Deep Learning

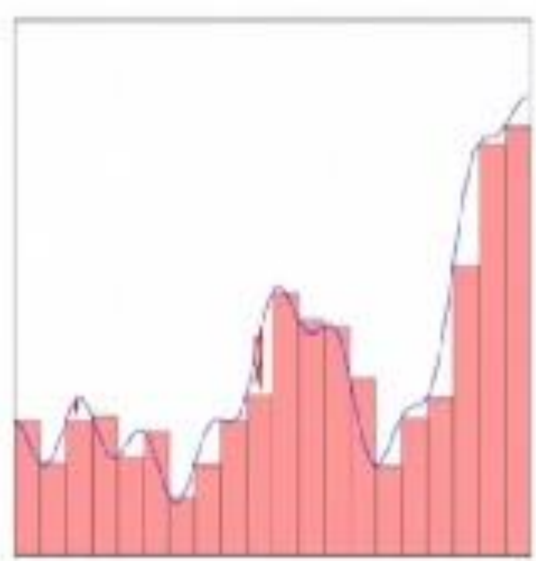
Representative Power of Multilayer Networks

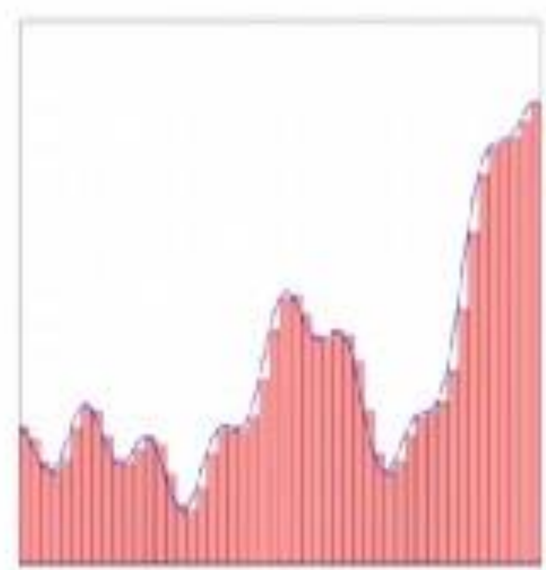
- A multilayer network of perceptrons with a single hidden layer can be used to approximate any Boolean function precisely

- A multilayer network of sigmoid neurons with a single hidden layer can be used to approximate any continuous function to any desired precision

Multilayer Network

- For any function $f(x)$: $R^n \longrightarrow R^m$, we can find a network with enough neurons, whose output $g(x)$ satisfies $|g(x) - f(x)| < \epsilon$
- Such an arbitrary function can be represented by several tower functions



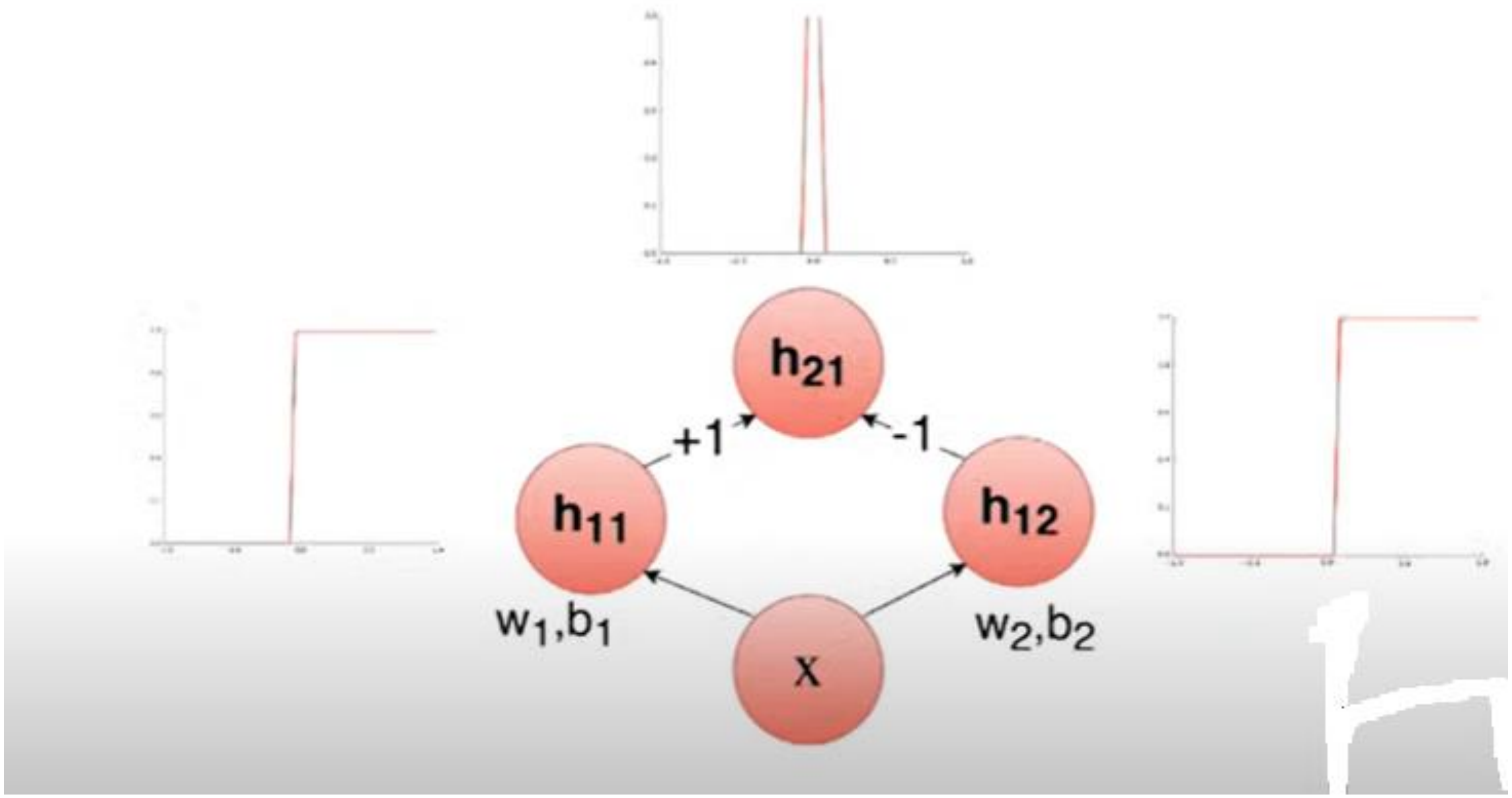


Multilayer Network

- All tower functions are similar and only differ in height and position on x-axis
- A black box takes some input and constructs a tower function
  - A network can add them up to approximate the function
- If we take the logistic function and set w to a very high value, we can recover step function
  - w controls the slope of the logistic function
- Can also adjust value of b to control position on x-axis at which function transitions from 0 to 1

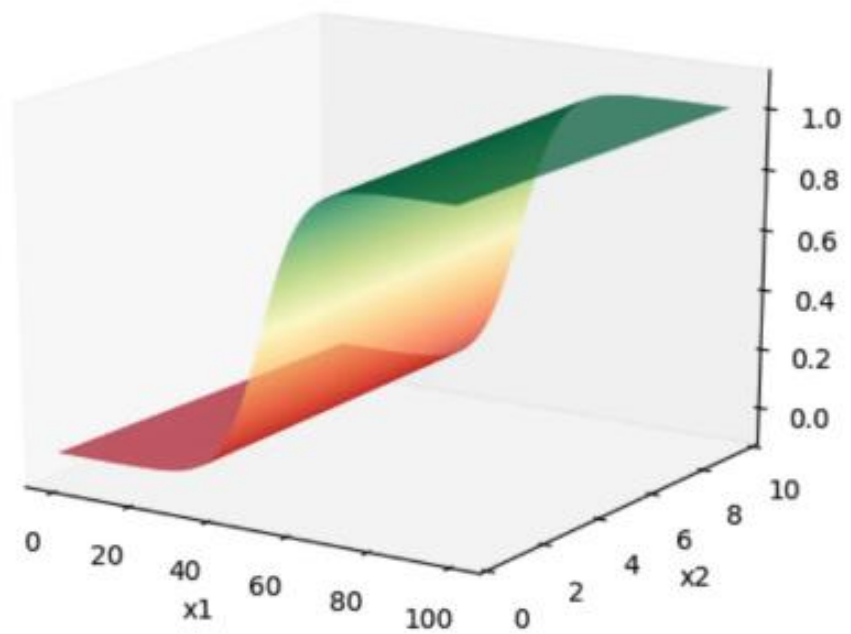
Multilayer Network

Take two such sigmoid functions, with different *b*’s, and subtract them – will get a tower function



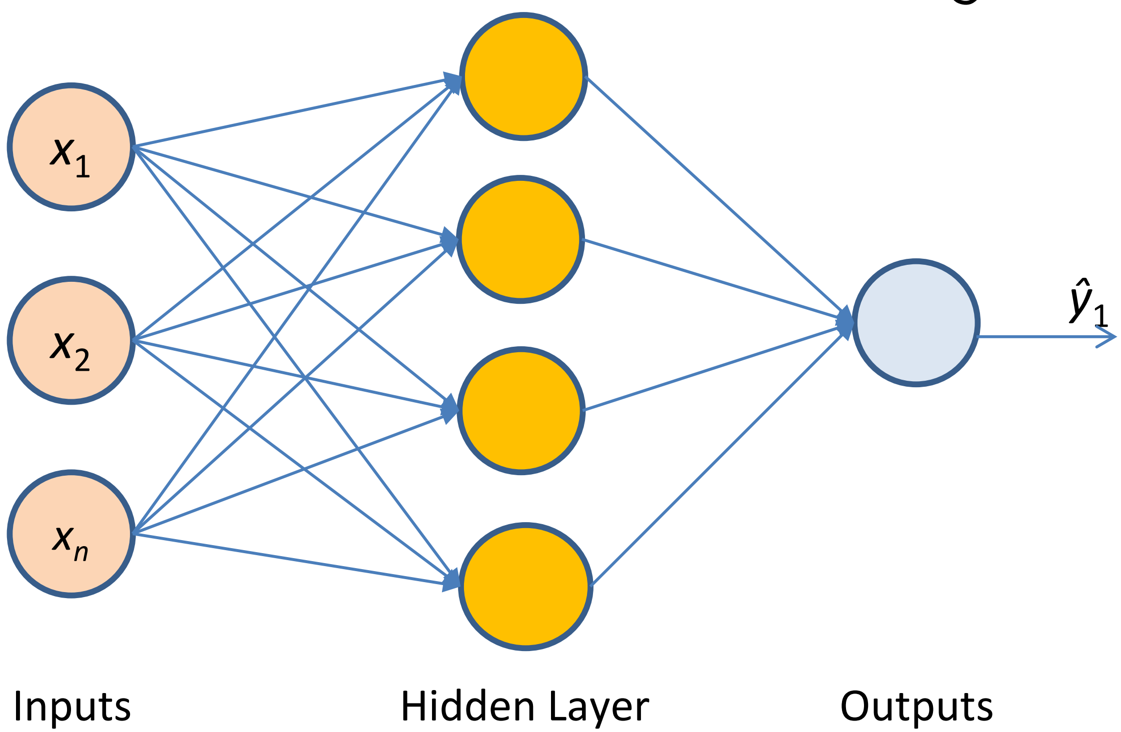
Multilayer Network

- More input parameters??
- Ex. 2 parameters



Single Hidden Layer Neural Network

2 Layer NN



Hidden Layer: States of nodes are unobserved

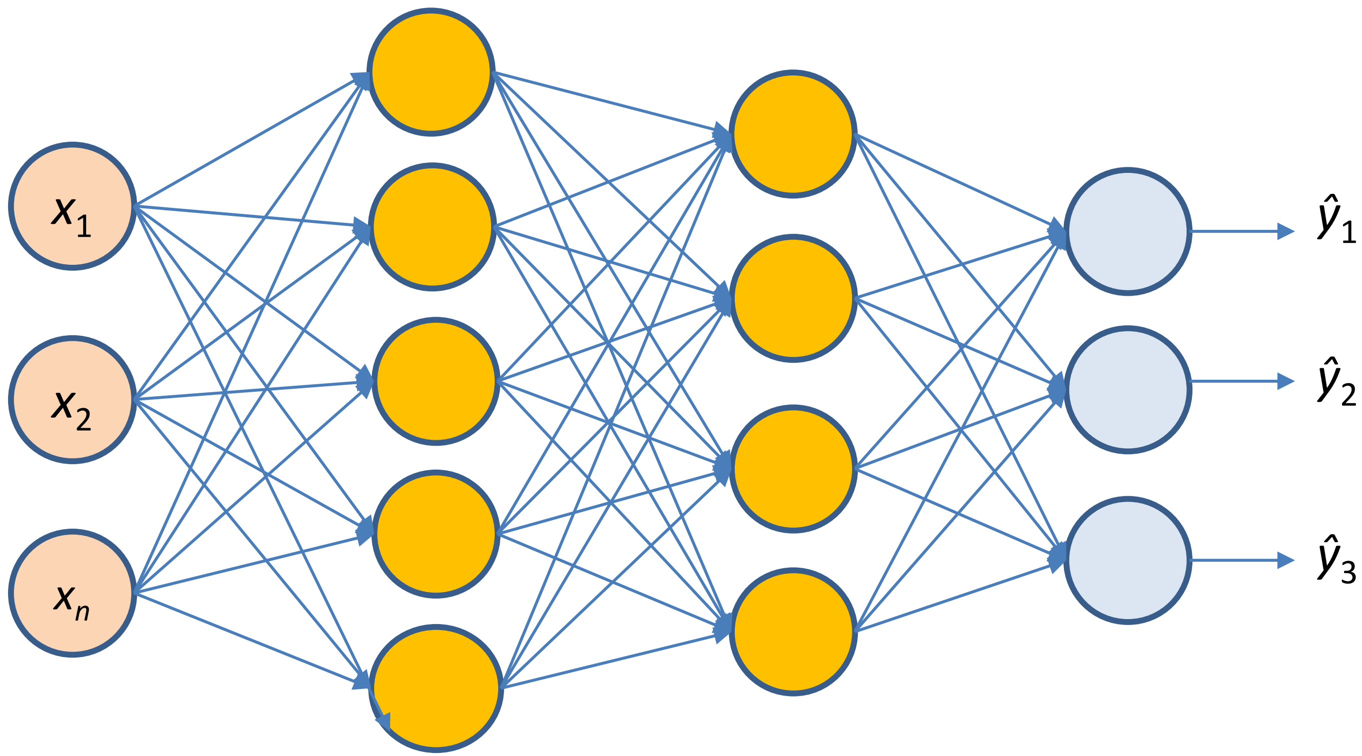
Inputs are densely connected to perceptrons, hence they are called **Dense** layers or **Fully Connected** layers

Feedforward Neural Network

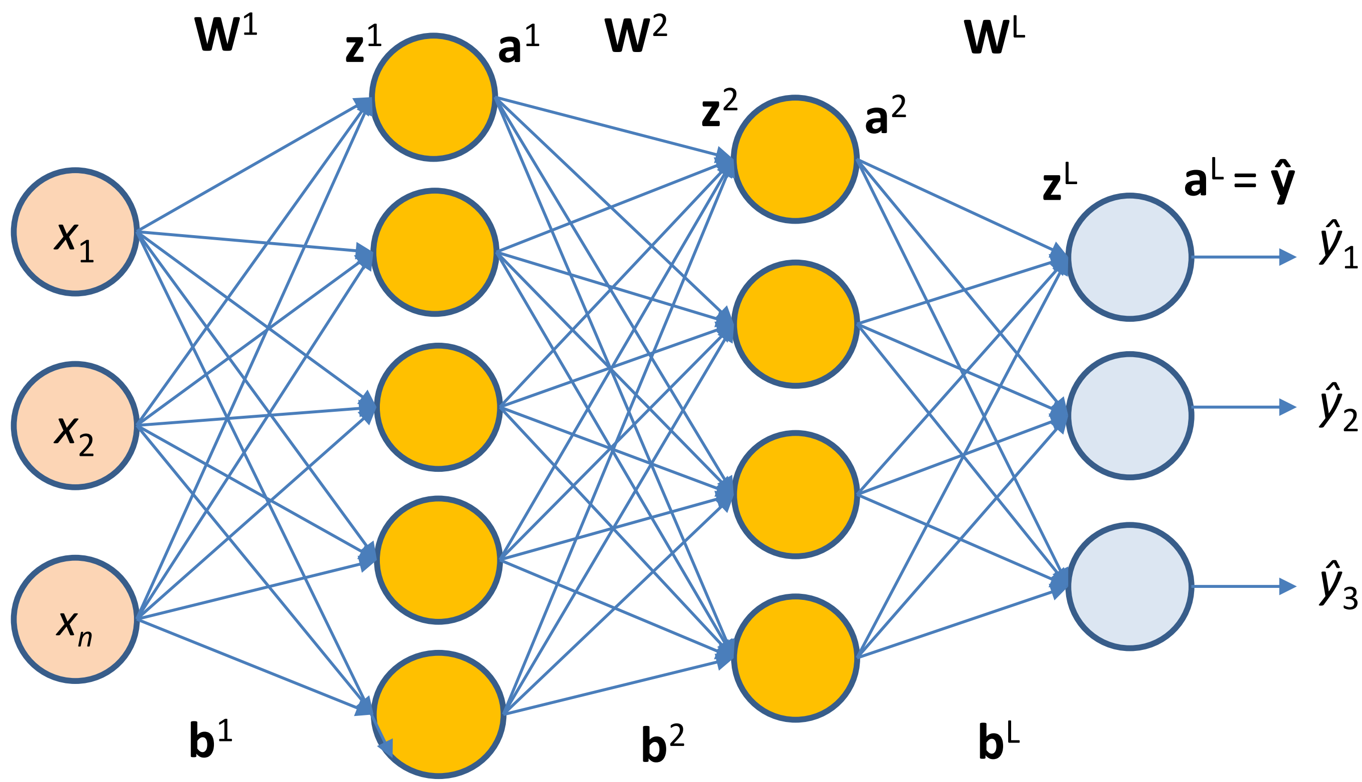
- Input is an *n*-dimensional vector (0^{th} layer) $\in R^n$

W^1

- Network has *L*-1 hidden layers
- 1 output layer containing *k* neurons (ex. for *k* classes)
- Each neuron – aggregation and activation

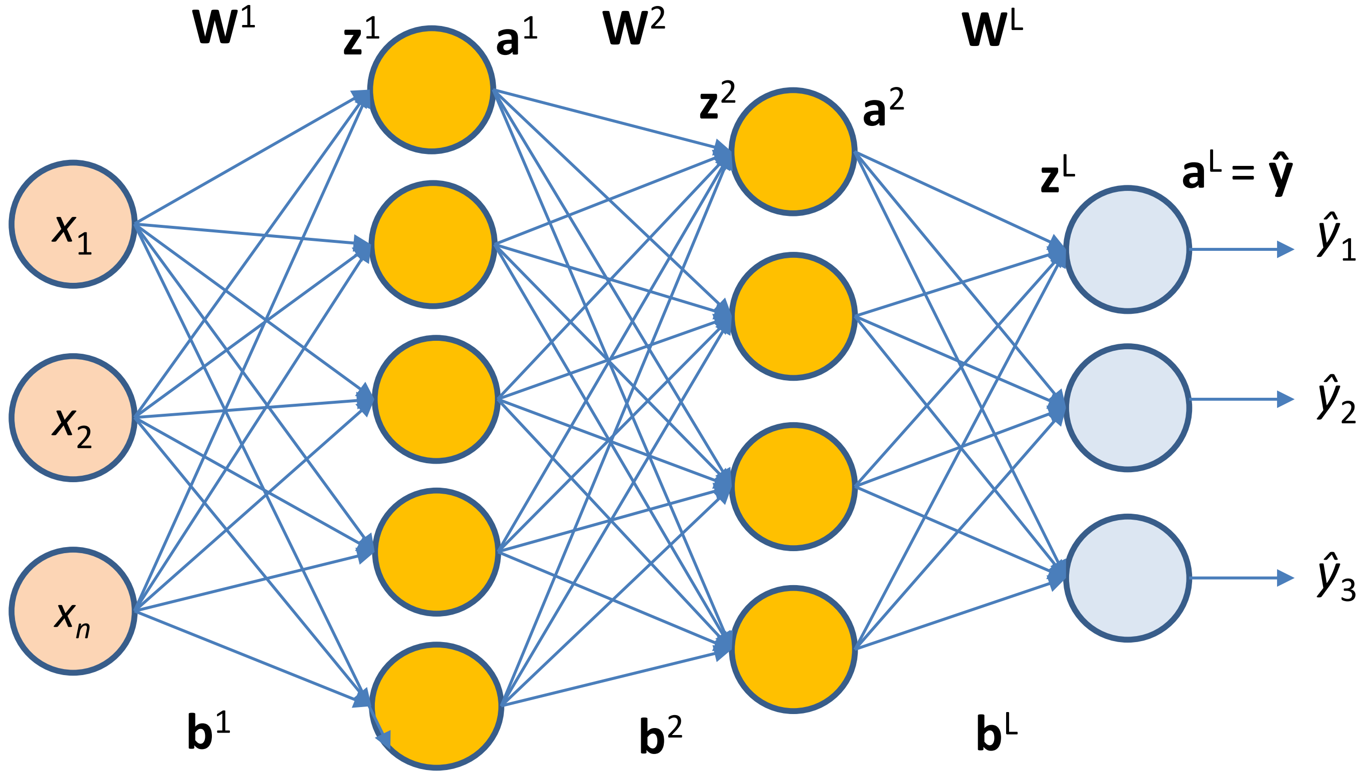


Feedforward Neural Network



Assuming n^i neurons in hidden layer h^i, $W^i \in R^{n(i-1)*ni}$ and $b^i \in R^{ni}$ between layers i -1 and i for $0<i<L$

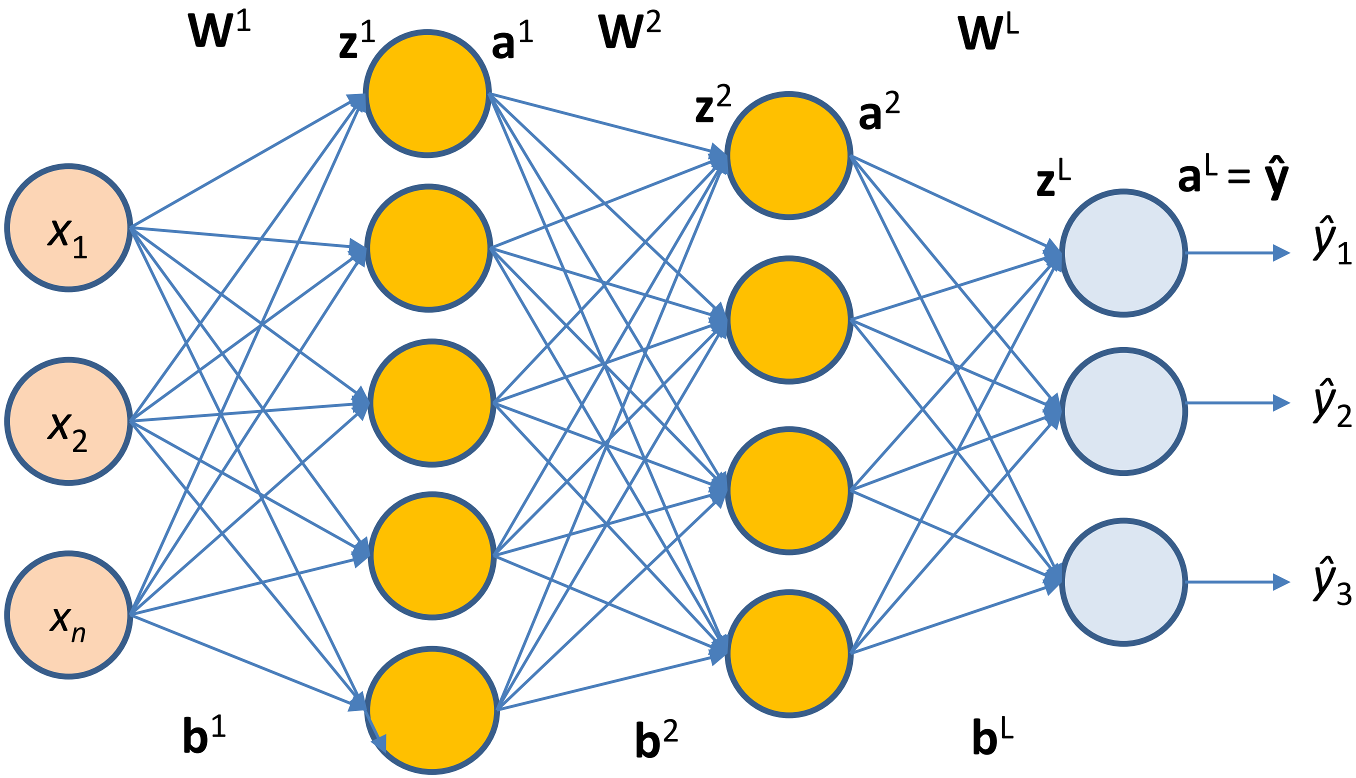
$W^L \in R^{ni*k}$ and $b^L \in R^k$ between last hidden layer and output layer



Aggregation at layer i : $\mathbf{z}^{\mathbf{i}} = \mathbf{W}^{\mathbf{i}}\, \mathbf{a}^{\mathbf{i-1}} + \mathbf{b}^{\mathbf{i}}$

For first hidden layer: $\mathbf{z}^1 = \mathbf{W}^1\, \mathbf{a}^0 + \mathbf{b}^1$

$$\begin{pmatrix} z_1{}^1 \\ z_2{}^1 \\ z_3{}^1 \end{pmatrix} = \begin{pmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} \sum W_{1i} x_i + b_1 \\ \sum W_{2i} x_i + b_2 \\ \sum W_{3i} x_i + b_3 \end{pmatrix}$$



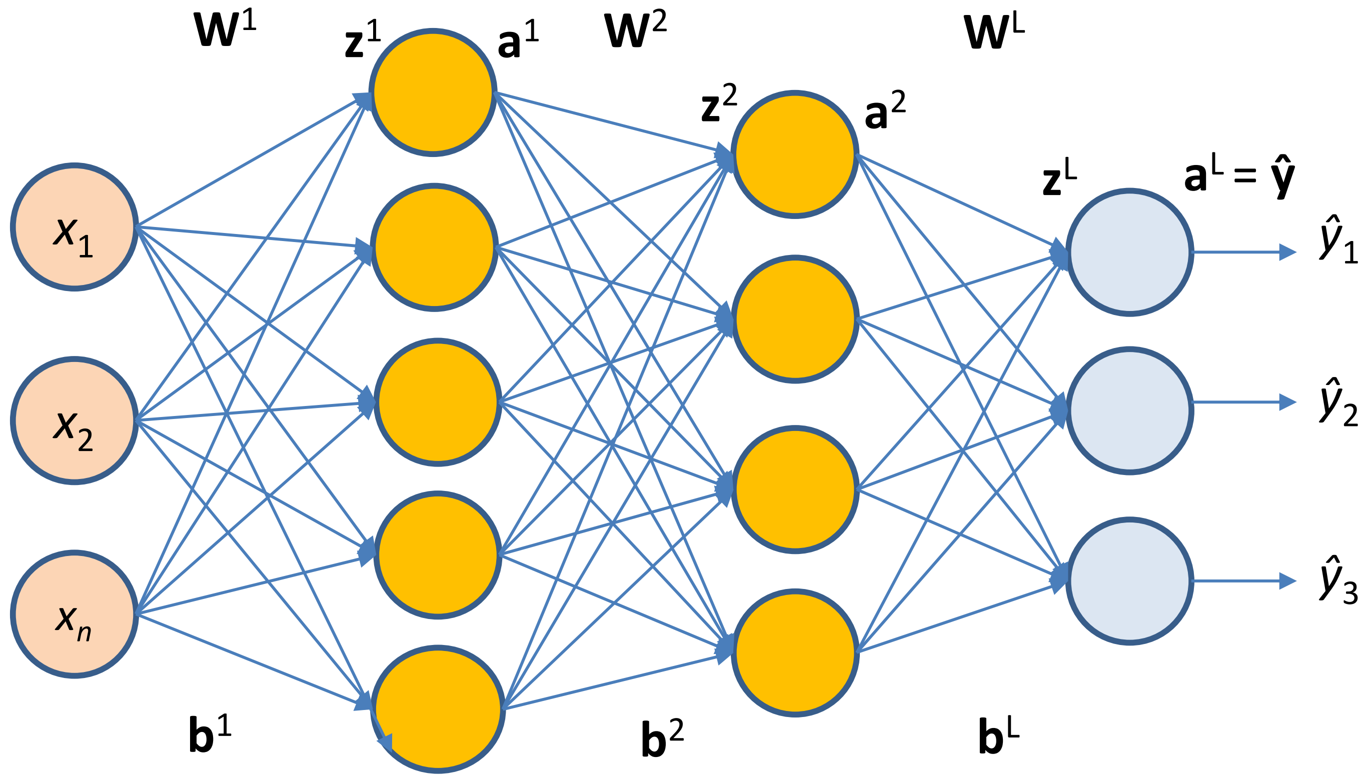
Activation at layer i = $g(\mathbf{z}^{\mathbf{i}}) = g(\mathbf{b}^{\mathbf{i}} + \mathbf{W}^{\mathbf{i}}\, \mathbf{a}^{\mathbf{i-1}})$

For first hidden layer: $g(\mathbf{z}^{\mathbf{1}}) = g(\mathbf{b}^{\mathbf{1}} + \mathbf{W}^{\mathbf{1}}\, \mathbf{a}^{\mathbf{0}})$

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} g(z_1) \\ g(z_2) \\ g(z_3) \end{pmatrix}$$

Eg. $g(z_1) = \sigma(z_1) = 1 / (1 + e^{-z1})$

g: activation function (logistic, tanh, linear etc.)



Aggregation at output layer L $= z^L = \mathbf{W^L\, a^{L\text{-}1} + b^L}$

$$z_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b$$

$$z_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b$$

Activation at output layer L $= \hat{\mathbf{y}} = g(z^L) = g(\mathbf{W^L\, a^{L\text{-}1} + b^L})$

$$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} g(z_1) \\ g(z_2) \end{bmatrix}$$

Learning parameters

In given example, dimensions of parameters:

- $\mathbf{W^1}$: $n^1 * n$ $\mathbf{b^1}$:n^1
- $\mathbf{W^2}$: $n^2 * n^1$ $\mathbf{b^2}$:n^2
- $\mathbf{W^L}$: $n^2 * k$ $\mathbf{b^L}$:k

- Assuming L layers and n^i neurons in hidden layer h^i and k neurons in output layer, no. of parameters to be learned:
  - Weights: $(L\text{-}1)*(n^{i-1} * n^i) + (n*k)$ for $0<i<L$
  - Bias: $(L\text{-}1)*n^i + k$

Learning parameters

- **Data:** $\{x_i, y_i\}$ $\quad\quad\quad\quad$ $i = 1..m$
- **Model:**

$$\hat{\mathbf{y}} = f(\mathbf{x}) = g(\mathbf{W}^3 g(\mathbf{W}^2 g(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2) + \mathbf{b}^3)$$

$$\hat{\mathbf{y}} = [\hat{y}^1 \quad \hat{y}^2 \dots \hat{y}^k]$$

- **Algorithm:** Gradient Descent with back Propagation
- **Loss/Error function:** Sum of squared error loss

$min \frac{1}{N} \sum_{i=1}^{m} \sum_{j=1}^{k} (\hat{y}_j^i - y_j^i)$ for i^{th} sample for all classes j

Learning parameters

- Gradient Descent:

  t:=0;

  $max\_iterations$:=1000;

  Initialize $\boldsymbol{\theta_0} := [\mathbf{W^1}_0, \ldots \mathbf{W^L}_0, \mathbf{b^1}_0 \ldots \mathbf{b^L}_0]$;

  while t++ < $max\_iterations$ do

  $\boldsymbol{\theta_{t+1}} := \boldsymbol{\theta_t} - \eta \nabla \boldsymbol{\theta_t}$;

  end

where, $\nabla\theta_t = [\ \frac{\partial L(\theta)}{\partial W_t}, \frac{\partial L(\theta)}{\partial b_t}\ ]^T$

$\nabla\theta$ composed of:

- $\nabla W^1, \nabla W^2, \ldots \nabla W^{L-1} \in R^{n(i-1)xni}$, $\nabla W^L \in R^{nxk}$
- $\nabla b^1, \nabla b^2, \ldots \nabla b^{L-1} \in R^{ni}$, $\nabla b^L \in R^k$

Loss function

- Loss function should capture how much $\hat{y}_i$ deviates from y_i
- $y_i \in R^n$ then squared error loss can be used:

$$L(\theta) = (1/N)*\sum (y_i - \hat{y}_i)^2$$

- Problems with squared error loss:

$$\frac{\partial L(w,b)}{\partial w} = (\hat{y} - y) * \hat{y} * (1 - \hat{y}) * x$$

  - If $y_i = 1$ and $\hat{y}_i \sim 0$, $\frac{\partial L(w,b)}{\partial w} \sim 0$ Undesirable
  - If $y_i = 0$ and $\hat{y}_i \sim 1$, $\frac{\partial L(w,b)}{\partial w} \sim 0$ Undesirable
  - Weight updation becomes very slow

Loss function

- Information content (IC):
  - Events with high probability have low information content
    - "The sun will rise tomorrow"
  - Events with low probability have high information content
    - "There will be a cyclone tomorrow"
- $IC(\text{A}) = -\log_2(p(\text{A}))$
- Entropy: Expected information content = $\sum p_i * IC(i)$

  $= - \sum p_i \log_2(\text{p}_\text{i})$

Loss function

Entropy: y_i = [0 1 0 0] //Team B wins game

$\hat{y}_i$ = [0.2 0.1 0.4 0.3] //Our prediction

10K 5K 8K 1K //Profit for each team win

Expected profit??

- Entropy: Expected information content = $\sum p_i \, IC(i)$

  $= - \sum p_i \log_2(p_i)$

Loss function

- Cross-entropy: gives a measure on how close a predicted distribution is to a true distribution
  - True distribution p_i , Estimated distribution q_i
  - Estimated information content = $- \sum p_i \log_2(q_i)$
  - Capture difference between two probability distributions
  - If prediction is close to actual, cross entropy will be low

$L(\theta) = - \sum y_c \log_2(\hat{y}_c)$	for all k classes
$y_c = 1$	if $c = t$ (true class)
$= 0$	otherwise
$L(\theta) = - \log_2(\hat{y}_t)$	

Loss function

- Objective function for classification:
  - Cross-entropy Loss

    minimize: $L(\theta) = -\log_2(\hat{y}_t)$

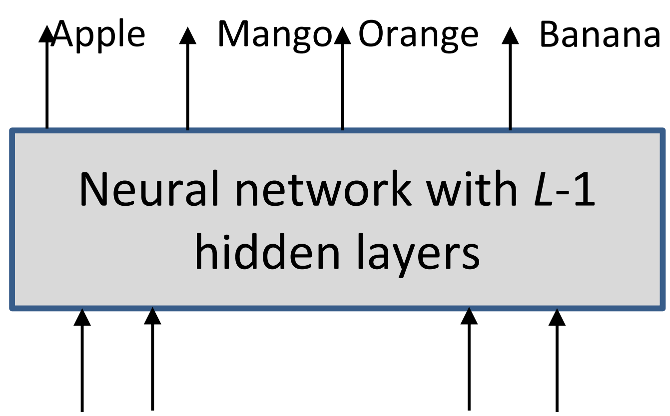
$\hat{y}_t$: predicted probability of correct event

$\log_2(\hat{y}_t)$: probability that x belongs to t^{th} class, log-likelihood of data

Output and Loss function

- Output activation function:
  - Sum of outputs should be 1
  - $\hat{y}$ should be a probability distribution
  - Sigmoid – probabilities will be $0<p<1$ but sum not equal to 1

$y_i = \{1 \quad 0 \quad 0 \quad 0\}$



Classification problem

apple

Output Activation Function

- Softmax function

  $z^L = b^L + W^L a^{L-1}$

  $\hat{y} = g(z^L_j) = e^{z}_j / \sum e^{z}_j$ for $j = 1..k$

  z^L_j is j^{th} element of z^L

- Example: $z^L = [10 \quad 20 \quad -30]$

  $\hat{y} = [e^{10}/(e^{10} + e^{20} + e^{-30}) \quad e^{20}/(e^{10} + e^{20} + e^{-30}) \quad e^{-30}/(e^{10} + e^{20} + e^{-30})\ ]$

NOTE: Exponent converts –ve values to +ve values

Loss function

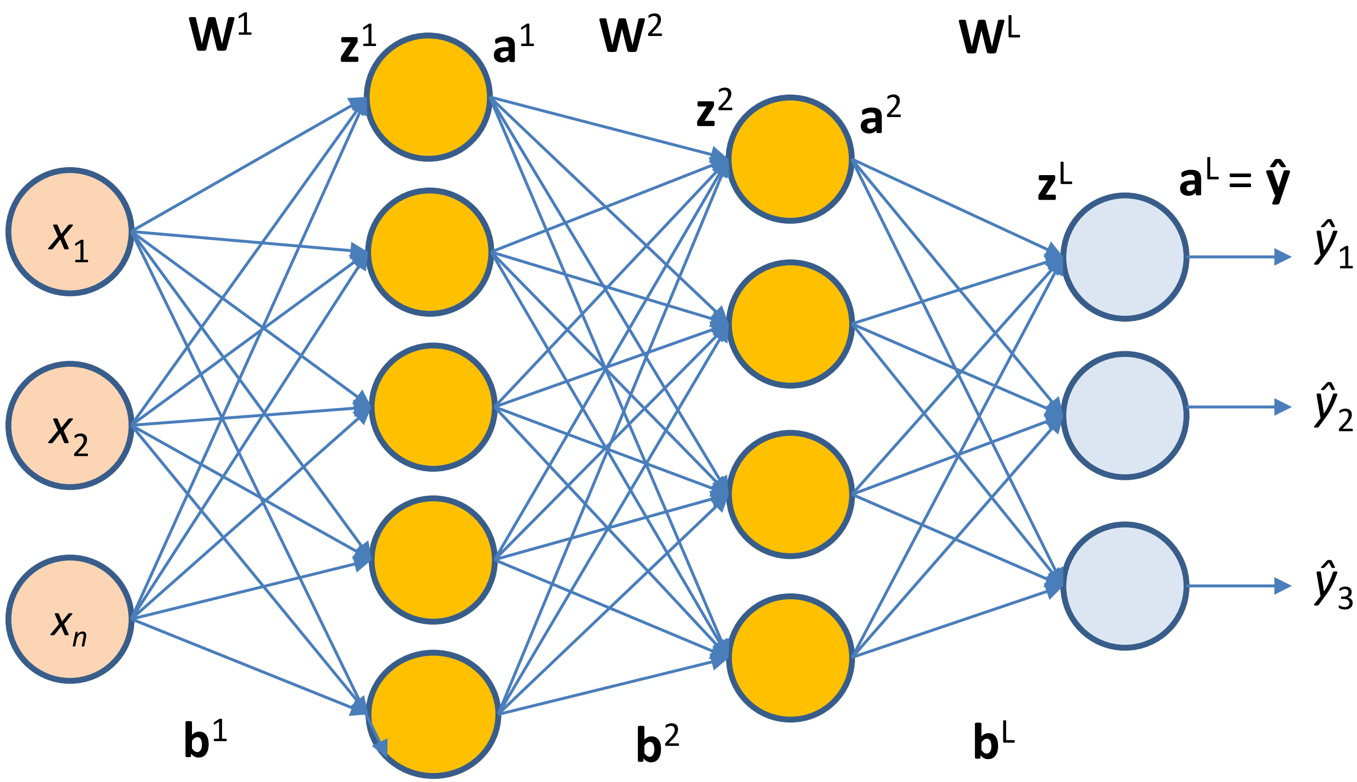
	Outputs	
	Real values	**Probabilities**
Output activation	Linear	Softmax
Loss function	Squared error	Cross-entropy

Backpropagation

How to compute $\nabla\theta$ composed of:

$\nabla W^1, \nabla W^2, \ldots \nabla W^{L-1} \in R^{nxn}$, $\nabla W^L \in R^{nxk}$

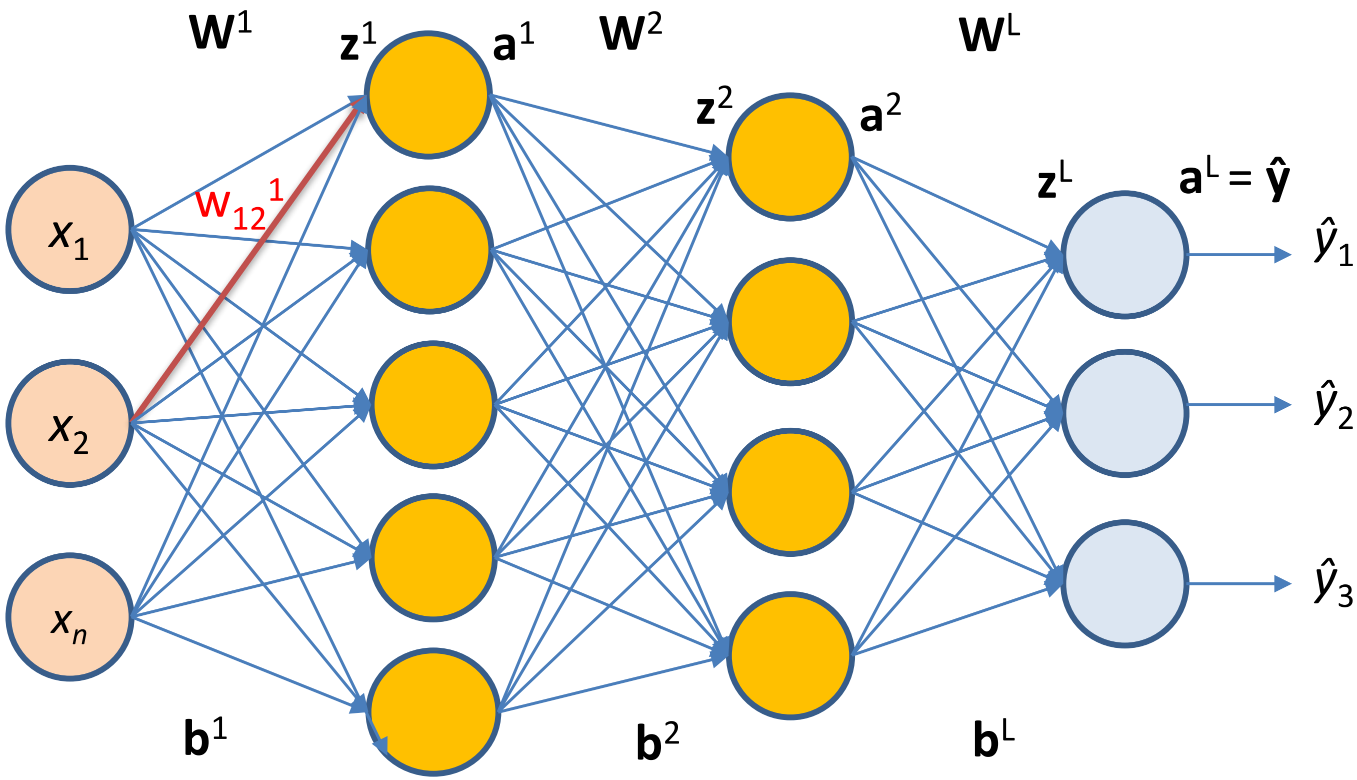
$\nabla b^1, \nabla b^2, \ldots \nabla b^{L-1} \in R^n$, $\nabla b^L \in R^k$



Backpropagation

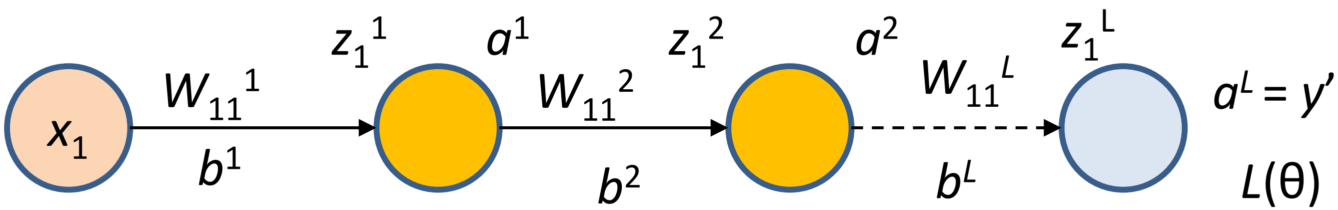
Assuming classification problem, $L(\theta) = -\log_2(\hat{y}_t)$

- To learn weight w_{12}^{1} use SGD and compute $\frac{\partial L(w,b)}{\partial W_{12}}$



Backpropagation

Assume a deep thin network, who is responsible for the loss??



Find derivative by chain rule:

$$\frac{\partial L(\theta)}{\partial W_{11}^1} = \underbrace{\frac{\partial L(\theta)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1^L}}_{\text{Output layer}} * \underbrace{\frac{\partial z_1^L}{\partial a_1^2} * \frac{\partial a_1^2}{\partial z_1^2}}_{\text{Previous hidden layer}} * \underbrace{\frac{\partial z_1^2}{\partial a_1^1} * \frac{\partial a_1^1}{\partial z_1^1}}_{\text{Previous hidden layer}} * \underbrace{\frac{\partial z_1^1}{\partial W_{11}^1}}_{\text{Weights}}$$

If we change W_{11}, how much does the loss change

$$L = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial \hat{y}}{\partial z} = \sigma(z)(1-\sigma(z))$$

$$= \hat{y}(1-\hat{y})$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z}$$

$$= \cancel{\hat{y}}\cancel{(1-\hat{y})}\left(\frac{-y}{\cancel{\hat{y}}} + \frac{1-y}{\cancel{1-\hat{y}}}\right)$$

$$= -y(1-\hat{y}) + \hat{y}(1-y)$$

$$= -y$$

$$= \hat{y} - y$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial z} \times \frac{\partial z}{\partial w}$$

$$= (\hat{y} - y)x.$$

$$\frac{\partial L}{\partial a^1} = \underbrace{\frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z^2}} \times \frac{\partial z^2}{\partial a^1}$$

$$\Rightarrow \quad z^2 = w_2 a_1 + b$$

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

$$\frac{\partial L}{\partial z^1} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z^2} \times \frac{\partial z^2}{\partial a_1} \times \frac{\partial a^1}{\partial z^1}$$

$$\frac{\partial L}{\partial w_{11}^{2}} = \underbrace{\frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z}} \times \frac{\partial z}{\partial w}$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}} = -4$$

$$= -4$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z} = \hat{y} - y = -0.75$$

$$\frac{\partial L}{\partial w} = (-0.75)(0.37) = -0.2775$$

$$w_{11}^{2} = w_{11}^{2} - \eta \times \frac{\partial L}{\partial w_{11}^{2}} = 12 - 0.00(-0.2775$$

$$= 12.0028.$$

$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial Z_1^2} \times \frac{\partial z_1^2}{\partial a_1^1} \times \frac{\partial a_1^1}{\partial w_{11}^1}$$

$$\frac{\partial a_1^1}{\partial w_{11}^1} = \frac{\partial a_1^1}{\partial z_1^1} \times \frac{\partial z_1^1}{\partial w_{11}}$$

$$= -0.75 \times 12 \times 0.5\,(1-0.5) \times 60$$

$$= -135$$

Homework

$$\frac{\partial L}{\partial w_{13}^1} =$$