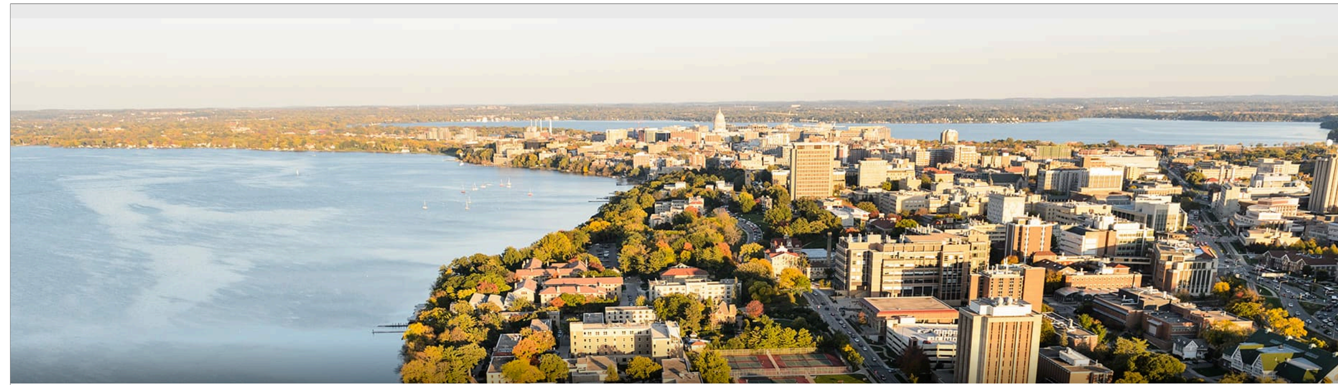


CS 540 Introduction to Artificial Intelligence

Neural Networks (II)

Yingyu Liang
University of Wisconsin-Madison

Oct 21, 2021

Slides created by Sharon Li [modified by Yingyu Liang]

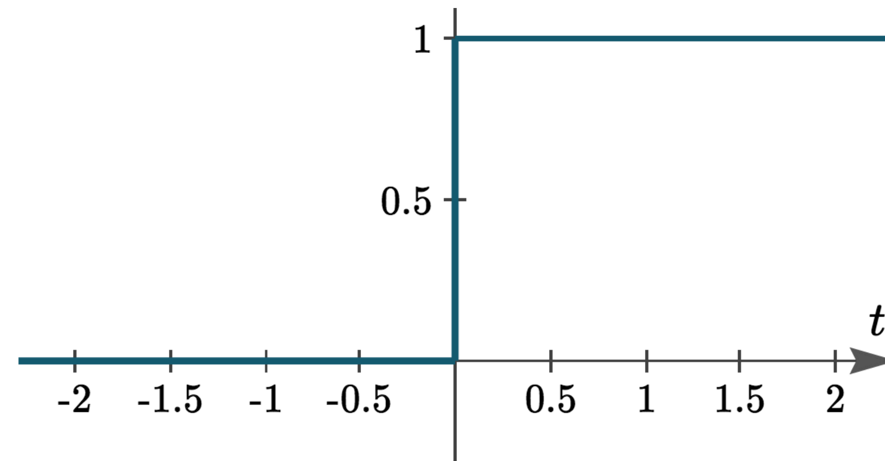
Today's outline

- Single-layer Perceptron Continued
- Multi-layer Perceptron
  - Single output
  - Multiple output
- How to train neural networks
  - Gradient descent

Step Function activation

Step function is discontinuous, which cannot be used for gradient descent

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

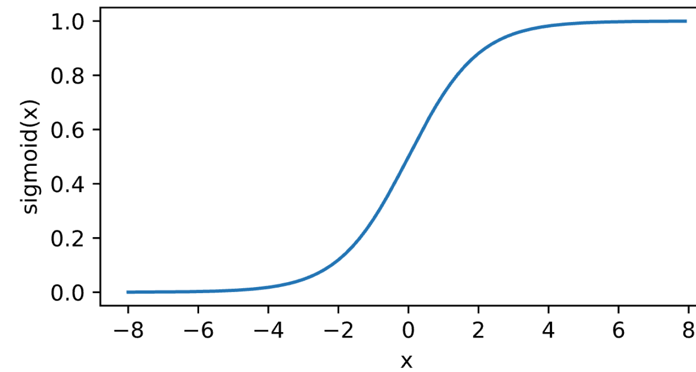


The standard perceptron uses the step function activation. There are other variants.

Sigmoid/Logistic Activation

Map input into [0, 1], a **soft** version of $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



The sigmoid or logistic activation function is typically used in the output layer of the network for classification. Will learn more later.

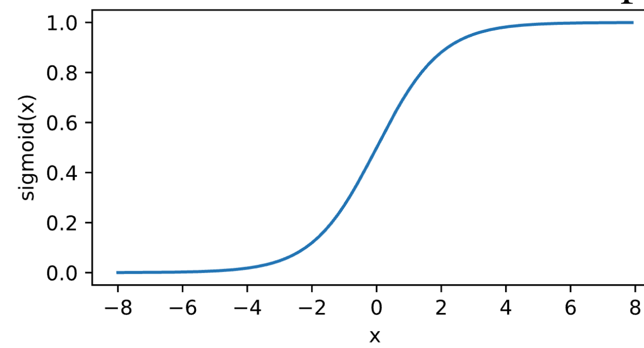
The sigmoid perceptron with the sigmoid activation is just the model for logistic regression. (Note that logistic regression is actually for classification, not for regression!)

Logistic regression

$\mathbf{x} \in \mathbb{R}^d, y = \{-1, +1\}$

$$p(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}$$

$$p(y = -1 \mid \mathbf{x}) = 1 - \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^T\mathbf{x})}$$



Logistic regression

Given: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$

Training: maximize likelihood estimate (on the conditional probability)

$$\max_{\mathbf{w}} \sum_i \log \frac{1}{1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)}$$

Training: MLE

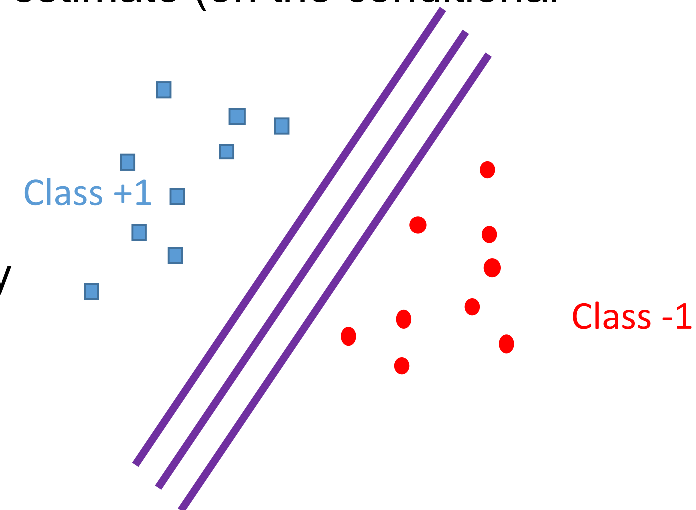
Optimization: typically use SGD or minibatch SGD

Logistic regression

Given: $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$

Training: maximize likelihood estimate (on the conditional probability)

When training data is linearly separable, many solutions



Issue: when the training data is linearly separable with some margin, there exist many solutions. We would like to impose some preference

Logistic regression

Given: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$

Training: maximum A posteriori (MAP)

$$\min_{\mathbf{w}} \sum_i -\log \frac{1}{1+\exp(-y_i \mathbf{w}^T \mathbf{x}_i)} + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

- Convex optimization
- Solve via (stochastic) gradient descent

We can impose the l2 regularization, ie, we prefer the solution with a small norm weight vector.

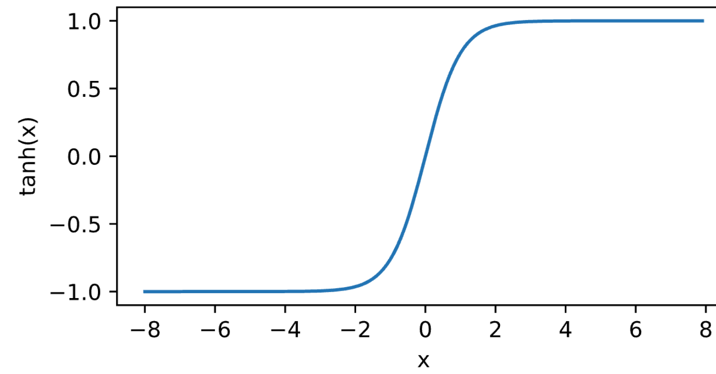
Comments (not required to understand or derive):

1. This regularized training objective can be derived using MAP, where the prior is a Gaussian distribution of w.
2. The small norm weight vector turns out to give large margin between the two classes.

Tanh Activation

Map inputs into (-1, 1)

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

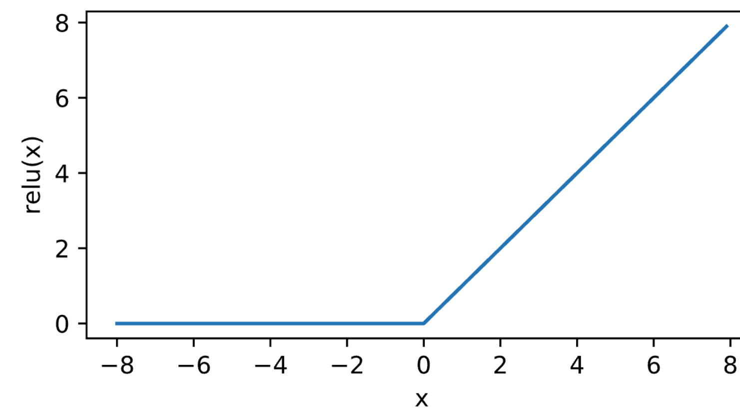


Another activation: Tanh. Closely related to the sigmoid function

ReLU Activation

ReLU: rectified linear unit (commonly used in modern neural networks)

$$\text{ReLU}(x) = \max(x, 0)$$



A commonly used activation: ReLU.

Simple, efficient to compute. The gradient is also simple: 0 for negative inputs, and 1 for nonnegative inputs. (Strictly speaking it doesn't have gradient at x=0 but conventionally we just define that to be 1)

Quiz Break

Which one of the following is valid activation function

A) Step function

B) Sigmoid function

C) ReLU function

D) all of above

Quiz Break

Which one of the following is valid activation function

a)Step function

b) Sigmoid function

C) ReLU function

D) all of above

Quiz Break

Let $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. Which of the following functions is NOT an element-wise operation that can be used as an activation function?

A $f(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

B $f(x) = \begin{bmatrix} \max(0, x_1) \\ \max(0, x_2) \end{bmatrix}$

C $f(x) = \begin{bmatrix} \exp(x_1) \\ \exp(x_2) \end{bmatrix}$

D $f(x) = \begin{bmatrix} \exp(x_1 + x_2) \\ \exp(x_2) \end{bmatrix}$

Quiz Break

Let $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$. Which of the following functions is NOT an element-wise operation that can be used as an activation function?

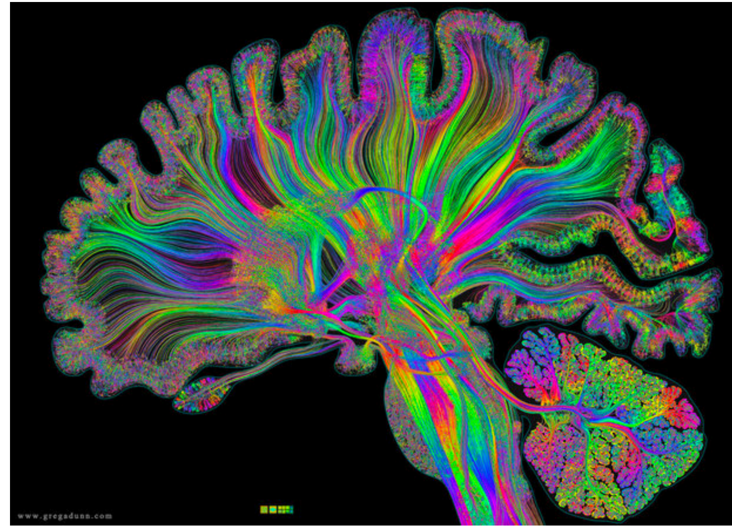
A $\mathrm{f}(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

B $\mathrm{f}(x) = \begin{bmatrix} \max(0, x_1) \\ \max(0, x_2) \end{bmatrix}$

C $\mathrm{f}(x) = \begin{bmatrix} \exp(x_1) \\ \exp(x_2) \end{bmatrix}$

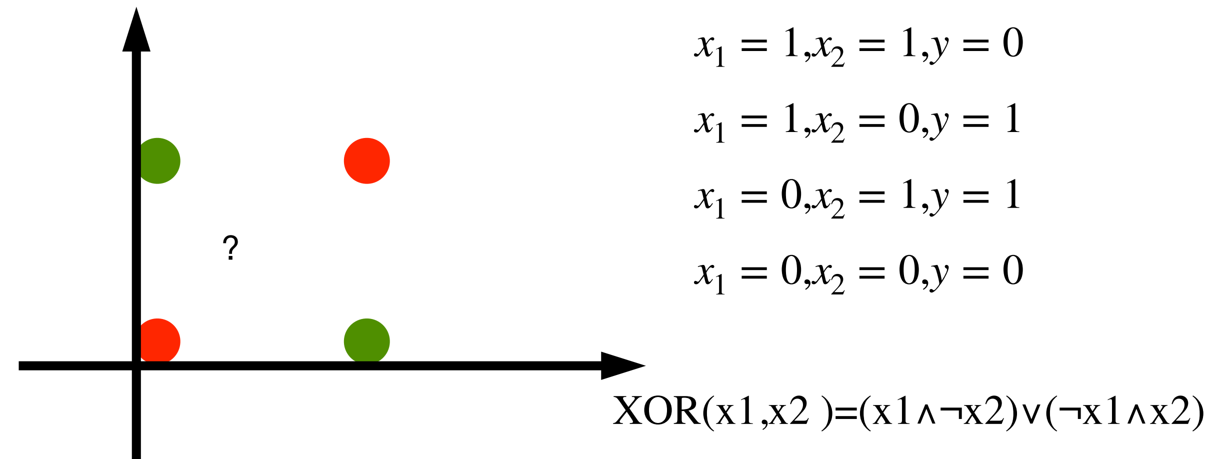
D $\mathrm{f}(x) = \begin{bmatrix} \exp(x_1 + x_2) \\ \exp(x_2) \end{bmatrix}$

Multilayer Perceptron



The limited power of a single neuron

The perceptron cannot learn an **XOR** function
(neurons can only generate linear separators)



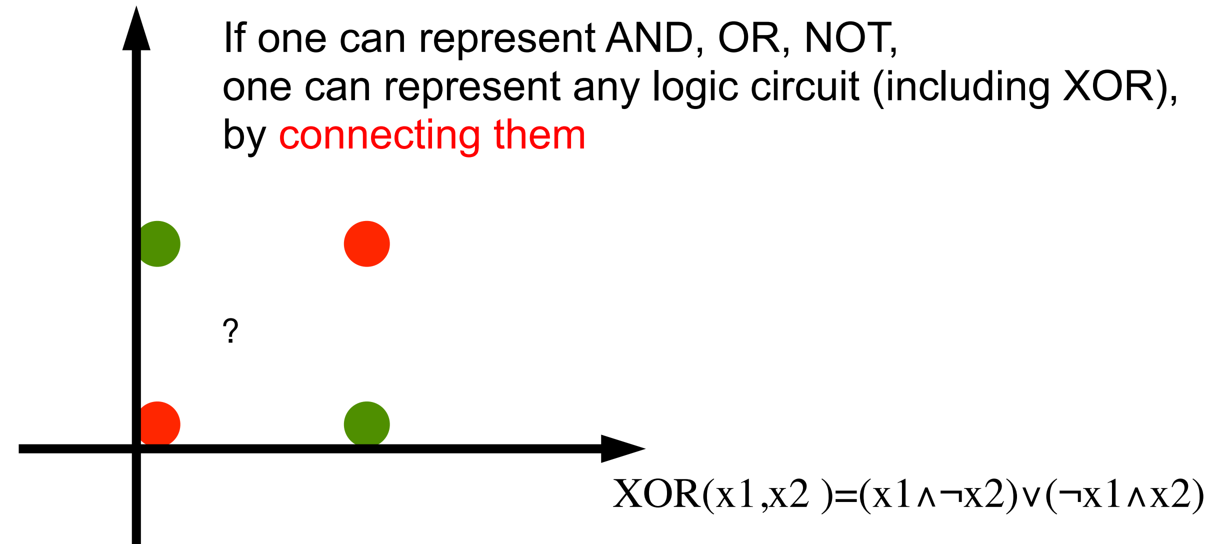
Recall the limitation of the standard perceptron: linear decision boundary. Thus cannot represent XOR which has a nonlinear decision boundary.

However, we know that XOR (actually any logic functions) can be represented using compositions of the standard logic functions AND OR NOT, and the standard logic functions can be represented using perceptrons. Therefore, we can use compositions of the perceptron to represent any logic functions including XOR.

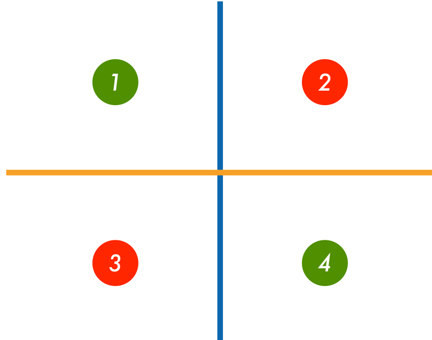
This motivates to connect perceptrons/neurons into networks to get more powerful models.

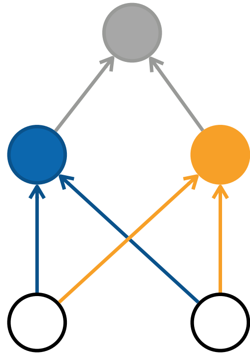
The limited power of a single neuron

XOR problem



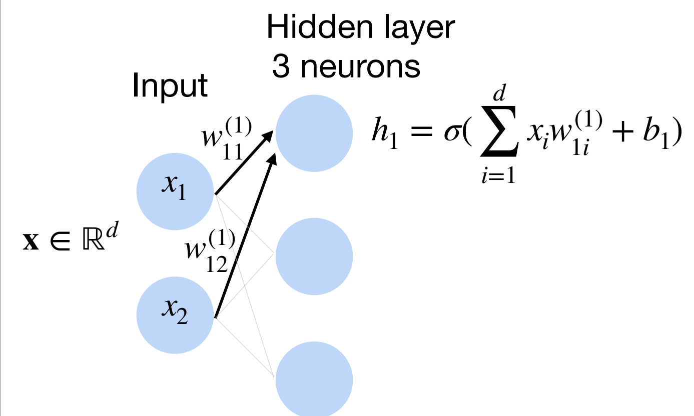
Learning XOR





Multi-layer perceptron: Example

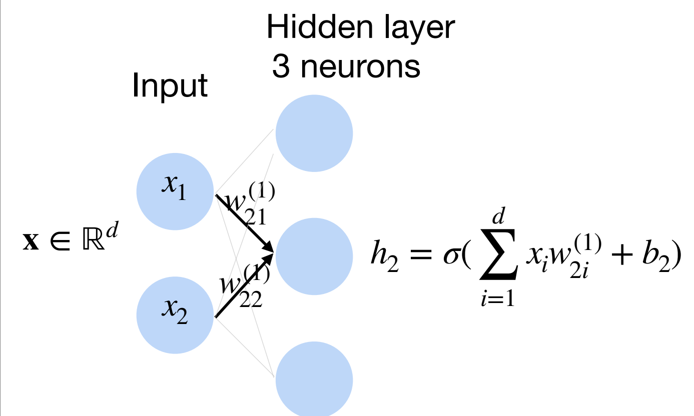
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



$$h_1 = \sigma\left(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1\right)$$

Multi-layer perceptron: Example

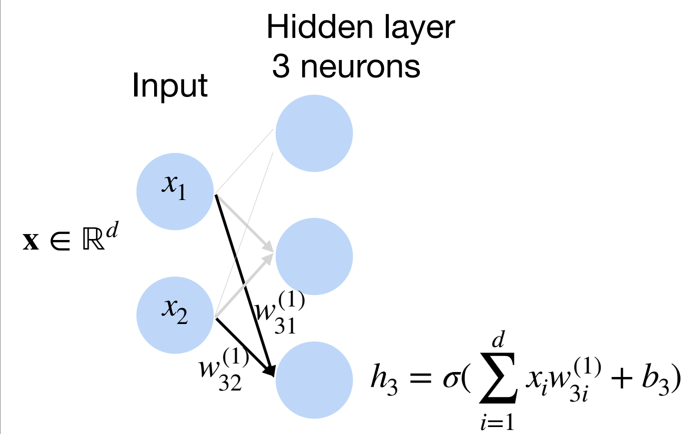
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



$$h_2 = \sigma(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2)$$

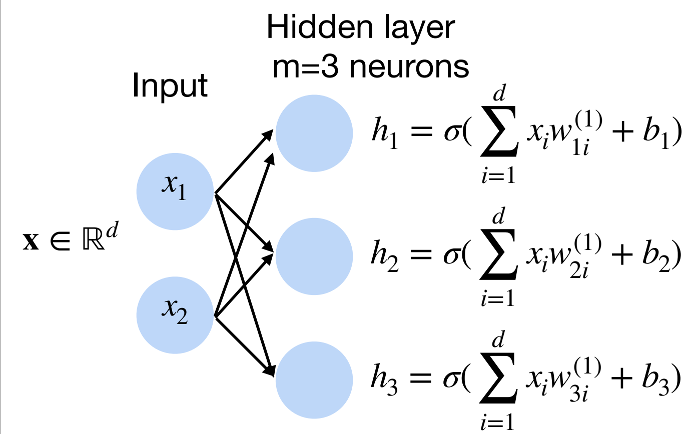
Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



Multi-layer perceptron: Example

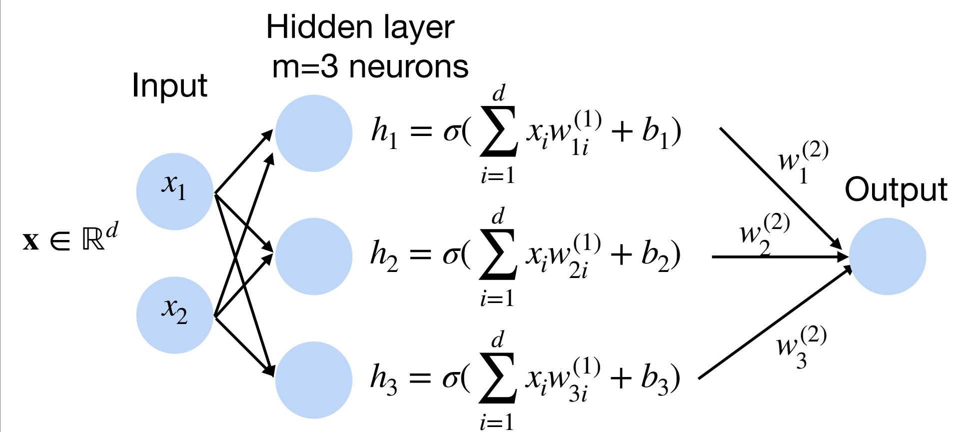
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



Multiple neurons in the hidden layer, each is a perceptron.

Multi-layer perceptron: Example

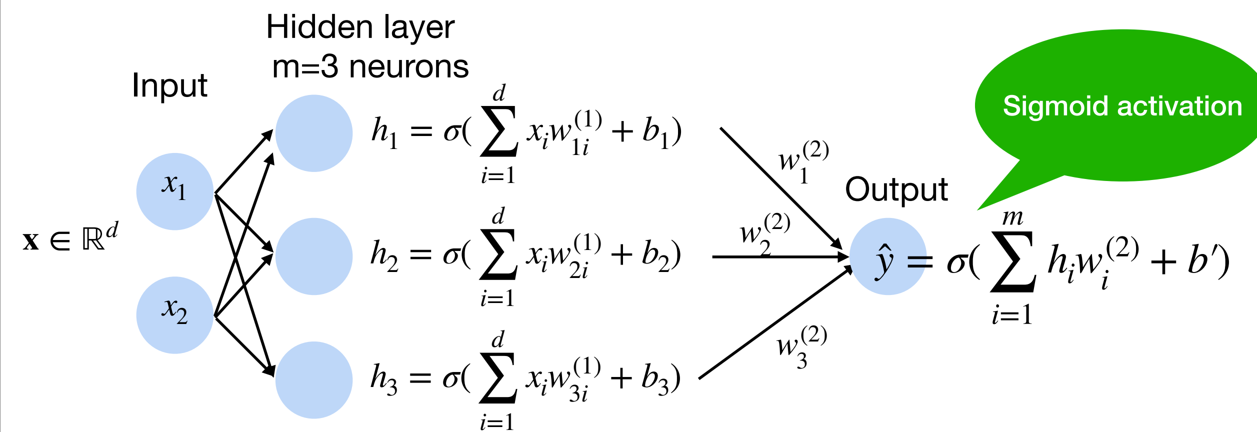
- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



Now view the output of the hidden neurons as an input vector, and build another neuron on top of it (the output layer).

Multi-layer perceptron: Example

- Standard way to connect Perceptrons
- Example: 1 hidden layer, 1 output layer, depth = 2



For binary classification: we use sigmoid activation function in the output neuron.
For regression: just no activation function

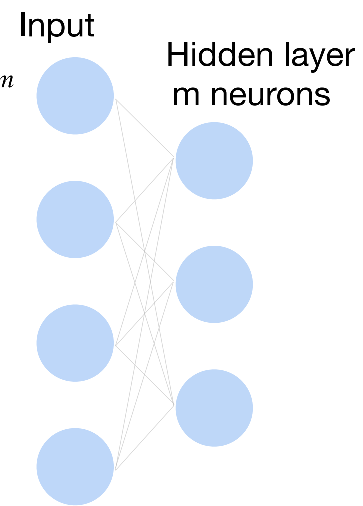
For multiple classes: we use softmax (see later slides)

Multi-layer perceptron: Matrix Notation

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$
- Intermediate output

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b})$$

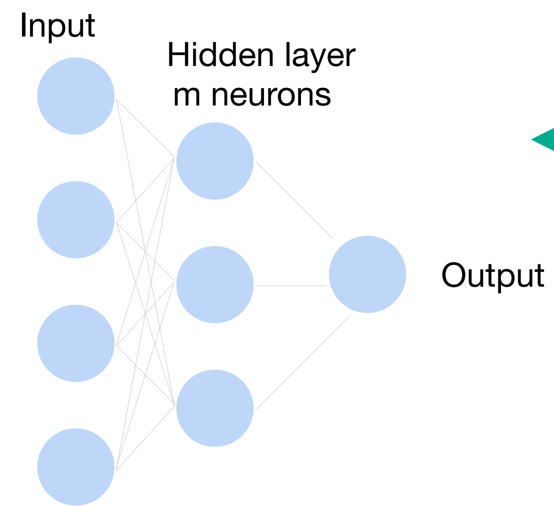
$$\mathbf{h} \in \mathbb{R}^m$$



For simplicity, we can use matrix notation to describe neural networks.

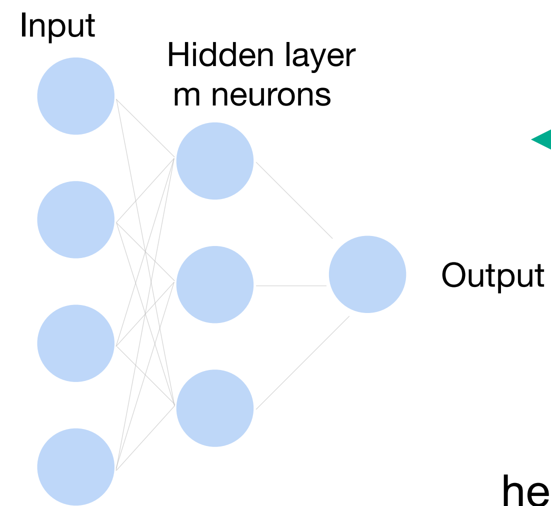
Stack the weight vectors of the neurons in the hidden layer as the rows of a weight matrix W^{(1)}, stack their bias terms into a vector. Let h denote the vector of the outputs of these neurons. Then we can write the hidden layer compactly as h=\sigma(W^{(1)}x + b). Note that here \sigma is applied to each element of the linear transformation vector W^{(1)}x + b and get h.

Multi-layer perceptron



Why do we need an a nonlinear activation?

Multi-layer perceptron



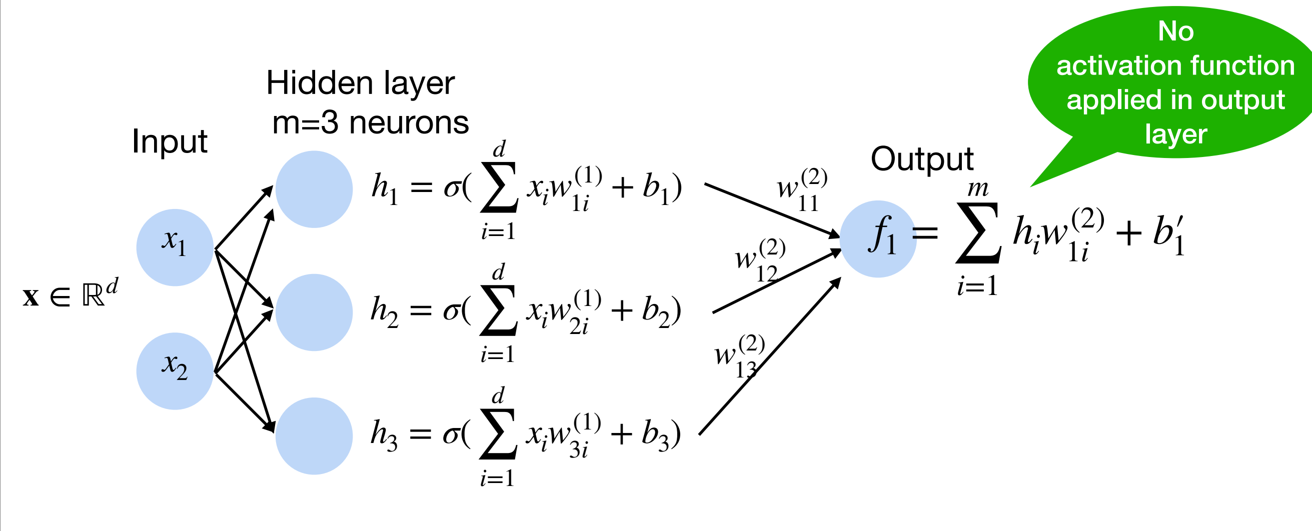
Why do we need an a nonlinear activation?

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$f = \mathbf{w}_2^T\mathbf{h} + b_2$$

hence $f = \mathbf{w}_2^\top \mathbf{W}\mathbf{x} + b'$

If we don’t have nonlinear activation then the whole network is still a linear function, which is not desired.



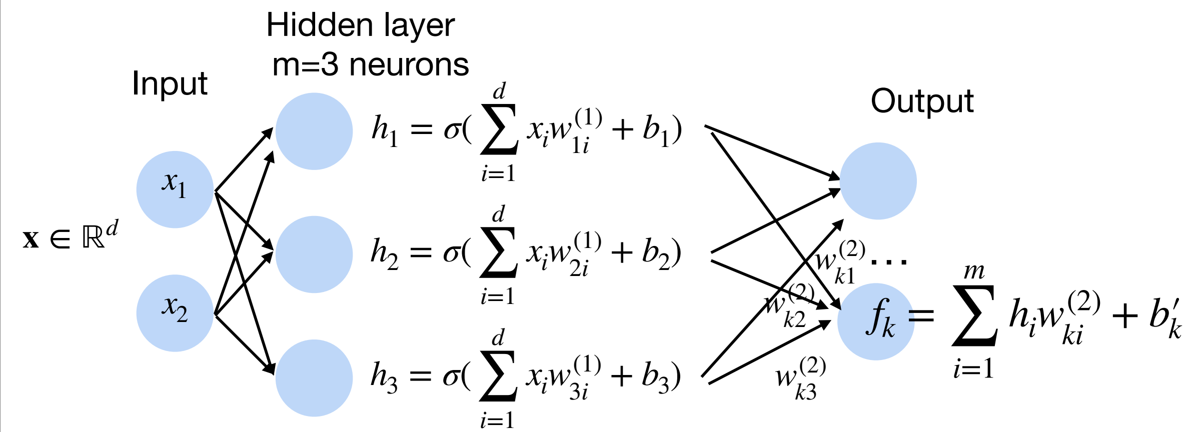
For multi-class classification, we can use the softmax operation.

First compute the linear transformation, denoted as f_i

Neural network for k-way classification

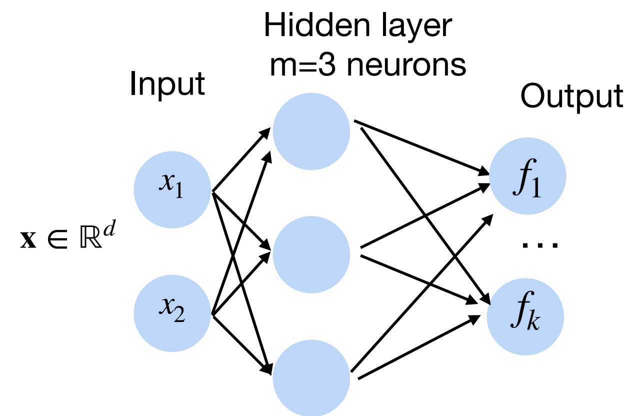
- K outputs units in the final layer

Multi-class classification (e.g., ImageNet with k=1000)



Softmax

Turns outputs f into probabilities (sum up to 1 across k classes)



$$p(y \mid \mathbf{x}) = \text{softmax}(f)$$

$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

Then squash the vector f into a probabilistic vector over the K classes: first apply the exponential function and then normalize.

The elements f_i are nonnegative and sum up to 1: can be viewed as a probabilistic distribution over the K classes. Typically used to represent the conditional distribution p(y|x).

Softmax

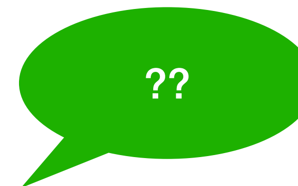
Turns outputs f into probabilities (sum up to 1 across k classes)

Output layer

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \\ 0.7 \\ 1.1 \end{bmatrix} \rightarrow$$

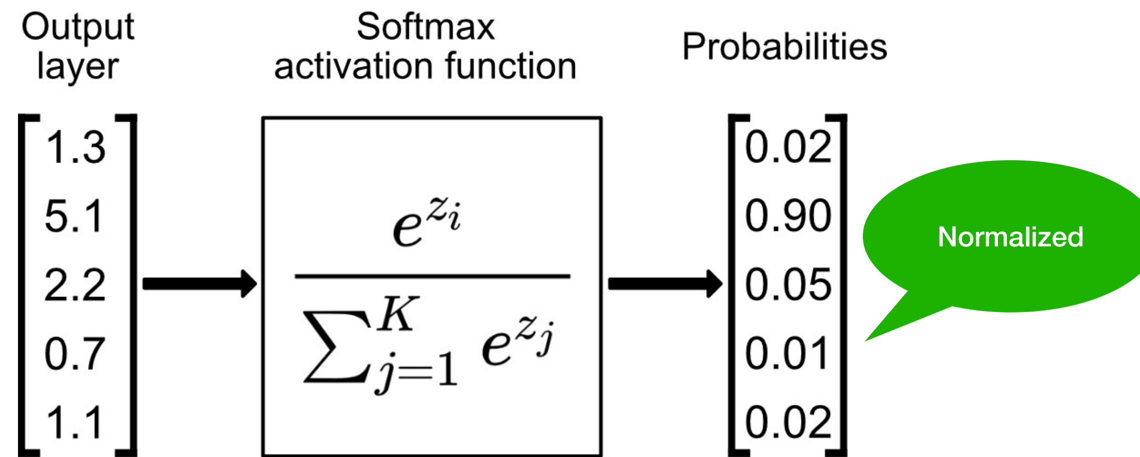
Softmax activation function

$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \rightarrow$$



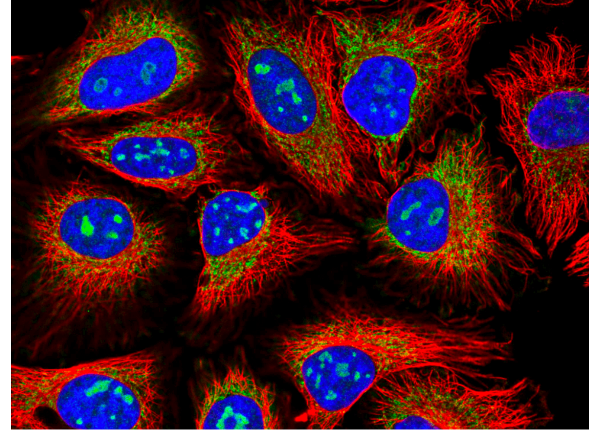
Softmax

Turns outputs f into probabilities (sum up to 1 across k classes)



Classification Tasks at Kaggle

Classify human protein microscope images into 28 categories

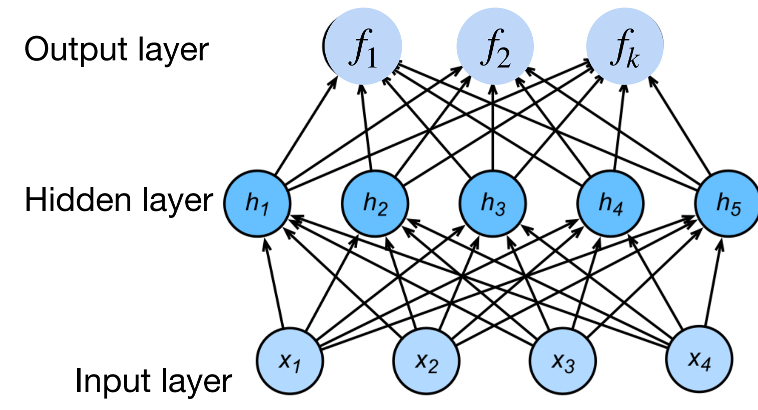


```
0.  Nucleoplasm
1.  Nuclear membrane
2.  Nucleoli
3.  Nucleoli fibrillar
4.  Nuclear speckles
5.  Nuclear bodies
6.  Endoplasmic reticu
7.  Golgi apparatus
8.  Peroxisomes
9.  Endosomes
10.  Lysosomes
11.  Intermediate fila
12.  Actin filaments
13.  Focal adhesion si
14.  Microtubules
15.  Microtubule ends
16.  Cytokinetic bridg
```

https://www.kaggle.com/c/human-protein-atlas-image-classification

More complicated neural networks

$$y_1, y_2, \ldots, y_k = \text{softmax}(f_1, f_2, \ldots, f_k)$$

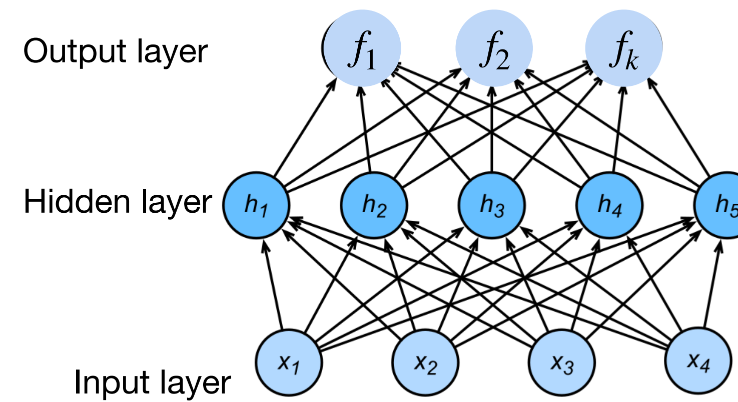


More complicated neural networks

- Input $\mathbf{x} \in \mathbb{R}^d$
- Hidden $\mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b})$$
$$\mathbf{f} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$

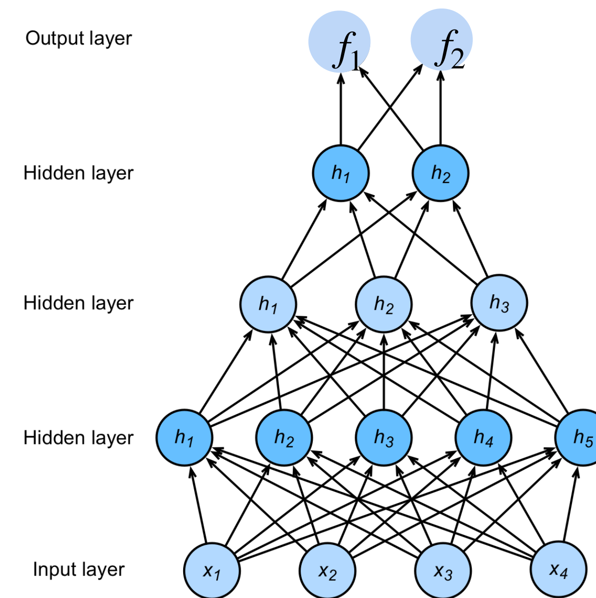
$$y_1, y_2, \ldots, y_k = \text{softmax}(f_1, f_2, \ldots, f_k)$$



We can also use the matrix notation.

More complicated neural networks: multiple hidden layers

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{f} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$
$$\mathbf{y} = \text{softmax}(\mathbf{f})$$



Multiple layer networks: composition of layers; each layer is linear transformation+nonlinear activation functions; hidden layers typically use element-wise activation functions; the output layer can use no activation (for regression), sigmoid (for binary classification), or softmax (for multi-class classification).

Quiz Break

Which output function is often used for multi-class classification tasks?

A Sigmoid function

B Rectified Linear Unit (ReLU)

C Softmax function

D Max function

Quiz Break

Which output function is often used for multi-class classification tasks?

A Sigmoid function

B Rectified Linear Unit (ReLU)

C Softmax function

D Max function

Quiz Break

Suppose you are given a 3-layer multilayer perceptron (2 hidden layers h1 and h2 and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose h1 has 1024 units and h2 has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?

Quiz Break

Suppose you are given a 3-layer multilayer perceptron (2 hidden layers h1 and h2 and 1 output layer). All activation functions are sigmoids, and the output layer uses a softmax function. Suppose h1 has 1024 units and h2 has 512 units. Given a dataset with 2 input features and 3 unique class labels, how many learnable parameters does the perceptron have in total?

1024 * 2 + 1024 + 512 * 1024 + 512 + 512 * 3 + 3 = 529411

Quiz Break

Consider a three-layer network with **linear Perceptrons** for binary classification. The hidden layer has 3 neurons. Can the network represent a XOR problem?

a) Yes

b) No

Quiz Break

Consider a three-layer network with **linear Perceptrons** for binary classification. The hidden layer has 3 neurons. Can the network represent a XOR problem?

a)Yes

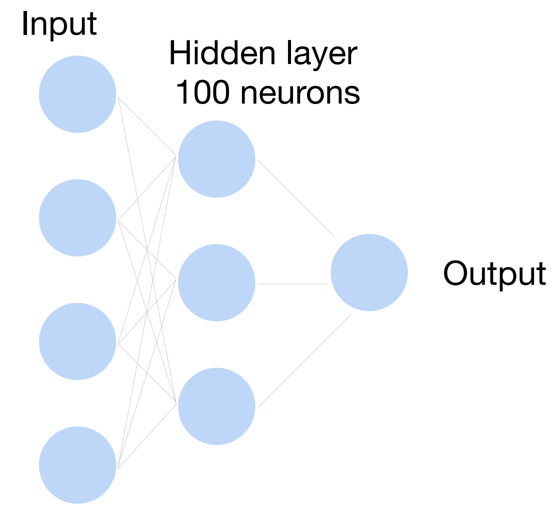
b)No

Solution:

A combination of linear Perceptrons is still a linear function.

How to train a neural network?

Classify cats vs. dogs



How to train a neural network?

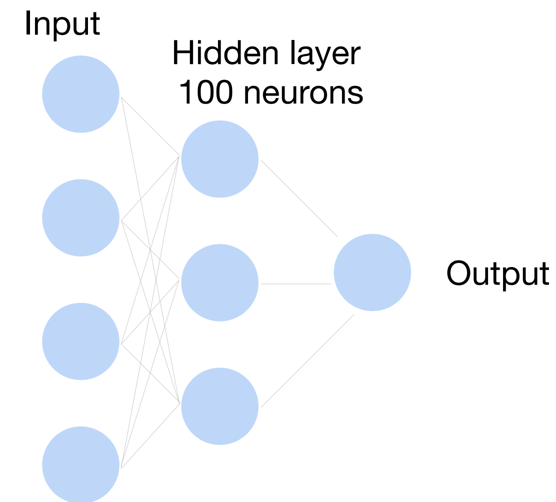
$\mathbf{x} \in \mathbb{R}^d$ One training data point in the training set D

$\hat{y}$ Model output for example **x**

y Ground truth label for example **x**

Learning by matching the output to the label

We want $\hat{y} \rightarrow 1$ **when** $y = 1$**,**
and $\hat{y} \rightarrow 0$ **when** $y = 0$



Three elements to specify a machine learning method:

1. Model class (here the networks)
2. Loss function (cross-entropy for classification, usually squared loss for regression)
3. Optimization method (gradient descent)

We will consider classification.

We have specify the model class, now we specify the loss.
The loss on the training set is an average over the training data points. The loss for each data point is meant to measure the difference between the output and the true label.

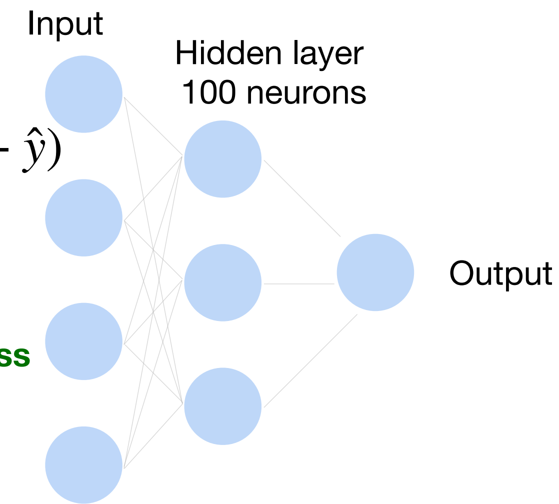
How to train a neural network?

Loss function: $\frac{1}{|D|}\sum_i \ell(\mathbf{x}_i, y_i)$

Per-sample loss:

$$\ell(\mathbf{x}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Also known as binary cross-entropy loss



For binary classification with labels {0,1}, the loss for each data point is defined as shown. When y=0, minimizing this binary cross-entropy loss means pushing the prediction \hat{y} to 0. When y=1, minimizing it means pushing the prediction \hat{y} to 1.

How to train a neural network?

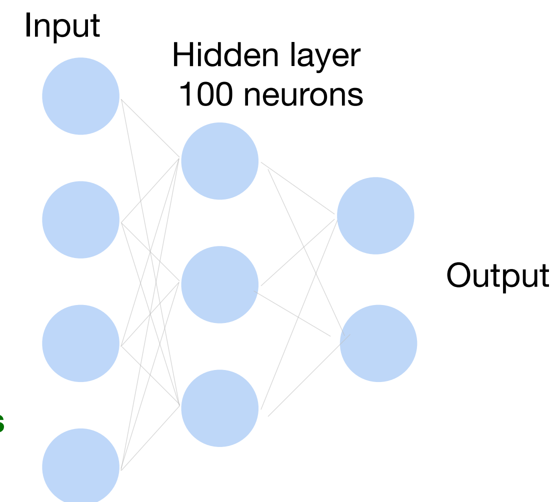
Loss function: $\frac{1}{|D|} \sum_{(\mathbf{x},y) \in D} \ell(\mathbf{x}, y)$

Per-sample loss:

$$\ell(\mathbf{x}, y) = \sum_{j=1}^{K} -Y_j \log p_j = -\log p_y$$

where Y is one-hot encoding of

Also known as cross-entropy loss or softmax loss



For multi-class classification, the loss for each data point (per sample loss) is defined to be the cross-entropy between the output probabilistic vector (a distribution over the k classes) and the true probabilistic vector (another distribution over the k classes). cross-entropy is a notion from information theory, measuring the “difference” between two distributions. Here it’s applied on the output probabilities and the true probabilities.

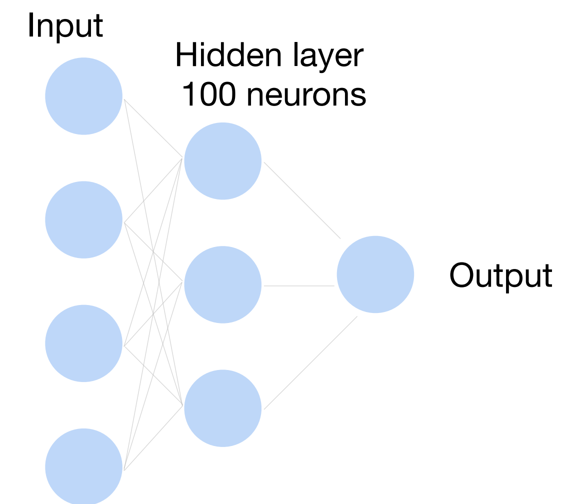
The output probabilities: the output after softmax in the network.
The true probabilities: turn the label y (taking value in {1,2,…,k}) into a one-hot encoding vector Y. Here, Y is a vector of dimension k (corresponding to the k classes), and has value 1 on the dimension corresponding to the true label class y, and has value 0 for the other dimensions. It can be viewed as a probabilistic vector, putting probability mass 1 on the class y, and mass 0 on the other classes.

Since Y is a one-hot vector, the cross-entropy reduces to the negation of the log on the output probability over the true class y. Minimizing the loss is maximizing the output probability over the true class y (i.e., pushing the vector p to be the same as the vector Y).

How to train a neural network?

Update the weights W to minimize the loss function

$$\frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} \ell(\mathbf{x}, y)$$

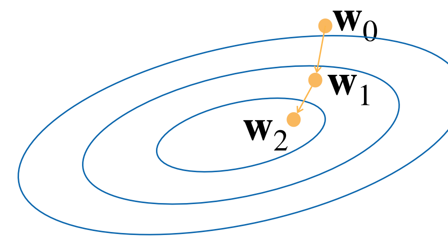


Use gradient descent!

Now we've specified the models and the loss, we are ready to specify the third element in the learning method: the optimization method for minimizing the loss.

For neural networks, typically we use gradient descent. More precisely, we use some variants of gradient descent.

Gradient Descent



- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For t =1,2,…
  - Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{\partial L}{\partial \mathbf{w}_{t-1}}$$

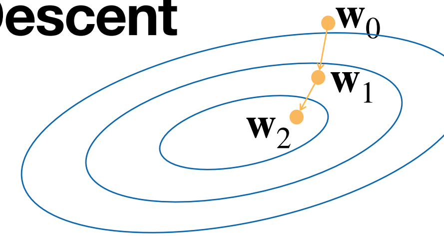
$$= \mathbf{w}_{t-1} - \alpha \frac{1}{|D|} \sum_{(\mathbf{x},y)\in D} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}}$$

D can be very large. Expensive

- Repeat until converges

One way is to do gradient descent.

Minibatch Stochastic Gradient Descent



- Choose a learning rate $\alpha > 0$
- Initialize the model parameters w_0
- For t =1,2,…
  - **Randomly sample a subset (mini-batch)** $B \subset D$
    Update parameters:

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \alpha \frac{1}{|B|} \sum_{(\mathbf{x},y)\in B} \frac{\partial \ell(\mathbf{x}, y)}{\partial \mathbf{w}_{t-1}}$$

The gradient w.r.t. all parameters is obtained by concatenating the partial derivatives w.r.t. each parameter

- Repeat

We typically use stochastic gradient descent.

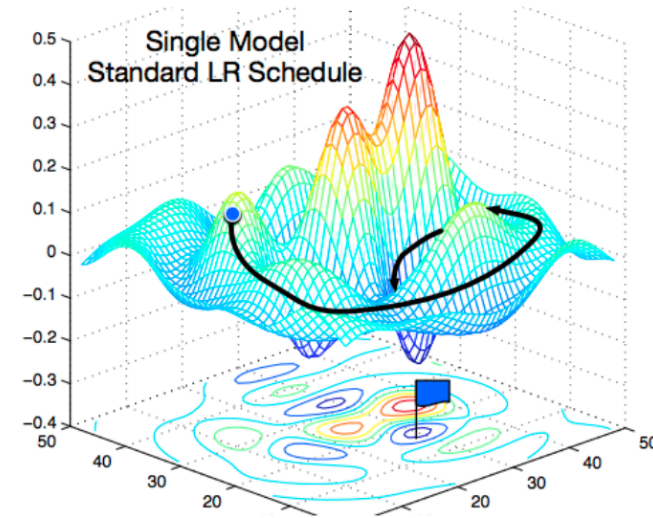
We initialize the model parameter (usually using Gaussian random numbers). We also pick some learning rate \alpha.
Then we run in iterations. In each iteration, randomly sample a batch of training data points; average the gradient of the loss on the sampled training data points w.r.t. the parameters; move along the negation of the average gradient direction with step size \alpha, to get a new set of parameter values. Repeat this until we run out of time budget or are satisfied with the loss.

What’s the gradient of the loss on a sampled training data point w.r.t. the parameters?

1. It’s a concatenation of the partial derivative w.r.t. each parameter.
2. The partial derivative w.r.t. one particular parameter: we can view the loss as a single-variable function on that particular parameter, and take the gradient of that function. That is, view all the other variables as constant, and compute the gradient. This is the partial derivative w.r.t. that particular parameter.

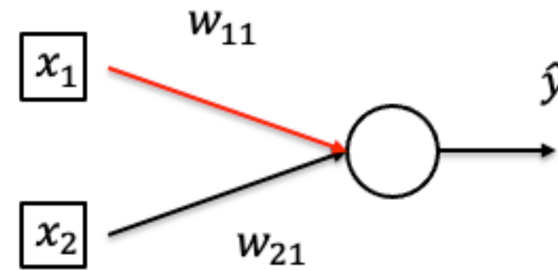
Note: Here we describe the algorithm as iterations and each iteration samples a batch. In practice, usually it’s slightly different: we run in epochs; in each epoch we randomly partition the training data into batches, and then use the batches one by one (ie, run in iterations and each iteration uses one batch).

Non-convex Optimization



[Gao and Li et al., 2018]

Calculate Gradient (on one data point)



sigmoid activation, binary cross-entropy loss

$\hat{y} = \sigma(w_1 x_1 + w_2 x_2)$

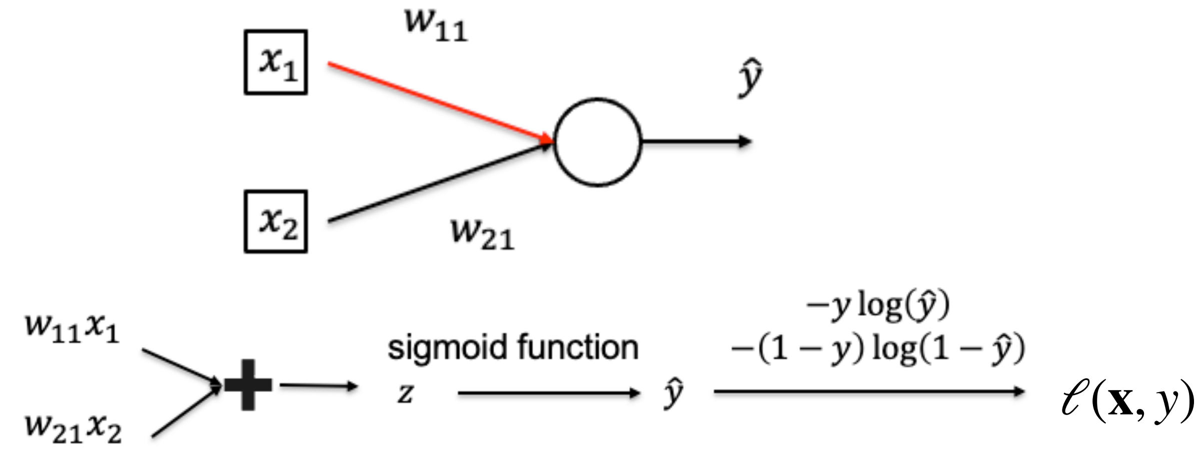
$\ell(\mathbf{x}, y) = -y \log \hat{y} - (1-y)\log(1-\hat{y})$

- Want to compute $\dfrac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

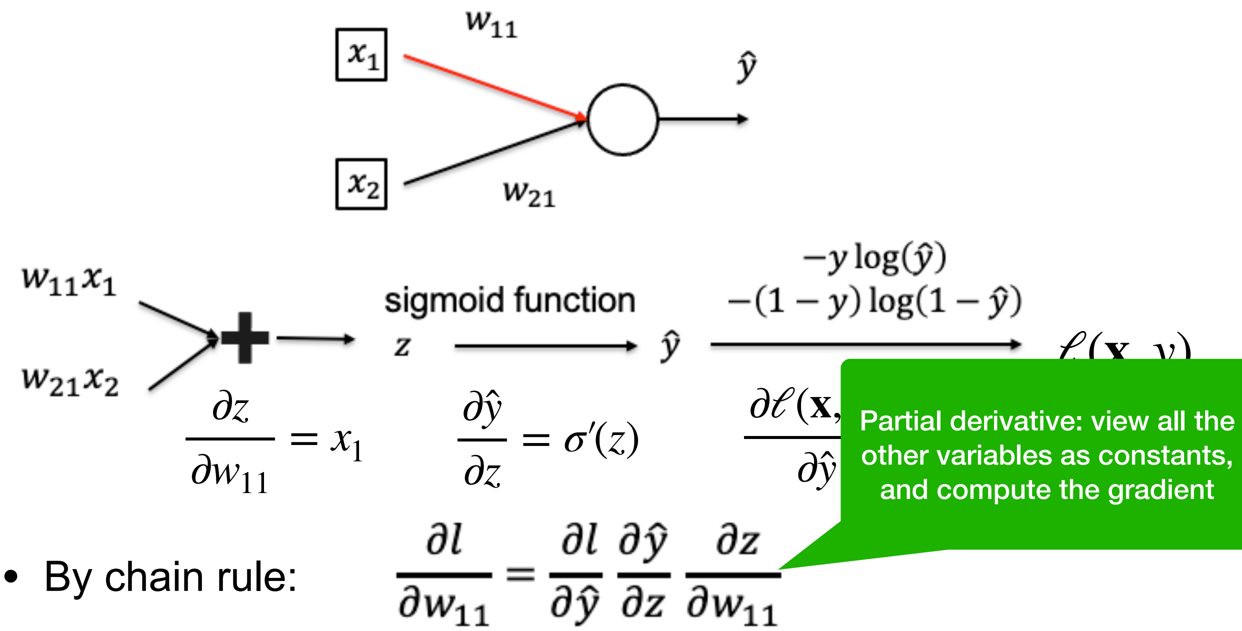
Now the question is how to compute the derivative of the loss on one data point w.r.t. a particular parameter. The method for this is called back propagation.

Here we give an example to demonstrate the method. For simplicity, we assume no bias.

Calculate Gradient (on one data point)

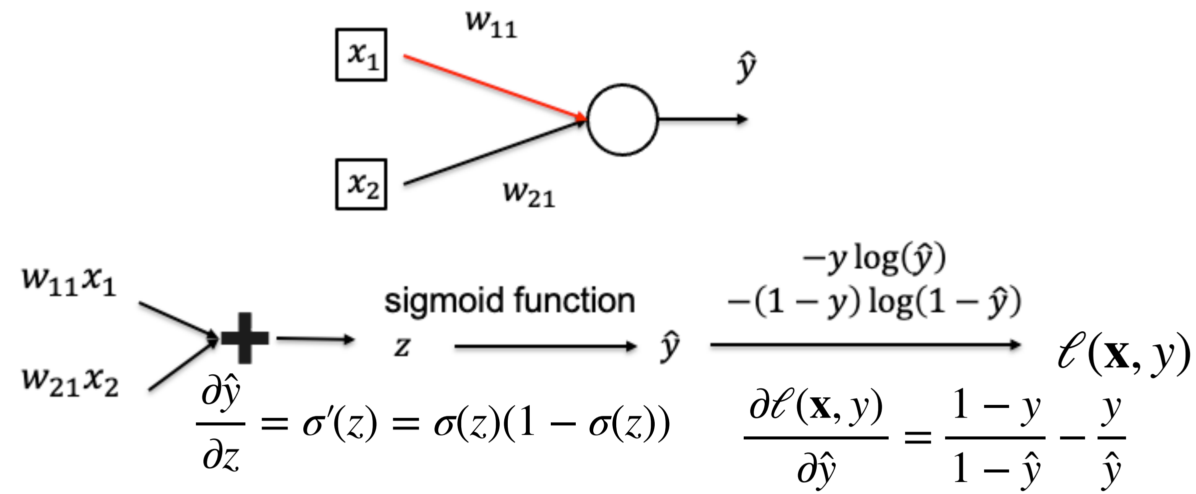


First write down the forward computation chain (computation graph) from input to the final loss on the data point.



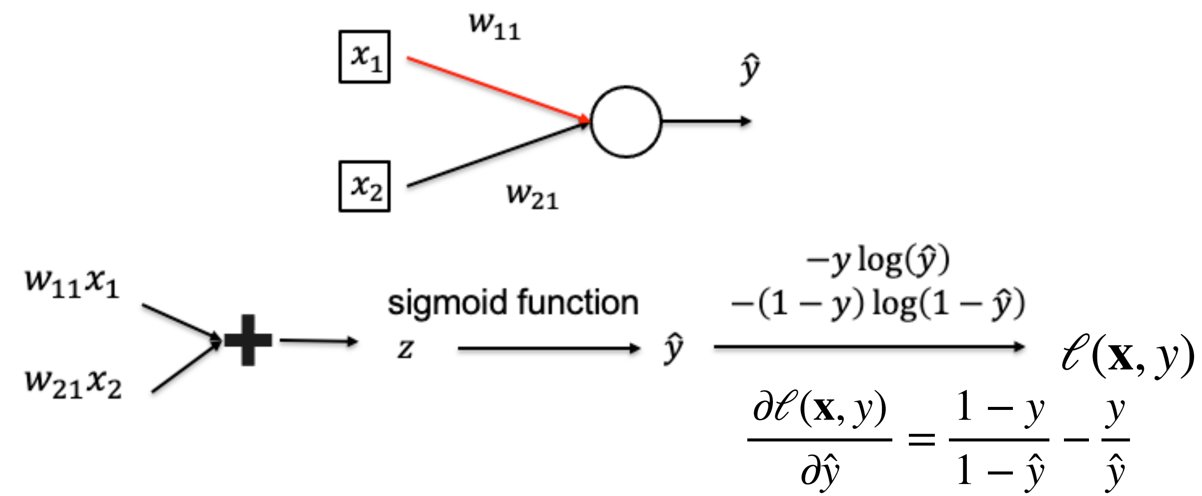
Next consider computing the gradient by chain rule. Then compute each term separately and multiply them.

Calculate Gradient (on one data point)



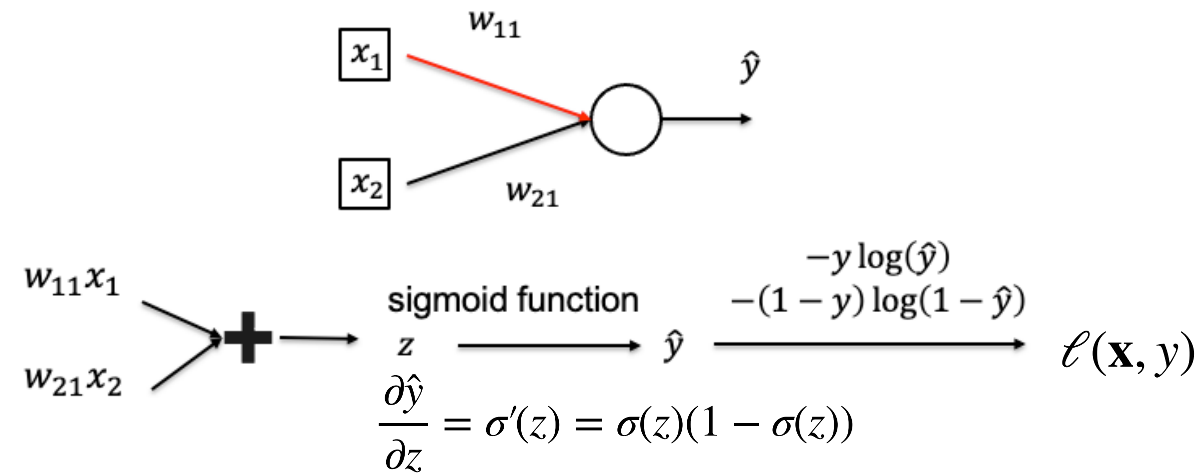
- By chain rule: $\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} x_1$

Calculate Gradient (on one data point)



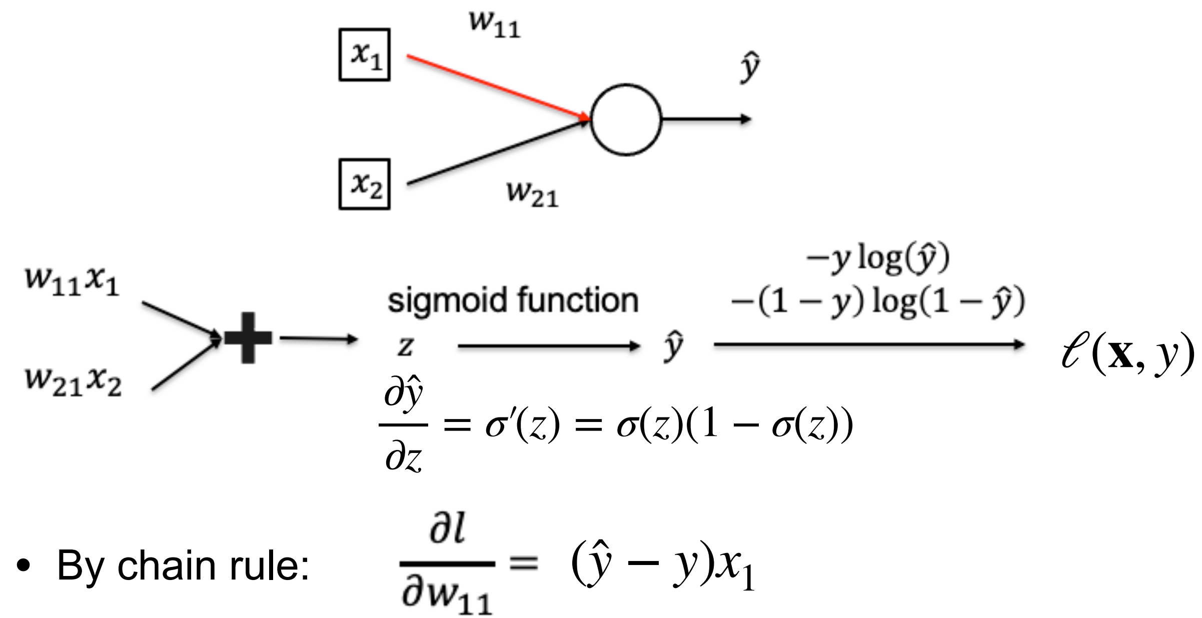
- By chain rule: $\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \; \hat{y}(1-\hat{y})x_1$

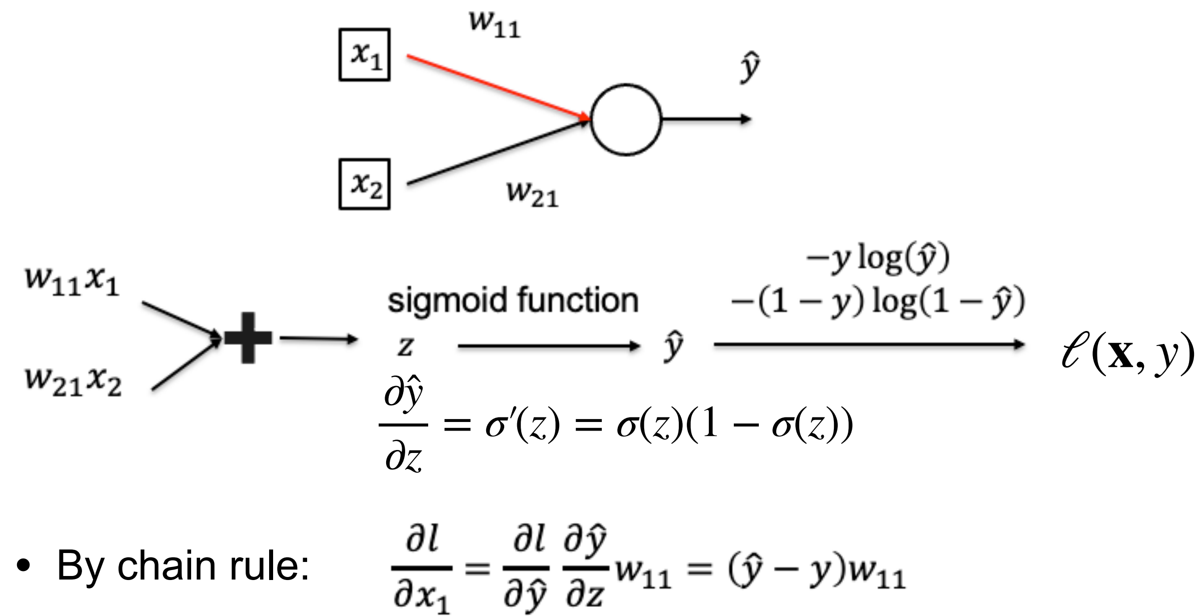
Calculate Gradient (on one data point)



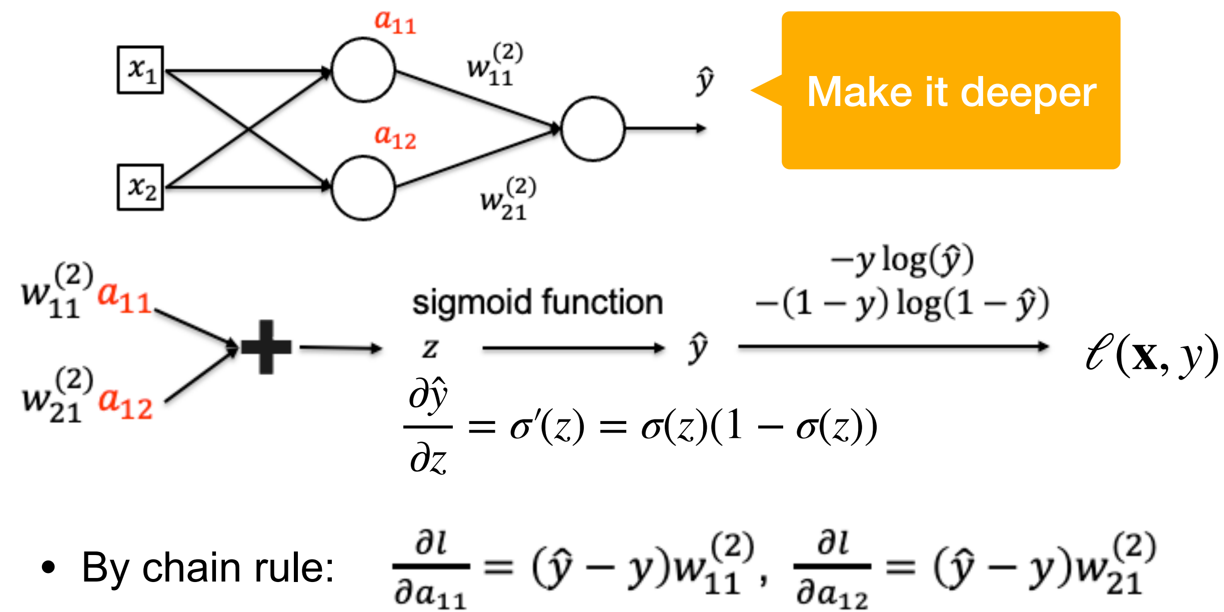
- By chain rule: $$\frac{\partial l}{\partial w_{11}} = (\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}})\hat{y}(1-\hat{y})x_1$$

Calculate Gradient (on one data point)



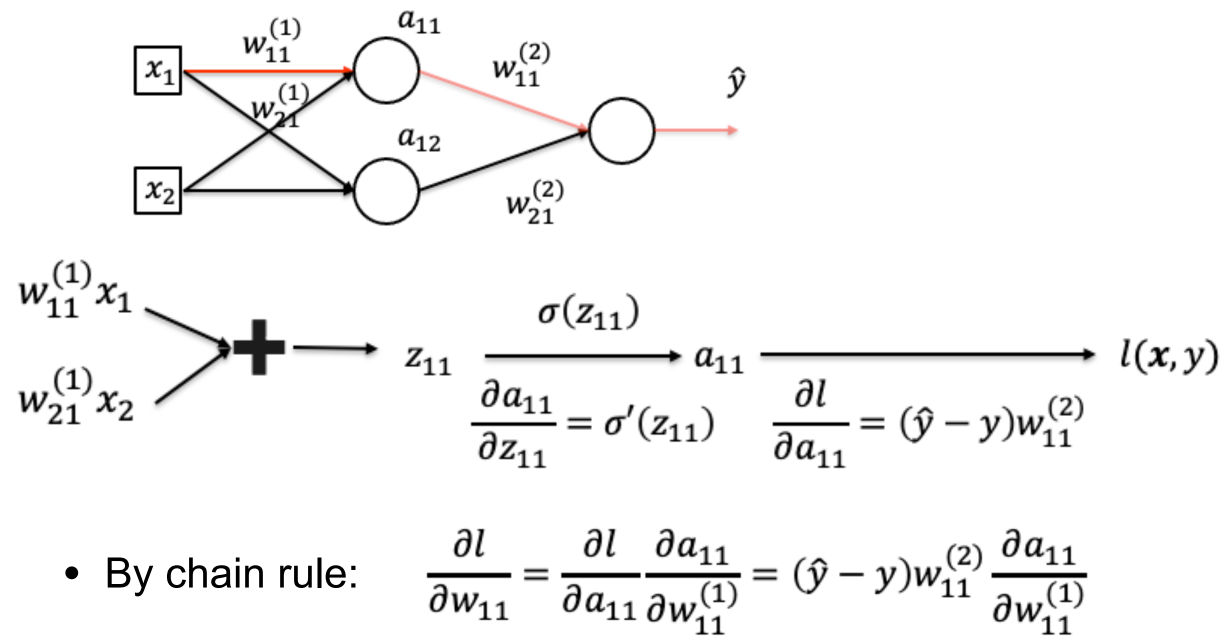


Similarly, we can also compute the derivative w.r.t. x_1 and x_2.



Now consider a deeper network.

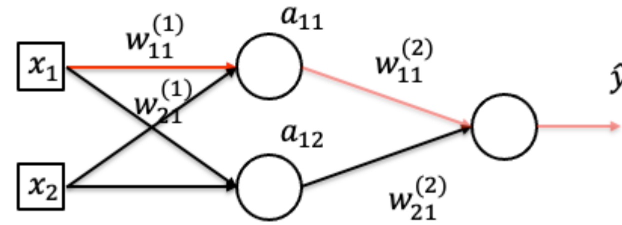
By our computation in the previous slide, we know the derivatives w.r.t. the output of the first layer a_11 and a_12. (just view them as the x_1 and x_2 in the previous slide!)

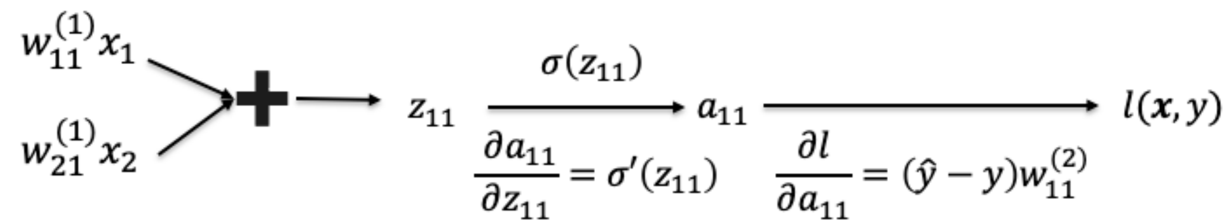


Now we can compute the derivative w.r.t. the weights in the first layer. We can apply chain rule and reuse the computed derivatives w.r.t. a_11 and a_12. All we need to do is to compute a step backward: the derivative of a_11 w.r.t. the weight w^{(1)}_11.

This is called back-propagation.

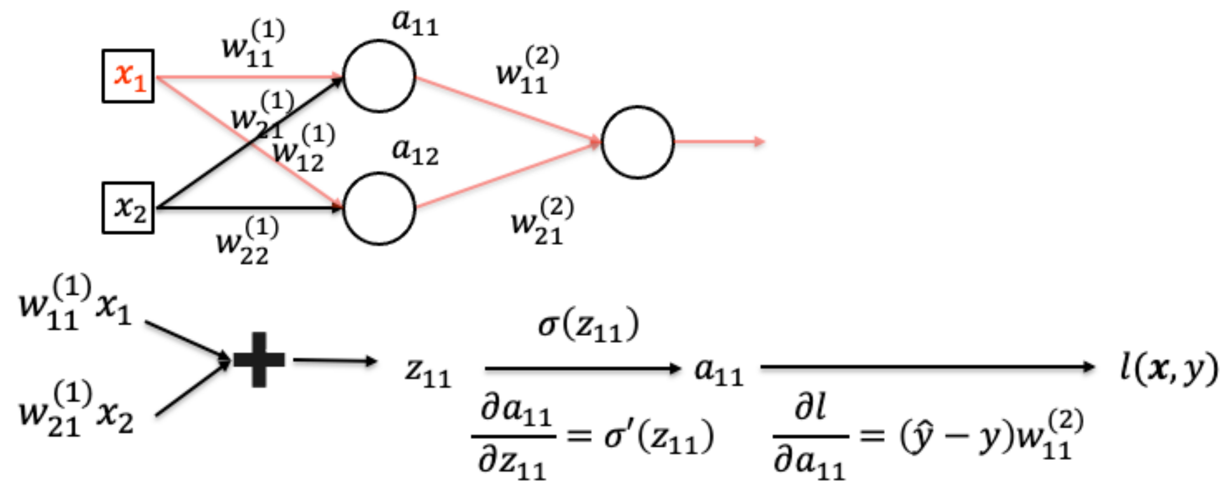
Calculate Gradient (on one data point)





- By chain rule: $\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y)w_{11}^{(2)} a_{11}(1 - a_{11})x_1$

Calculate Gradient (on one data point)



- By chain rule: $$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial x_1} + \frac{\partial l}{\partial a_{12}} \frac{\partial a_{12}}{\partial x_1}$$

Similarly, we can compute the derivatives w.r.t. the input.

Quiz Break

Gradient Descent in neural network training computes the ______ of a loss function with respect to the model ______ until convergence.

A gradients, parameters

B parameters, gradients

C loss, parameters

D parameters, loss

Quiz Break

Gradient Descent in neural network training computes the _______ of a loss function with respect to the model _______ until convergence.

A gradients, parameters

B parameters, gradients

C loss, parameters

D parameters, loss

Quiz Break

Suppose you are given a dataset with 1,000,000 images to train with. Which of the following methods is more desirable if training resources are limit but enough accuracy is needed?

A Gradient Descent

B Stochastic Gradient Descent

C Minibatch Stochastic Gradient Descent

D Computation Graph

Quiz Break

Suppose you are given a dataset with 1,000,000 images to train with. Which of the following methods is more desirable if training resources are limit but enough accuracy is needed?

A Gradient Descent

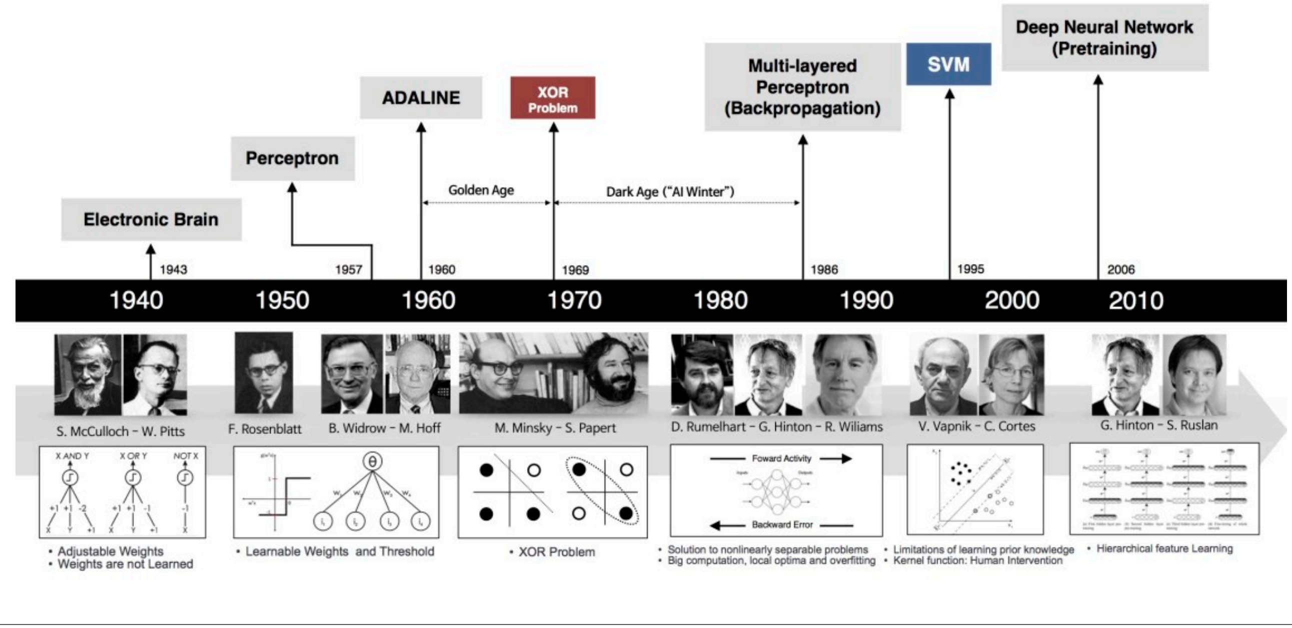
B Stochastic Gradient Descent

C Minibatch Stochastic Gradient Descent

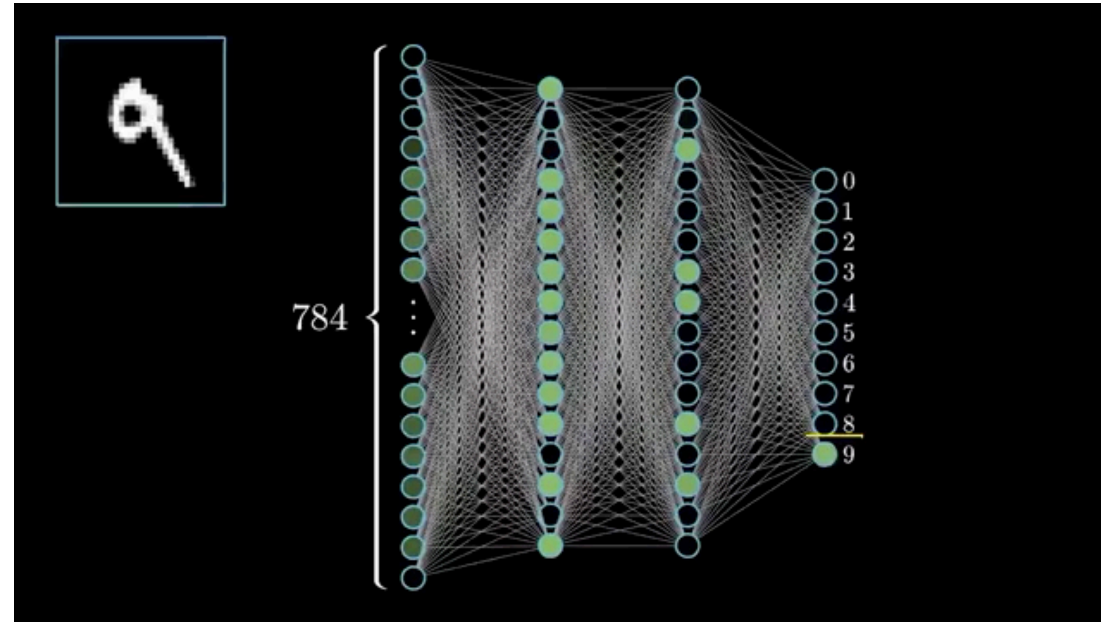
D Computation Graph

Too large dataset for limited training resources, so cannot use Gradient descent. SGD with one sample each time: may not be accurate. Computation graph: an auxiliary technique for doing gradient computation.

Brief history of neural networks

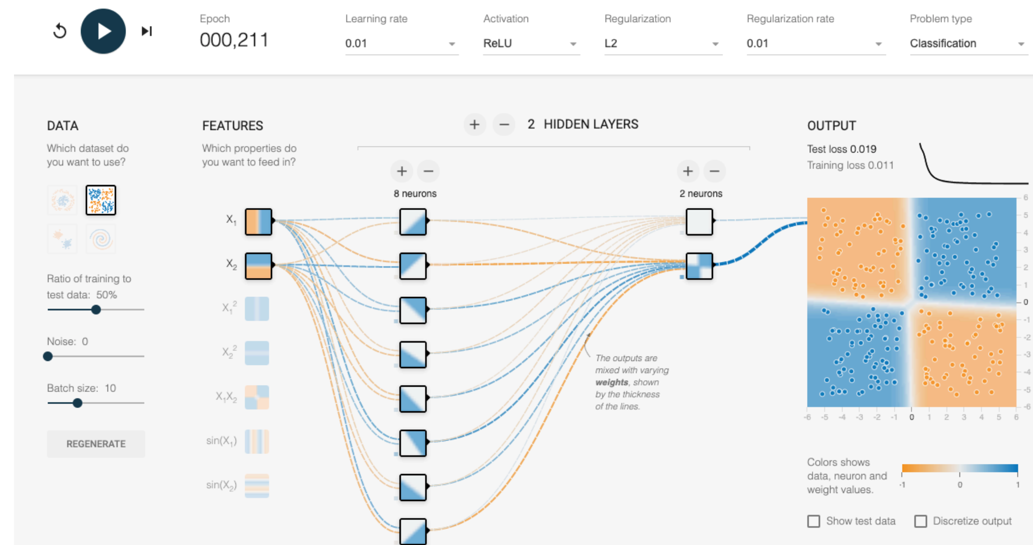


HW6



HW6 (working with MNIST dataset)

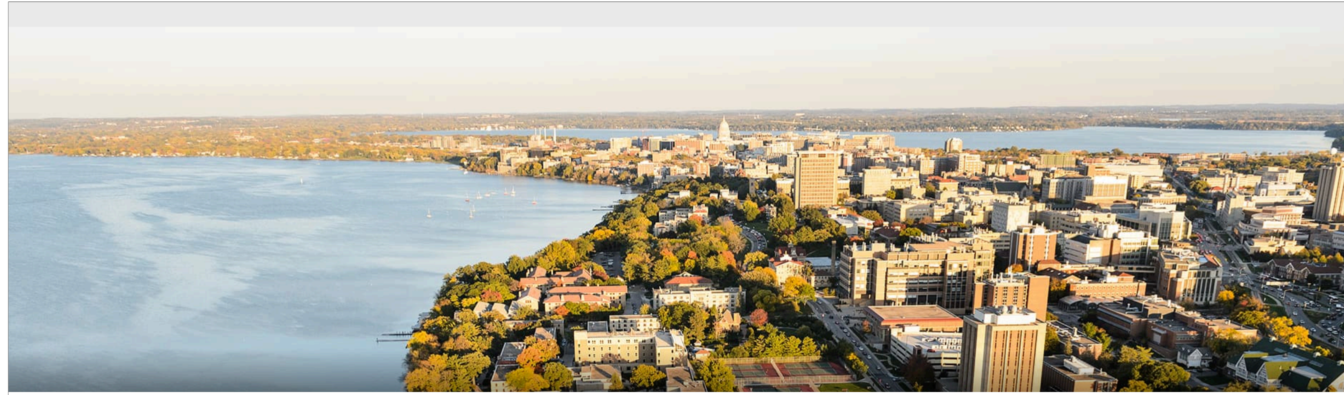
Demo: Learning XOR using neural net



- https://playground.tensorflow.org/

What we’ve learned today…

- Single-layer Perceptron Review
- Multi-layer Perceptron
  - Single output
  - Multiple output
- How to train neural networks
  - Gradient descent



Thanks!