Backprop

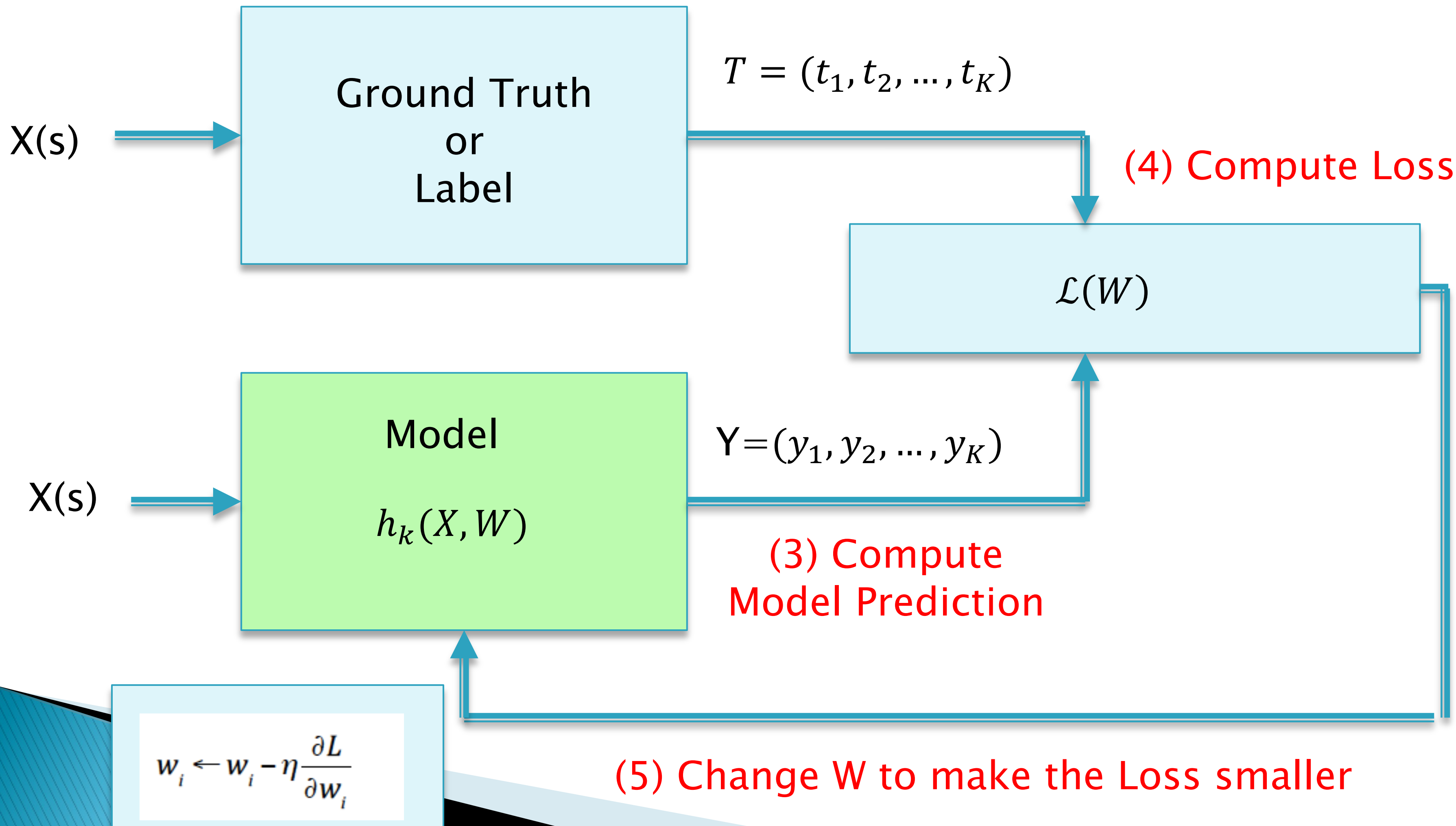
Lecture 5

Subir Varma

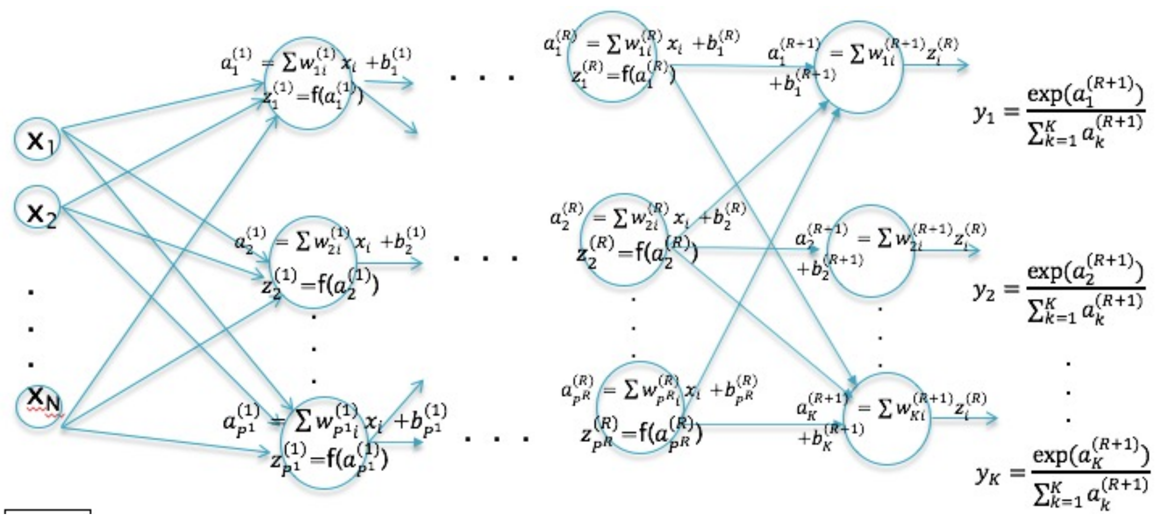
Framework for Supervised Learning

(1) Collect Labeled Data

(2) Choose Model $h_k(X,W)$



What Problem are we Solving?



$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

Need a way of Efficiently Computing $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ for EVERY Weight!!

224x224x3 image with a 100 node layer → 15 million weights!

Numerical Differentiation

$$\frac{\partial L(w_1, w_2, \cdots, w_i, \cdots, w_n)}{\partial w_i} \approx \frac{L(w_1, w_2, \cdots, w_i + \Delta w_i, \cdots, w_n) - L(w_1, w_2, \cdots, w_i, \cdots, w_n)}{\Delta w_i}$$

What is wrong with this??

With a million weights, need million and one passes through the network to compute all the derivatives!!!

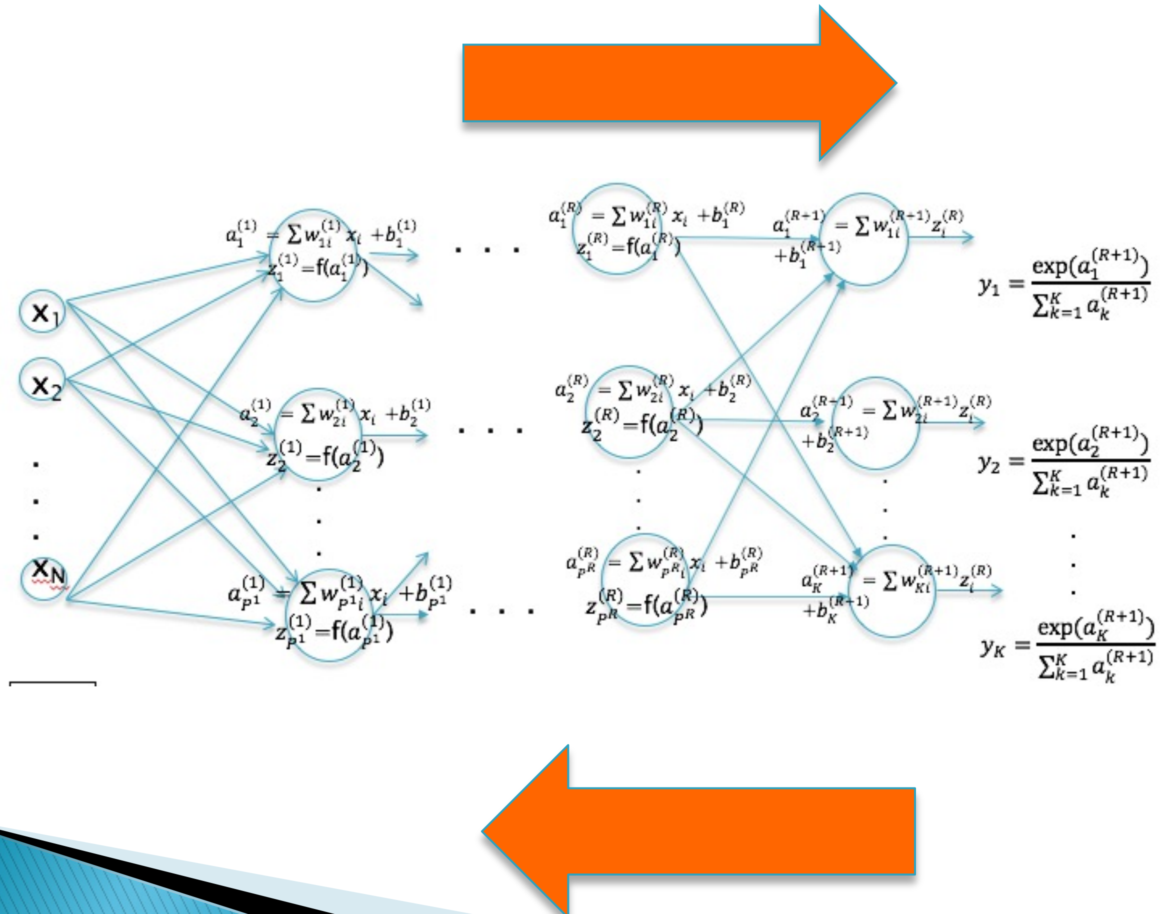
Historical Context

- By the late 1960s, people realized that hidden layers were needed to increase the modeling power of Neural Networks.
- There was little progress in this area until the mid-1980s, since there was no efficient algorithm for computing $\frac{\partial \mathcal{L}}{\partial w}$
- The Backprop algorithm (1986) met this need, and today remains a key part of the training scheme for all kinds of new deep architectures that have been discovered since then.

Using Backprop

- Backprop requires only TWO passes to compute ALL the derivatives, irrespective of the size of the network!

FORWARD PASS
Compute the Node Activations z and output y



$$\frac{\partial \mathcal{L}}{\partial w} = z\delta$$

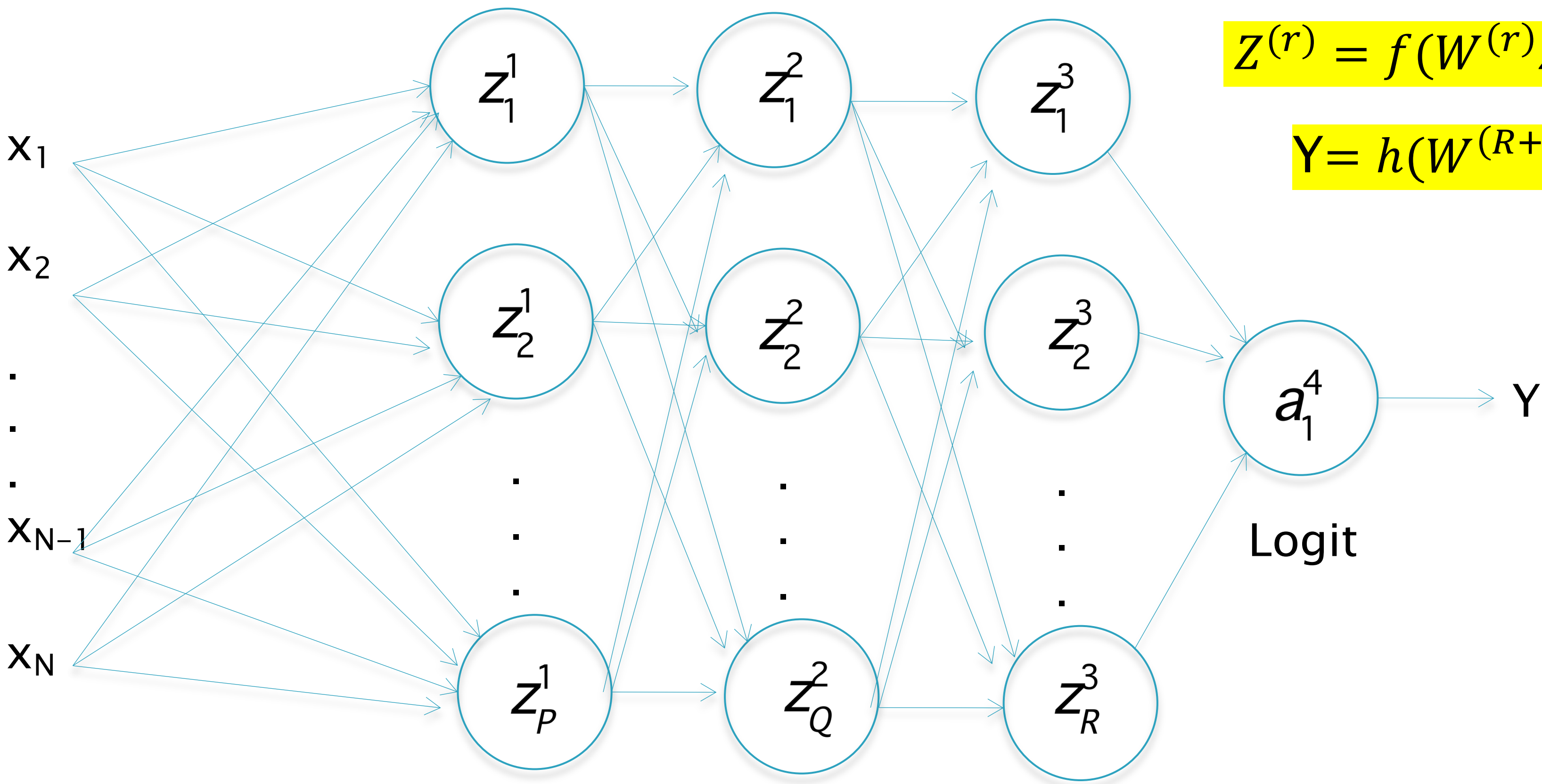
BACKWARD PASS
Compute the Node Gradients δ

Backprop – Forward Pass

$Z^{(1)} = f(W^{(1)}X)$

$Z^{(r)} = f(W^{(r)}Z^{(r-1)})$

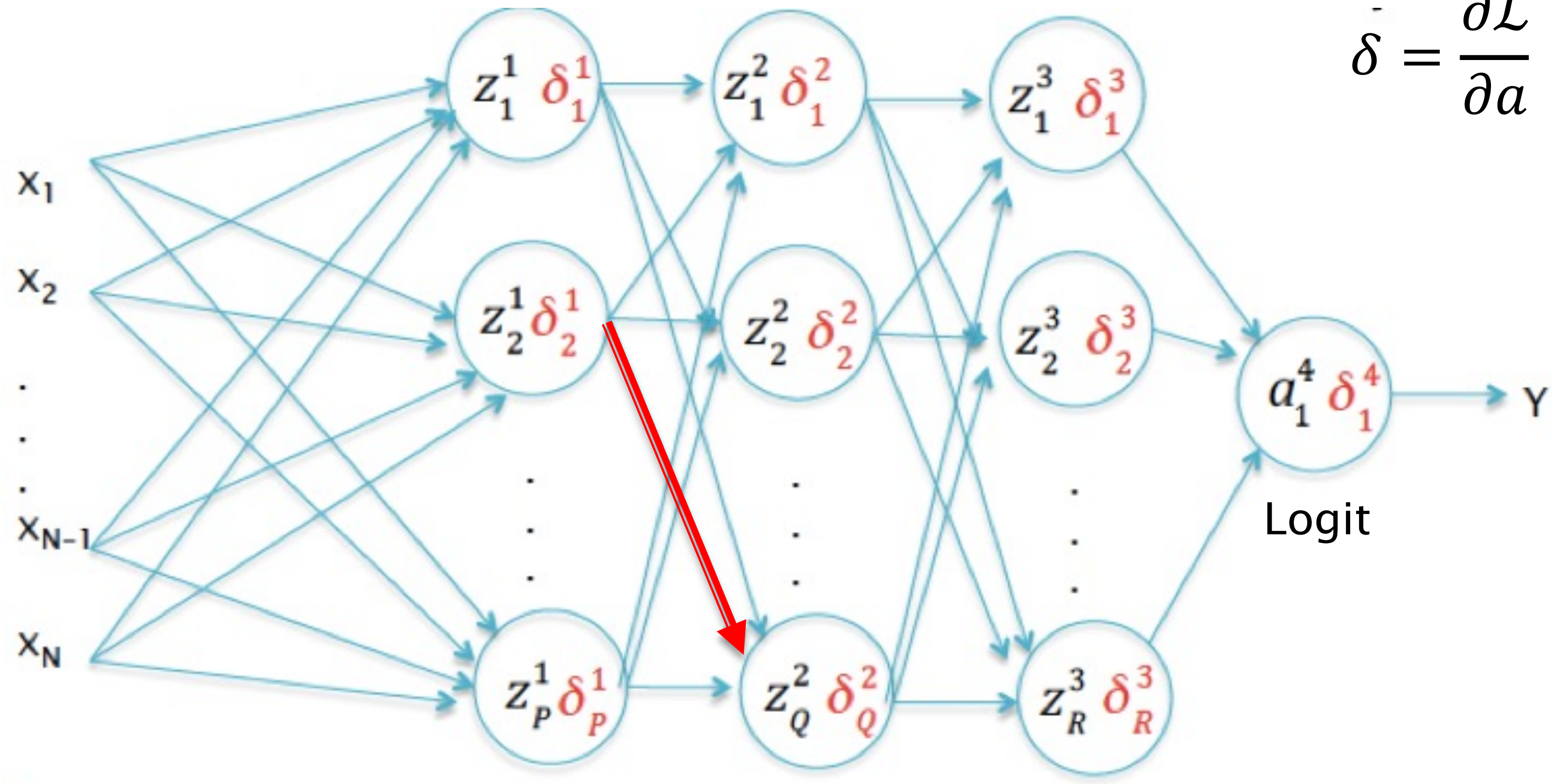
Y$= h(W^{(R+1)}Z^{(R)})$



Given an input vector X, compute the activations z for each neuron in the network

Backprop – Backward Pass

$$\delta = \frac{\partial \mathcal{L}}{\partial a}$$



$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(r)}} = z_j^{(r)}\, \delta_i^{(r+1)}$$

Today's Class

```
import keras
keras.__version__
```

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

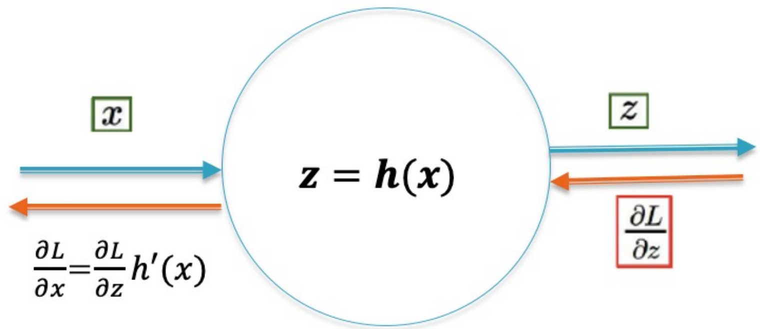
Backprop

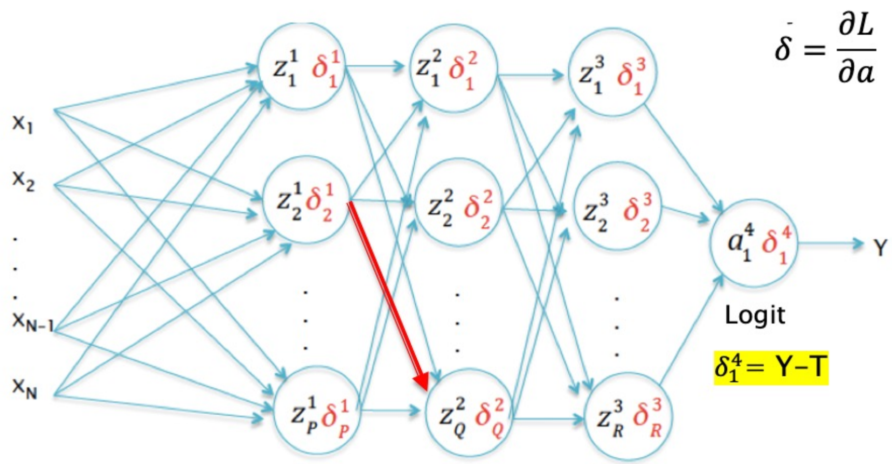
```
network.compile(optimizer='sgd',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

```
history = network.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)
```

Gradient Flow Calculus

Strategy for $\delta = \frac{\partial L}{\partial a}$ Computation

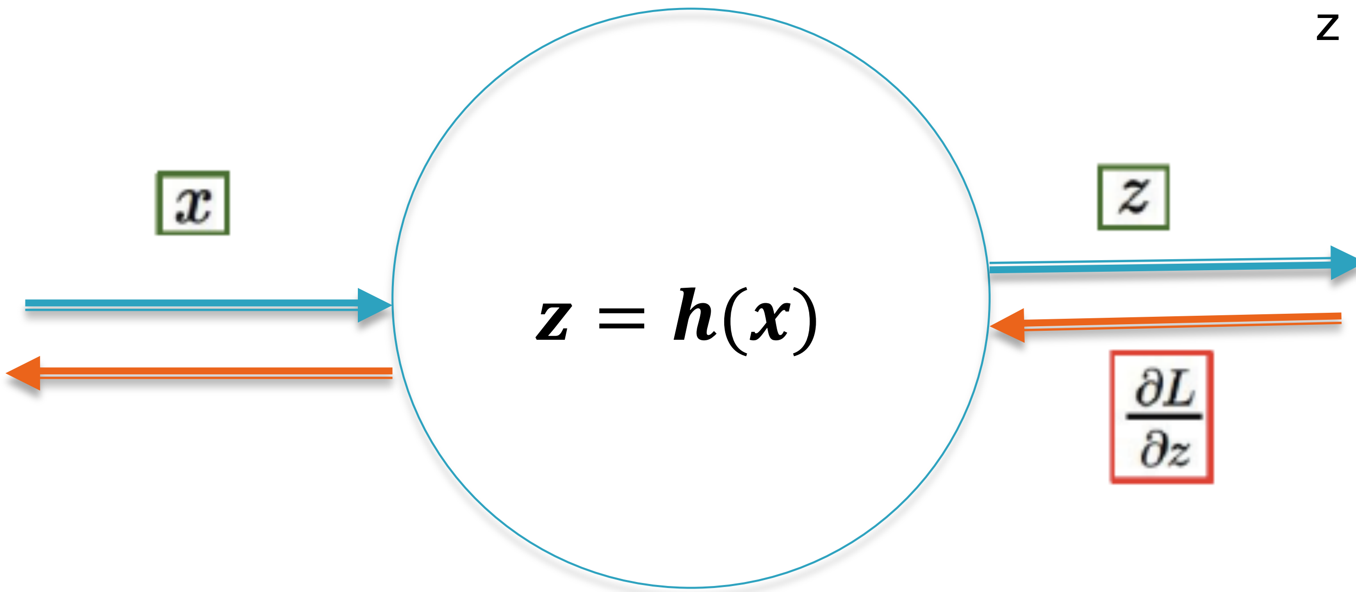




1. Compute the gradients $\frac{\partial L}{\partial a}$ at the final Logit Layer
2. Figure out how the gradients change as they traverse a single node
   in the network
3. Apply these rules to the whole network, one node (layer) at
   a time.

Gradient Flow Calculus

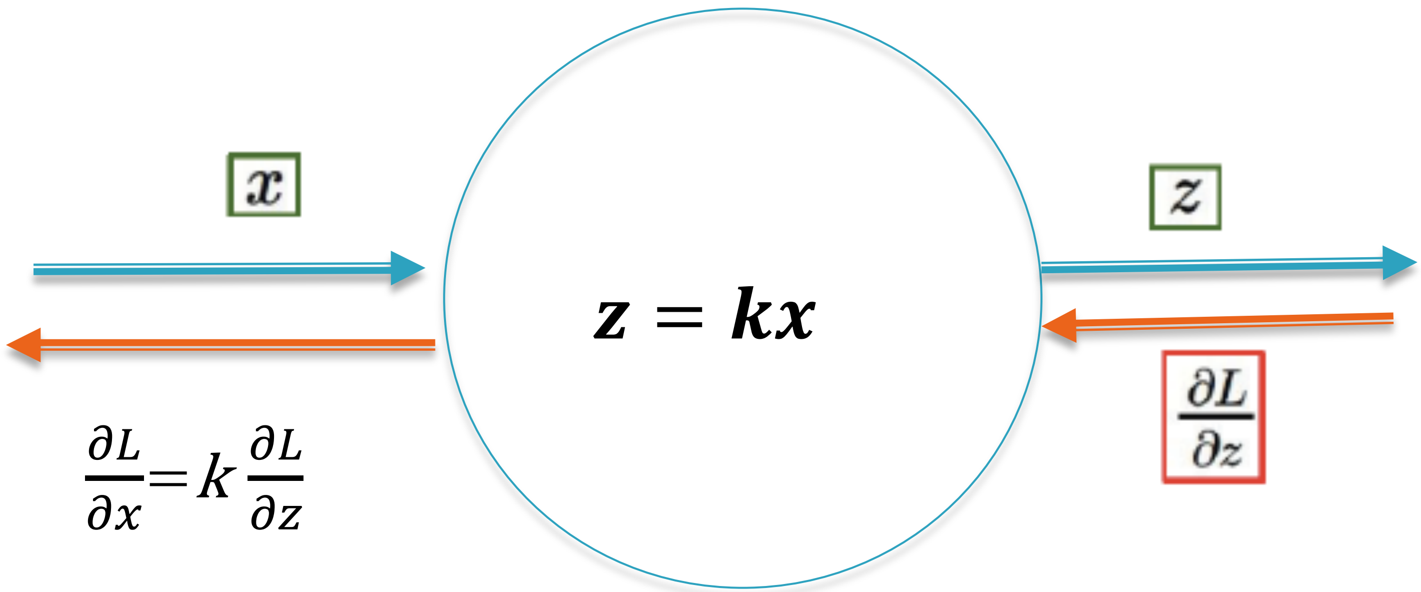
L = f(z)
z = h(x)



$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial L}{\partial z} h'(x)$$

By Chain Rule of Derivatives

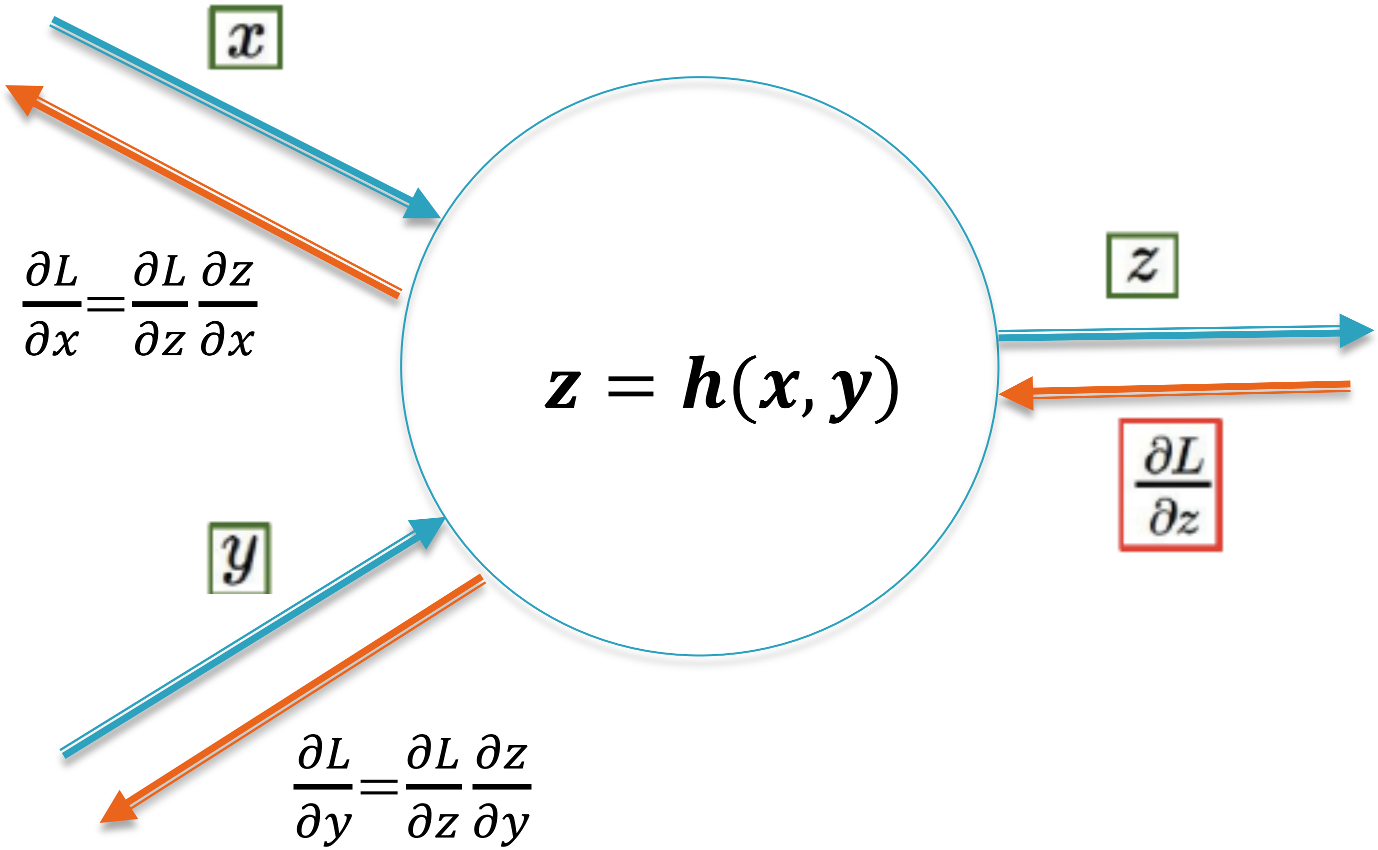
Gradient Flow Calculus: Multiplication by a Constant



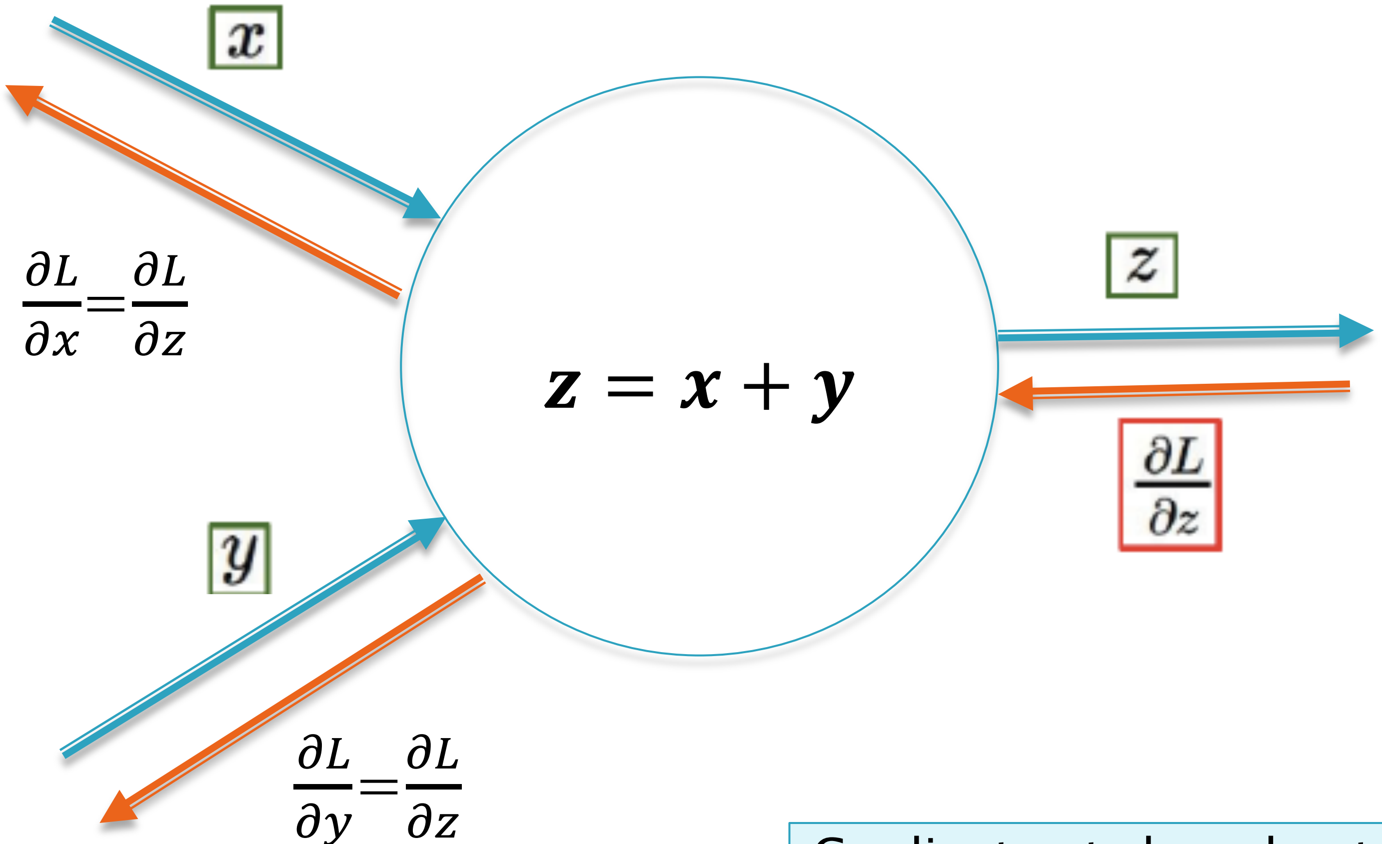
Gradient Multiplied by same Constant

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}$$

Gradient Flow Calculus



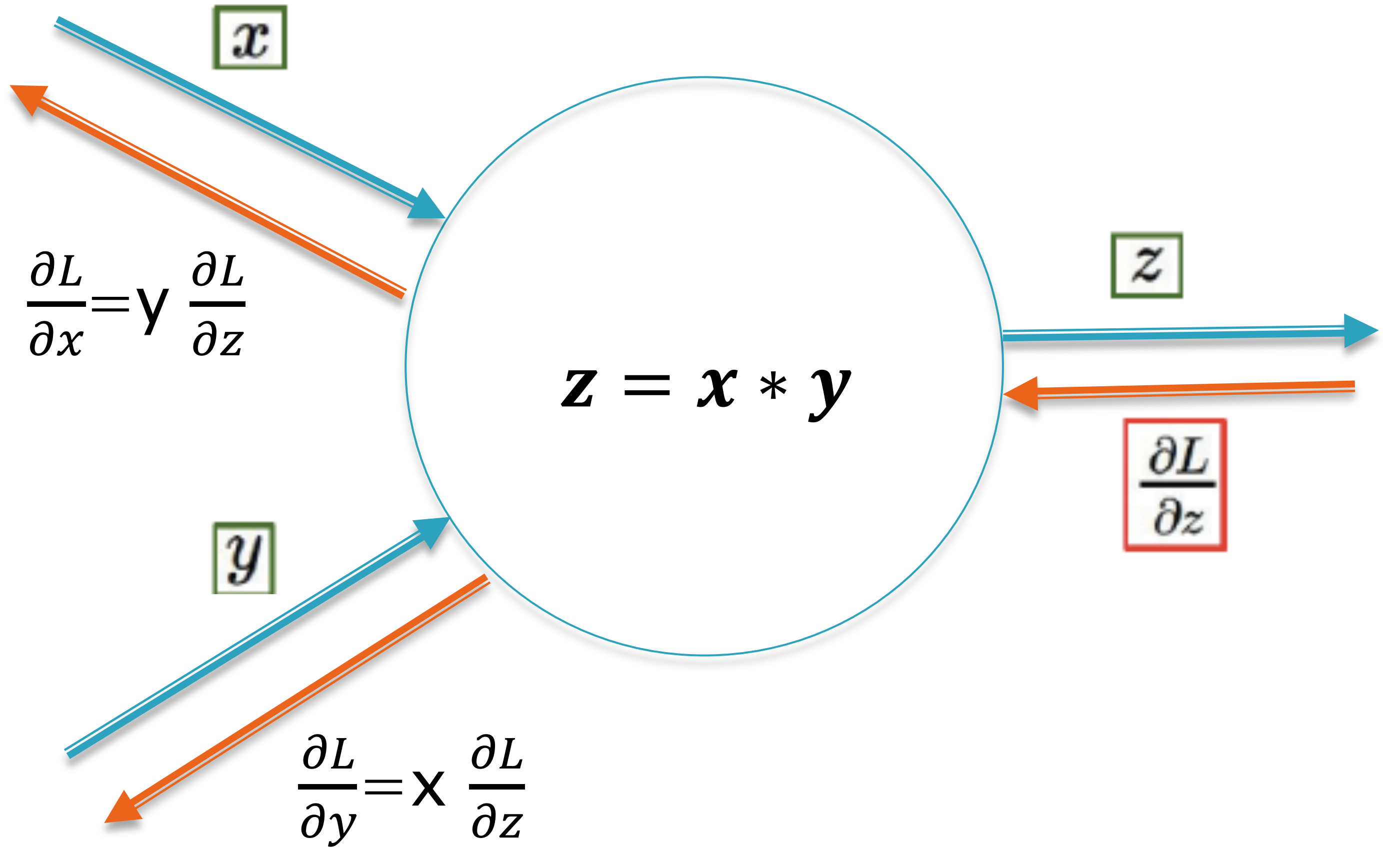
Gradient Flow Calculus: Addition



Gradient gets broadcast across all branches

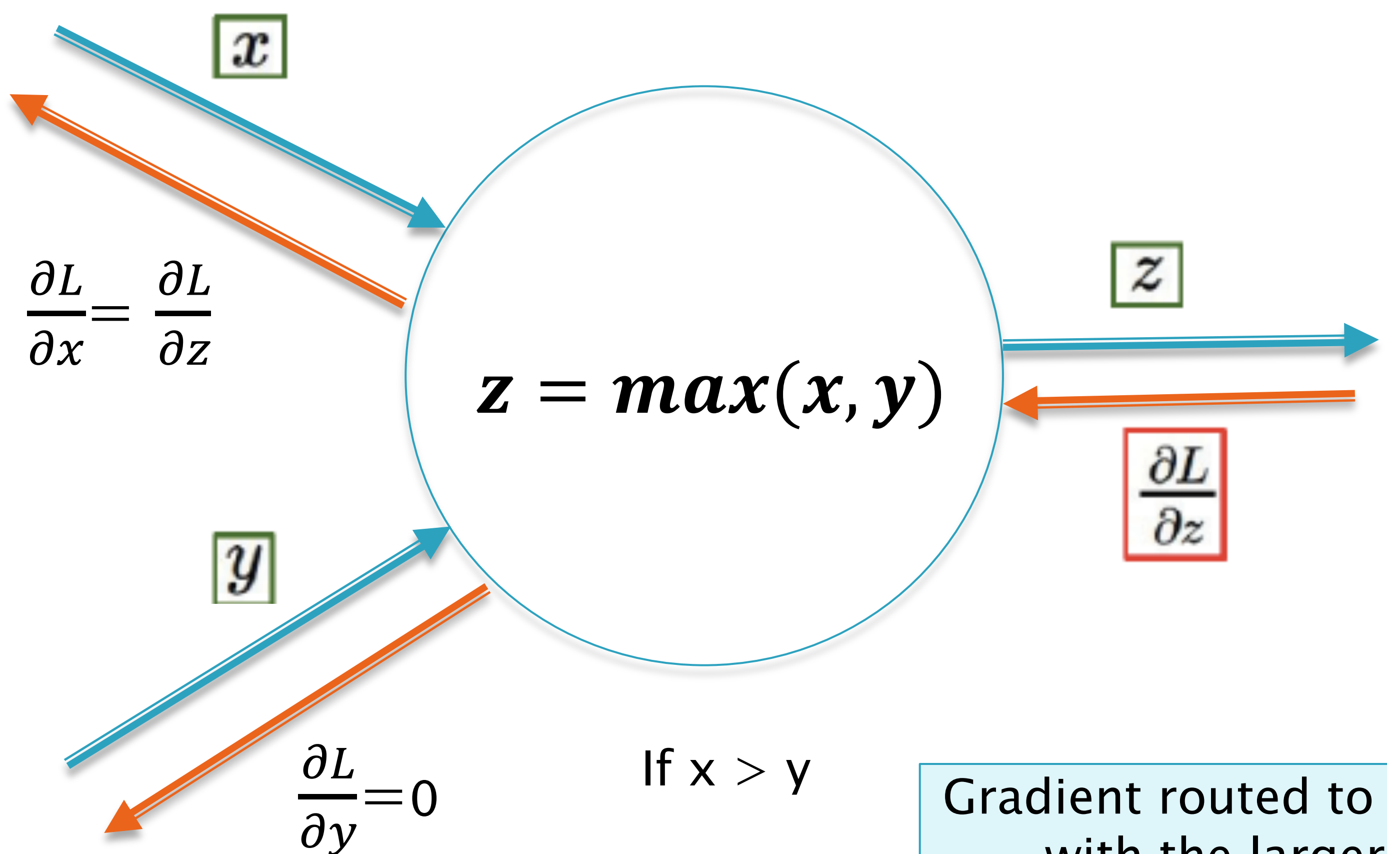
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

Gradient Flow Calculus: Multiplication



$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}$$

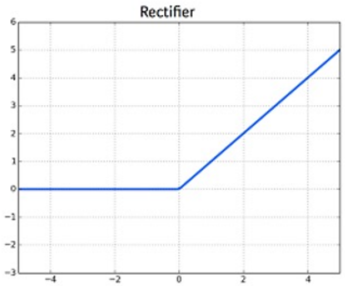
Gradient Flow Calculus: Max Operation

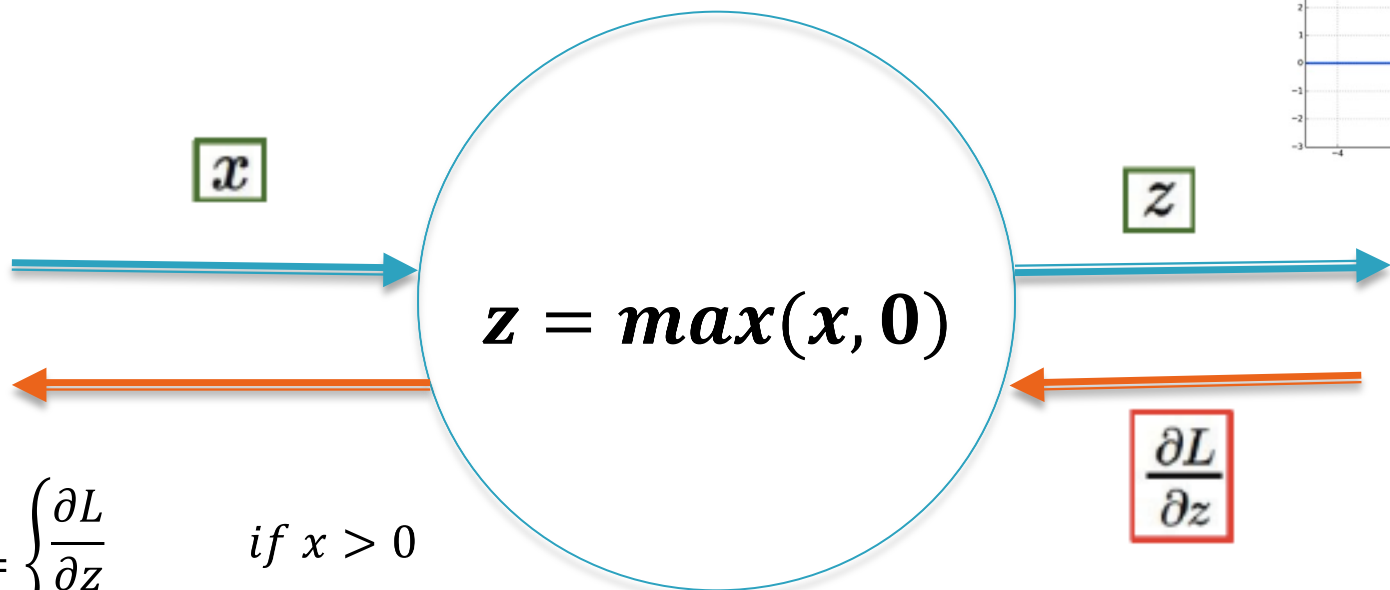


Gradient routed to the branch with the larger input

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

Gradient Flow Calculus: max(x,0)





$$\frac{\partial L}{\partial x} = \begin{cases} \frac{\partial L}{\partial z} & if\ x > 0 \\ 0 & otherwise \end{cases}$$

Gradient passes through if x > 0, otherwise it is suppressed

Input x acts like a switch control

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

Gradient Flow Calculus: Sigmoid Function

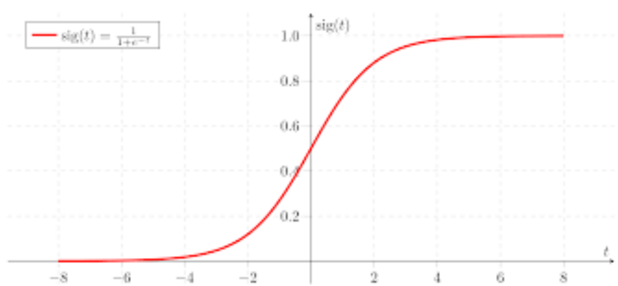
x

z

$$z = \boldsymbol{\sigma}(x) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{exp}(-x)}$$

$$\boxed{\frac{\partial L}{\partial z}}$$

$$\frac{\partial L}{\partial x} = z(1 - z)\frac{\partial L}{\partial z}$$



Gradient flows through only if x lies near the origin, Otherwise it is suppressed.

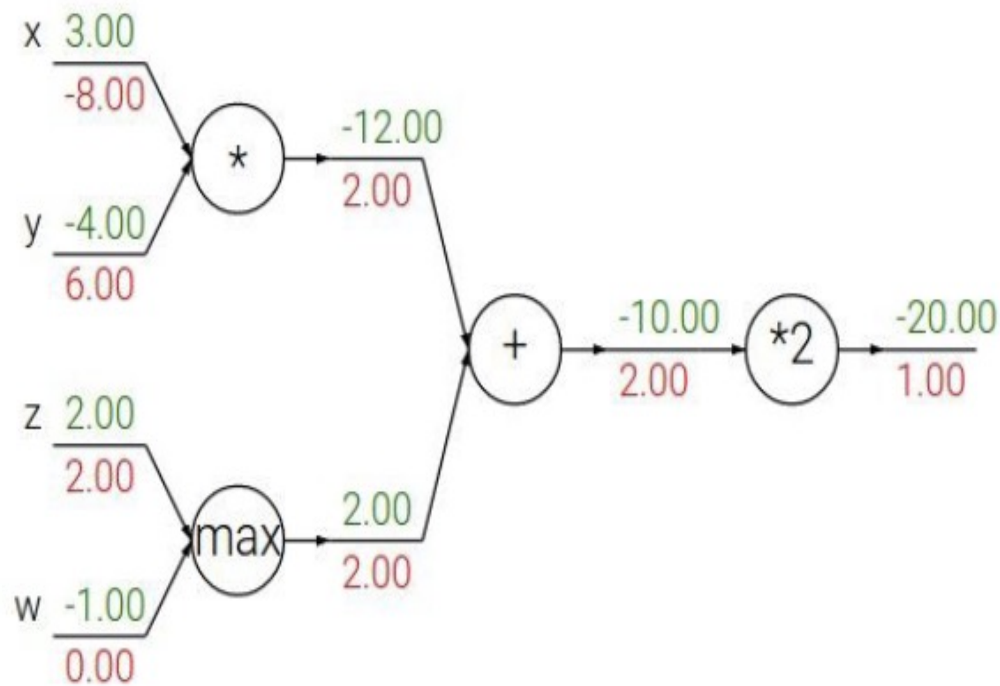
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

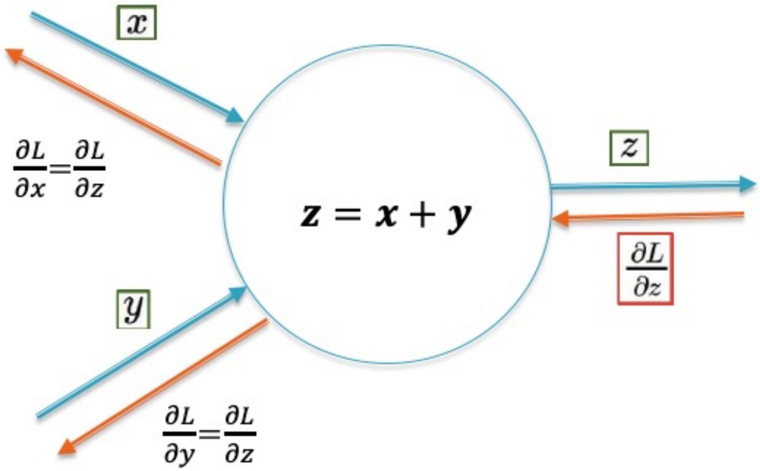
Example: Gradient Flow Rules

add gate: gradient distributor

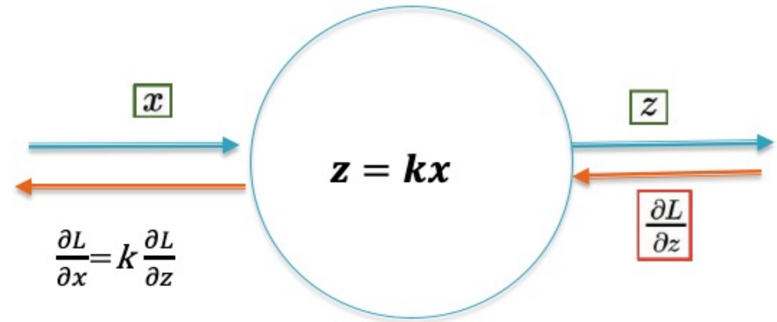
max gate: gradient router

mul gate: gradient switcher

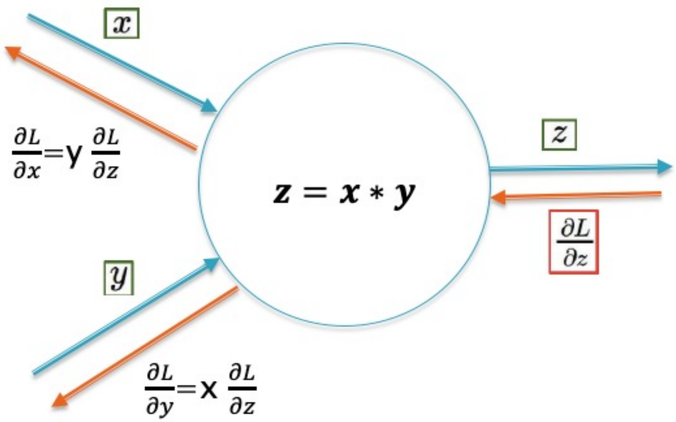




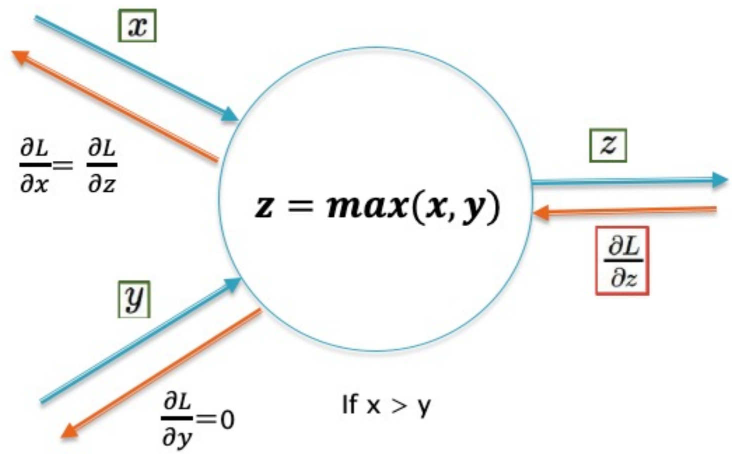
1. Addition Rule



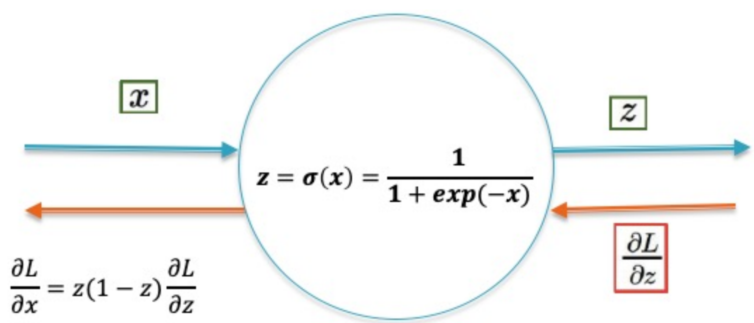
2. Multiplication by a Constant



3. Multiplication



4. Max Operation



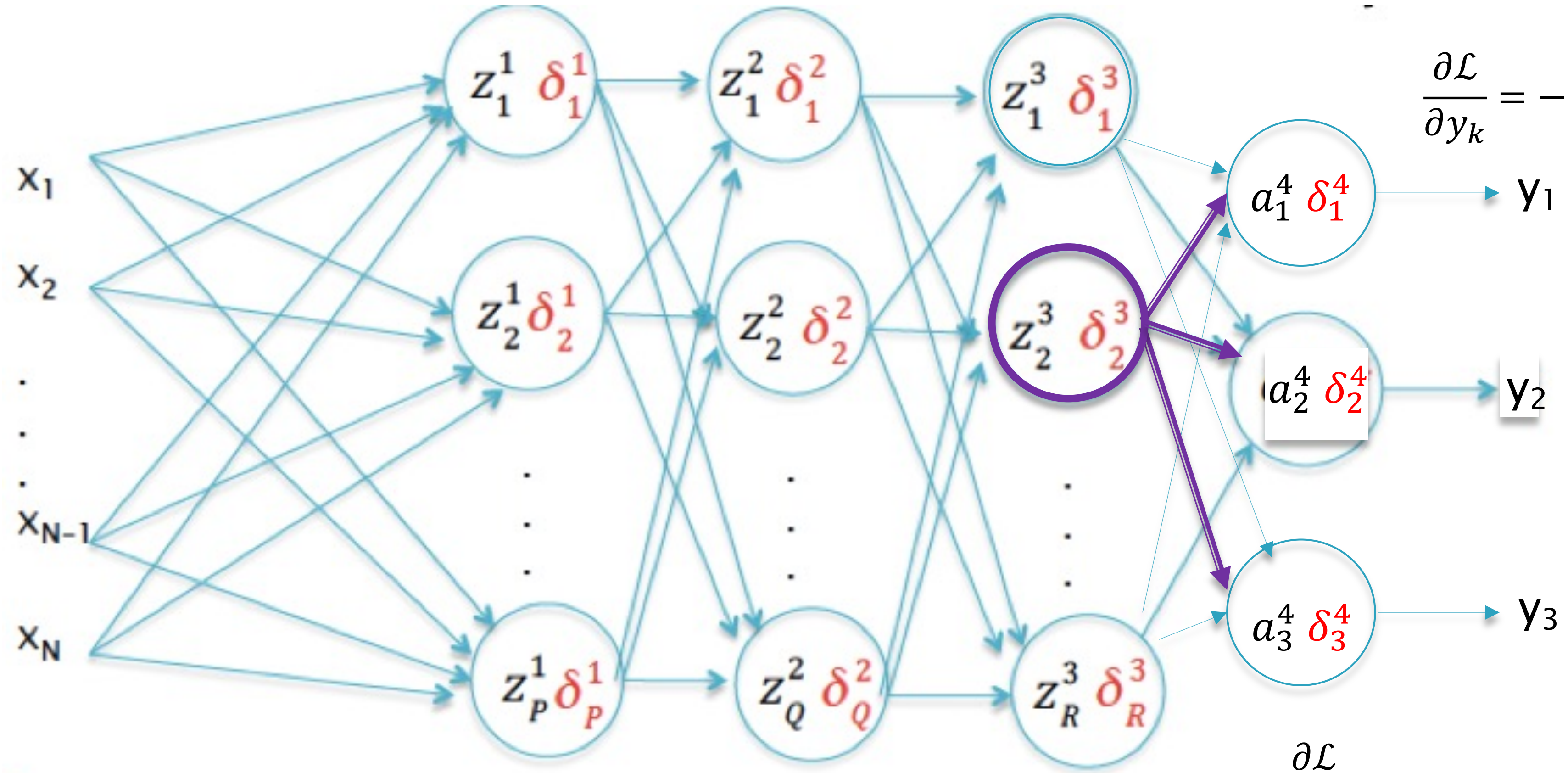
5. Sigmoid Operation

Backprop: Derivation

Backprop – Backward Pass

$$\mathcal{L} = -\sum_{k=1}^{K} t_k \log y_k$$

$$\frac{\partial \mathcal{L}}{\partial y_k} = -\frac{t_k}{y_k}$$



$$\frac{\partial \mathcal{L}}{\partial a_k} = y_k - t_k$$

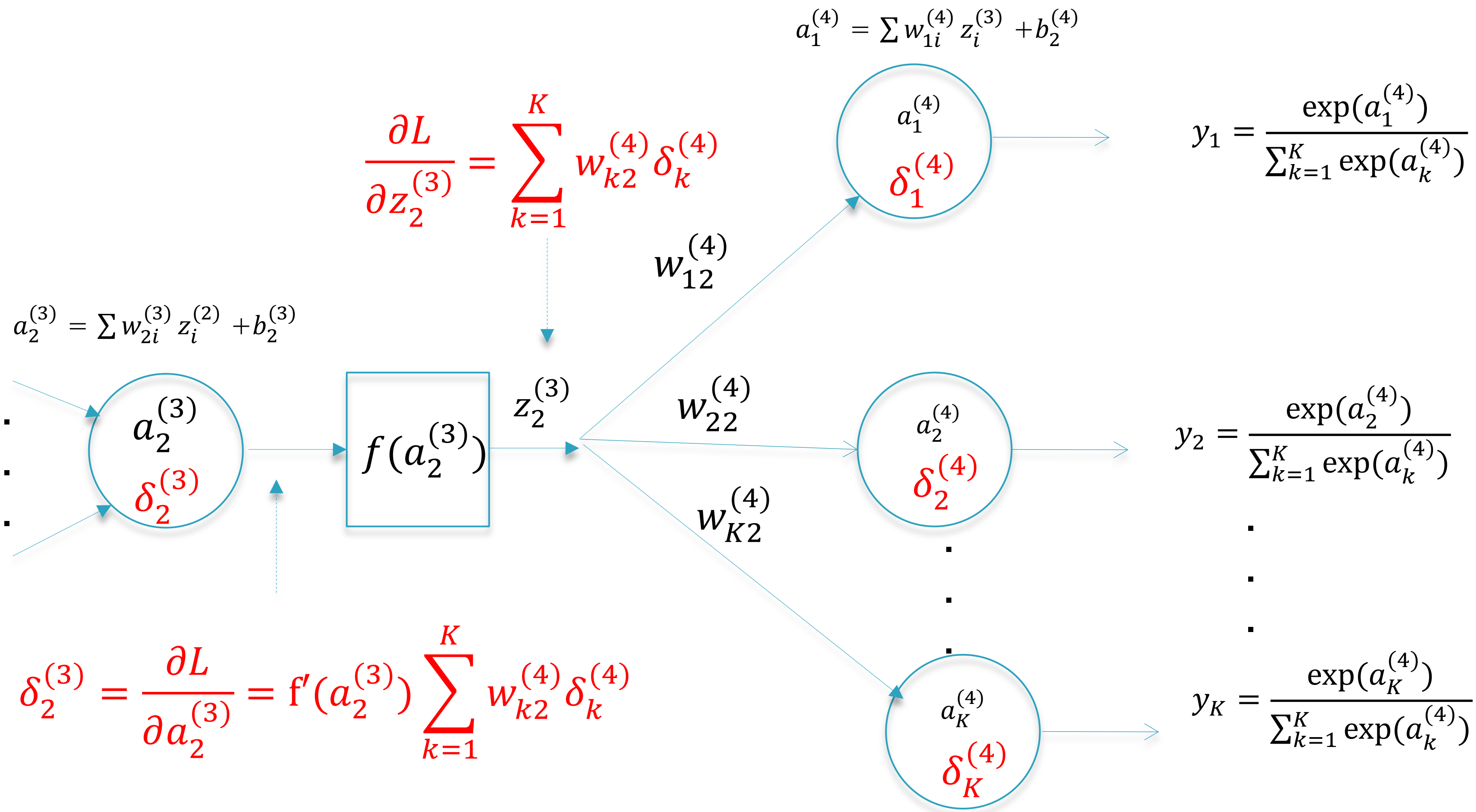
$$a_2^{(3)} = \sum w_{2i}^{(3)} z_i^{(2)} + b_2^{(3)}$$

$$z_2^{(3)} = \mathrm{f}(a_2^{(3)})$$

$$\boxed{\delta_2^3 = \frac{\partial L}{\partial a_2^{(3)}}}$$

$$\delta_k^{(4)} = \frac{\partial L}{\partial a_2^{(4)}} = y_k - t_k$$

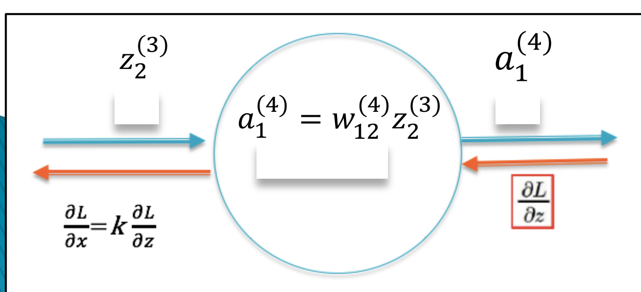
Backprop – Backward Pass



$$\frac{\partial L}{\partial z_2^{(3)}} = \sum_{k=1}^{K} w_{k2}^{(4)} \delta_k^{(4)}$$

$$\delta_2^{(3)} = \frac{\partial L}{\partial a_2^{(3)}} = f'(a_2^{(3)}) \sum_{k=1}^{K} w_{k2}^{(4)} \delta_k^{(4)}$$

$$\delta_k^{(4)} = \frac{\partial L}{\partial a_2^{(4)}} = y_k - t_k$$

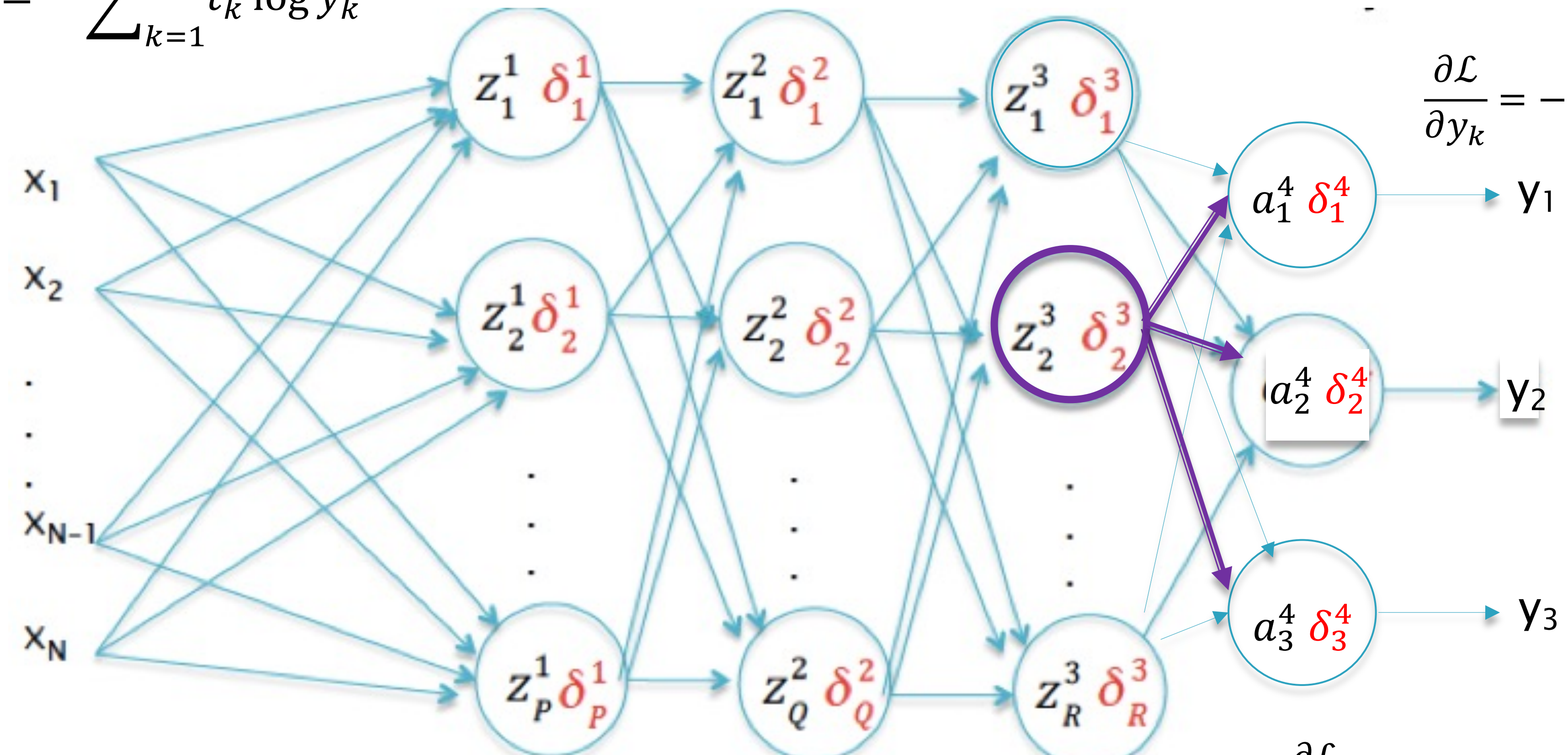


Backprop – Backward Pass

$$\delta = \frac{\partial L}{\partial a}$$

$$\mathcal{L} = -\sum_{k=1}^{K} t_k \log y_k$$

$$\frac{\partial \mathcal{L}}{\partial y_k} = -\frac{t_k}{y_k}$$



$$\frac{\partial \mathcal{L}}{\partial a_k} = y_k - t_k$$

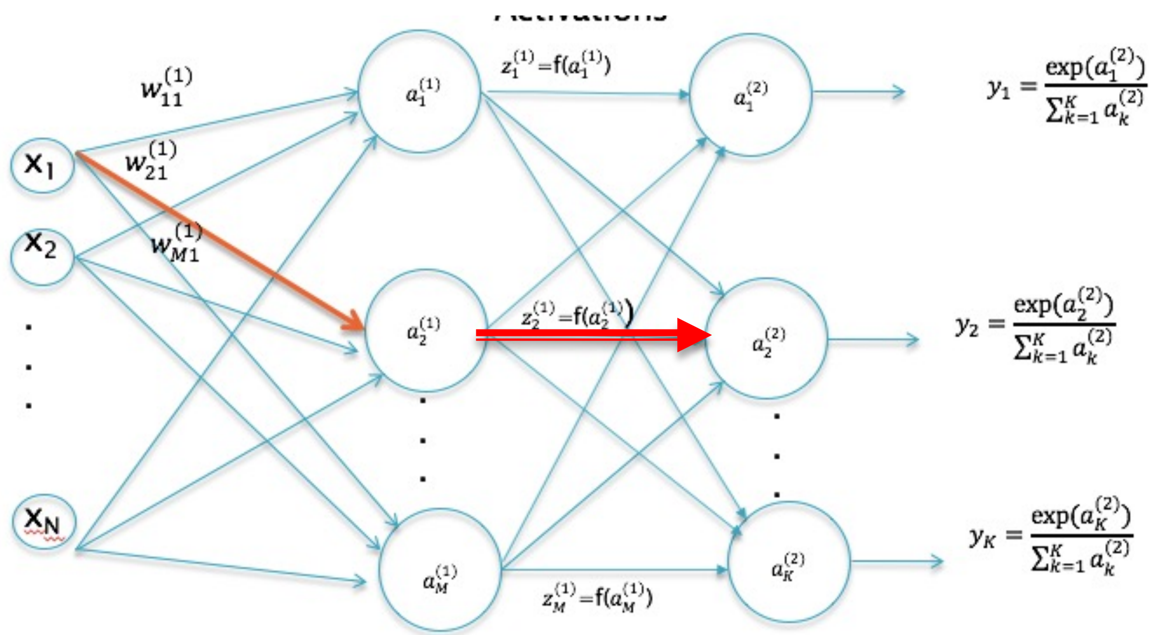
$$\delta_j^{(r)} = f'(a_j^{(r)}) \sum_k w_{kj}^{(r+1)} \delta_k^{(r+1)}$$

$$\Delta^{(r)} = f'\left(A^{(r)}\right) \odot W^T \Delta^{(r+1)}$$

$$\Delta^{(R+1)} = \mathrm{Y-T}$$

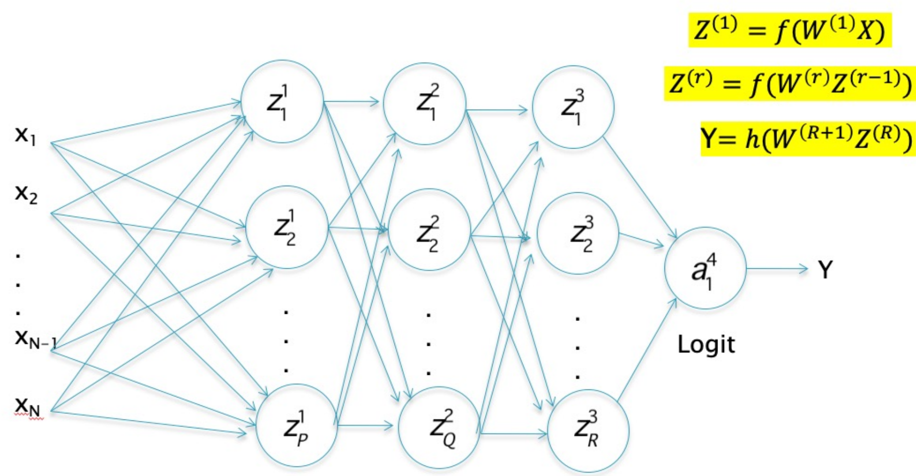
Backprop Product Formula

$$a_2^{(2)} = \sum w_{2j}^{(2)} z_j^{(1)} + b_2^{(2)}$$



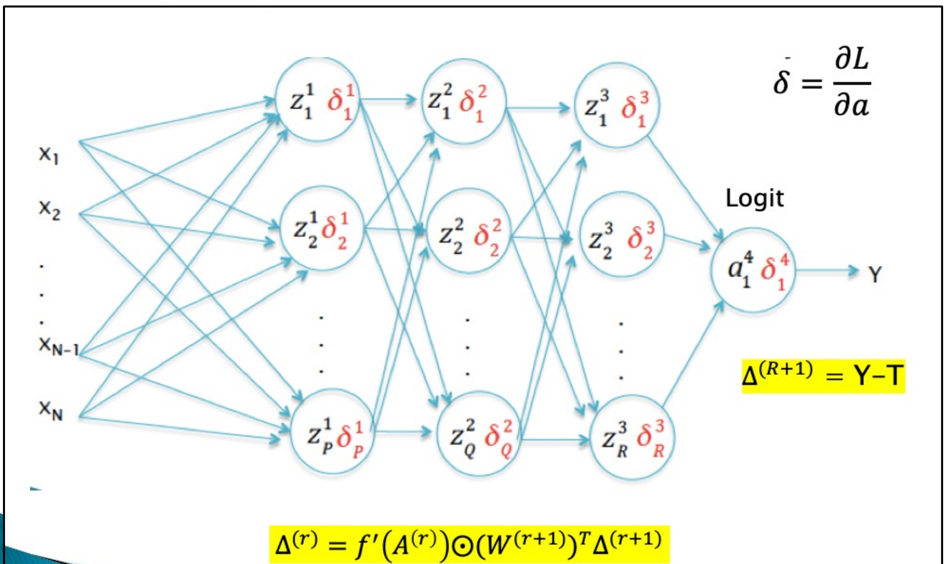
$$\frac{\partial \mathcal{L}}{\partial w_{22}^{(2)}} = \frac{\partial \mathcal{L}}{\partial a_2^{(2)}} \frac{\partial a_2^{(2)}}{\partial w_{22}^{(2)}} = \frac{\partial \mathcal{L}}{\partial a_2^{(2)}} z_2^{(1)} = \delta_2^{(2)} z_2^{(1)}$$

1. Forward Pass

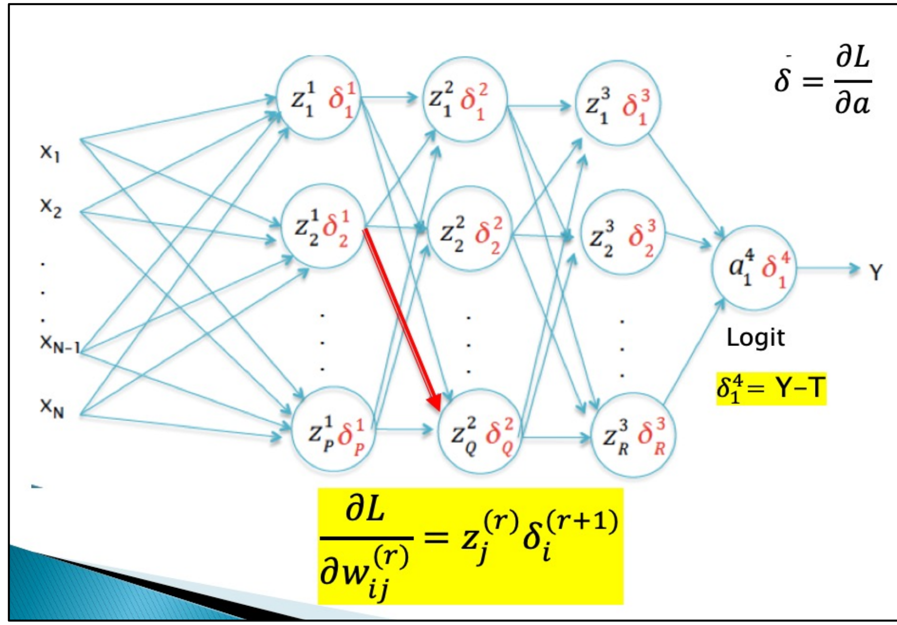


Given an input vector X, compute the activations z for each neuron in the network

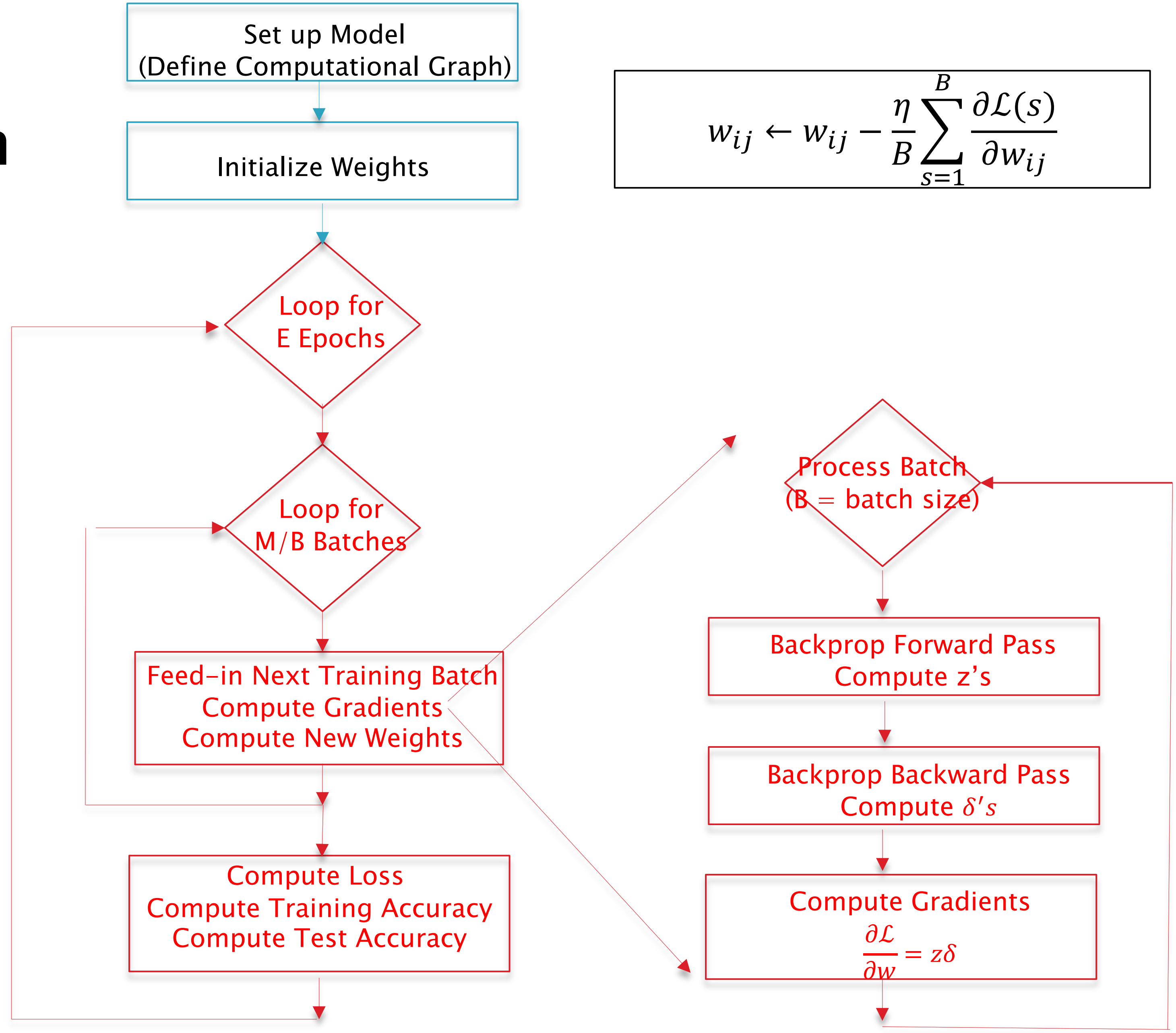
2. Backward Pass



3. Gradient Computation



Training Algorithm Keras



$$w_{ij} \leftarrow w_{ij} - \frac{\eta}{B} \sum_{s=1}^{B} \frac{\partial \mathcal{L}(s)}{\partial w_{ij}}$$

```
history = network.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)
```

Verifying Backprop

This is done using Numerical Differentiation

$$\frac{\partial L}{\partial w_i} \approx \frac{L(w_1, \cdots, w_i + \varepsilon, \cdots, w_n) - L(w_1, \cdots, w_i - \varepsilon, \cdots, w_n)}{2\varepsilon}$$

For small values of epsilon, say $\varepsilon = 10^{-4}$

Some Important Dates

- Mid-Term Exam: Nov 9, 7:35-9:10PM
  - Syllabus: Lectures 1 to 14

- Project Proposal Due: Nov 2
  - Once you have settled on a Project Idea, talk to me (before starting work)!

- Project Presentations (Dec 7):
  - 15 minutes per presentation + 2 minutes Q&A

Data Repositories

- Popular Open Data Repositories
  - Kaggle: www.kaggle.com
  - Amazon's AWS datasets: aws.amazon.com/fr/datasets/
  - UC Irvine ML Repository: archive.ics.uci.edu/ml/
- Meta Portals (list of open data repositories)
  - dataportals.org
  - opendatamonitor.eu
  - quandl.com
- Other pages
  - Wikipedia's List of ML Datasets: https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research
  - Quora.com question: goo.gl.zDR78y/
  - Datasets subreddit: www.reddit.com/r/datasets
- Your own:
  - With Transfer Learning you can reuse parts of existing trained models, and train your datasets using smaller samples (1000 vs a million)

Rubric for Project Evaluation

- Quality and definition of the project idea: scored as 1,2,3
- Execution of the idea, i.e., coding, results. 1,2,3
- Quality of presentation and how much the others learned from the presentation: 1,2,3

Some Example Projects

- Marathi to English Translator
- Dog Breed Classifier
- Fashion Item Classifier
- X-Ray Image Classifier
- Quora Classifier
- Time Series Analysis using RNNs
- Question Answering System
- VIX Prediction

Further Reading

- Chapters 8: TrainingNNsBackprop
  https://srdas.github.io/DLBook2/