Neural Networks for Computer Vision

(Supervised) Machine Learning (ML)

- Start with past data
  - For 1,000,000 images, what's in the image?
  - For 1,000,000 past customers (and their consumer behaviour), did they repay their loans?
  - For 1,000,000 patients admitted to the hospital, what was the patient outcome?
- Predict the outcome for new data
  - What's in the new image, based on the raw pixel data?
  - What should the credit rating be for the new customer, based on their demographics and past behavior?
  - What is the prognosis for a new patient, given their vitals, lab data, etc. so far
- ML algorithms are often treated as "black boxes"
  - Goal: understand *how* ML-based systems produce the outputs they produce

Machine Learning vs. Intro to Programming

- Intro to Programming
  - Write code that processes inputs in a specific way to produce the desired output, for any provided input
  - Sample inputs and outputs are sometimes provided
    - Write a function f that does [...]. For example, $f(x^{(1)})$ should output $y^{(1)}$, and $f(x^{(2)})$ should output $y^{(2)}$
- Machine Learning (ML)
  - A **Training Set** contains sample inputs and sample outputs
    - Training set: $\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(N)}, y^{(N)}\right)$
  - The ML algorithm uses the **Training Set** to (hopefully) produce the desired outputs
  - The ML algorithm does that by using the **Training Set** to find the parameters θ such that $h_\theta\left(x^{(i)}\right) \approx y^{(i)}$

e.g.: set $h_{(\theta_1,\theta_2,\theta_3)}(x) = \theta_1 + \theta_2 x + \theta_3 x^2$, try to find the best $\theta = (\theta_1, \theta_2, \theta_3)$

Shotgun debugging

Machine Learning vs. Intro to Programming

- Intro to Programming ***done badly***

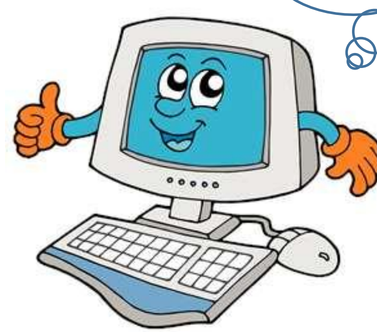
```
def double_list(L):
  for e in L:
    e *= 2
  return L

>>> double_list([0, 0])
[0, 0]
>>> double_list([1, 2])
[1, 2]
```

Change the for to while?

- Machine Learning ***done right***

`>>>` $h_{(0,1.2,0.1)}$`([0, 0])`
`[0, 0]`
`>>>` $h_{(0,1.2,0.1)}$`([1, 2])`
`[1.3, 2.8]`

Change θ_2 to 1.3?



Machine learning (kind of)

$$h_{(\theta_1,\theta_2,\theta_3)}(x) = \theta_1 + \theta_2 x + \theta_3 x^2$$

Supervised Machine Learning

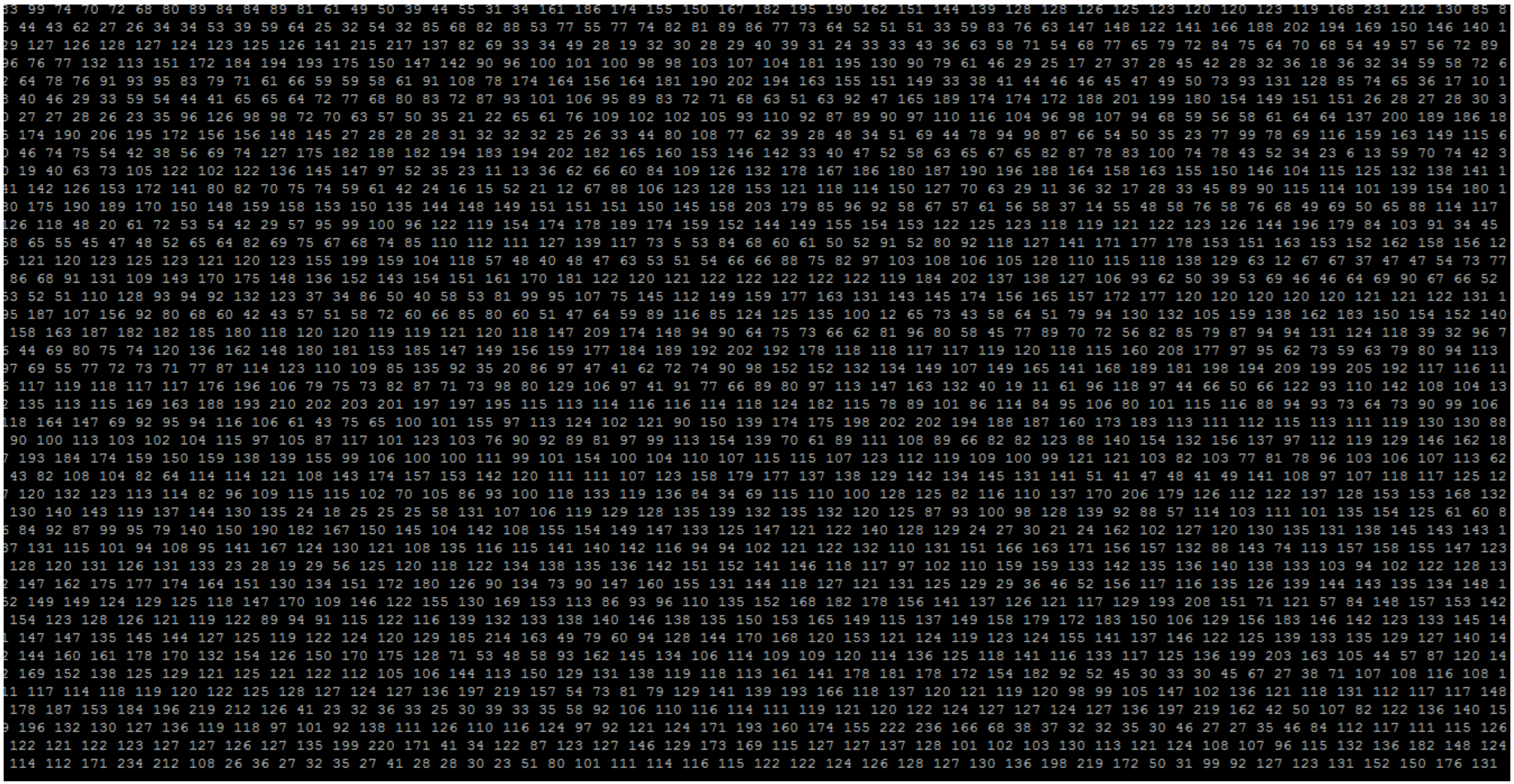
- Training set:
  - Training example 1: $\mathrm{x}^{(1)} = \left(x_1^{(1)}, x_2^{(1)}, \dots, x_m^{(1)}\right)$ output: $y^{(1)}$
  - Training example 2: $\mathrm{x}^{(2)} = \left(x_1^{(2)}, x_2^{(2)}, \dots, x_m^{(2)}\right)$ output: $y^{(2)}$

  …
  - Training example N: $\mathrm{x}^{(\mathrm{N})} = \left(x_1^{(N)}, x_2^{(N)}, \dots, x_m^{(N)}\right)$ output: $y^{(N)}$
- Test set:
  - Test Example 1: $\mathrm{x}^{(N+1)} = \left(x_1^{(N+1)}, x_2^{(N+1)}, \dots, x_m^{(N+1)}\right)$ output: $y^{(N+1)}$
  - Test Example 2: $\mathrm{x}^{(N+2)} = \left(x_1^{(N+2)}, x_2^{(N+2)}, \dots, x_m^{(N+2)}\right)$ output: $y^{(N+2)}$
  - …
  - Test Example K: $\mathrm{x}^{(N+K)} = \left(x_1^{(N+K)}, x_2^{(N+K)}, \dots, x_m^{(N+K)}\right)$ output: $y^{(N+K)}$

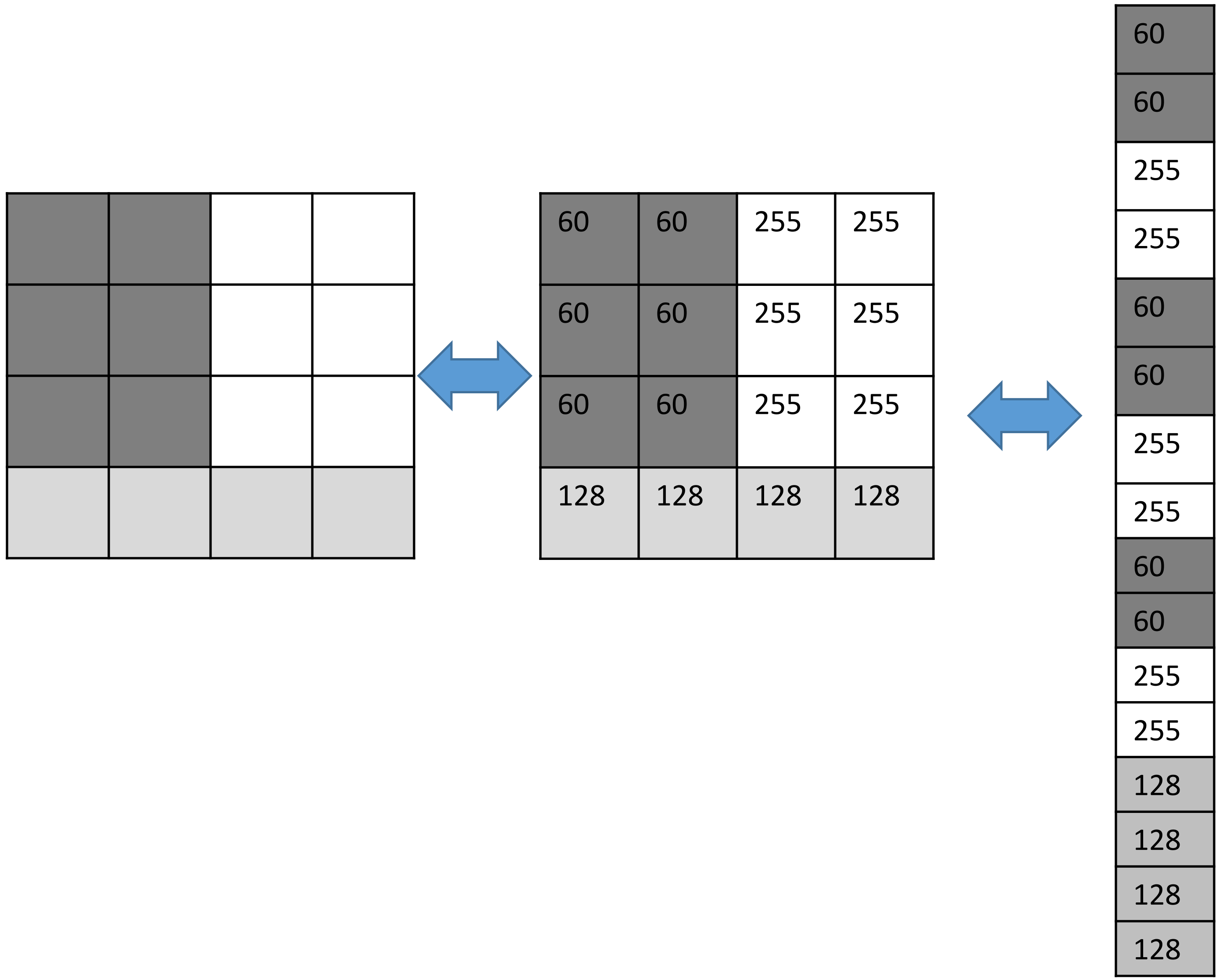
- Goal: Find a θ such that $h_\theta\left(x^{(i)}\right) \approx y^{(i)}$ for $i \in 1, \dots, N$
- Hope: $h_\theta\left(x^{(i)}\right) \approx y^{(i)}$ for any i
- For new input x, predict $h_\theta(x)$

Sample ML task: Recognizing Justin Bieber

What Justin Bieber looks like to a computer



Images ↔ Vectors



The Face Recognition Task

- Training set:
  - $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(N)}, y^{(N)})\}$
    - $x^{(i)}$ is a k-dimensional vector consisting of the intensities of all the pixels in in the i-th photo (20×20 photo $\rightarrow x^{(i)}$ is 400-dimensional)
    - $y^{(i)}$ is the *label* (i.e., name)
- Test phase:
  - We have an input vector x, and want to assign a label y to it
    - Whose photo is it?

Face Recognition using 1-Nearest Neighbors (1NN)

- Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})\}$
- Input: x
- 1-Nearest Neighbor algorithm:
  - Find the training photo/vector $x^{(i)}$ that's as "close" as possible to x, and output the label $y^{(i)}$

Input x

Closest training image to the input x

Output: Paul

Are the two images a and b close?

- Key idea: think of the images as *vectors*
  - Reminder: to turn an image into a vector, simply “flatten” all the pixels into a 1D vector
- Is the distance between the endpoints of vectors a and b small?

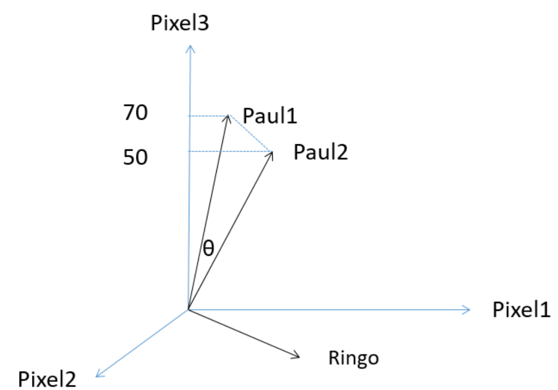
$$|a-b| = \sqrt{\sum_i (a_i - b_i)^2} \text{ small}$$

- Is the cosine of the angle between the vectors a and b large?

$$\cos\theta_{ab} = \frac{a\cdot b}{|a||b|} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}} \text{ large}$$

By the law of cosines

- Is $a \cdot b = \sum_i a_i b_i$ large?
  - Assume $|a| \approx |b| \approx const$

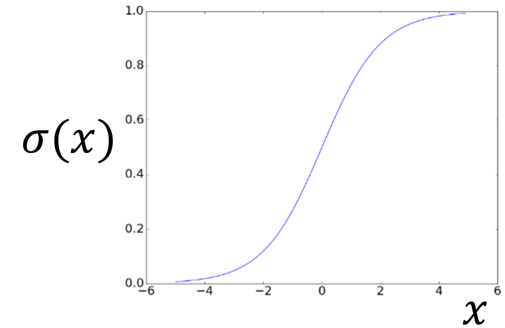


Sample task

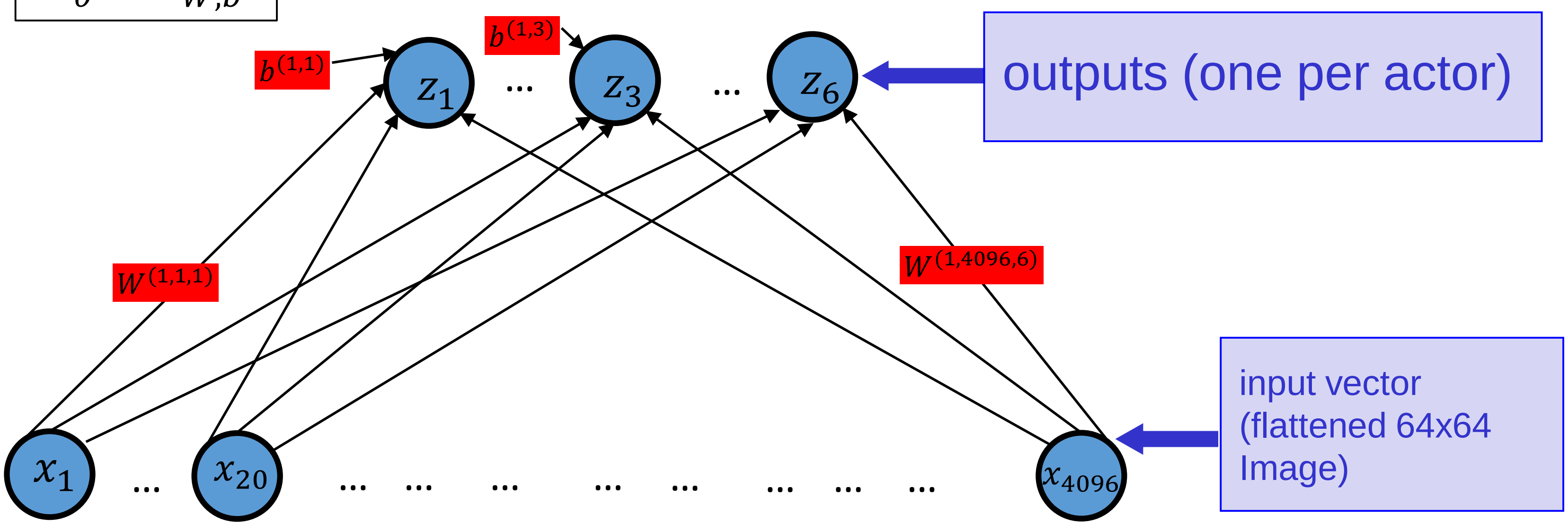
- Training set: 6 actors, with 100 64×64 photos of faces for each
- Test set: photos of faces of the same 6 actors
- Want to classify each face as one of ['Fran Drescher', 'America Ferrera', 'Kristin Chenoweth', 'Alec Baldwin', 'Bill Hader', 'Steve Carell']

The Simplest Possible Neural Network for Face Recognition

$$z_k = \sigma\left(\sum_{j=1}^{4096} W^{(1,j,k)}\, x_j + b^{(1,k)}\right)$$
$$= \sigma\left(W^{(1,*,k)} \cdot x + b^{(1,k)}\right)$$

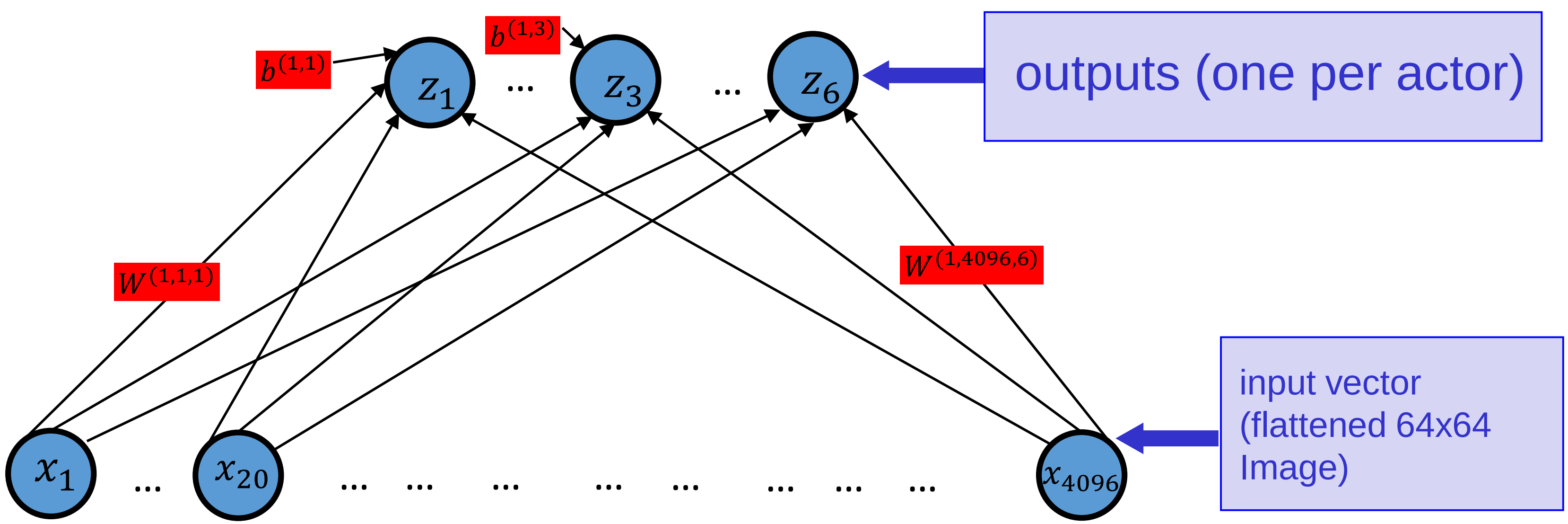


$$\boxed{h_\theta = h_{W,b}}$$



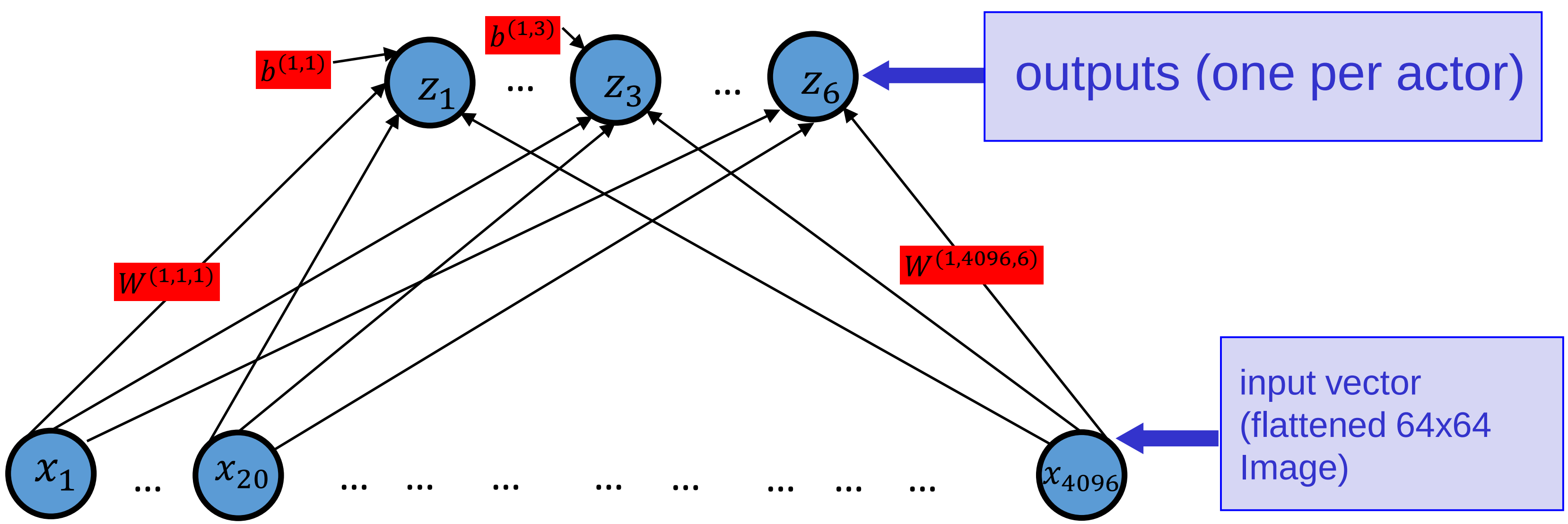
Training a neural network

- Adjust the W's (4096×6 coefs) and b's (6 coefs)
  - Try to make it so that if
    x is an image of actor 1, z is as close as possible to (1, 0, 0, 0, 0, 0)
    x is an image of actor 2, z is as close as possible to (0, 1, 0, 0, 0, 0)
    ……



Face recognition

- Compute the z for a new image x
- If z_k is the largest output, output name k



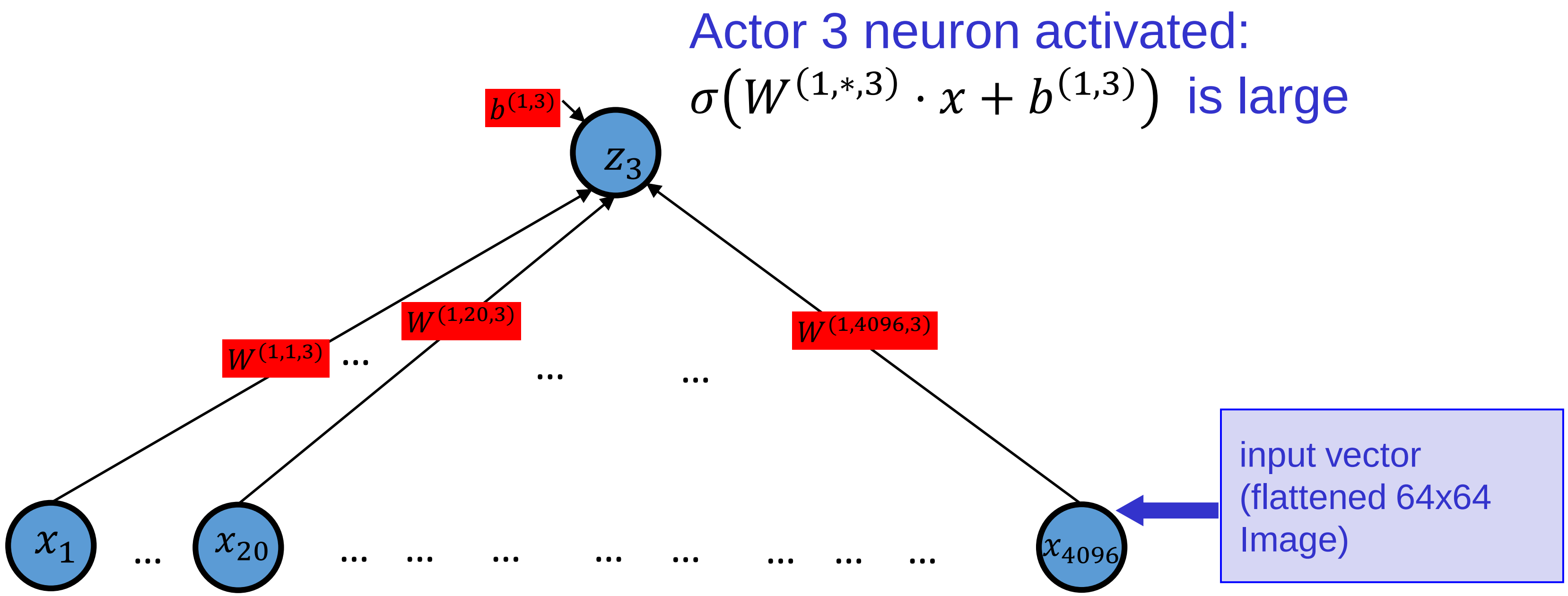
An interpretation

z_1 is large if $W^{(1,*,1)} \cdot x$ is large
z_2 is large if $W^{(1,*,2)} \cdot x$ is large

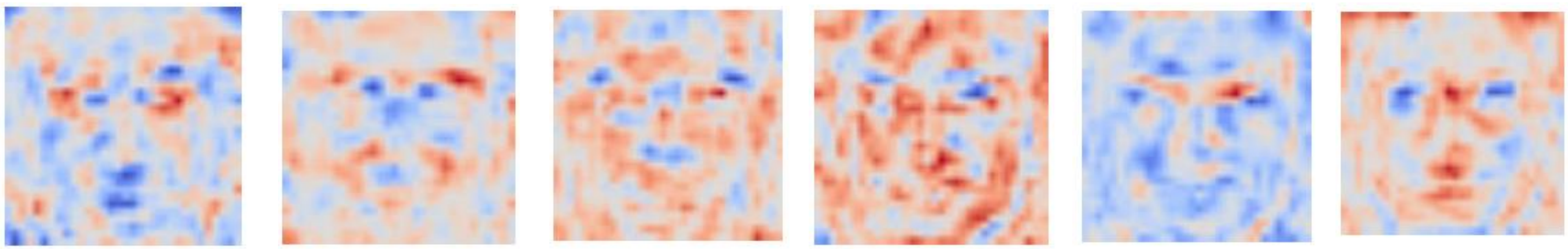
z_3 is large if $W^{(1,*,3)} \cdot x$ is large

....

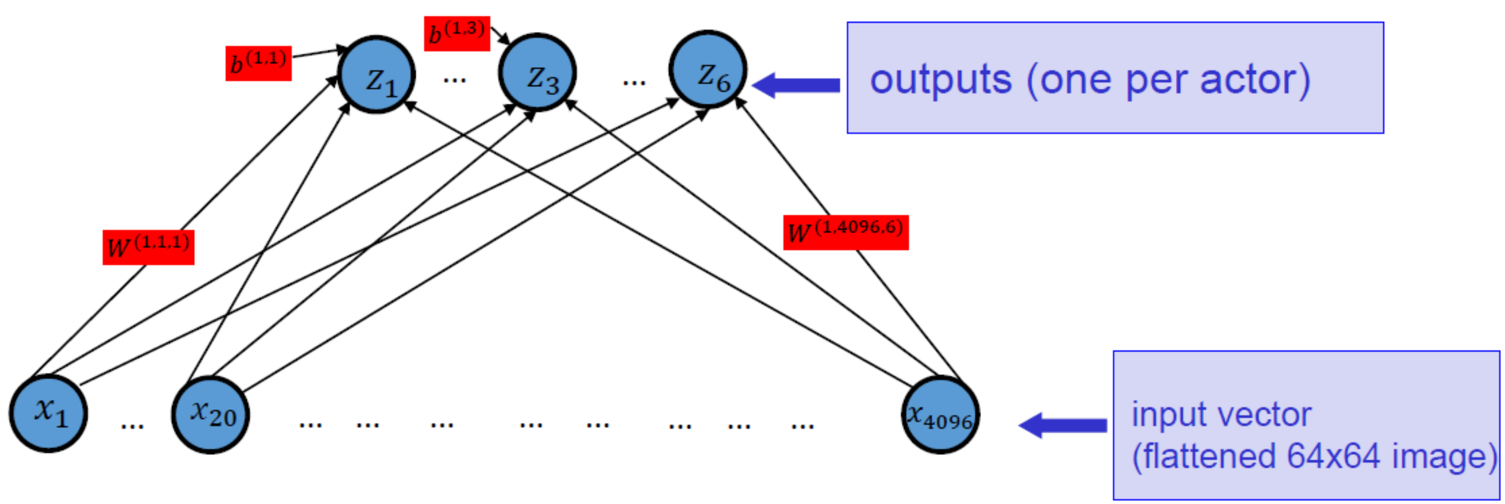
$W^{(1,*,1)}$, $W^{(1,*,2)}$, ..., $W^{(1,*,6)}$ are *templates* for the faces of actor 1, actor 2, ..., actor 6



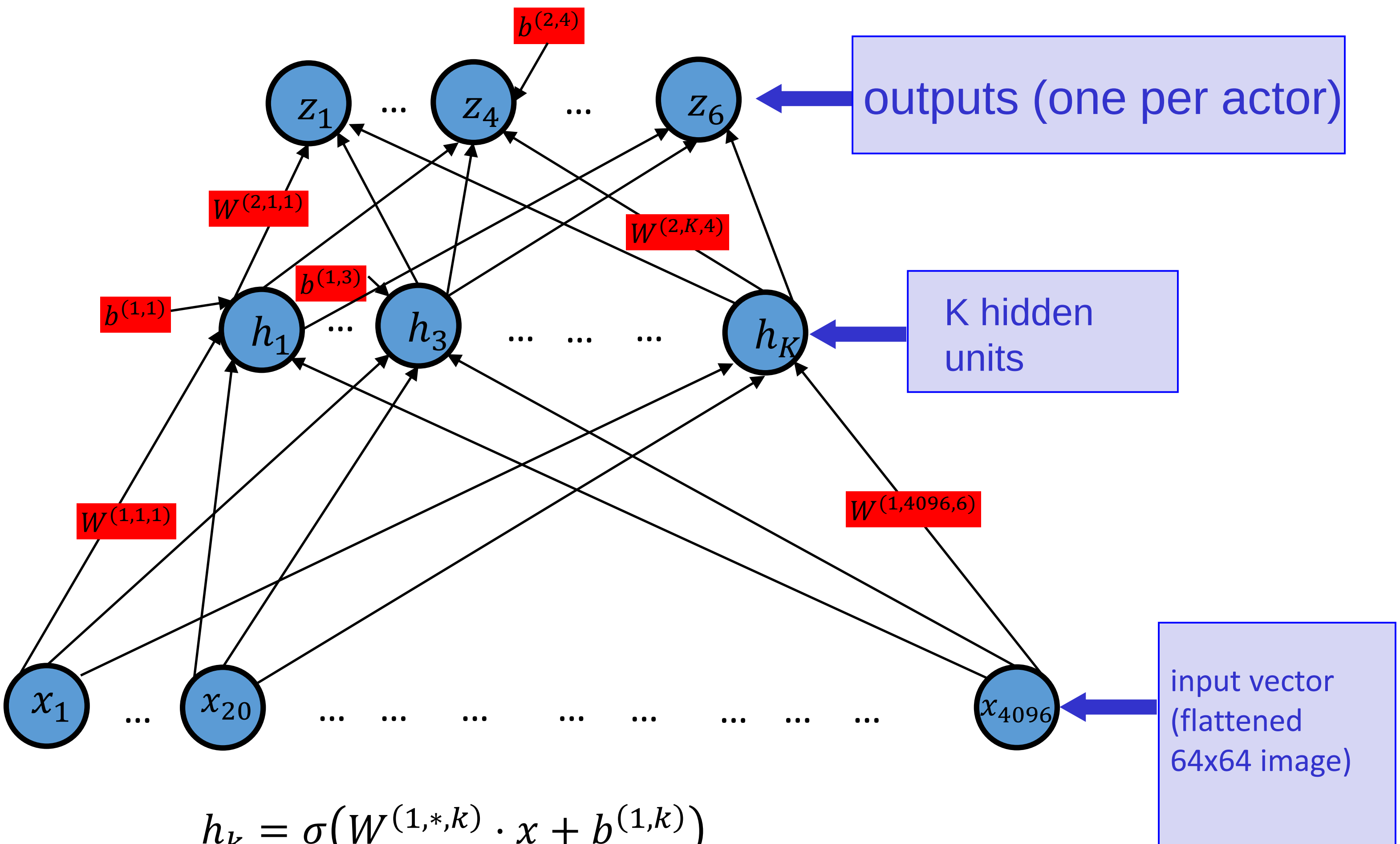
Visualizing the parameters W



Baldwin $W^{(1,*,1)}$ | Carrel $W^{(1,*,2)}$ | Hader $W^{(1,*,3)}$ | Ferrera $W^{(1,*,4)}$ | Drescher $W^{(1,*,5)}$ | Chenoweth $W^{(1,*,6)}$



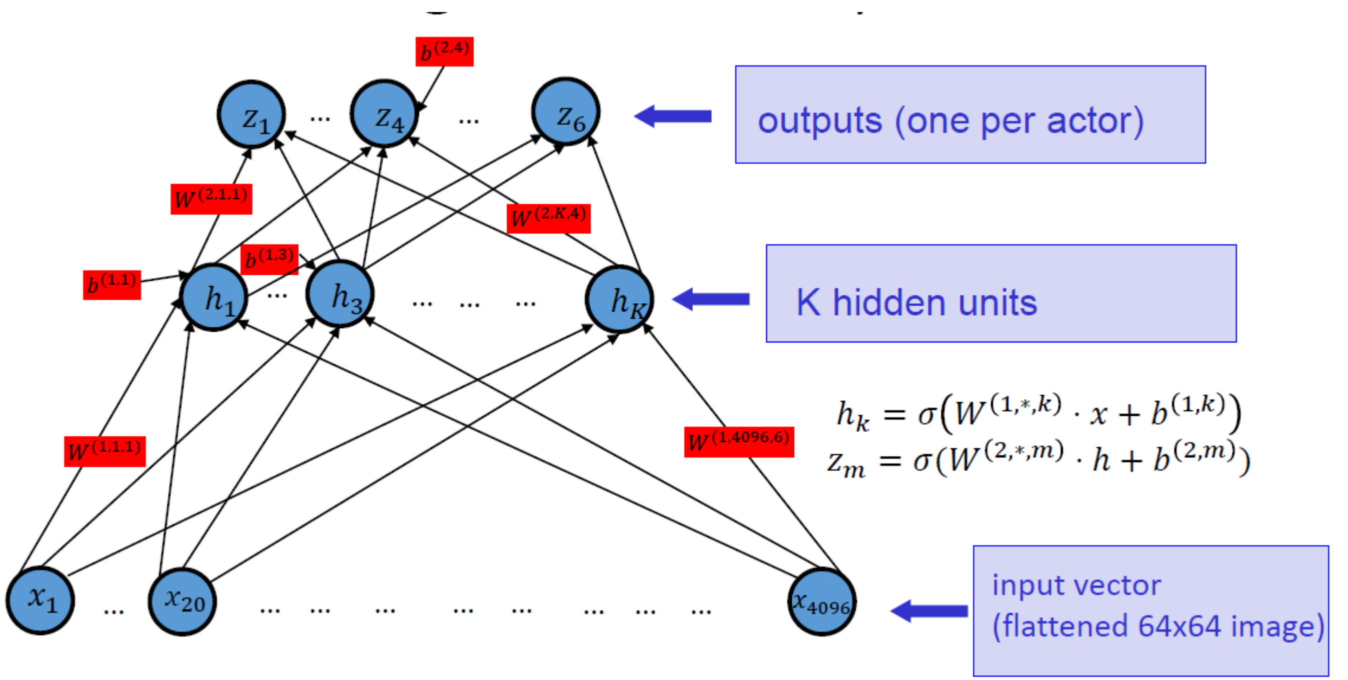
Deep Neural Networks: Introducing Hidden Layers



$$h_k = \sigma\left(W^{(1,*,k)} \cdot x + b^{(1,k)}\right)$$
$$z_m = \sigma(W^{(2,*,m)} \cdot h + b^{(2,m)})$$

Why a hidden layer?

- Instead of checking whether x looks like one of 6 templates, we'll be checking whether x looks like one of K templates, for a large K
  - If template k (i.e., $W^{(1,*,k)}$) looks like actor 6, $W^{(2,k,6)}$ will be large



Recap: Face Recognition with ML

- 1-Nearest-Neighbor: match x to all the images in the training set
- 0-hidden-layer neural network*: match x to several templates, with one template per actor
  - The templates work better than any individual photo
- 1-hidden-layer neural network: match x to K templates
  - The templates work better than any individual photo
  - More templates means better accuracy on the training set

*A.K.A. multinomial logistic regression to its friends

Visualizing a One-Hidden-Layer NN

Deep Neural Networks as a Model of Computation

- Most people's first instinct a face classifier is to write a complicated computer program
- A deep neural network *is* a computer program:

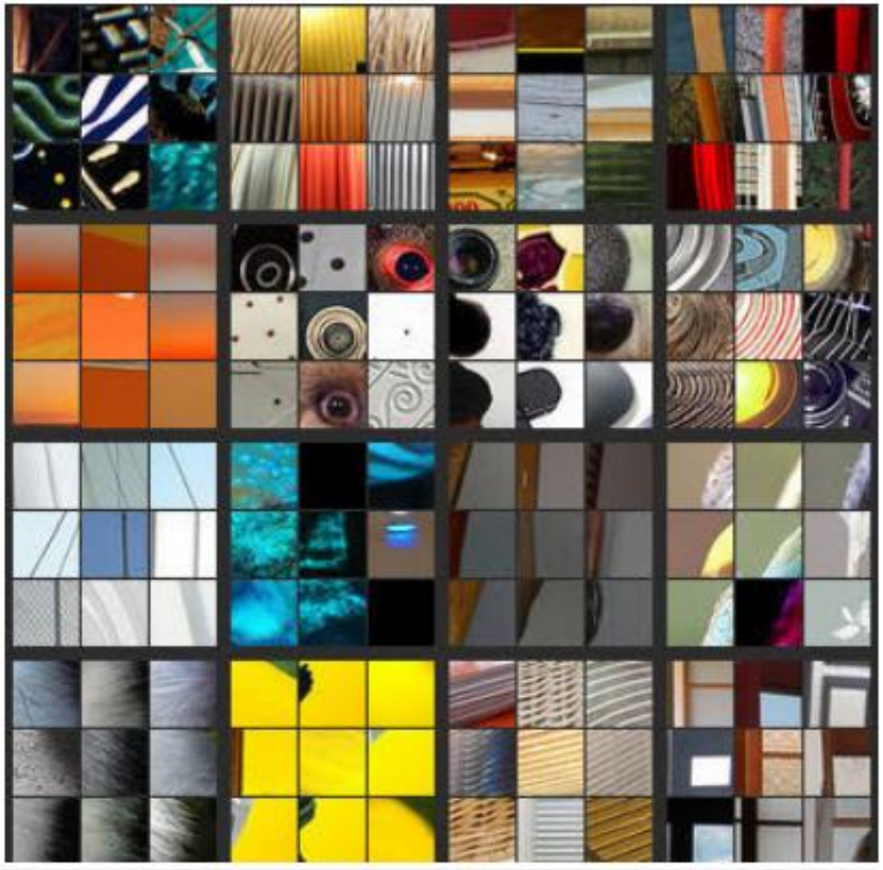
```
h1 = f1(x)
h2 = f2(h1)
h3 = f3(h2)
…
h9 = f9(h8)
```

- Can think of every layer of a neural network as one step of a parallel computation
- Features/templates are the functions that are applied to the previous layers
- Learning features ⇔ Learning what function to apply at step t of the algorithm

What are the hidden units doing?

- Find the images in the dataset that activate the units the *most*
- *Let's see some visualizations of neurons of a large deep network trained to recognize objects in images*
    - Then network classifies images as one of 1000 objects (sample objects: toy poodle, flute, forklift, goldfish…)
    - The network has 8 layers
    - Note: more tricks were used in designing the networks than we have time to mention! In particular, a convolutional architecture is crucial

Units in Layer 3



Matthew Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks” (ECCV 2014)

Units in Layer 4

Matthew Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks” (ECCV 2014)

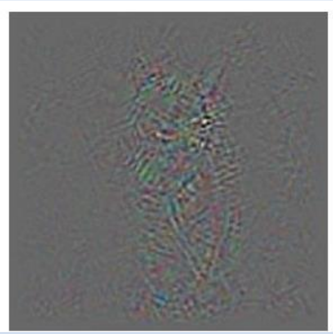
Units in Layer 5

Matthew Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks” (ECCV 2014)

Which pixels are responsible for the output?

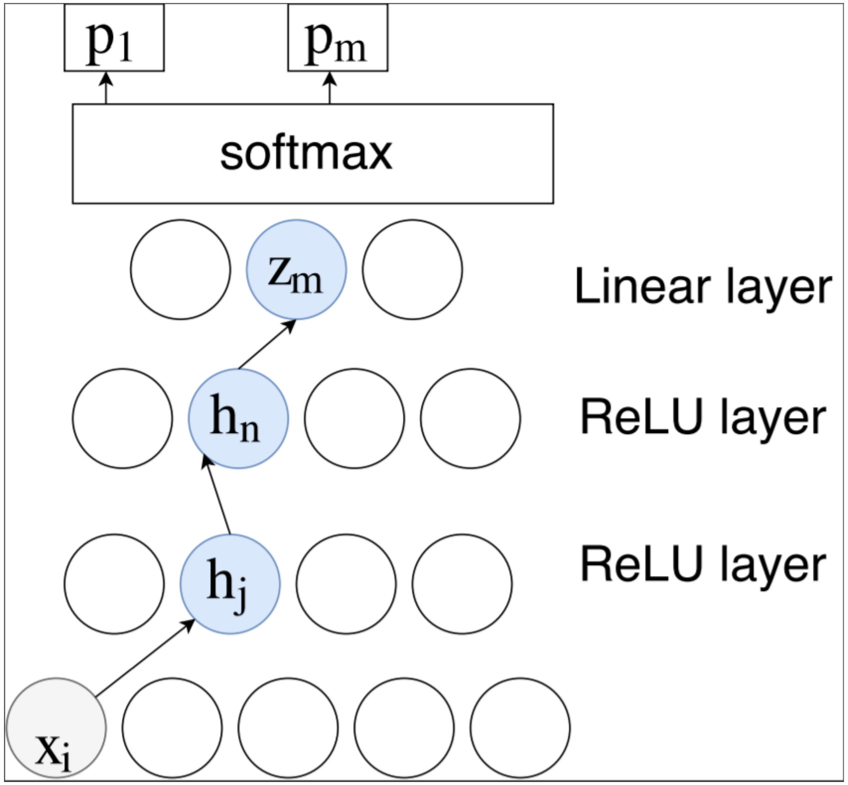
- For each pixel in a particular image ask:
  - If I changed this pixel j by a little bit, how would that influence the output i
  - Equivalent to asking: what's the gradient $\frac{\partial output_i}{\partial input_j}$
  - We can visualize why a particular output was chosen by the network by computing $\frac{\partial output_i}{\partial input_j}$ for every j, and displaying that as an image

Gradient and Guided Backpropagation

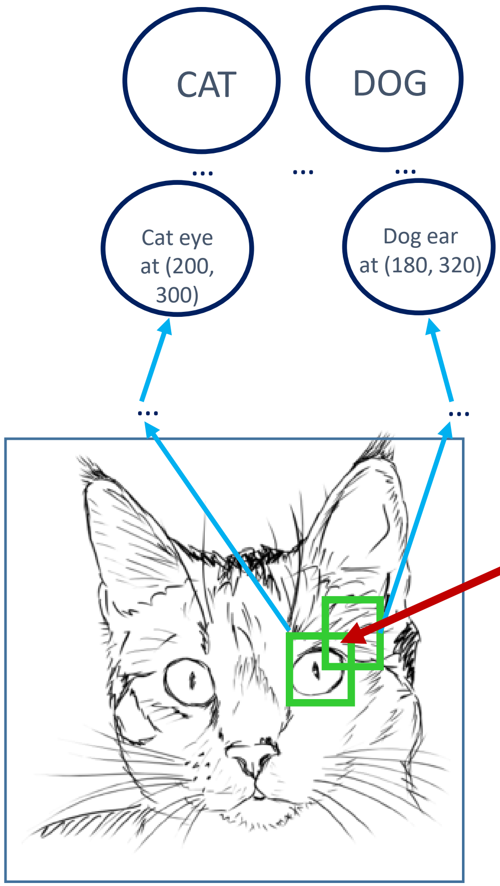
Image I	
$\frac{\partial \text{Cat-Neuron}}{\partial I}$	
Guided Backpropagation visualization	

Guided backpropagation

- Instead of computing $\frac{\partial p_m}{\partial x}$, only consider paths from x to p_m where the weights are positive and all the units are positive (and greater than 0). Compute this modified version of $\frac{\partial p_m}{\partial x}$
- Only consider evidence for neurons being active, discard evidence for neurons having to be not active



Guided Backpropagation Intuition



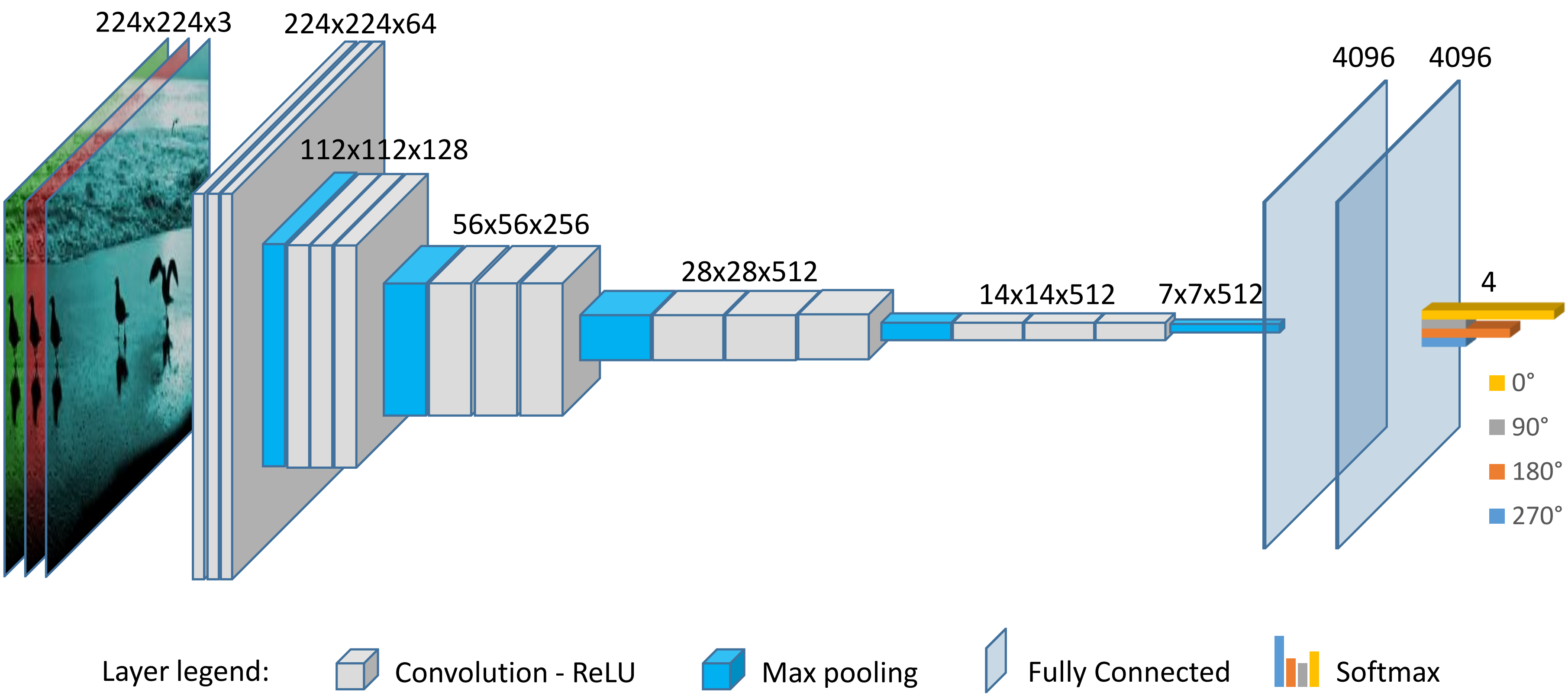
Pixel provides both positive (via a cat eye detection) and negative (via absence of cat eye detection) evidence for a cat in the image

Application: Photo Orientation

- Detect the correct orientation of a consumer photograph
- Input photo is rotated by 0°, 90°, 180° or 270°
- Help speed up the digitization of analog photos
- Need correctly oriented photos as inputs for other systems

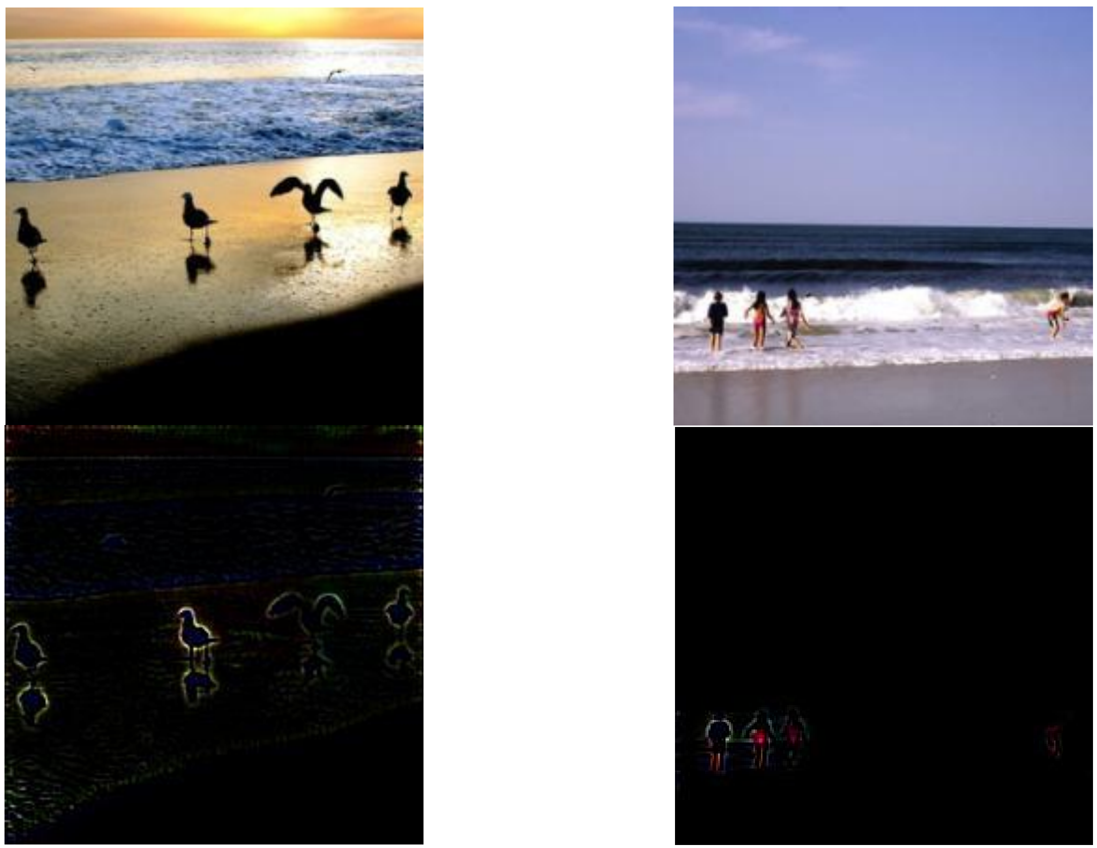
0° 90° 180° 270°

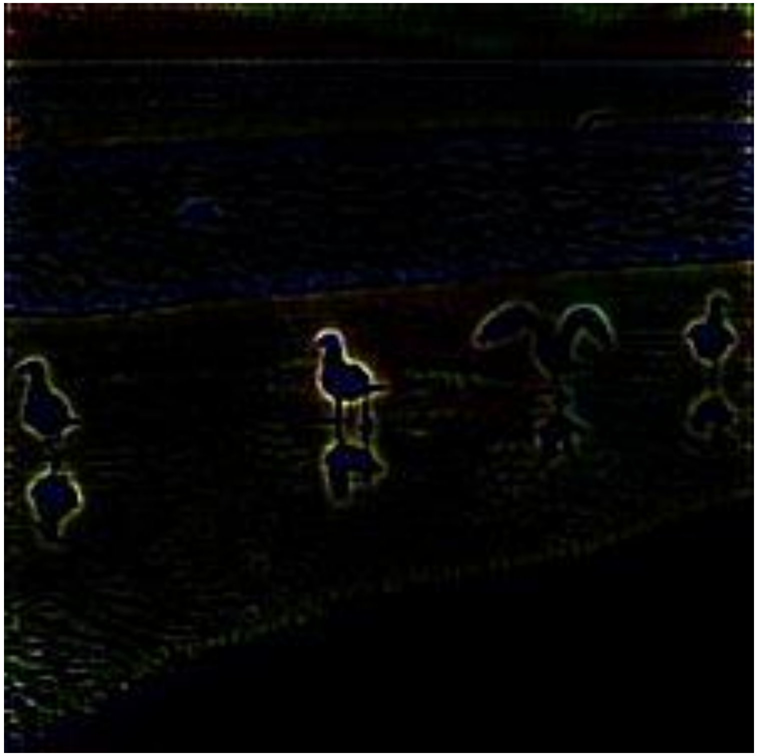
A Neural Network for Photo Orientation

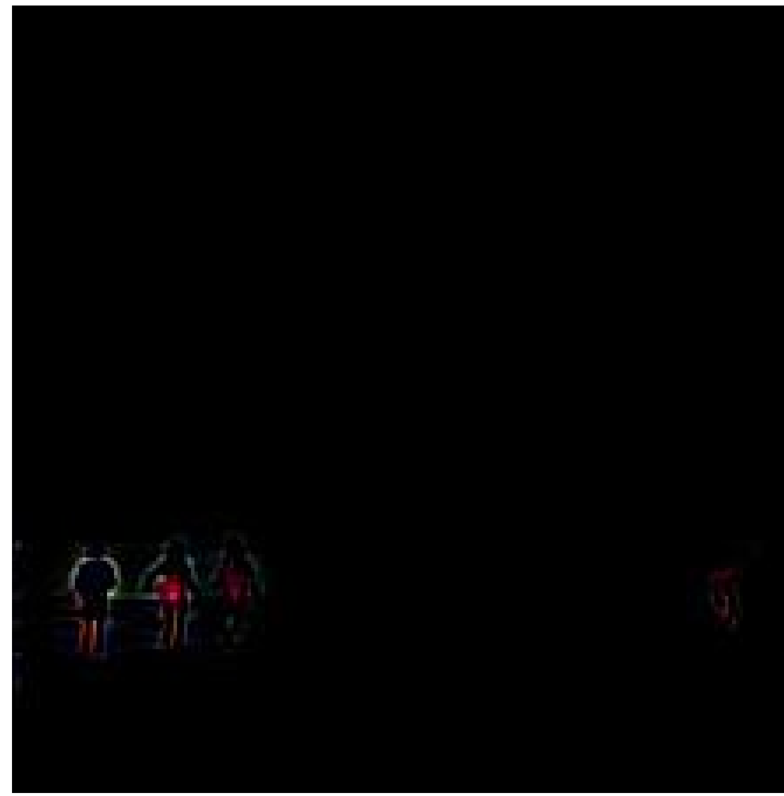


Correctly Oriented Photos

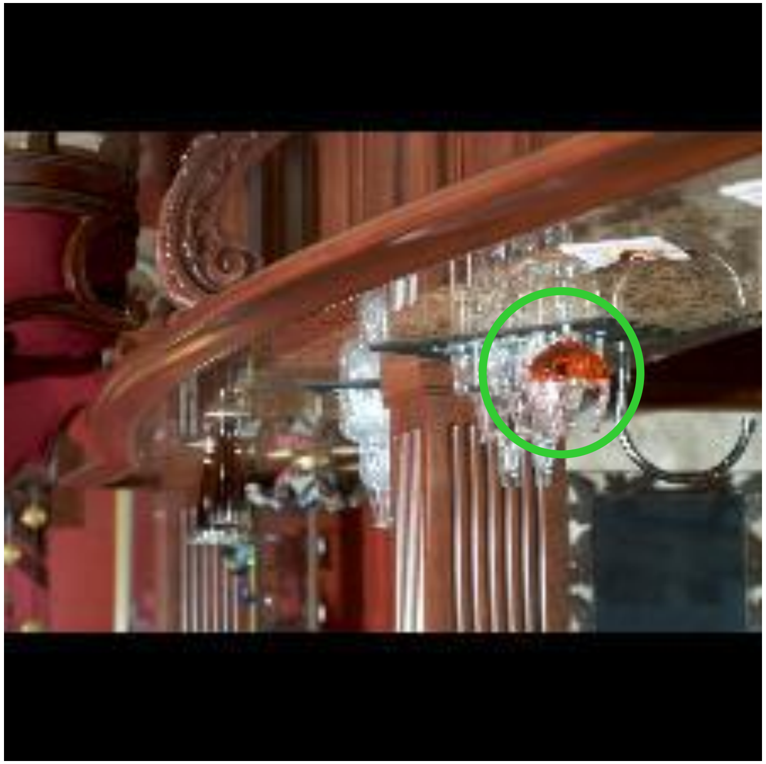
- Display pixels that provide direct positive evidence for 0°





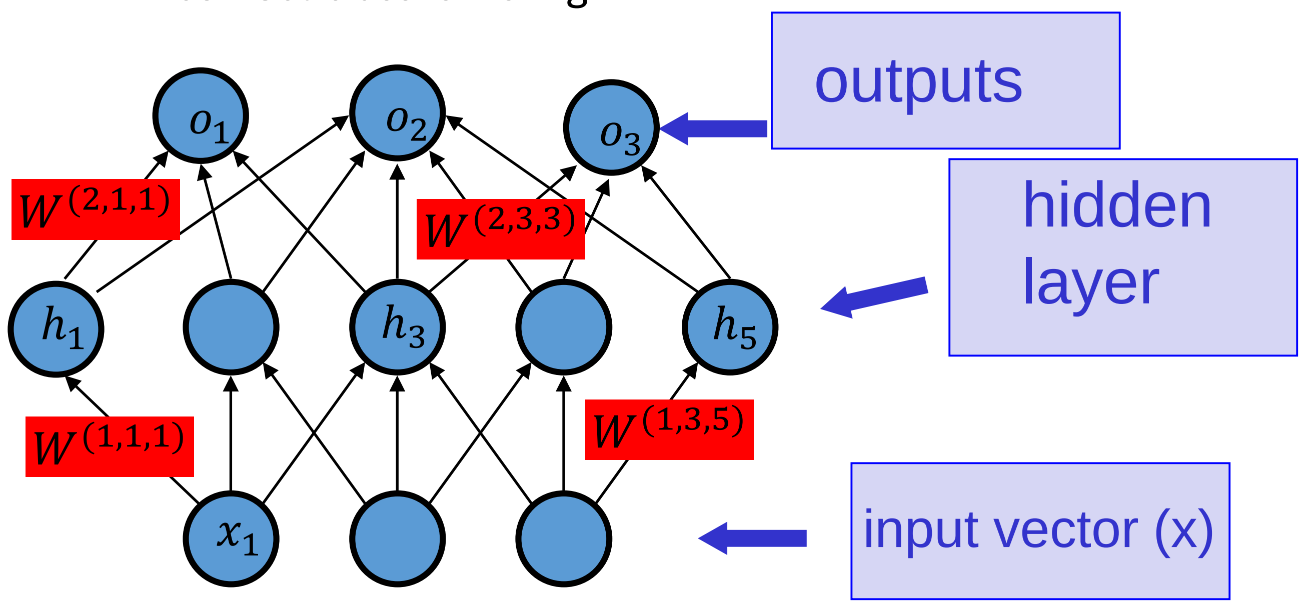


Incorrectly-oriented photos




Backpropagation

o_i is large if the probability that the correct class is *i* is high



A possible cost function:

$$C(o, y) = \sum_{i=1}^{m} |y^{(i)} - o^{(i)}|^2$$

Make $y^{(i)}$'s and $o^{(i)}$ as close as possible

Partial Derivatives of the Cost Function

- We need the partial derivatives of the cost function $C(o, y)$ w.r.t all the W and b
- $o_i = \sigma\left(\sum_j W^{(2,j,i)} h_j + b^{(2,i)}\right)$
- Partial derivative of $C(o, y)$ w.r.t $W^{(2,j,i)}$
- $$\frac{\partial C}{\partial W^{(2,j,i)}}(x, y, W, b, h, o) = \frac{\partial o_i}{\partial W^{(2,j,i)}}(x, y, W, b, h, o)\frac{\partial C}{\partial o_i}(x, y, W, b, h, o)$$

$$= \frac{\partial\left(\sum_j W^{(2,j,i)} h_j\right)}{\partial W^{(2,j,i)}}(x, y, W, b, h, o)\frac{\partial \sigma}{\partial\left(\sum_j W^{(2,j,i)} h_j\right)}(x, y, W, b, h, o)\frac{\partial C}{\partial o_i}(x, y, W, b, h, o)$$

$$= h_j \frac{\partial \sigma}{\partial \sum_j W^{(2,j,i)} h_j}(x, y, W, b, h, o)\frac{\partial C}{\partial o_i}(x, y, W, b, h, o)$$

$$= h_j \sigma'\left(\sum_j W^{(2,j,i)} h_j + b^{(2,j)}\right)\frac{\partial}{\partial o_i} C(o, y)$$

$$h_j g'\left(\sum_j W^{(2,j,i)} h_j + +b^{(2,j)}\right)\frac{\partial C}{\partial o_i}(o,y)$$

- $\sigma(t) = \frac{1}{1+\exp(-t)}$

$$\sigma'(t) = \frac{\exp(-t)}{(1+\exp(-t))^2} = \frac{1}{(1+\exp(-t))}\frac{\exp(-x)}{(1+\exp(-t))} = \sigma(t)\big(1-\sigma(t)\big)$$

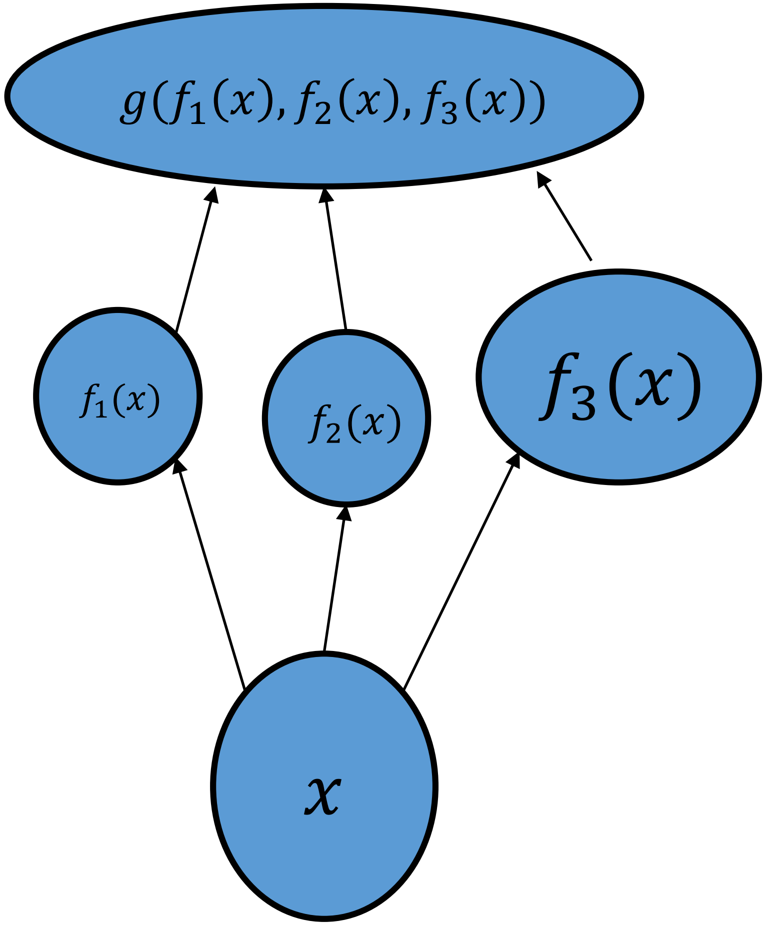
- $C(o,y) = \sum_{i=1}^{N}(o_i - y_i)^2$

$$\frac{\partial}{\partial o_i}\sum_{i=1}^{N}(o_i - y_i)^2 = 2(o_i - y_i)$$

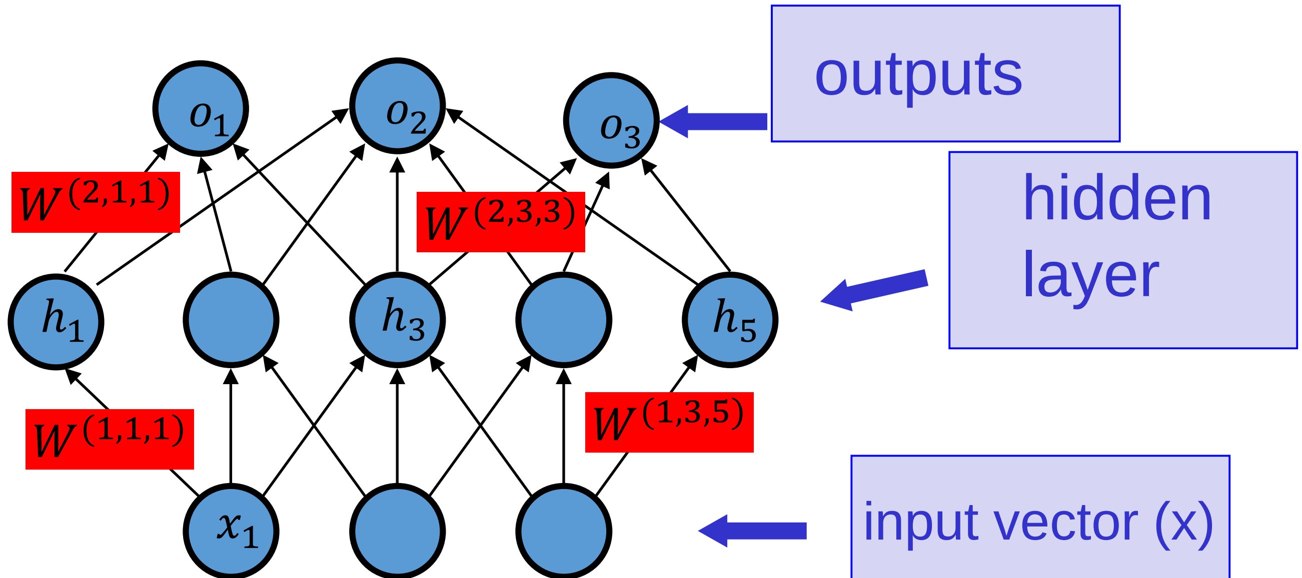
$$\frac{\partial C}{\partial W^{(2,j,i)}}(x,y,W,b,h,o) = h_j \sigma'\left(\sum_j W^{(2,j,i)} h_j + b^{(2,j)}\right)\frac{\partial C}{\partial o_i}(o,y)$$

$$= 2h_j\ \sigma\left(\sum_j W^{(2,j,i)} h_j + b^{(2,j)}\right)\left(1 - \sigma\left(\sum_j W^{(2,j,i)} h_j + b^{(2,j)}\right)\right)(o_i - y_i)$$

Multivariate Chain Rule



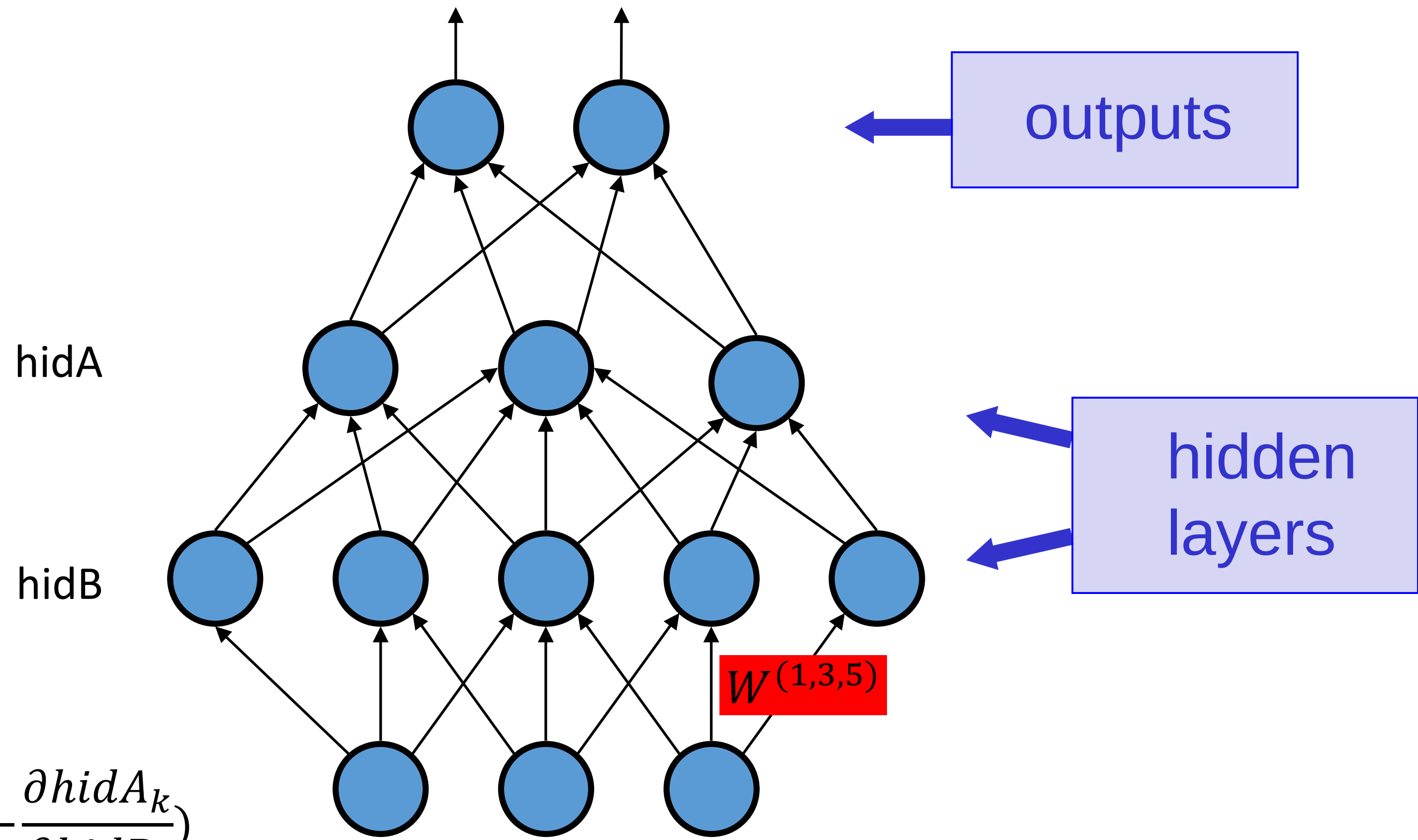
$$\frac{\partial g}{\partial x} = \sum \frac{\partial g}{\partial f_i} \frac{\partial f_i}{\partial x}$$



$$\frac{\partial C}{\partial h_i} = \sum_k \left(\frac{\partial C}{\partial o_k} \frac{\partial o_k}{h_i}\right)$$

$$\frac{\partial C}{\partial W^{(1,j,i)}} = \frac{\partial C}{\partial h_i} \frac{\partial h_i}{\partial W^{(1,j,i)}}$$

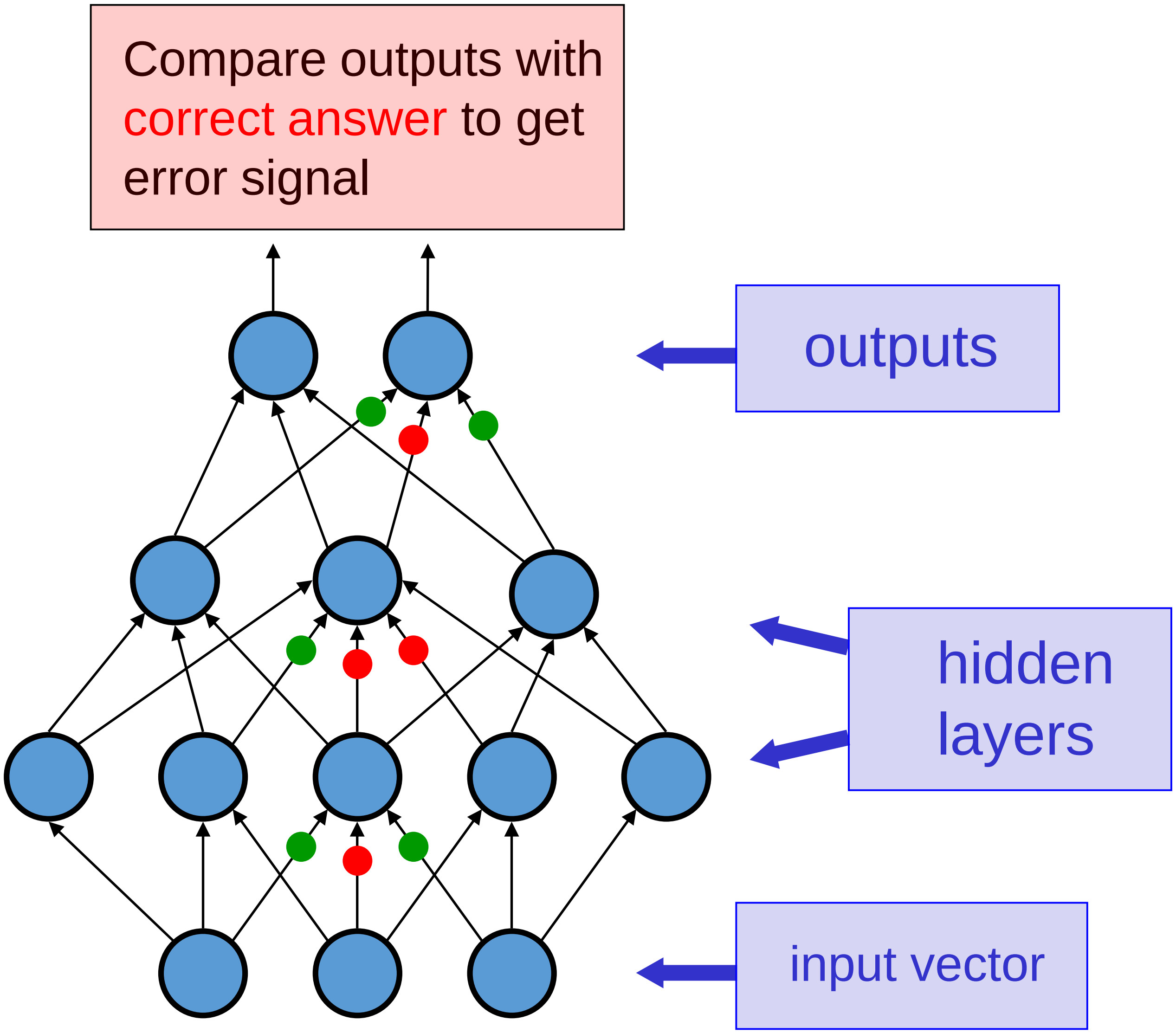
Backpropagation: dynamic programming



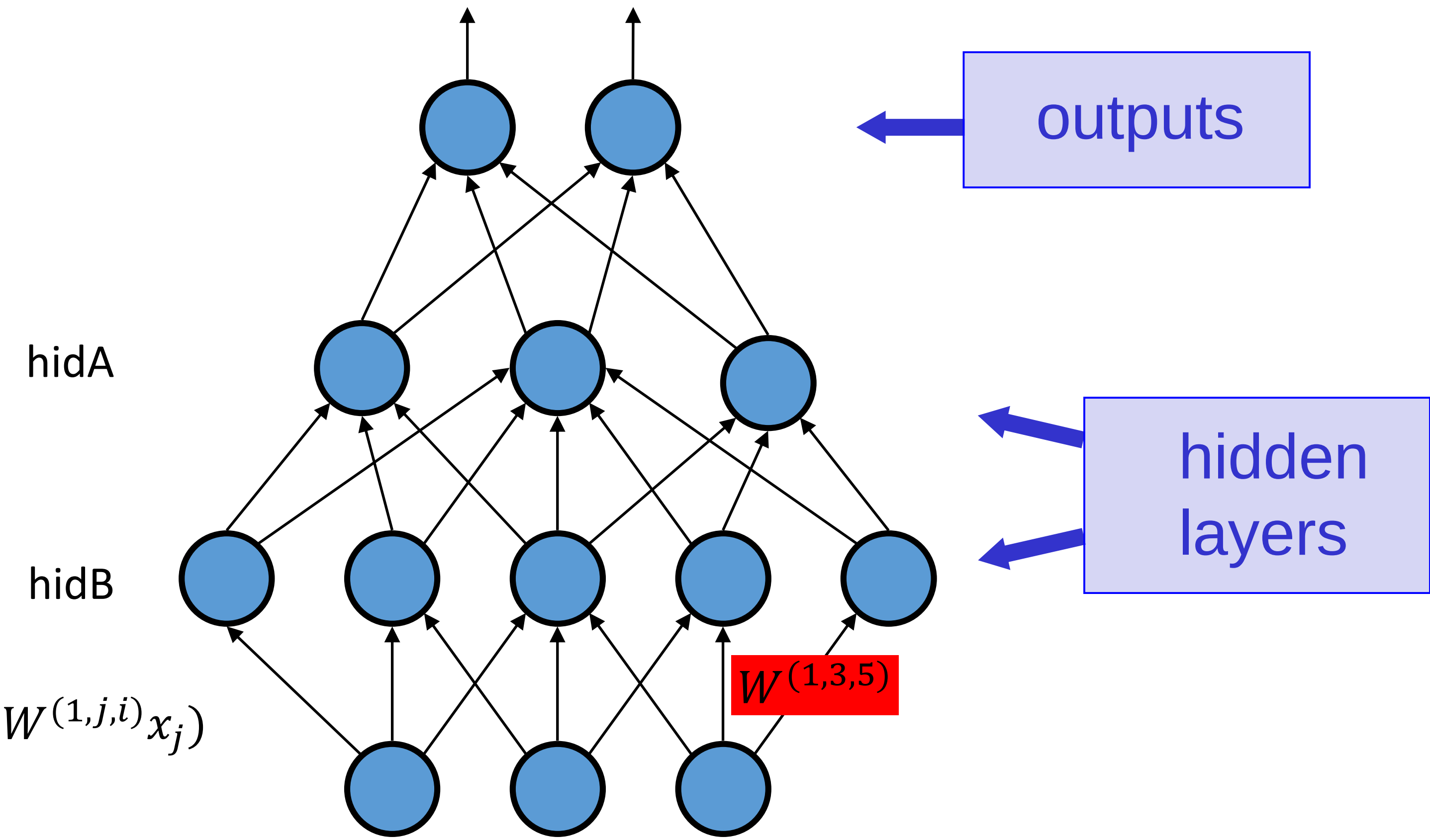
$$\frac{\partial C}{\partial hidB_i} = \sum_k \left(\frac{\partial C}{\partial hidA_k} \frac{\partial hidA_k}{\partial hidB_i}\right)$$

$$\frac{\partial C}{\partial W^{(1,j,i)}} = \frac{\partial C}{\partial hidB_i} \frac{\partial hidB_i}{\partial W^{(1,j,i)}}$$

Back-propagate error signal to get derivatives for learning



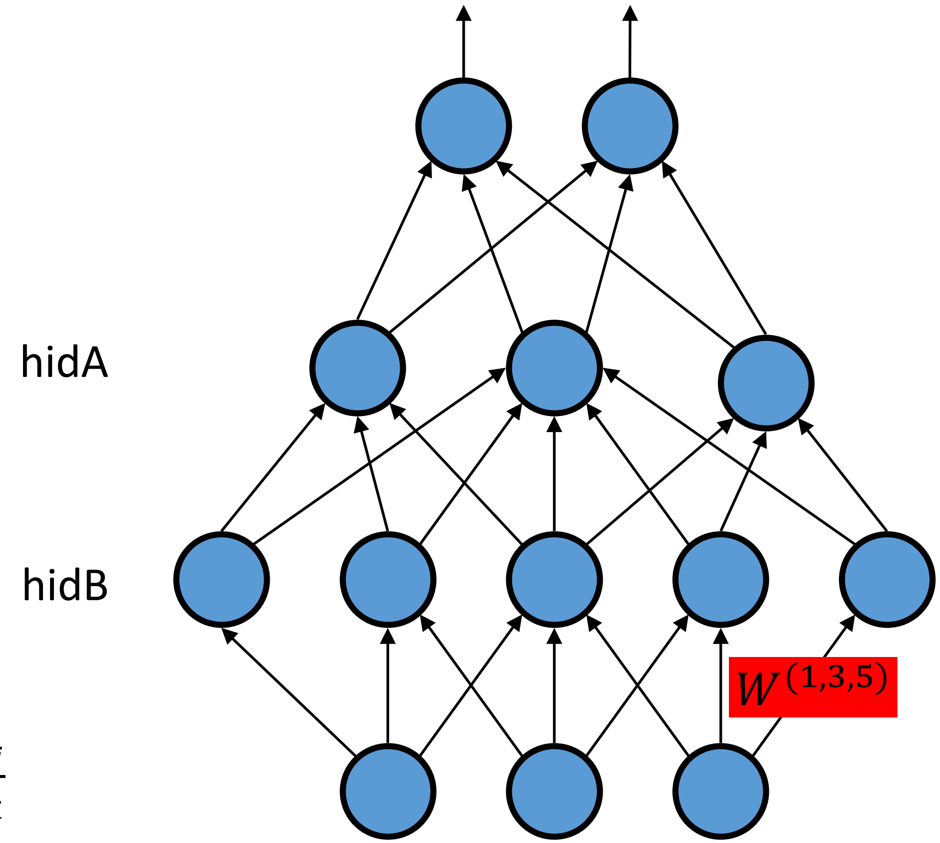
Another view of Guided Backpropagation



$$hidB_i = ReLU(\sum_j W^{(1,j,i)} x_j)$$

$$hidA_i = ReLU(\sum_j W^{(2,j,i)} hidB_j)$$

$$ReLU(z) = \begin{cases} z, z > 0 \\ 0, o/w \end{cases}$$



$$\frac{\partial Cost}{\partial hidB_i} = \sum_j \frac{\partial Cost}{\partial hidA_j} \frac{\partial hidA_j}{\partial hidB_i}$$

$$\frac{\partial hidA_j}{\partial hidB_i} = \frac{\partial ReLU(\sum_{j\prime} W^{(2,j\prime,i)} hidB_j)}{\partial hidB_i} = \begin{cases} W^{(2,j,i)}, \sum_{j\prime} W^{(2,j',i)} hidB_j > 0 \\ 0, o/w \end{cases}$$

Guided backprop: set $\frac{\partial hidA_j}{\partial hidB_i}$ to 0 if it's negative, and proceed with backprop computation