Neural Networks 101

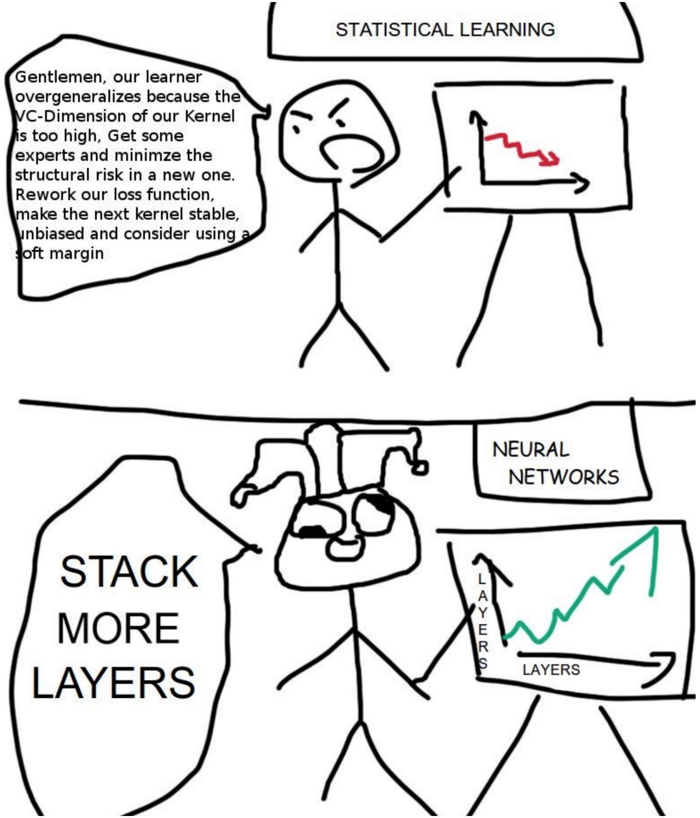
Deep Learning for Natural Language Processing

Vladislav Lialin, Text Machine Lab

Administrative

- Homework 2 is due on Thursday
- Homework 3 will be assigned on Thursday
    - HW3 and the subsequent homeworks will be peer-review graded.
      More details when it is released
- Python code style quiz (graded!): https://forms.gle/DSoju13ZH1gQCzBU7

Why neural networks



Why neural networks

- Accuracy scales better with the amounts of data
- NNs allow to solve conceptually new tasks
  - Multi-task models
  - Transfer learning
  - Multilingual models
  - Unsupervised zero-shot task acquisition (GPT-2, GPT-3)
- Interesting to study as a separate topic (even outside NLP and CV)
  - Very different from traditional ML models
  - A lot of unexpected properties

What you should remember after this lecture

- Fully-connected neural networks
- Activation functions
- Activation function roles
- Stochastic gradient descent
- Backpropagation

What is a neural network?

What is a linear model?

Linear model

$$s = Wx + b$$

Given training data $\{x_i, y_i^{true}\}^N_{i=0}$
how to find the best W and b?

Loss

$$L_i = -\,log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Softmax loss (logistic regression)

$$L_i = \sum_{j \neq y_i} max(0,\, s_j - s_{y_i} + 1)$$

Max-margin loss (SVM)

Really bad idea: random search

```
# assume X_train is the data where each column is an example (e.g. 3073 x 50,000)
# assume Y_train are the labels (e.g. 1D array of 50,000)
# assume the function L evaluates the loss function

bestloss = float("inf") # Python assigns the highest possible float value
for num in xrange(1000):
  W = np.random.randn(10, 3073) * 0.0001 # generate random parameters
  loss = L(X_train, Y_train, W) # get the loss over the entire training set
  if loss < bestloss: # keep track of the best solution
    bestloss = loss
    bestW = W
  print 'in attempt %d the loss was %f, best %f' % (num, loss, bestloss)

# prints:
# in attempt 0 the loss was 9.401632, best 9.401632
# in attempt 1 the loss was 8.959668, best 8.959668
# in attempt 2 the loss was 9.044034, best 8.959668
# in attempt 3 the loss was 9.278948, best 8.959668
# in attempt 4 the loss was 8.857370, best 8.857370
# in attempt 5 the loss was 8.943151, best 8.857370
# in attempt 6 the loss was 8.605604, best 8.605604
# ... (trunctated: continues for 1000 lines)
```

Slide credit: Stanford CS231n

Results

```
# Assume X_test is [3073 x 10000], Y_test [10000 x 1]
scores = Wbest.dot(Xte_cols) # 10 x 10000, the class scores for all test examples
# find the index with max score in each column (the predicted class)
Yte_predict = np.argmax(scores, axis = 0)
# and calculate accuracy (fraction of predictions that are correct)
np.mean(Yte_predict == Yte)
# returns 0.1555
```

15.5% accuracy! not bad!
(SOTA is ~99.7%)

Slide credit: Stanford CS231n

Alternatives to random search

What do we want?

- We want our loss to improve every step (ideally)
- We want our final solution to be a **global minimum** of the function (ideally)

Alternatives to random search

What do we want?

- We want our loss to improve every step (ideally)
- We want our final solution to be a **global minimum** of the function (ideally)

Possible solutions:

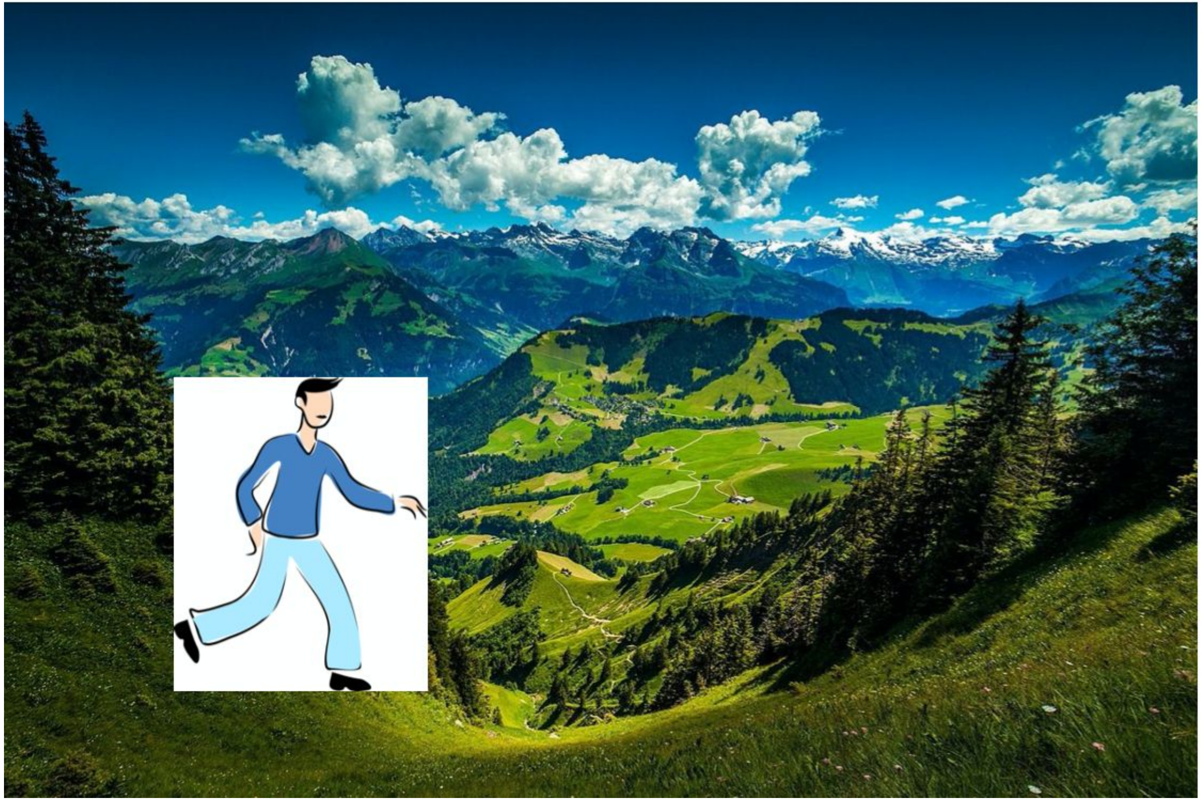
- More clever random search: genetic algorithms
- Quadratic optimization (SVM)
- Gradient descent

Gradient descent

$$W_i = W_{i-1} - \eta \frac{\partial L(y, \hat{y})}{\partial W}$$

$$b_i = b_{i-1} - \eta \frac{\partial L(y, \hat{y})}{\partial b}$$

Gradient descent

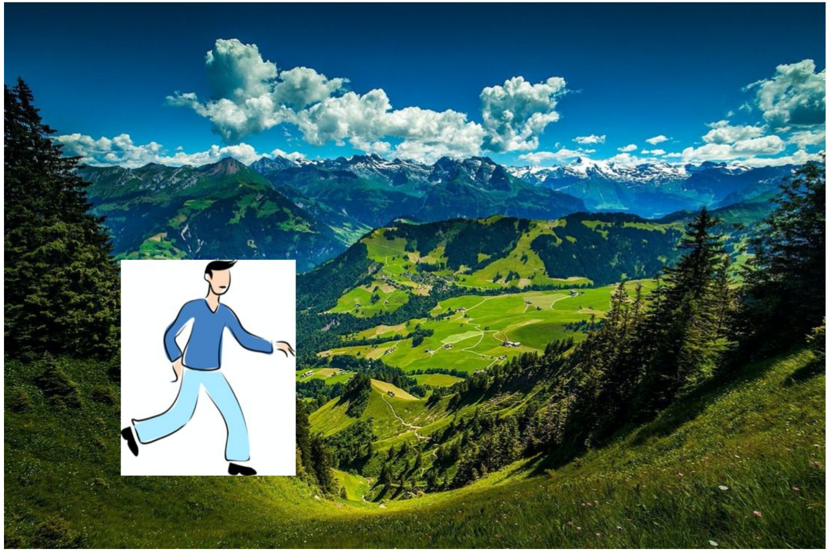


Gradient descent

```python
X_train, y_train = dataset
n_examples, n_features = X.shape
n_examples, n_classes = y.shape

W = np.random.randn(n_features, n_classes) * 0.0001
b = np.random.randn(n_classes) * 0.0001

lr = 0.01  # learning rate

for _ in range(1000):
  scores = X_train @ W + b
  loss = L(scores, y_train)

  W_grad, b_grad = get_gradient(loss, with_respect_to=(W, b))

  W -= lr * W_grad
  b -= lr * b_grad
```

1. Init W and b with random numbers
2. Compute loss
3. Compute loss gradient with respect to W and b
4. Update W and b
5. Repeat 2-4 until some stop criterion is triggered

Gradient descent



$$W_i = W_{i-1} - \eta \frac{\partial L(y, \hat{y})}{\partial W}$$

$$b_i = b_{i-1} - \eta \frac{\partial L(y, \hat{y})}{\partial b}$$

```
while True:
    weights_grad = compute_gradient(loss_fn, data, weights)
    weights -= step_size * weights_grad
```

Slide credit: Stanford CS231n
Image credit: [Landscape image], [walking man image]

Gradient descent demo

Click

Problem: how to compute gradients

$$s = Wx + b$$

$$L_i = -\,log\left(\frac{e^{s y_i}}{\sum_j e^{s_j}}\right)$$

$$\frac{\partial L(y, \hat{y})}{\partial W_1} = ?$$

$$\frac{\partial L(y, \hat{y})}{\partial W_2} = ?$$

Problem: how to compute gradients

$$s = Wx + b$$

$$L_i = -\,log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

$$\frac{\partial L(y, \hat{y})}{\partial W_1} = ?$$

$$\frac{\partial L(y, \hat{y})}{\partial W_2} = ?$$

Just compute them analytically

What is a neural network?

Single-layer neural network (perceptron)

$$y = xW + b$$

Single-layer neural network (perceptron)

$$y = f(xW + b)$$

f is an (arbitrary) element-wise nonlinear function - activation function

Single-layer neural network (perceptron)

Logistic regression

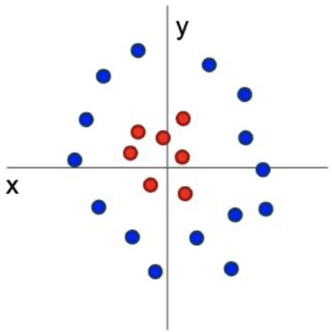
$$y = f(xW + b)$$

f = 1 / (1 - exp(-x)) – sigmoid function

Why single layer is not enough

Visual viewpoint:
linear models can only learn one template per class

Geometric viewpoint:
linear models can only draw linear boundaries



$$y = f(xW + b)$$

Slide credit: Stanford CS231n
Image credit: Stanford CS231n

Solution: Multilayer Neural Network

(Other names: multilayer perceptron / feedforward NN / fully-connected NN)

$$y_1 = f(x_1 W_1 + b_1)$$

$$y_2 = f(y_1 W_2 + b_2)$$

$$\cdots$$

$$y_n = f(y_{n-1} W_n \quad + b_n \quad)$$

Main idea: following layer receives **nonlinearly** transformed features from the previous layer

Solution: Multilayer Neural Network

(Other names: multilayer perceptron / feedforward NN / fully-connected NN)

$$y_1 = \underline{f}(x_1 W_1 + b_1)$$

$$y_2 = \underline{f}(y_1 W_2 + b_2)$$

$$\cdots$$

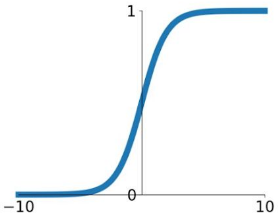
$$y_n = \underline{f}(y_{n-1} W_n \quad + b_n \quad)$$

What if the activation function is linear?

Activation functions

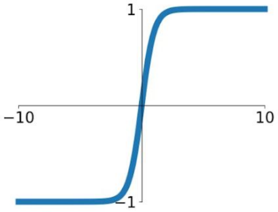
Sigmoid

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



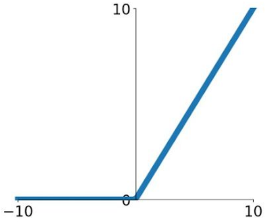
tanh

$\tanh(x)$



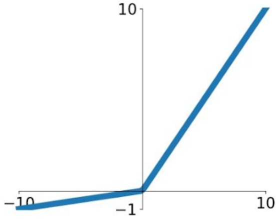
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

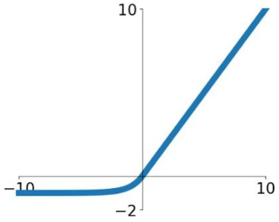


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

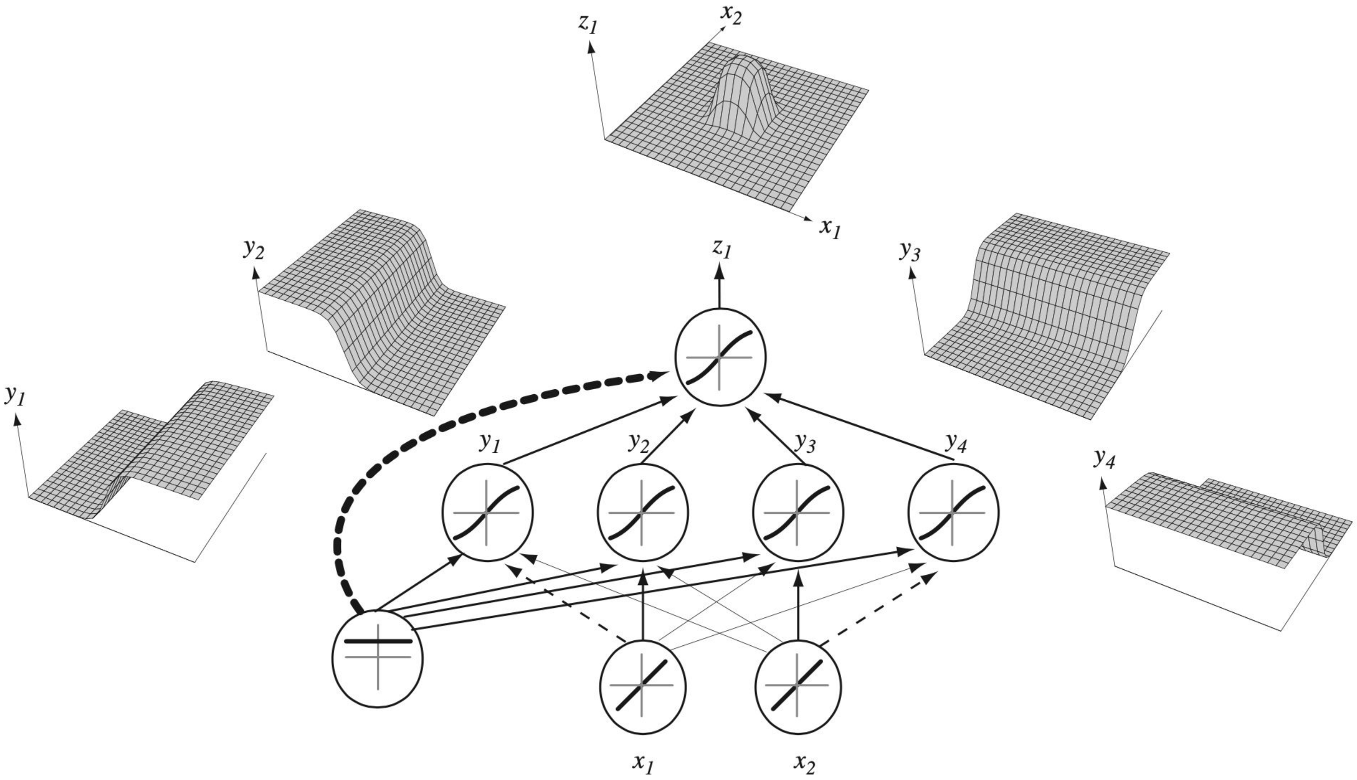
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



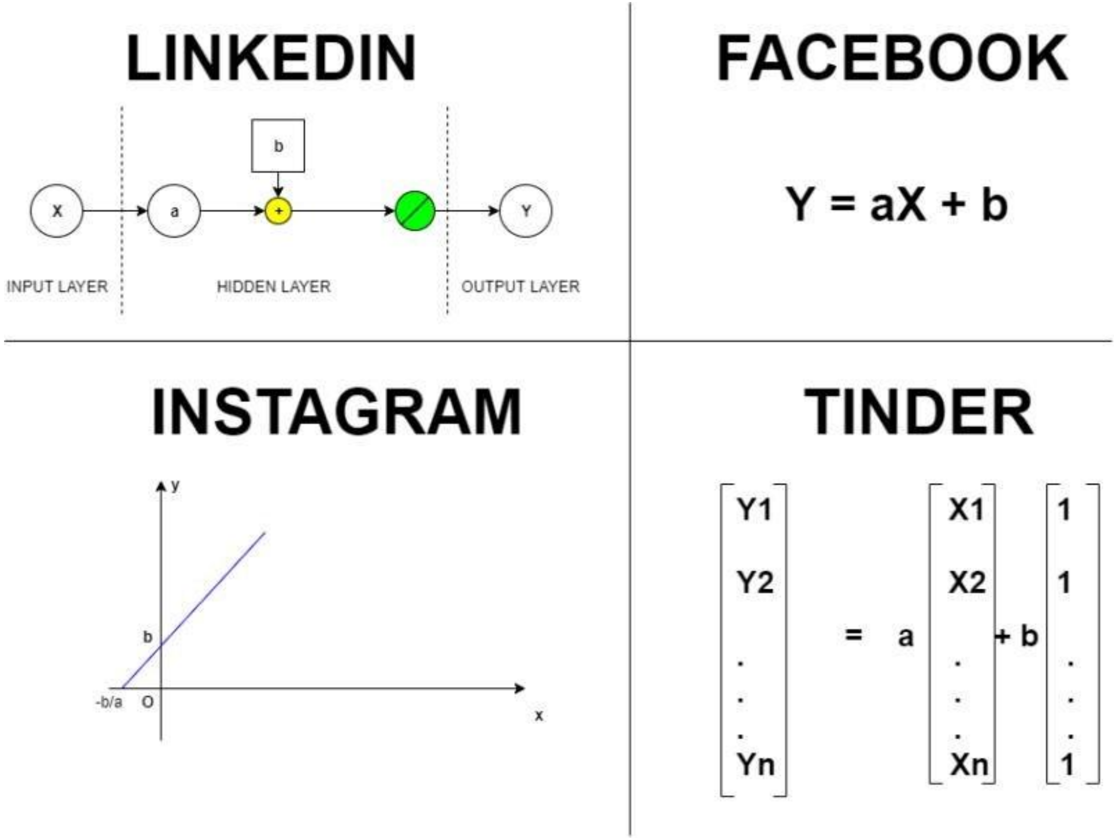
We'll talk more about choosing activation functions in the next lecture

Slide credit: Stanford CS231n
Image credit: Stanford CS231n

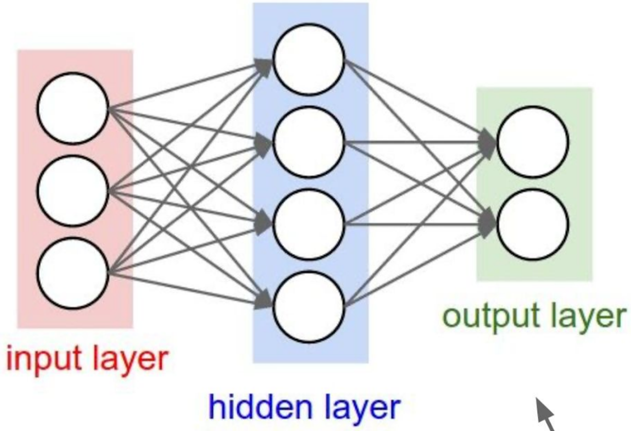
Example: neural network with tanh activation

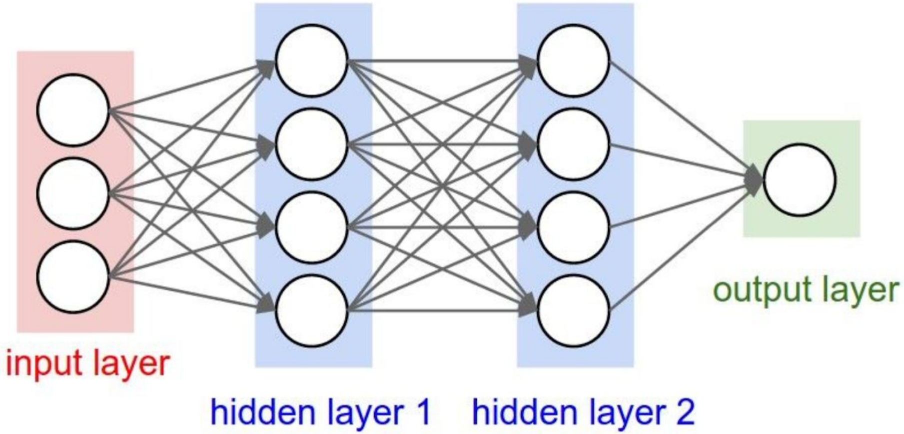


(Too many) ways to describe neural networks



Architecture of a neural network



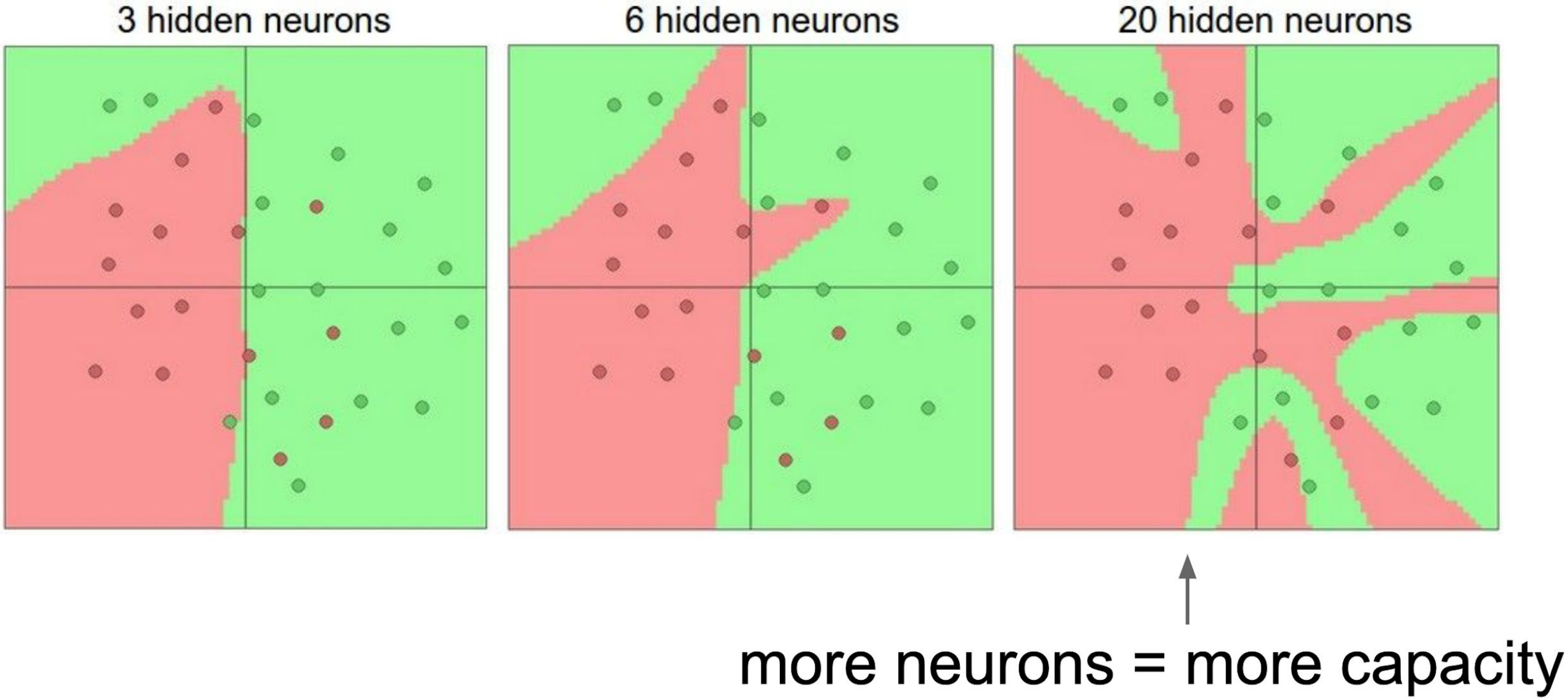


“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“Fully-connected” layers

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

Effect of a hidden layer size

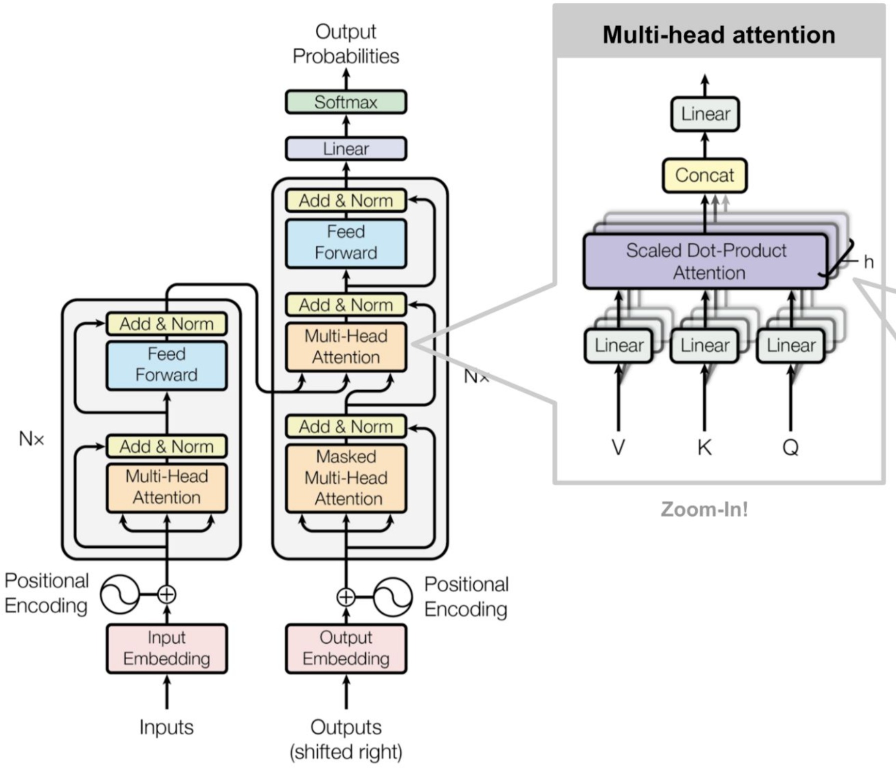


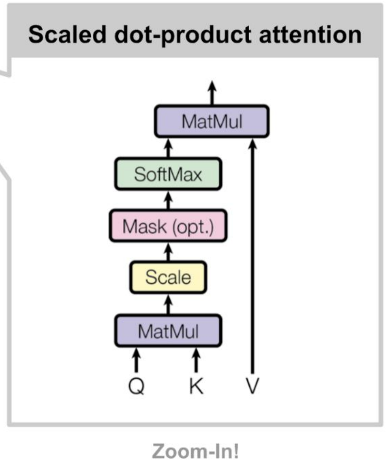
Slide credit: Stanford CS231n

More about fully-connected networks

- Visual and interactive recap on linear models
  by Jay Alammar
- Toy 2d classification with 2-layer neural network
  by Andrej Karpathy
- Tinker With a Neural Network Right Here in Your Browser
  by Daniel Smilkov

Architecture of a neural network

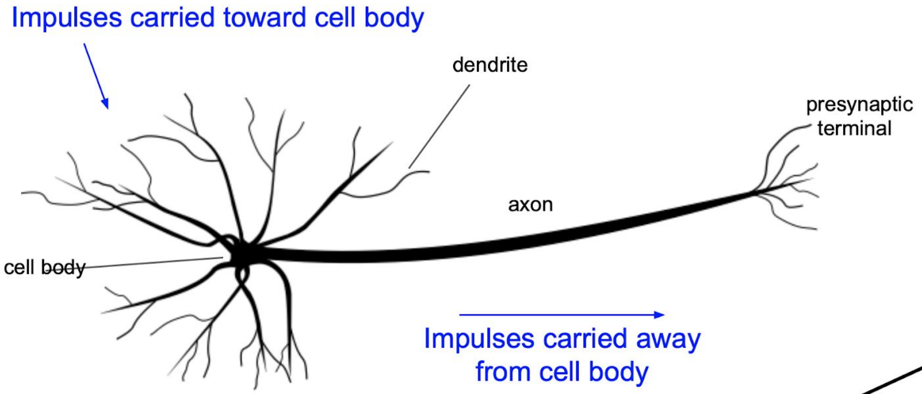




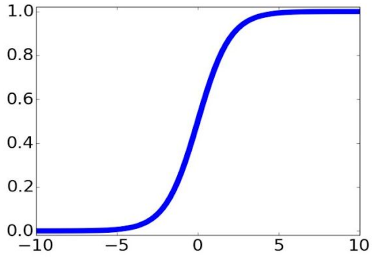
It becomes harder to draw them, but the equations are still pretty simple

Image credit:
arxiv.org/abs/1706.03762 and primo.ai

Biological analogy: neuron

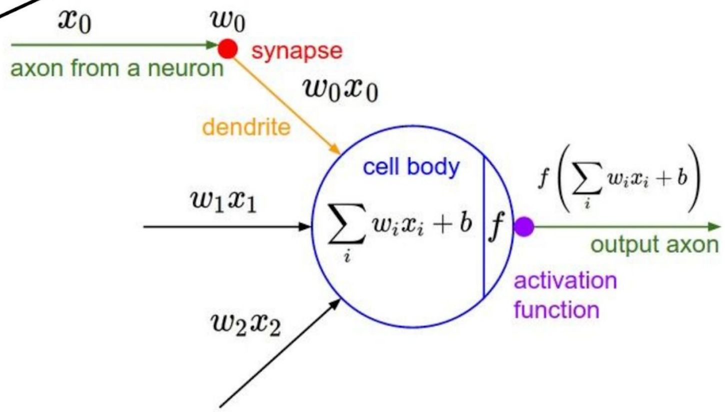


This image by Felipe Perucho is licensed under CC-BY 3.0



sigmoid activation function

$$\frac{1}{1+e^{-x}}$$



Slide credit: Stanford CS231n
Image credit: Stanford CS231n and Felipe Perucho

Be careful with your brain analogies

Biological Neurons:

- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system
- Biological neural networks build complex connectivity patterns
- Artificial neural networks
  - Organized into regular layers for computational efficiency
  - But can work even with random connections [1]

Slide credit: Stanford CS231n

1: Xie et al, “Exploring Randomly Wired Neural Networks for Image Recognition”

Break

Training neural networks, backpropagation

Problem: how to compute gradients

$$y = max\{xW_1 + b_1, 0\}$$

$$p = \sigma(yW_2 + b_2)$$

$$L = -(\hat{p}\log p + (1-\hat{p})\log(1-p))$$

$$\frac{\partial L(y,\hat{y})}{\partial W_1} = ?$$

$$\frac{\partial L(y,\hat{y})}{\partial W_2} = ?$$

By hand?

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

Full implementation of training a 2-layer neural network from scratch in 20 lines of code

Problem: how to compute gradients

$$y_1 = max\{xW_1 + b_1, 0\}$$
$$y_2 = max\{y_1W_2 + b_2, 0\}$$
$$\cdots$$
$$y_{n-1} = max\{y_2W_{n-2} + b_{n-2}, 0\}$$
$$p = softmax(yW_{n-1} + b_{n-1})$$
$$L = -\sum \hat{p} \log p$$

$$\frac{\partial L(y, \hat{y})}{\partial W_1} = ?$$
$$\cdots$$
$$\frac{\partial L(y, \hat{y})}{\partial W_{n-1}} = ?$$

By hand?

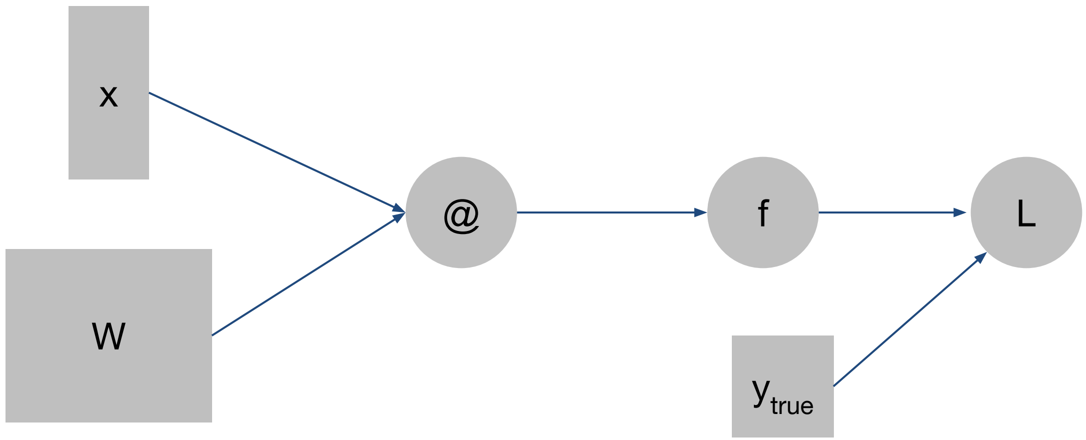
Solution: backpropagation

Two main ideas:

- Chain rule
- Memorizing intermediate values

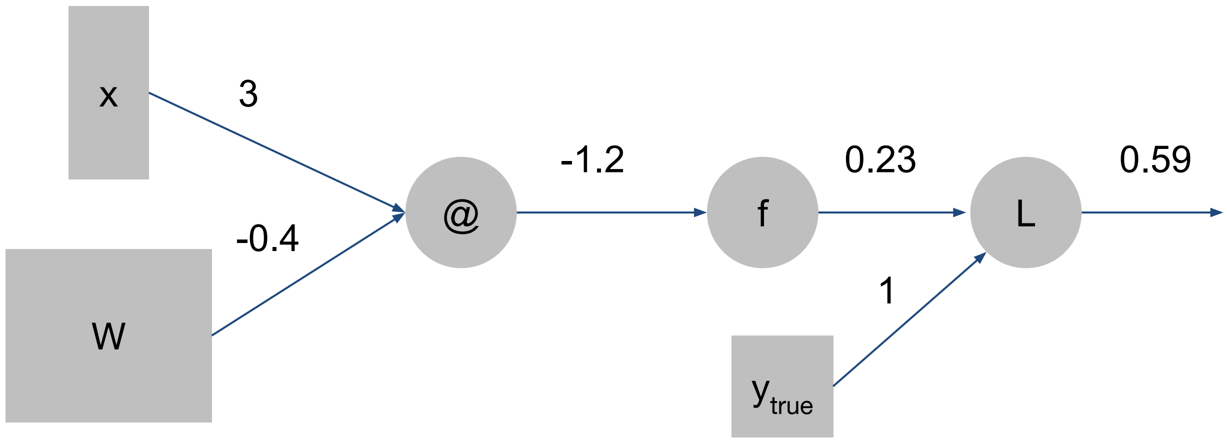
$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \quad + \quad \text{memorization}$$

Computational graph



@ - matrix multiplication

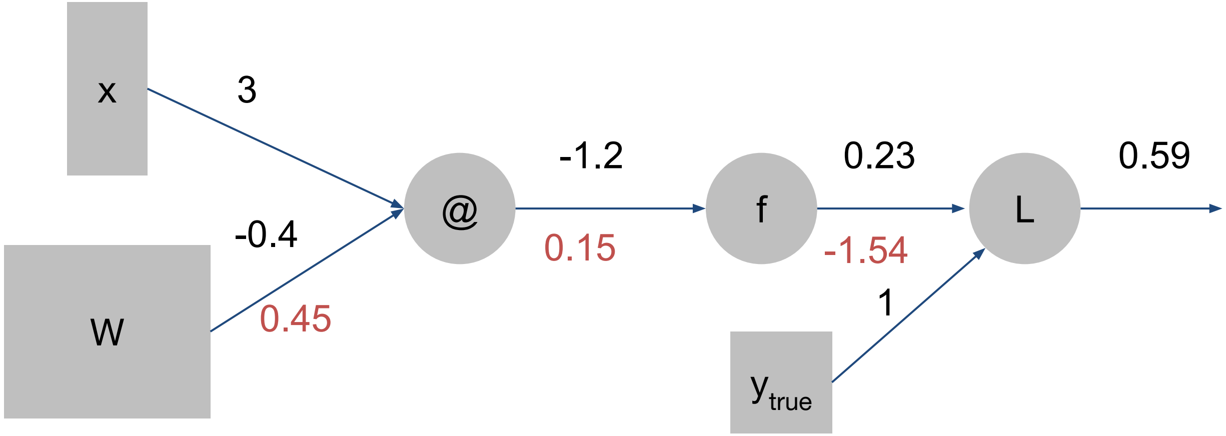
Forward pass (execution)



f - sigmoid

$L(y, y_{true}) = (y - y_{true})^2$

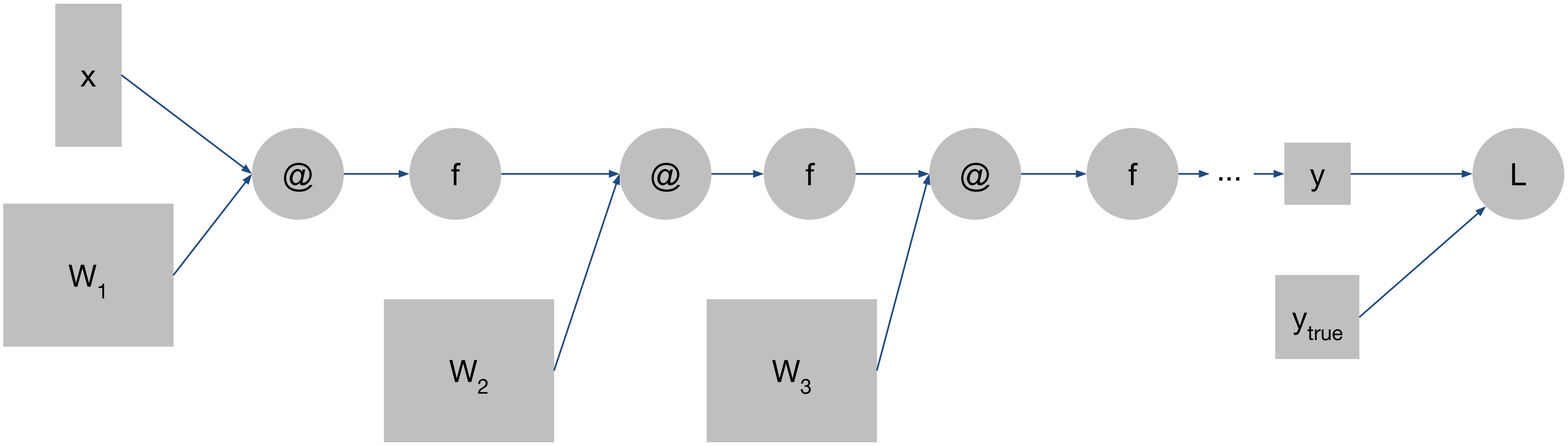
Backward pass (computing the gradients)



f - sigmoid

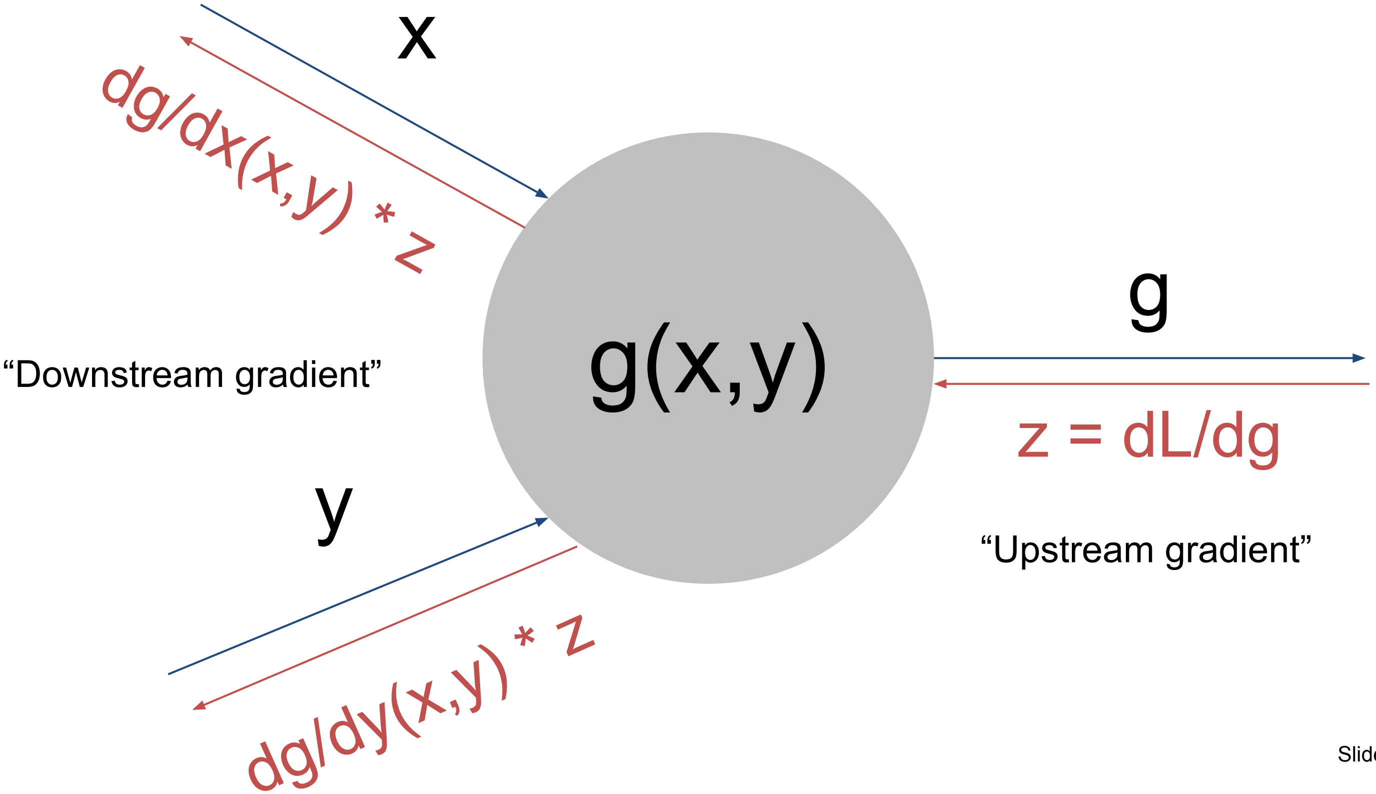
$L(y, y_{true}) = (y - y_{true})^2$

Other graph, same principle



Note: you need to store intermediate values to compute backward pass efficiently.

A closer look



Slide credit: Stanford CS231n