CSCI 497P/597P: Computer Vision

(Stochastic) Gradient Descent
Neural networks

Readings

with a great deal more detail…

- http://cs231n.github.io/optimization-1/
- http://cs231n.github.io/optimization-2/
- http://cs231n.github.io/neural-networks-1/
- http://cs231n.github.io/neural-networks-2/
- http://cs231n.github.io/neural-networks-3/

Announcements

- Last project - P4 (AlexNet)
  - Out tomorrow 5/27
  - Due a week from tomorrow 6/3
- HW3 out, due Monday
  - your lowest homework score is dropped

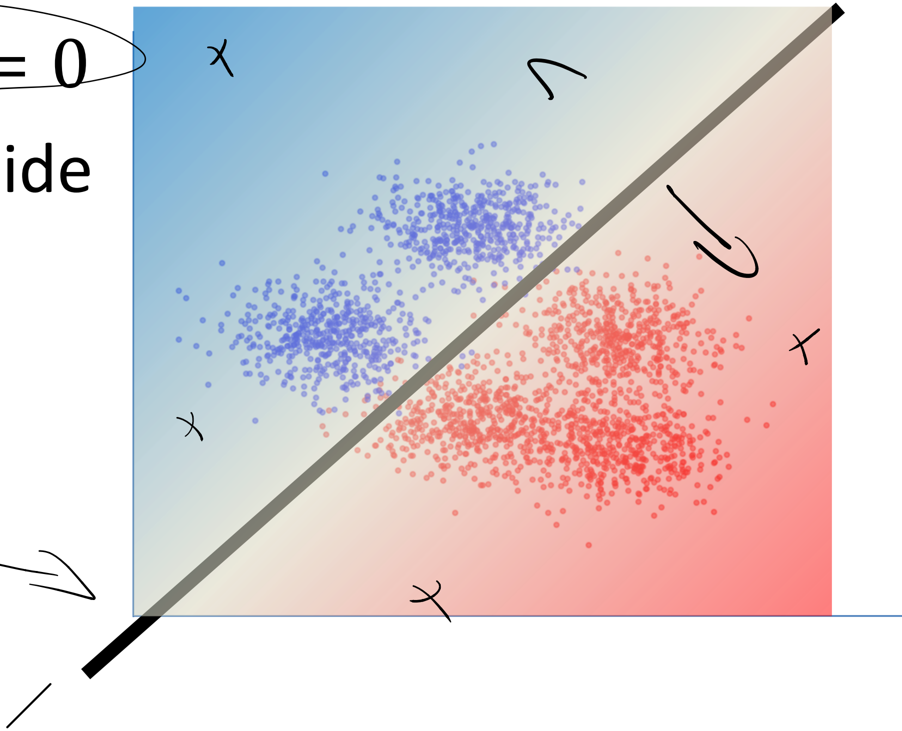
Goals

- Understand how to train a classifier by minimizing a loss function using gradient descent.
- Understand the intuition behind using Stochastic (Minibatch) Gradient Descent.
- Understand neural networks as a stack of linear classifiers with nonlinearities (activation functions) in between.
  - Understand why we need activation functions.
  - Understand the vanishing gradients problem.

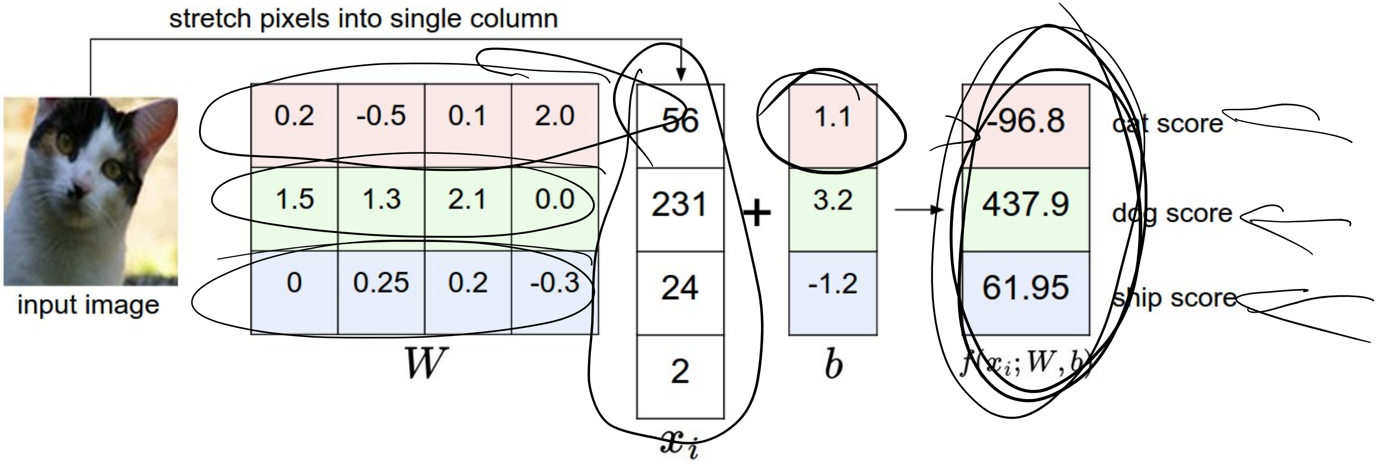
Linear classifiers

- Equation: $\overset{1\times d}{w^T}\overset{d\times 1}{x} + b = 0$
- Points on the same side are the same class

$ax+by+c=0$

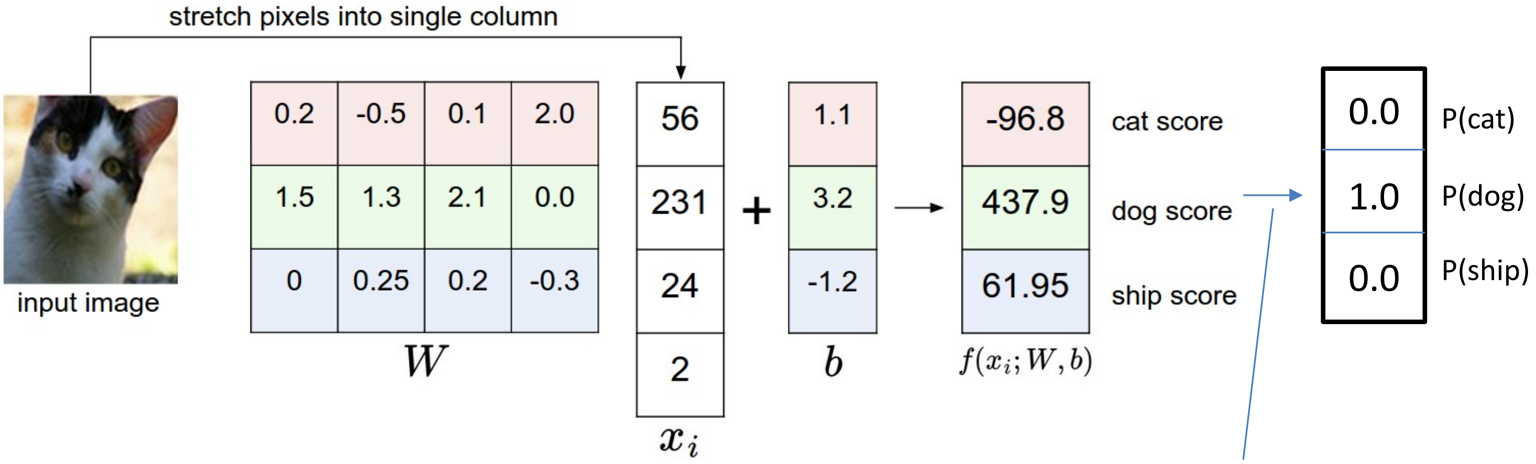


Multiclass Linear Classifiers: Stack multiple w^T into a matrix.



$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} \rightarrow \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{f(x_i; W, b)}$$

Multiclass Linear Classifiers: Stack multiple w^T into a matrix.

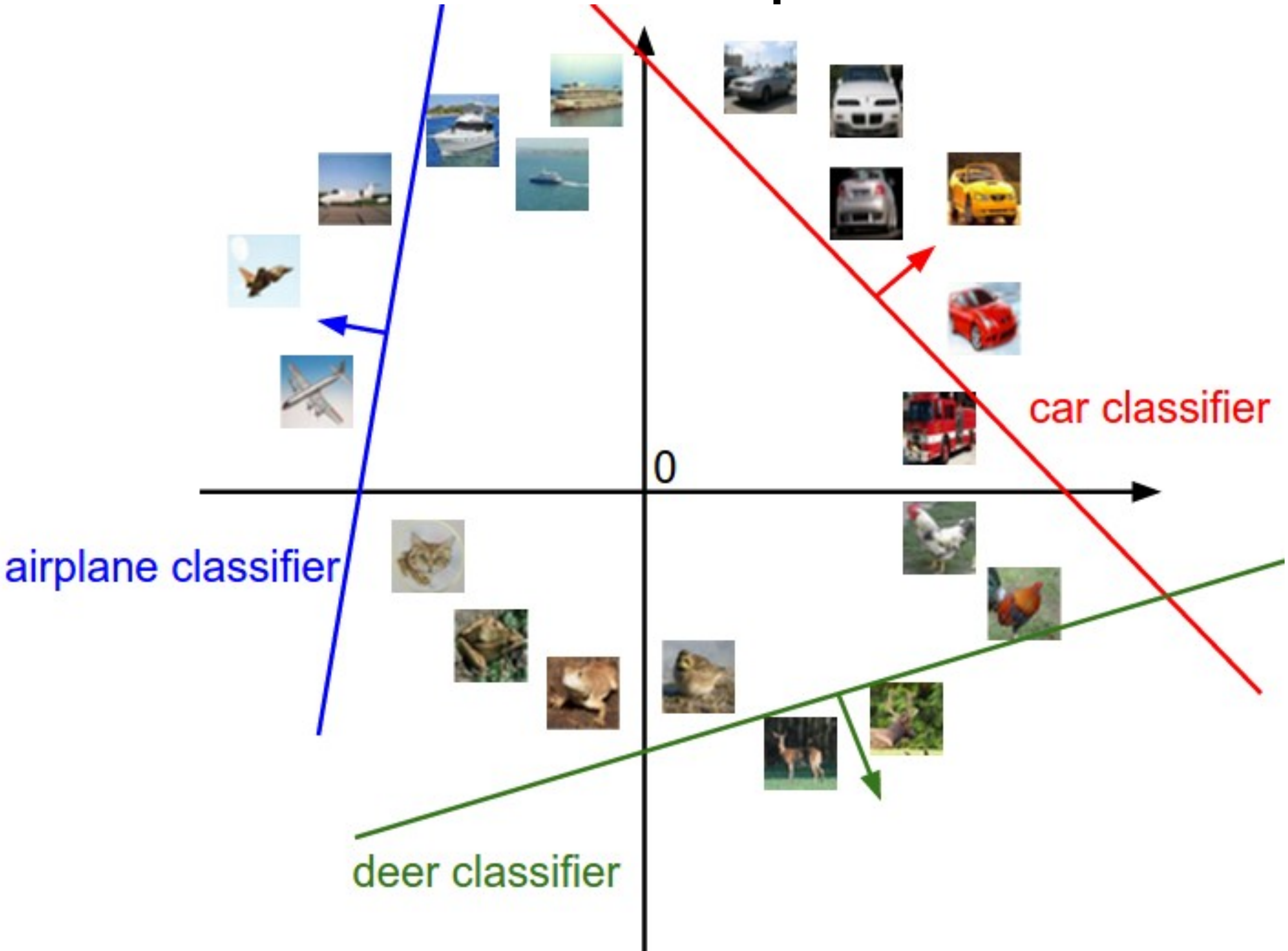


$$\underset{W}{\begin{bmatrix}0.2 & -0.5 & 0.1 & 2.0\\ 1.5 & 1.3 & 2.1 & 0.0\\ 0 & 0.25 & 0.2 & -0.3\end{bmatrix}} \underset{x_i}{\begin{bmatrix}56\\ 231\\ 24\\ 2\end{bmatrix}} + \underset{b}{\begin{bmatrix}1.1\\ 3.2\\ -1.2\end{bmatrix}} \rightarrow \underset{f(x_i; W, b)}{\begin{bmatrix}-96.8\\ 437.9\\ 61.95\end{bmatrix}} \rightarrow \begin{bmatrix}0.0\\ 1.0\\ 0.0\end{bmatrix}$$

Softmax normalization

$$\frac{e^{f_k}}{\sum_j e^{f_j}}$$

Multiclass Linear Classifier: Geometric Interpretation



Taking stock

- We have:
  - ϕ = unravel(rgb2gray(img)), a feature extractor
  - $h(x) = \underline{W}^T x$, a multiclass linear classifier
  - $\boxed{L} = \sum_{i=0}^{N} \boxed{L_i}$, a loss function

for each data example

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

- We don't have:
  - a way to find a W that results in a small L.

Minimizing the Loss

- Use **optimization** to find the W that *minimizes* the loss function.
  - Linear regression: solvable in closed form
  - Most of the time: no closed form.

Optimization

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Walking man image is CC0 1.0 public domain

Finding a W that minimizes L

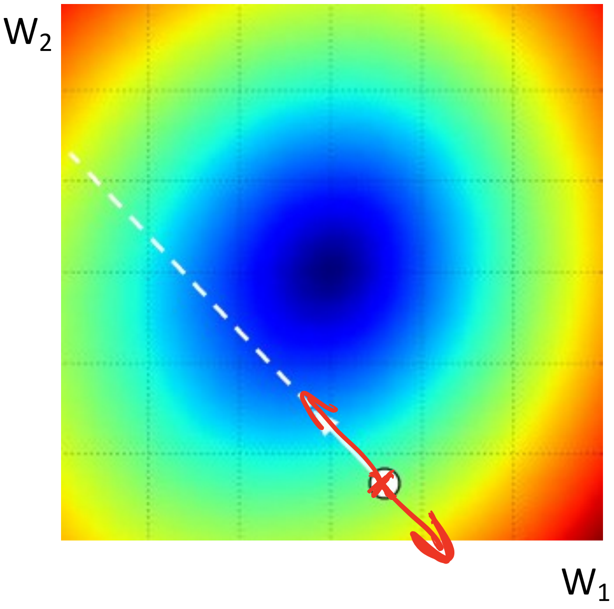
- Simple idea: walk downhill.

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Gradient Descent: Generally

$$L\left(\overbrace{X,Y,}^{\text{Fixed}}\ \overbrace{w_1,w_2}^{\text{weights}}\right)$$

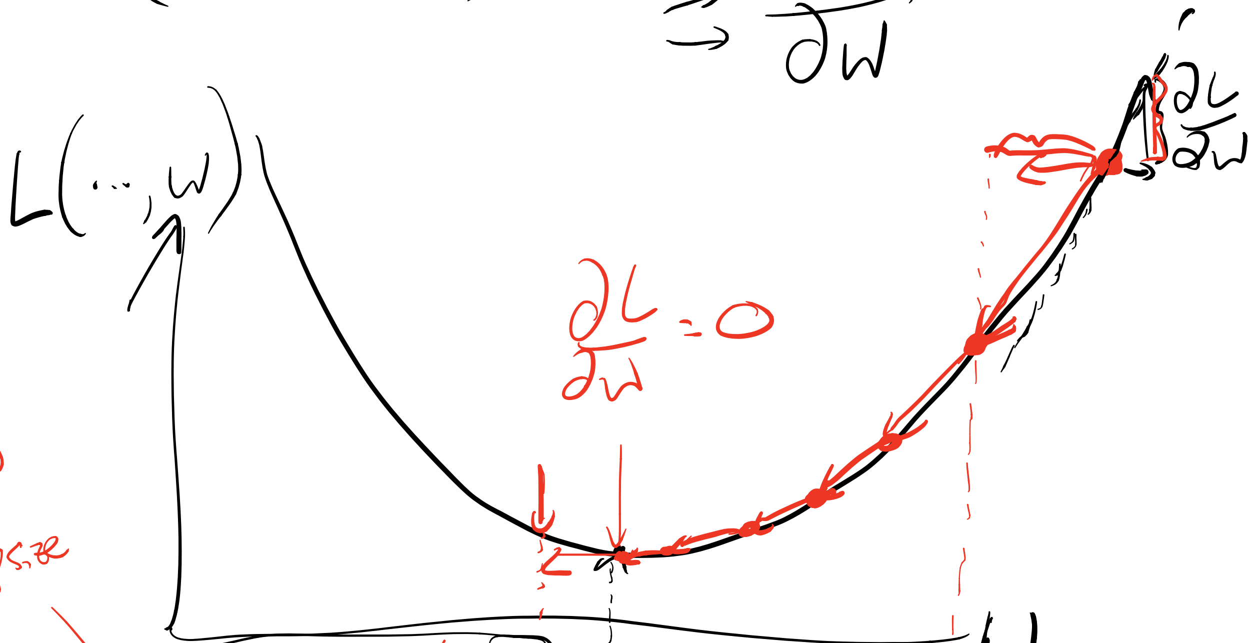
- Gradient of the loss function with respect to the *weights* tells us how to change the weights to improve the loss.



Gradient Descent: Intuition

inputs ↓ labels ↓ parameter of classifier ↓

$$L(X, Y, w) \Rightarrow \frac{\partial L}{\partial w}$$



learning rate

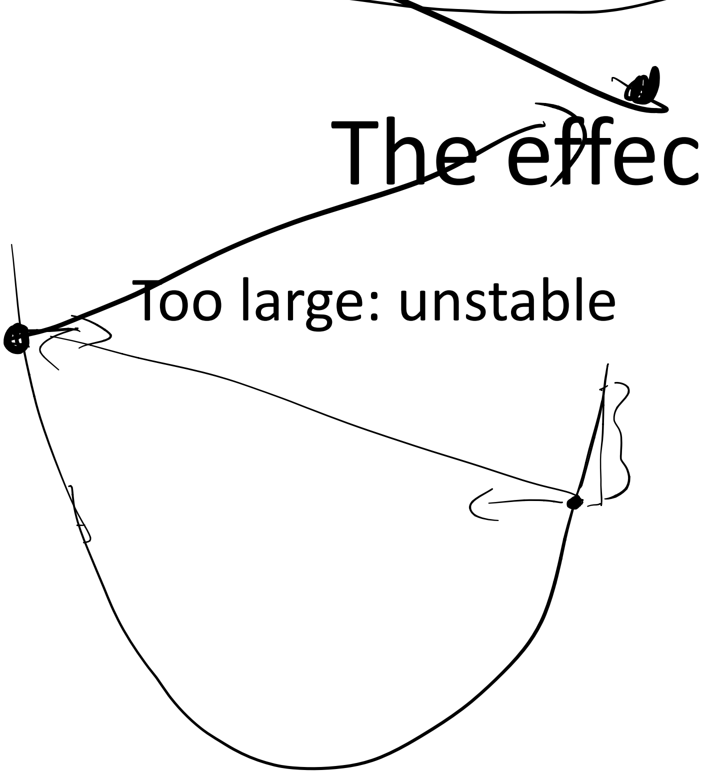
step size

gradient of loss wrt weight

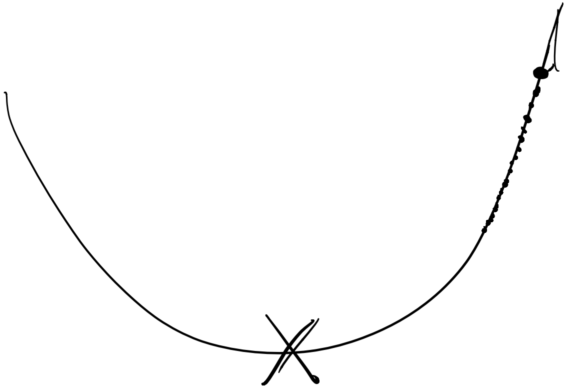
$$\boxed{w \leftarrow w - \alpha \frac{\partial L}{\partial w}}$$

The effect of Step Size

Too large: unstable

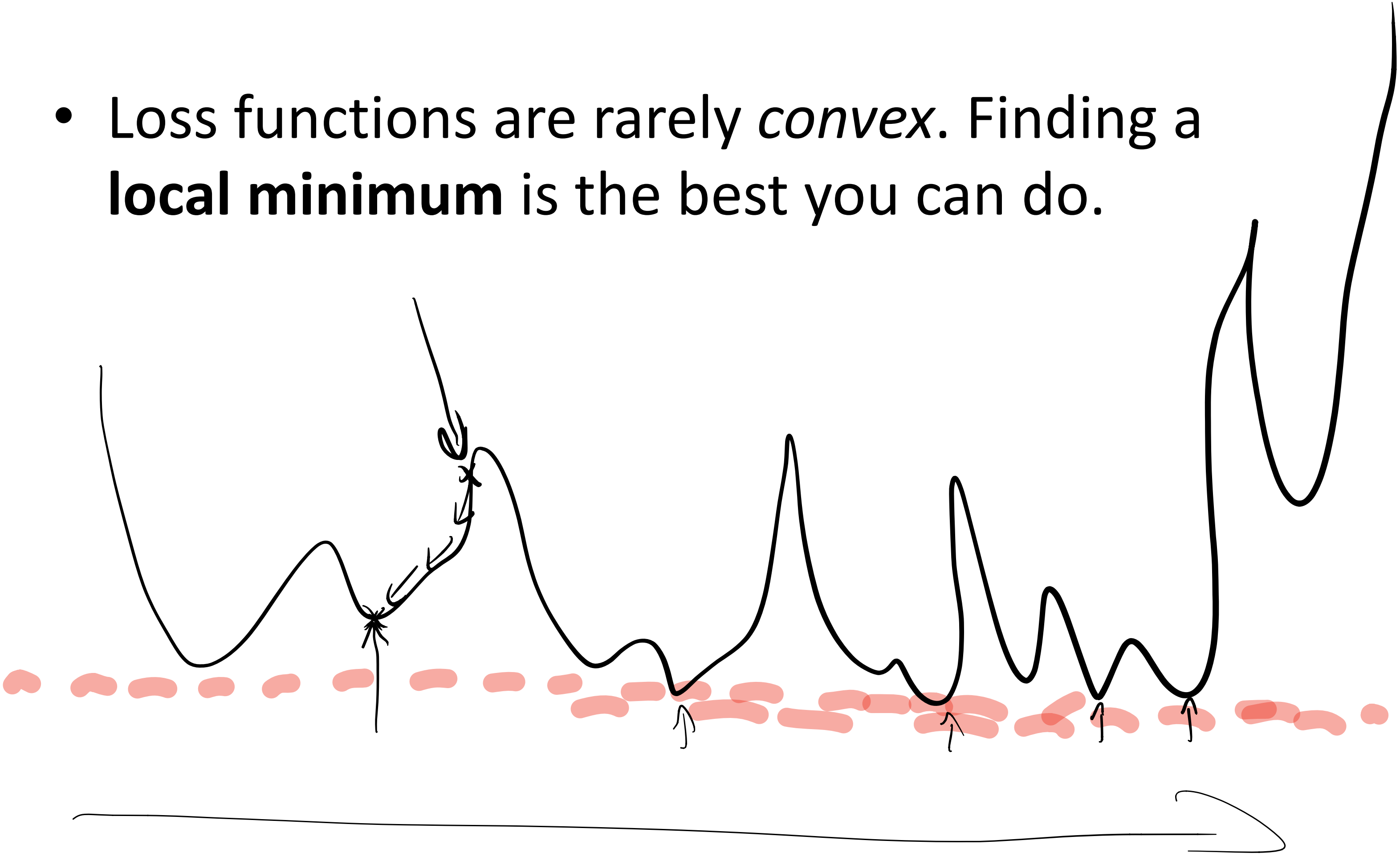


Too small: slow convergence

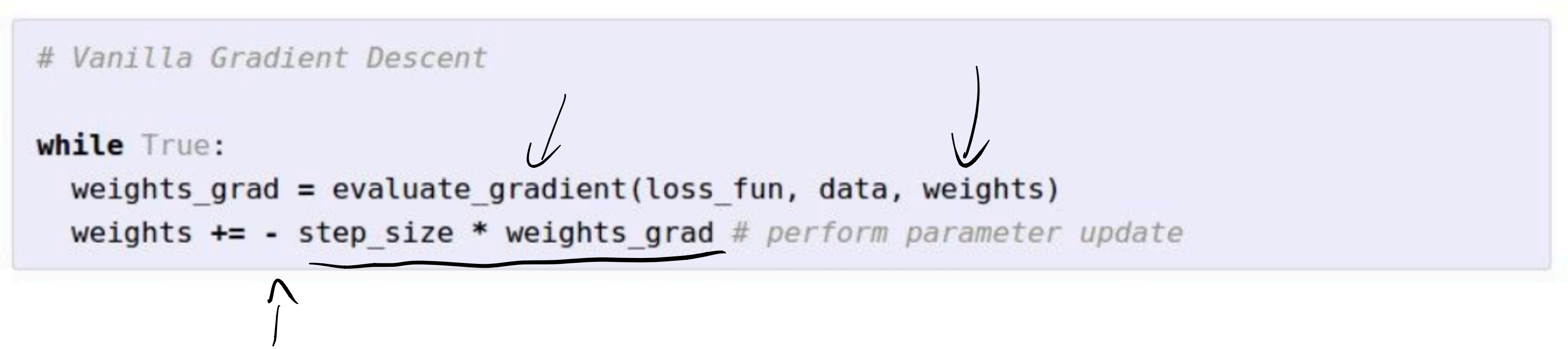


Reality isn't quite so pretty

- Loss functions are rarely *convex*. Finding a **local minimum** is the best you can do.



Gradient Descent

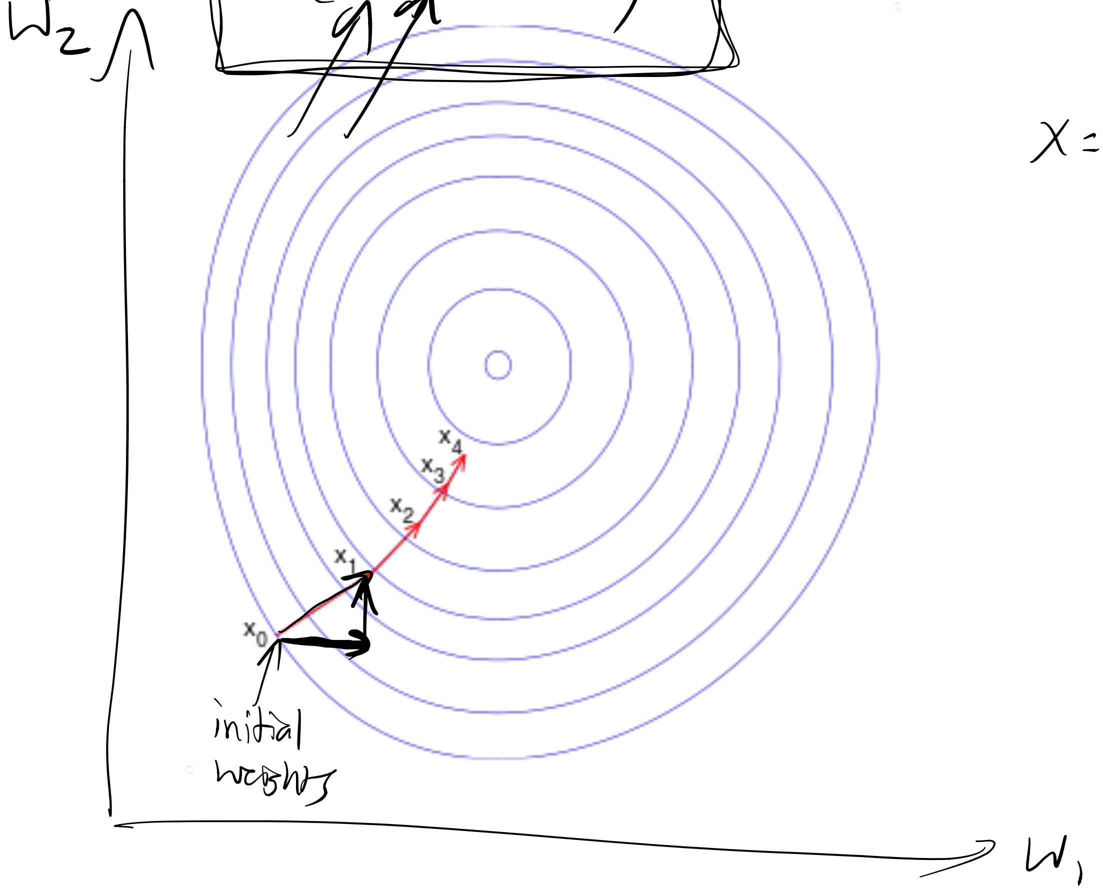


```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Gradient Descent: Intuition

$\boxed{L(X, Y, w_1, w_2)}$

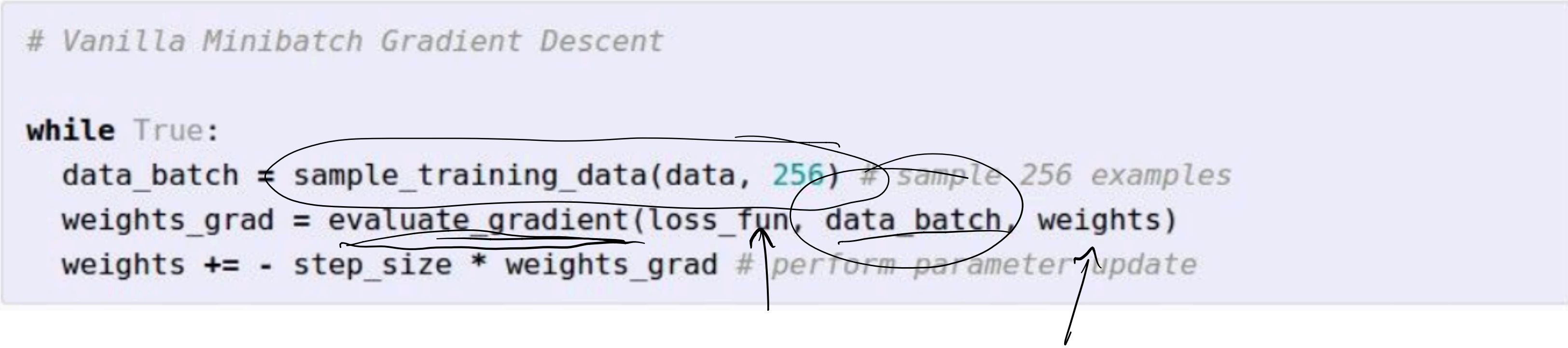


$X = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$

Gradient Descent: Demo

- http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/
  - select “Softmax” radio button at the bottom

Stochastic Gradient Descent



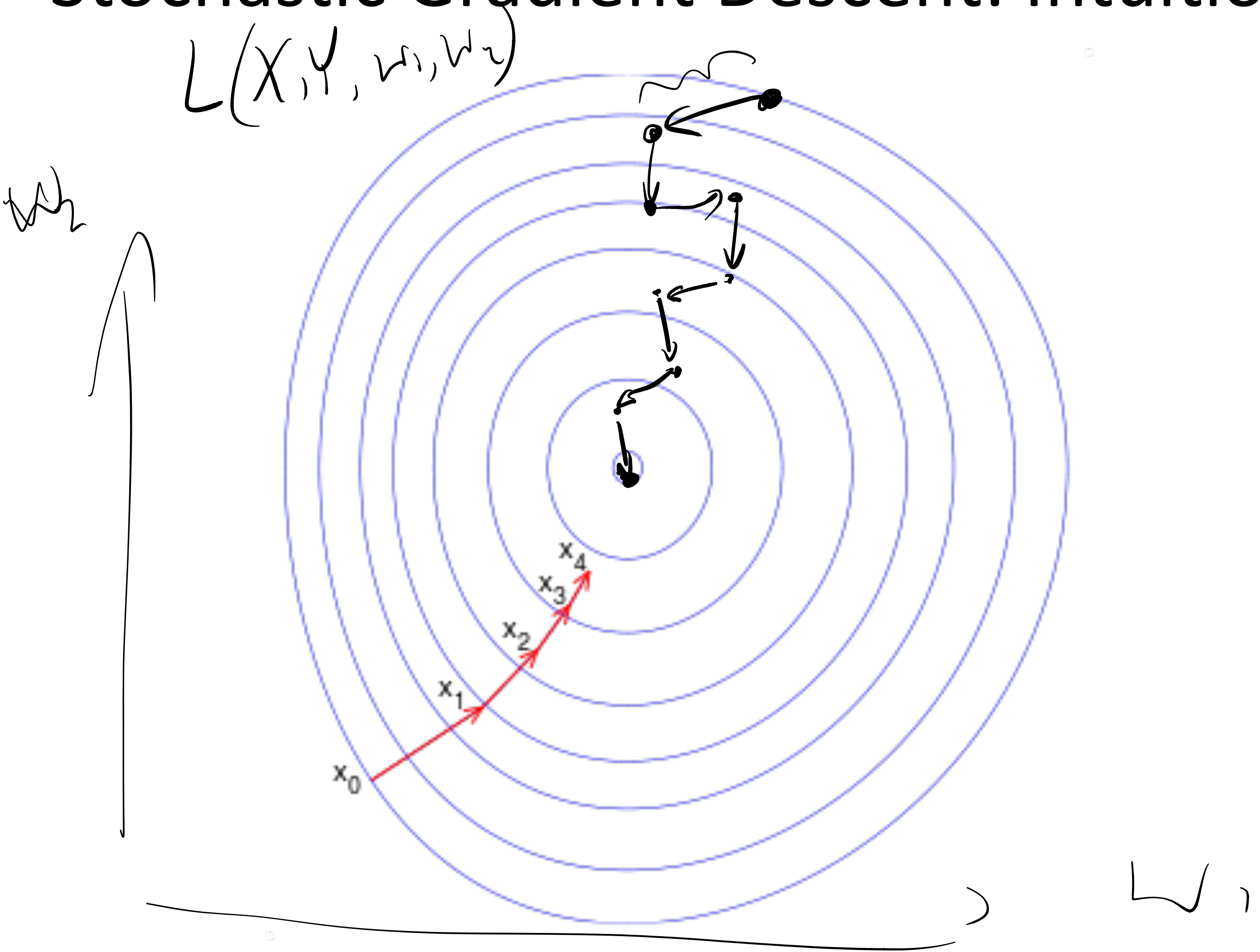
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

- L(X, Y; W) depends on
  - All data points $x_1..x_n$
  - Ground truth labels $y_1..y_n$
  - Weights W
- Very expensive to evaluate if you have a lot of data.

Stochastic Gradient Descent

- Idea: consider only a few data points at a time.
- Loss is now computed using only a small batch (minibatch) of data points.
- Update weights the same way using the gradient of L wrt the weights.

Stochastic Gradient Descent: Intuition



Taking stock

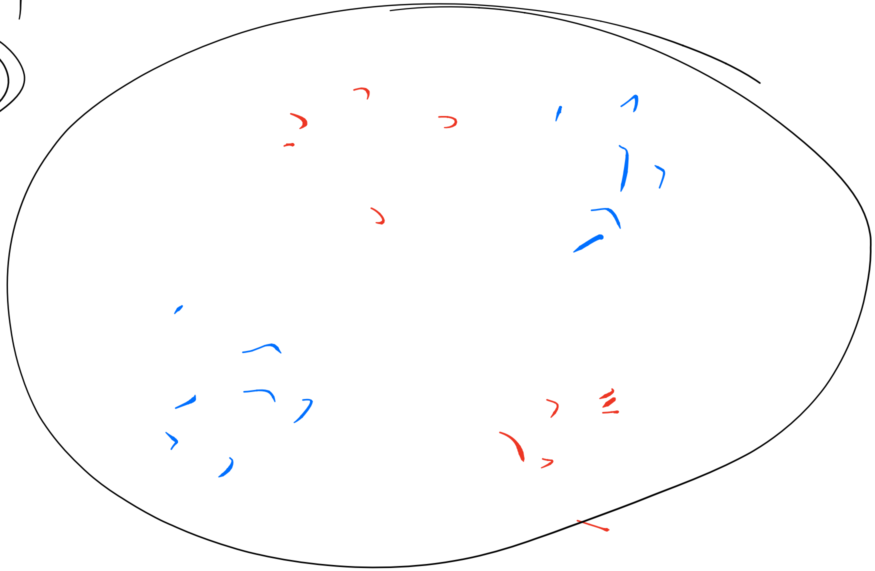
- We have:
  - ϕ = unravel(rgb2gray(img)), a feature extractor
  - h(x) = W^T x, a multiclass linear classifier
  - ∂ L = $\sum_{i=0}^{N} L_i$, a loss function

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

  - A way too adjust W until we can't make L any smaller.

So about that linearly separable assumption...

- Ideas:
  - ϕ = unravel(rgb2gray(img)), a feature extractor
    - use a fancier ϕ?
  - Learn ϕ too.



Neural Networks

Neural Network

Linear
classifiers

Neural networks: without the brain stuff

(**Before**) Linear score function: $f = Wx$

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Neural networks: without the brain stuff

(**Before**) Linear score function: $f = Wx$

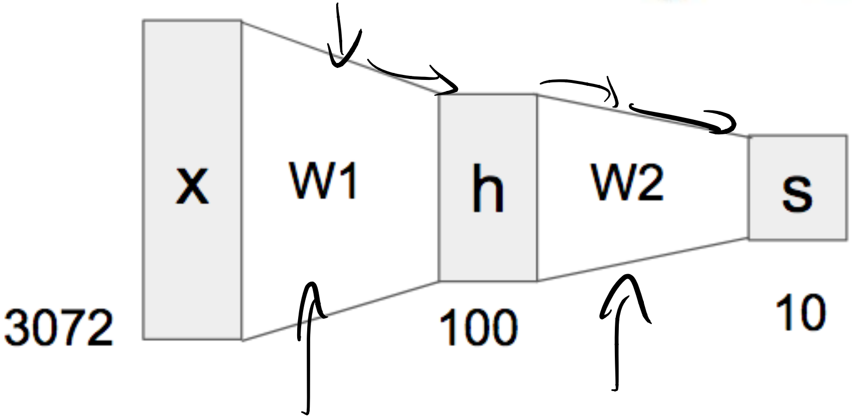
(**Now**) 2-layer Neural Network $f = W_2 \underline{\max(0,} W_1 x)$

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Neural networks: without the brain stuff

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Neural networks: without the brain stuff

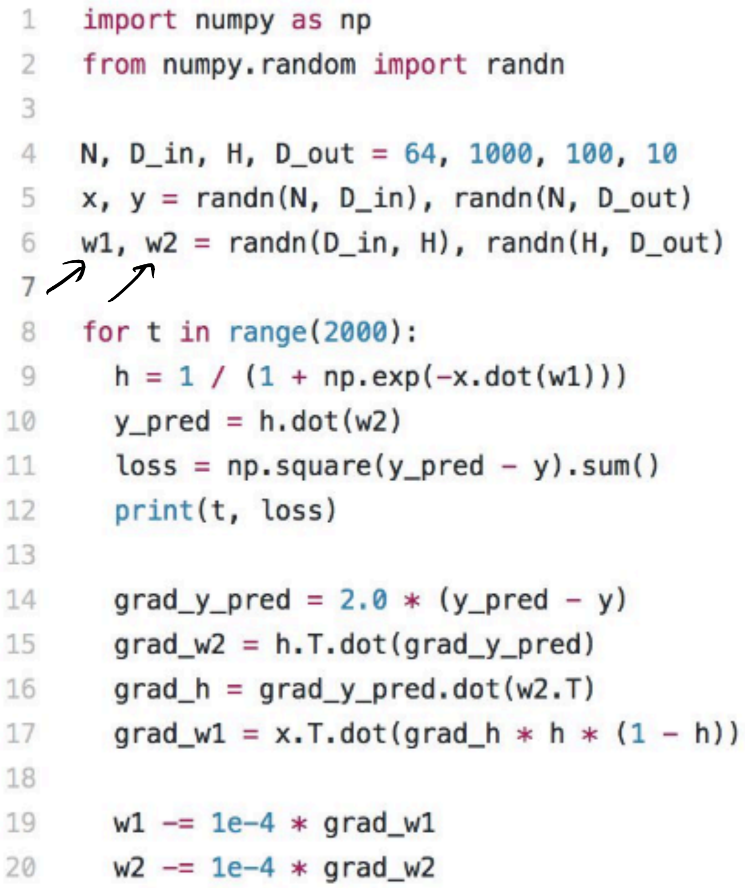
(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

$$f = W_3 \underline{\max(0,} W_2 \max(0, W_1 x))$$

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

Training a 2 layer neural network in 20 lines of python

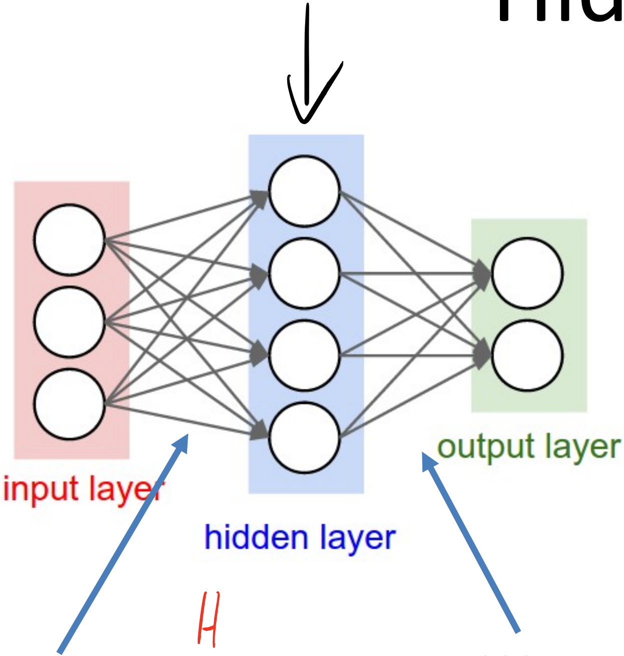


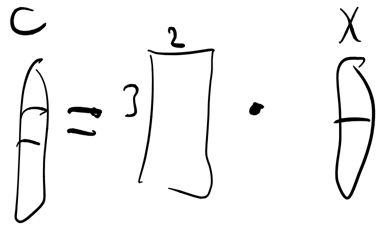
```
import numpy as np
from numpy.random import randn

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
```

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

"Hidden Layers"

H=4





$$\overset{C\times 1}{h} \leftarrow \overset{C\times H}{W_2}\ \overset{H\times d}{W_1}\ \overset{d\times 1}{x}$$

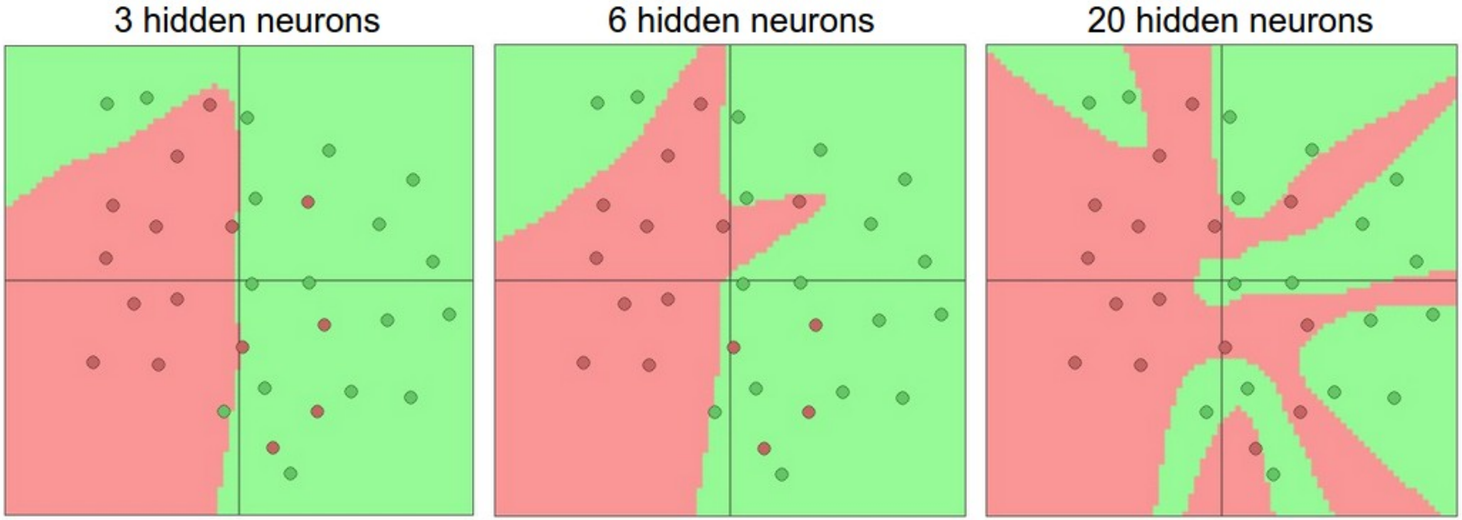
pick this!

W_1, a 3x$\overset{H}{4}$ matrix converts input into hidden layer activations

W_2, a $\overset{H}{4}$x2 matrix transforms hidden layer activations to output scores

Figures: Fei-Fei Li, Justin Johnson, & Serena Yeung

Neural Networks: Nonlinear Classifiers built from Linear Classifiers



Figures: Fei-Fei Li, Justin Johnson, & Serena Yeung

Neural networks: without the brain stuff

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

or 3-layer Neural Network

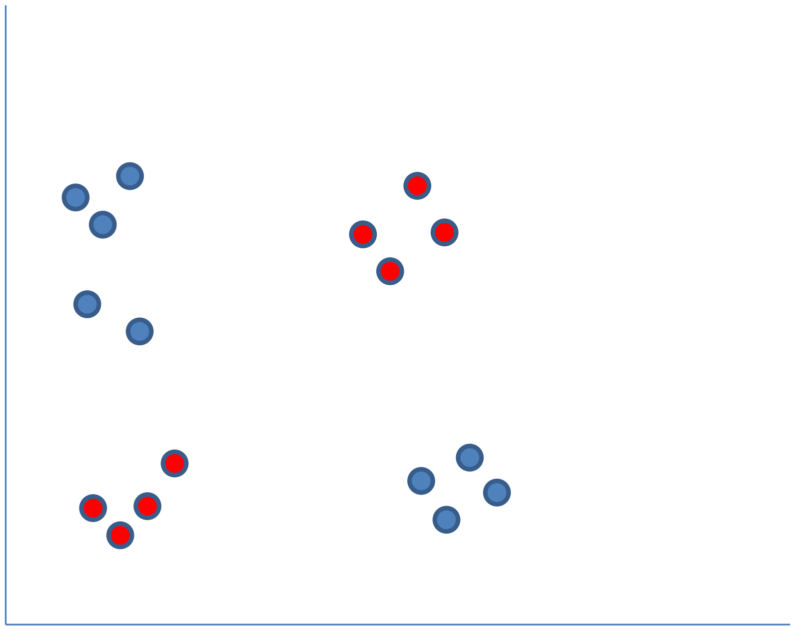
$$f = W_3 \boxed{\max(0,} W_2 \boxed{\max(0,} W_1 x))$$

???

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung

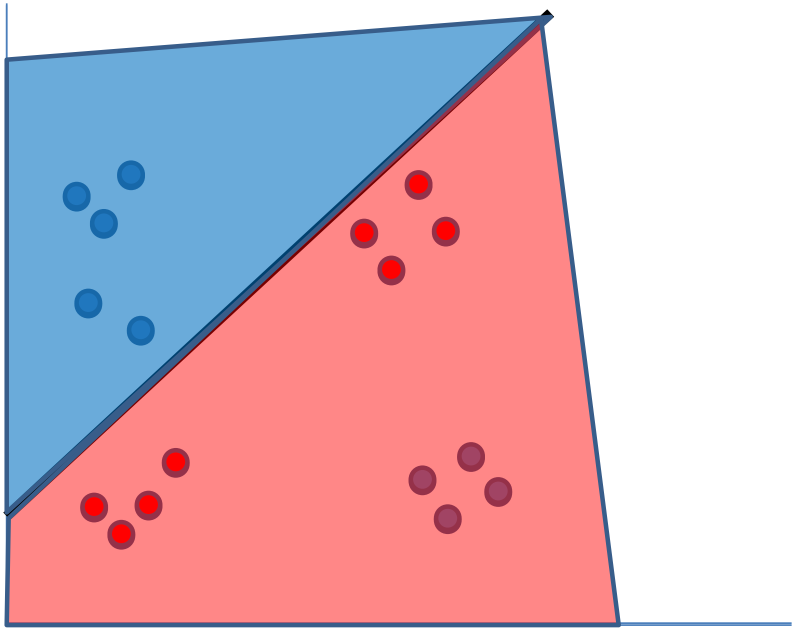
Activation Functions

$$f(x, W) = Wx$$



Activation Functions

$$f(x, W) = Wx$$

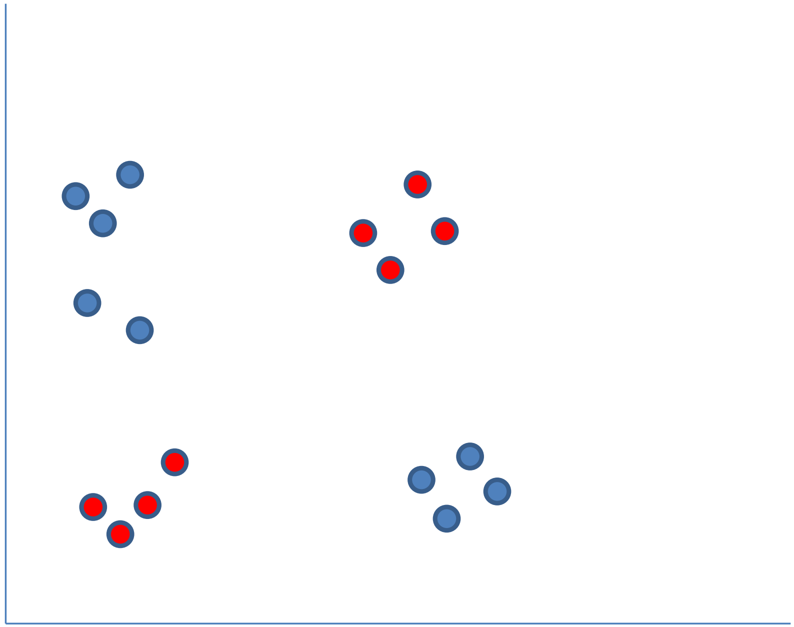


A linear classifier can only do so well…

Activation Functions

$$f(x, W) = Wx$$

$$f(x, W_1, W_2) = W_1(W_2x)$$



Let’s try stacking two linear classifiers together

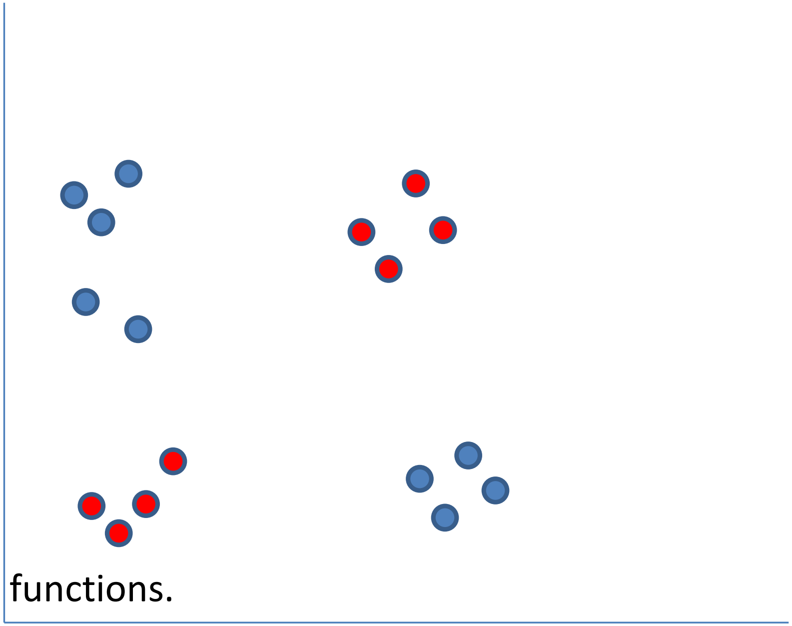
Activation Functions

$$f(x, W) = Wx$$

$$f(x, W_1, W_2) = W_1(W_2x)$$

$$W \leftarrow W_1W_2$$

$$f(x, W) = Wx$$

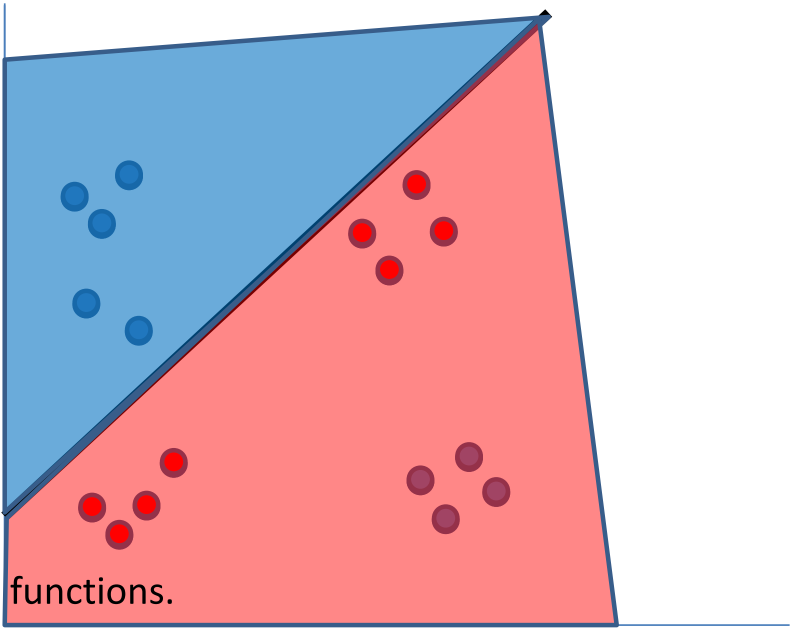


Uh oh – linear functions compose to linear functions.

Activation Functions

$$f(x, W) = Wx$$

$$f(x, W_1, W_2) = W_1(W_2 x)$$



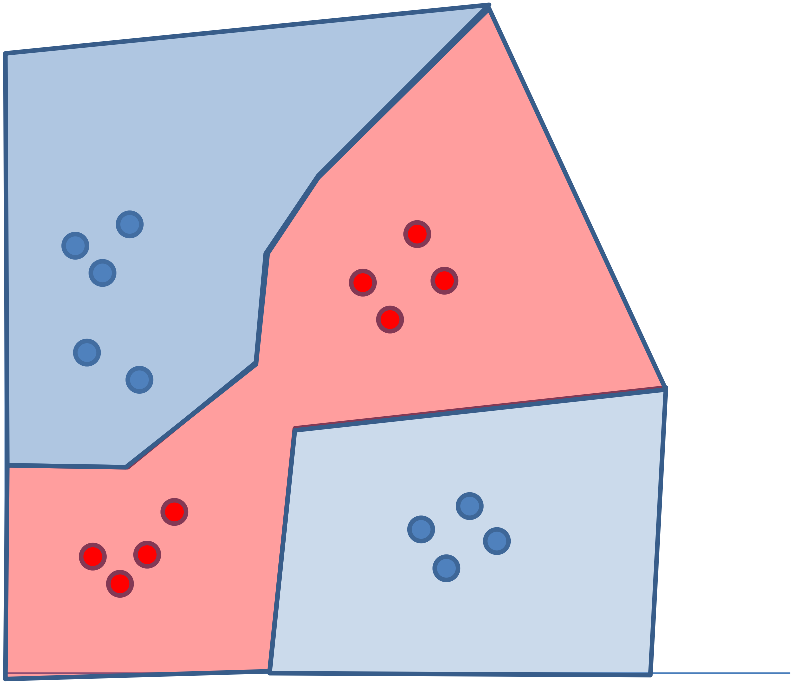
$$W \leftarrow W_1 W_2$$

$$f(x, W) = Wx$$

Uh oh – linear functions compose to linear functions.

Activation Functions

$$f(x, W_1, W_2, W_3) = W_3 \max(0, W_2 \max(0, W_1 x))$$



Nonlinearities prevent the composed linear functions from collapsing into a single one.

Neural Networks

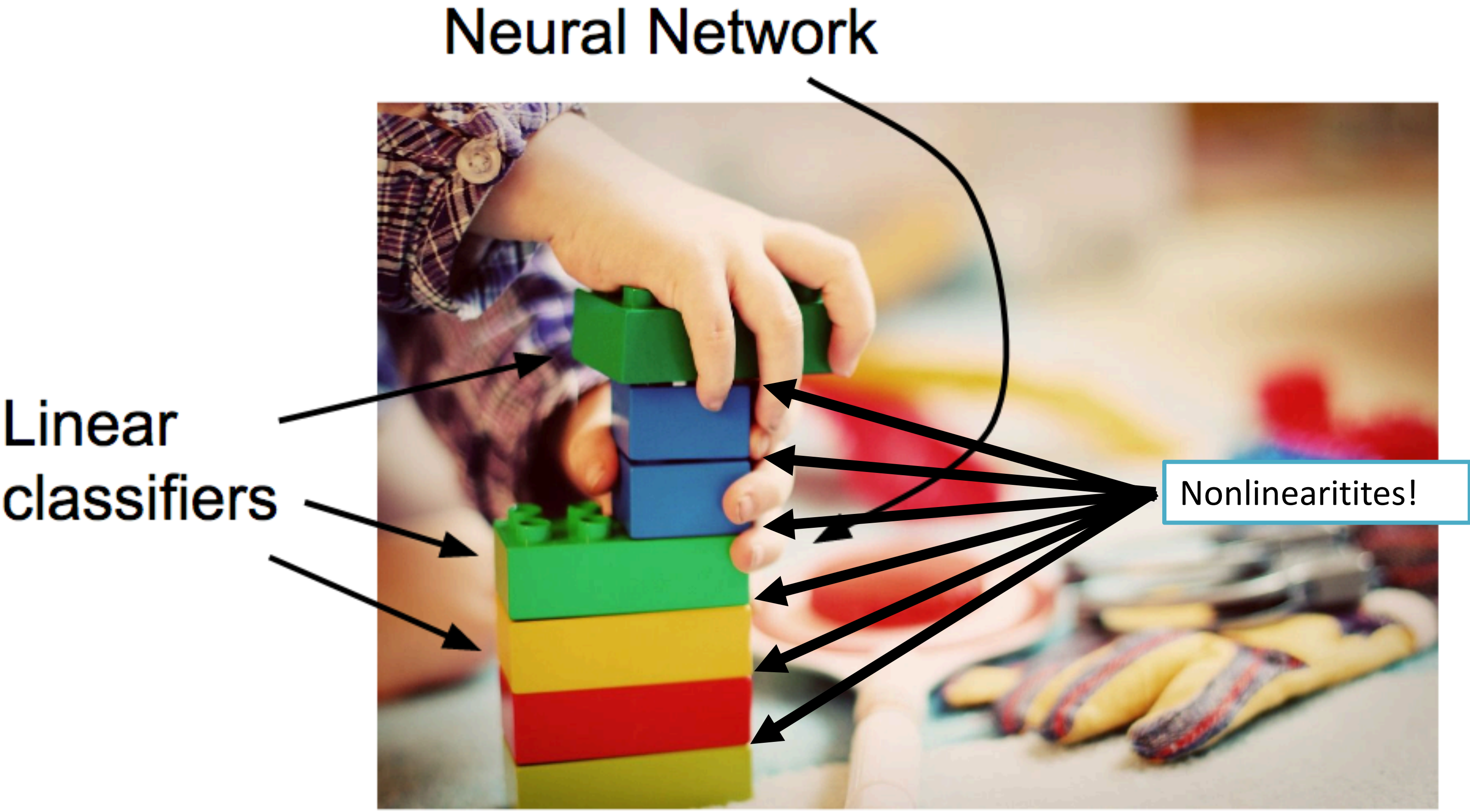
Neural Network

Linear classifiers

This image is CC0 1.0 public domain

Slide: Fei-Fei Li, Justin Johnson, & Serena Young

Neural Networks



Neural Network

Linear classifiers

Nonlinearitites!

Slide: Fei-Fei Li, Justin Johnson, & Serena Young

Neural Networks: Nonlinear Classifiers built from Linear Classifiers

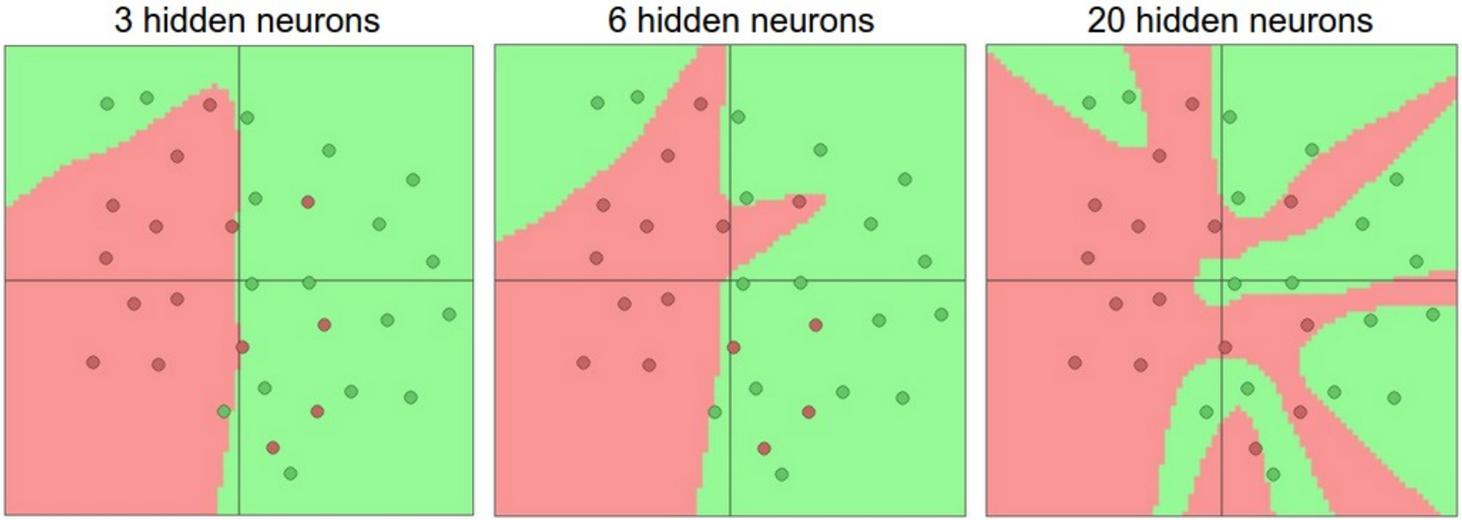
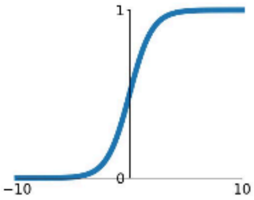


Figure: Fei-Fei Li, Justin Johnson, & Serena Yeung

Activation Functions

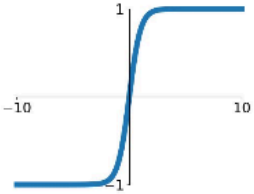
Sigmoid

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



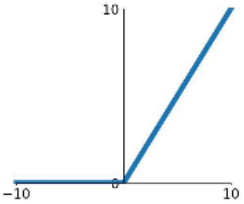
tanh

$\tanh(x)$



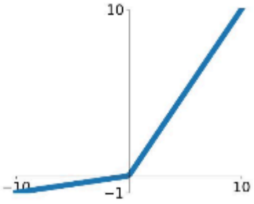
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

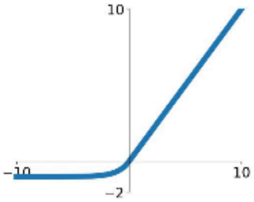


Maxout

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

ELU

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



Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung