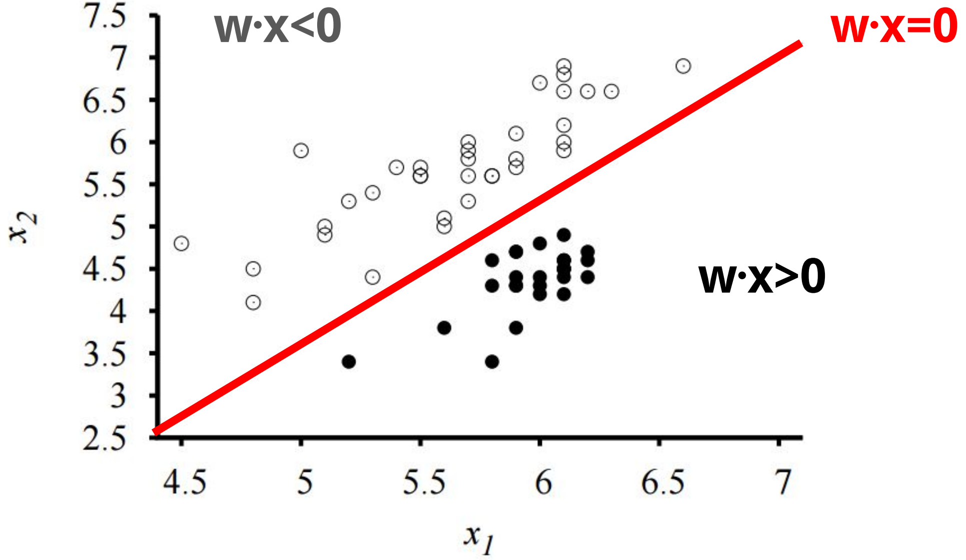
4700: Neural Networks (continued)

[Some slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]

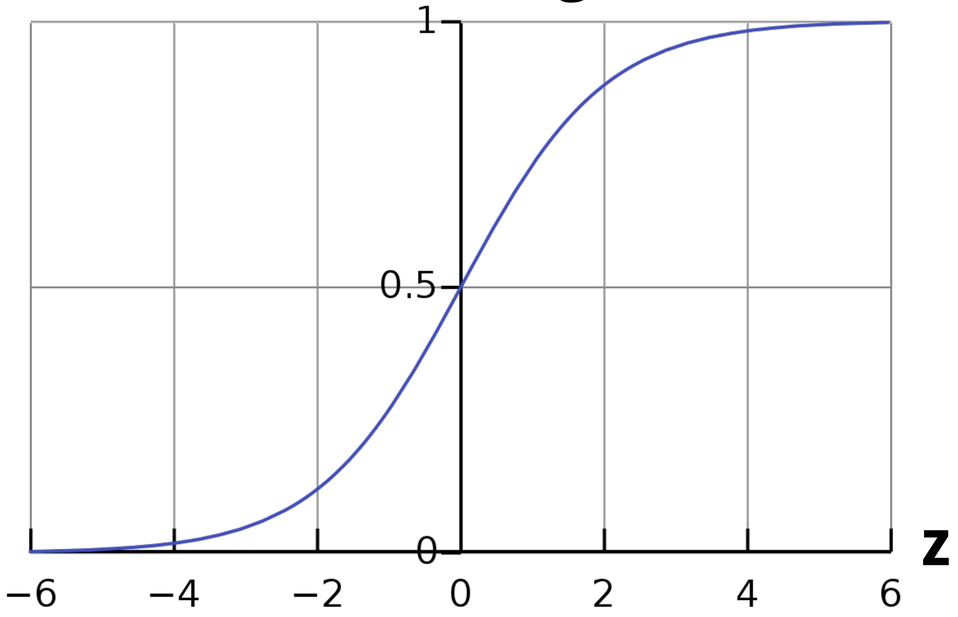


w·x<0
w·x=0
w·x>0
7.5
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
4.5
5
5.5
6
6.5
7
x_2
x_1

Sigmoid(0)=0.5
As z->+∞, Sigmoid(z)->1
As z->-∞, Sigmoid(z)->0

$$\text{Sigmoid}(z) = \frac{1}{1+e^{-z}}$$

Sigmoid(z)



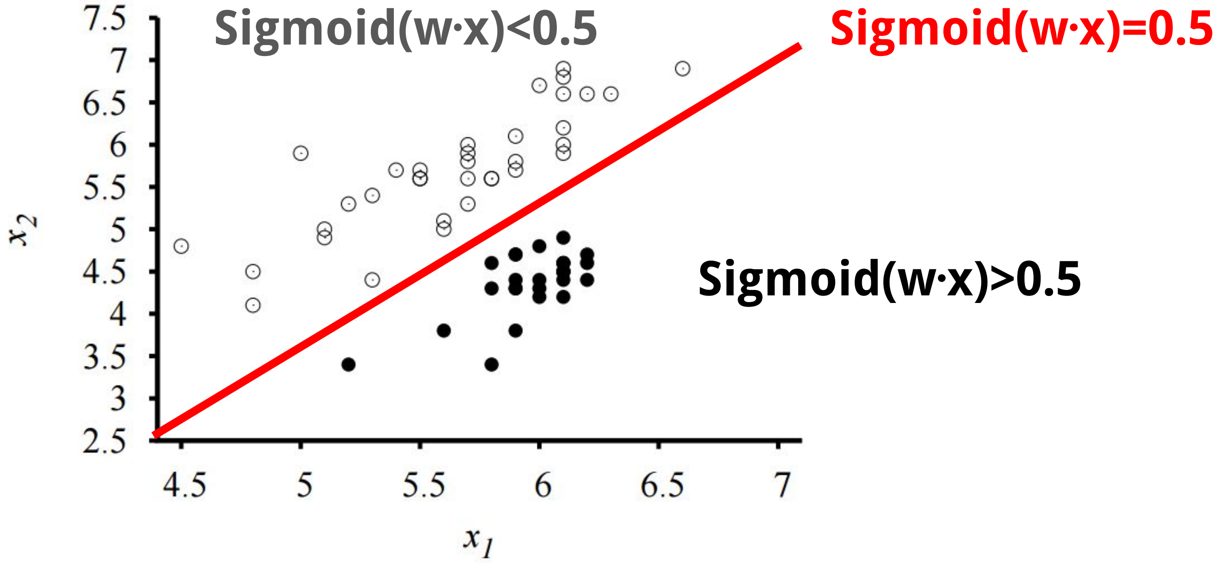
Logistic Regression

P(y=+1 | x, w) = Sigmoid(w·x)

Best w?

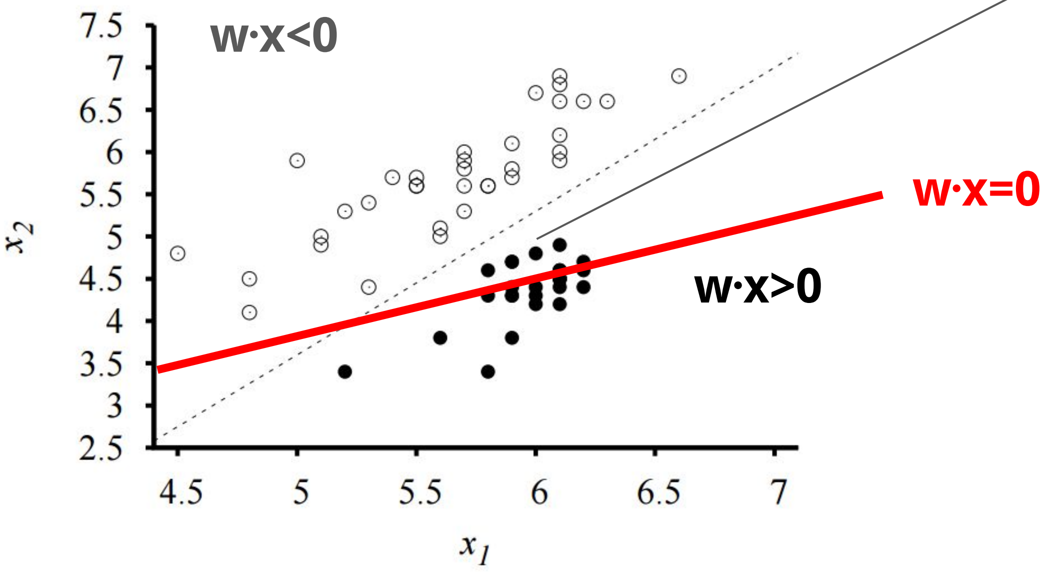
- Maximum likelihood estimation:

$$\max_{w} \quad ll(w) = \max_{w} \quad \sum \log P(y^{(i)}|x^{(i)}; w)$$



Sigmoid(w·x)<0.5
Sigmoid(w·x)=0.5
Sigmoid(w·x)>0.5
7.5
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
4.5
5
5.5
6
6.5
7
x_2
x_1

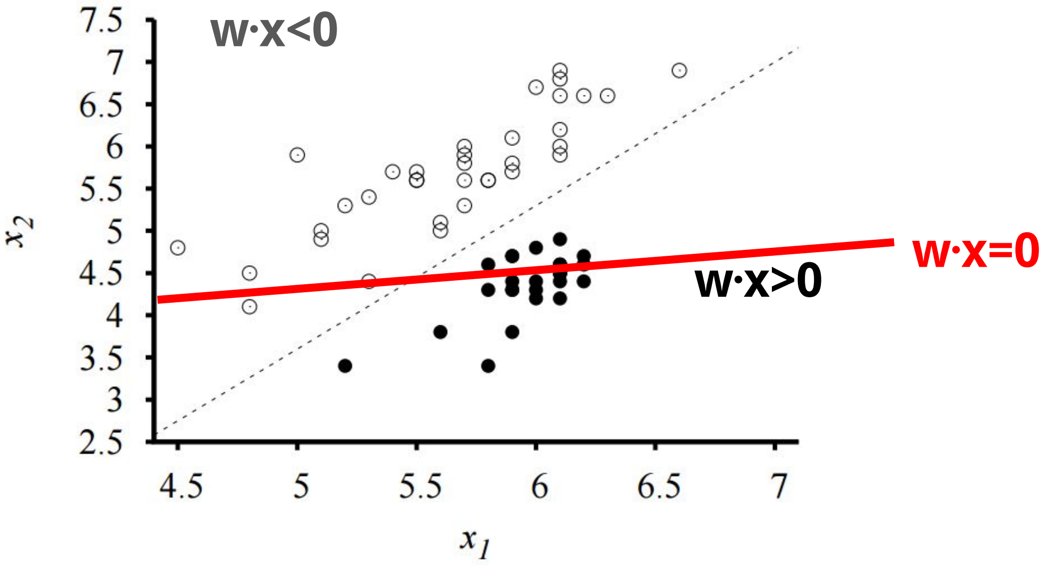
Classification - What error are we minimizing?



These misclassified points increase our error

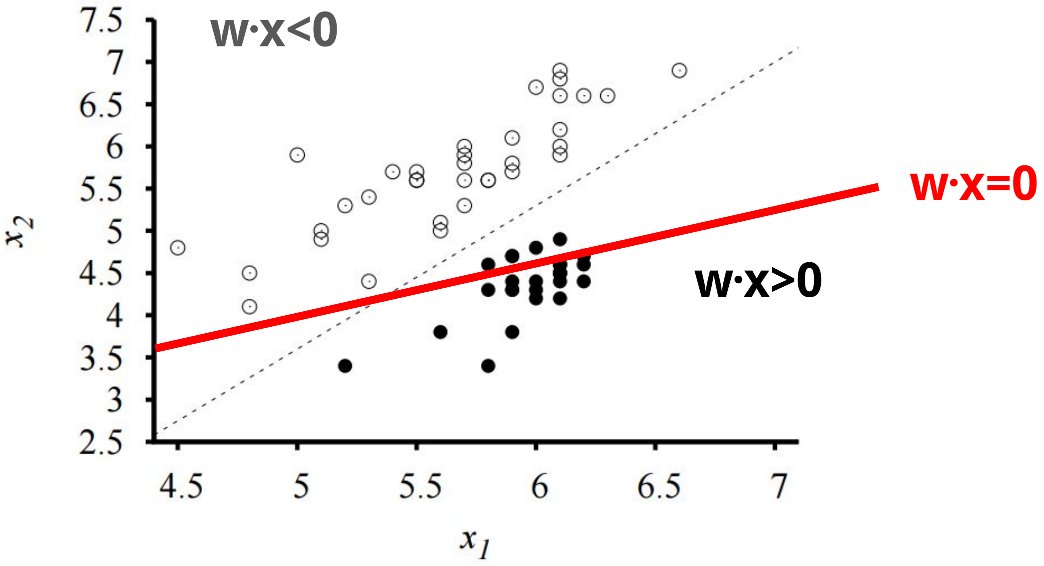
Classification - What error are we minimizing?

Yikes, even worse!

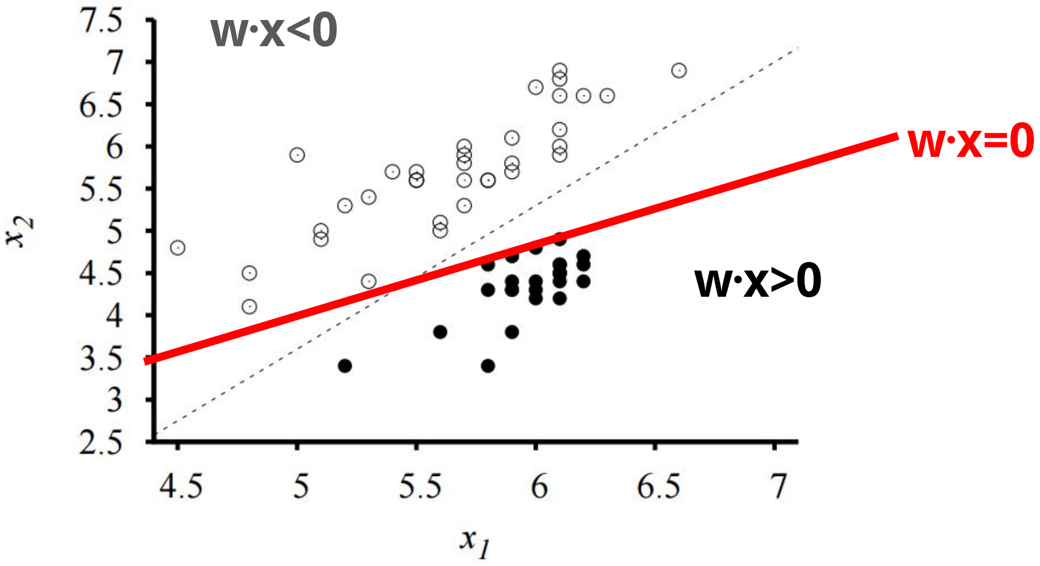


Classification - What error are we minimizing?

A bit better



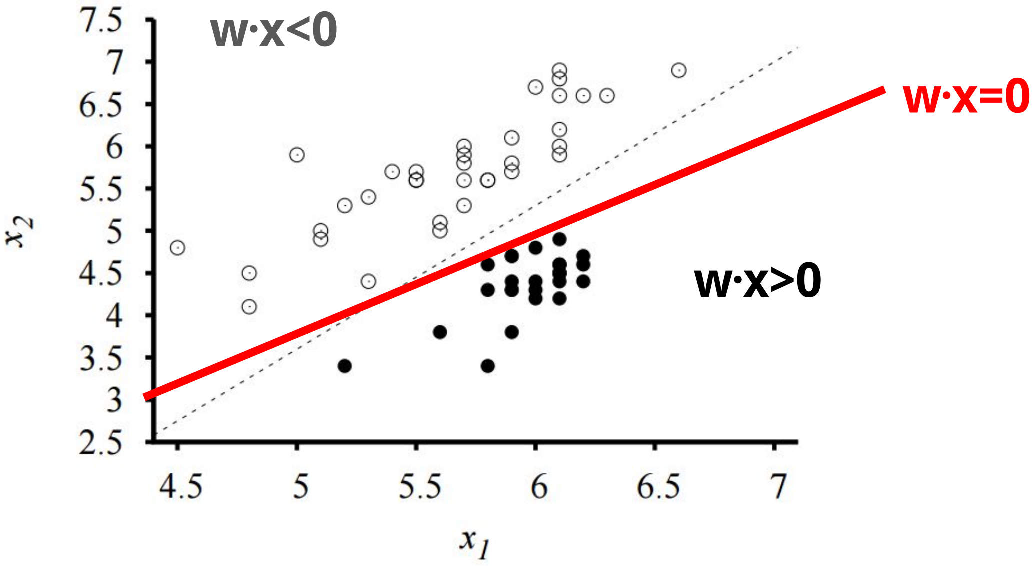
Classification - What error are we minimizing?



Perfectly classifies, but can it be improved?

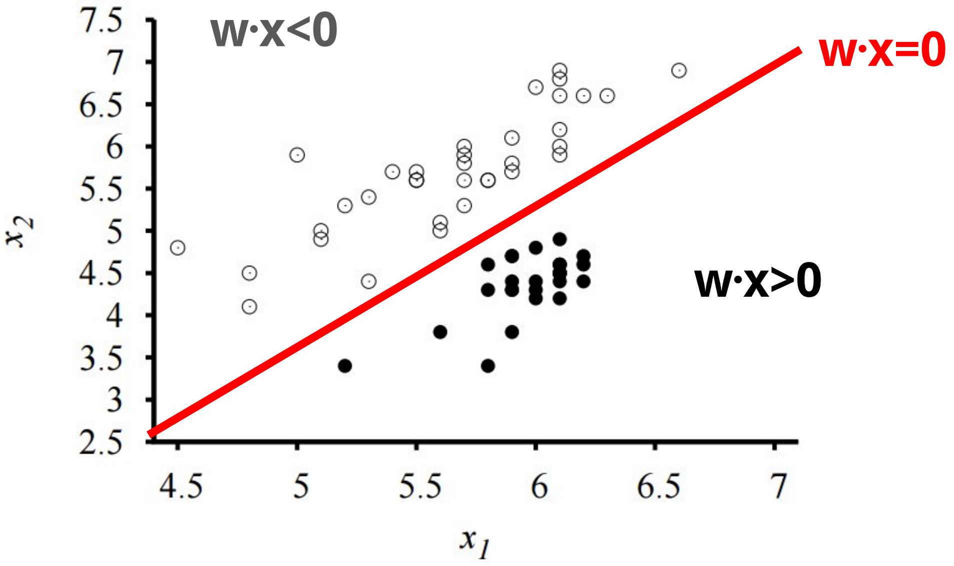
Classification - What error are we minimizing?

Getting better…



Classification - What error are we minimizing?

Perfect!



Local search for weight vector

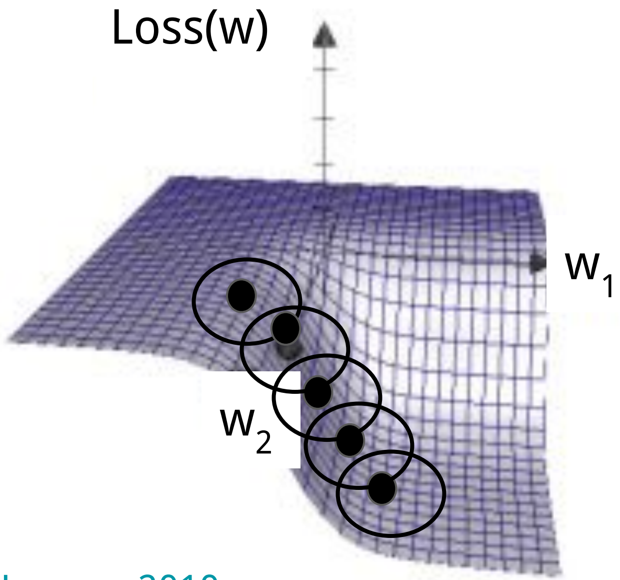


Figure from [Chaudhuri & Solar-Lezama 2010]

Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider: $g(w_1, w_2)$

  - Updates:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

  - Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

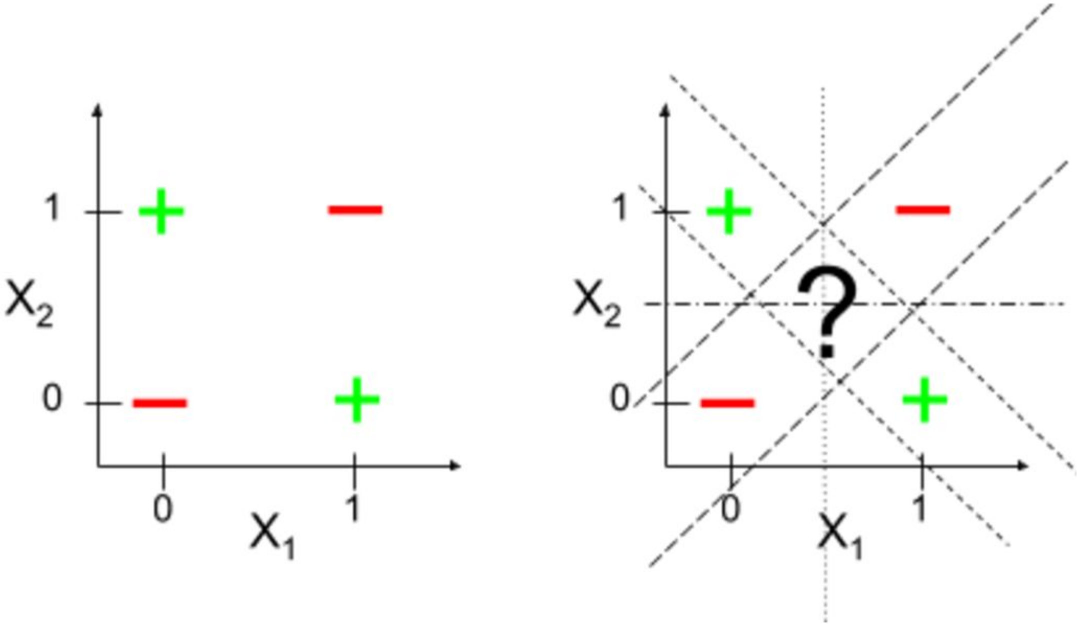
with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ **= gradient**

What can logistic regression learn?

Lines*

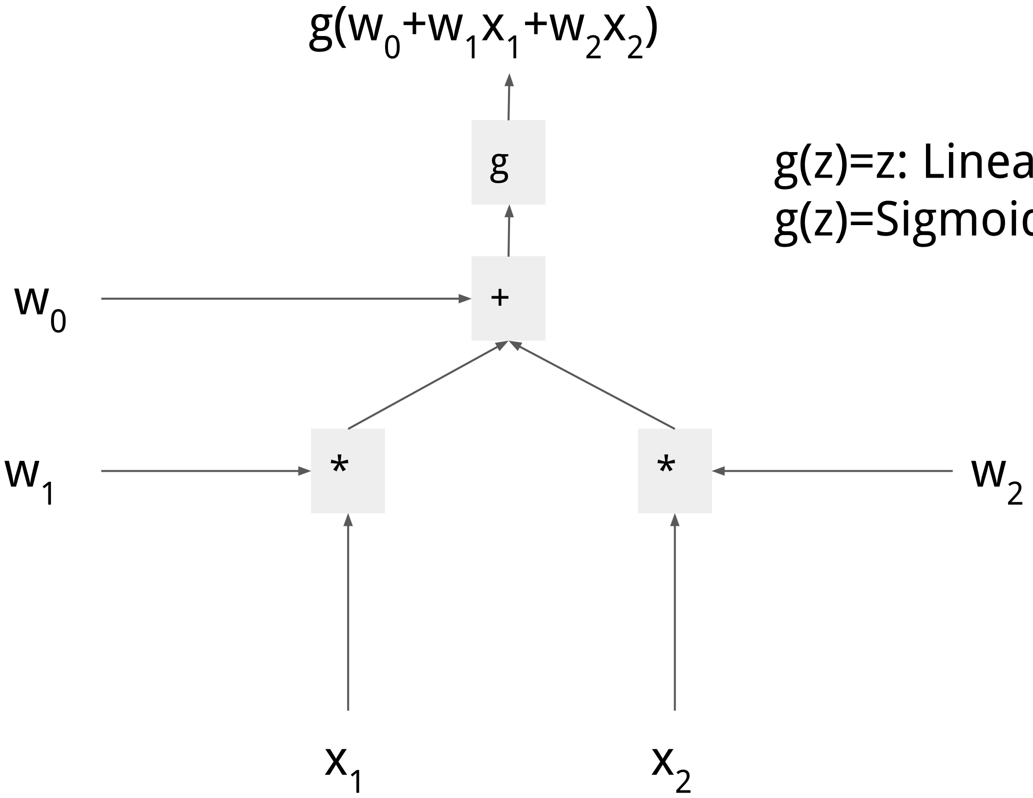
*in high dimensional spaces

Richer functions – Can a linear classifier fit this data?



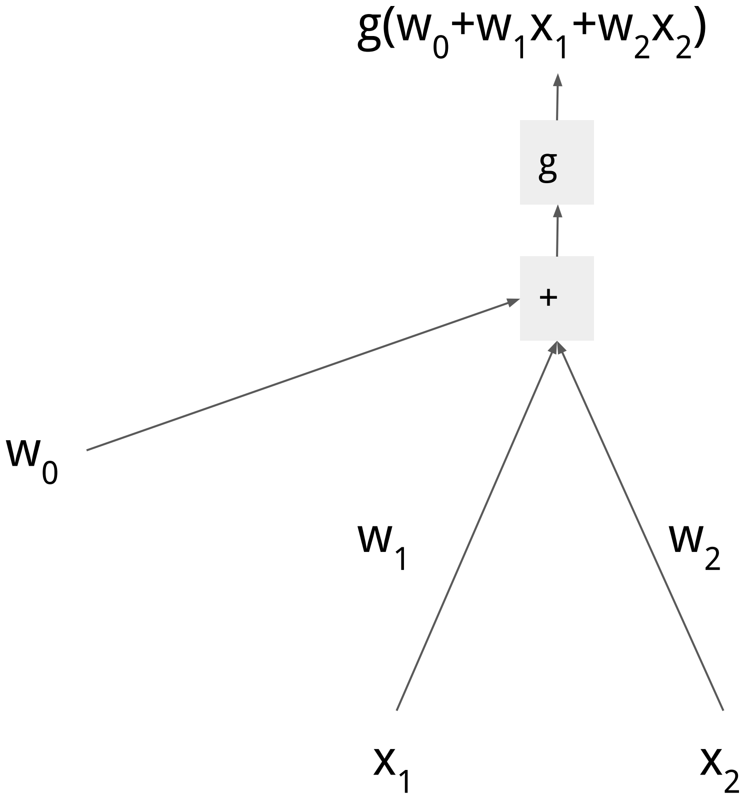
(https://medium.com/@claude.coulombe/the-revenge-of-perceptron-learning-xor-with-tensorflow-eb52cbdf6c60)

A different view of linear models



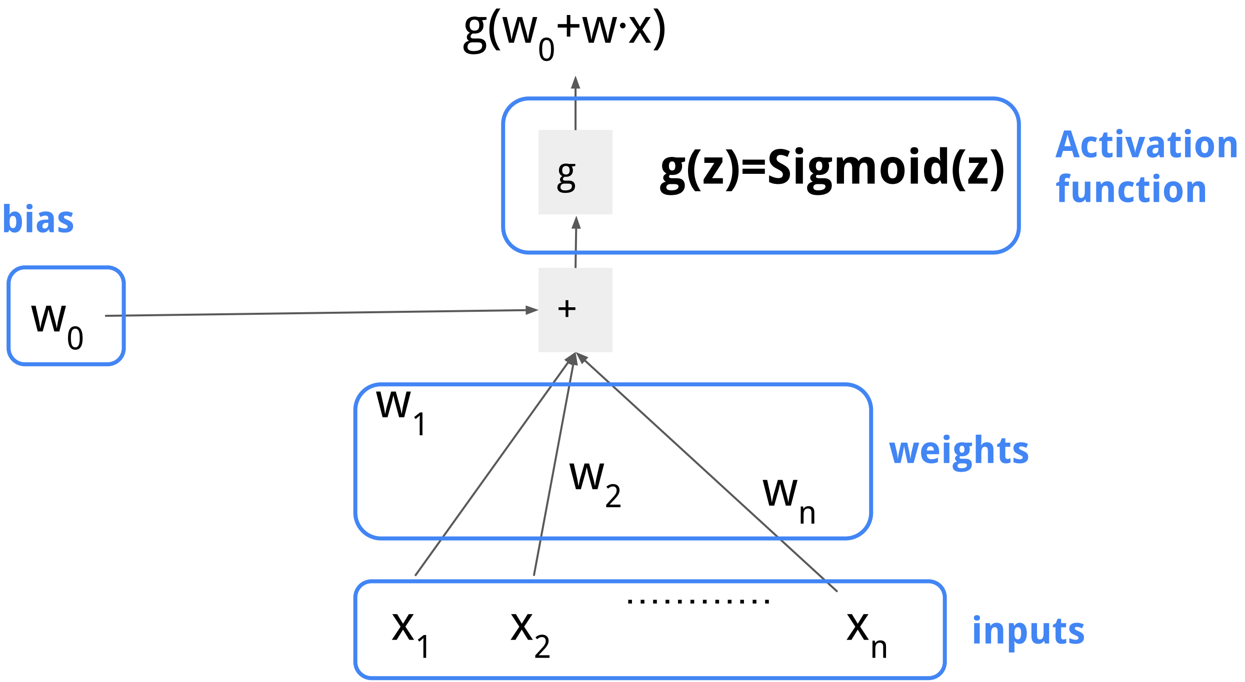
$g(z)=z$: Linear Regression
$g(z)=\text{Sigmoid}(z)$: Logistic Regression

A different view of linear models



g(z)=z: Linear Regression
g(z)=Sigmoid(z): Logistic Regression

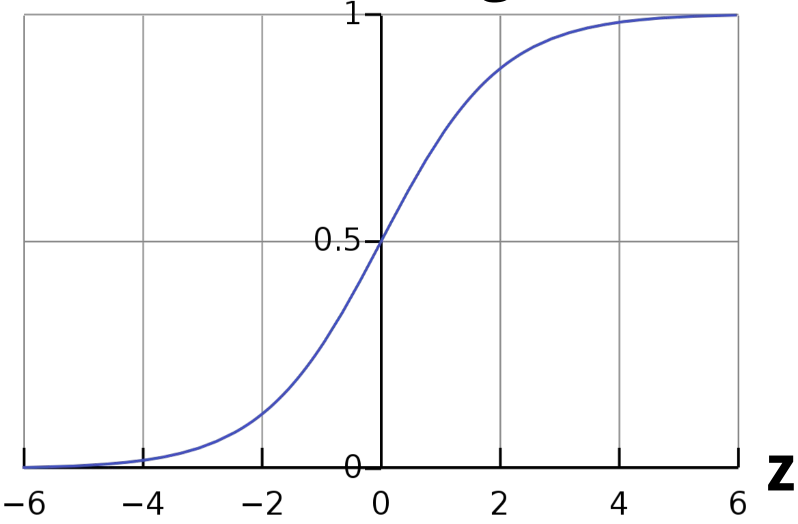
“Neuron”



$g(w_0+w\cdot x)$
g
g(z)=Sigmoid(z)
Activation function
bias
w_0
+
w_1
w_2
w_n
weights
x_1 x_2 ………… x_n
inputs

Nonlinear Logistic Activation +Linear Combination

Sigmoid(z)



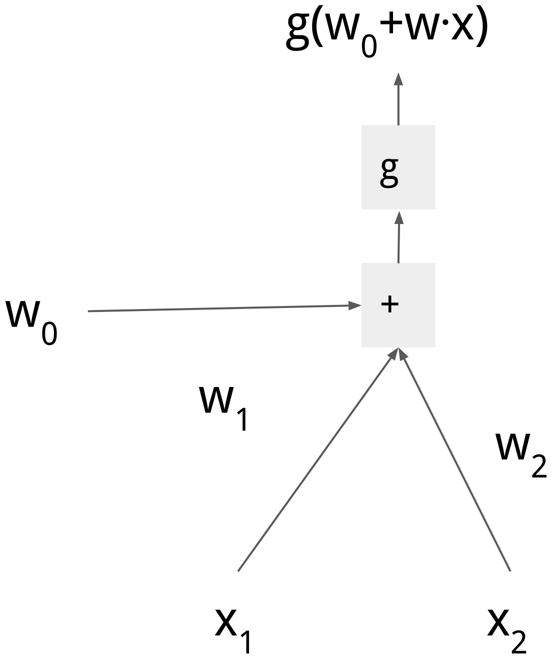
z=weighted input to neuron

$z = w_0 + w \cdot x$

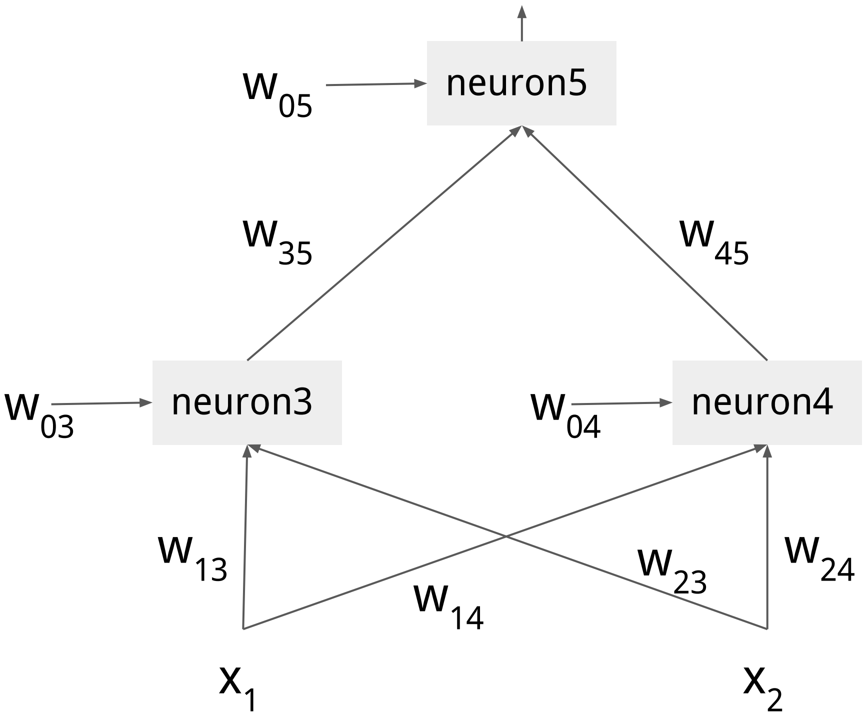
Bias: threshold for neuron "firing"

Weights: positive/negative? excitatory/inhibitory?

More than one neuron

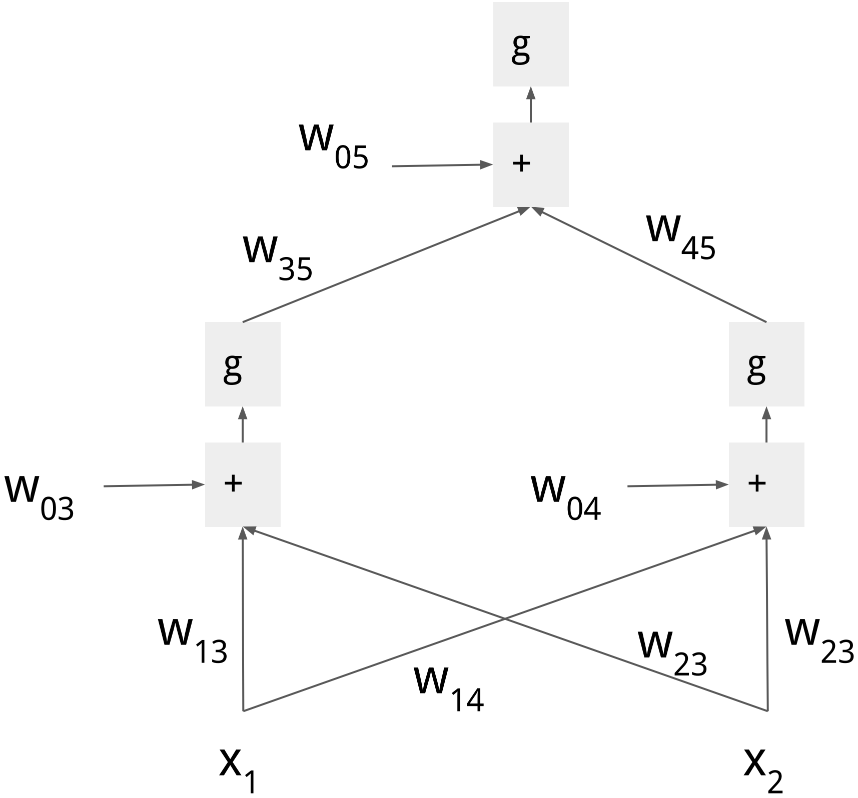


More than one neuron



Bias for neuron i: w_{0i}
Weight from i->j: w_{ij}

More than one neuron

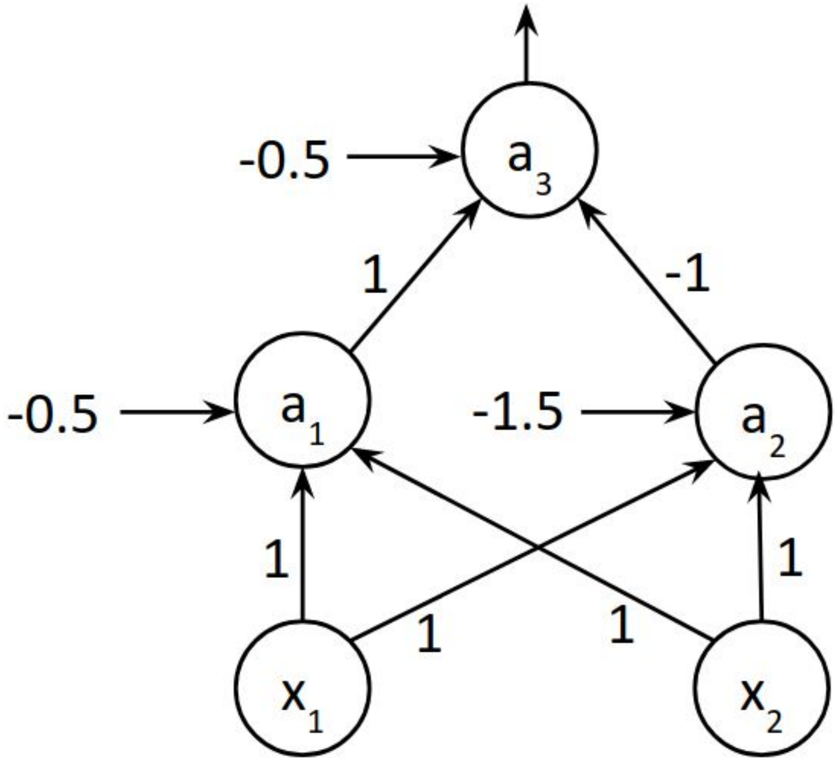


Bias for neuron i: w_{0i}
Weight from i->j: w_{ij}

Does it solve xor?

xor(0,0)=0 xor(1,0)=1

xor(0,1)=1 xor(1,1)=0



a_1=Logistic(-0.5+1*x_1+1*x_2)
~ is $x_1+x_2-0.5 > 0$
~ **OR** x_1 x_2
a_2=Logistic(-1.5+1*x_1+1*x_2)
~ is $x_1+x_2-1.5 > 0$
~ **AND** x_1 x_2
a_3=Logistic(-0.5+1*a_1-1*a_2)
~ is $a_1-a_2-0.5 > 0$
~ **AND** a_1 (**NOT** a_2)

Does it solve the problem of needing feature extractors?

feature vector recap

Actual thing we want to classify

FEATURE EXTRACTOR

“feature vector”

$x_n \in \mathbb{R}^D$

$x_n = (x_{n1}, x_{n2}, x_{n3}, \ldots, , x_{nD})$

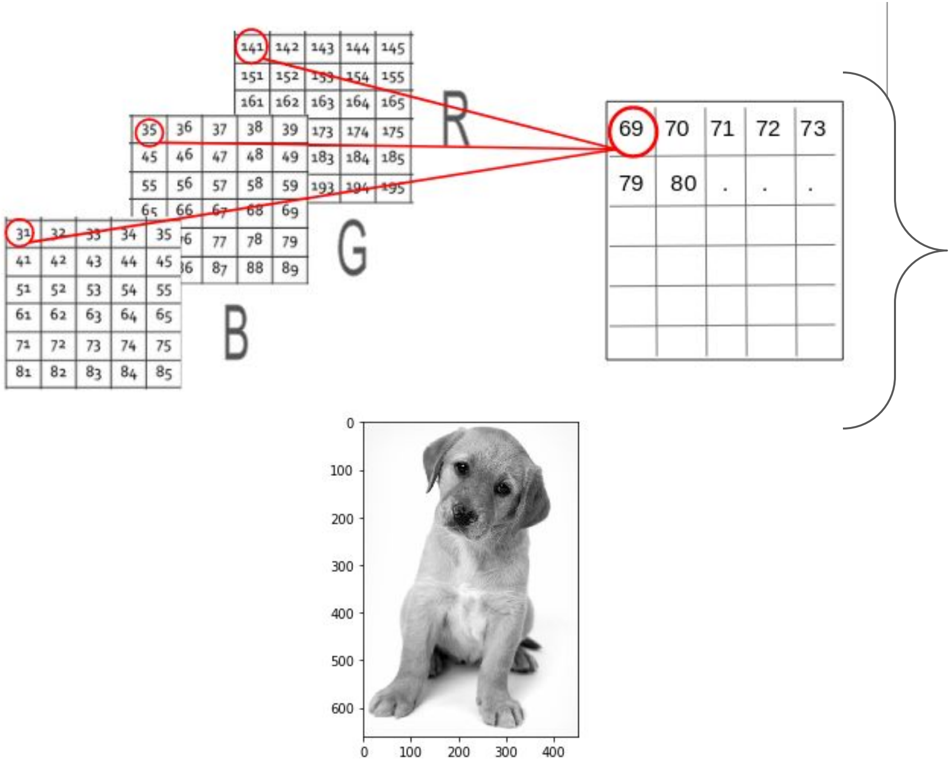
f

Is it dog or not

y

Example Image Feature Extractor

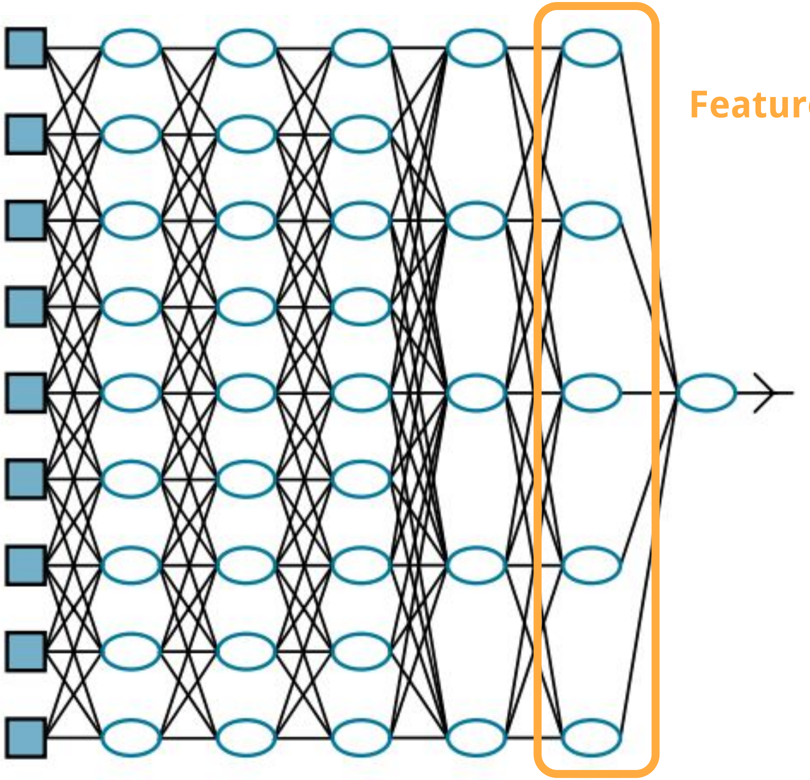
Colour Image



Flatten into
1d array

source

Neural Net Implicitly Does a Feature Extractor



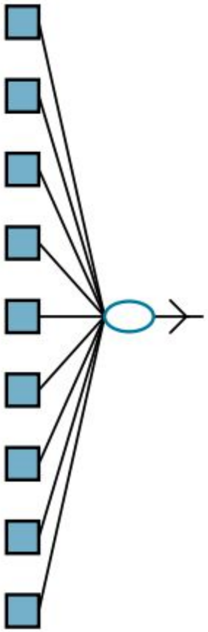
Features of x

Output neuron:
Sigmoid(w_0+w·f)

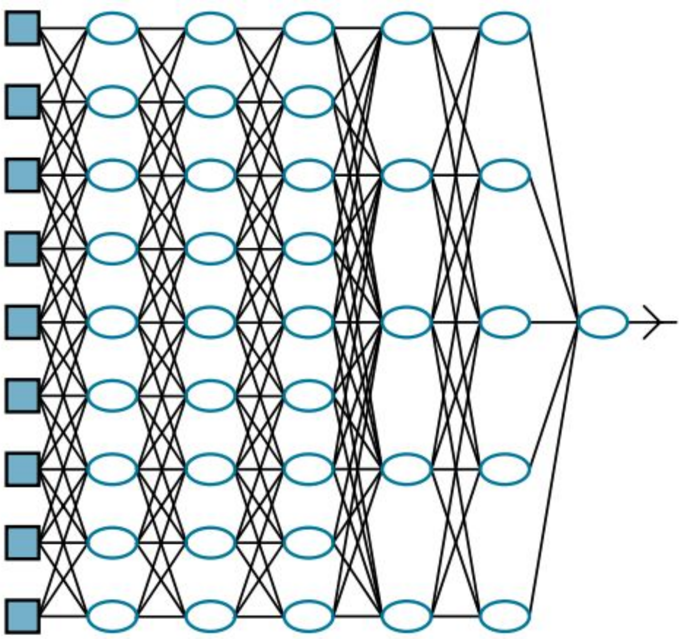
Inputs: x

Second-to-last neurons: features

Neural Net Structure



Linear Classifier or Regressor

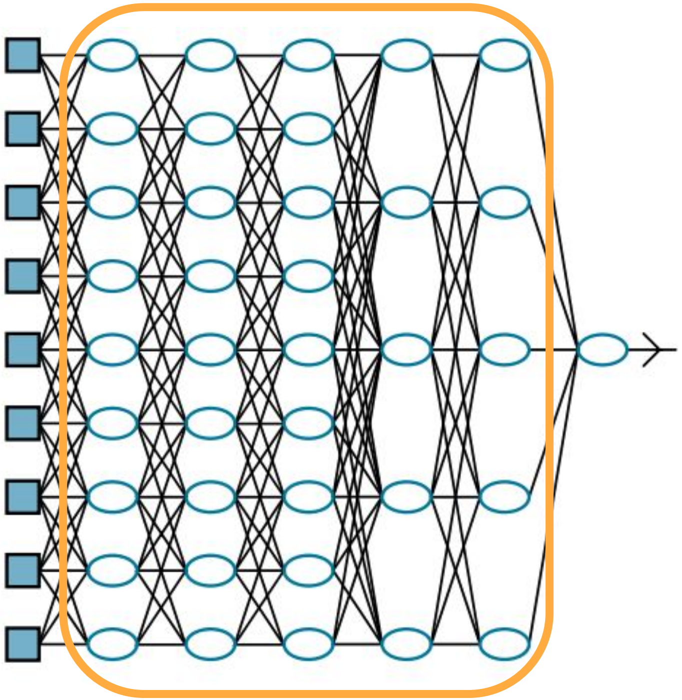


Neural Network

Neural Net Structure

Many ***layers*** of neurons

Input Neurons



Output Neurons

Hidden Neurons

Neural Networks Summary so far

Biologically inspired computation model

1 logistic regression ~ 1 neuron

Wire up many different neurons, arranged in layers

Neural network output depends on:

1. Input, from the training data
2. Weights and biases, which we'll learn

output = neural_network(input, w)

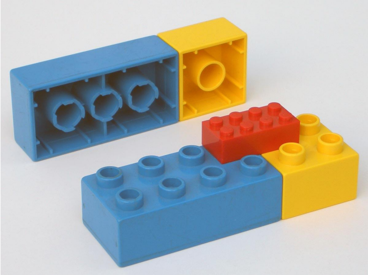
Real-valued, can be a single number, a vector of numbers, a 2D array of numbers, …

Network input

"Parameters" -- a (typically huge) vector of real numbers
Weights and biases

Differentiable:
Can calculate:
d/dw neural_network(input, w)

This is a neural network with parameters (w_1,w_2)



$$
\boxed{\begin{array}{l} \text{output}_2 = \text{neural\_network}_2(\text{output}_1, w_2) \\ \text{output}_1 = \text{neural\_network}_1(\text{input}, w_1) \end{array}}
$$

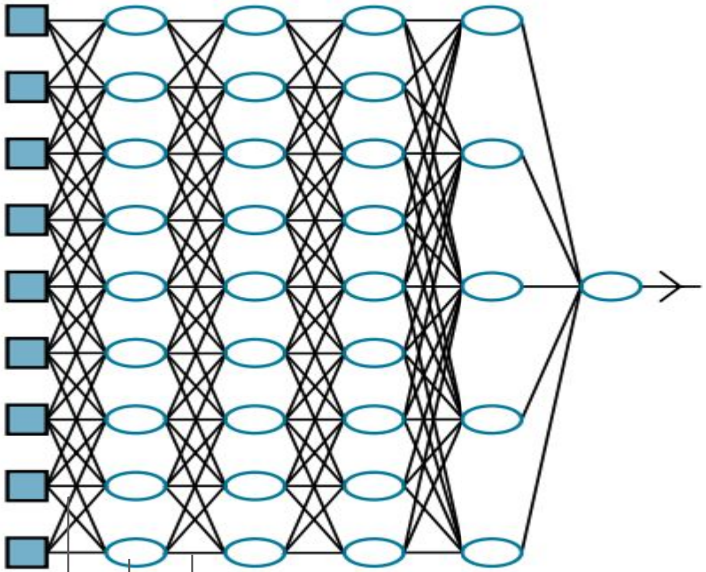
Real-valued, can be a single number, a vector of numbers, a 2D array of numbers, …

Network input

"Parameters" -- a (typically huge) vector of real numbers
Weights and biases

Differentiable:
Can calculate:
d/dw neural_network(input, w)

Layers =
Function Composition



Inputs x

Output y

Layer 1 parameters: w1

Layer1 output: a1

Layer 2 parameters: w2

Layer 2 output:
a2=Layer2(w2,a1)
a2=Layer2(w2,Layer1(w1,x))

Neural Network Supervised Learning

Training data: $(x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N)$

Weights (and biases): $w \in \mathbb{R}^d$

$$w = \underset{w \in \mathbb{R}^D}{\operatorname{argmin}} \sum_n \mathrm{Loss}(\mathrm{neural\_network}(w, x_n), y_n)$$

$$w = \underset{w \in \mathbb{R}^D}{\operatorname{argmin}} \ \mathrm{Loss}(w)$$

Loss is a smooth, differentiable function of w

=> gradient descent

Gradient Ascent

- consider: $g(w_1, w_2)$

  - Updates:

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

  - Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$

Slide taken from ai.berkeley.edu

Gradient Descent Challenges

1. Taking a bunch of derivatives of a complicated function

Gradient Descent Challenges

1. Taking a bunch of derivatives of a complicated function

2. Gradient descent takes lots of little steps.

   Each step requires looking at the entire training set–which might be huge

Gradient Descent Challenges

1. Taking a bunch of derivatives of a complicated function

   => backpropagation, automatic differentiation, pytorch, tensorflow

2. Gradient descent takes lots of little steps.

   Each step requires looking at the entire training set–which might be huge

   => stochastic gradient descent, see previous lecture

Next up

1. Taking a bunch of derivatives of a complicated function

   => backpropagation, automatic differentiation, pytorch, tensorflow

2. Gradient descent takes lots of little steps.

   Each step requires looking at the entire training set–which might be huge

   => stochastic gradient descent

Neural network layers = Function composition

$$L_2(w_2, L_1(w_1, x))$$

Layer 2

Weight in layer 2

Layer 1

Weight in layer 1

input

Neural network layers = Function composition

$$\underbrace{\frac{\mathrm{d}}{\mathrm{d}w_1} L_2}_{\text{Layer 2}}(\underbrace{w_2}_{\text{Weight in layer 2}}, \underbrace{L_1}_{\text{Layer 1}}(\underbrace{w_1}_{\text{Weight in layer 1}}, \underbrace{x}_{\text{input}}))$$

Neural network layers = Function composition

$$\frac{\mathrm{d}}{\mathrm{d}w_1} L_2(w_2, L_1(w_1, x))$$

Layer 2

Weight in layer 2

Layer 1

Weight in layer 1

input

$$= \frac{\mathrm{d}L_2(w_2, L_1)}{\mathrm{d}L_1} \frac{\mathrm{d}L_1(w_1, x)}{\mathrm{d}w_1}$$

Modern Neural Network Libraries Do Backpropagation

```
# begin of forward computation
>>> w=torch.rand(6, 6, requires_grad=True)
>>> a3=logistic(w[0,3]+w[1,3]*x1+w[2,3]*x2)
>>> a4=logistic(w[0,4]+w[1,4]*x1+w[2,4]*x2)
>>> a5=logistic(w[0,5]+w[3,5]*a3+w[4,5]*a4)
>>> y=0.7
>>> loss=(a5-y)**2 # end of forward computation
>>> loss.backward() # run back propagation
>>> w.grad[3,5] # dLoss / d w_{35}
tensor(0.0570) # automatic differentiation
```

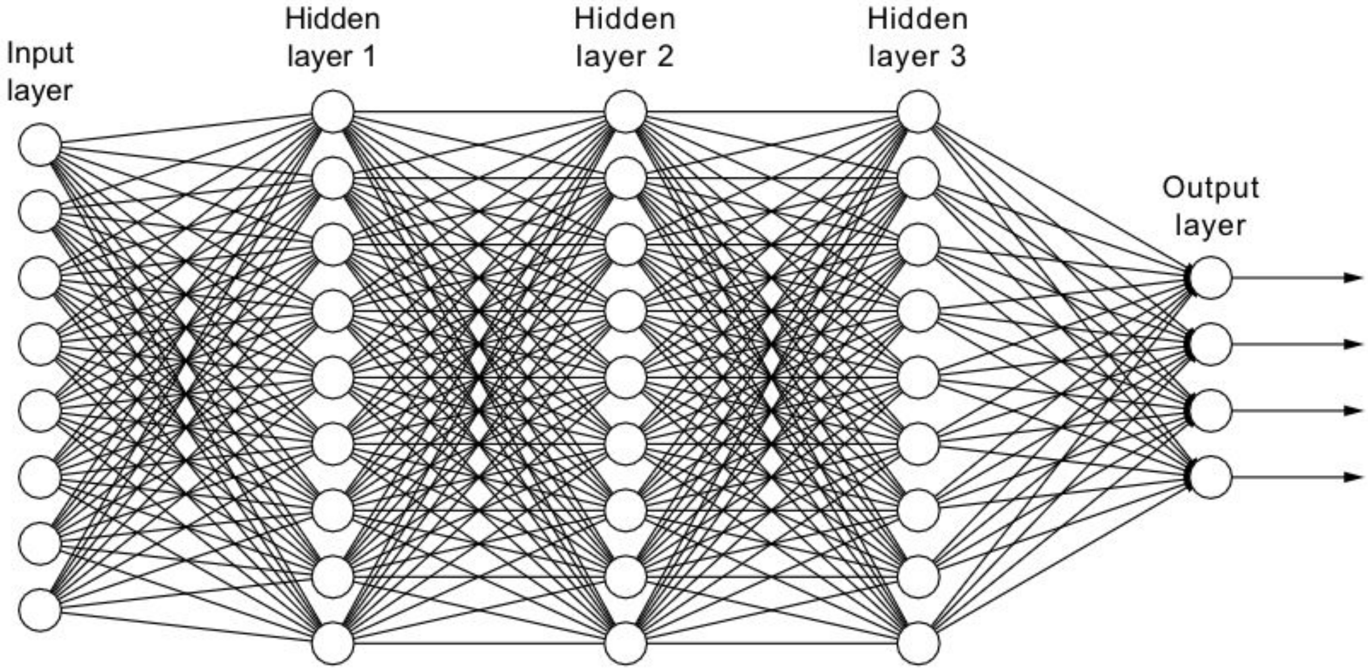
Learning Autodifferentiation

Implement it! https://sidsite.com/posts/autodiff/

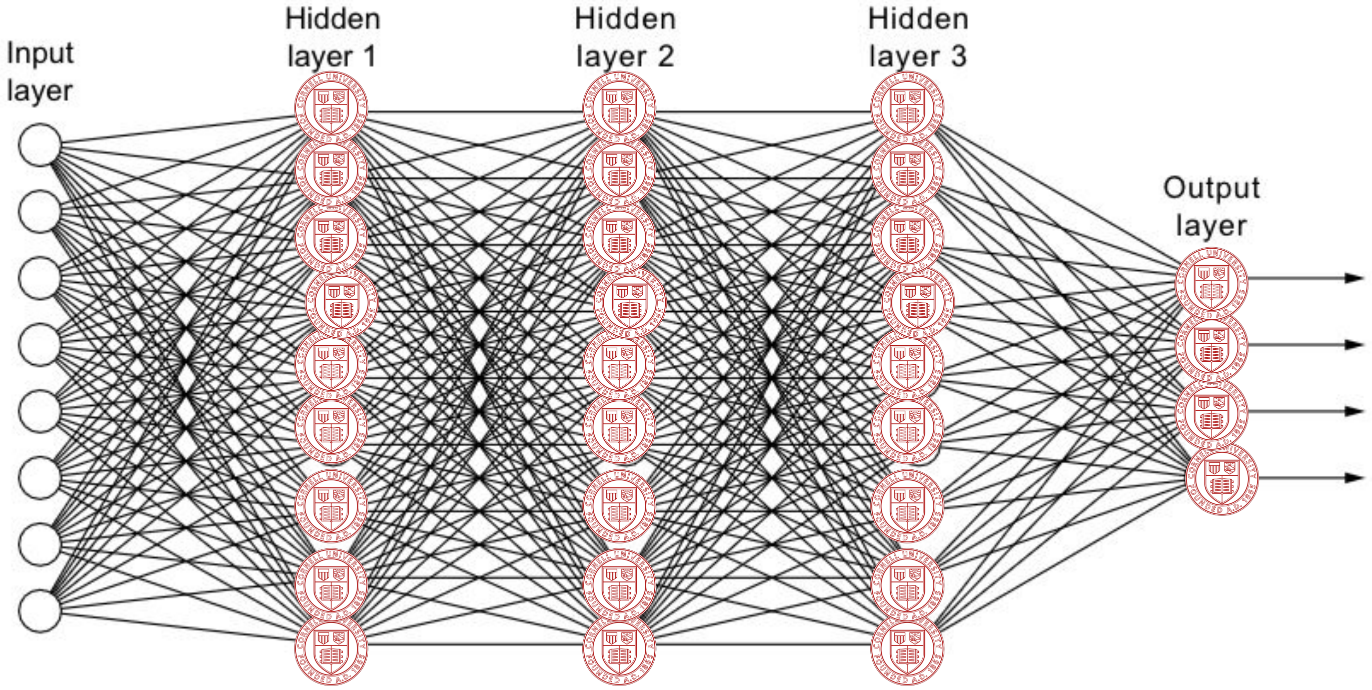
Slides at the end of this lecture cover neural network training /without/ automatic differentiation. The algebra is horrible.

Neural Net Architectures

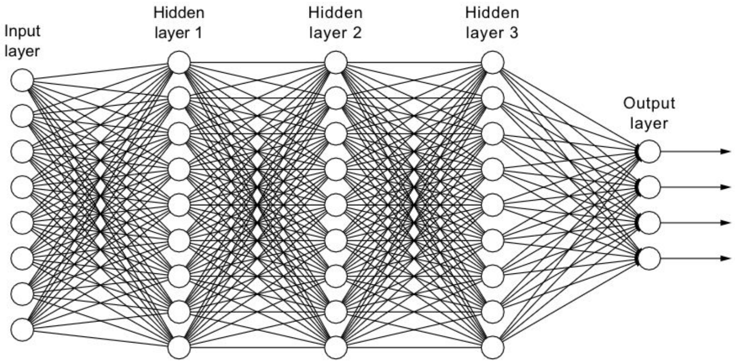
Multilayer Perceptron (MLP)



Multilayer Perceptron (MLP)



Multilayer Perceptron (MLP)



Advantages:

Doesn’t assume much about the inputs

Could permute the inputs and the learner would be oblivious

Disadvantages:

Doesn’t assume much about the inputs

Could permute the inputs and the learner would be oblivious

Specialized Neural Architectures:

Images
Text

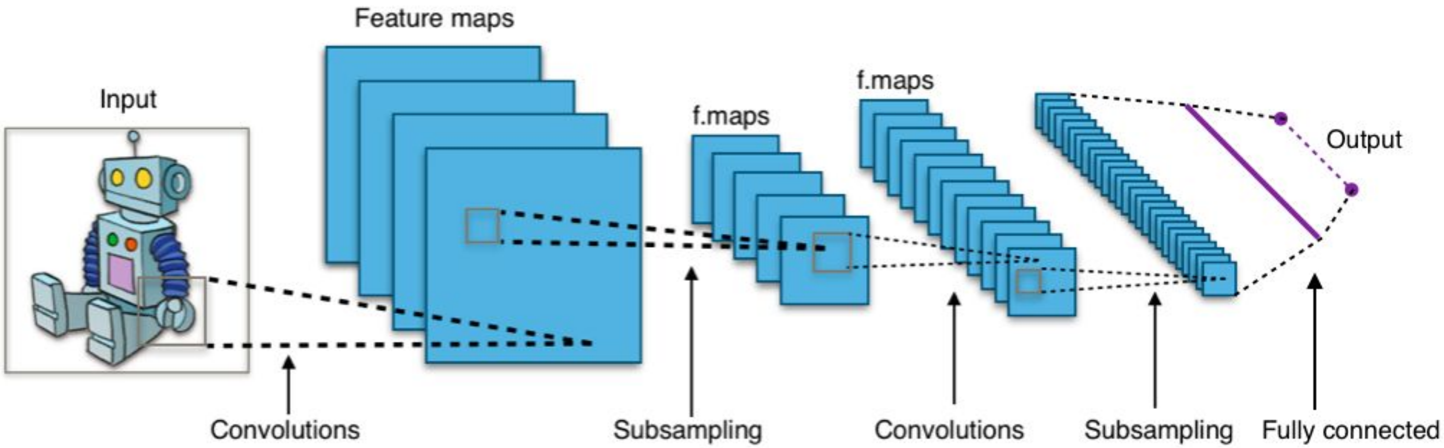
Specialized Neural Architectures:

Images (2D arrays)
Text (1D sequences)

Images (2D array)

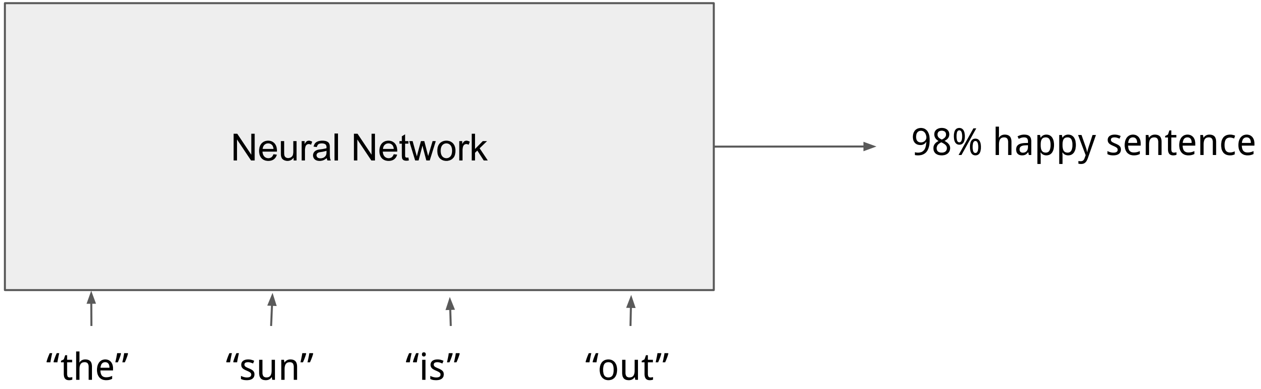
Convolutional neural networks

- Neuron that takes 5x5 grid from an image as input (“receptive field”)
- Make copies of it for every 5x5 window in image
- Assign them all the same weights
- Pooling layer: Is something present in *any* of the windows

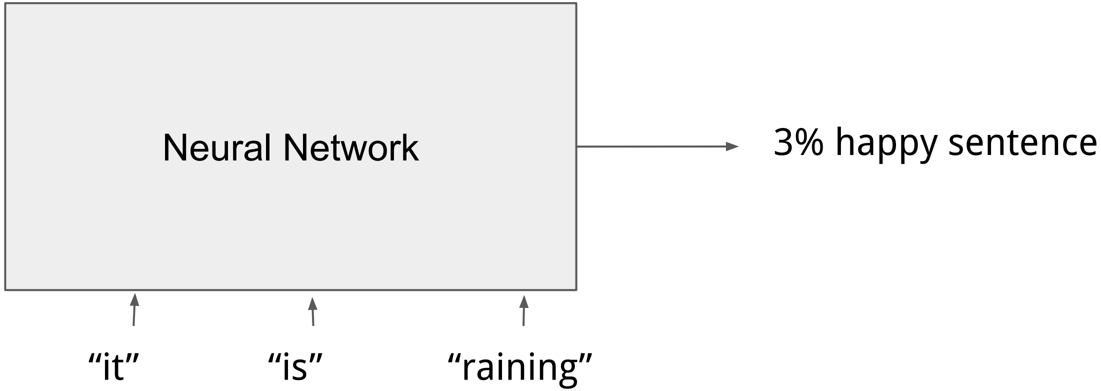


https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

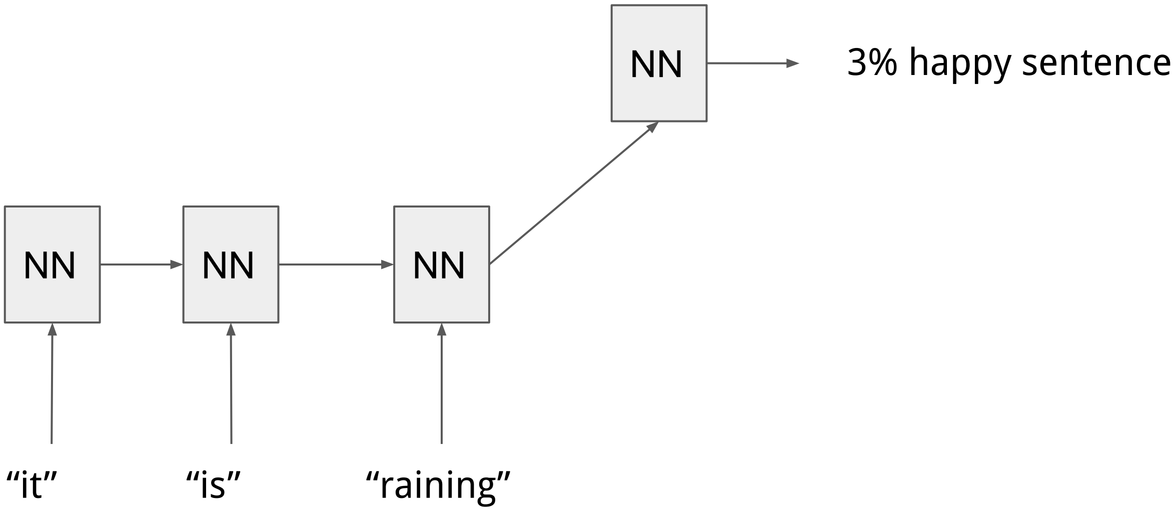
Text (1D Sequences)



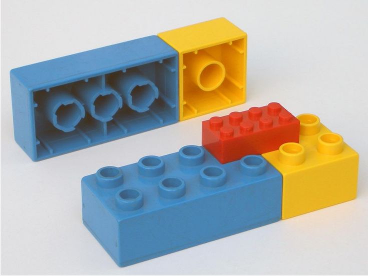
Text (1D Sequences)



Text (1D Sequences)



This is a neural network with parameters (w_1, w_2)



$$output_2 = neural\_network_2(output_1, w_2)$$
$$output_1 = neural\_network_1(input, w_1)$$

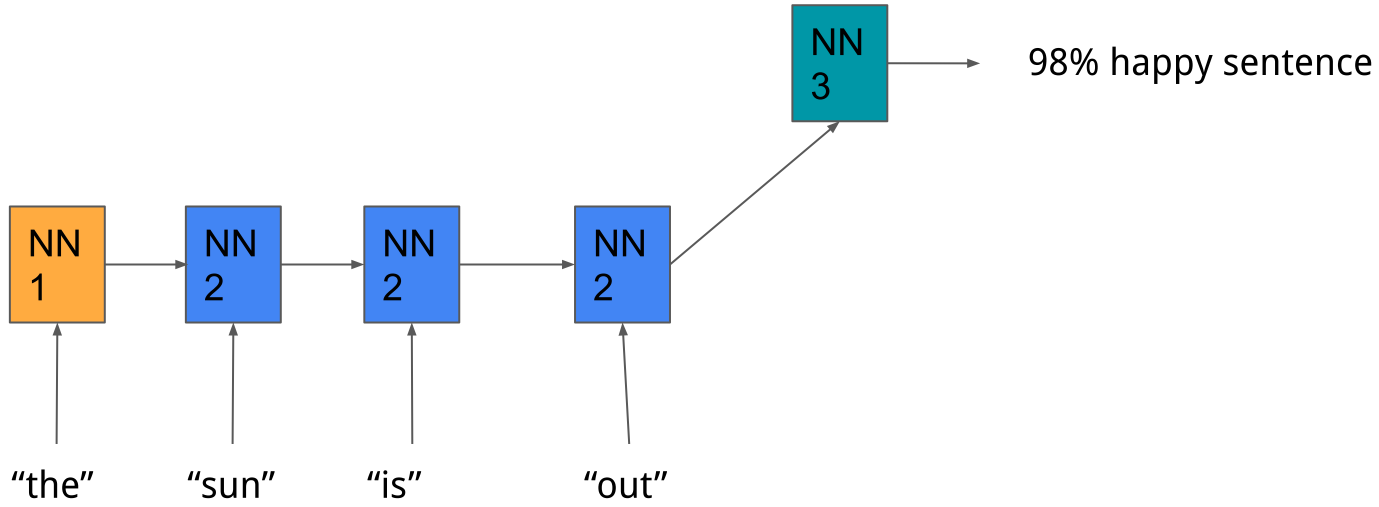
Real-valued, can be a single number, a vector of numbers, a 2D array of numbers, …

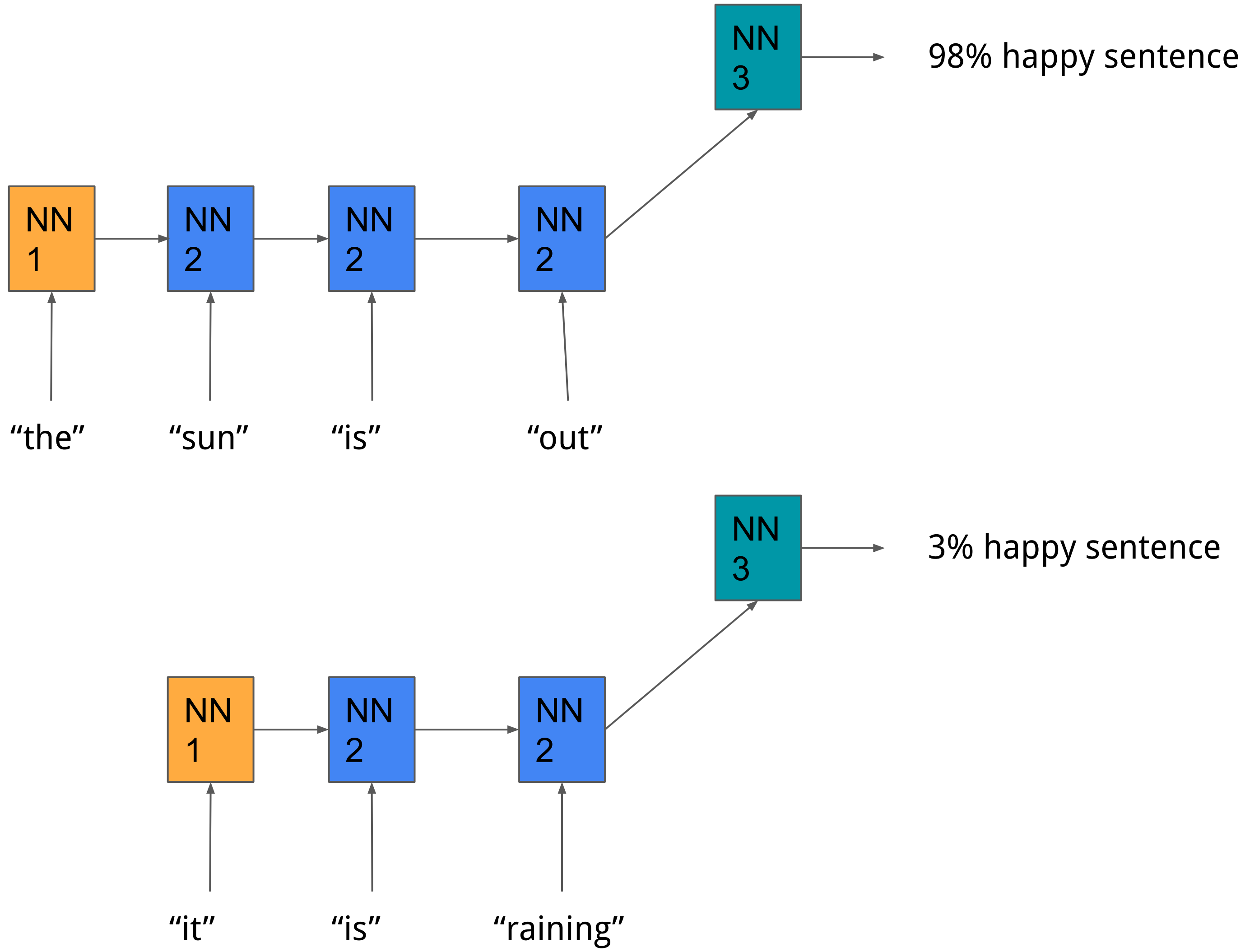
Network input

“Parameters” -- a (typically huge) vector of real numbers
Weights and biases

Differentiable:
Can calculate:
d/dw neural_network(input, w)

Text (1D Sequences)

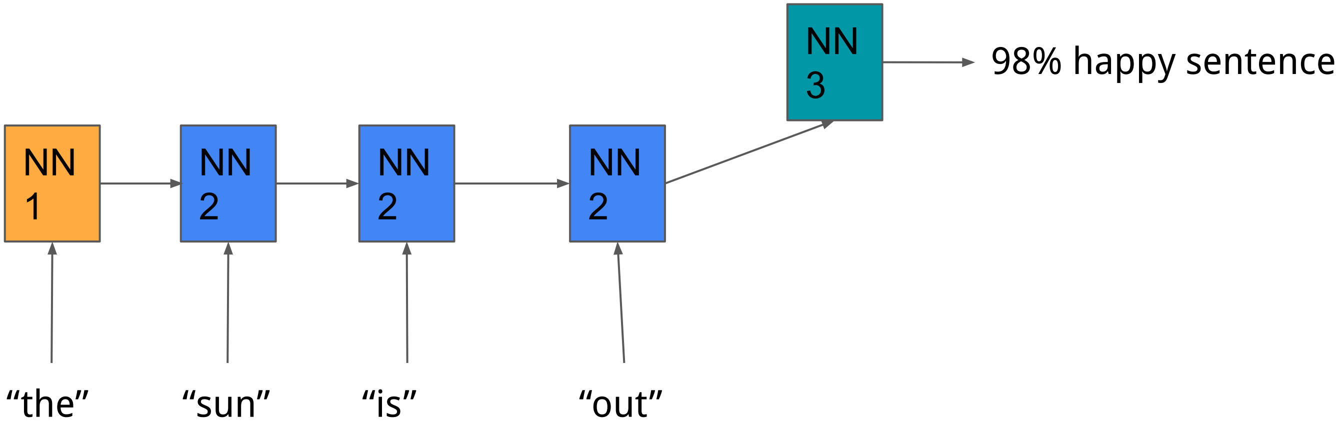



NN 3
98% happy sentence
NN 1
NN 2
NN 2
NN 2
"the"
"sun"
"is"
"out"
NN 3
3% happy sentence
NN 1
NN 2
NN 2
"it"
"is"
"raining"

Recurrent Neural Network

Further details:

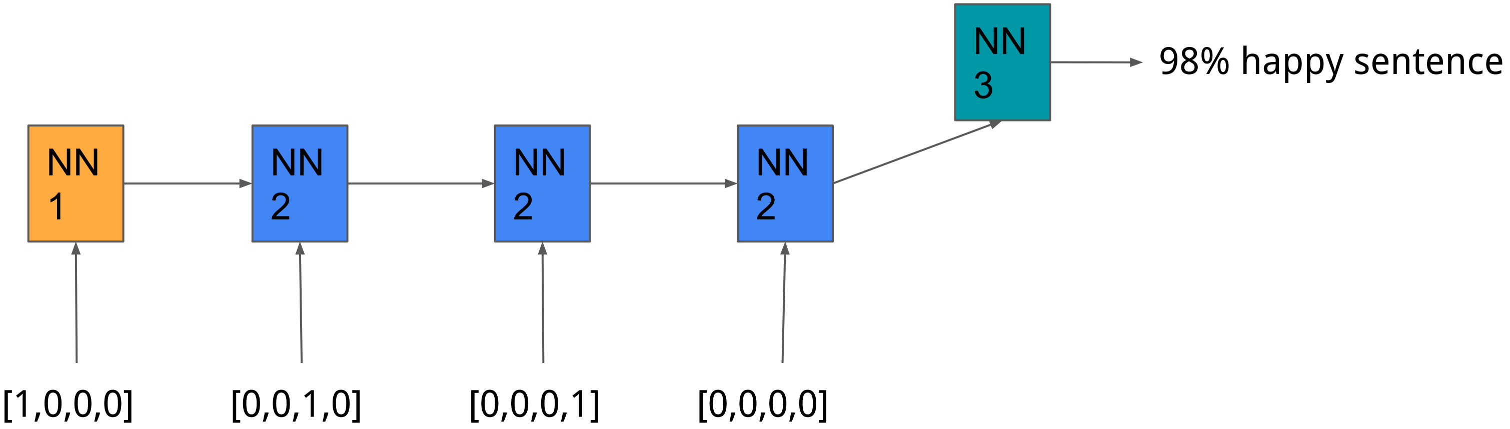
1. Each word is turned into a one-hot vector
2. Usually NN1 is just NN2 but with a dummy input fed in on the left
3. Usually NN3 is just a logistic regressor



Recurrent Neural Network

Further details:

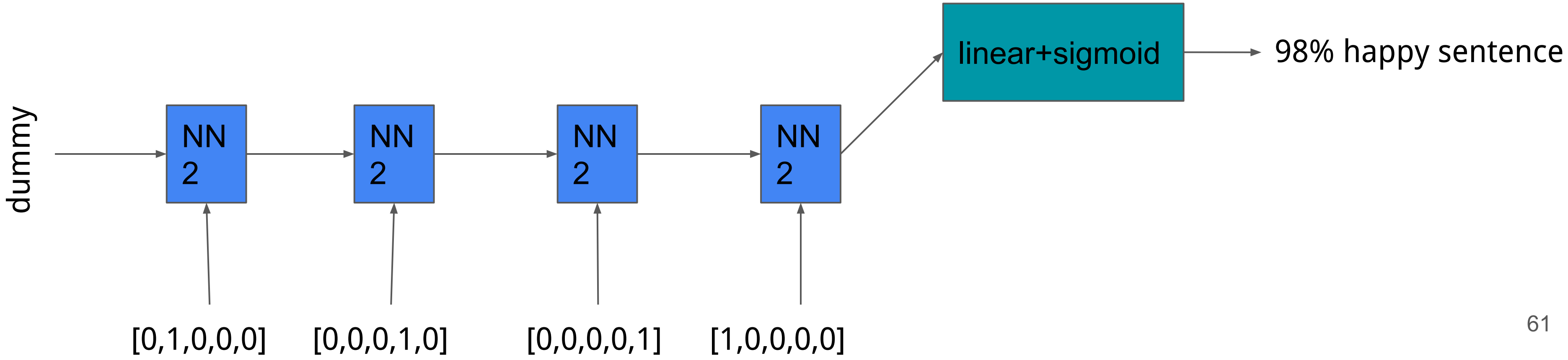
1. Each word is turned into a one-hot vector
2. Usually NN1 is just NN2 but with a dummy input fed in on the left
3. Usually NN3 is just a logistic regressor



Recurrent Neural Network

Further details:

1. Each word is turned into a one-hot vector
2. Usually NN1 is just NN2 but with a dummy input fed in on the left
3. Usually NN3 is just a logistic regressor



Summary, architecture

Convolutional Neural Network:

- Runs the same neural net computation over sliding windows of the image
- Downsamples the input
- Repeatedly alternates sliding windows and down sampling

Recurrent Neural Network:

- Runs the same neural net computation over each element of a sequence
- Feeds the output of earlier inputs into later input computations

Bonus Slides on backpropagation

Automatic differentiation makes it so that you don’t have to do the computations on the following slides.

None of this is on any of your homework or exams.

It should make you appreciate automatic differentiation, however.

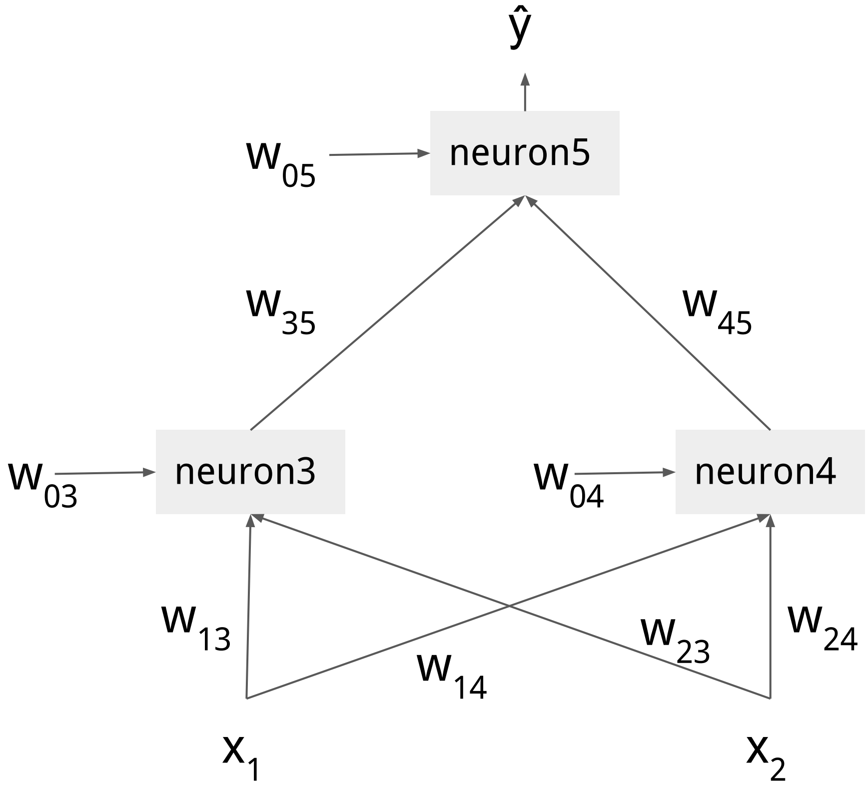
Chain Rule

$$\frac{\mathrm{d}}{\mathrm{d}x} f(g(x)) = f'(g(x)) \frac{\mathrm{d}}{\mathrm{d}x} g(x)$$

$$z = g(x)$$

$$y = f(z)$$

$$\frac{\mathrm{d}}{\mathrm{d}x} y = \frac{\mathrm{d}}{\mathrm{d}x} f(g(x)) = \frac{\mathrm{d}f(z)}{\mathrm{d}z} \frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}f(z)}{\mathrm{d}z} \frac{\mathrm{d}g(z)}{\mathrm{d}x}$$

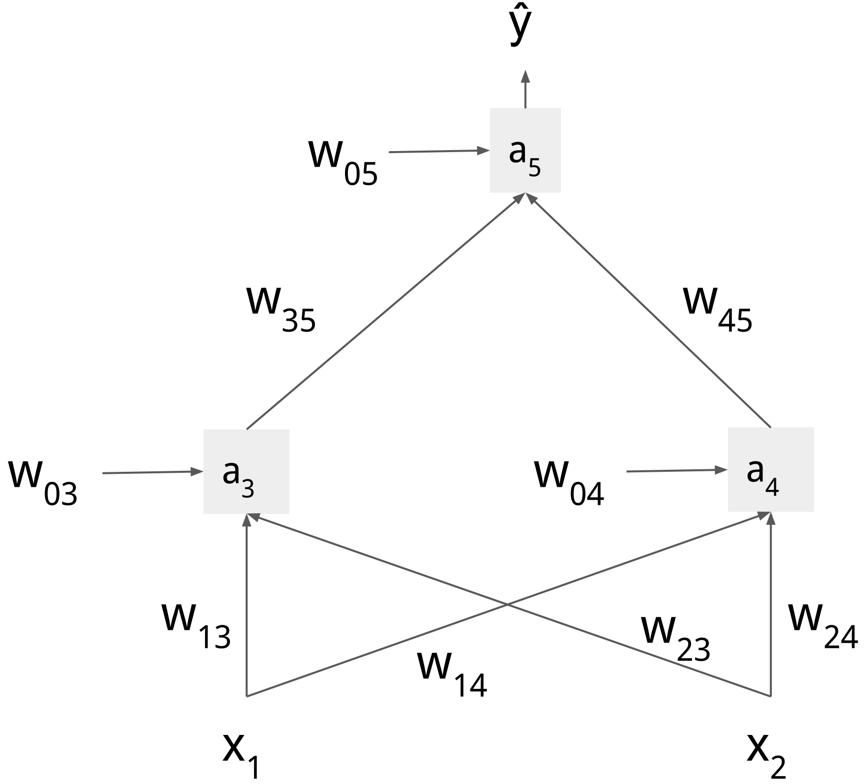


$$Loss(w)=(y-\hat{y})^2$$

Target output, from training example

Predicted output, function of w and training input x

Bias for neuron i: w_{0i}
Weight from i->j: w_{ij}



$$Loss(w)=(y-\hat{y})^2$$

Target output, from training example

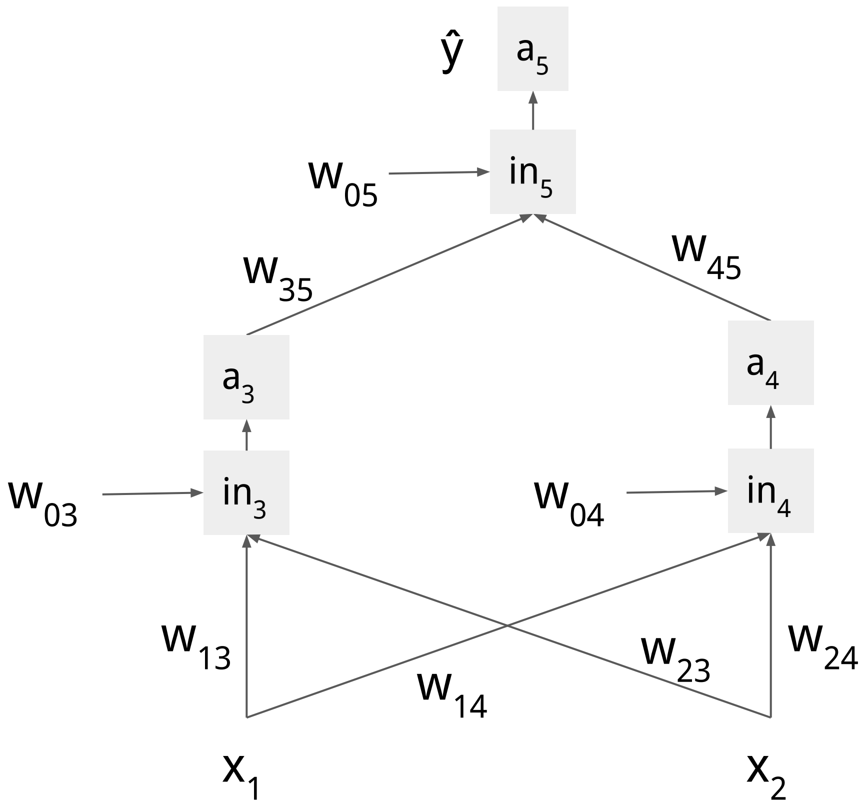
Predicted output, function of w and training input x

Bias for neuron i: w_{0i}
Weight from i->j: w_{ij}

$Loss(w)=(y-\hat{y})^2$

Target output, from training example

Predicted output, function of w and training input x

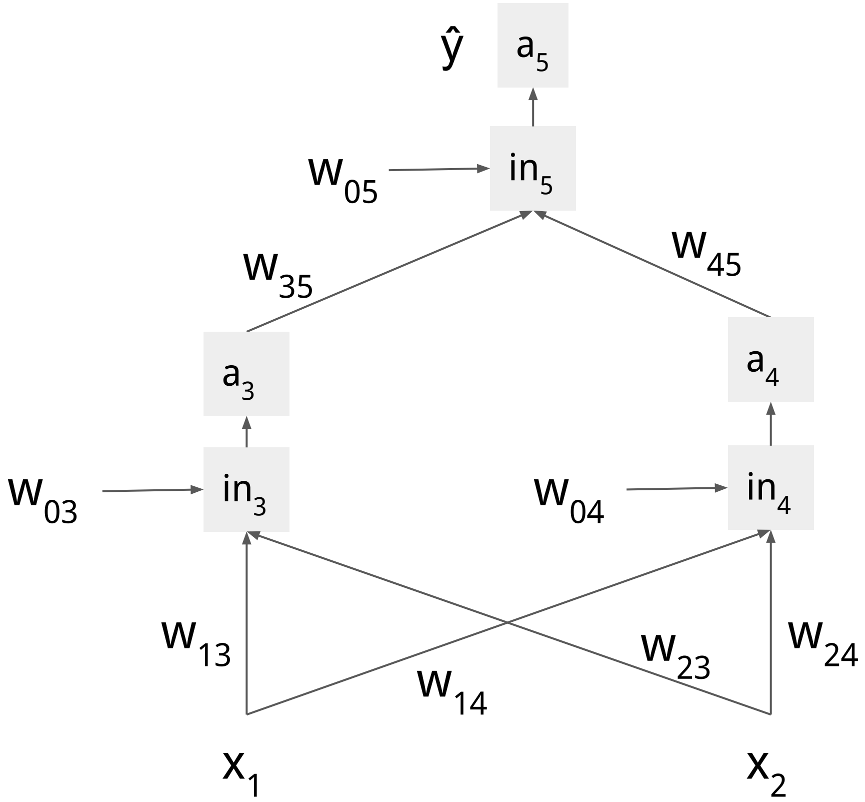


Bias for neuron i: w_{0i}

Weight from i->j: w_{ij}

Activation output: $a_i=g(in_i)$

Linear input: $in_i = w_{0i} + \sum_{j\to i} w_{ji} a_j$



Loss(w)=(y-ŷ)^2

$$\frac{\mathrm{dLoss}}{dw_{05}} = \frac{\mathrm{dLoss}}{da_5}\frac{\mathrm{d}a_5}{dw_{05}}$$

$$= \frac{\mathrm{dLoss}}{da_5}\frac{\mathrm{d}a_5}{d\mathrm{in}_5}\frac{\mathrm{d}\,\mathrm{in}_5}{dw_{05}}$$

$$= \frac{\mathrm{dLoss}}{da_5}\frac{\mathrm{d}a_5}{d\mathrm{in}_5}\frac{\mathrm{d}}{dw_{05}}(w_{05} + w_{35}a_3 + w_{45}a_4)$$

$$= \frac{\mathrm{dLoss}}{da_5}\frac{\mathrm{d}a_5}{d\mathrm{in}_5}(1 + 0 + 0)$$

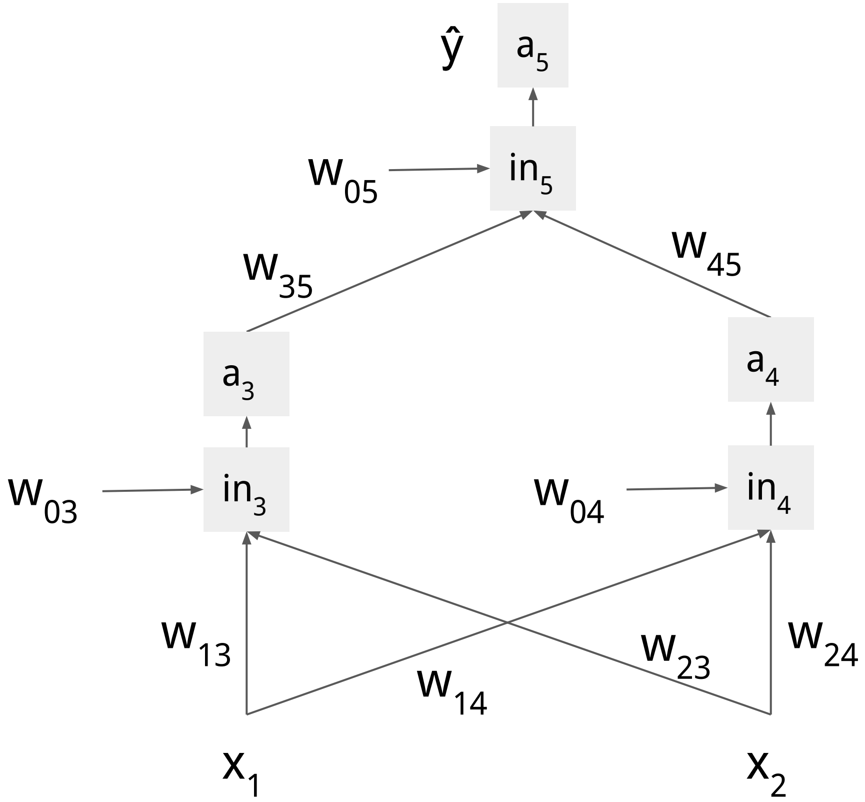
$$= \frac{\mathrm{dLoss}}{da_5}\frac{\mathrm{d}a_5}{d\mathrm{in}_5}$$

$$= \frac{\mathrm{dLoss}}{da_5}g'(\mathrm{in}_5) = 2(y - a_5)g'(\mathrm{in}_5)$$

Bias for neuron i: w_{0i}
Weight from i->j: w_{ij}

Activation output: $a_i=g(in_i)$

Linear input: $in_i = w_{0i} + \sum_{j\to i} w_{ji}a_j$



$\text{Loss}(w) = (y - \hat{y})^2$

$$\frac{\text{dLoss}}{dw_{35}} = \frac{\text{dLoss}}{da_5}\frac{\text{d}a_5}{dw_{35}}$$

$$= \frac{\text{dLoss}}{da_5}\frac{\text{d}a_5}{d\text{in}_5}\frac{\text{din}_5}{dw_{35}}$$

$$= \frac{\text{dLoss}}{da_5}\frac{\text{d}a_5}{d\text{in}_5}\frac{\text{d}}{dw_{35}}(w_{05} + w_{35}a_3 + w_{45}a_4)$$

$$= \frac{\text{dLoss}}{da_5}\frac{\text{d}a_5}{d\text{in}_5}(0 + a_3 + 0)$$

$$= \frac{\text{dLoss}}{da_5}\frac{\text{d}a_5}{d\text{in}_5}a_3$$

$$= \frac{\text{dLoss}}{da_5}g'(\text{in}_5)a_3 = 2(y - a_5)g'(\text{in}_5)a_3$$

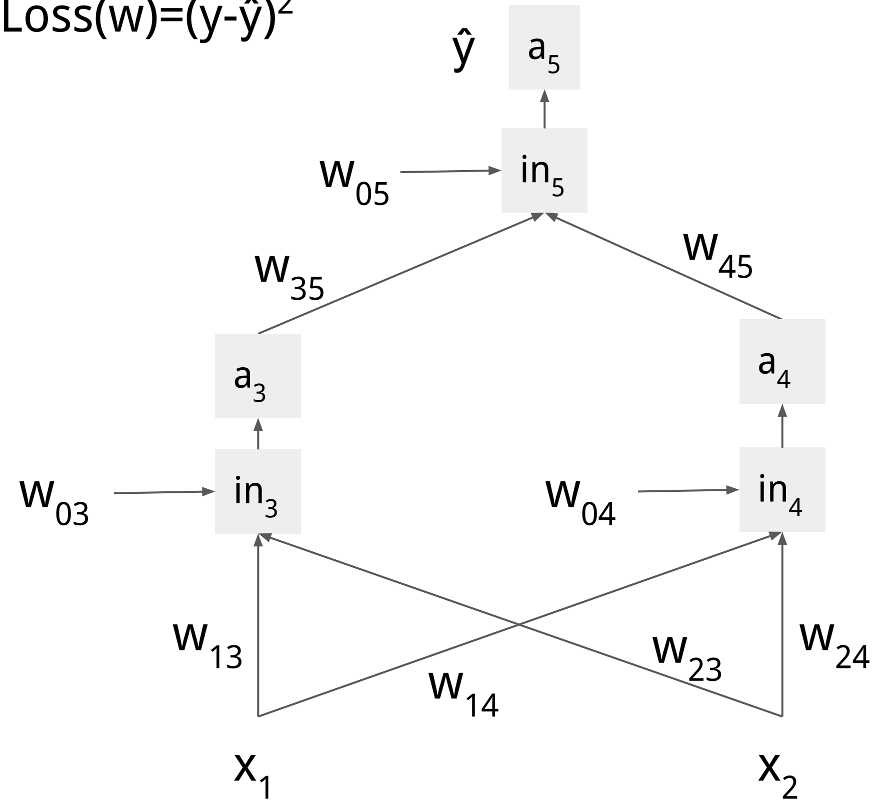
Bias for neuron i: w_{0i}
Weight from i->j: w_{ij}

Activation output: $a_i = g(\text{in}_i)$

Linear input: $\text{in}_i = w_{0i} + \sum_{j \to i} w_{ji} a_j$

<many derivatives later>

Loss(w)=(y-ŷ)^2



$$\frac{\text{dLoss}}{d\text{in}_5} = 2(y - a_5)g'(\text{in}_5)$$

$$\frac{\text{dLoss}}{dw_{05}} = \frac{\text{dLoss}}{d\text{in}_5} \times 1$$

$$\frac{\text{dLoss}}{dw_{35}} = \frac{\text{dLoss}}{d\text{in}_5} \times a_3$$

$$\frac{\text{dLoss}}{d\text{in}_3} = \frac{\text{dLoss}}{d\text{in}_5} w_{35}\, g'(\text{in}_3)$$

$$\frac{\text{dLoss}}{dw_{03}} = \frac{\text{dLoss}}{d\text{in}_3} \times 1$$

$$\frac{\text{dLoss}}{dw_{13}} = \frac{\text{dLoss}}{d\text{in}_3} \times x_1$$

Activation output: $a_i = g(\text{in}_i)$

Bias for neuron i: w_{0i}

Weight from i->j: w_{ij}

Linear input: $\text{in}_i = w_{0i} + \sum_{j \to i} w_{ji} a_j$

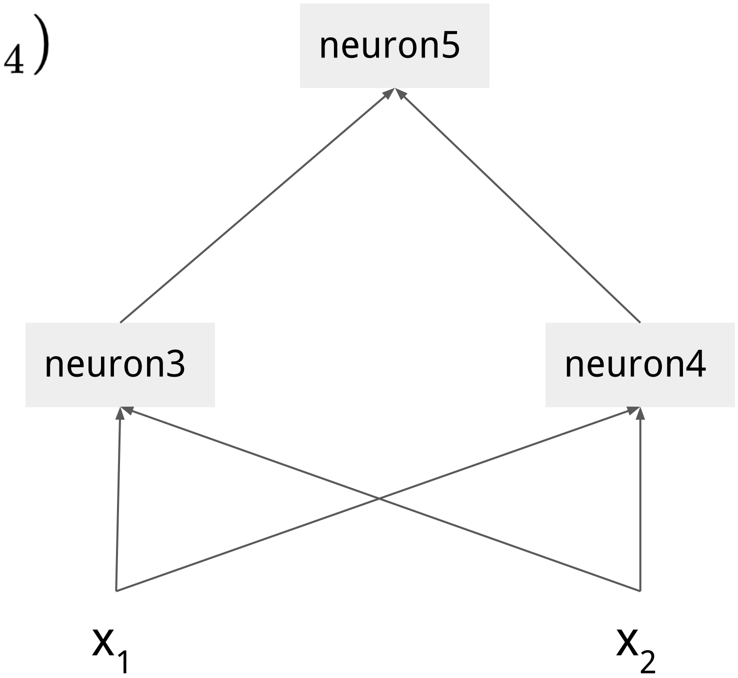
A pattern is emerging

$$\frac{\text{dLoss}}{d\text{in}_4} = \frac{\text{dLoss}}{d\text{in}_5} w_{45}\, g'(\text{in}_4)$$

$$\frac{\text{dLoss}}{dw_{04}} = \frac{\text{dLoss}}{d\text{in}_4} \times 1$$

$$\frac{\text{dLoss}}{dw_{14}} = \frac{\text{dLoss}}{d\text{in}_4} \times x_1$$

$$\frac{\text{dLoss}}{dw_{24}} = \frac{\text{dLoss}}{d\text{in}_4} \times x_2$$



$$\frac{\text{dLoss}}{d\text{in}_3} = \frac{\text{dLoss}}{d\text{in}_5} w_{35}\, g'(\text{in}_3)$$

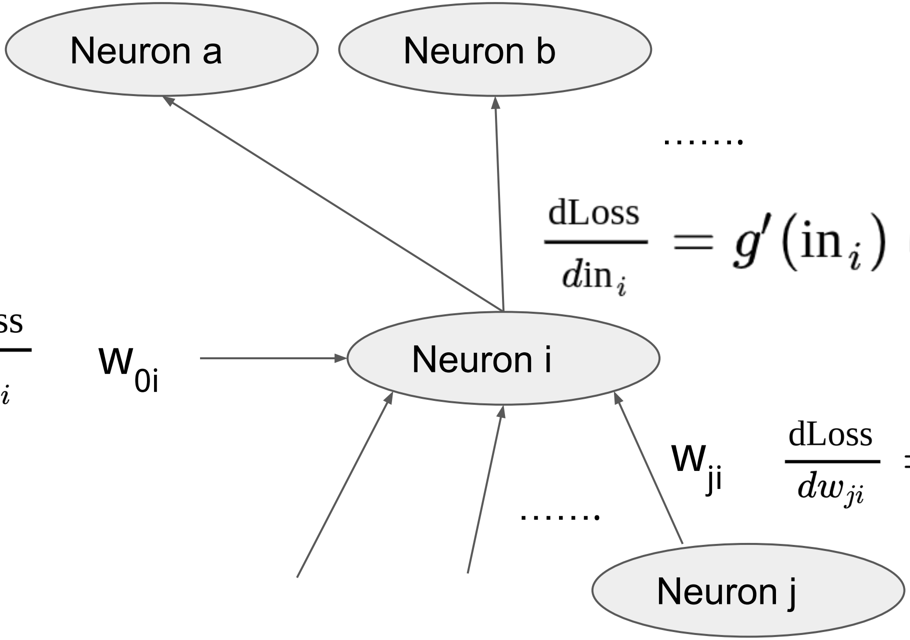
$$\frac{\text{dLoss}}{dw_{03}} = \frac{\text{dLoss}}{d\text{in}_3} \times 1$$

$$\frac{\text{dLoss}}{dw_{13}} = \frac{\text{dLoss}}{d\text{in}_3} \times x_1$$

$$\frac{\text{dLoss}}{dw_{23}} = \frac{\text{dLoss}}{d\text{in}_3} \times x_2$$

Generalizing the pattern

Key quantity: $\dfrac{\text{dLoss}}{d\text{in}_i}$ (how much loss changes as input to neuron i changes)
("perceived error" at neuron i)



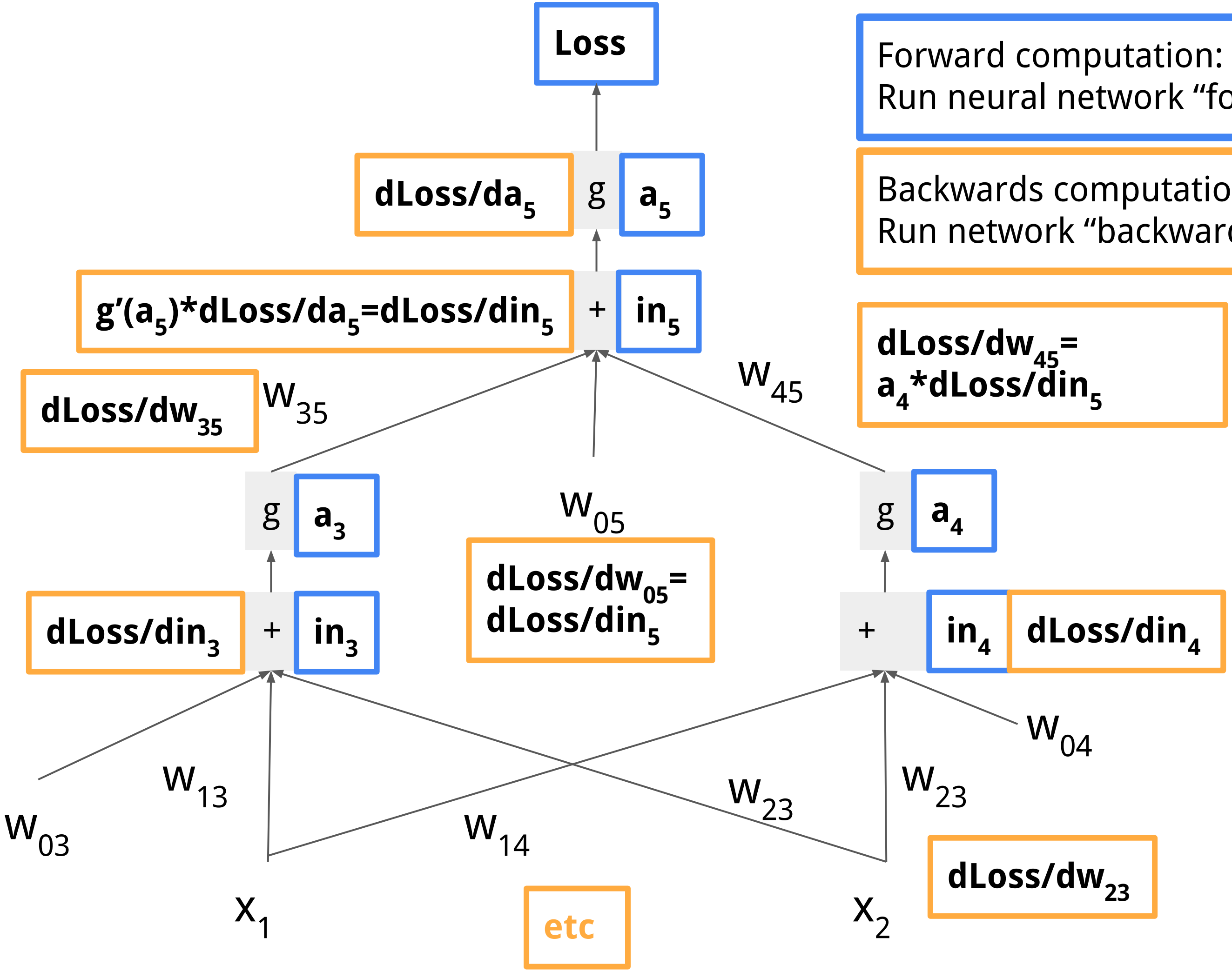
$$\frac{\text{dLoss}}{d\text{in}_i} = g'(\text{in}_i)\left(\frac{\text{dLoss}}{d\text{in}_a}w_{ia} + \frac{\text{dLoss}}{d\text{in}_b}w_{ib} + \cdots\right)$$

$$\frac{\text{dLoss}}{dw_{0i}} = \frac{\text{dLoss}}{d\text{in}_i}$$

$$\frac{\text{dLoss}}{dw_{ji}} = \frac{\text{dLoss}}{d\text{in}_i}a_j$$

Forward computation:
Run neural network “forward” and compute loss

Backwards computation:
Run network “backward” and compute derivatives



Backpropagation

Objective: compute dLoss / dw_{ij} for each weight/bias w_{ij}

1. Run the network forward to compute each neuron's output
2. Compute the loss
3. Compute the derivative of the loss with respect to network output:

   dLoss/dy

4. Compute derivative of loss with respect to each neuron's total input

$$\frac{\mathrm{dLoss}}{d\mathrm{in}_i} = g'(\mathrm{in}_i) \sum_{i \to j} \frac{\mathrm{dLoss}}{d\mathrm{in}_j} w_{ij}$$

5. Compute derivative of loss with respect to weights/biases

$$\frac{\mathrm{dLoss}}{dw_{ji}} = \frac{\mathrm{dLoss}}{d\mathrm{in}_i} a_j$$