Fundamentals of Machine Learning

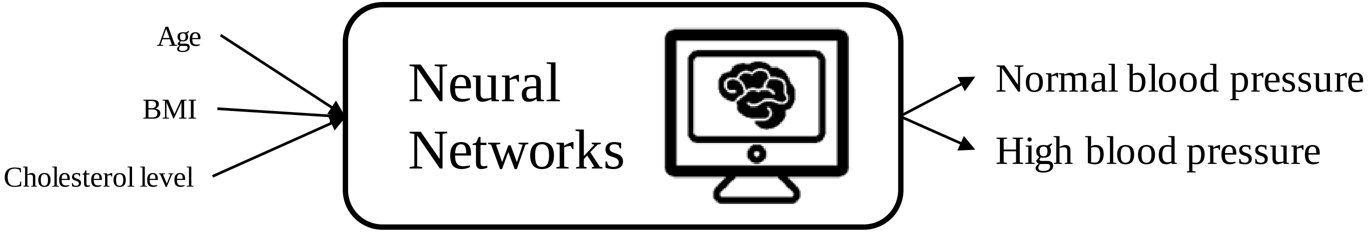
BASICS of DNNs

Amit K Roy-Chowdhury

Acknowledgments: Adapted from slides at https://probml.github.io/pml-book/teaching1.html by Prof. Saw Shier Nee

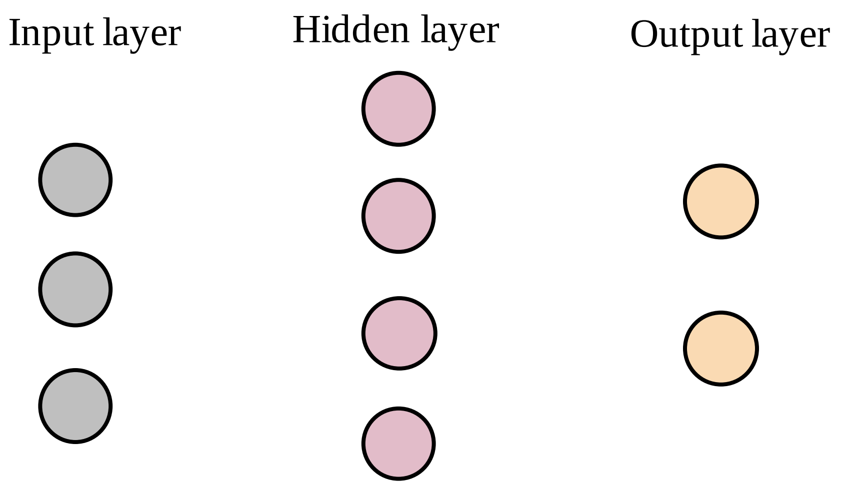
Neural Networks

- Data:
  - Input (Age, BMI, Cholesterol level)
  - Output (Normal / High Blood Pressure)
- We teach the model using the data to make accurate prediction by optimizing parameters



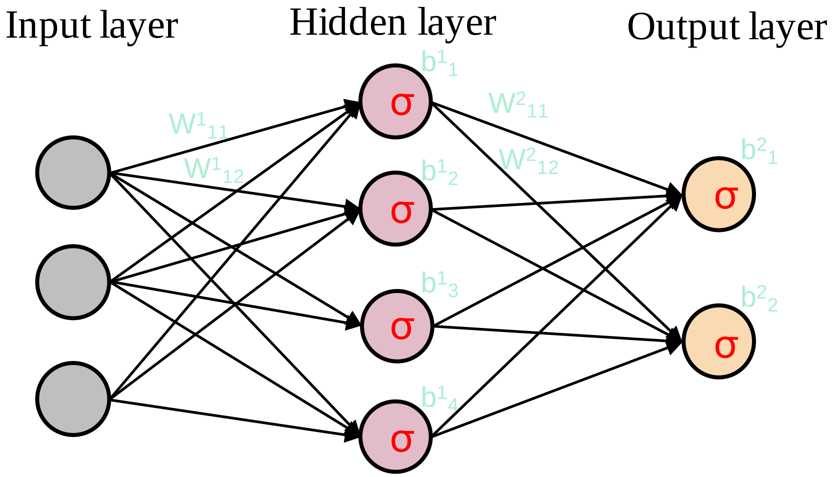
Neural Networks Basics

- Made up of <u>multiple layers of nodes</u>

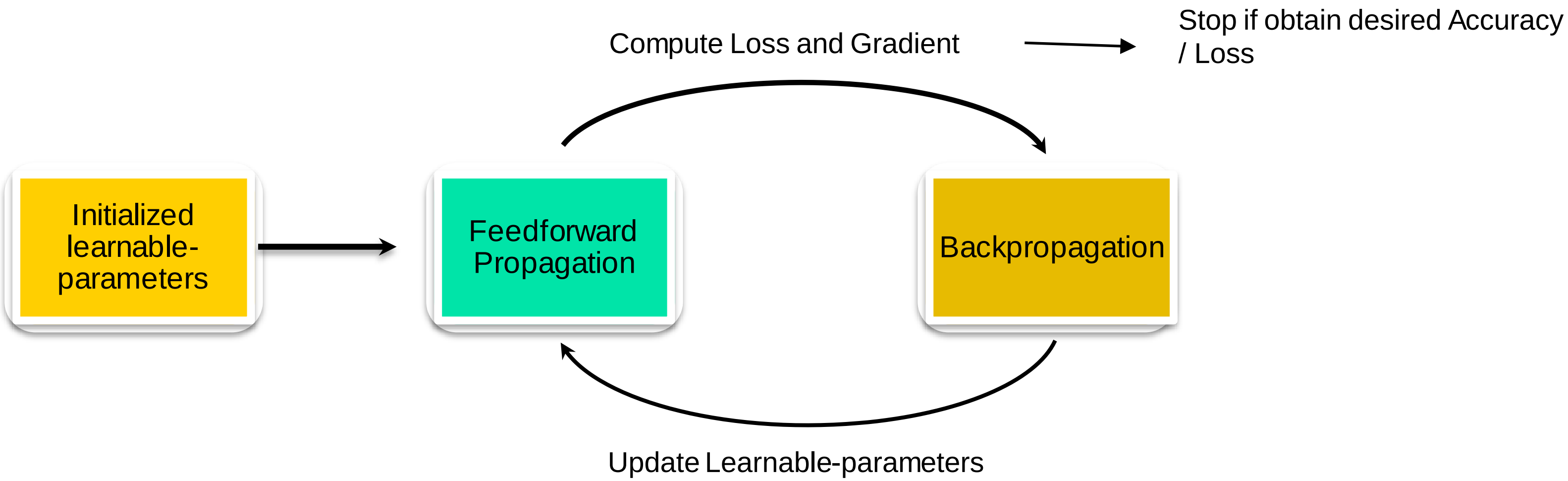


Neural Networks Basics

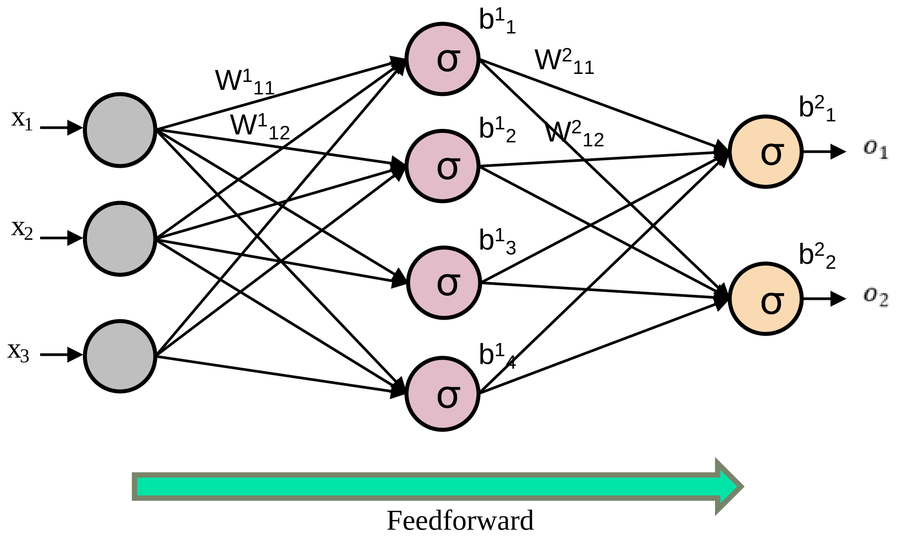
- Made up of <u>multiple layers of nodes</u>.
- Each layer makes simple decisions using different <u>weights, *w*, bias, *b*</u> <u>and</u> <u>activation function, *σ*</u>.



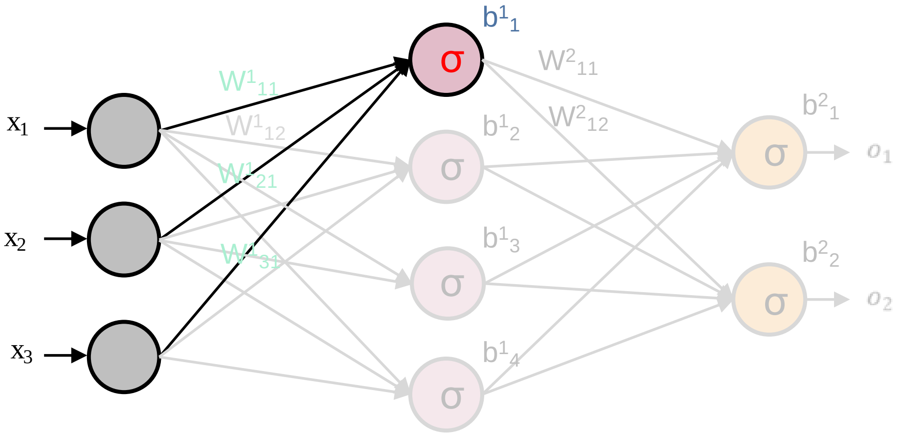
Neural Networks Training Process



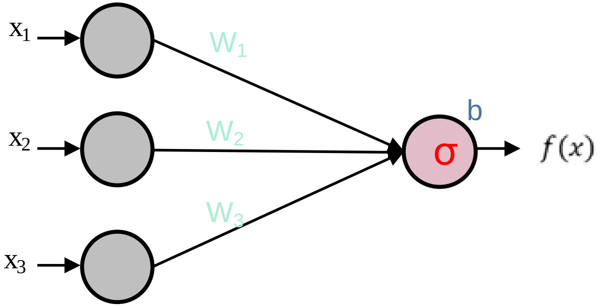
Feedforward Propagation



Feedforward Propagation



Feedforward Propagation (One Perceptron)



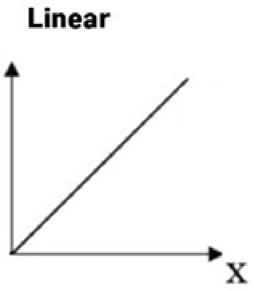
Input x Weight + Bias $$(x_1 w_1 + x_2 w_2 + x_3 w_3 + b)$$

Apply Activation Function $$\sigma(x_1 w_1 + x_2 w_2 + x_3 w_3 + b)$$

Obtain output $$f(x) = \sigma(x_1 w_1 + x_2 w_2 + x_3 w_3 + b)$$

Activation Function

- A mathematical function that determines the output of each perceptron in the neural network



$\sigma(x) = x, for\ all\ x$

$$f(x) = \sigma(x_1 w_1 + x_2 w_2 + x_3 w_3 + b)$$

$$= \sigma(20 \cdot (-3) + 23 \cdot 2 + 4 \cdot 9 + 8)$$

$$= \sigma(-60 + 46 + 36 + 8)$$

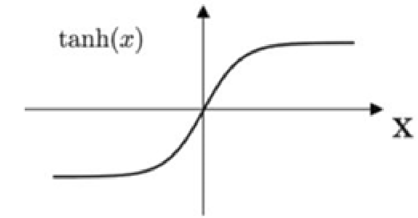
$$= \sigma(30)$$

$$= 30$$

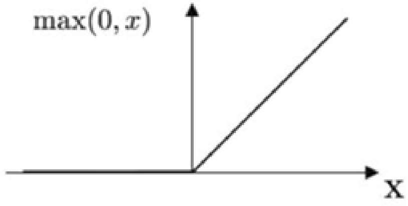
Activation Function

- Common activation functions:

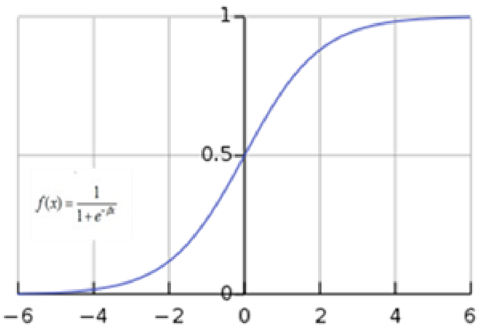
Tanh



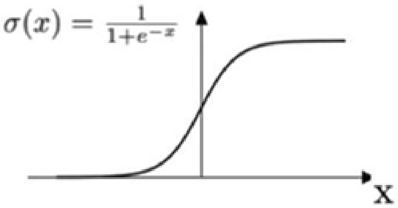
ReLU



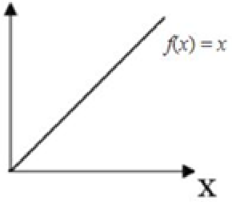
Softmax



Sigmoid



Linear

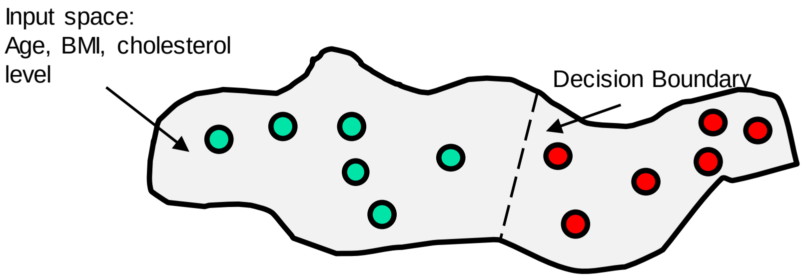


Why different activation functions?

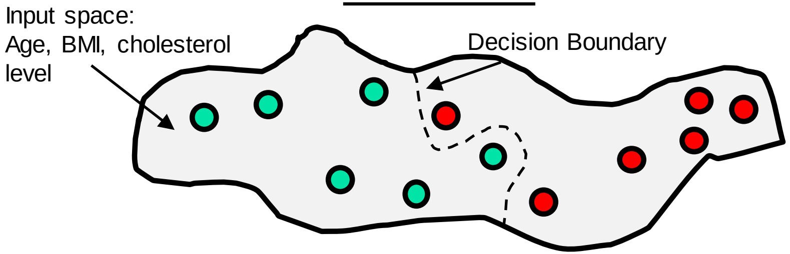
Normal Blood Pressure

High Blood Pressure

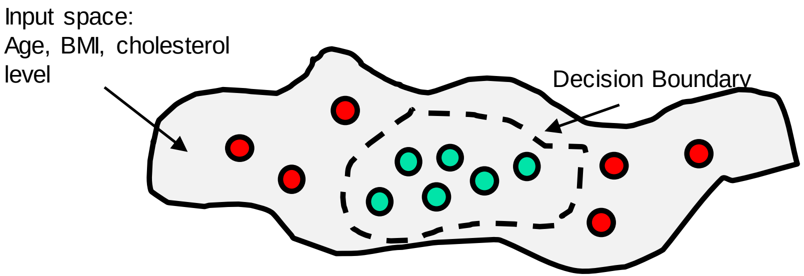
Scenario 1



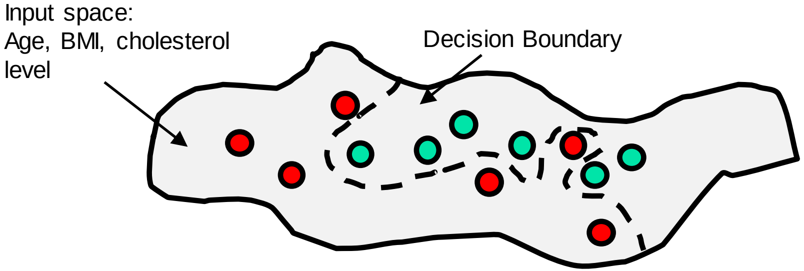
Scenario 2



Scenario 3

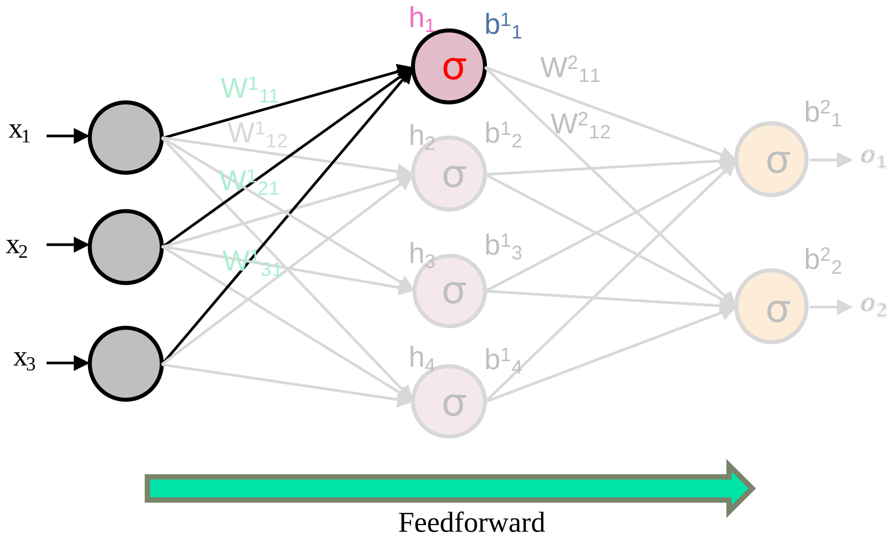


Scenario 4



Feedforward All Perceptrons

<u>Feedforward Neural Network</u>

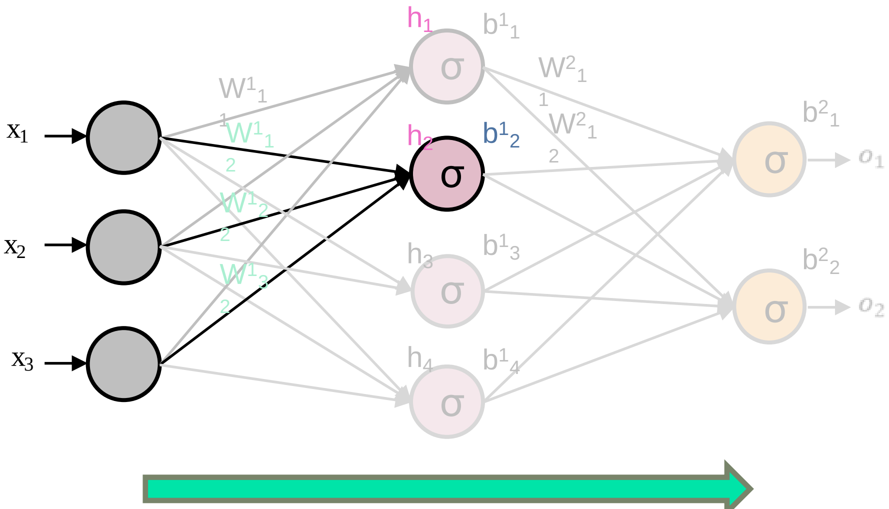


Input x Weight + Bias

$$h_1 = \sigma\overbrace{(x_1 w^1_{11} + x_2 w^1_{21} + x_3 w^1_{31} + b^1_1)}$$

Activation function

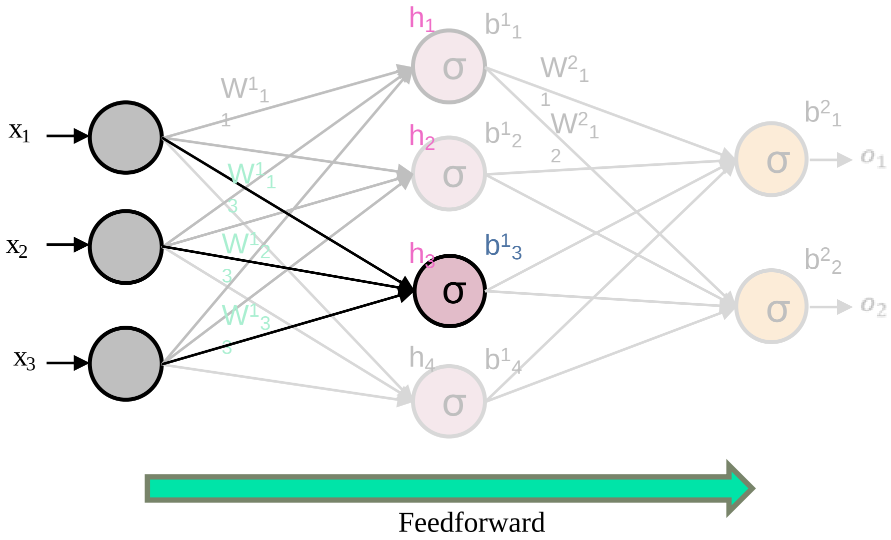
Feedforward All Perceptrons



$$h_1 = \sigma(x_1 w^1_{11} + x_2 w^1_{21} + x_3 w^1_{31} + b^1_1)$$

$$h_2 = \sigma(x_1 w^1_{12} + x_2 w^1_{22} + x_3 w^1_{32} + b^1_2)$$

Feedforward All Perceptrons



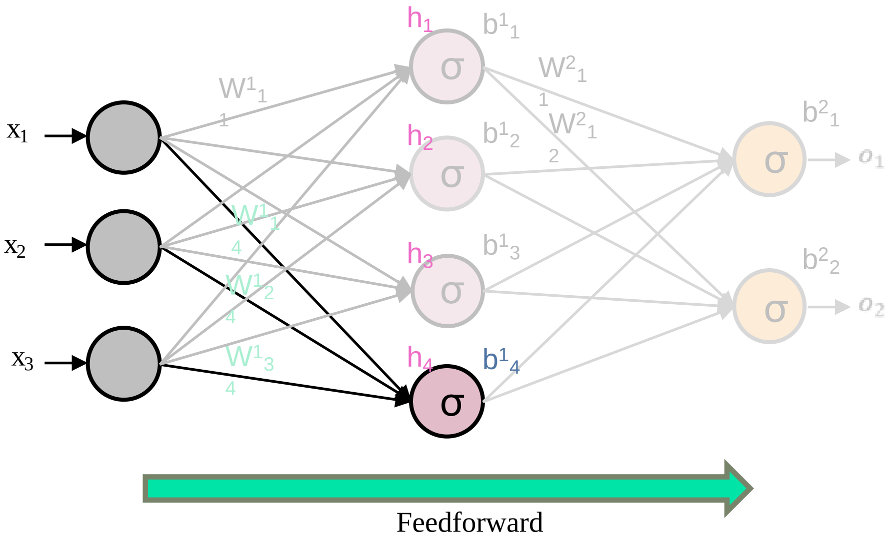
$$h_1 = \sigma(x_1 w^1_{11} + x_2 w^1_{21} + x_3 w^1_{31} + b^1_1)$$

$$h_2 = \sigma(x_1 w^1_{12} + x_2 w^1_{22} + x_3 w^1_{32} + b^1_2)$$

$$h_3 = \sigma(x_1 w^1_{13} + x_2 w^1_{23} + x_3 w^1_{33} + b^1_3)$$

UCR Bourns College of Engineering

Feedforward All Perceptrons



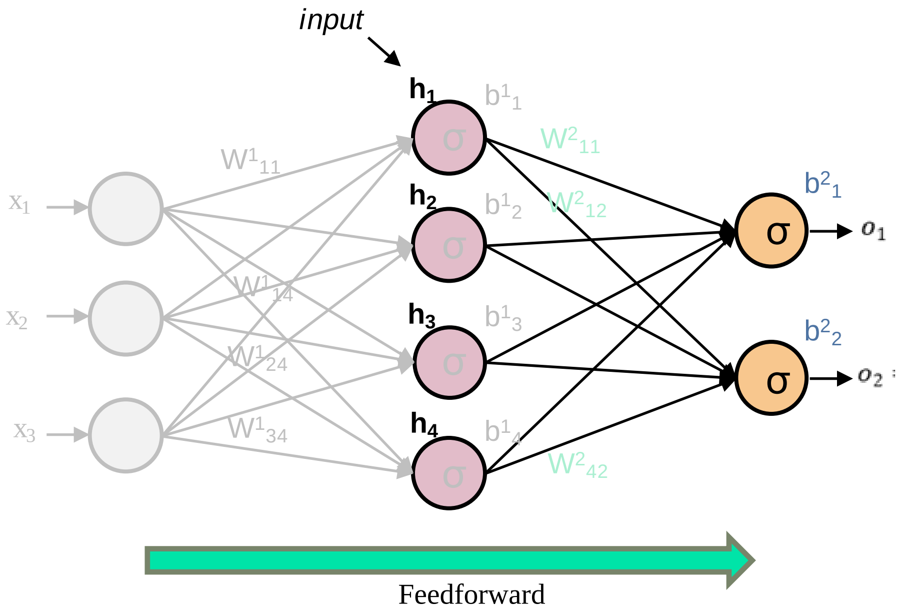
$$h_1 = \sigma(x_1 w_{11}^1 + x_2 w_{21}^1 + x_3 w_{31}^1 + b_1^1)$$

$$h_2 = \sigma(x_1 w_{12}^1 + x_2 w_{22}^1 + x_3 w_{32}^1 + b_2^1)$$

$$h_3 = \sigma(x_1 w_{13}^1 + x_2 w_{23}^1 + x_3 w_{33}^1 + b_3^1)$$

$$h_4 = \sigma(x_1 w_{14}^1 + x_2 w_{24}^1 + x_3 w_{34}^1 + b_4^1)$$

Feedforward All Perceptrons

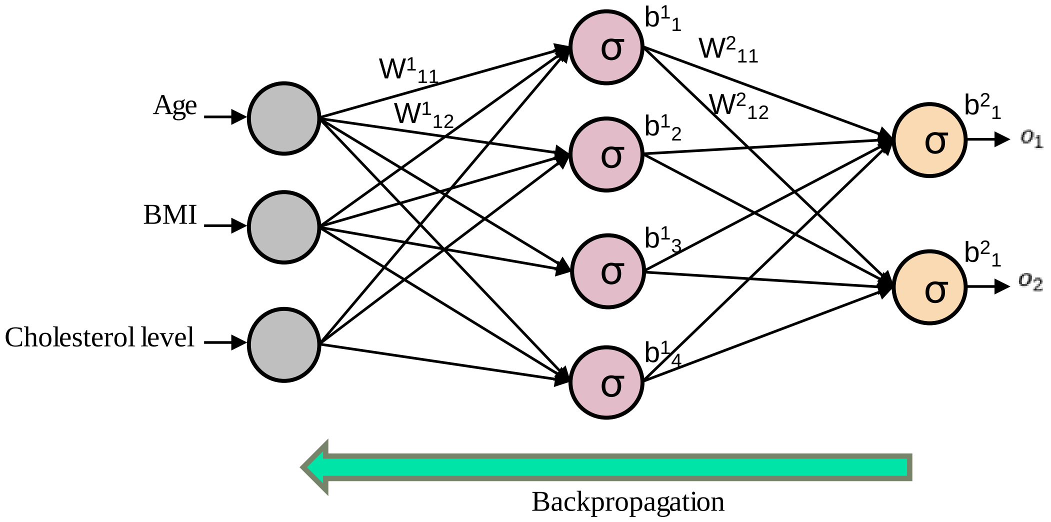


$$o_1 = \sigma(h_1 w_{11}^2 + h_2 w_{21}^2 + h_3 w_{31}^2 + h_4 w_{41}^2 + b_1^2)$$

$$o_2 = \sigma(h_1 w_{12}^2 + h_2 w_{22}^2 + h_3 w_{32}^2 + h_4 w_{42}^2 + b_2^2)$$

Backpropagations to update weight

- Compute Gradients, i.e: $\frac{\partial L}{\partial w_{ij}^{n}}, \frac{\partial L}{\partial b_{i}^{n}}$
- Backpropagate the gradient to updates the weights



Gradient Descent

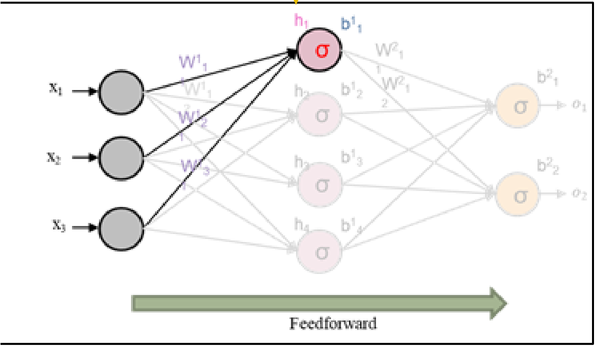
$$\theta_{t+1} = \theta_t + \rho_t d_t$$

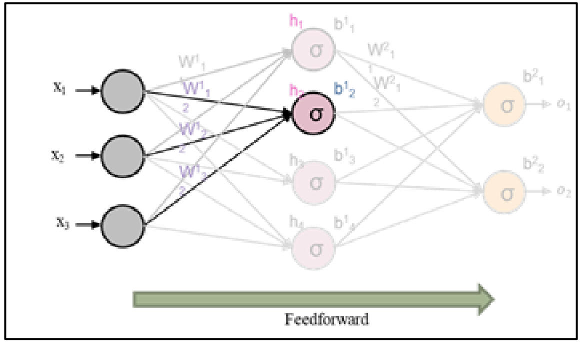
Updated parameter
(weight, bias)

Step size

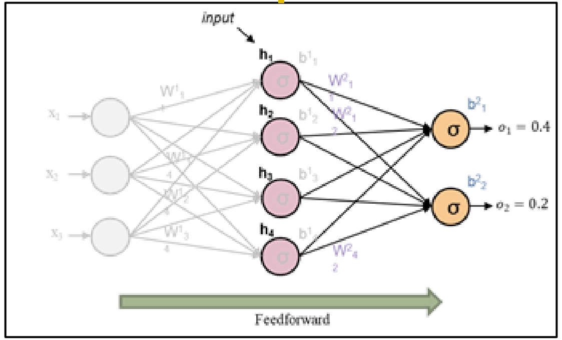
Descent direction
(Gradient: $\frac{\partial L}{\partial w_{ij}^n}$, $\frac{\partial L}{\partial b_i^n}$)

Compute Gradients,
i.e: $\frac{\partial L}{\partial w_{ij}^n}$, $\frac{\partial L}{\partial b_i^n}$

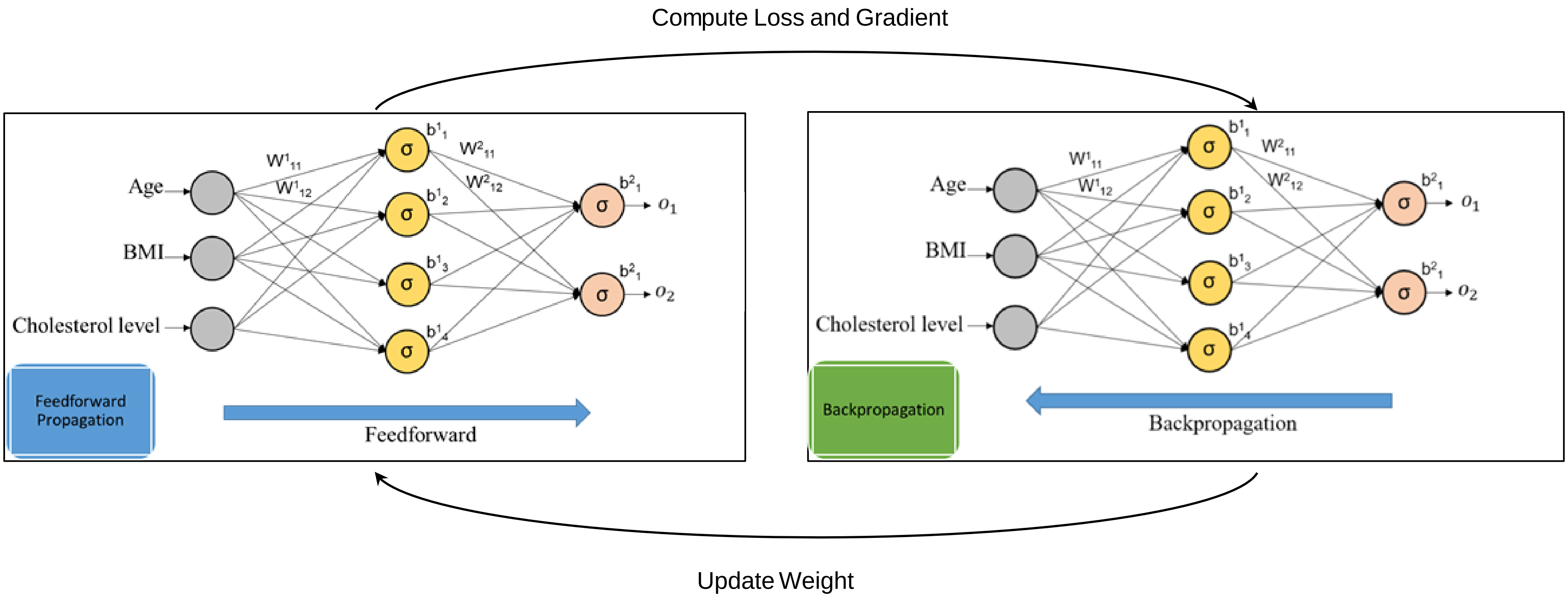




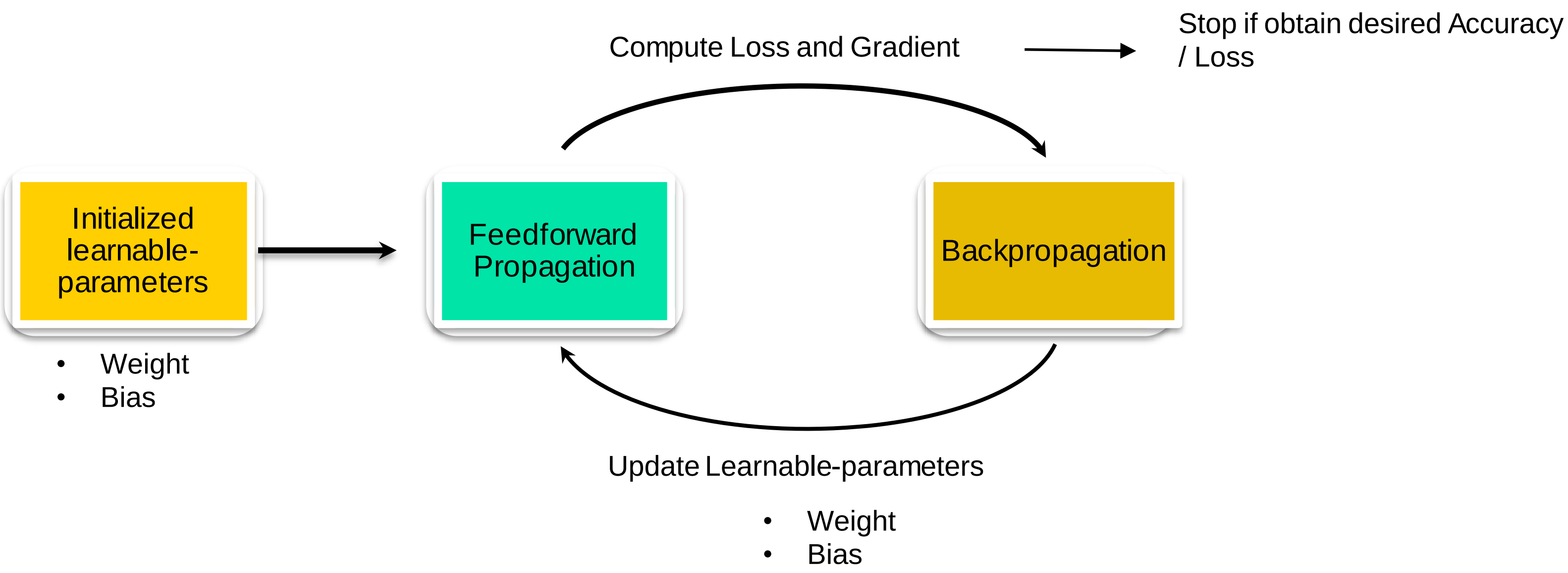
...



Training Iteratively until Loss ~ 0

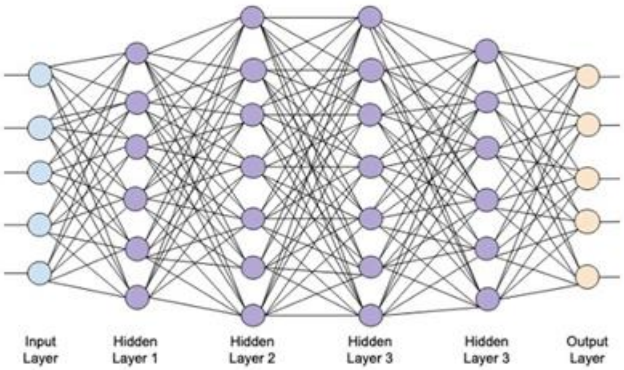


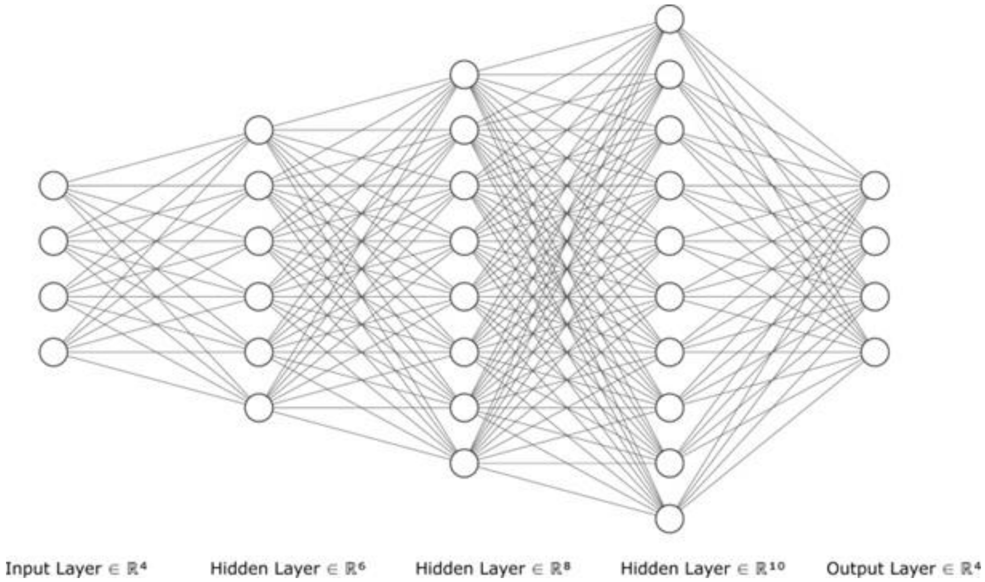
Summary Neural Networks Training Process



Larger Network

- Increase number of node
- Increase number of hidden layer





Larger Network

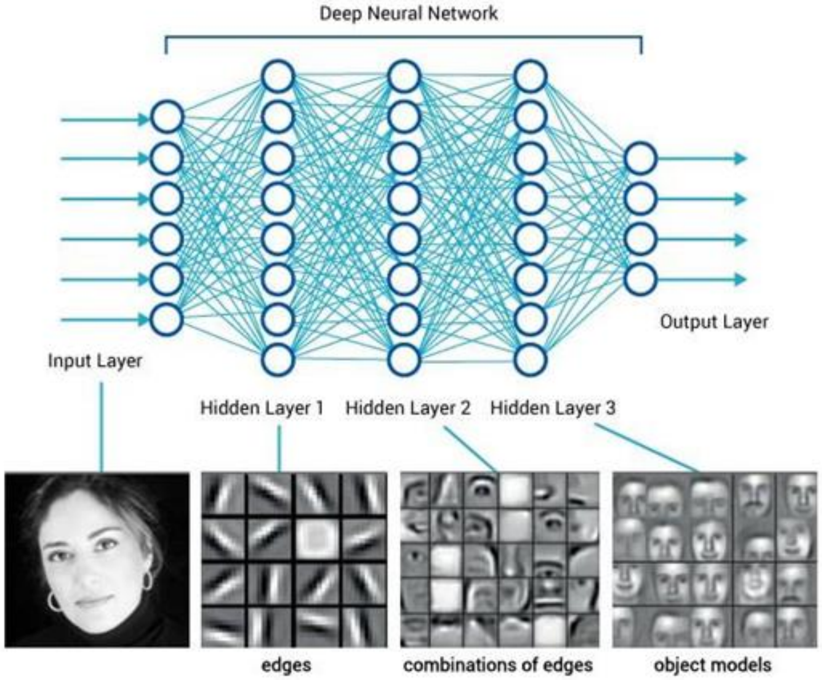
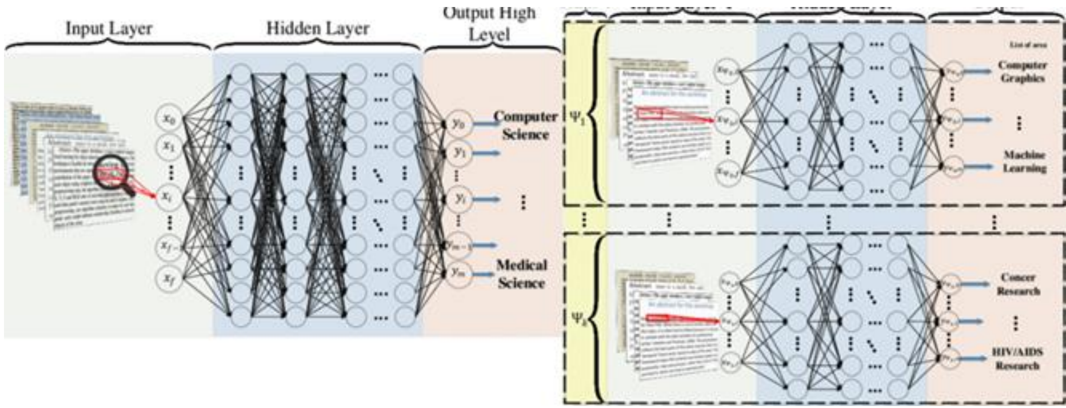


Figure from https://medium.com/diaryofawannapreneur/deep-learning-for-computer-vision-for-the-average-person-861661d8aa61



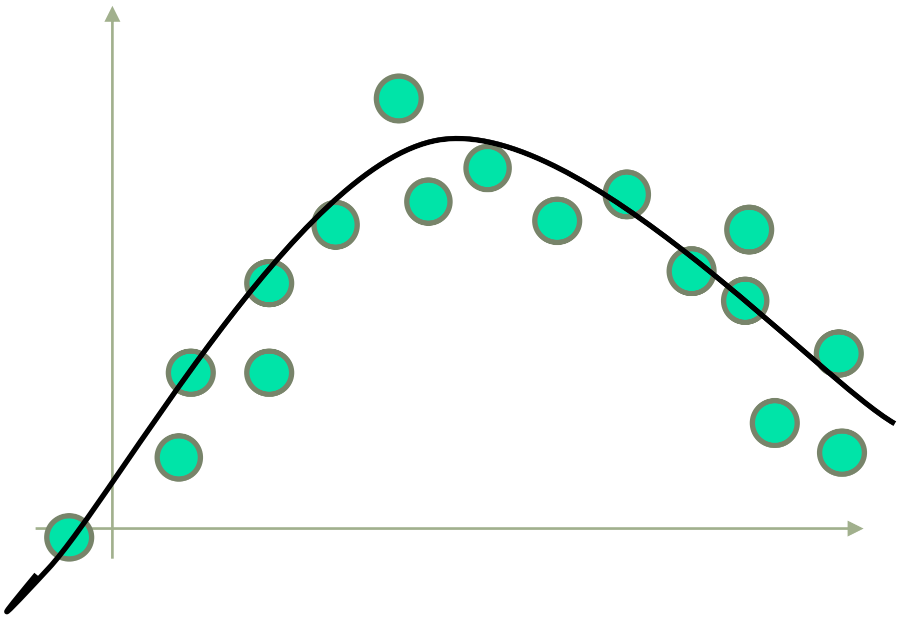
Kowsari et al. HDLTex: Hierarchical Deep Learning for Text Classification

Larger Network

- The larger the better???
- No overfitting (layman term – memorize training data only, poor performance when testing data is used)
- What the reason of overfitting??? One of the reason is we have too many parameters

Overfitting

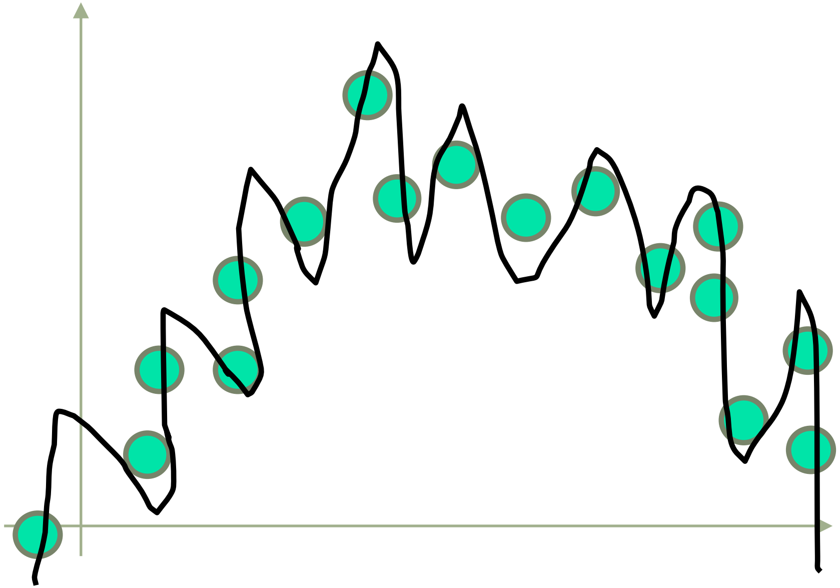
If we have the following graph



- We know it should be a quadratic graph
- Let's assume the best fit graph would be $y = -3x^2 + x + 0.5$

Overfitting

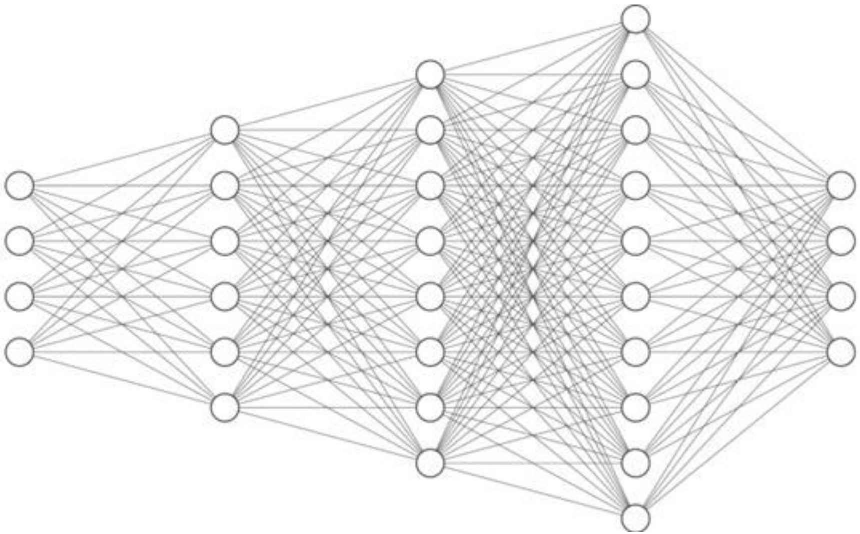
If we have the following graph



- We know it should be a quadratic graph
- Let's assume the best fit graph would be $y = -3x^2 + x + 0.5$
- If we fit with a higher order arbitrary polynomial function, $y = 0.4x^8 + 1.9x^7 - 1.4x^6 + \cdots + 1.9$ → overfit
- Have too many parameters
  - Quadratic – 3 parameters, m1, m2, b
  - 8th order – 9 parameters, m1, m2, …, m8, b

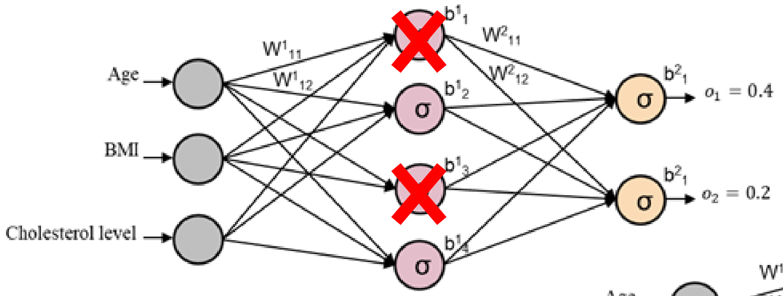
Overfitting

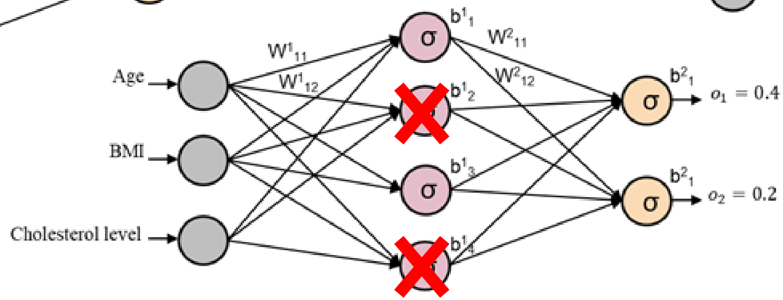
- Reason: Have too many parameters
- How to solve???
- Reduce the model size number of node / hidden layer

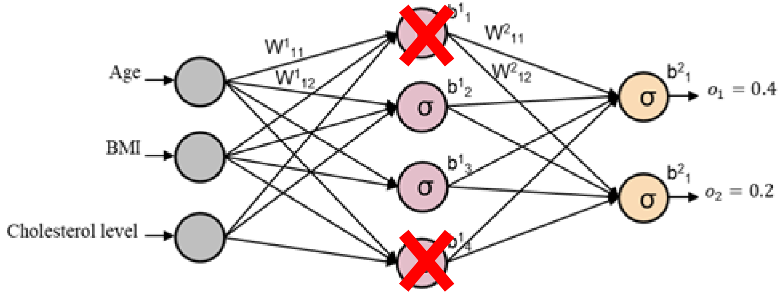


Overfitting

- Another way to solve: Use the idea of ensemble model
- Ensemble model - train multiple model then combine all of them.







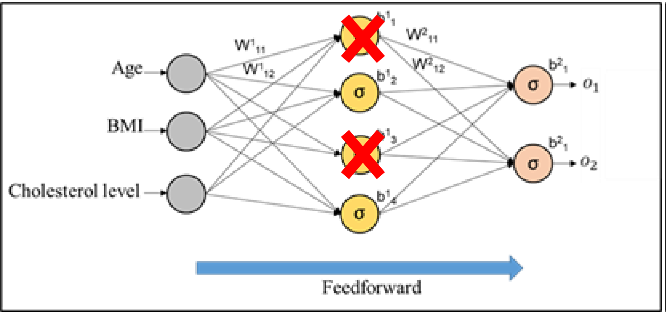
High computational cost

UCR | Bourns College of Engineering

Dropout

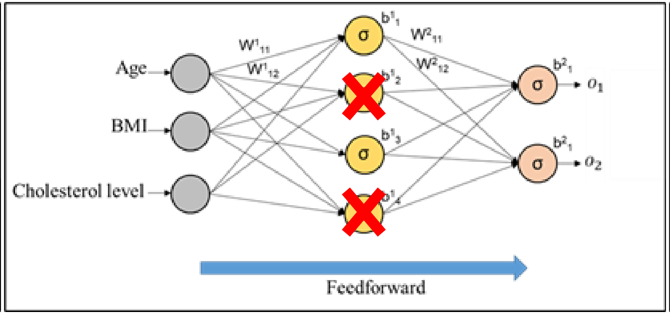
- Randomly drop the node

1st iteration (Model1)



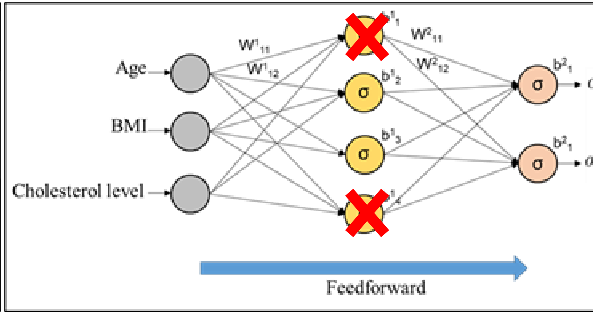
1. Forward
2. Compute gradient
3. Update weight

2nd iteration (Model2)



1. Forward
2. Compute gradient
3. Update weight

3rd iteration (Model3)



1. Forward
2. Compute gradient
3. Update weight

Idea: Train different model at every iteration.