CS 4803 / 7643: Deep Learning

Topics:

- Regularization
- Neural Networks
- Optimization
- Computing Gradients

Dhruv Batra

Georgia Tech

Recap from last time

Parametric Approach: Linear Classifier

$$\underbrace{\boxed{f(x,W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\overbrace{\boxed{x}}^{3072\text{x}1} + \underbrace{\boxed{b}}_{10\text{x}1}$$

Image

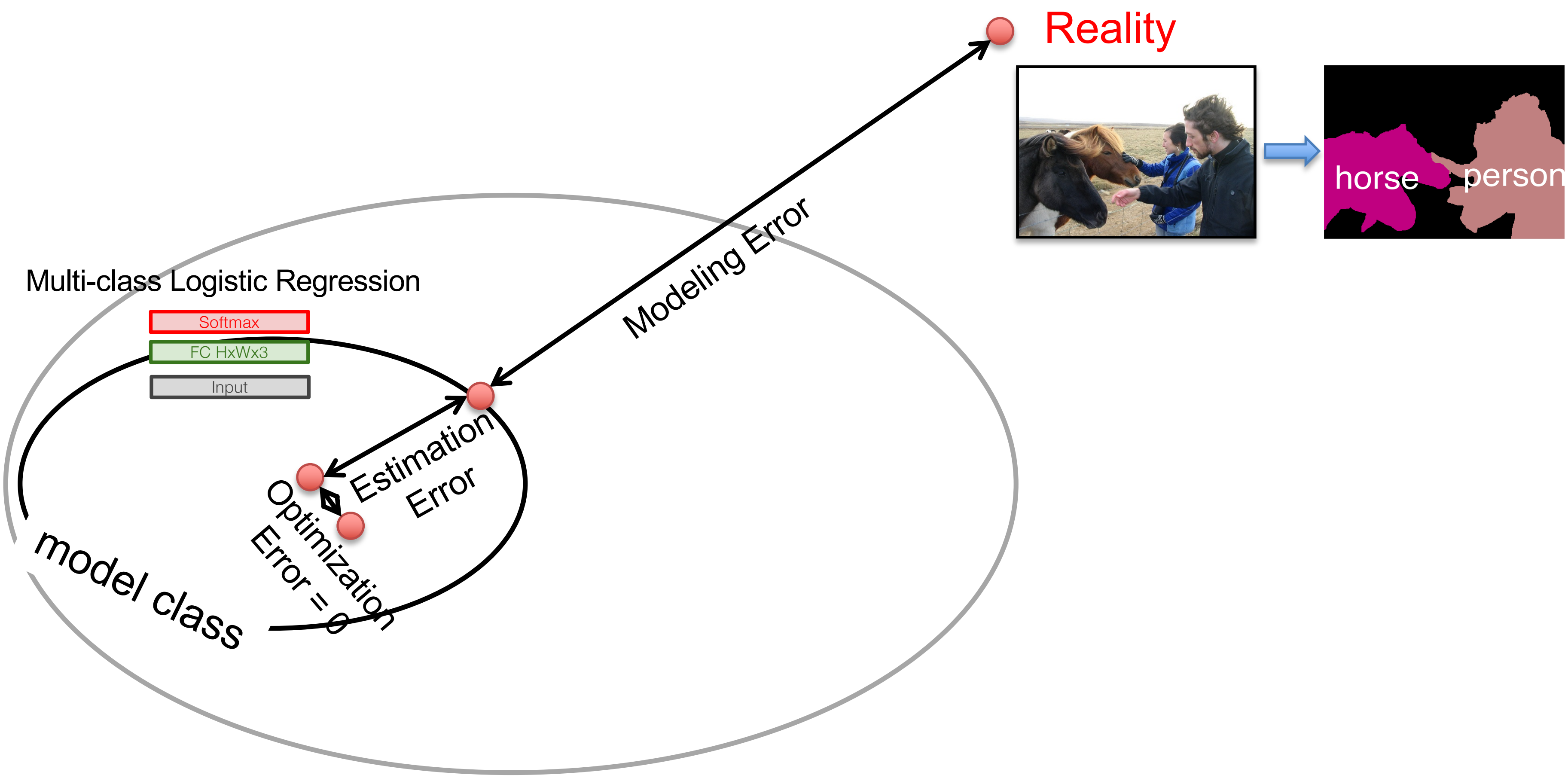
Array of **32x32x3** numbers
(3072 numbers total)

$\longrightarrow$ f(**x**,**W**) $\longrightarrow$ **10** numbers giving class scores

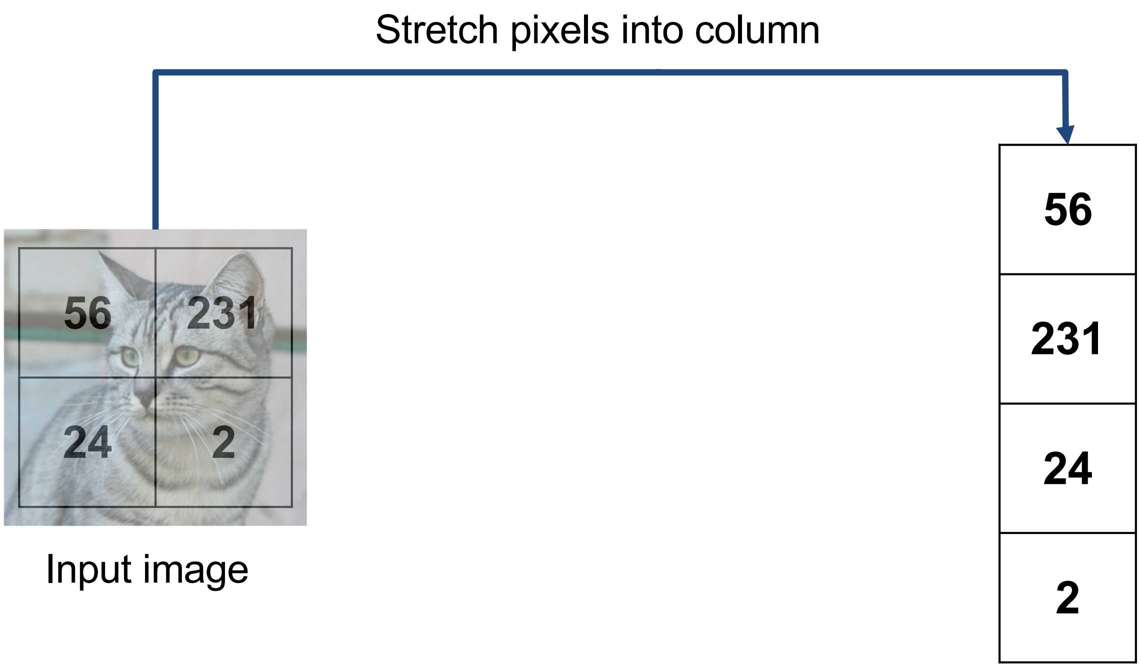
W
parameters
or weights

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Error Decomposition



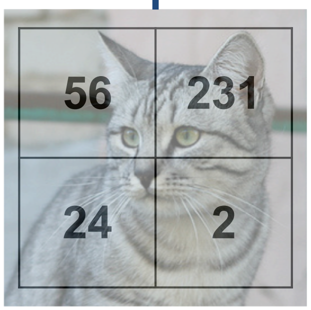
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Stretch pixels into column



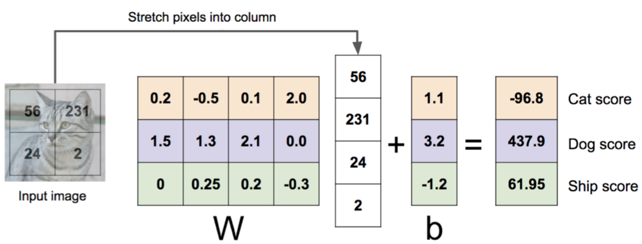
Input image

$$
\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W}
\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}
+
\underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b}
=
\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}
\begin{array}{l} \text{Cat score} \\ \text{Dog score} \\ \text{Ship score} \end{array}
$$

Linear Classifier: Three Viewpoints

Algebraic Viewpoint

$f(x,W) = Wx$

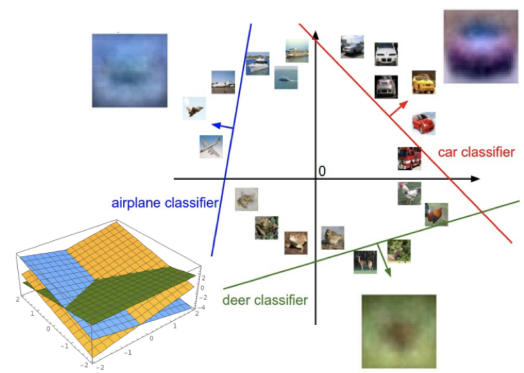


Visual Viewpoint

One template
per class

Geometric Viewpoint

Hyperplanes
cutting up space



Recall from last time: Linear Classifier

airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

1. Come up with a way of efficiently finding the parameters that minimize the loss function. **(optimization)**

Softmax vs. SVM

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

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

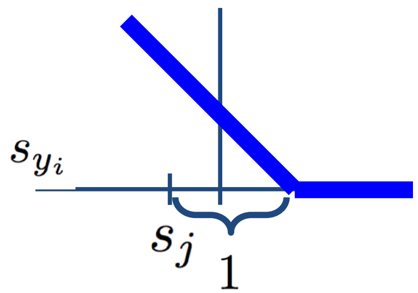
Suppose: 3 training examples, 3 classes.
With some W the scores $f(x,W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:

“Hinge loss”



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

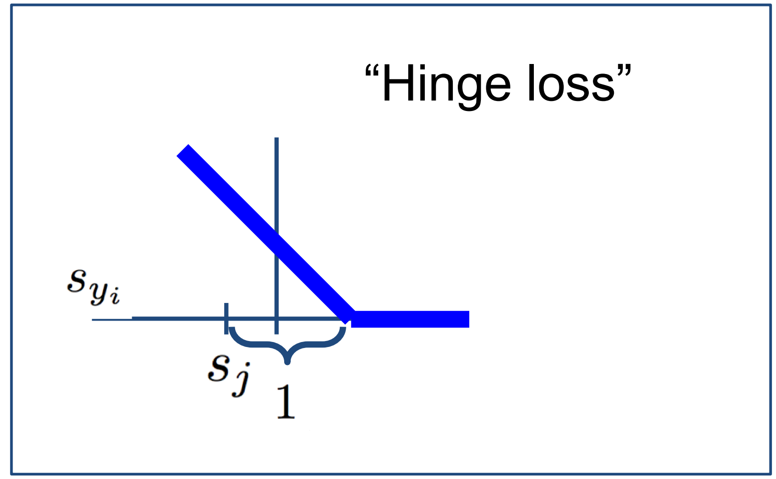
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:



$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Softmax vs. SVM

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

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

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$ Softmax Function

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Softmax Function

Probabilities must be >= 0

cat	**3.2**		**24.5**
car	5.1	$\xrightarrow{\text{exp}}$	164.0
frog	-1.7		0.18

unnormalized probabilities

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

	scores	exp → Probabilities must be >= 0 (unnormalized probabilities)	normalize → Probabilities must sum to 1 (probabilities)
cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$ Softmax Function

	Unnormalized log-probabilities / logits	$\xrightarrow{\text{exp}}$	unnormalized probabilities (Probabilities must be >= 0)	$\xrightarrow{\text{normalize}}$	probabilities (Probabilities must sum to 1)
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$ Softmax Function

$$L_i = -\log P(Y = y_i | X = x_i)$$

	Unnormalized log-probabilities / logits	exp →	unnormalized probabilities (Probabilities must be >= 0)	normalize →	probabilities (Probabilities must sum to 1)
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

→ $L_i = -\log(0.13)$
$= \mathbf{2.04}$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i | X = x_i)$$

	Unnormalized log-probabilities / logits	unnormalized probabilities	probabilities
cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

exp → normalize →

→ $L_i = -\log(0.13)$ $= \mathbf{2.04}$

Maximum Likelihood Estimation
Choose probabilities to maximize the likelihood of the observed data

Log-Likelihood / KL-Divergence / Cross-Entropy

$D = \{(x_i, y_i)\}$ IID $\sim P^*$

$$\boxed{\hat{w}_{MLE}} = \max P(D \mid w)$$

$$= \max \log P(D \mid w)$$

$$= \max \boxed{\sum_i} \log P(y_i \mid x_i, w)$$

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

$$L_i = -\log P(Y = y_i | X = x_i)$$

Probabilities must be >= 0

Probabilities must sum to 1

	Unnormalized log-probabilities / logits		unnormalized probabilities		probabilities		Correct probs
cat	**3.2**	exp →	**24.5**	normalize →	**0.13**	→ compare ←	**1.00**
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$\boxed{s = f(x_i; W)}$$

$$\boxed{P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i|X = x_i)$$

	Unnormalized log-probabilities / logits	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities	$\rightarrow$ compare $\leftarrow$	Correct probs
cat	**3.2**		**24.5**		**0.13**		**1.00**
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Kullback–Leibler divergence

$$D_{KL}(P\|Q) = \sum_y P(y) \log \frac{P(y)}{Q(y)}$$

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

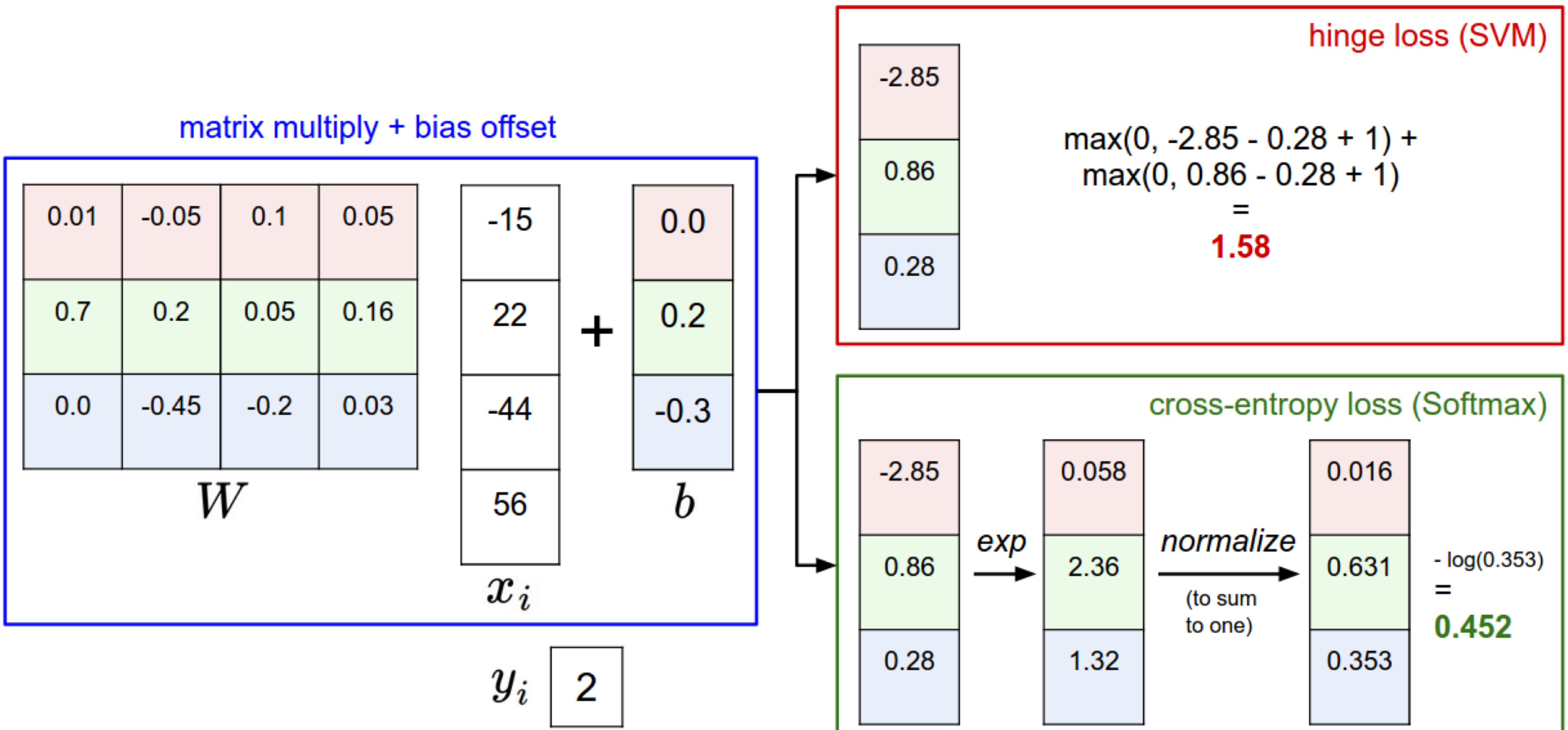
$$L_i = -\log P(Y = y_i | X = x_i)$$

	Unnormalized log-probabilities / logits	$\xrightarrow{\text{exp}}$ unnormalized probabilities	$\xrightarrow{\text{normalize}}$ probabilities		Correct probs
cat	**3.2**	**24.5**	**0.13**	→ compare ←	**1.00**
car	5.1	164.0	0.87		0.00
frog	-1.7	0.18	0.00		0.00

Cross Entropy

$$H(\underline{P}, \underline{Q}) = H(p) + D_{KL}(P \| Q)$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Plan for Today

- Regularization
- Neural Networks
- Optimization
- Computing Gradients

Regularization

$$\underbrace{L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{}$$

Data loss: Model predictions should match training data

Regularization

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization

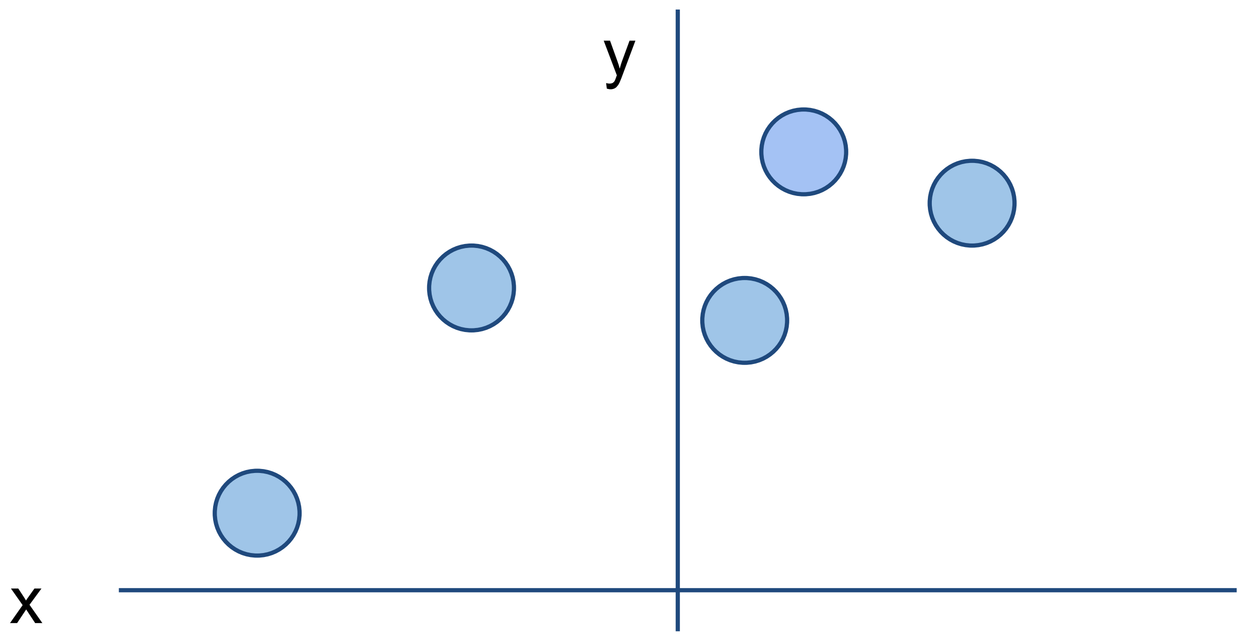
λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization: Prefer Simpler Models



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Polynomial Regression

$$\hat{y} = w_0 + w_1 x$$

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \cdots + w_d x^d$$

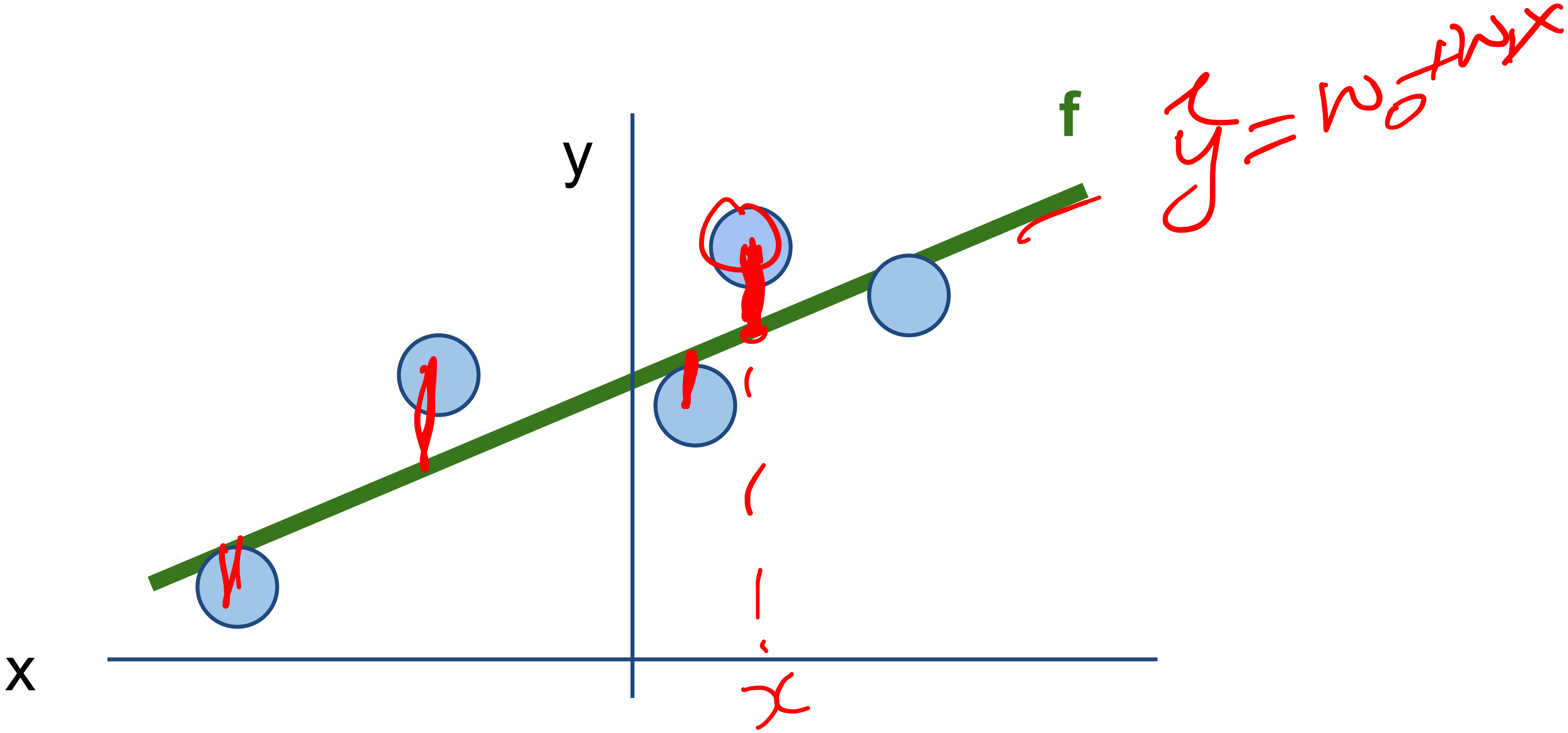
$$= [w_0 \; \cdots \; w_d] \underbrace{\begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^d \end{bmatrix}}_{} = \underline{\vec{w}}^T \underline{\phi(x)}$$

$$D = \{(x_i, y_i)\}$$

$$w^* = \min_w \frac{1}{2} \sum (y_i - \hat{y}_i)^2$$

Polynomial Regression

Polynomial Regression



Polynomial Regression

- Demo:
  - https://arachnoid.com/polysolve/

- Data:
  - 10 6
  - 15 9
  - 20 11
  - 25 12
  - 29 13
  - 40 11
  - 50 10
  - 60 9

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \underbrace{\lambda R(W)}_{}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout

Batch normalization

Stochastic depth, fractional pooling, etc

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Why regularize?

- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature

$$\hat{y} = \underbrace{w_0 + w_1 x + \cdots}_{} + \overset{0}{\boxed{w_d}} x^d$$

Recap

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

Softmax

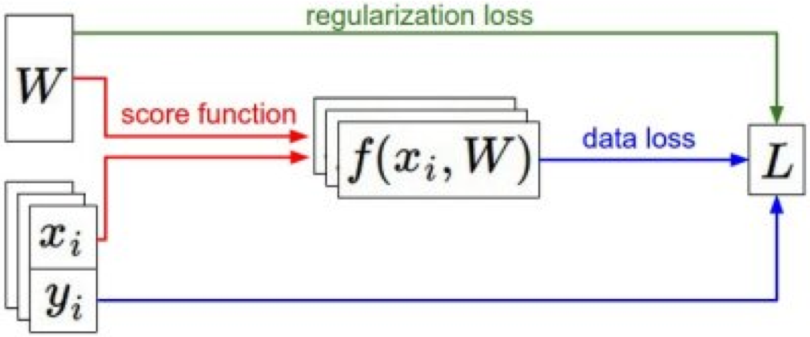
$$\boxed{L_i} = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

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

$$\boxed{L} = \underbrace{\frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \boxed{R(W)}}$$

Full loss



Recap

How do we find the best W?

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x; W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

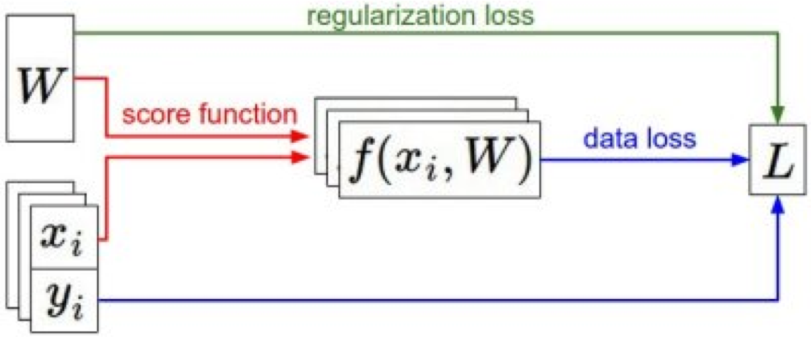
Softmax

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

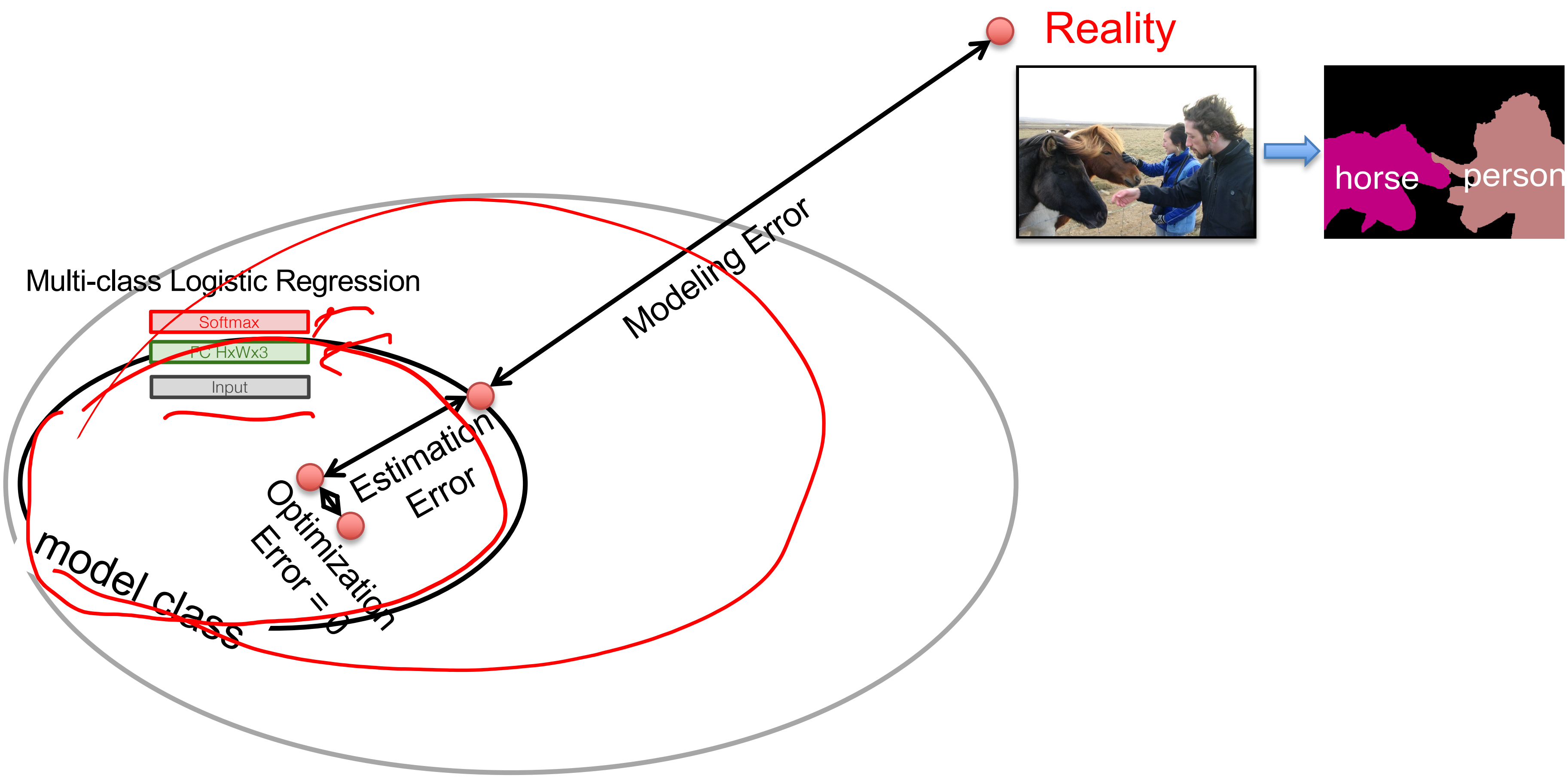
SVM

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

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$ Full loss



Error Decomposition



Next: Neural Networks

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Neural networks: without the brain stuff

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

$f = W_2 W_1 x$

Neural networks: without the brain stuff

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

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

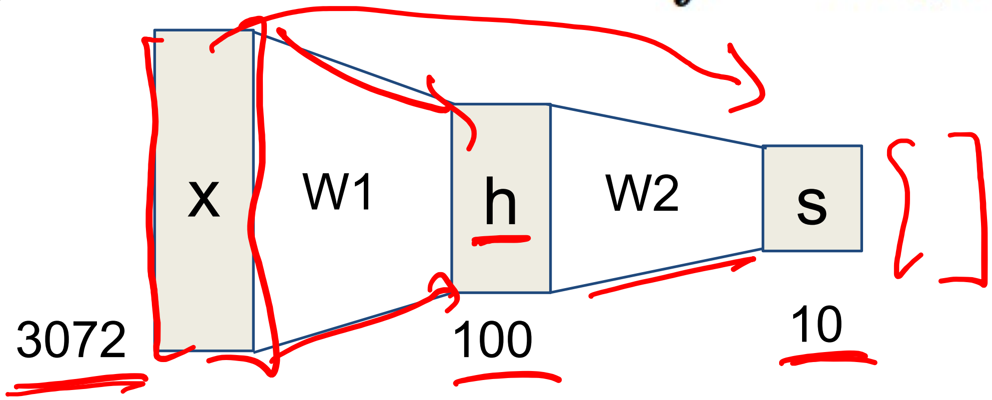
W_2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Neural networks: without the brain stuff

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

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



$\mathbb{R}^{100} \ni h = \left[\quad\right]_{100 \times 3072} \left[\quad\right]^{x}$

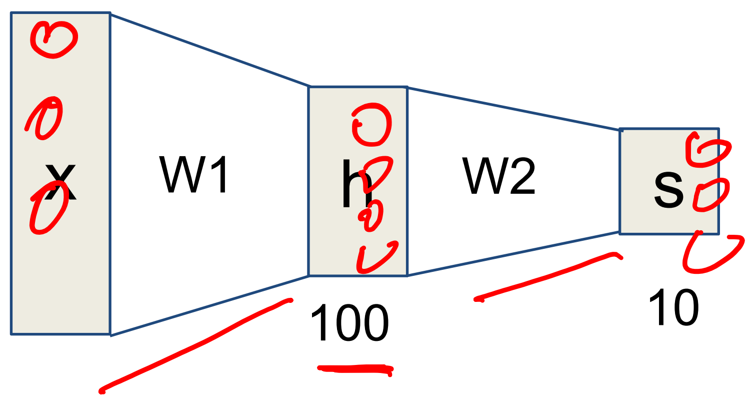
$s = W_2 \left[\quad\right]_{10 \times 100}$

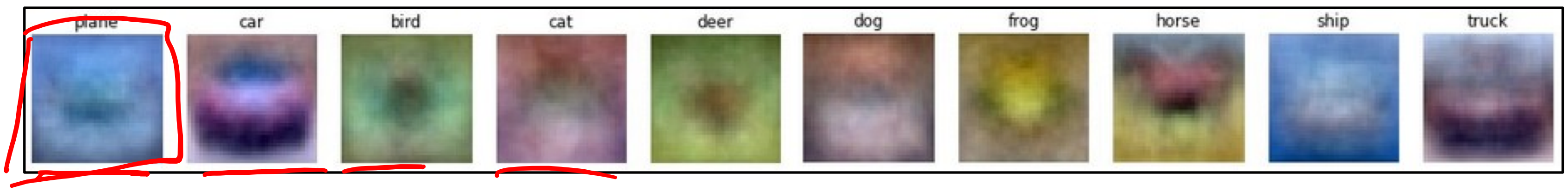
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

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 Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

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

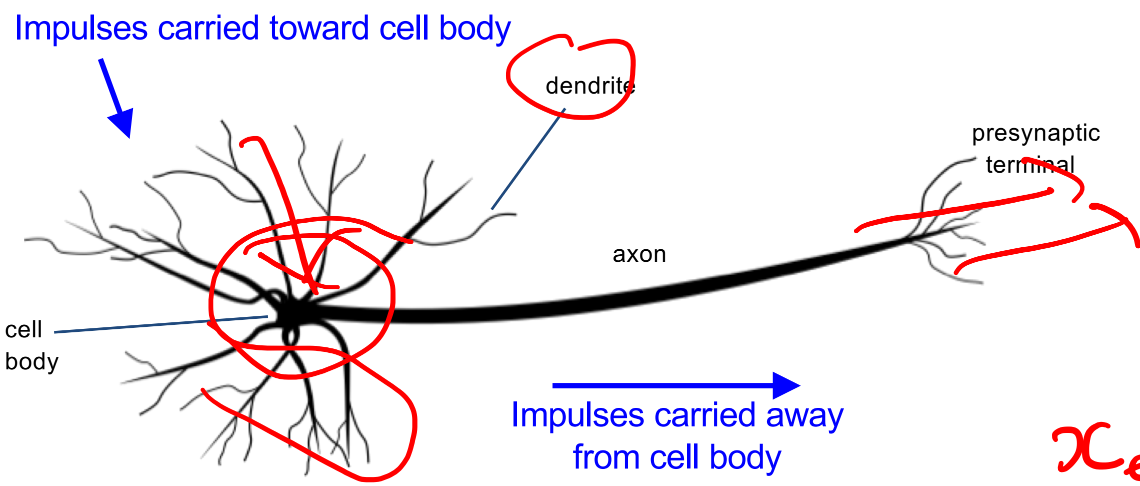
$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

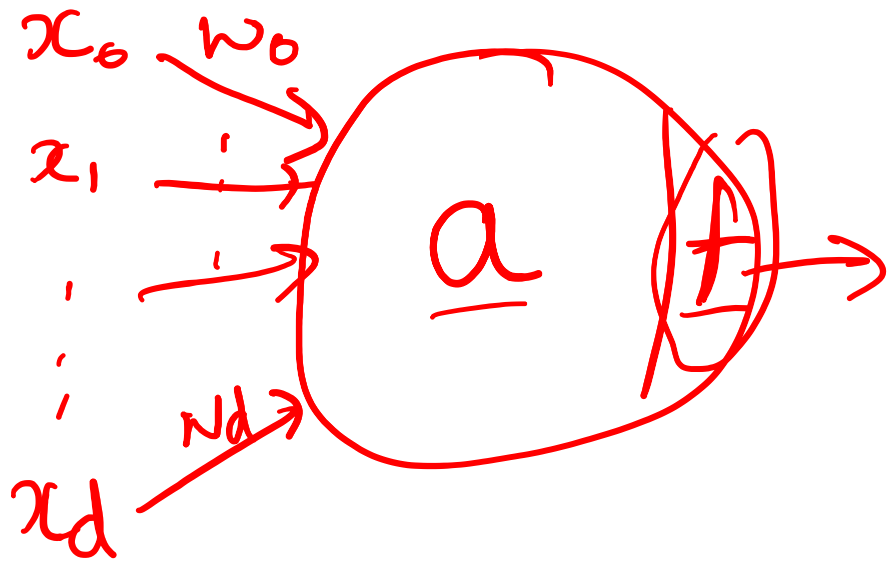
Full implementation of training a 2-layer Neural Network needs ~20 lines:

```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
```

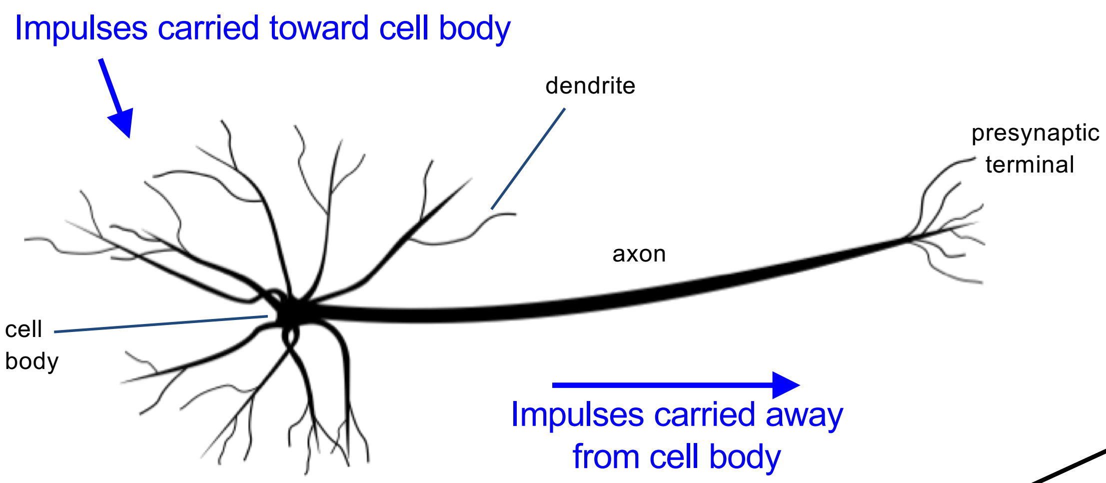
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

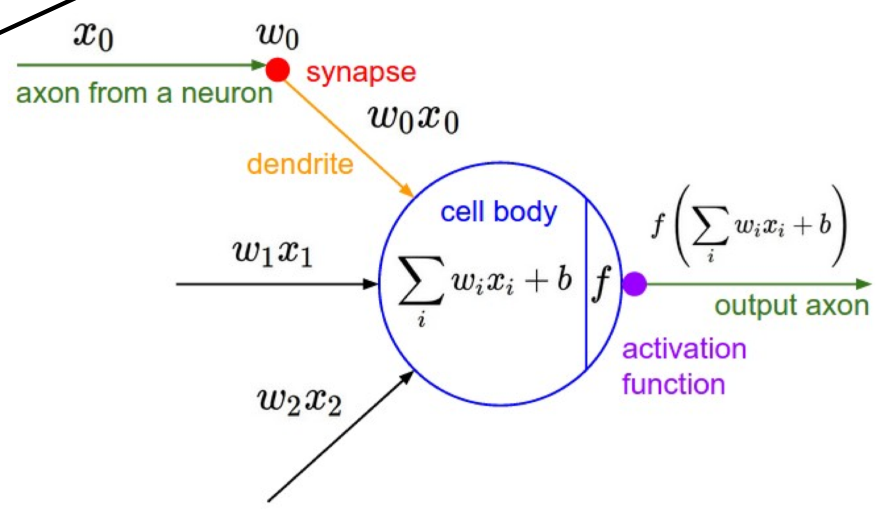
$$a = \sum_j w_j x_j = \vec{w}^T \vec{x}$$



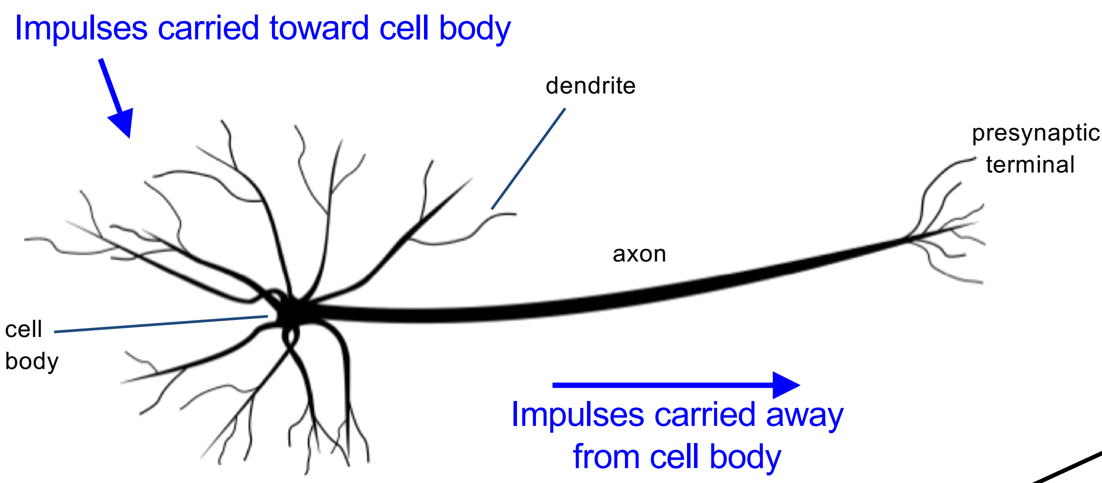


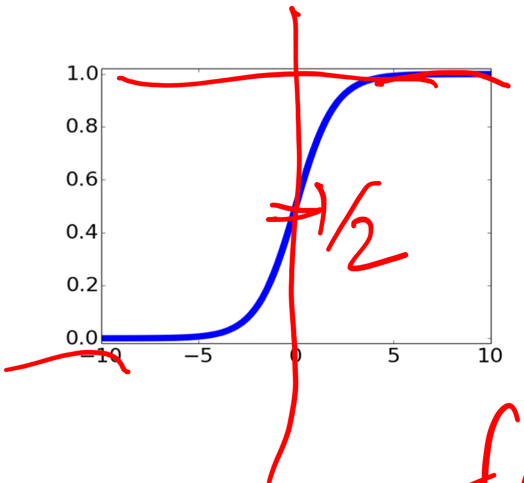
$$y = f(a)$$





Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

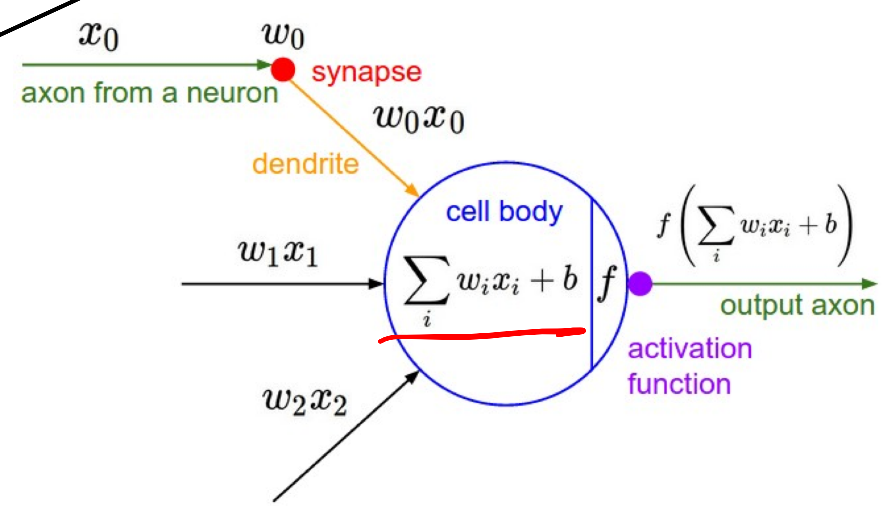




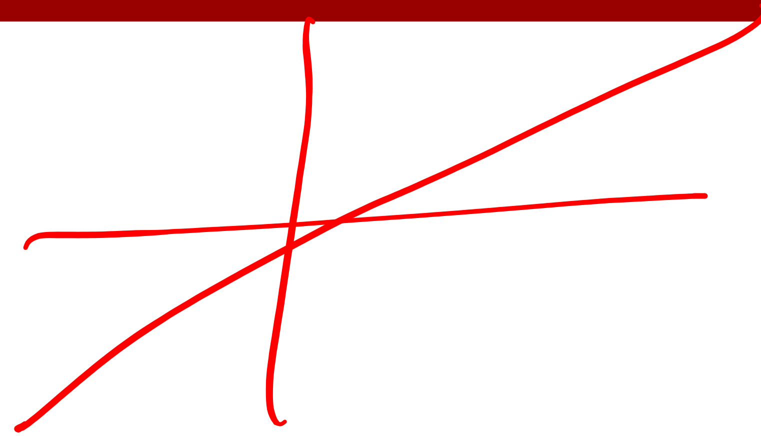
sigmoid activation function

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

$$f(a) \quad \frac{1}{1+e^{-a}}$$

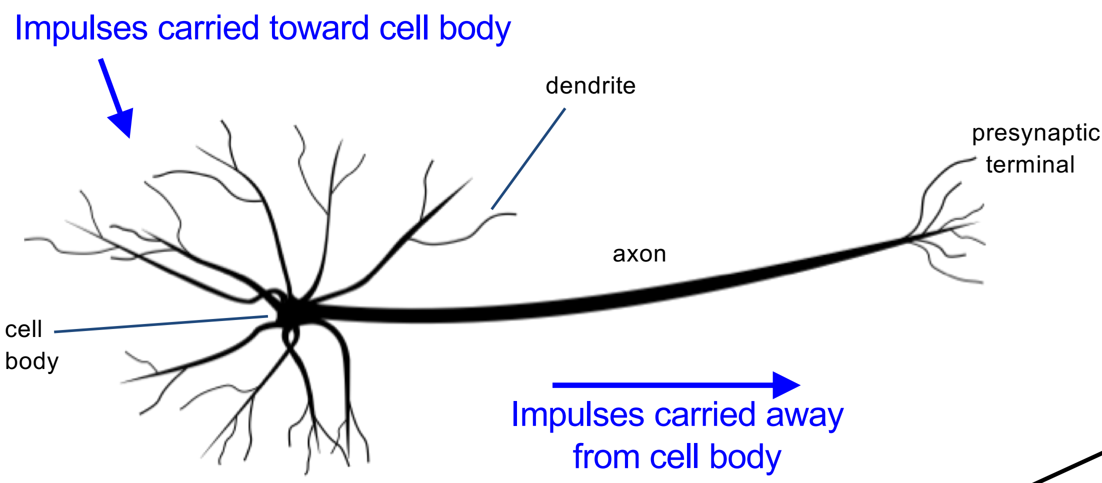


$$f(a) = a$$

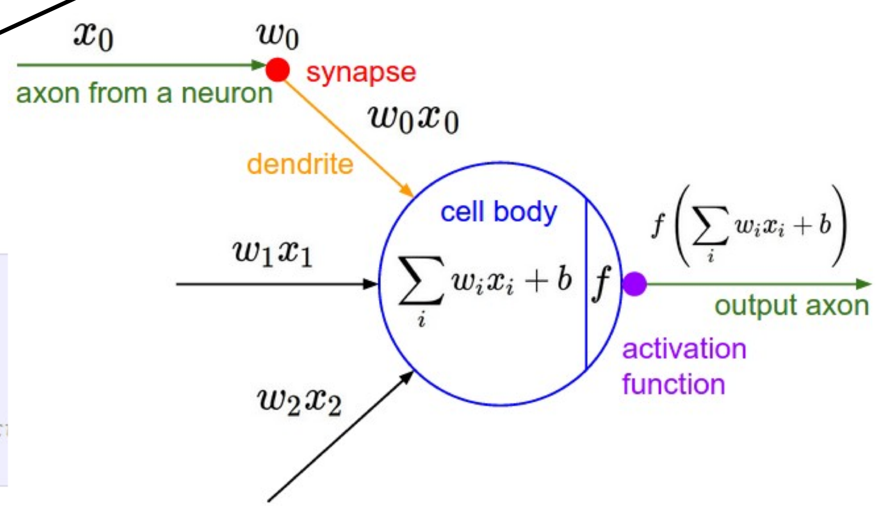


$$\cancel{f}\ y = \vec{w}^T \vec{x}$$

$$f(a) = \max(0, a)$$




```
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation func
    return firing_rate
```



Be very careful with your brain analogies!

Biological Neurons:

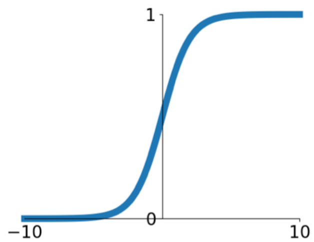
- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system
- Rate code may not be adequate

[Dendritic Computation. London and Hausser]

Activation functions

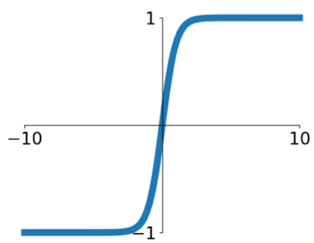
Sigmoid

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



tanh

$\tanh(x)$

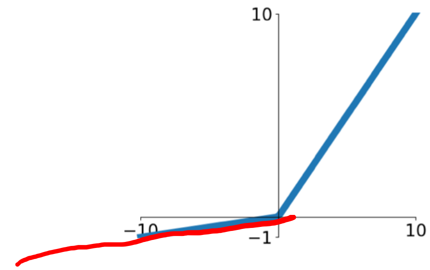


ReLU

$\max(0, x)$

Leaky ReLU

$\max(0.1x, x)$

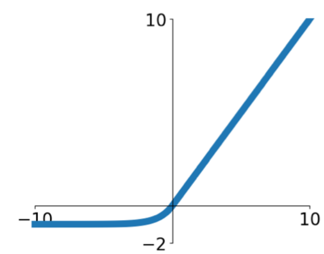


Maxout

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

ELU

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



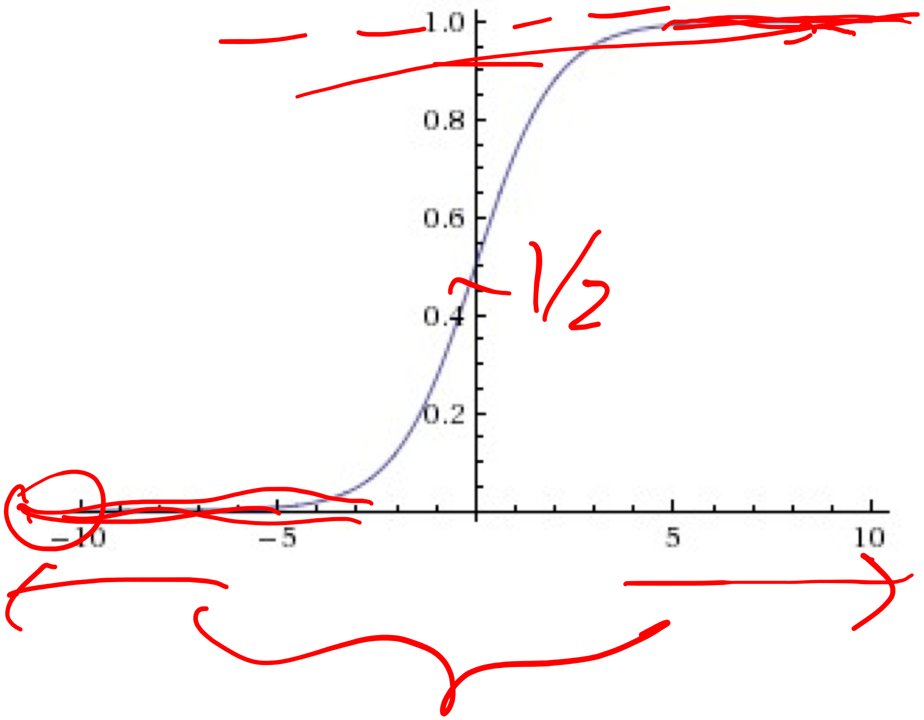
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

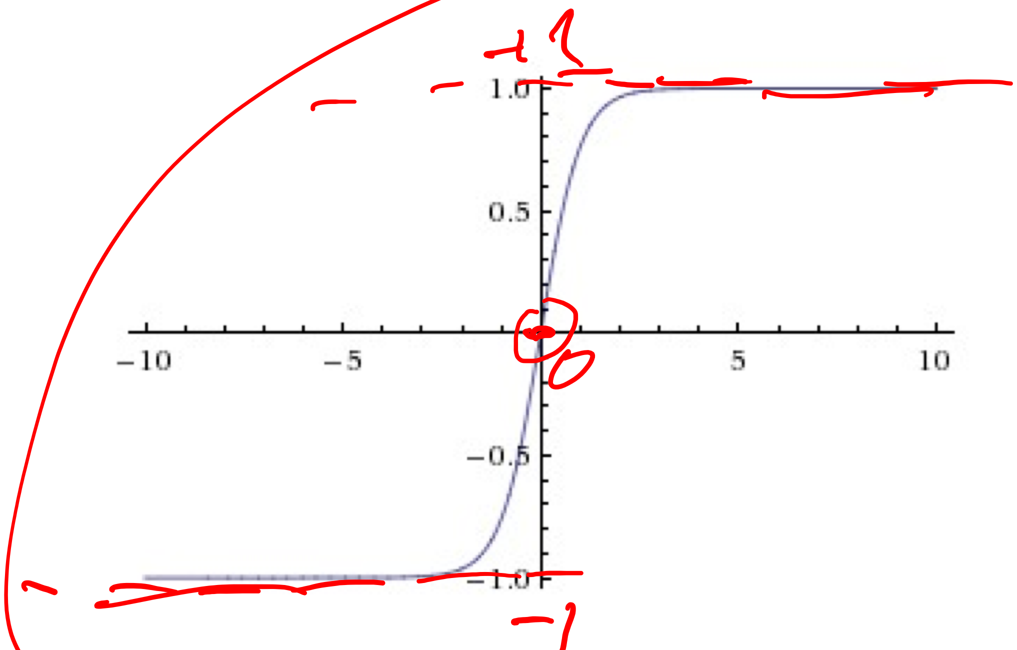
Activation Functions

- sigmoid vs tanh

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

$\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$





$= 2\sigma(2a) - 1$

$\sigma(\sigma(\sigma(a)))$

A quick note

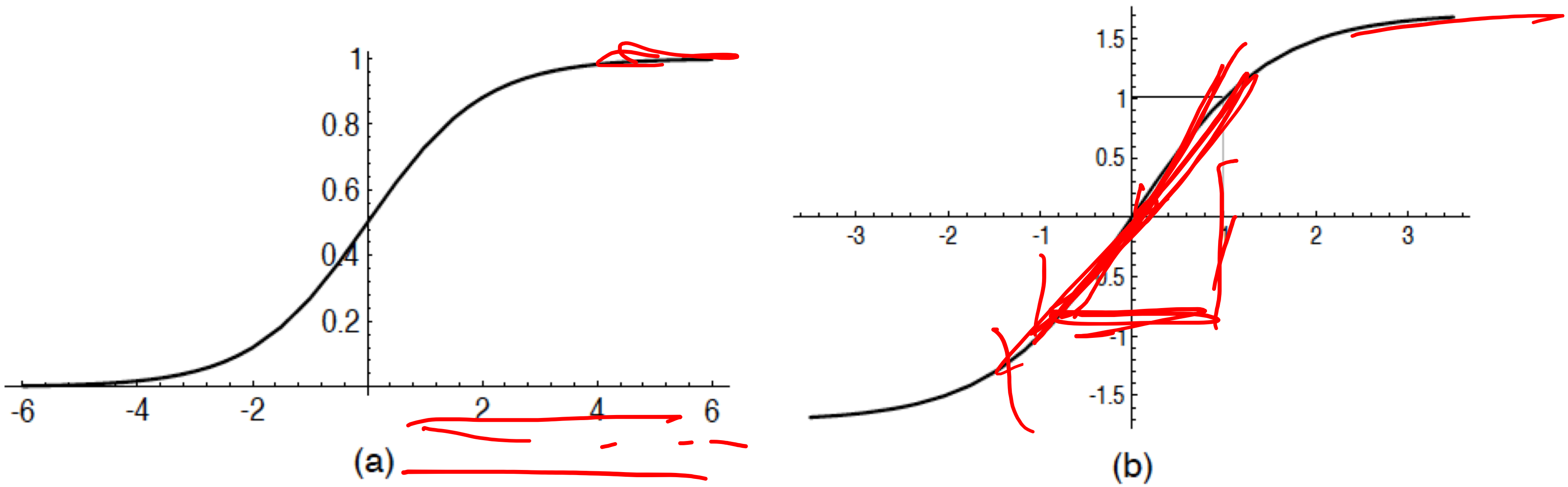
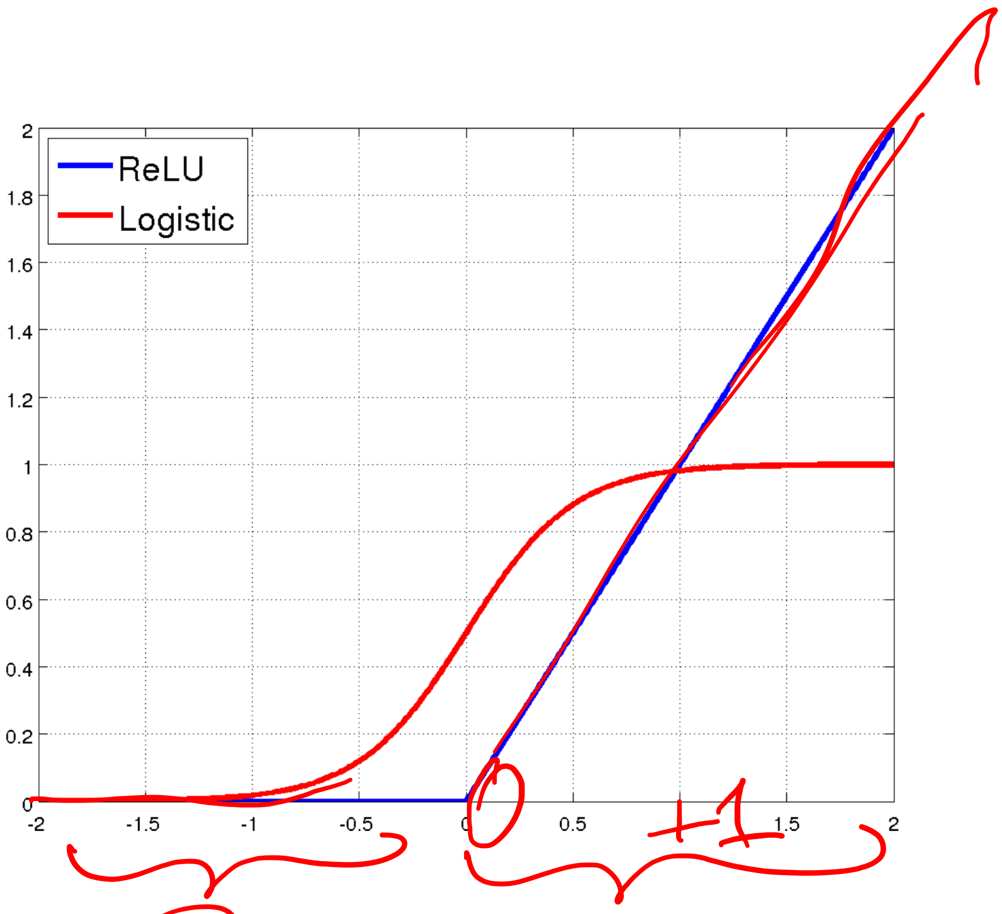


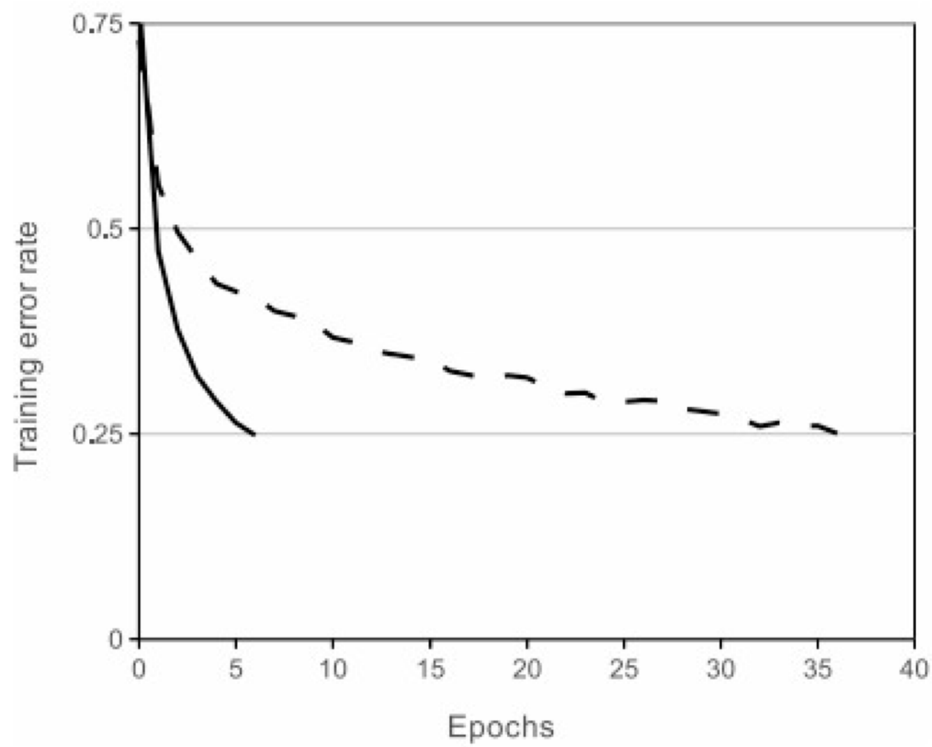
Fig. 4. (a) Not recommended: the standard logistic function, $f(x) = 1/(1 + e^{-x})$. (b) Hyperbolic tangent, $f(x) = 1.7159 \ \tanh\left(\frac{2}{3}x\right)$.

Image Credit: LeCun et al. '98

Rectified Linear Units (ReLU)



$f(a) = \max(0, a)$



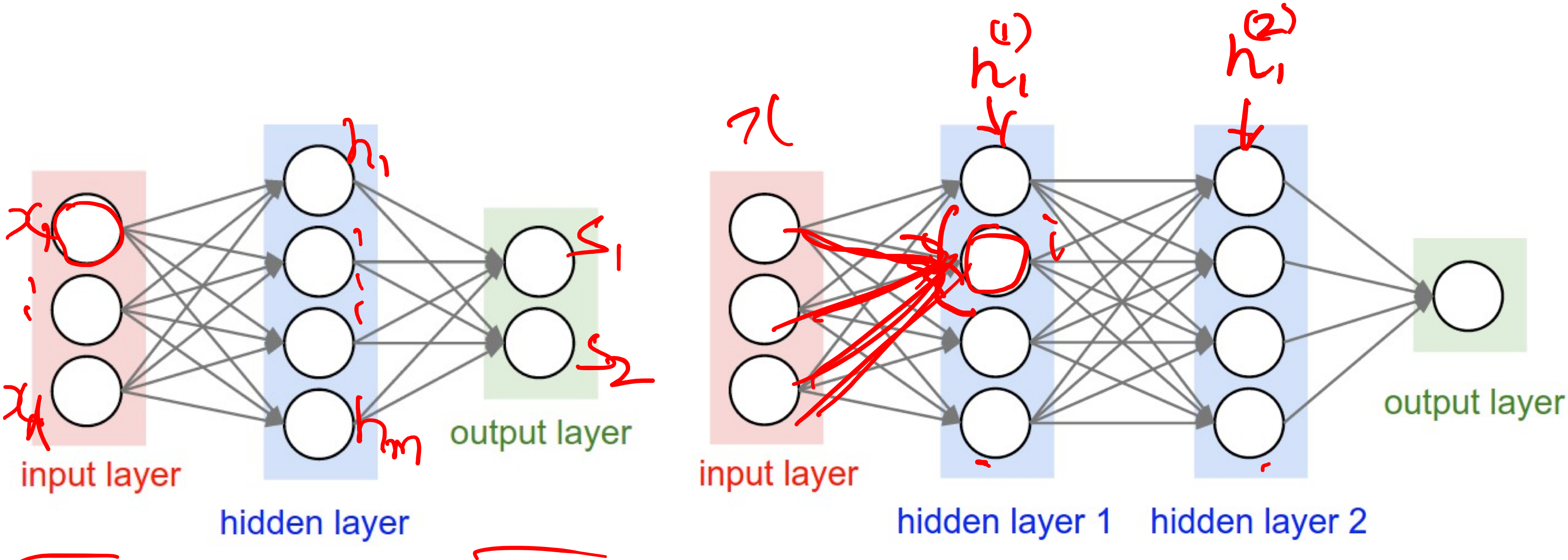
[Krizhevsky et al., NIPS12]

Limitation

- A single "neuron" is still a linear decision boundary
- What to do?
- Idea: Stack a bunch of them together!

Multilayer Networks

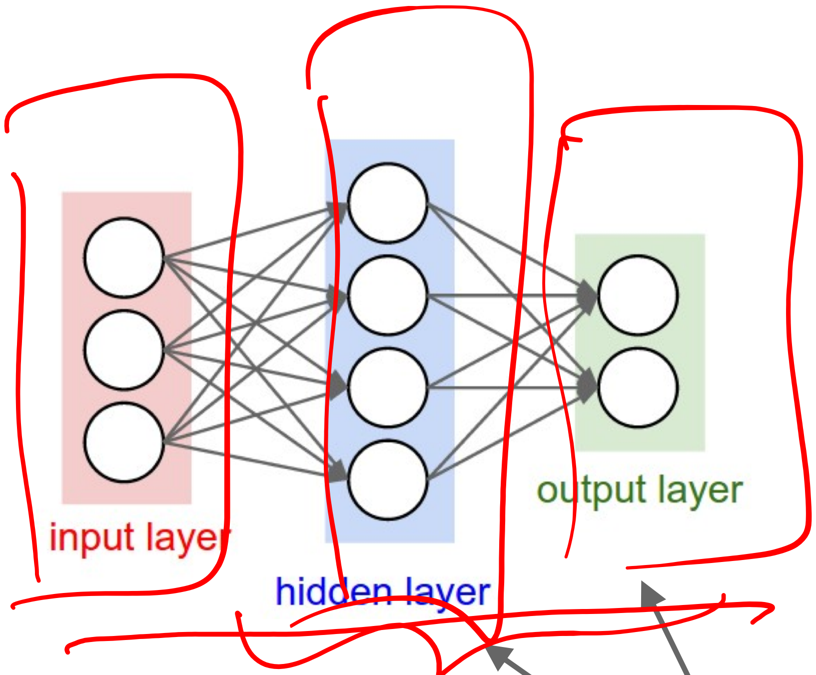
- Cascade Neurons together
- The output from one layer is the input to the next
- Each Layer has its own sets of weights

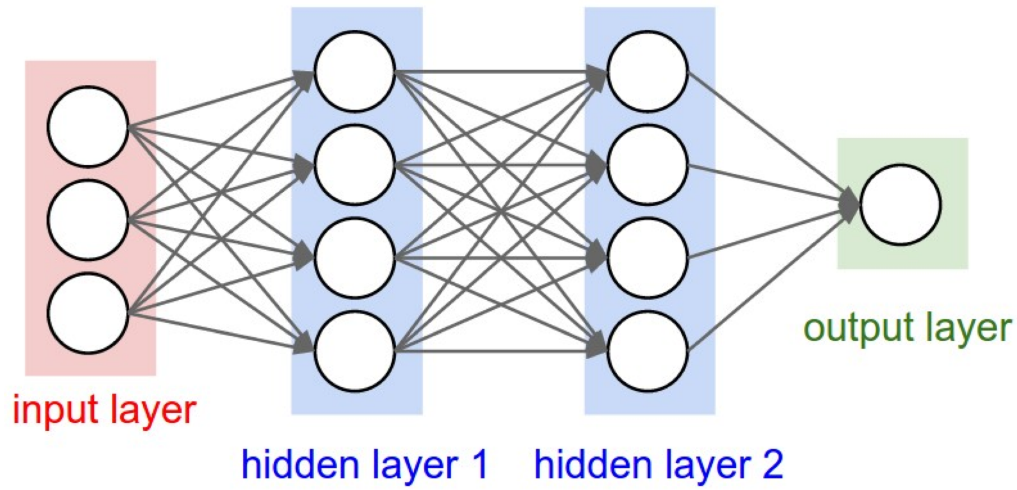


$$h_i^{(1)} = f(\vec{w}_i^T x)$$

Image Credit: Andrej Karpathy, CS231n

Neural networks: Architectures





"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"Fully-connected" layers

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Demo Time

- https://playground.tensorflow.org

Optimization

$$\min_{w} L(\underline{w}; D)$$

Strategy: **Follow the slope**

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Strategy: **Follow the slope**

$$\begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \end{bmatrix}$$

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient
The direction of steepest descent is the **negative gradient**

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Gradient Descent

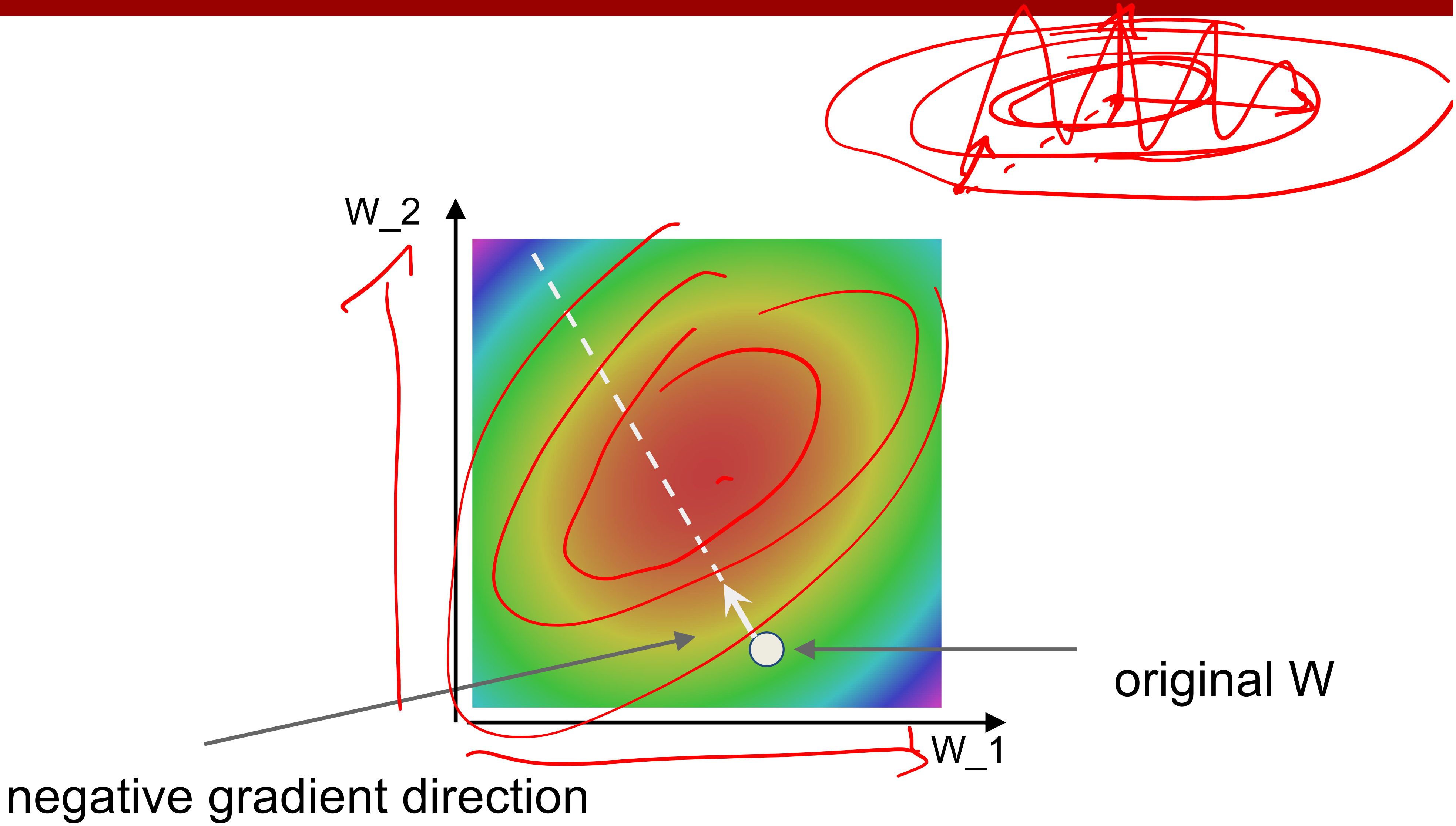
```
# Vanilla Gradient Descent

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

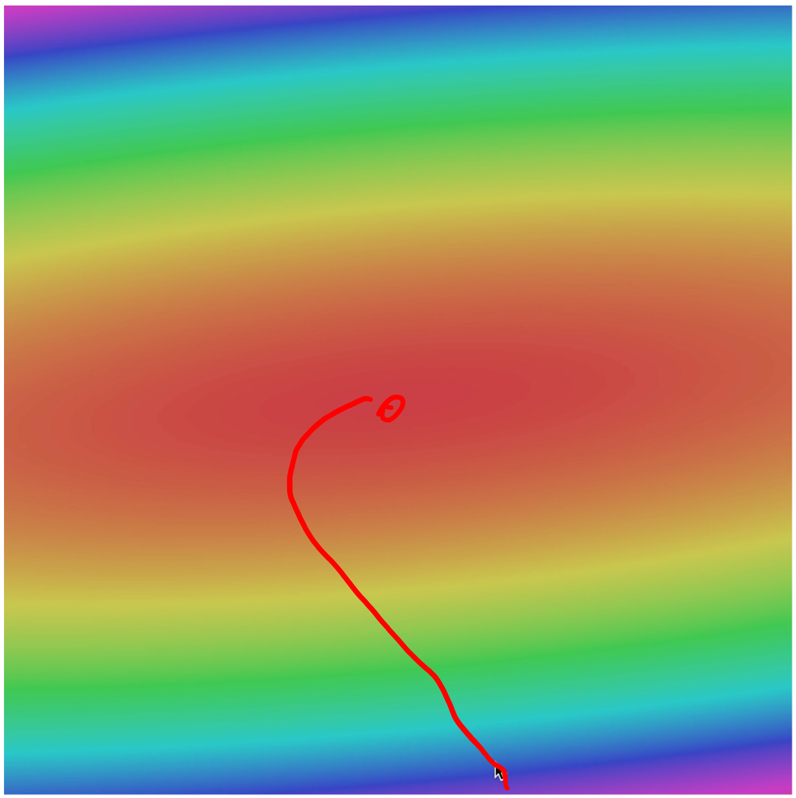
backprop

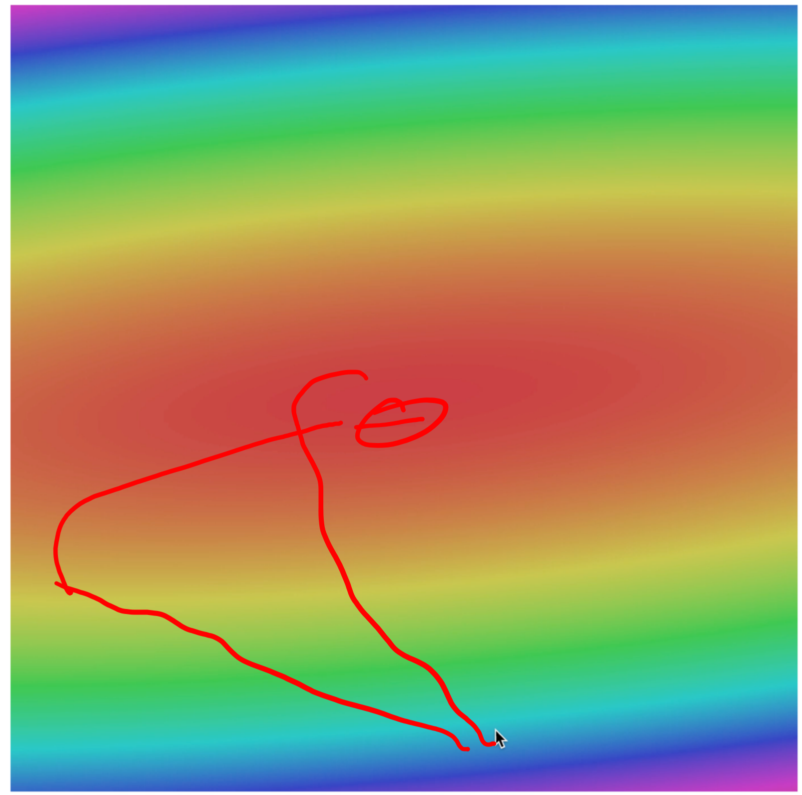
$w^{(0)} = \text{init}$

for $t = 1 \ldots$ tired

$$\vec{w}^{(t+1)} = \vec{w}^{t} - \eta \boxed{\nabla_w L}$$


W_2
W_1
original W
negative gradient direction





Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n