Deep Learning and Applications

Today's Outline

- Course Logistics
- Introduction to the Course
- Getting Started with Neural Nets
  - Classification
  - Backpropagation
  - Feedforward Neural Nets

Course Topics

- We will cover several tools under the umbrella of
  - Deep Learning
  - Online and Reinforcement Learning

Introduction to the Course

20000 Ft View

- You need a critical understanding of the domain to be successful in shipping solutions
- Before venturing into a complex technique, try a shallow/easy technique

A Business Analyst's Toolkit

- Techniques
  - Prediction
    - Decision Trees
    - Linear classifiers and logistic regression
    - Naïve Bayes classifier
    - SVMs
    - Neural networks (and deep learning)
    - Online/reinforcement learning
  - Exploration
    - Clustering
    - Market basket analysis

Data Variety

- Structured data
  - Examples:
    - Medical/healthcare data, advertising data
  - Have ordinal, integer, binary or categorical fields
  - Deep learning allows embedding of categorical features
- Unstructured data
  - Examples:
    - Images (tensor, i.e., typically a 3 dimensional array) and videos (a sequence of images), text strings/documents
  - Deep learning reduces feature engineering effort here

Complex Decisions

- Decisions
  - Examples:
    - which articles to show, how to price products
  - May use many predictions
  - May need to be taken repeatedly for different contexts
  - May have longer term goals
  - Online and reinforcement learning methods address this 'learning on the go' problem

Two Themes of the Course

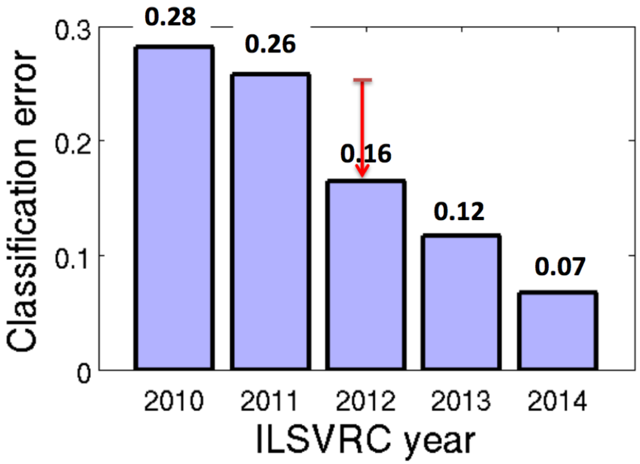
- Data Variety
  - Images and Videos/Audio
  - Text and Language
- Complex Decisions
  - Sequential Decision Making

Techniques covered in the Course

- To address data variety and complex decision problems, we will look at:
  - Deep Learning
  - Online and Reinforcement Learning + Deep Learning

Deep Learning

- One example (in vision) of its success is at the ILSVRC[1]
- ImageNet dataset has 22000 categories across 14 million images
- ILSVRC Task 1 was a classification challenge
  - Given 1000 categories and1.5 million images, predict 5 categories for a test image



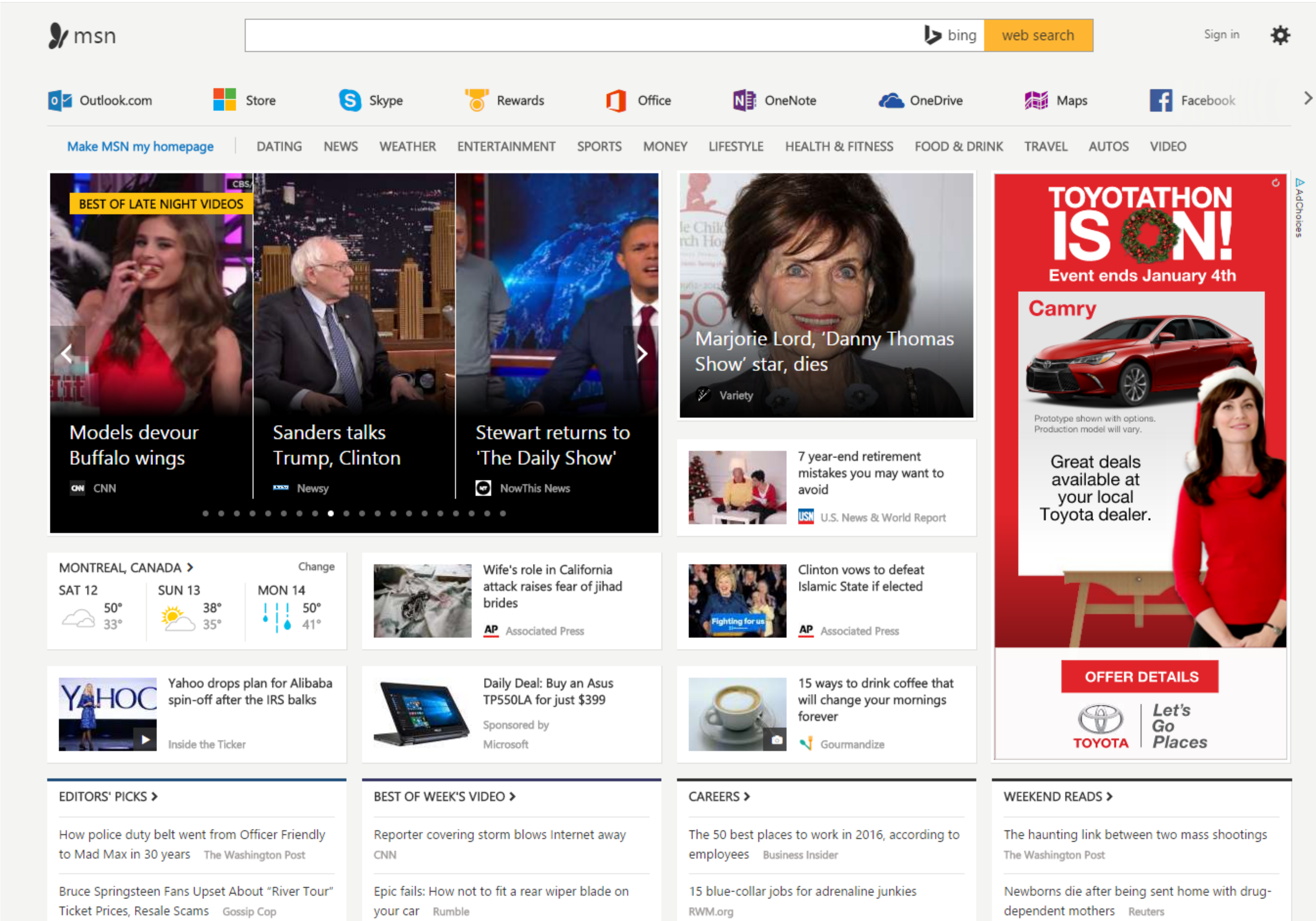
[1]ImageNet Large Scale Visual Recognition Challenge
[2]Figure: Russakovsky et al. arxiv:1409.0575

Deep Learning

- Neural nets are not new (1960s). Applied to handwritten digit recognition back in 1998
- Were not mainstream till around 2010/2012*
  - What changed? Access to GPUs and Data
- Caveat:
  - Deep learning achieves good performance on some tasks
  - Typically has not worked well beyond classification…
  - There is a lot of scope for improvement, engineering, system building, model building

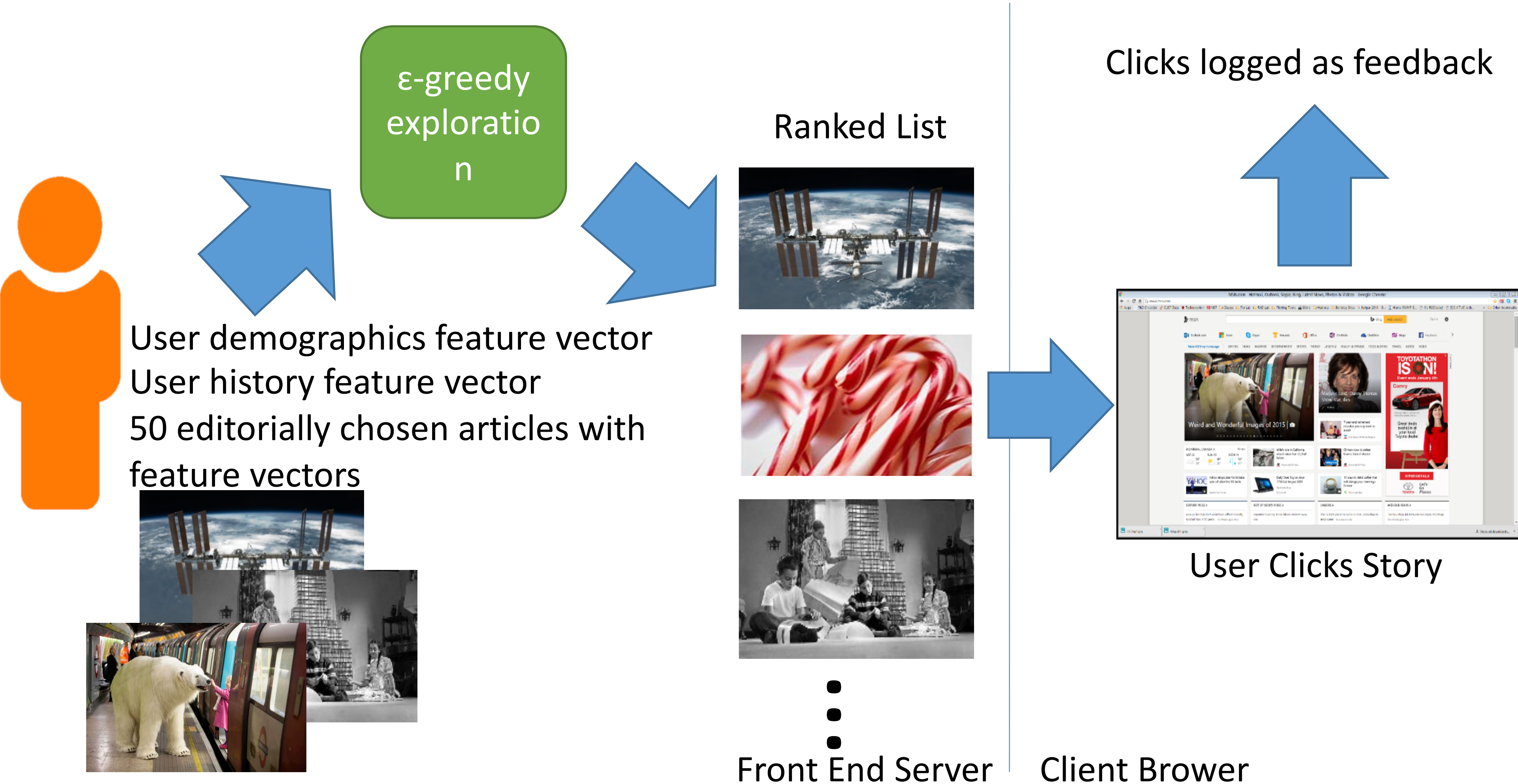
*Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition, Dahl et al. 2010
Imagenet classification with deep convolutional neural networks, Krizhevsky et al. 2012

Online/Reinforcement Learning



[1] Reference: Alekh Agarwal et al., http://arxiv.org/abs/1606.03966

Online/Reinforcement Learning

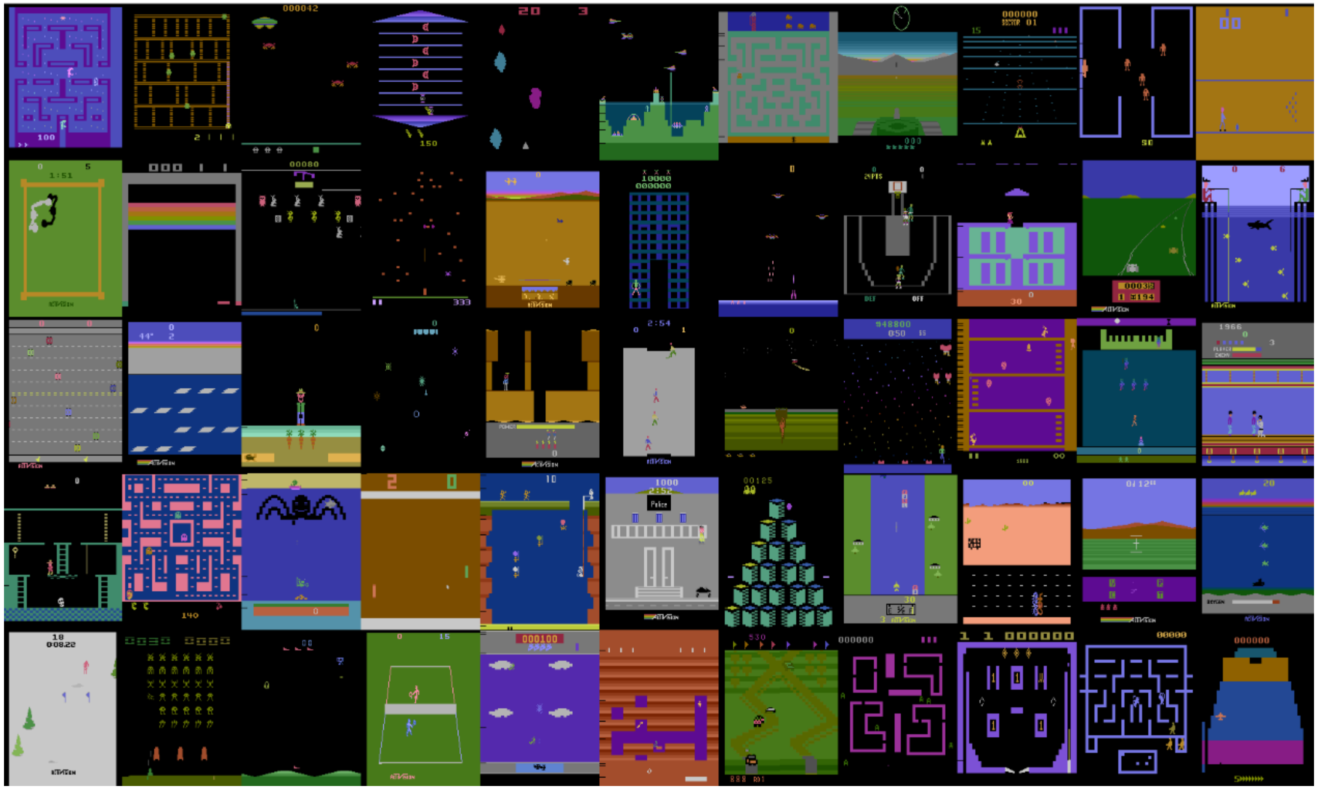


[1]Reference: Alekh Agarwal et al., http://arxiv.org/abs/1606.03966

Online/Reinforcement Learning

[1]Reference: DeepMind, March 2016

Online/Reinforcement Learning



[1]Figure: Defazio Graepel, Atari Learning Environment

Caveat with Any Technique

- Measurable metrics of business success take priority over technical success metrics
- Need to ask:
  - Does a Y% increase in classification accuracy help in X% increase in sales?
  - Does a Z% increase in classification accuracy due to using a deep learning solution help the bottom-line?
  - What is the technical debt incurred? Who will maintain?

Questions?

Today's Outline

- Course Logistics
- Introduction to the Course
- Getting Started with Neural Nets
  - Classification
  - Backpropagation

Classification

- Classification
  - Data
  - Model
  - Loss
  - Optimization

Classification

- To design the classifier, we need
  - Training data
  - Model specification for the classifier
  - Loss function to define the best model
  - Optimization to get to the best model

Data (I)

- Lets pick a domain: Vision
- What is an image?
  - A bunch of numbers between 0 to 255
  - A 3 dimensional array
  - The same object can look different based on
    - Location of the camera
    - Location of the light source
    - Rigidity of the object
    - Occluding objects
    - Background
    - Variation across objects of the same category

Data (II)

- Say we have N training examples $(x_i, y_i), i = 1, \dots, N$
  - x_i is the feature vector for the i^{th} example
  - y_i is the label for the i^{th} example

- Before deep learning
  - Carefully designed features
    - Histogram of colors
    - Histogram of Oriented Gradients (HOG)
    - Scale Invariant Feature Transform (SIFT)
    - Various types of filters

- With deep learning
  - Almost no feature engineering (for this type of data)

Model (I)

- Parametric vs non-parametric
- Example:
  - Logistic classifier is parametric
  - K-Nearest Neighbor is a non-parametric classifier
- We will focus on parametric models
- A fixed set of parameters and hyper-parameters determine a model completely

Model (II)

- Pick a concrete parametric model $f(x, W, b)$
  - x is the input ($d \times 1$ dimensional)
    - Vectorize the image or get features
  - W is a parameter ($p \times d$ dimensional)
  - b is also a parameter ($p \times 1$ dimensional)

- Let $f(x, W, b) = \ Wx + b$
  - This is a linear model
  - We will change this later
  - The output of the linear model is a vector of scores

Model (III)

- Given a model (i.e., a fixed W, b pair) our classifier can be
  - Pick the index with the highest 'score'
    - $\hat{l} = \text{argmax}_{\{j=1,\dots,p\}} f(x, W, b)$
  - Pick the index with the highest 'probability'
    - Need a map/function from scores to probabilities
- We want to use the best model. How?
  - Define best: Loss function
  - Find the best: Optimization

Loss functions (I)

- Let the j^{th} coordinate of $f(x, W, b)$ be s_j
- Loss L_{data} is defined over the training data
- Is chosen to be decomposable over N terms, one per example
  - $L_{data} = \sum_{i=1}^{N} L_i$

Loss functions (I)

- Let the j^{th} coordinate of $f(x, W, b)$ be s_j
- Loss L_{data} is defined over the training data
- Is chosen to be decomposable over N terms, one per example
  - $L_{data} = \sum_{i=1}^{N} L_i$
- Logistic loss (Cross-entropy or softmax) for example i
  - $L_i = -\log P(Y = y_i | X = x_i)$ where
    - $P(Y = j | X = x_i) = \dfrac{e^{s_j}}{\sum_k e^{s_k}}$
- SVM loss (2 class, W is a row vector) for example i
  - $L_i = \max(0, 1 - y_i s_{y_i})$

Loss functions (II)

- Need for regularization
  - Unique model
  - Desired model
  - Control overfitting

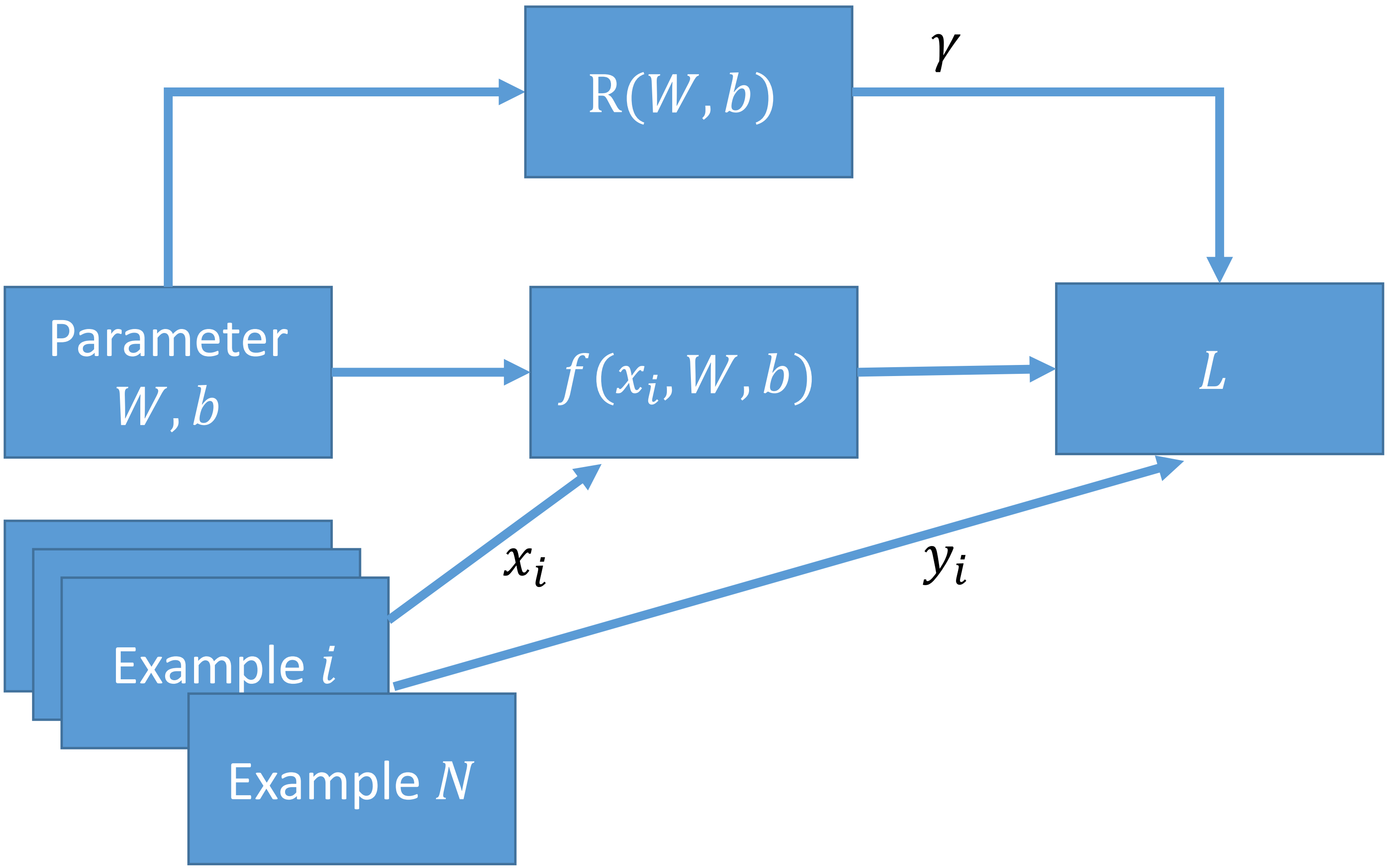
- Final loss $L = L_{data} + \lambda R(W, b)$
- $R(W, b)$ can be just a function of W or b or both

Loss functions (III)

- L2 regularization: $||W||_2^2 = \sum_i \sum_j W_{ij}^2$
- L1: $||W||_1 = \sum_i \sum_j |W_{ij}|$
- Elastic net: $\alpha\left||W|\right|_1 + (1-\alpha)\left||W|\right|_2^2$

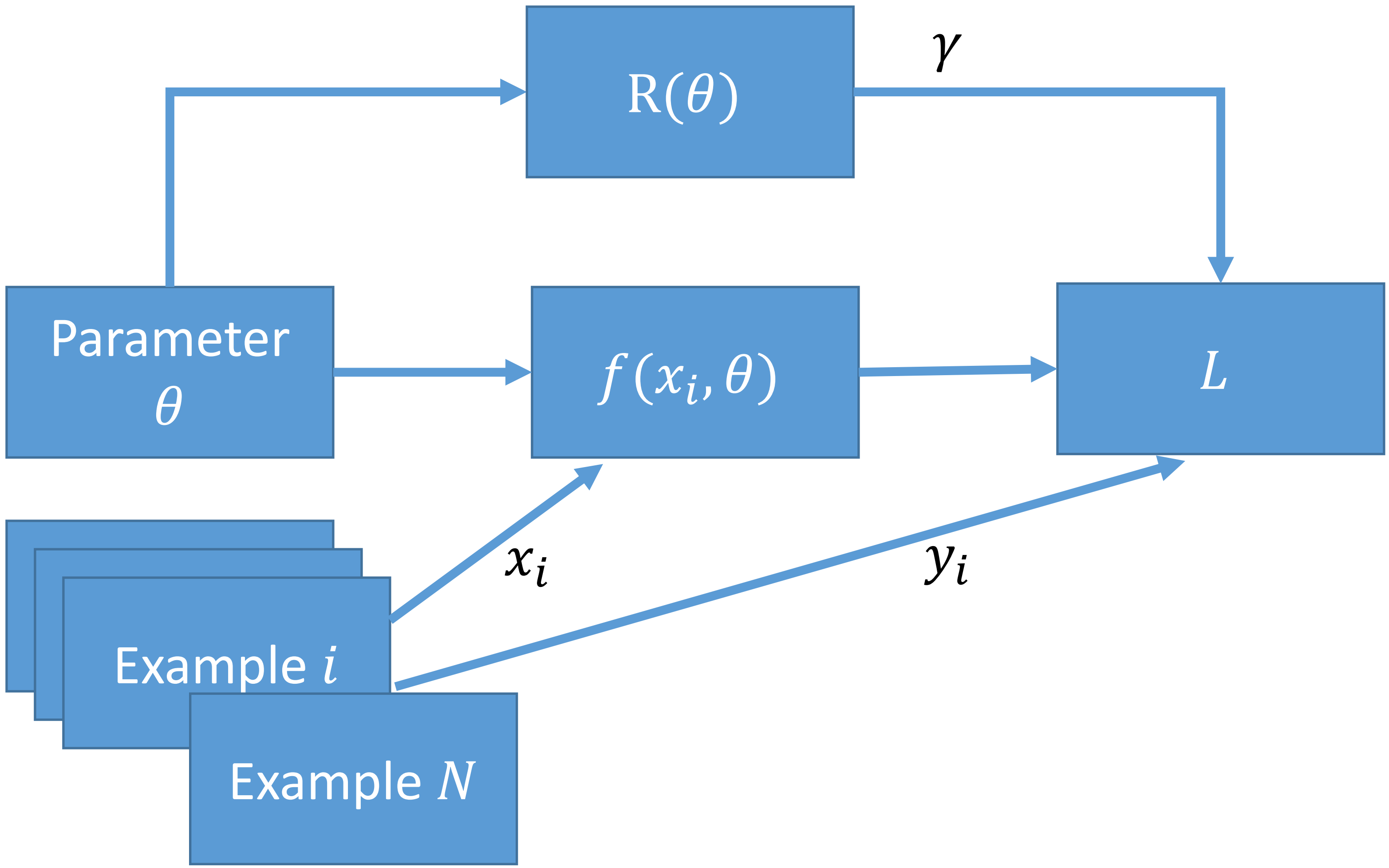
- Regularization may not always be and explicit function of the parameters
  - We will see dropout later

Optimization (I)



Need to find parameters W, b and hyper-parameter γ

Optimization (I)



Need to find parameters θ and hyper-parameter γ

Optimization (II)

- Many ways to optimize differentiable models
- We will focus on first order methods
  - Key ingredient: Gradient

- Gradient is the vector of partial derivatives of a function

- Can be computed
  - Numerically: $\lim_{h \to 0} \frac{f(z+h)-f(z)}{h}$
  - Analytically: Calculus and chain rule

Optimization (III)

- Intuition
  - Start with a model (i.e., W_0, b_0)
  - Evaluate L for this model on the training data
  - Change W_0, b_0 to W_1, b_1 such that the new L is smaller
  - Repeat

- This intuition is the essence of Gradient Descent methods
  - Gradient of L with respect to the parameters is used to change W_0, b_0 to W_1, b_1

Optimization (IV)

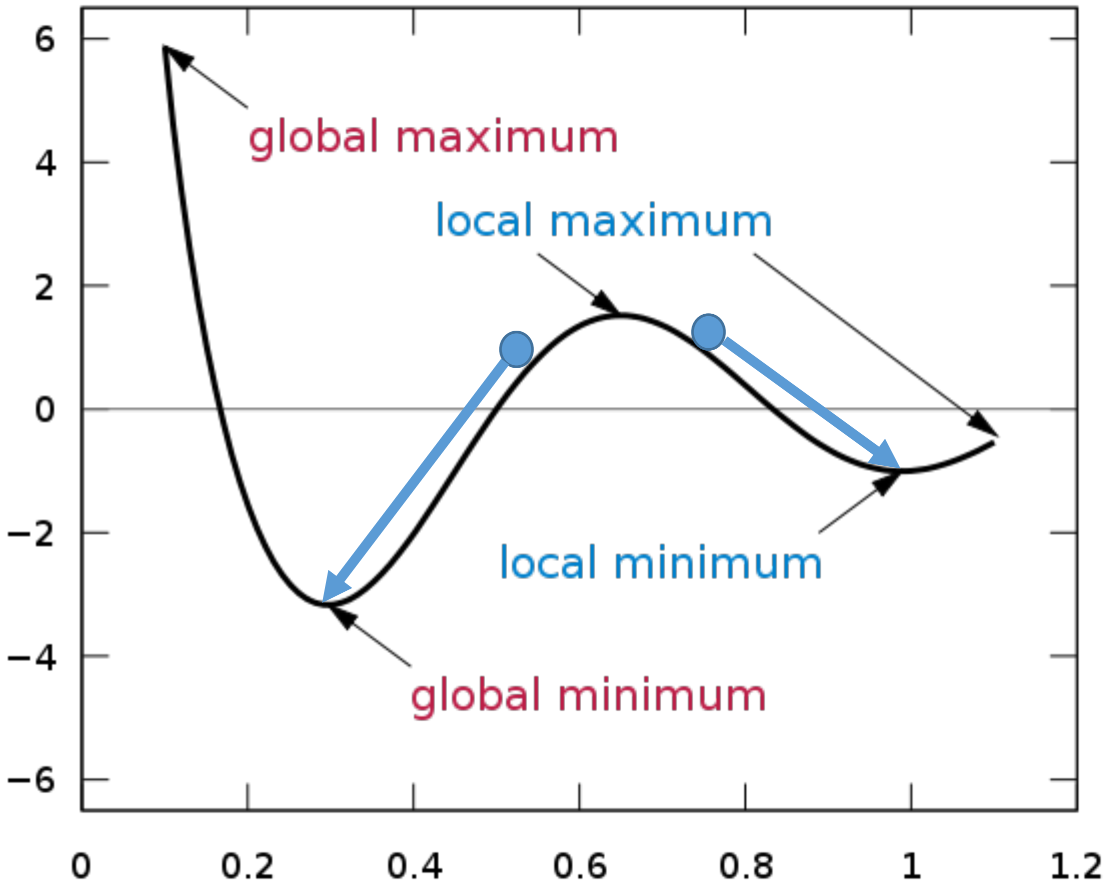
- Example method: Batched Gradient Descent
- Get a sample of training data
  - Example: AlexNet[1] used 256 examples as one batch
- Get gradient of L with respect to parameters W, b
- Update
  - $W_{k+1} \leftarrow W_k \; - \alpha \nabla_W L$
  - $b_{k+1} \leftarrow b_k \; - \beta \nabla_b L$

- Step sizes (learning rates) α, β need careful choice

[1]Krizhevsky et al. NIPS Deep Learning Workshop 2012

Optimization (IV): Gradient Descent

Gradient descent can only reach local optima

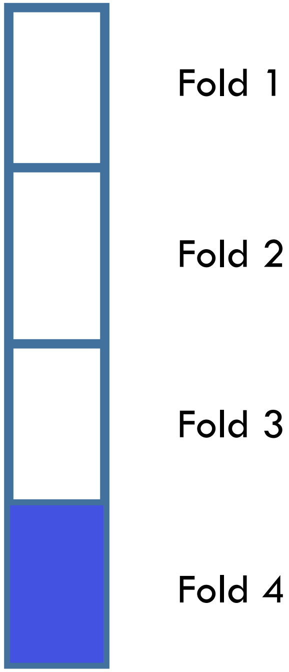


[1]By I, KSmrq, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2276449

Optimization (V)

- Tuning the hyper-parameter(s)
  - Break dataset into two parts: test and train
  - Remove test data access while you are tuning the parameters of your model
  - With training data, do cross validation to tune parameters and hyper-parameters

Essentially cycle through the choice of validation fold
Optimize parameters over the remaining folds



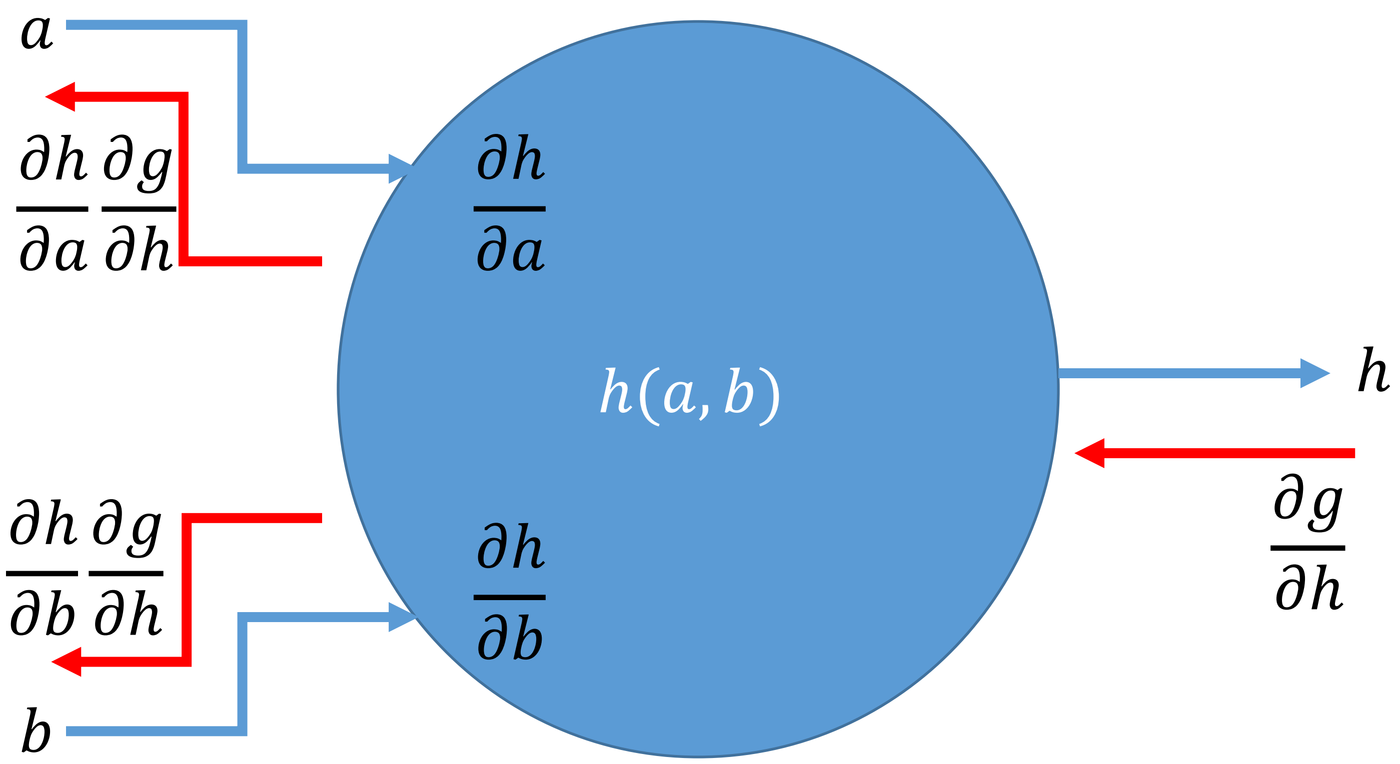
Questions?

Today's Outline

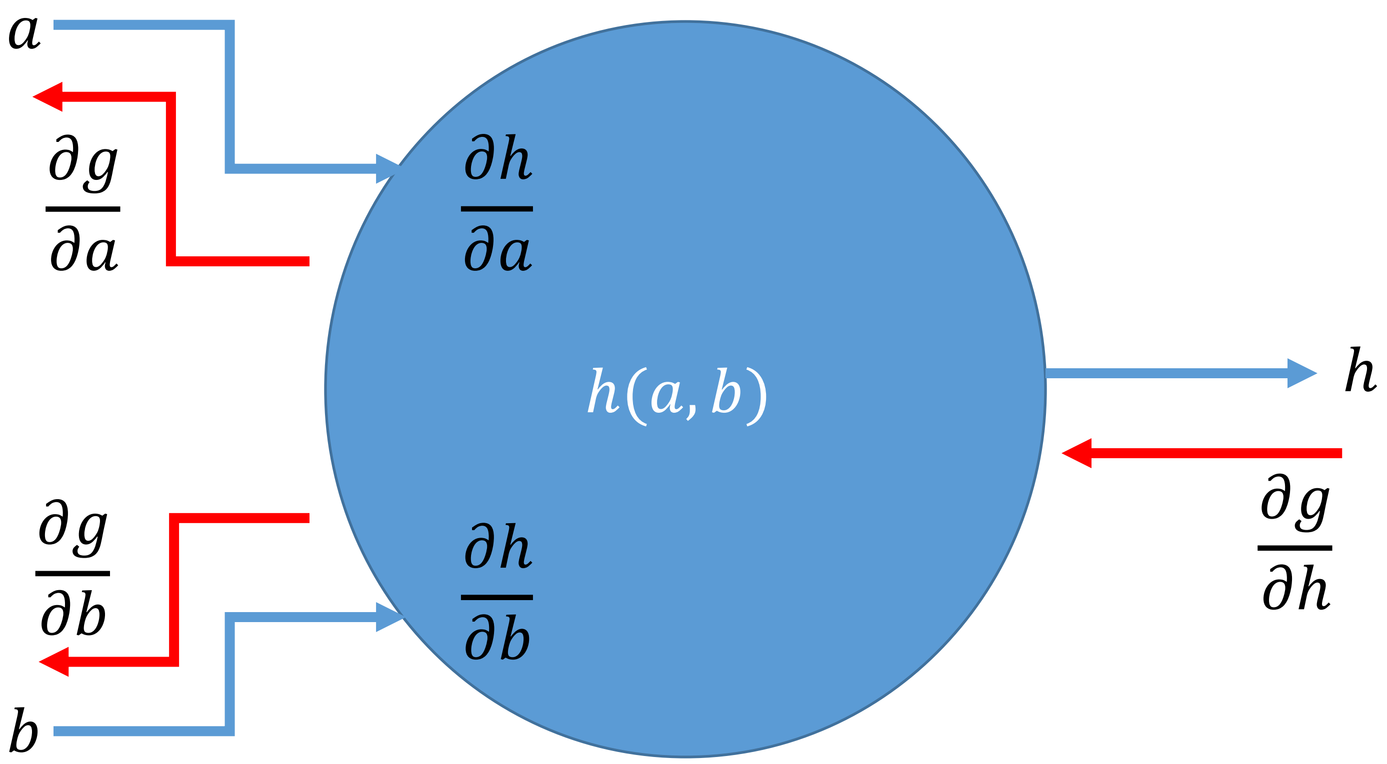
- Course Logistics
- Introduction to the Course
- Getting Started with Neural Nets
    - Classification
    - Backpropagation

Backpropagation

- An efficient way to get the gradient needed for optimization

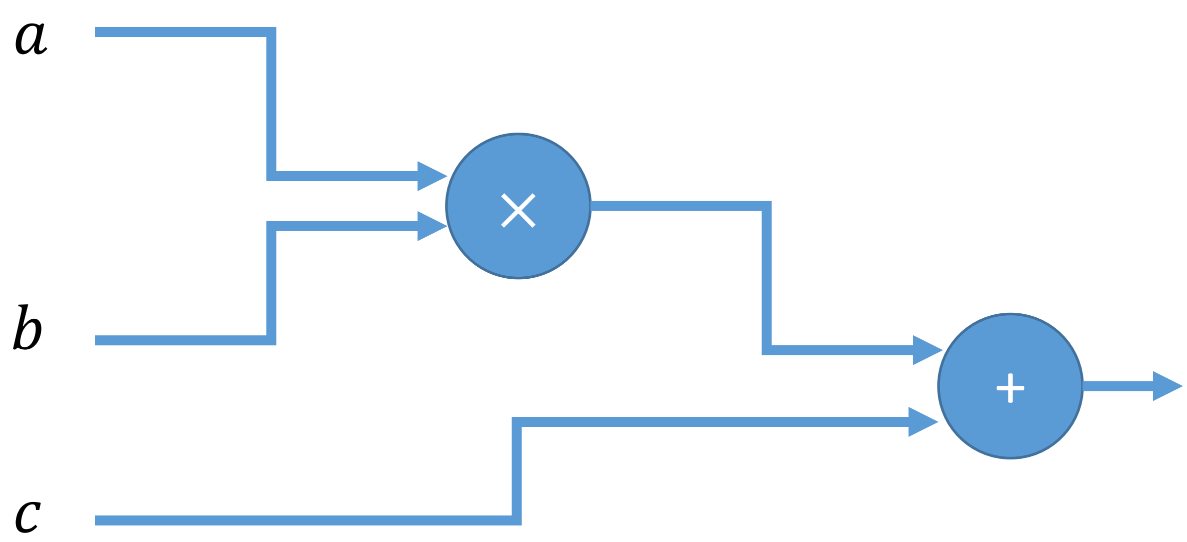


Backpropagation



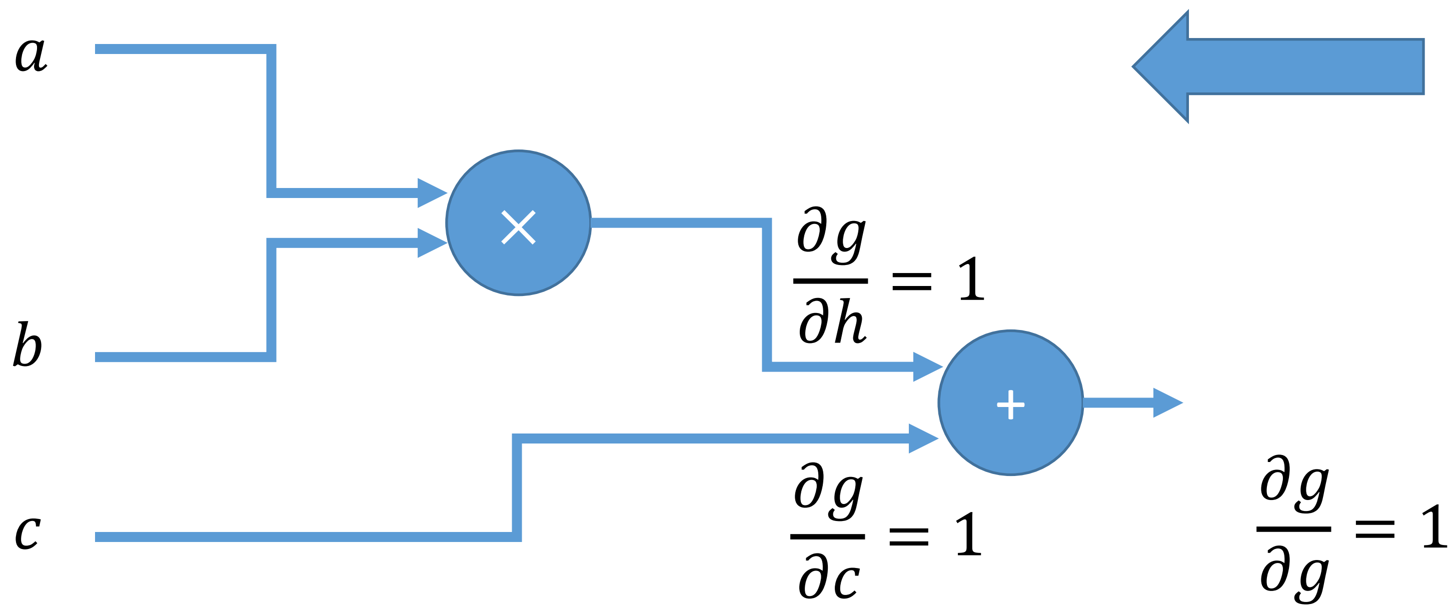
Notion of a Computational Graph

- Consider a function $g(a, b, c) = a * b + c$
- Draw a graph



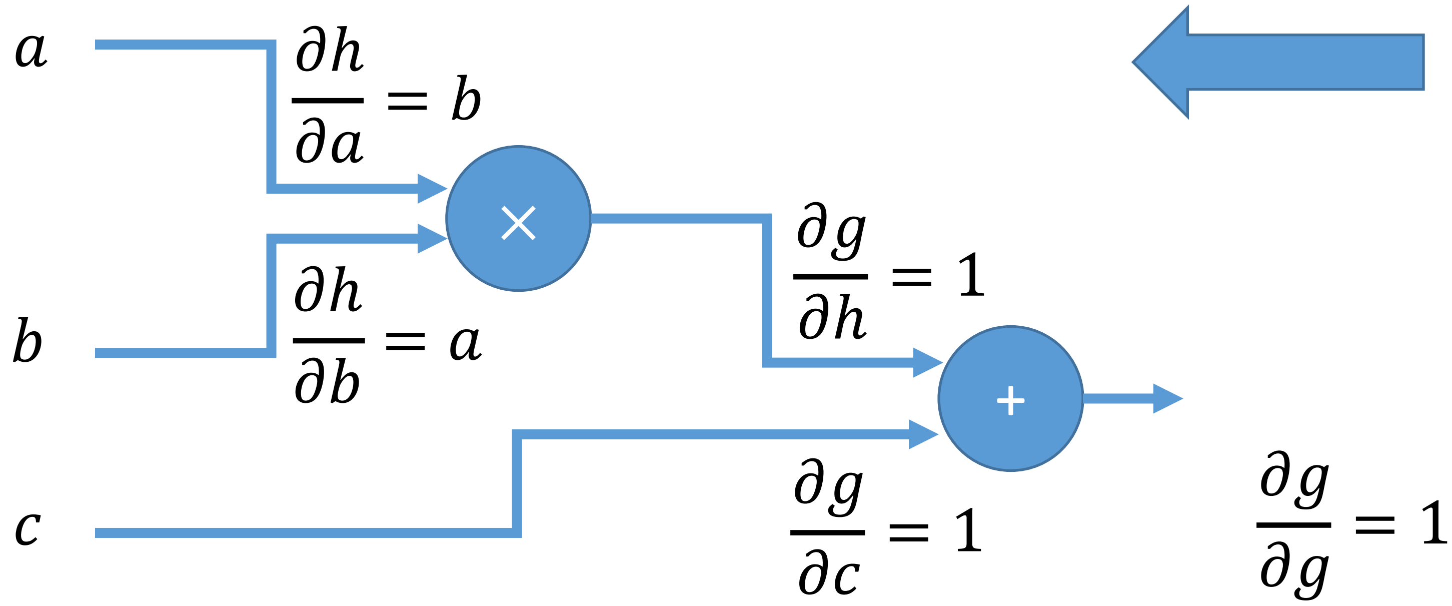
Backprop Example 1

- The circles represent compute nodes
- Let $h = a * b$. Then $g = h + c$



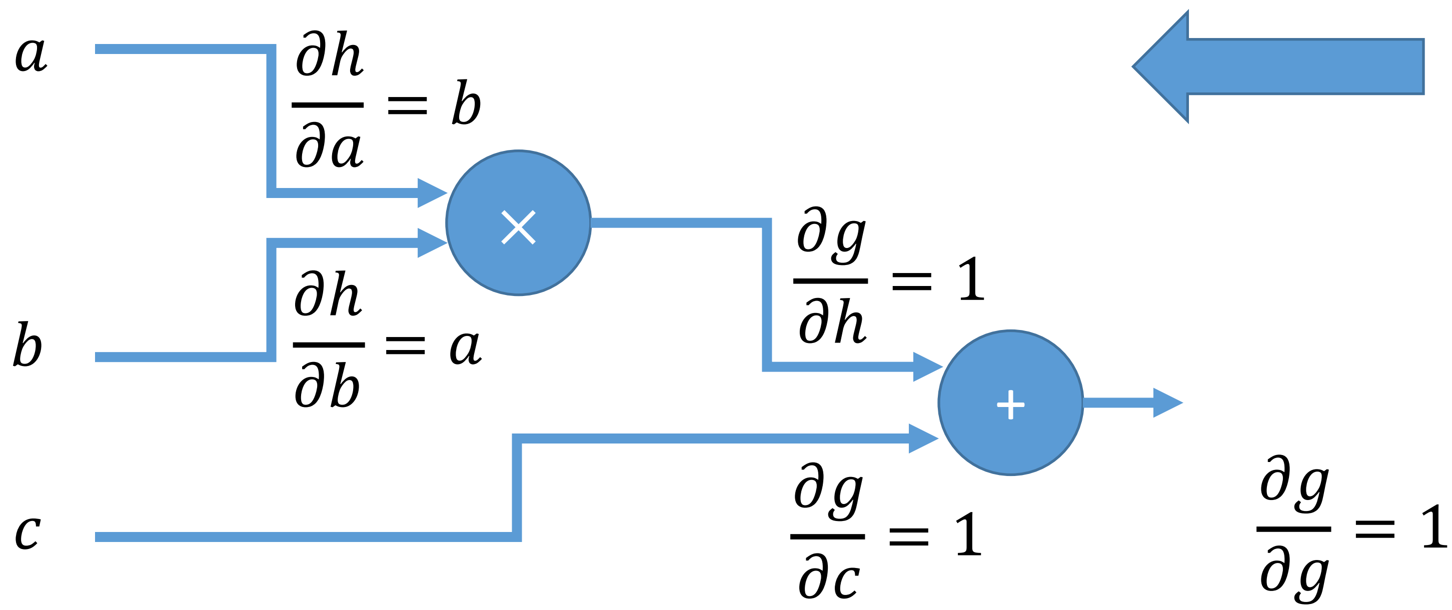
Backprop Example 1

- The circles represent compute nodes
- Let $h = a * b$. Then $g = h + c$



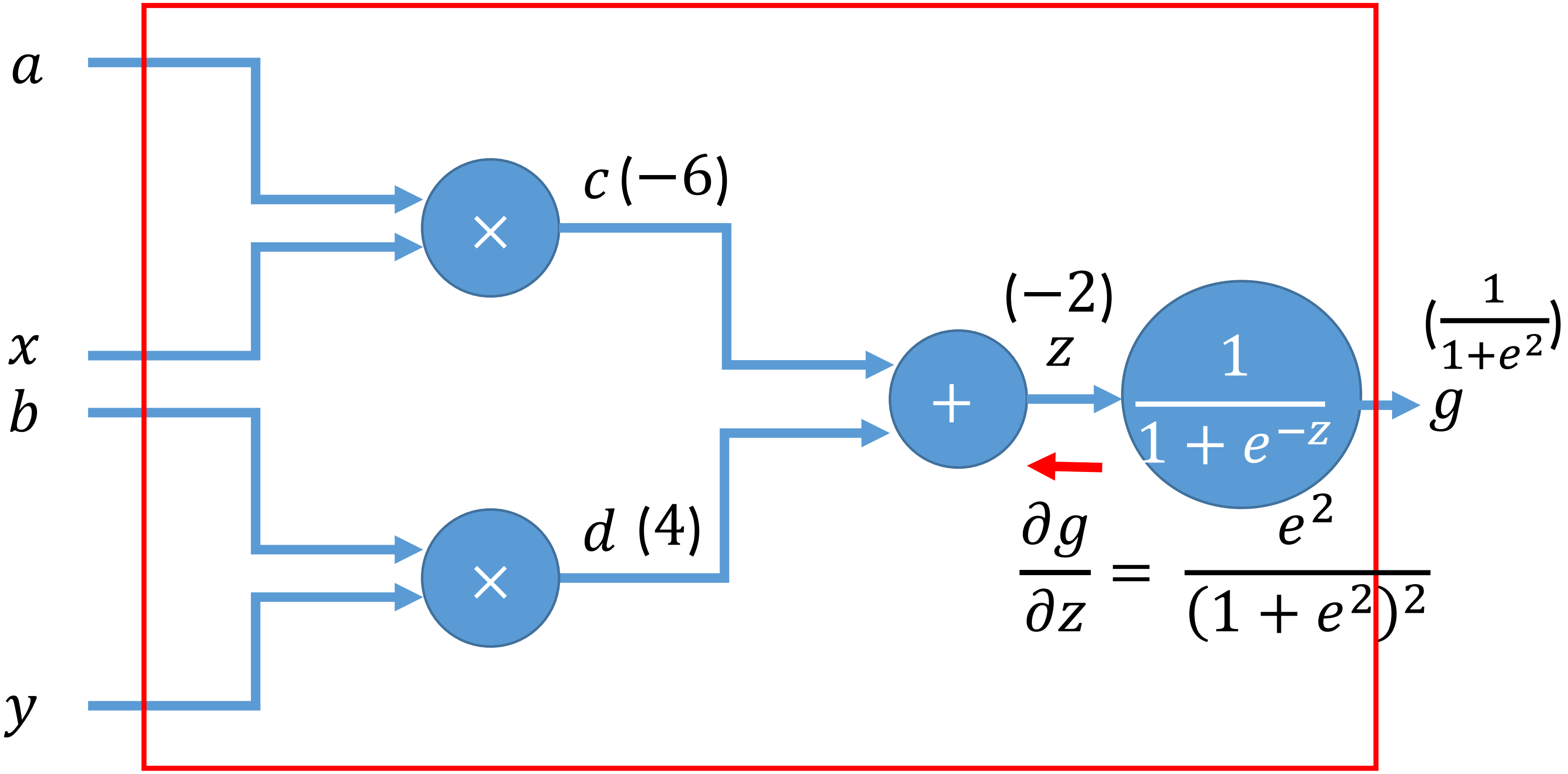
Backprop Example 1

- We can find $\frac{\partial g}{\partial a}, \frac{\partial g}{\partial b}$ and $\frac{\partial g}{\partial c}$ by chain rule!



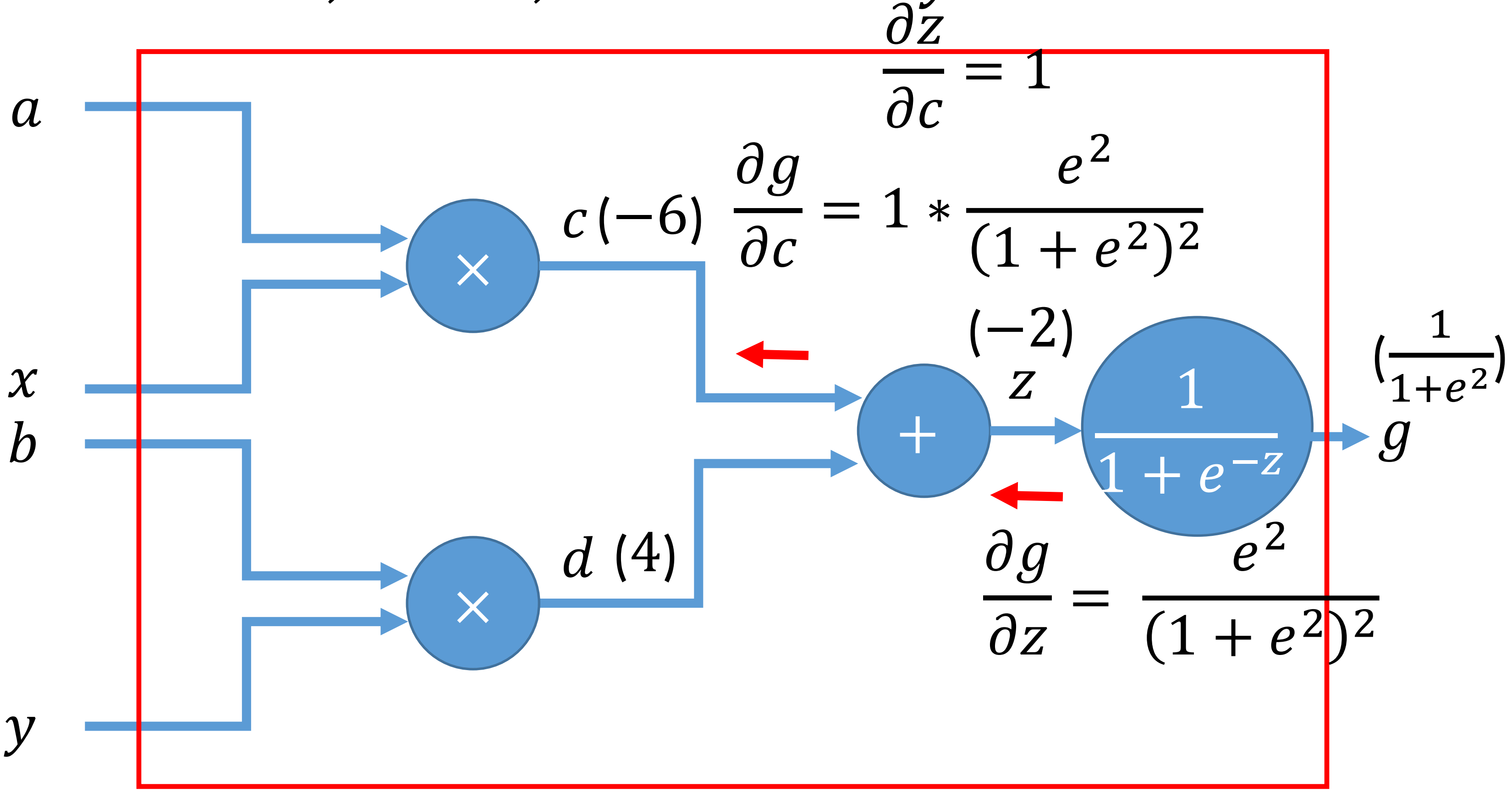
Backprop Example 2

- Consider a function $g(a, b, x, y) = \frac{1}{1+e^{-(ax+by)}}$
- Let $a = 2, b = 1, x = -3$ and $y = 4$



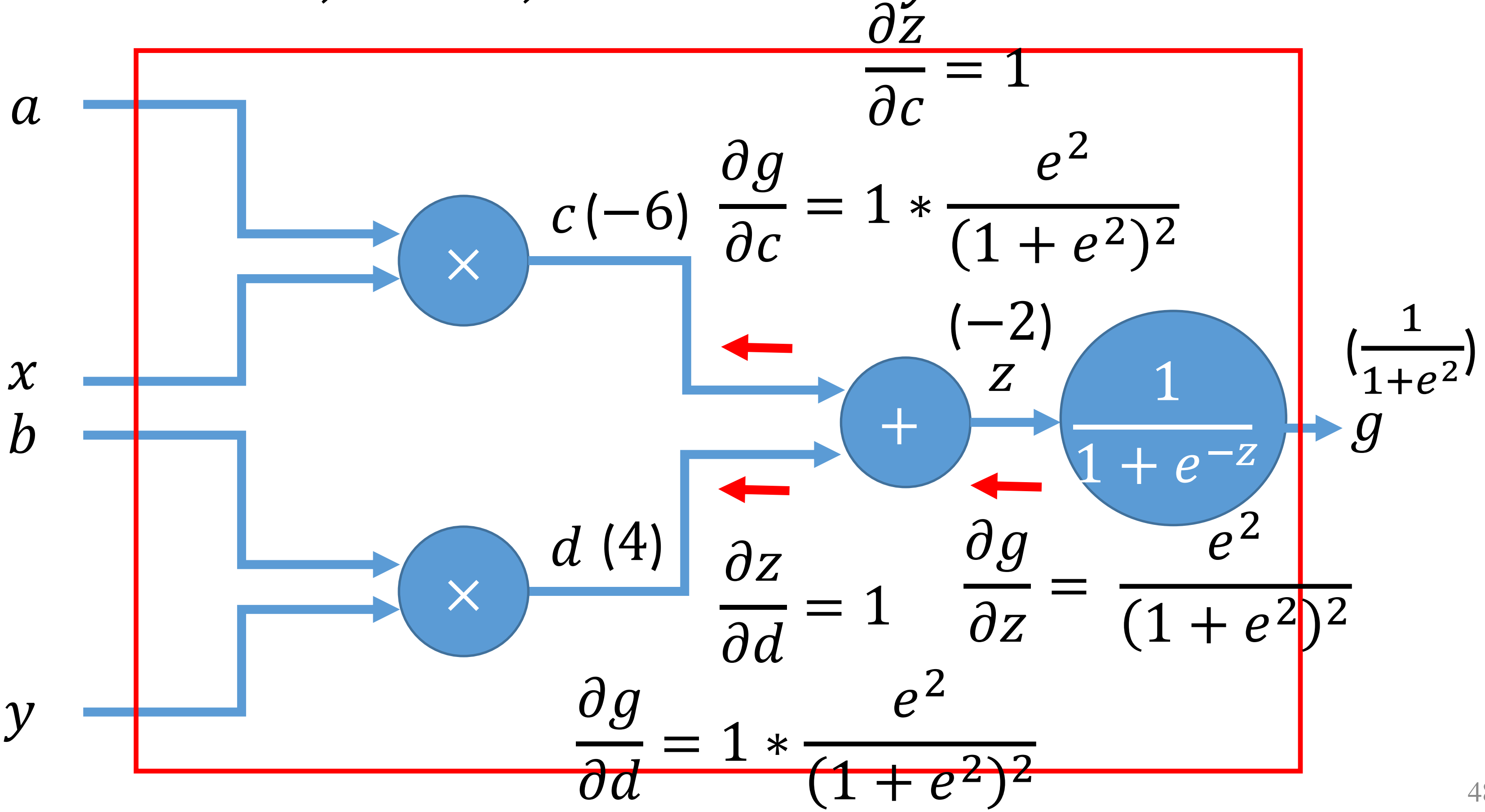
Backprop Example 2

- Consider a function $g(a,b,x,y) = \frac{1}{1+e^{-(ax+by)}}$
- Let $a = 2, b = 1, x = -3$ and $y = 4$



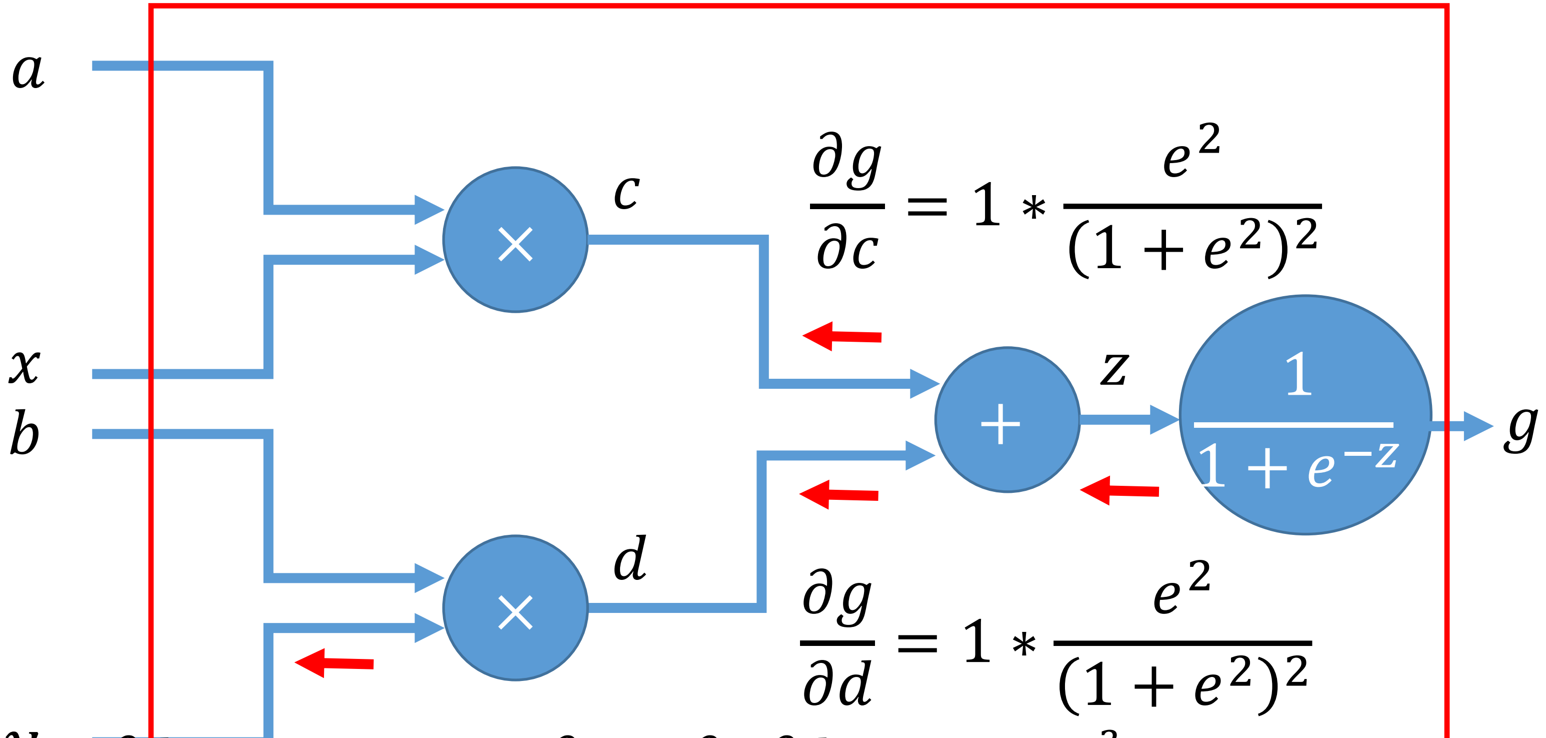
Backprop Example 2

- Consider a function $g(a,b,x,y) = \frac{1}{1+e^{-(ax+by)}}$
- Let $a = 2, b = 1, x = -3$ and $y = 4$



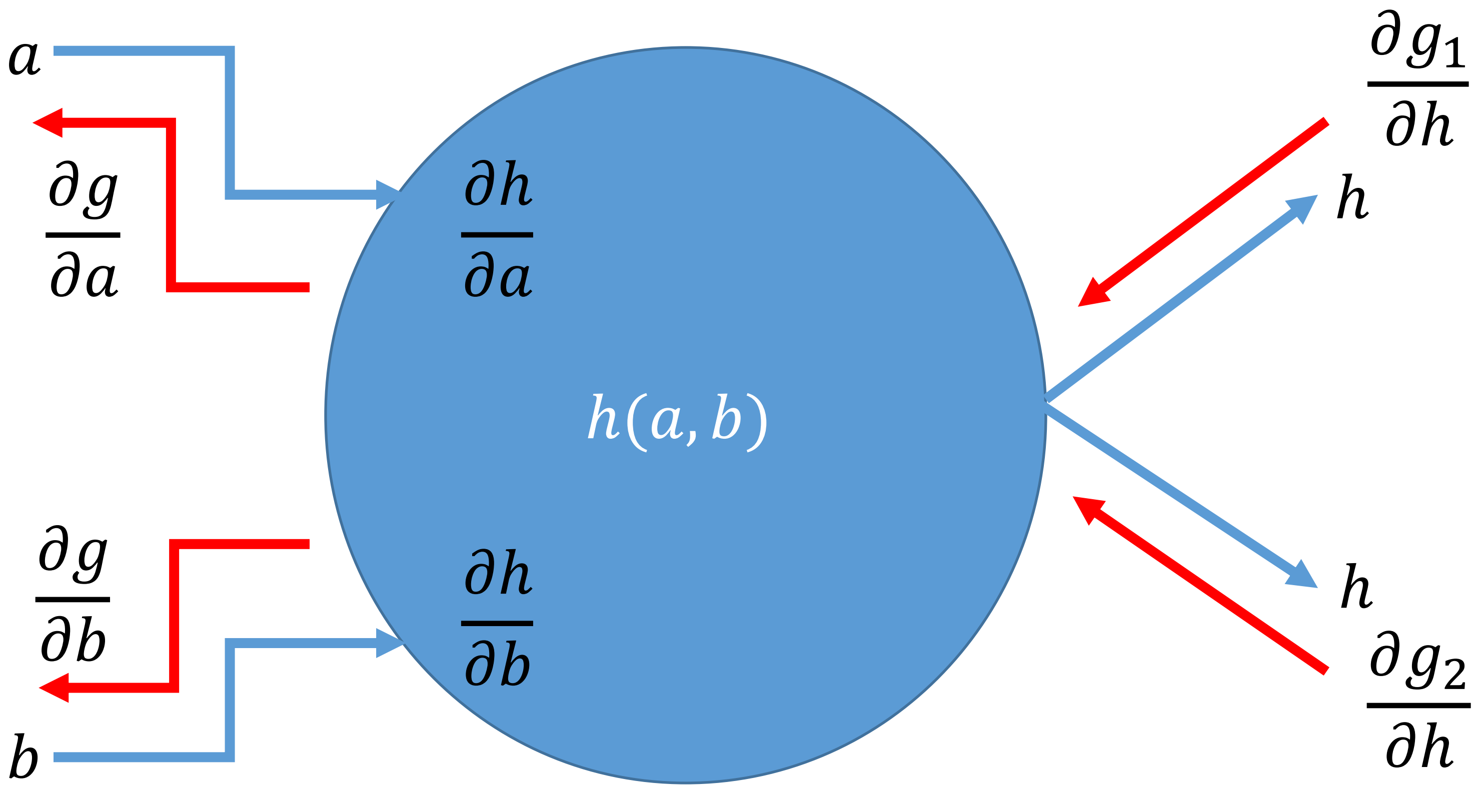
Backprop Example 2

- Consider a function $g(a, b, x, y) = \frac{1}{1+e^{-(ax+by)}}$
- Let $a = 2, b = 1, x = -3$ and $y = 4$



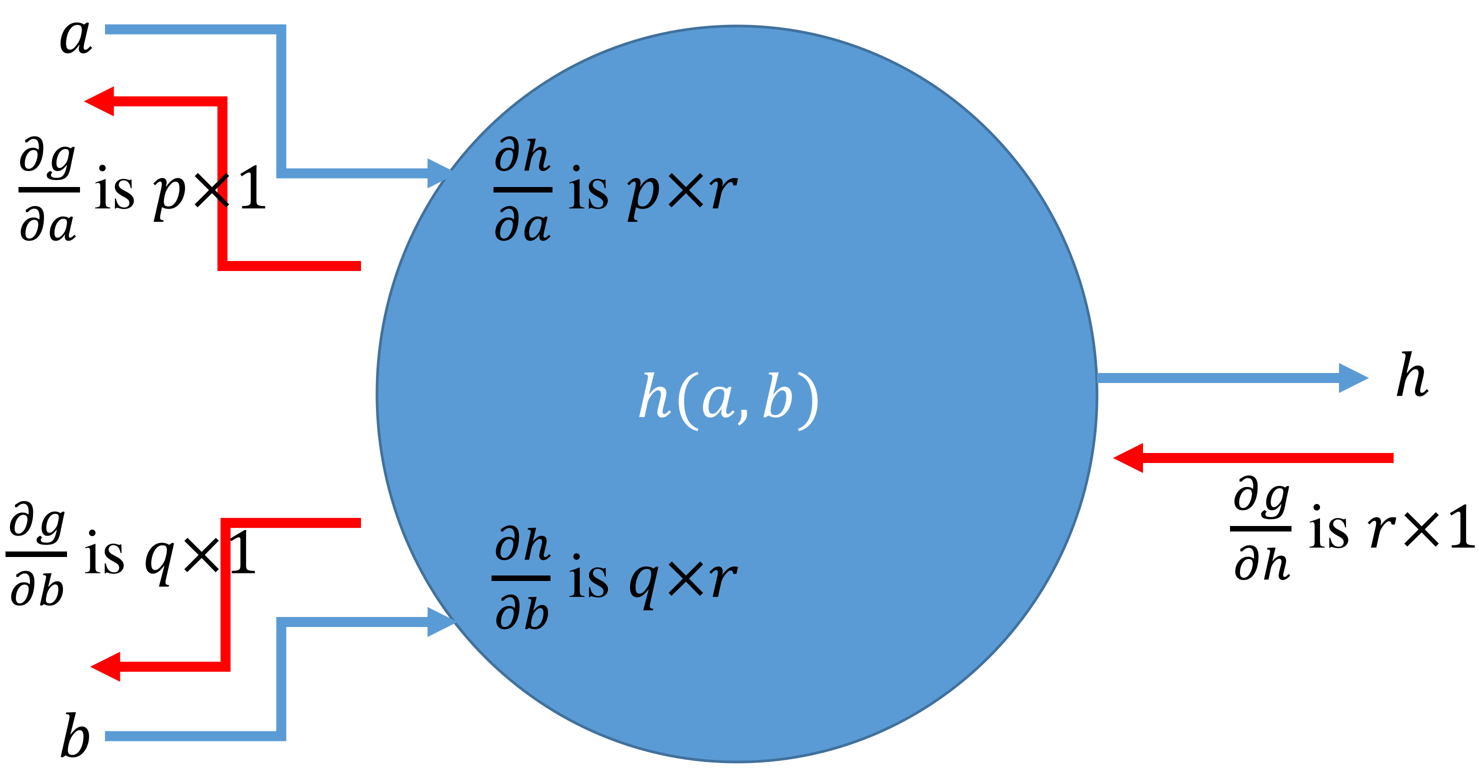
$\frac{\partial d}{\partial y} = b$. Hence, $\frac{\partial g}{\partial y} = \frac{\partial g}{\partial d}\frac{\partial d}{\partial y} = 1 * \frac{e^2}{(1+e^2)^2}$

Backprop for multiple outputs



Backprop for vectors

- Say, a is $p\times1$ dimensional, b is $q\times1$ dimensional and h is $r\times1$ dimensional and g is scalar



Node tracks matrices (cleverly)

Backprop API for a node

- Implement two functions
  - Forward
  - Backward
- Forward
  - Get input from preceding node(s)
  - Track inputs and local gradients
  - Return computation
- Backward
  - Get gradient from succeeding node(s)
  - Compute gradients (simple multiplication)
  - Return gradients to preceding node(s)

Computational Graph API

- Data structure a graph (nodes and directed edges)
- Implement two functions for it
  - Forward
  - Backward
- Forward
  - Recursively pass the inputs to the next nodes
  - Return L
- Backward
  - Recursively traverse the graph backwards
  - Return gradients

Backprop and batched Gradient Descent

- Choose a mini-batch (sample) of size B
- Forward propagate through the computation graph
  - Compute losses $L_{i_1}, L_{i_2}, \dots L_{i_B}$ and $R(W, b)$
  - Get loss L for the batch
- Backprop to compute gradients with respect to W, b
- Update parameters W, b
  - In the direction of the negative gradient

Linear Classifier in Python

```
#Example modified from http://cs231n.github.io/neural-networks-case-study/

#Imports
import numpy as np #Represent ndarrays a.k.a. tensors
import matplotlib.pyplot as plt #For plotting
np.random.seed(0) #For repeatability of the experiment
import pickle #To read data for this experiment

#Setup
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
```

Linear Classifier in Python

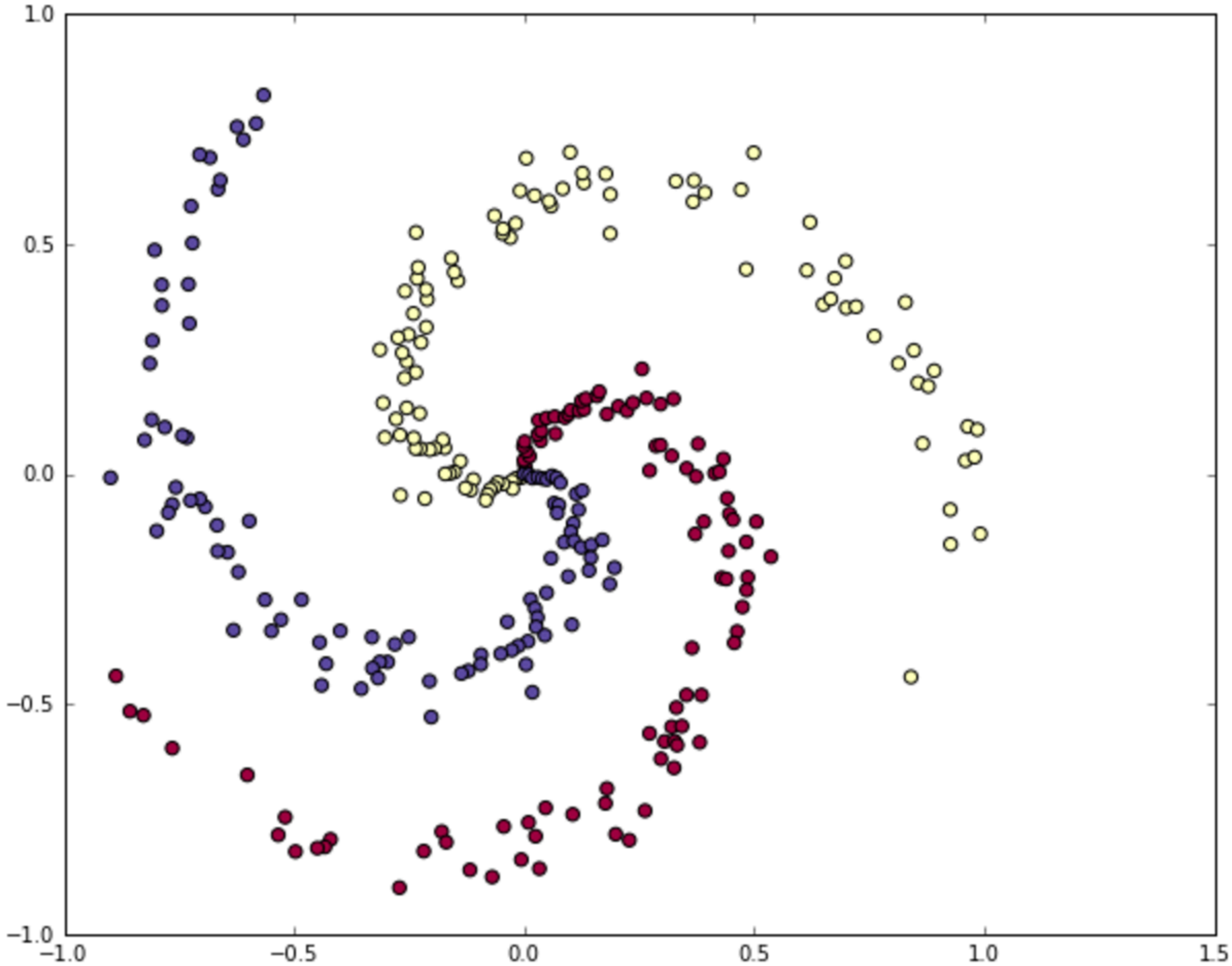
Data

```
#Read data
X = pickle.load(open('dataX.pickle','rb'))
y = pickle.load(open('dataY.pickle','rb'))

#Define some local varaibles
D = X.shape[1] #Number of features
K = max(y)+1 #Number of classes assuming class index starts from 0

#Plot the data
fig = plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)
```

Linear Classifier in Python



Linear Classifier in Python

Model

```
# Linear model

# Start with an initialize parameters randomly
W = 0.01 * np.random.randn(D,K)
b = np.zeros((1,K))

# Initial values from hyperparameter
reg = 1e-3 # regularization strength

#For simplicity, we will not optimize this using grid search here.
```

Linear Classifier in Python

```
#Perform batch SGD using backprop

#For simplicity we will take the batch size to be the same as number of examples
num_examples = X.shape[0]

#Initial value for the Gradient Descent Parameter
step_size = 1e-0 #Also called learning rate

#For simplicity, we will not hand tune this algorithm parameter as well.
```

Linear Classifier in Python

```python
# gradient descent loop
for i in xrange(200):

    # evaluate class scores, [N x K]
    scores = np.dot(X, W) + b

    # compute the class probabilities
    exp_scores = np.exp(scores)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True) # [N x K]

    # compute the loss: average cross-entropy loss and regularization
    corect_logprobs = -np.log(probs[range(num_examples),y])
    data_loss = np.sum(corect_logprobs)/num_examples
    reg_loss = 0.5*reg*np.sum(W*W)
    loss = data_loss + reg_loss
    if i % 10 == 0:
        print "iteration %d: loss %f" % (i, loss)

    # compute the gradient on scores
    dscores = probs
    dscores[range(num_examples),y] -= 1
    dscores /= num_examples

    # backpropate the gradient to the parameters (W,b)
    dW = np.dot(X.T, dscores)
    db = np.sum(dscores, axis=0, keepdims=True)

    dW += reg*W # regularization gradient

    # perform a parameter update
    W += -step_size * dW
    b += -step_size * db
```

Linear Classifier in Python

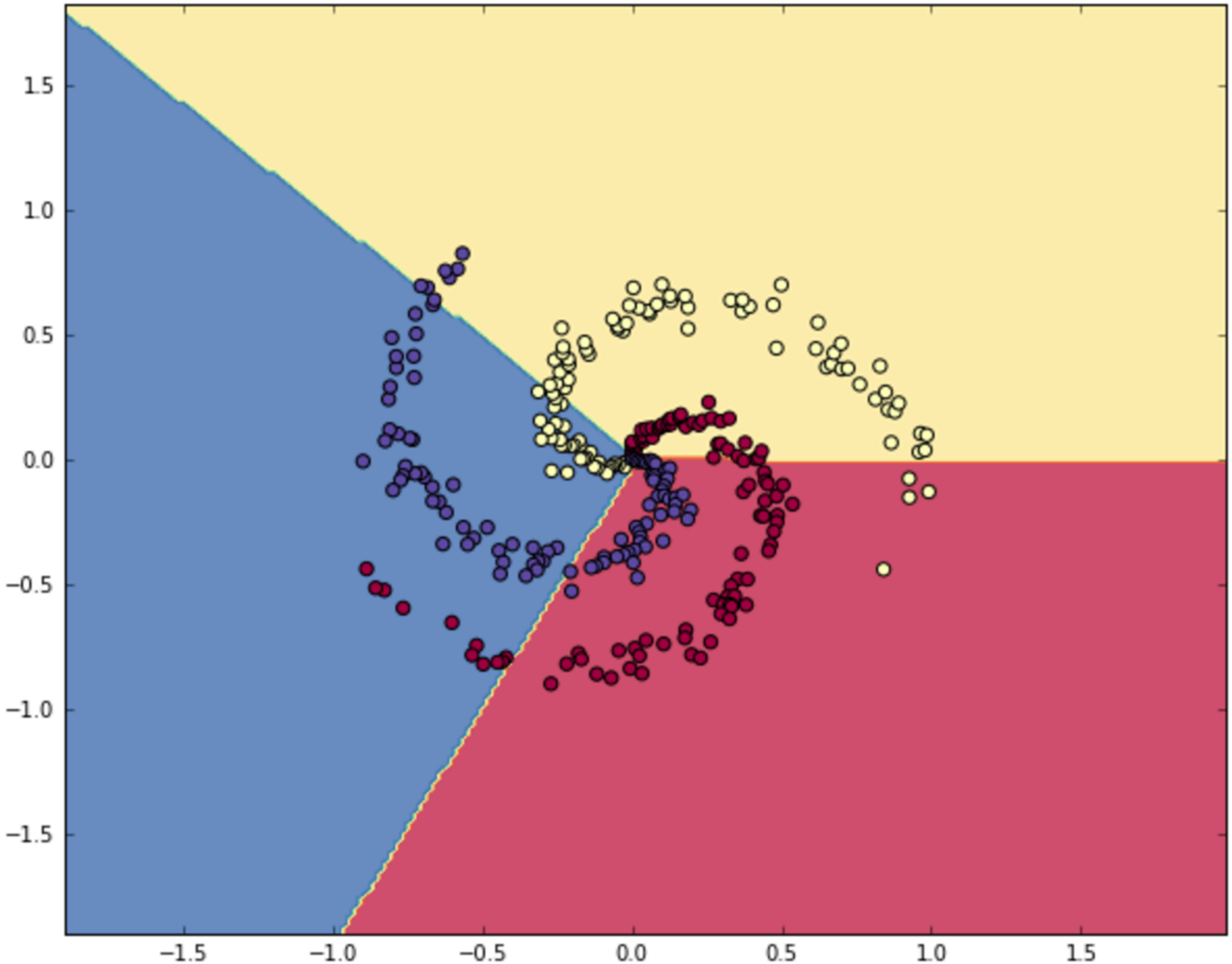
Post Training

```
# Post-training: evaluate test set accuracy

#For simplicity, we will use training data as proxy for test. Do not do this.
X_test = X
y_test = y

scores = np.dot(X_test, W) + b
predicted_class = np.argmax(scores, axis=1)
print 'test accuracy: %.2f' % (np.mean(predicted_class == y_test))
```

Linear Classifier in Python



Questions?

Summary

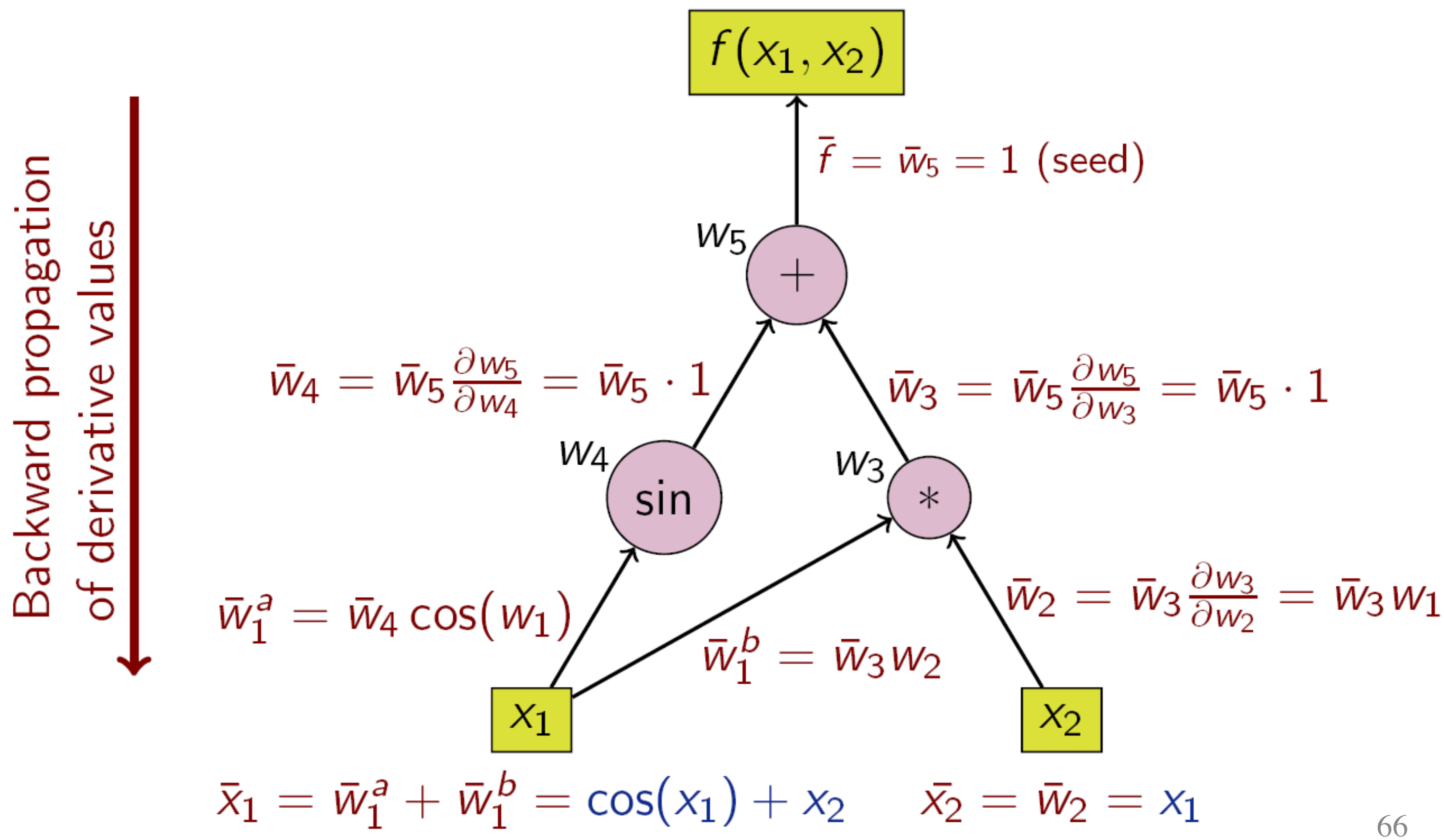
- Data variety poses challenges
  - Missing
  - Noisy
- Complex decisions poses challenges
  - Learning on the go
- We reviewed classification
  - Regression would have similar considerations
- Discussed backpropagation
  - A useful method for optimizing for the best model parameters

Appendix

Reverse mode AutoDiff

- Backpropagation is a case of reverse accumulation automatic differentiation[1]

An example from wikipedia



[1] See https://en.wikipedia.org/wiki/Automatic_differentiation