CMPE 258, Deep Learning

Neural Network, backpropagation

Feb 20, 2018

DMH 149A

Taehee Jeong

Ph.D., Data Scientist

SJSU SAN JOSÉ STATE UNIVERSITY

Assignment_1

Grading was updated.

Please talk with me if you need more time.

1 (10pts). Linear regression with one variable

2 (30pts). Linear regression with two variables

3 (60pts). Linear regression with multiple variables

- from scratch (for loop, gradient descent)
- using matrix (gradient descent)
- using normal equation
- using scikit-learn linear regression model
- using TensorFlow gradient descent method

Assignment_2

Please submit a hard copy today.

1 (30pts). Polynomial regression / overfitting / regularization

2 (30pts). Polynomial regression with train/validation/test

3 (40pts). Regularization with Tensorflow

- using L2 penalty (Ridge)
- using L1 penalty (Lasso)
- using matrix (gradient descent)
- using scikit-learn linear regression model
- using TensorFlow gradient descent method

Logistic Regression

$$\hat{y} = \sigma(W^T x + b)$$

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

$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1 - y^i)\log(1 - \widehat{y^i})]$$

<Machine Learning, Emily Fox & Carlos Guestrin>

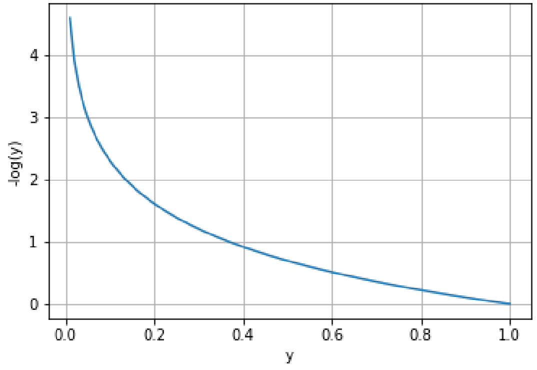
Logistic Regression

Loss function

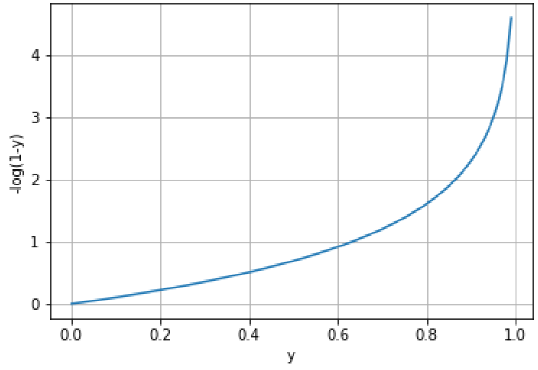
Given x, want $\widehat{y^i} \approx y^i$

$$L = -[y^i \log(\widehat{y^i}) + (1 - y^i)\log(1 - \widehat{y^i})]$$

If y = 1, $L = -\log(\widehat{y^i})$



If y = 0, $L = -\log(1 - \widehat{y^i})]$



Logistic Regression

Regularization

Cost function

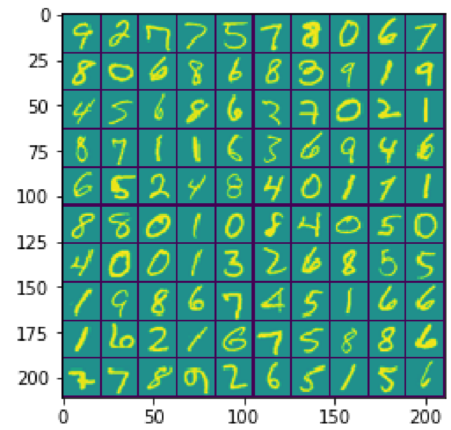
$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^i \log(\widehat{y^i}) + (1 - y^i)\log(1 - \widehat{y^i})] + \frac{\lambda}{m}\sum_{j=1}^{n} W_j^2$$

gradient

$$\frac{\partial J}{\partial W_0} = -\frac{1}{m}\sum_{i=1}^{m}[\,(\widehat{y^i} - y^i)x_j^i] \quad \text{for j=0}$$

$$\frac{\partial J}{\partial W_j} = -\frac{1}{m}\sum_{i=1}^{m}[\,(\widehat{y^i} - y^i)x_j^i]+\frac{2\lambda}{m}W_j \quad \text{for } j \geq 1$$

One-versus-all (OvA)



One-versus-the-rest

For example, one way to create a system that can classify the digit images into 10 classes (from 0 to 9) is to train 10 binary classifiers, one for each digit (a 0-detector, a 1-detector, a 2-detector, and so on).

Then when you want to classify an image, you get the decision score from each classifier for that image and you select the class whose classifier outputs the highest score.

Softmax function

Softmax score for class k

$$s_k(x) = W^k \cdot x$$

Softmax function

$$\boxed{p_k = \frac{\exp(s_k(x))}{\sum_{j=1}^{K} \exp(s_j(x))}}$$

- K is the number of classes.
- **s**(**x**) is a vector containing the scores of each class for the instance **x**.
- p_k is the estimated probability that the instance **x** belongs to class k given the scores of each class for that instance.

Softmax classifier

- Multiclass classifier
- Provides normalized class probabilities

$$p_k = \frac{\exp(s_k(x))}{\sum_{j=1}^{K} \exp(s_j(x))}$$

$$\boxed{\hat{y} = \underset{k}{\operatorname{argmax}}\, p_k}$$

It predicts the class with the highest estimated probability.

One-hot encoding

Method 1

y=[0; 1; 2; 3; 4; 5]

y	0	1	2	3	4	5
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	0	1

One-hot encoding

Method 2

$$y = \begin{bmatrix} 1 & 2 & 3 & 0 & 2 & 1 \end{bmatrix} \text{ is often converted to } \begin{array}{cl} \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} & \begin{matrix} \text{class} = 0 \\ \text{class} = 1 \\ \text{class} = 2 \\ \text{class} = 3 \end{matrix} \end{array}$$

<deep learning, Andrew Ng>

Biological Neurons

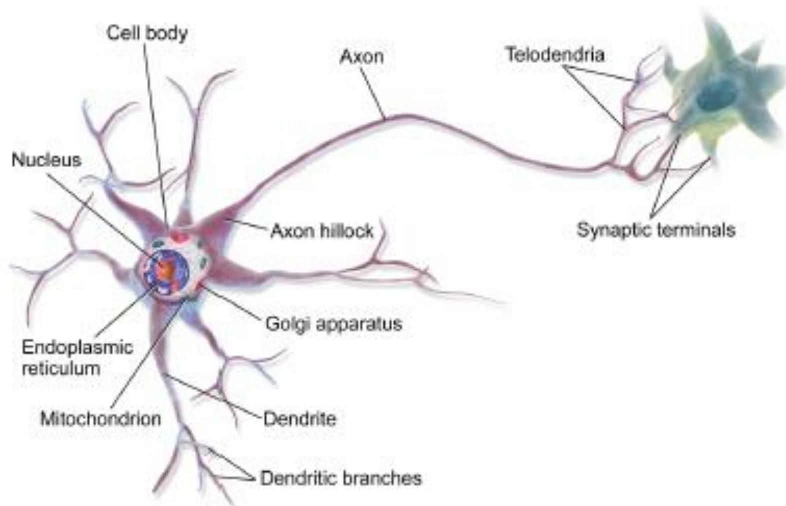
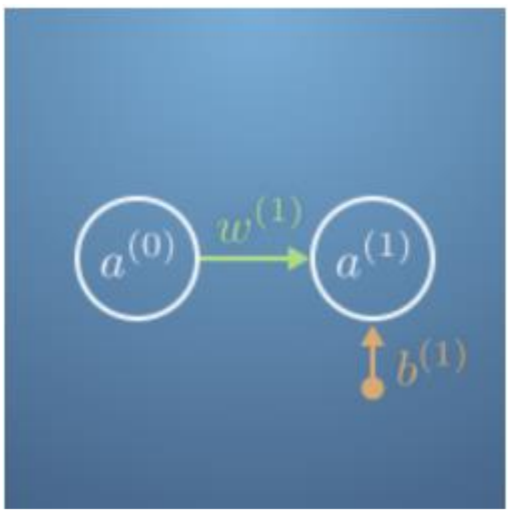


Image by Bruce Blaus (Creative Commons 3.0). Reproduced from *https://en.wikipedia.org/wiki/Neuron*.

Artificial neurons

Two neurons



<Mathematics for Machine Learning>

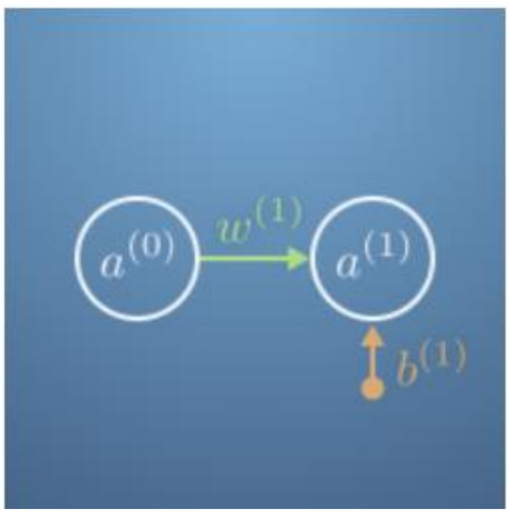
Univariate logistic regression

$$a^{[0]} = x^{[0]}$$

$$a^{[1]} = \sigma(w^{[1]} \cdot x^{[0]} + b^{[1]})$$

Artificial neurons

Two neurons



<Mathematics for Machine Learning>

Univariate logistic regression

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[1]} = w^{[1]} \cdot x^{[0]} + b^{[1]}$$

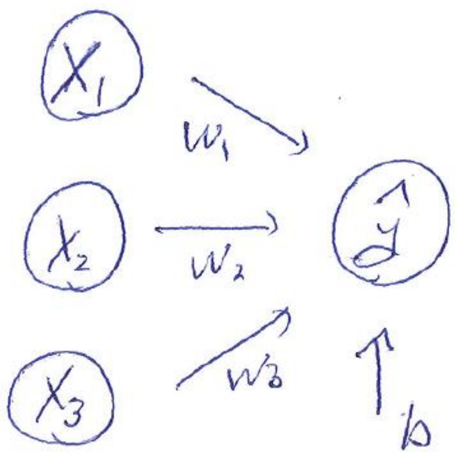
$$\sigma = \frac{1}{1+\exp(-z)}$$; activation function

$a^{[0]}$: input node

$a^{[1]}$: activation node
or output node

Neural Network

Multivariate Logistic Regression

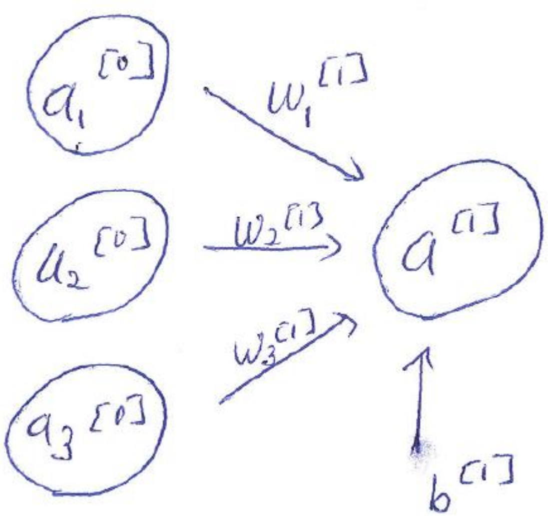


$$z = b + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3$$

$$\hat{y} = \sigma(z)$$

Neural Network

Multivariate Logistic Regression

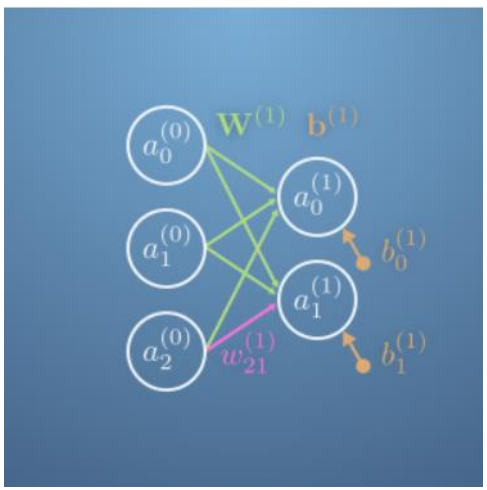


$$z^{[1]} = b^{[1]} + w_1^{[1]} \cdot a_1^{[0]} + w_2^{[1]} \cdot a_2^{[0]} + w_3^{[1]} \cdot a_3^{[0]}$$

$$a^{[1]} = \hat{y} = \sigma(z^{[1]})$$

Neural network

Multiclass classifier



<Mathematics for Machine Learning>

$$z_1^{[1]} = b_1^{[1]} + w_{11}^{[1]} \cdot a_1^{[0]} + w_{21}^{[1]} \cdot a_2^{[0]} + w_{31}^{[1]} \cdot a_3^{[0]}$$

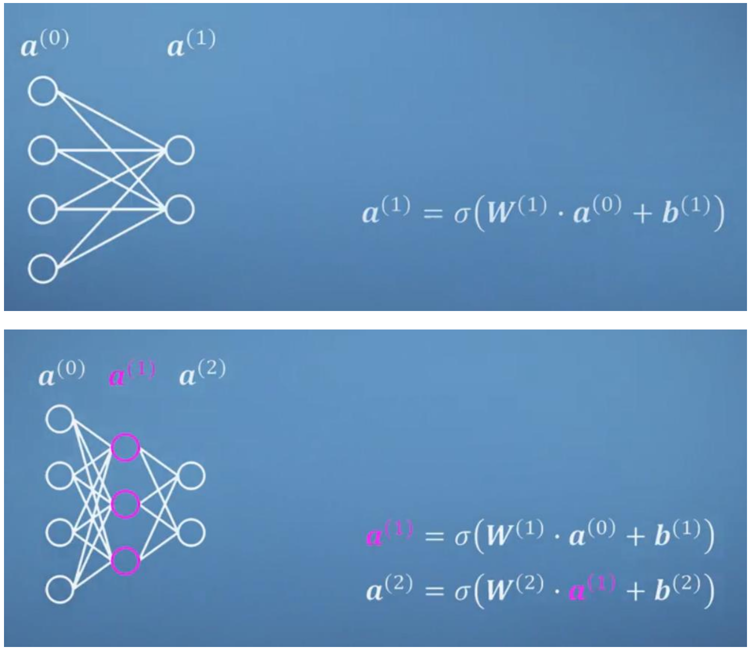
$$z_2^{[1]} = b_2^{[1]} + w_{12}^{[1]} \cdot a_1^{[0]} + w_{22}^{[1]} \cdot a_2^{[0]} + w_{32}^{[1]} \cdot a_3^{[0]}$$

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

Neural network

Hidden Layer

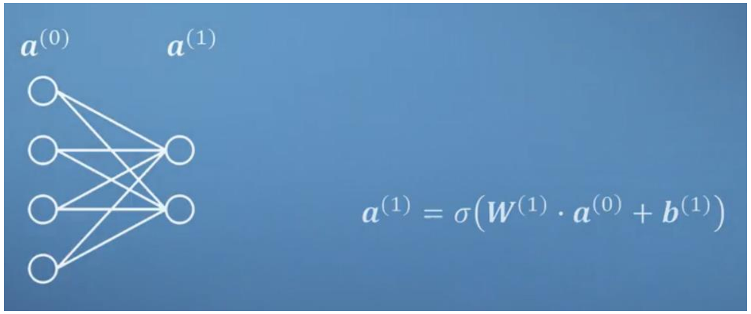


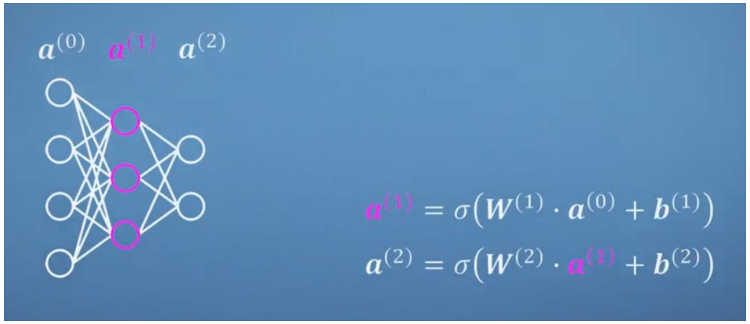
<Mathematics for Machine Learning>

- Another layer (hidden layer) can be inserted between input layer and output layer.
- It works well if there is no direct (or strong) correlation between input layer and output layer.

Neural network

- Machine Learning : No Hidden layer
- Neural Network : One hidden layer
- Deep Neural network: More than two hidden layers





<Mathematics for Machine Learning>

Neural Network

- Input layer : 4 neurons
- Hidden layer : 3 neurons

$$a_1^{[1]} = \sigma(z_1^{[1]})$$

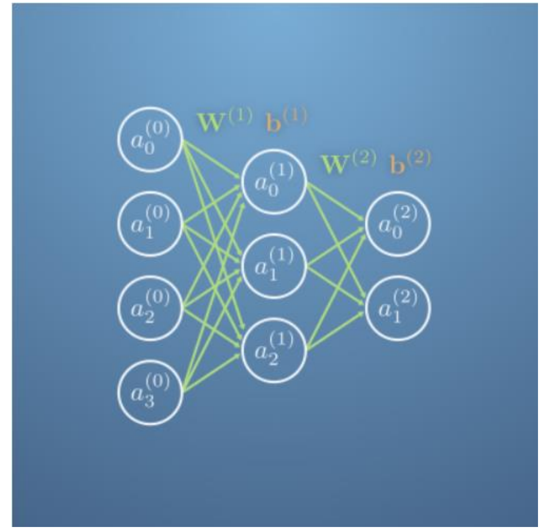
$$a_2^{[1]} = \sigma(z_2^{[1]})$$

$$a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_1^{[1]} = b_1^{[1]} + w_{11}^{[1]} \cdot a_1^{[0]} + w_{21}^{[1]} \cdot a_2^{[0]} + w_{31}^{[1]} \cdot a_3^{[0]} + w_{41}^{[1]} \cdot a_4^{[0]}$$

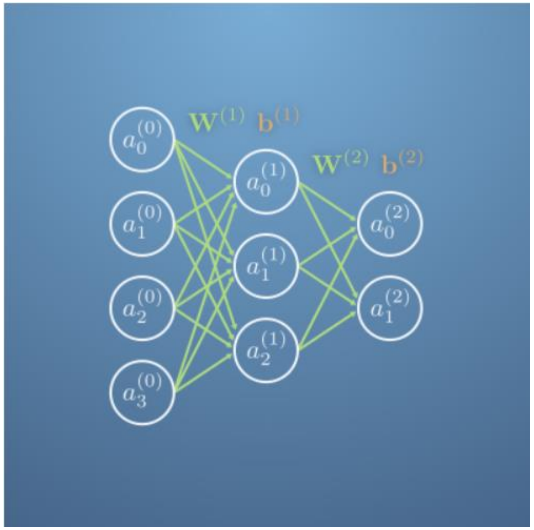
$$z_2^{[1]} = b_2^{[1]} + w_{12}^{[1]} \cdot a_1^{[0]} + w_{22}^{[1]} \cdot a_2^{[0]} + w_{32}^{[1]} \cdot a_3^{[0]} + w_{42}^{[1]} \cdot a_4^{[0]}$$

$$z_3^{[1]} = b_3^{[1]} + w_{13}^{[1]} \cdot a_1^{[0]} + w_{23}^{[1]} \cdot a_2^{[0]} + w_{33}^{[1]} \cdot a_3^{[0]} + w_{43}^{[1]} \cdot a_4^{[0]}$$



<Mathematics for Machine Learning>

Neural Network



<Mathematics for Machine Learning>

- Hidden layer : 3 neurons
- output layer : 2 neurons

$$a_1^{[2]} = \sigma(z_1^{[2]})$$

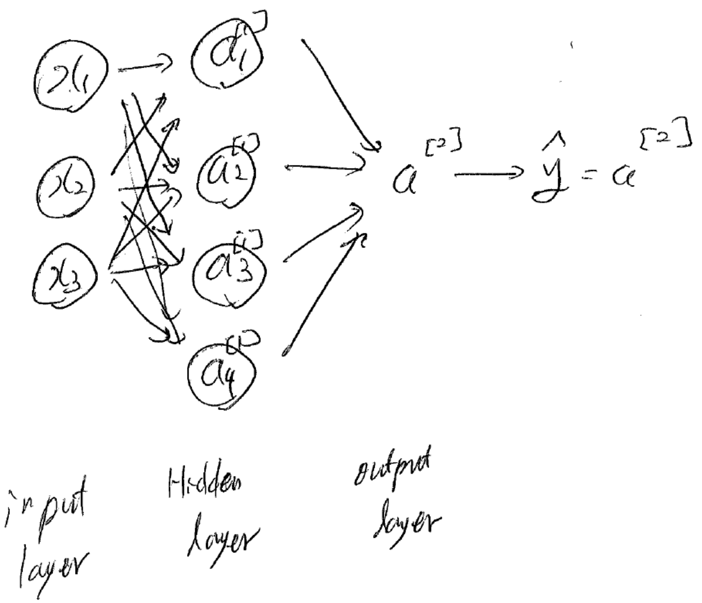
$$a_2^{[2]} = \sigma(z_2^{[2]})$$

$$z_1^{[2]} = b_1^{[2]} + w_{11}^{[2]} \cdot a_1^{[1]} + w_{21}^{[2]} \cdot a_2^{[1]} + w_{31}^{[2]} \cdot a_3^{[1]}$$

$$z_2^{[2]} = b_2^{[2]} + w_{12}^{[2]} \cdot a_1^{[1]} + w_{22}^{[2]} \cdot a_2^{[1]} + w_{32}^{[2]} \cdot a_3^{[1]}$$

Neural Network

- Input layer : 3 neurons
- Hidden layer : 4 neurons
- output layer : 1 neurons



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$

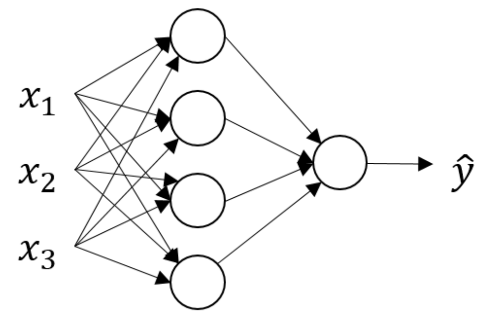
$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$

$$a_2^{[1]} = \sigma(z_2^{[1]})$$

Neural network

- Input layer : 3 neurons
- Hidden layer : 4 neurons
- output layer : 1 neurons



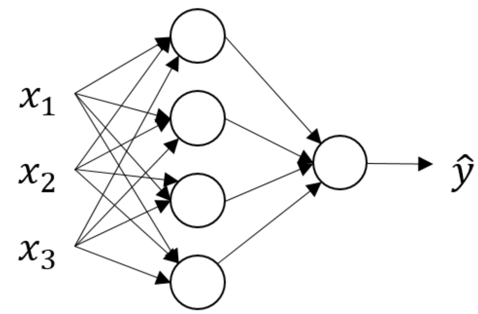
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

Neural Network



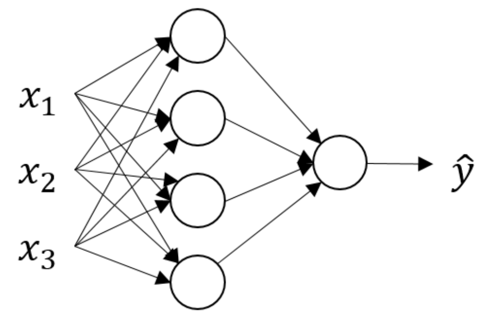
- Input layer : 3 neurons
- Hidden layer : 4 neurons
- output layer : 1 neurons

For Hidden layer (matrix form)

$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

$$= \begin{bmatrix} w_1^{[1]T}x + b_1^{[1]} \\ w_2^{[1]T}x + b_2^{[1]} \\ w_3^{[1]T}x + b_3^{[1]} \\ w_4^{[1]T}x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

Neural Network



- Input layer : 3 neurons
- Hidden layer : 4 neurons
- output layer : 1 neurons

$$\underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}}\underset{(3,1)}{x} + \underset{(4,1)}{b^{[1]}}$$

$$= W^{[1]} a^{[0]} + b^{[1]}$$

$$\underset{(4,1)}{a^{[1]}} = \sigma(\underset{(4,1)}{z^{[1]}})$$

Hidden layer

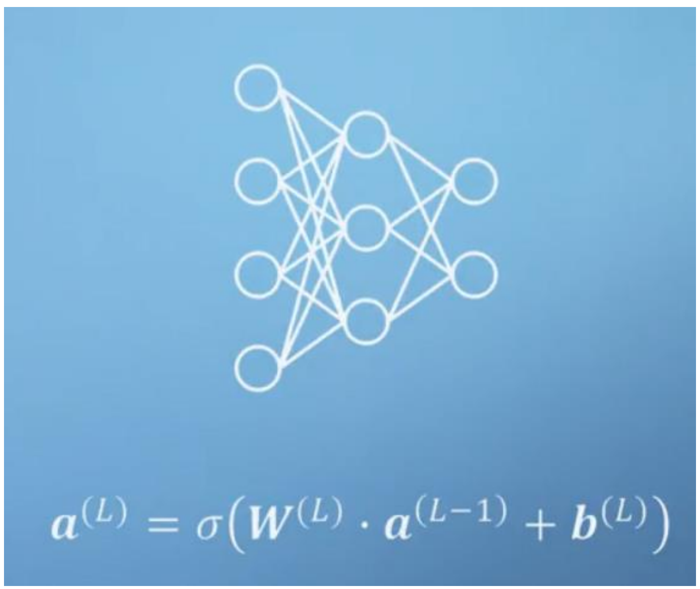
$$\underset{(1,1)}{z^{[2]}} = \underset{(1,4)}{W^{[2]}} \cdot \underset{(4,1)}{a^{[1]}} + \underset{(1,1)}{b^{[2]}}$$

$$\underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

Output layer

Neural Network

Summary



<Mathematics for Machine Learning>

- Input layer
- Hidden layer
- Output layer

- Machine Learning : No Hidden layer
- Neural Network : One hidden layer
- Deep Neural network: More than two hidden layers

A brief history of backpropagation

The backpropagation algorithm for learning multiple layers of features was invented several times in the 70's and 80's:

– Bryson & Ho (1969) linear
– Werbos (1974)
– Rumelhart et. al. in 1981
– Parker (1985)
– LeCun (1985)
– Rumelhart et. al. (1985)

Backpropagation clearly had great promise for learning multiple layers of non-linear feature detectors.

• But by the late 1990's most serious researchers in machine learning had given up on it.

– It was still widely used in psychological models and in practical applications such as credit card fraud detection.

<Neural Networks for Machine Learning, Geoffrey Hinton>

Why backpropagation failed

The popular explanation of why backpropagation failed in the 90's:
– It could not make good use of multiple hidden layers. (except in convolutional nets)
– It did not work well in recurrent networks or deep auto-encoders.
– Support Vector Machines worked better, required less expertise, produced repeatable results, and had much fancier theory.

The real reasons it failed:
– Computers were thousands of times too slow.
– Labeled datasets were hundreds of times too small.
– Deep networks were too small and not initialized sensibly.
• These issues prevented it from being successful for tasks where it would eventually be a big win.

<Neural Networks for Machine Learning, Geoffrey Hinton>

Backpropagation

Backpropagation is a way of computing gradients of expressions through recursive application of chain rule.

Differential equation (Derivative)

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

indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point

$$f(x+h) = f(x) + h\frac{df(x)}{dx}$$

when h is very small,
then the function is well approximated by a straight line,
and the derivative is its slope

Partial derivative & Gradient

$$f(x, y) = xy$$

$$\frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

Chain rule

$f(x, y, z) = (x + y)z$

$q = x + y$

$f(x, y, z) = q*z$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}}$$

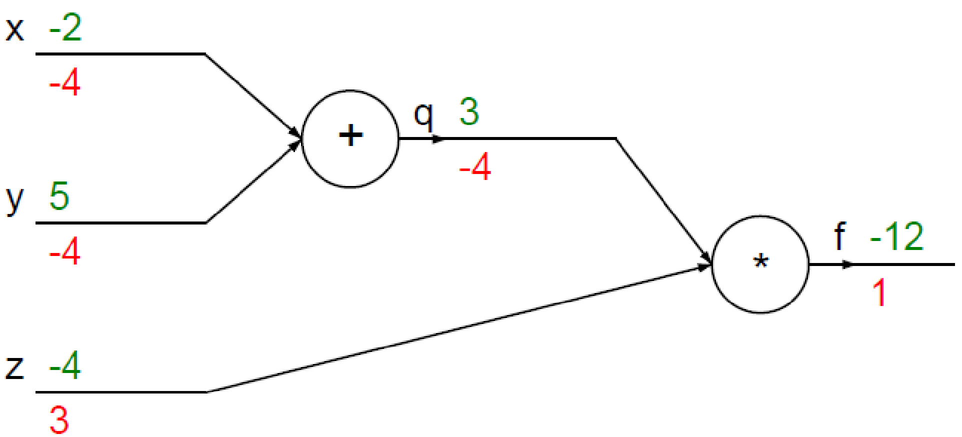
Forward pass

x=-2, y=5, z=-4

$f(x, y, z) = (x + y)z$

$q = x + y$

$f(x, y, z) = q*z$



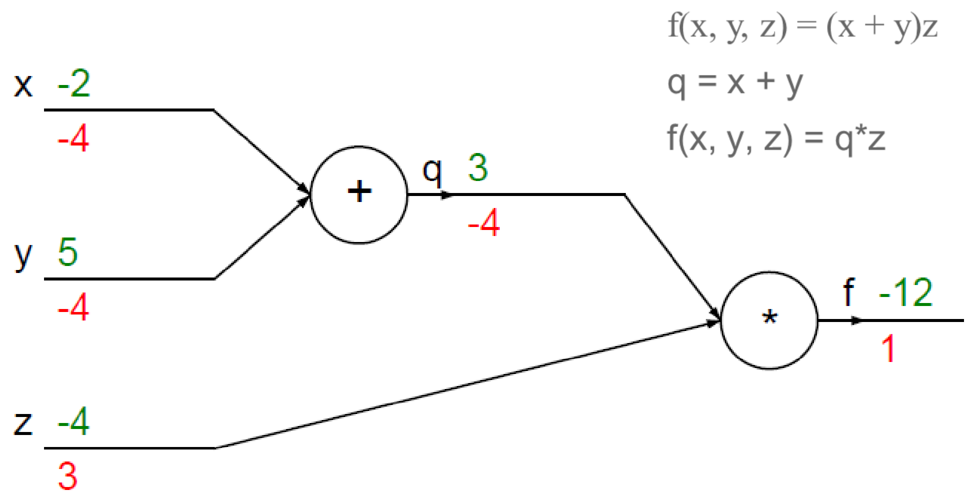
Backward pass

x=-2, y=5, z=-4

$$\frac{df}{dq} = z = -4$$

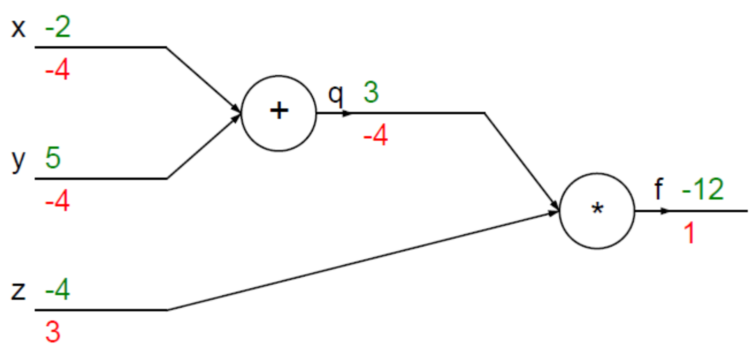
$$\frac{df}{dz} = q = 3$$

$$\frac{dq}{dx} = 1 \qquad \frac{dq}{dy} = 1$$



Backpropagation

The **forward pass** computes values from inputs to output.
The **backward pass** then performs **backpropagation** which starts at the end and recursively applies the chain rule to compute the gradients all the way to the inputs.
The gradients can be thought of as flowing backwards.



Backpropagation

Forward pass

$$f(x) = 5x$$

$$x(u) = 1 - u$$

$$u(t) = t^2$$

$$f(t) = 5(1 - t^2) = 5 - 5t^2$$

$$\frac{df}{dt} = -10t$$

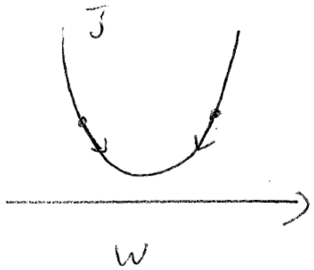
Backward pass

$$\frac{df}{dt} = \frac{df}{dx}\frac{dx}{du}\frac{du}{dt}$$

$$\frac{df}{dt} = (5)(-1)(2t)$$

$$= -10t$$

Linear regression & chain rule



$$W := W - \alpha \frac{\partial J}{\partial W}$$

$$J = \frac{1}{m} \sum (\hat{y} - y)^2$$

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W}$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{2}{m} \sum (\hat{y} - y)$$

$$\frac{\partial \hat{y}}{\partial W} = x \quad \text{since } \hat{y} = W \cdot x$$

$$\therefore \frac{\partial J}{\partial W} = \frac{2}{m} \sum (\hat{y} - y) \cdot x$$

Two neurons (Logistic regression)

$$a^{[1]} = \sigma(w \cdot a^{[0]} + b)$$

$$L = (a^{[1]} - y)^2$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial w}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial b}$$

$$z^{[1]} = w \cdot a^{[0]} + b$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial w}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial b}$$

Summary

- One-hot encoding
- Neural Network
- Back propagation