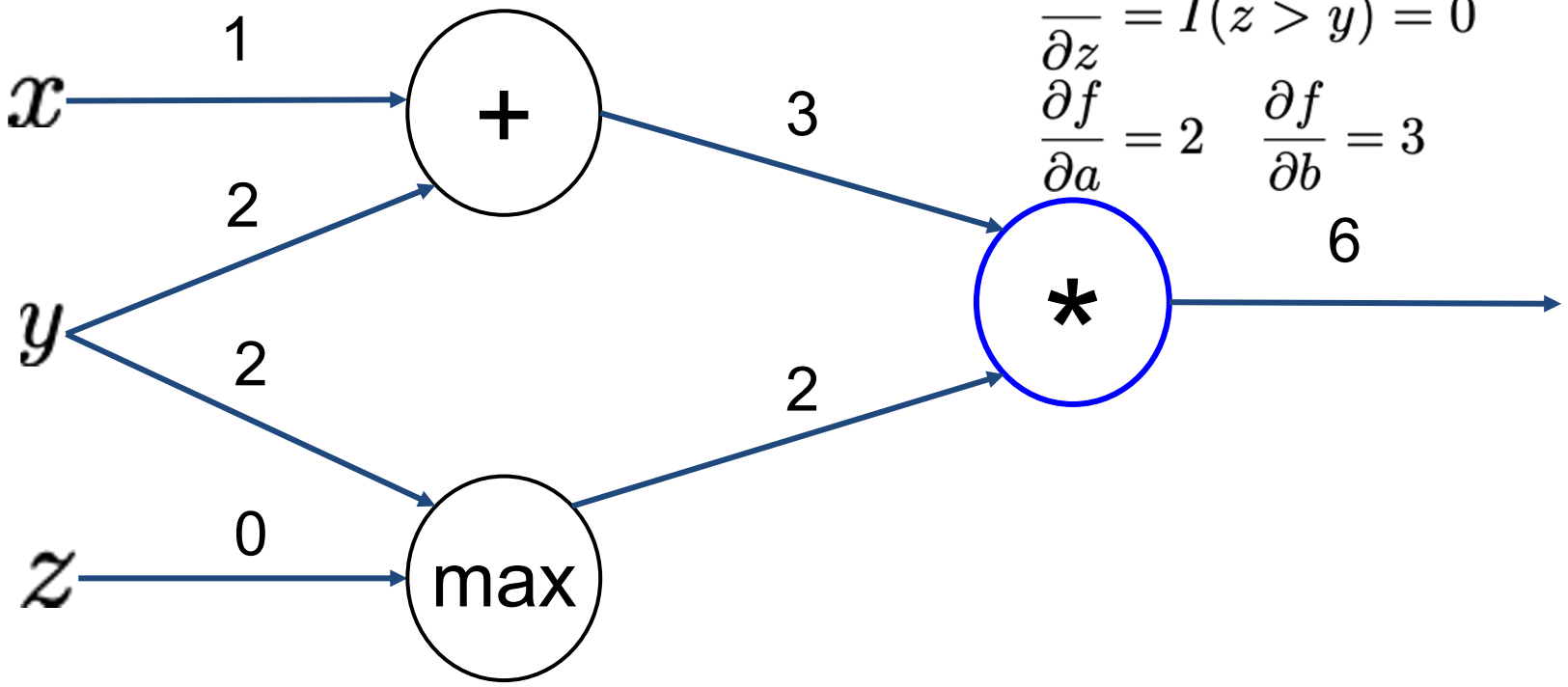
- Forward prop steps:
- a = x+y
- b= max(y,z)
- f = ab

- 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

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

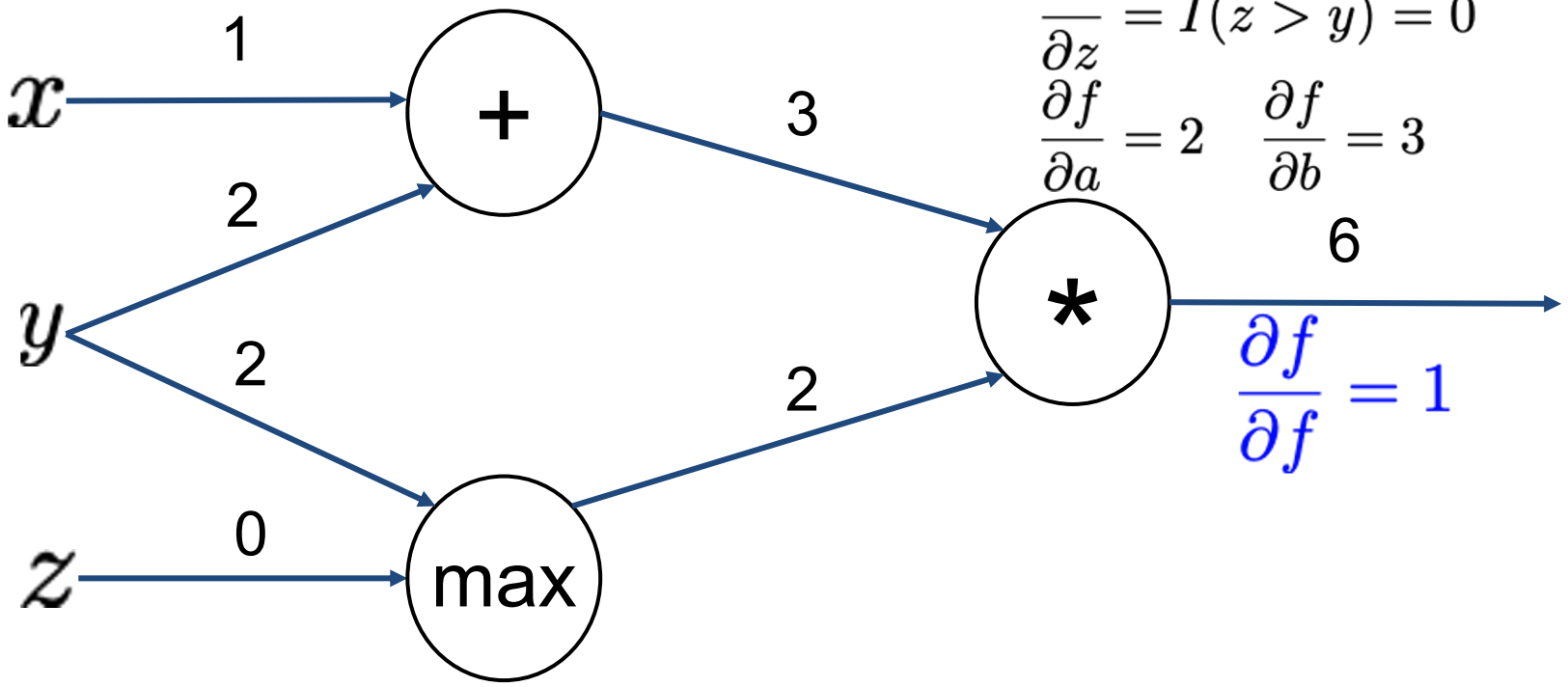
$$x = 1, y = 2, z = 0$$

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

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

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

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



An example

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

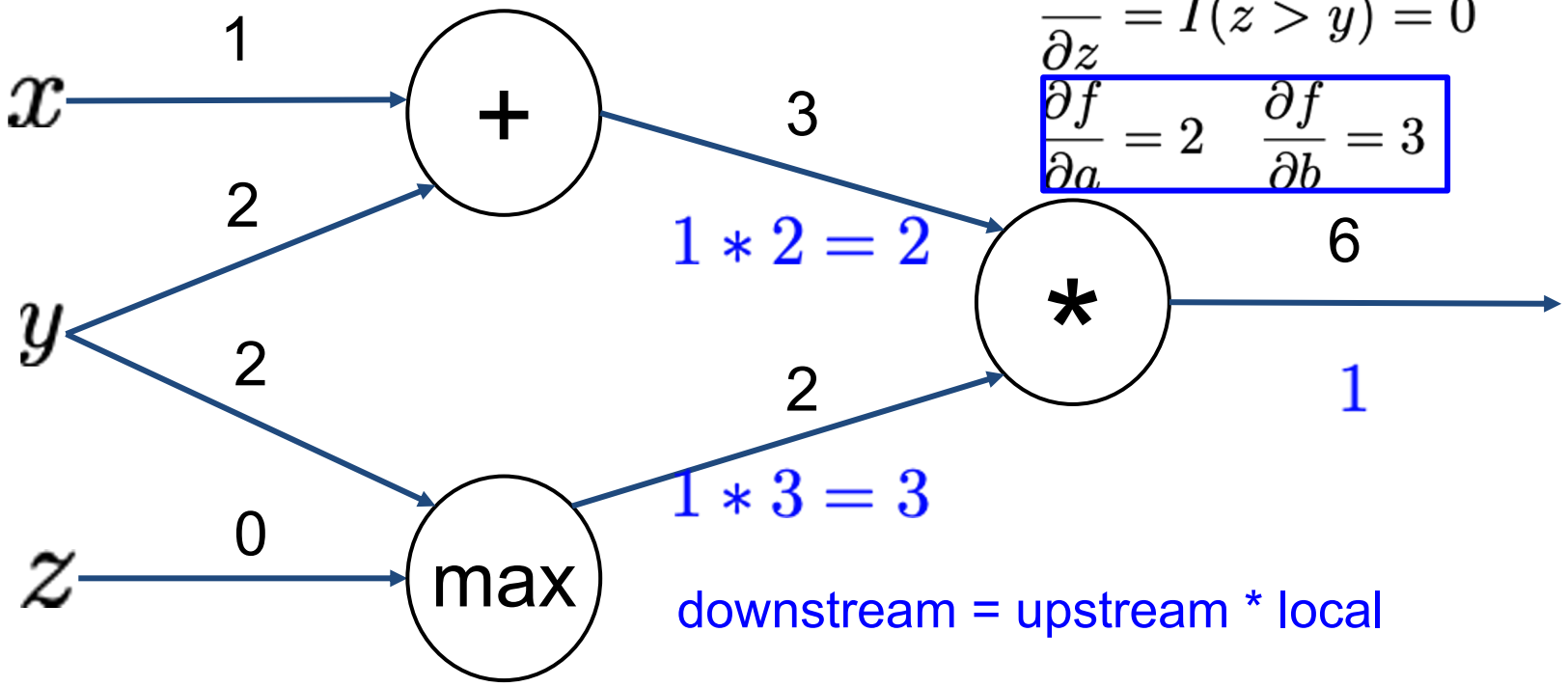
$$x = 1, y = 2, z = 0$$

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

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

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

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



An example

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

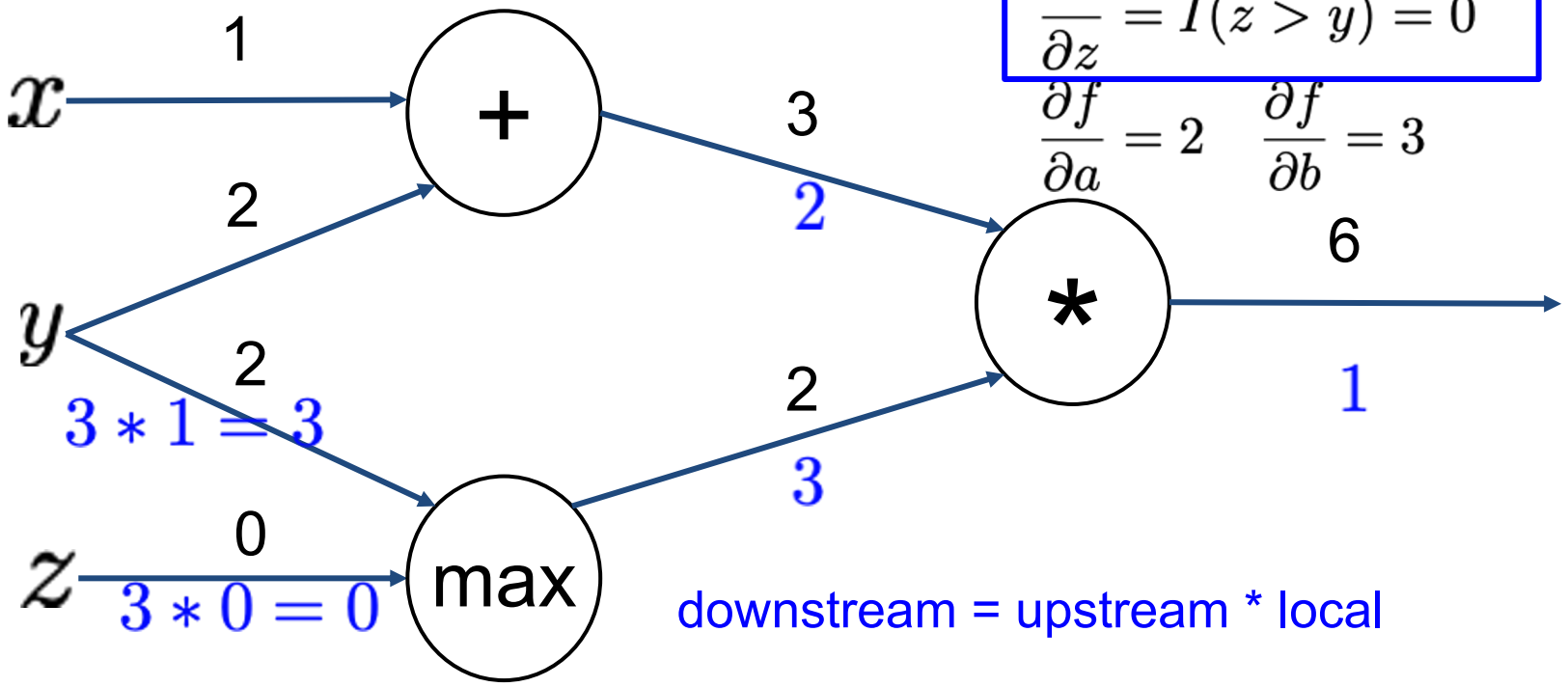
$$x = 1, y = 2, z = 0$$

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

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

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

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



downstream = upstream * local

An example

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

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

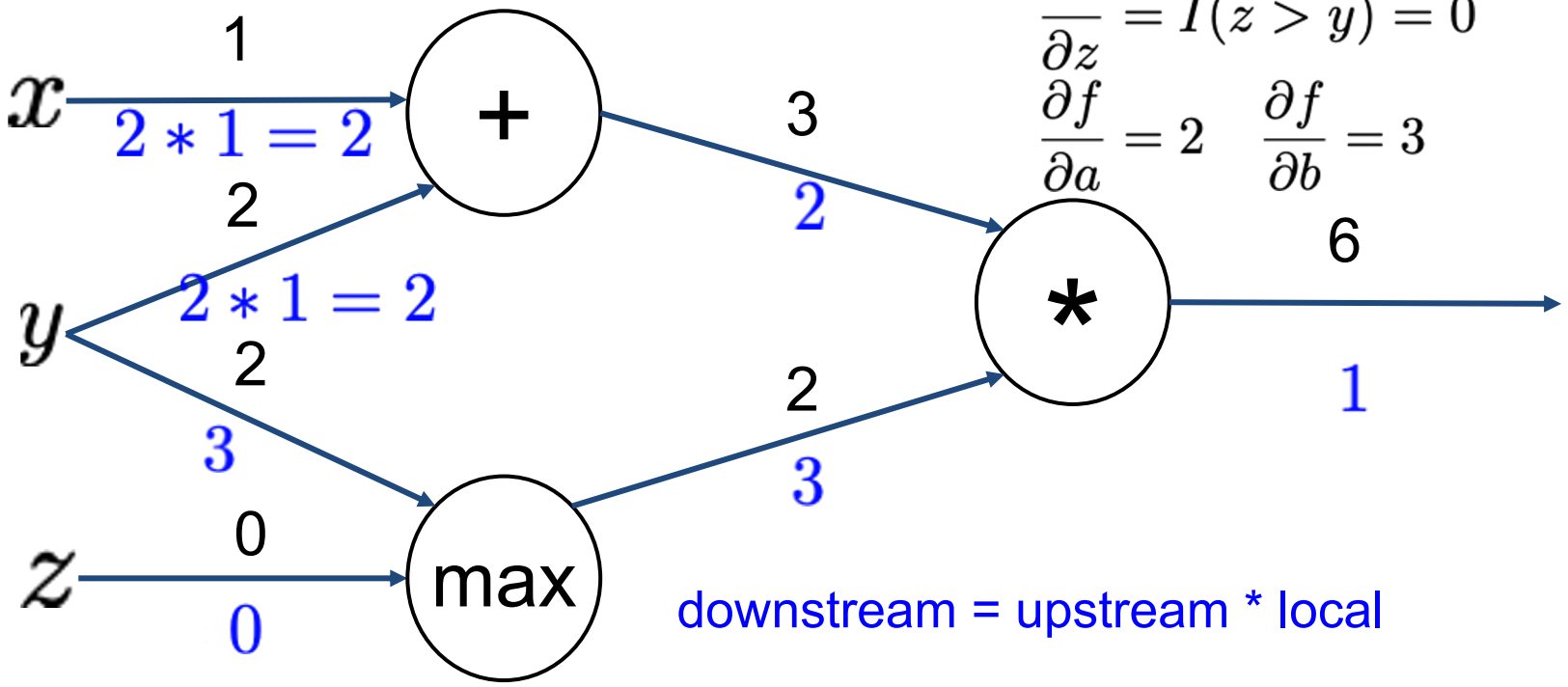
- Backpropagation
- Local gradients:

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

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

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

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



An example

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

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

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

$\frac{\partial f}{\partial x} = 2$

$\frac{\partial f}{\partial y} = 2 + 3 = 5$

$\frac{\partial f}{\partial z} = 0$

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

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

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