Neural Networks - 2

Parag Mantri, PhD

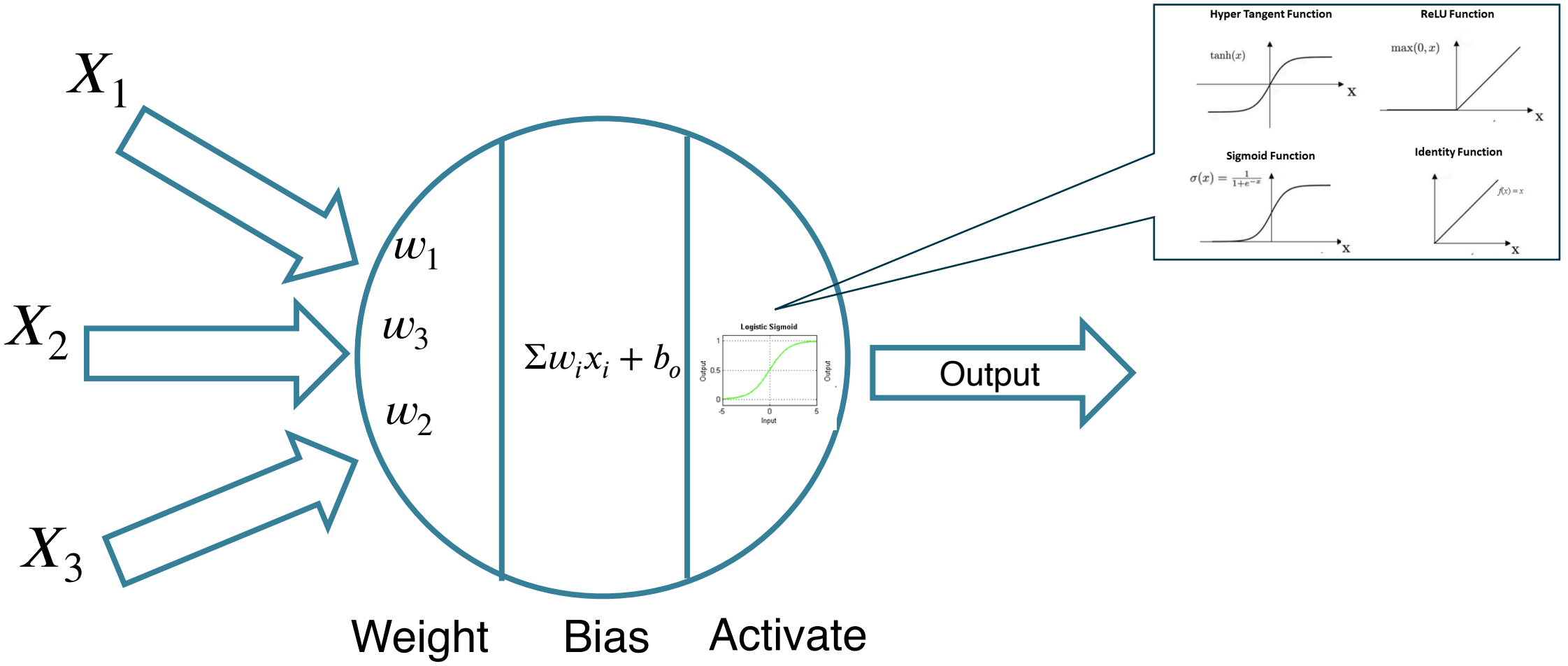
- Neural Networks Review
- Deep Learning
- Challenges with neural networks
  - Vanishing/Exploding gradients
  - Overfitting
  - Hyperparameter tuning
- Regularization and Normalization in NN
- Autoencoders
- Entity Embeddings
- Quiz

Learning Outcomes

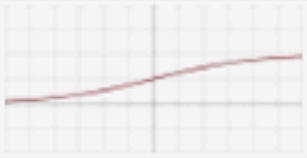
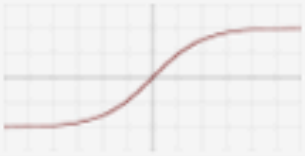
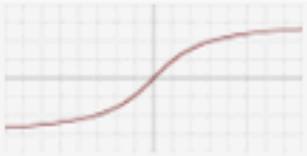
1	Relate Single Neuron with Neural Network and with Deep Learning
2	Understand DL hidden layers from feature hierarchies such as edges, corners and other patterns on image data
3	Be able explain the challenges of DL such as overfitting, unable to learn features deep through the layers
4	Be able to explain how the network learns to reconstruct the input data from the reduced dimensions at the code layer, the auto-encoder.
5	Be able to explain how autoencoders reduce the need for large data by reducing the feature size
6	Autoencoders for anomaly detection and non-linear PCA
7	Handle Overfitting in Deep Learning by regularization method such as Drop out, Batch Normalization, ADAM Learning rate
8	Be able to deal with categorical data using categorical embeddings and implement them
9	Implement dropout layer, dense layer, the input and output layers

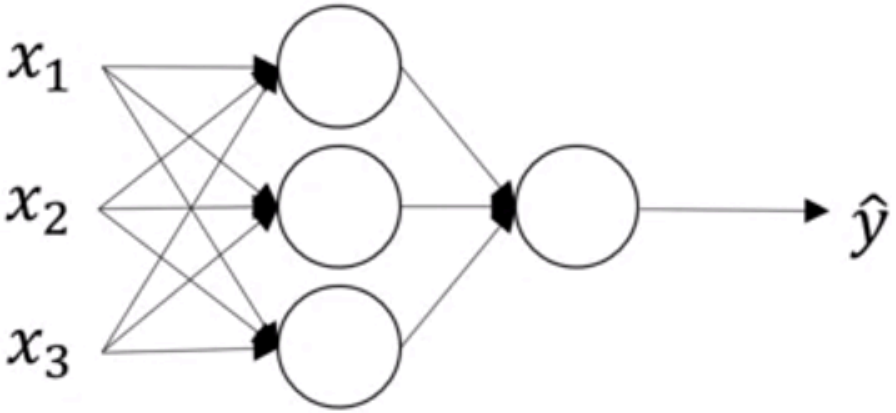
Review

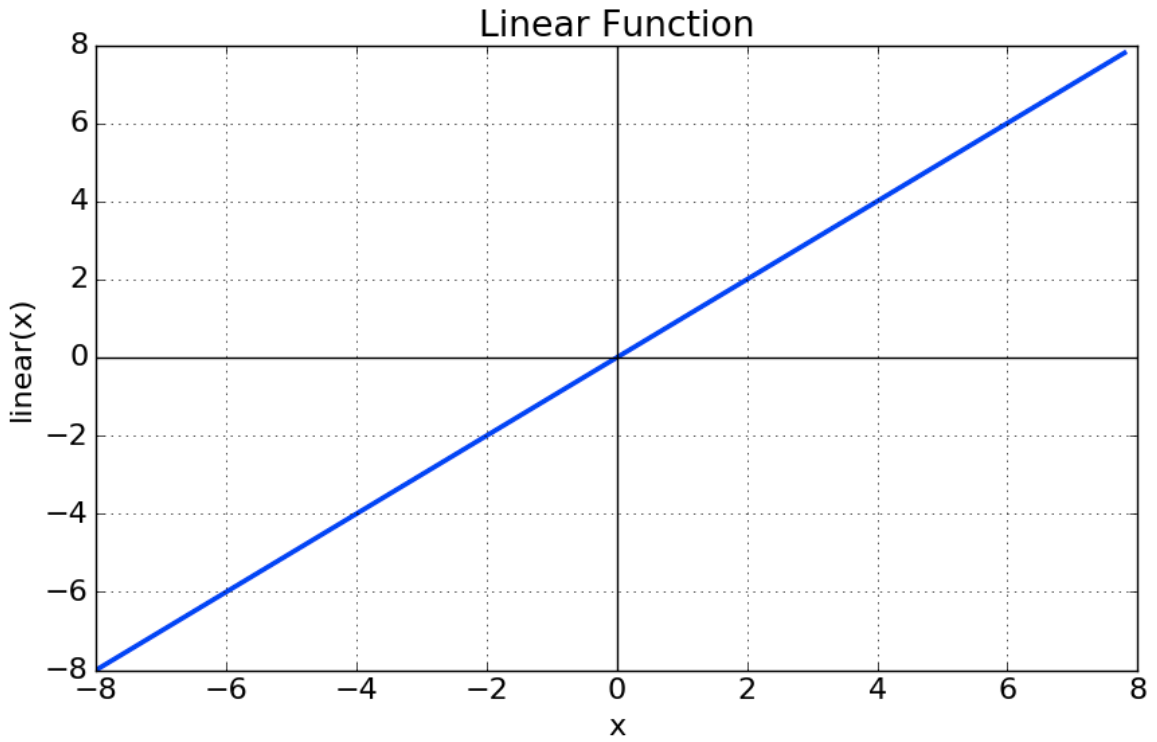
Neural Networks Lesson 1


X_1
X_2
X_3
w_1
w_3
w_2
$\Sigma w_i x_i + b_o$
Logistic Sigmoid
Output
Input
Weight
Bias
Activate
Hyper Tangent Function
$\tanh(x)$
ReLU Function
$\max(0, x)$
Sigmoid Function
$\sigma(x) = \frac{1}{1+e^{-x}}$
Identity Function
$f(x) = x$
x

Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$





Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

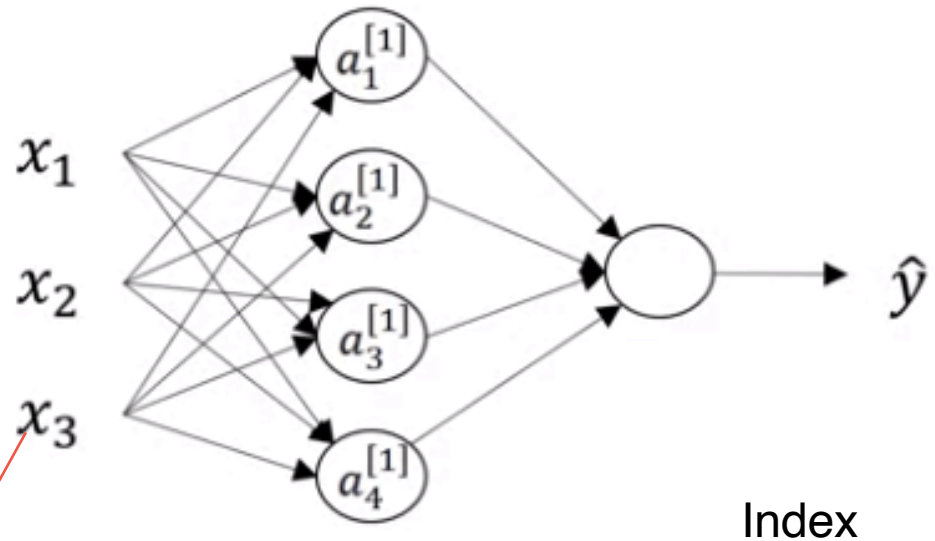
$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]})$$

Activation Function

$$W^{[2]}W^{[1]}X + W^{[2]}b^{[1]} + b^{[2]}$$



a[0]

Given input x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

Size 4 X 1

Size 4 X 1

Size 4 X 3

Size 3 X 1

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

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

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

$$W^{[1]} = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \\ W_{41} & W_{42} & W_{43} \end{bmatrix}$$

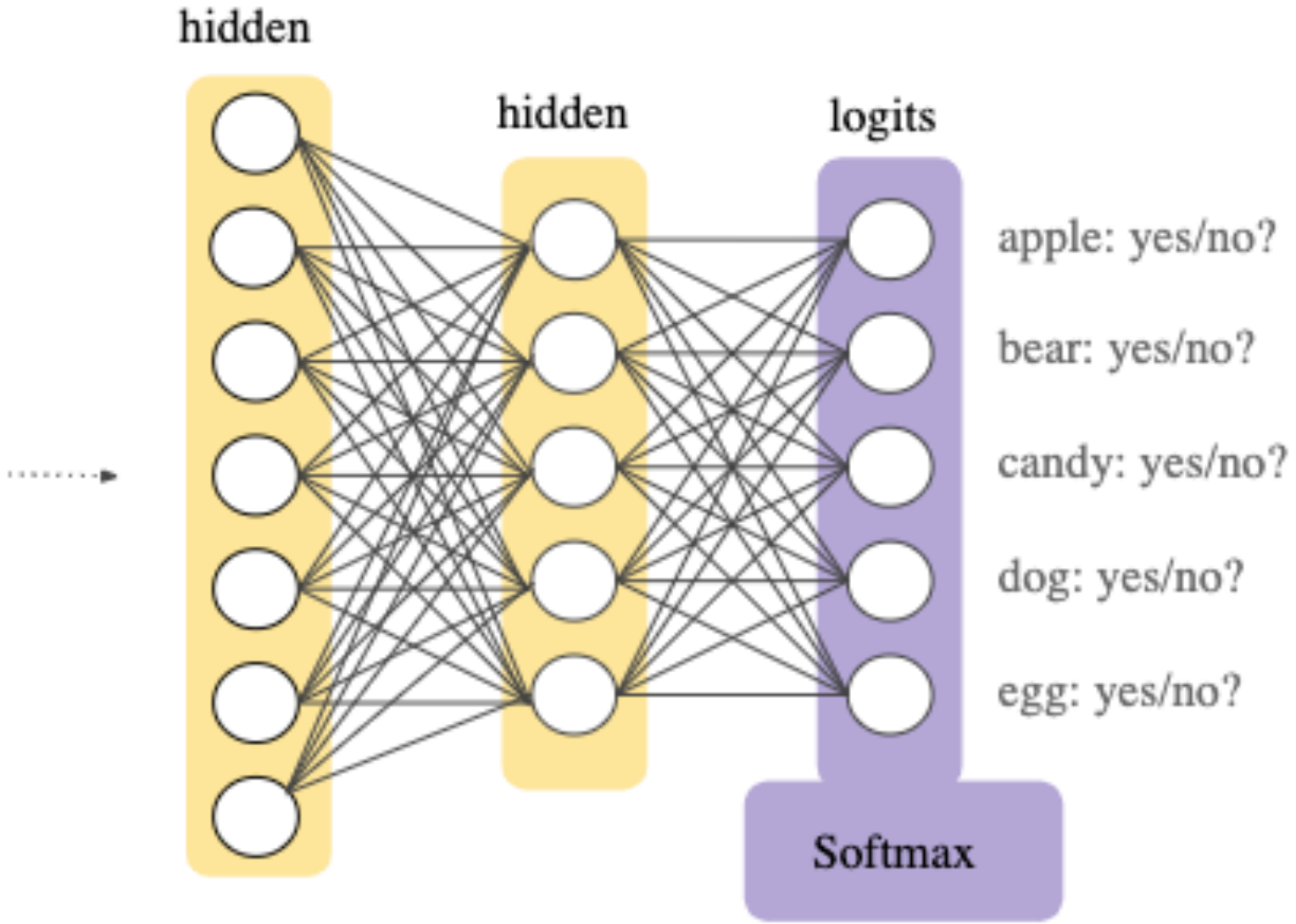
Index representing neuron in layer "From"

Index representing neuron in layer "To"

$$Z_1^{(2)}$$

Layer Number

Neuron Number


hidden
hidden
logits
apple: yes/no?
bear: yes/no?
candy: yes/no?
dog: yes/no?
egg: yes/no?
Softmax

Gradient Descent and Back Propagation

- $$w = w - \alpha \frac{dJ}{dw}$$

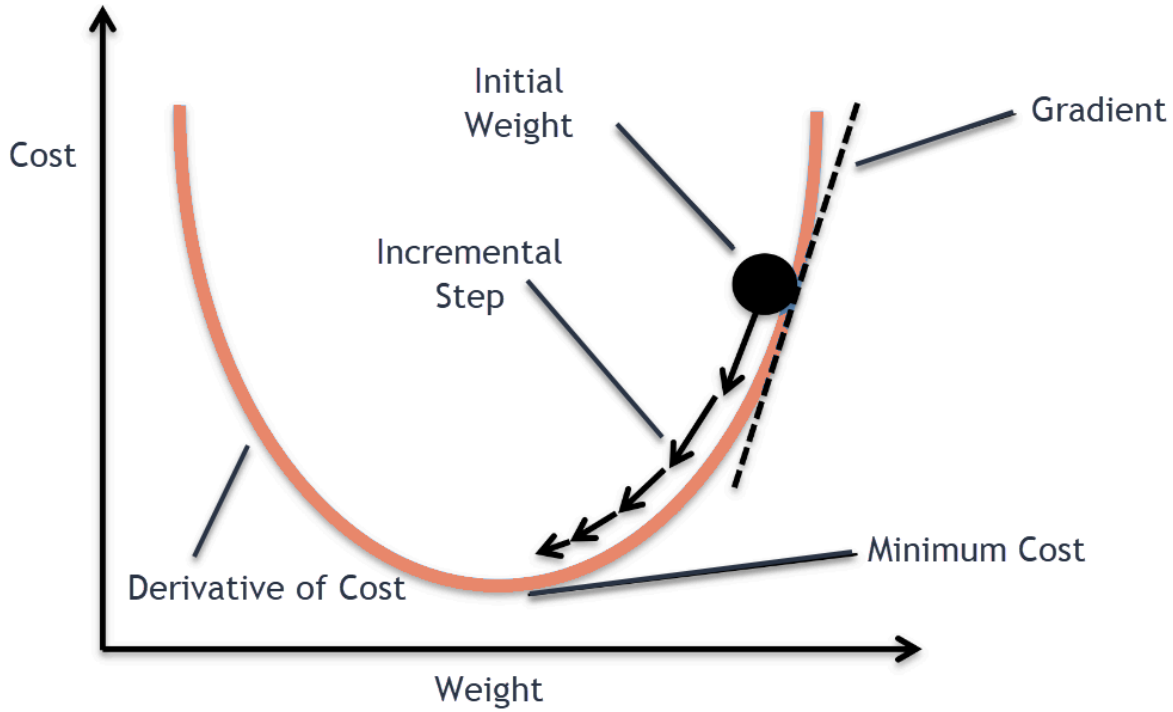


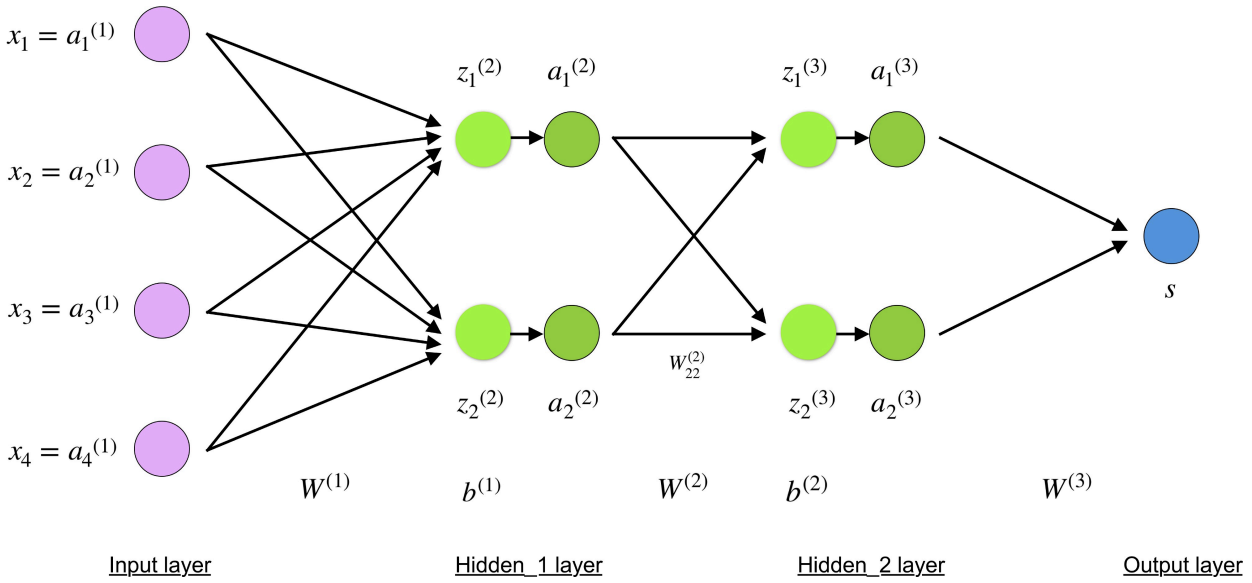
Image Ref: Stochastic Vs Batch Gradient Descent, Kapil Divakar

- $l = 2$

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Equations for z² and a²



- $l = 3$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Equations for z³ and a³

Activation function of Previous Layer is input to next layer

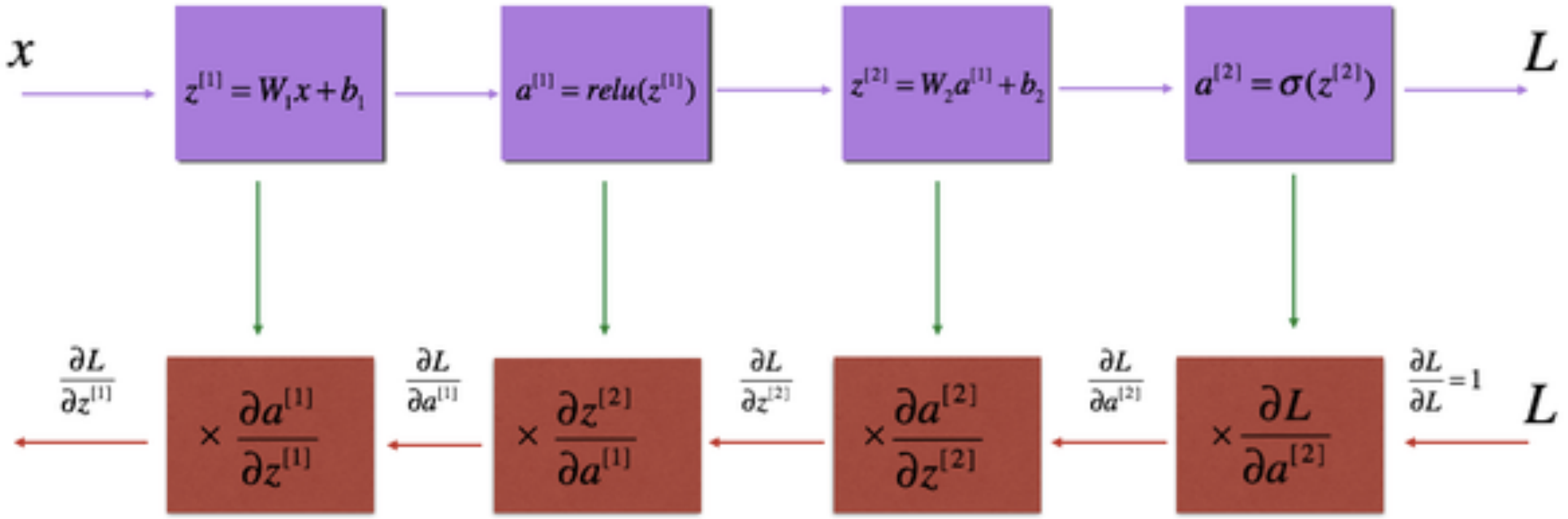
Ref: Understanding Back Propagation Algorithm, Simeon Kostadinov

$$w = w - \alpha \frac{dJ}{dw}$$

$$b = b - \alpha \frac{dJ}{db}$$

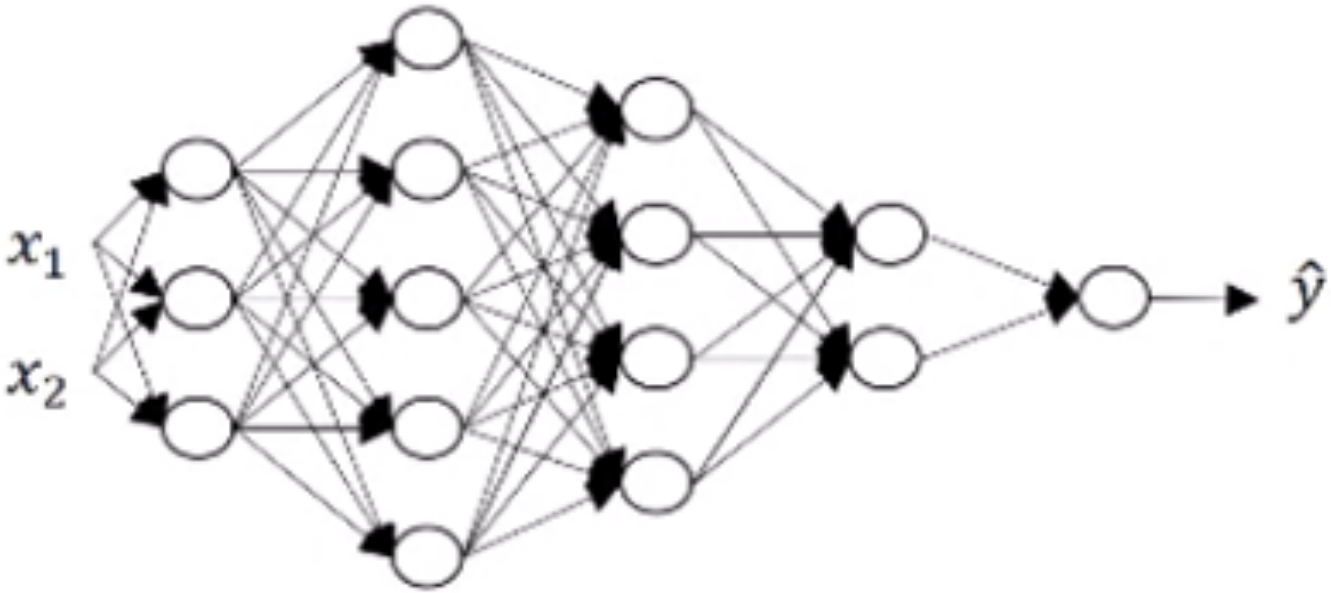
The error signal of a neuron is composed of two components:

- The weighted sum of the error signals in next layer's connection
- The derivative of this neuron's activation function.



- **Epoch:** One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.
- **Batch Gradient Descent**. Batch Size = Size of Training Set
- **Stochastic Gradient Descent**. Batch Size = 1
- **Mini-Batch Gradient Descent**. 1 < Batch Size < Size of Training Set

Why we cannot have linear activation function in Neural Networks?



What is the shape of weight matrix in 3rd hidden layer

- Between learning rate, weight and bias, which parameter is learnt and which are tuned?

- If you have implemented deep learning algorithm with 4 epochs, how many times it has run back propagation?

- These equations appear in forward propagation or backward propagation?

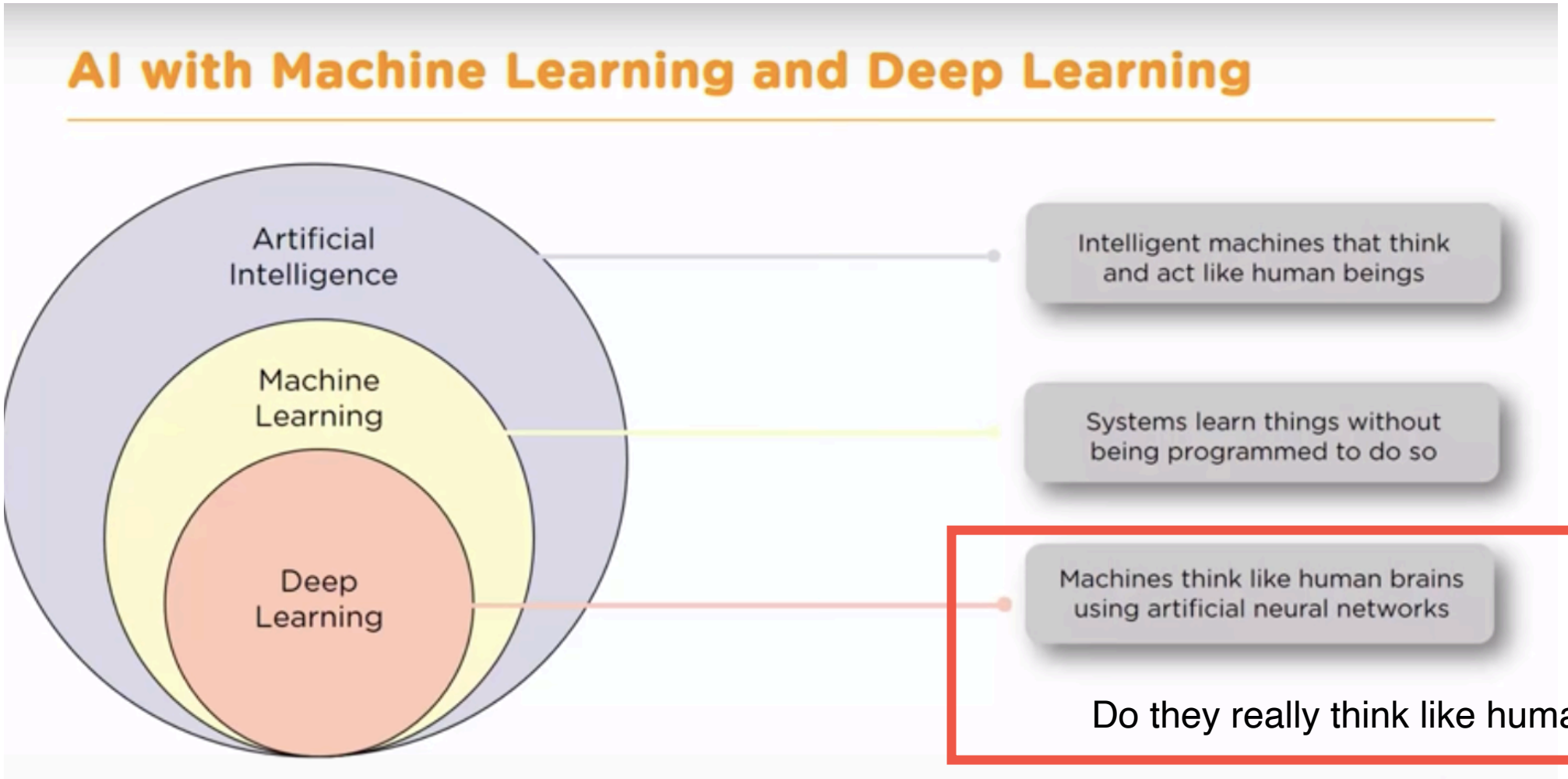
$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

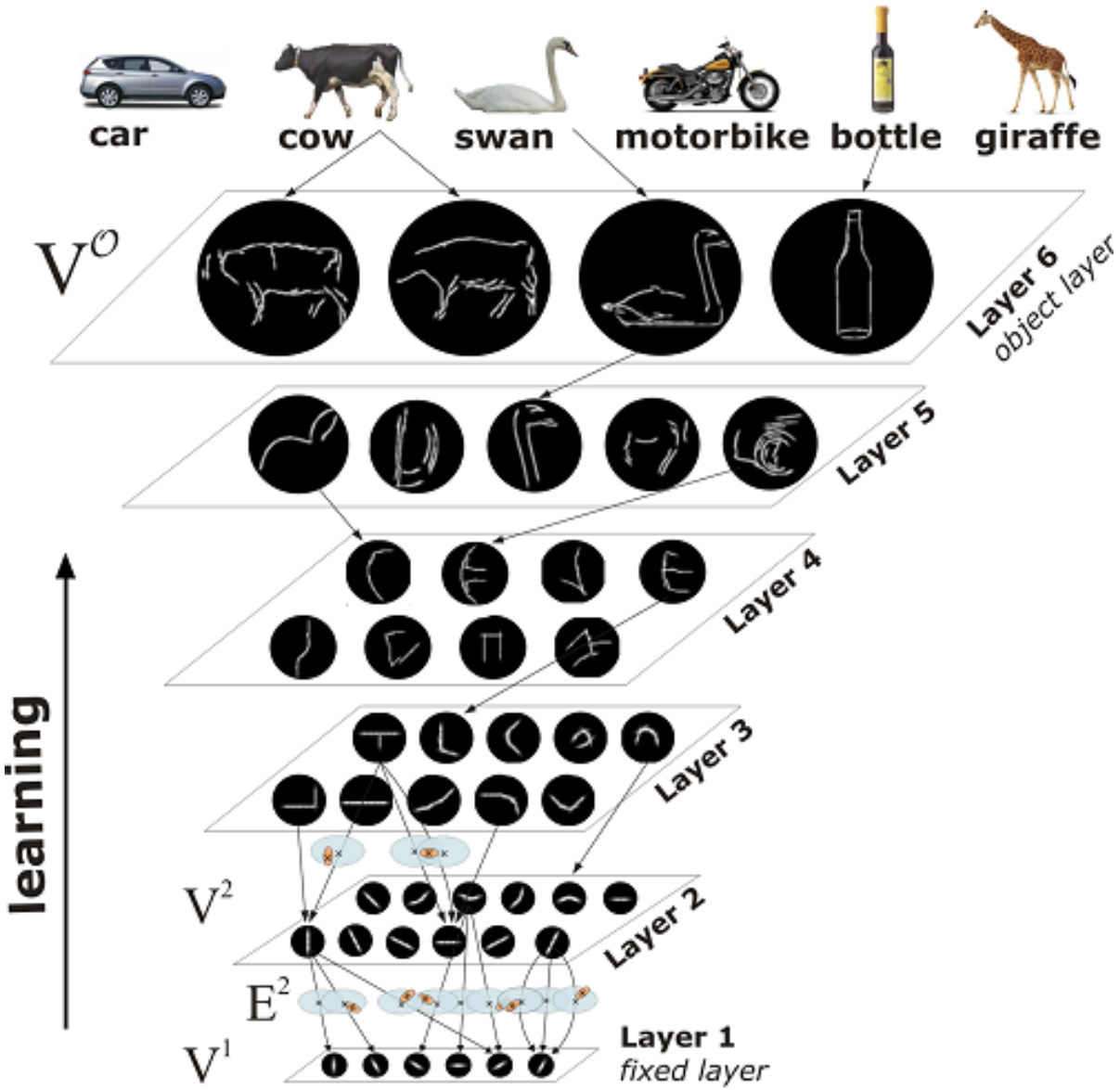
Deep Learning

Neural Networks Lesson 2

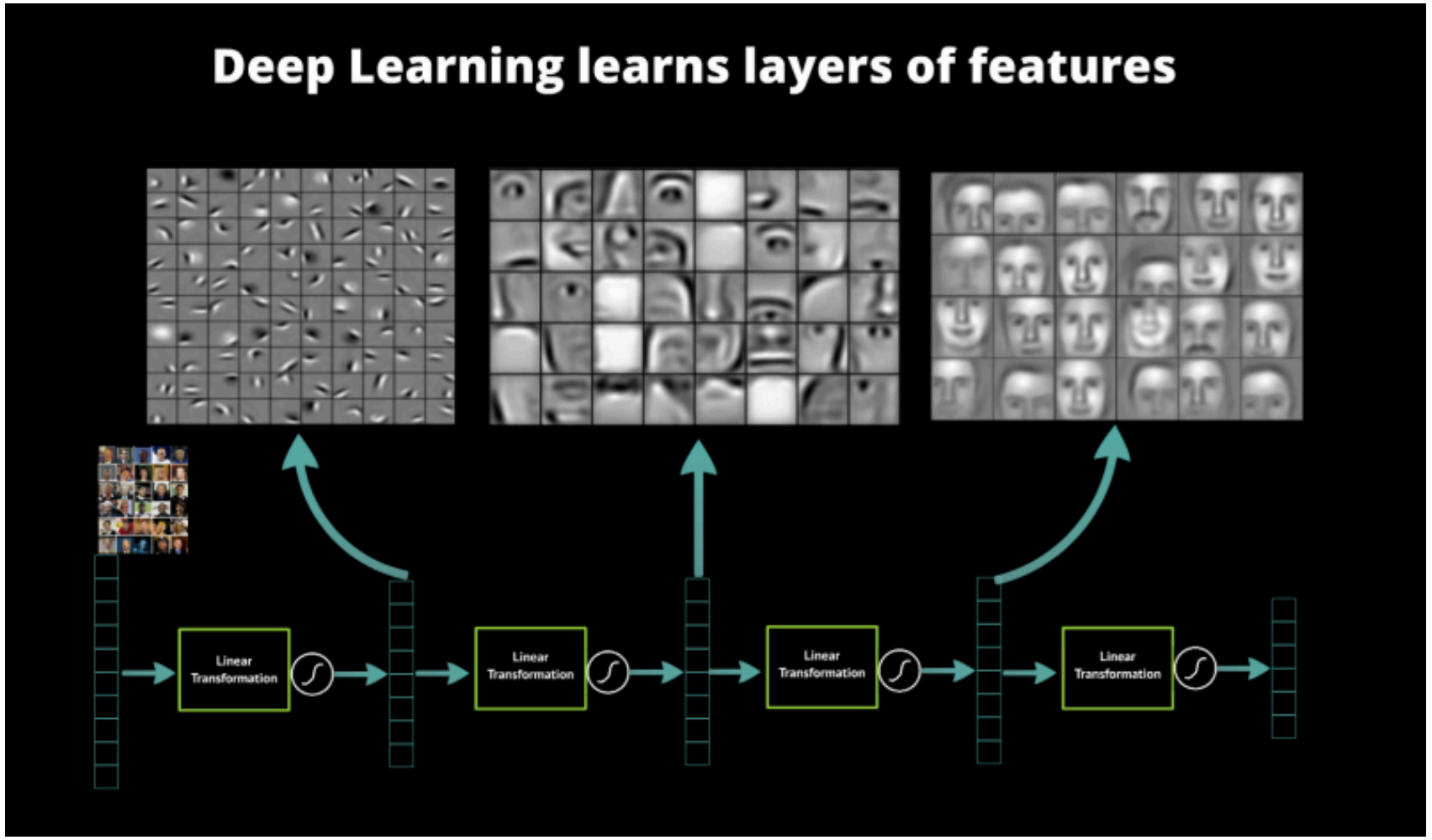
Background



Ref: YouTube video by SimpliLearn


car
cow
swan
motorbike
bottle
giraffe
V^O
Layer 6
object layer
Layer 5
Layer 4
Layer 3
V^2
Layer 2
E^2
V^1
Layer 1
fixed layer
learning

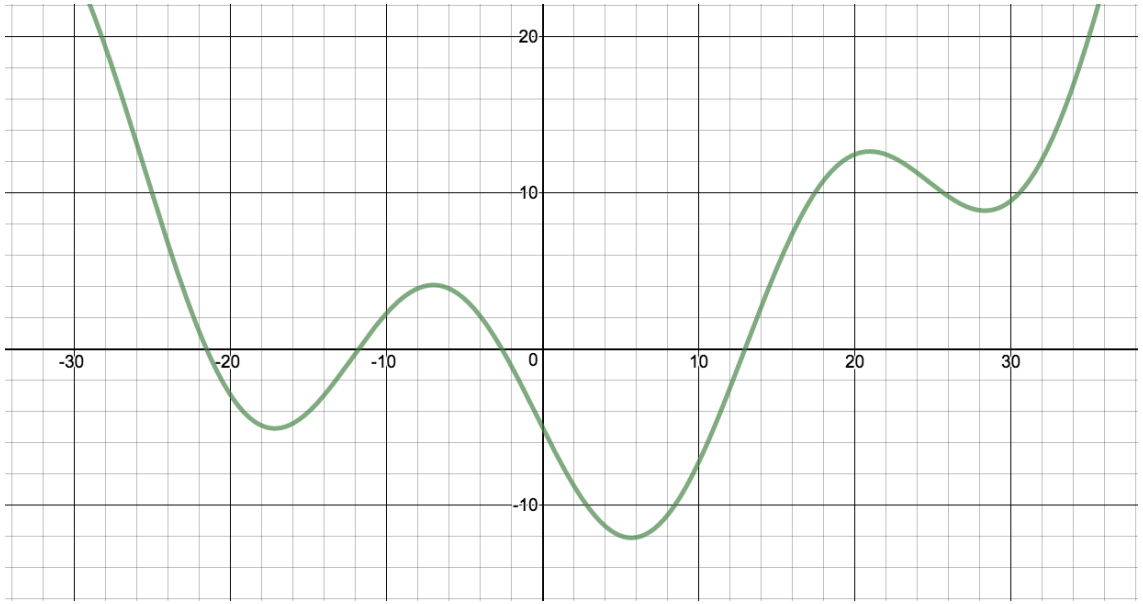

WE NEED TO GO
DEEPER

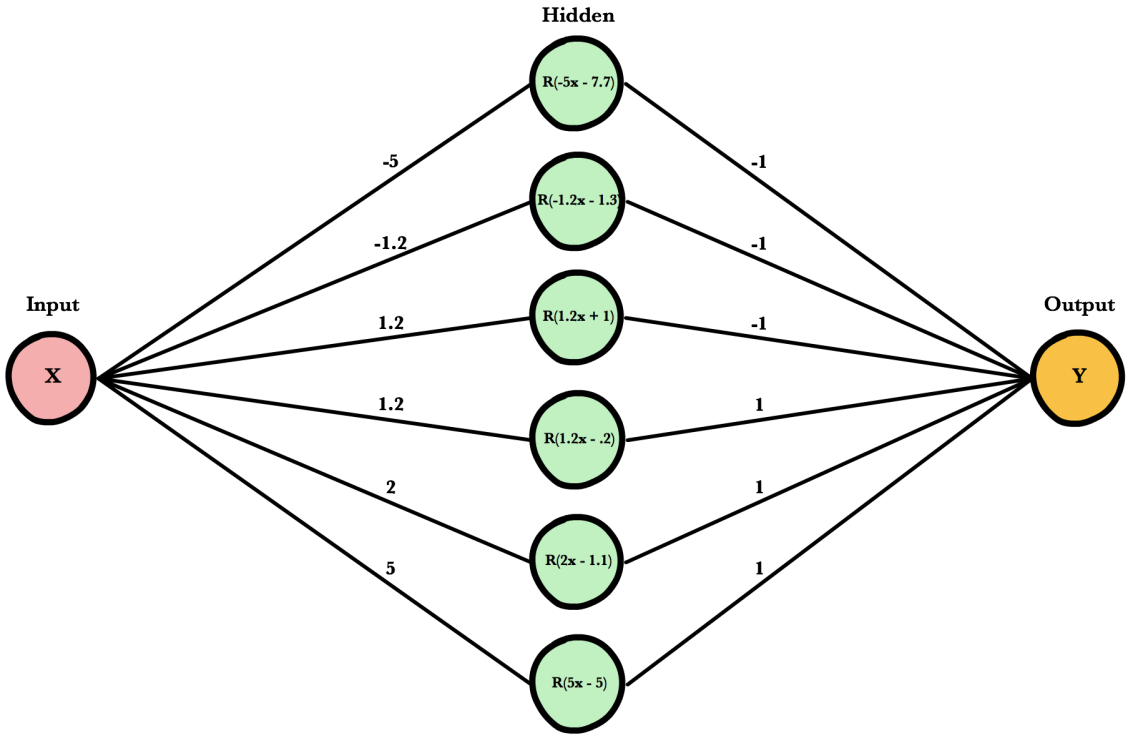


Hierarchy of features learned on face images in a classification task

https://www.datasciencecentral.com/profiles/blogs/a-primer-on-deep-learning

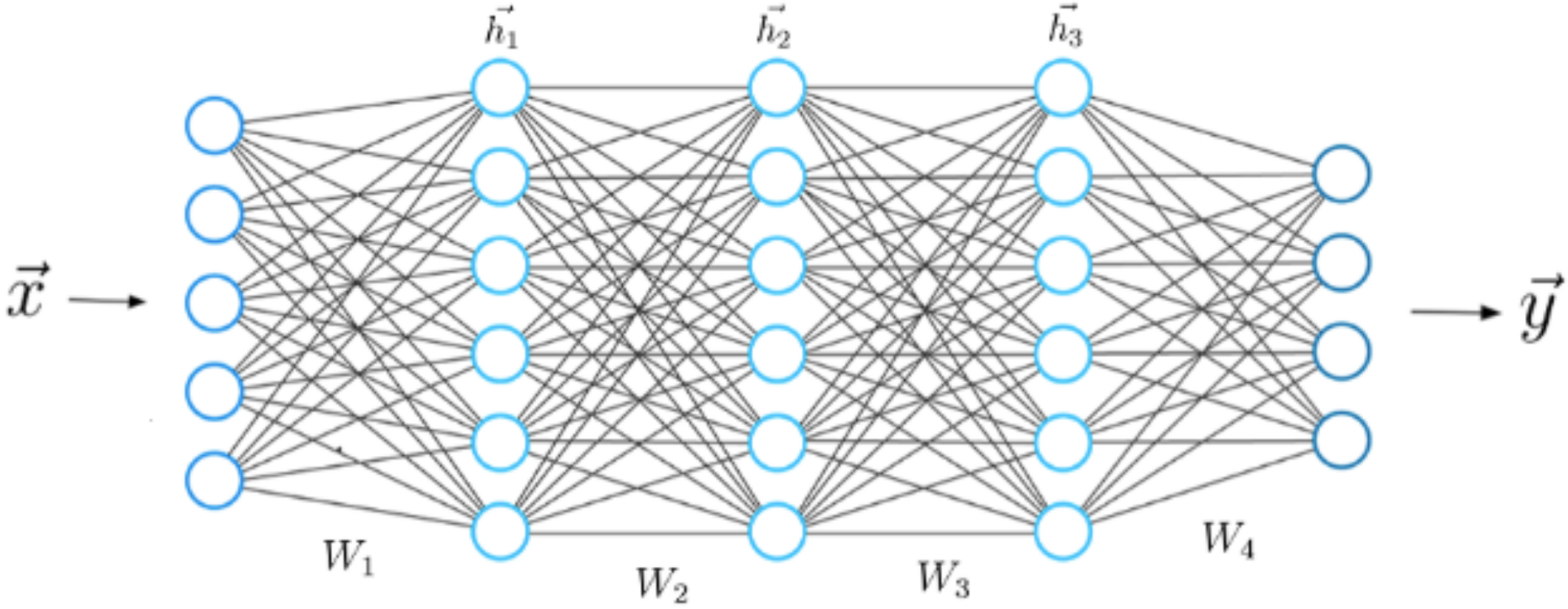
Architecture





A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.
— Ian Goodfellow

Ref: https://towardsdatascience.com/can-neural-networks-really-learn-any-function-65e106617fc6



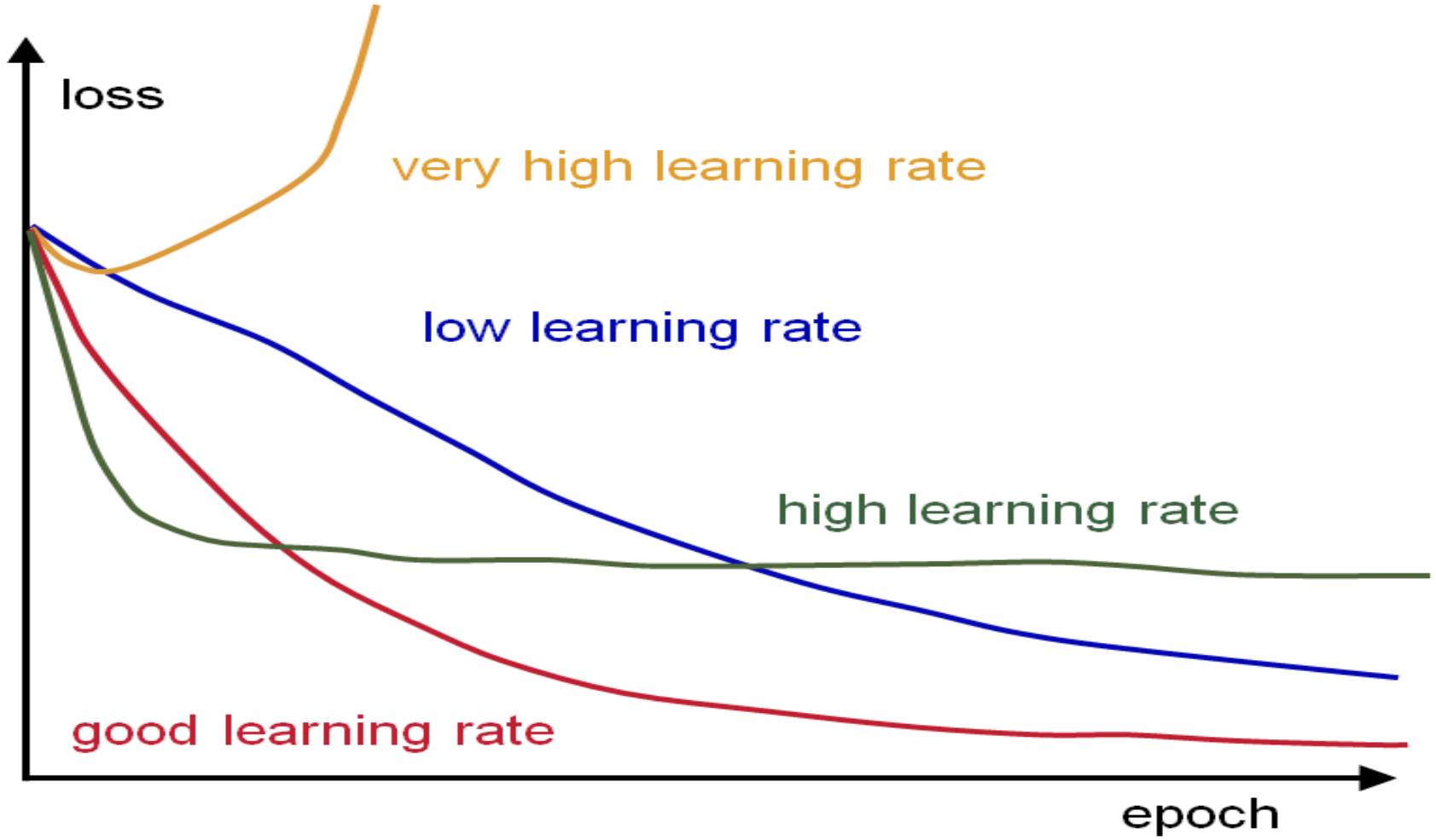
2 or more hidden layers is DEEP!

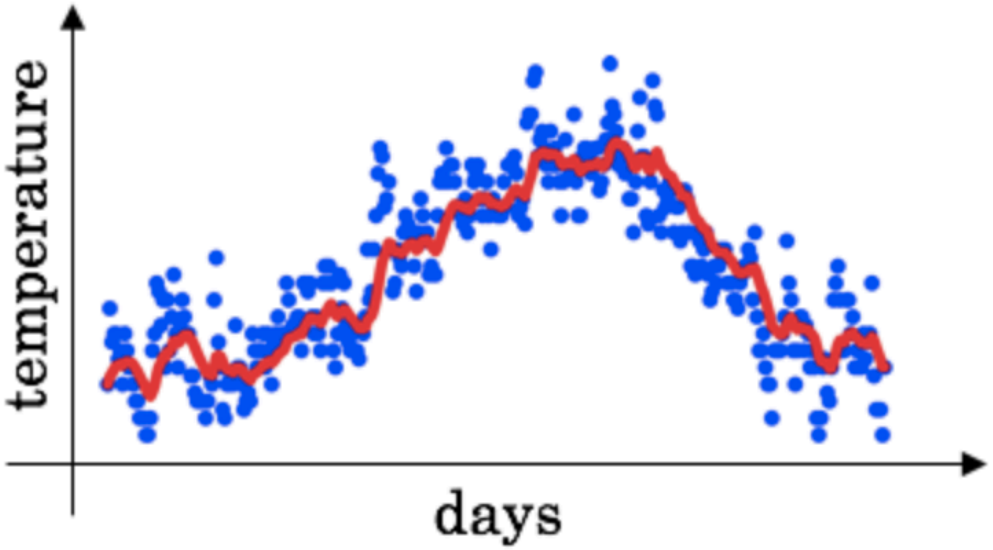
- Most structured-data problems can be handled with at-most 2 layers. Rare are problems where higher layers can be justified.

- “rules of thumb” exist but are absolute rules. Experiment and validate and then cross validate like any other hyper parameter

See ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hu

Effect of Learning rate





$$V_0 = 0$$
$$V_1 = V_0\beta + (1-\beta)t_1$$
$$V_2 = V_1\beta + (1-\beta)t_2$$

V is averaging temperature over $1-\beta$ days.

For more details on Exponentially weighted average, watch this
https://www.youtube.com/watch?v=lAq96T8FkTw

Basic idea is to faster convergence by smoothening these oscillations

$$W = W - \text{learning rate} * dW$$

$$b = b - \text{learning rate} * db$$

Back propagation

Initialize with 0

$$V_{dW} = \beta \times V_{dW} + (1 - \beta) \times dW$$

$\beta = 0.9$ is reasonable value

$$V_{db} = \beta \times V_{db} + (1 - \beta) \times db$$

Exponential Weighted Average

$$W = W - \text{learning rate} * V_{dW}$$

$$b = b - \text{learning rate} * V_{db}$$

Weight updates

Note that $dw = \frac{dj}{dw}$

$db = \frac{dj}{db}$

Problems with Adding more layers

- Overfitting
- Vanishing/Exploding gradients

Vanishing Gradients

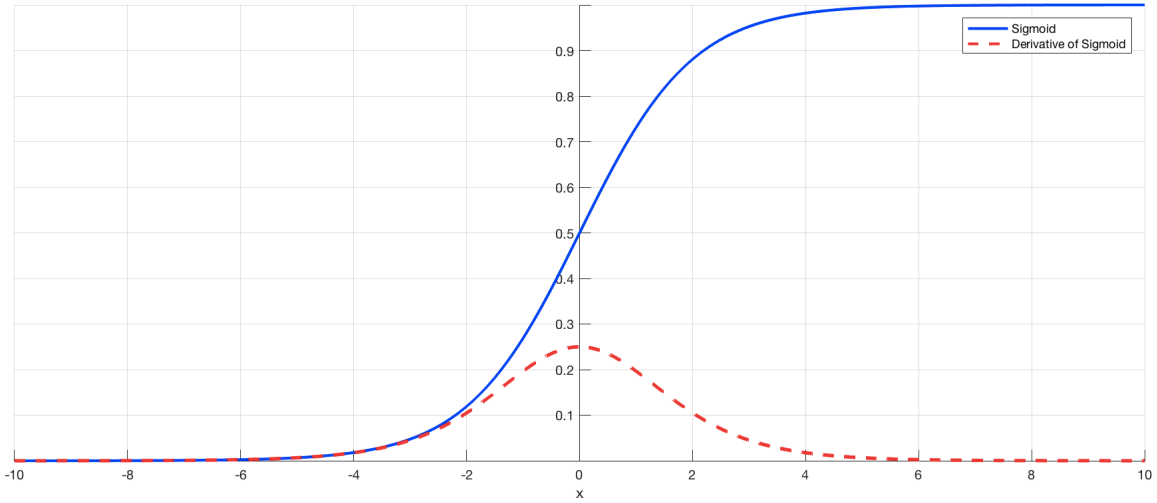
Backpropagation.

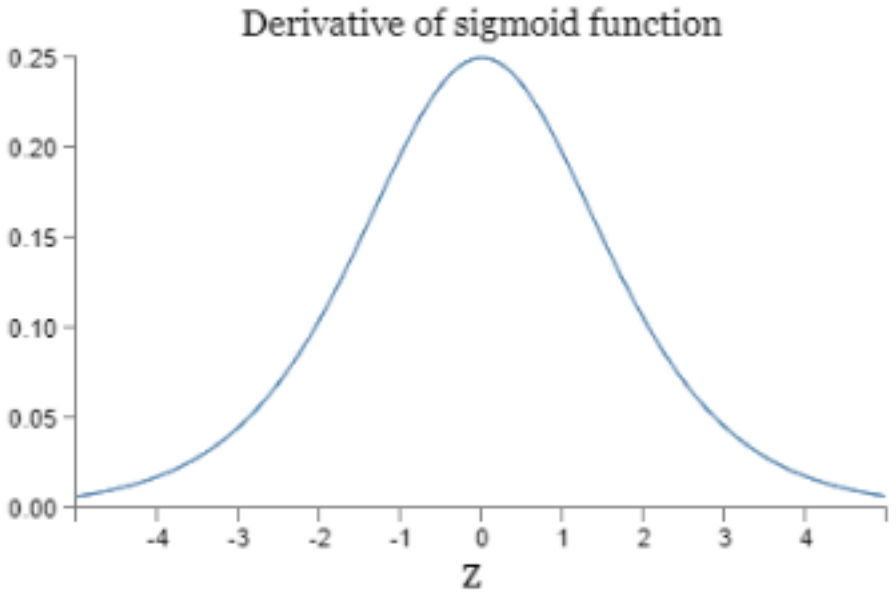
Gradients are calculated in backpropagation from last layer to first layer. Gradient and weights in each layer is multiplied to that in the previous layer (Chain rule)

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$

If weights are initialized too high or too low, then the product can vanish or explode by the time it reaches the first layer

Vanishing Gradient in Sigmoid Activation





Small value of sigmoid derivative (0.25) plays an important role in Vanishing Gradient.

→ Initialization

$$w \alpha \sqrt{\frac{1}{size^{[l-1]}}}$$

Xavier Initialization

$$w \alpha \sqrt{\frac{2}{size^{[l-1]}}}$$

Can be a hyperparameter

He et al. Initialization for RELU activation

*w=np.random.randn(layer_size[l],layer_size[l-1])*np.sqrt(2/ layer_size[l-1])*

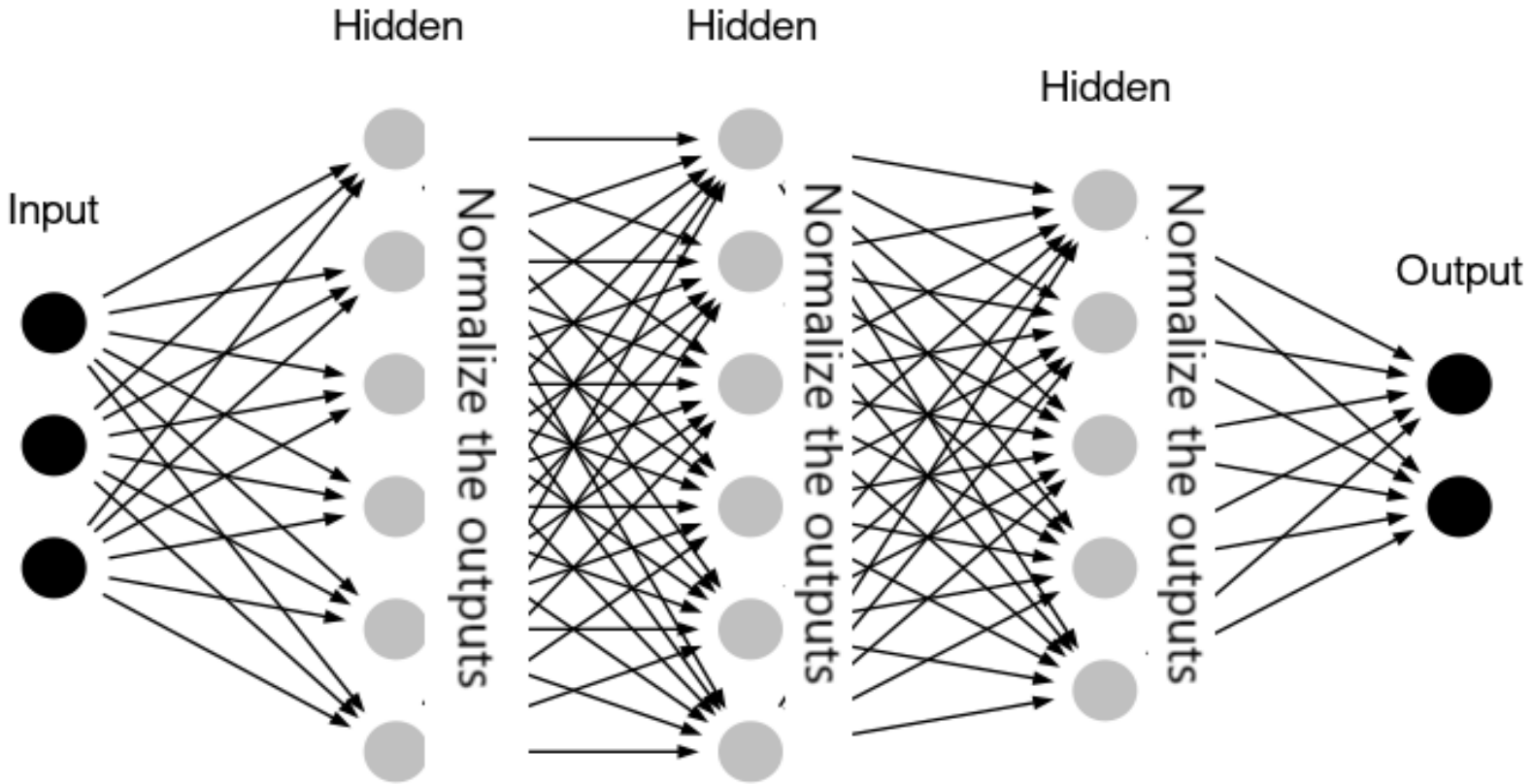
Can weights be all initialized to zero instead?

→ Normalization

Normalization: Motivation

- Like any algorithm, normalization also helps keeping the inputs in the same scale.
- Distribution of each layer's inputs change during training, as the parameters of the previous layers change.
- This phenomenon termed ***"Internal covariate shift"*** is addressed by normalizing layer inputs.
- Normalization is done for each training mini-batch which is referred to as "Batch normalization"

Ref: https://arxiv.org/pdf/1502.03167v3.pdf



Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$

Normalizing the layer before the activation, Z, is done more often but there are literature which normalizes after the activation as well (a)

Due to this normalization “layers” between each fully connected layers, **the range of input distribution of each layer stays the same**, no matter the changes in the previous layer. Given x inputs from k-th neuron.

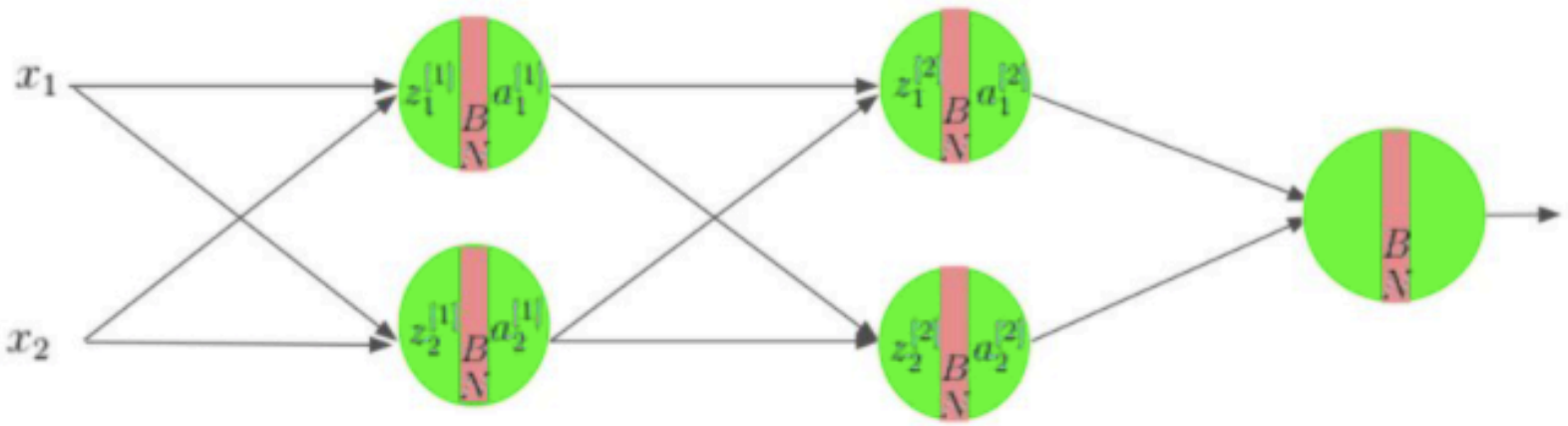
Consider a batch of activations at some layer. For making each feature dimension unit gaussian, use:

Where,
X^k = activation of layer k
$E[x^k]$= Mean

$\sqrt{\mathrm{Var}[x^{(k)}]}$ = standard deviation

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Sourced from: Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift by Ioffe and Szegedy , et.al 2015



$$z^{[l]} = W^{[l]} a^{[l-1]} \longrightarrow$$

$$\mu^{[l]} = \frac{1}{m} \sum_i z^{[l](i)}$$

$$\sigma^{[l]2} = \frac{1}{m} \sum_i (z^{[l](i)} - \mu^{[l]})^2$$

$$z_{norm}^{[l](i)} = \frac{z^{[l](i)} - \mu^{[l]}}{\sqrt{\sigma^{[l]2} + \epsilon}}$$

$$\tilde{z}^{[l](i)} = \gamma^{[l]} z_{norm}^{[l](i)} + \beta^{[l]}$$

$$\longrightarrow a^{[l]} = g^{[l]}(\tilde{z}^{[l]})$$

Ref: https://learnopencv.com/batch-normalization-in-deep-networks/

Batch normalization: Summary

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Parameter for numerical stability if variance is close to zero

γ *and* β *are learnable parameters.*
To make $y_i = \hat{x}_i$, use
$\beta = \mu$ *and*
$\gamma = \sqrt{(\sigma^2 + \epsilon)}$

https://arxiv.org/pdf/1502.03167v3.pdf

- Efficient (Same accuracy in lesser iterations)
- Acts as a regularizer too, in some cases eliminating the need for Dropout (Not necessarily always and should not replace regularizing)

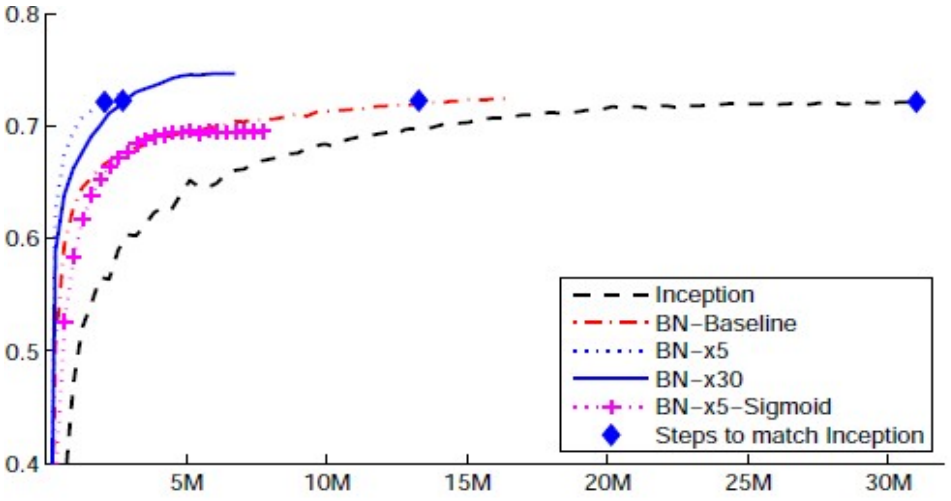


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Figure 3: *For Inception and the batch-normalized variants, the number of training steps required to reach the maximum accuracy of Inception (72.2%), and the maximum accuracy achieved by the network.*

Regularization

$$J\left(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}\right) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\sum_{l=1}^{L} \left\|w^{[L]}\right\|_F^2$$

L2 regularization

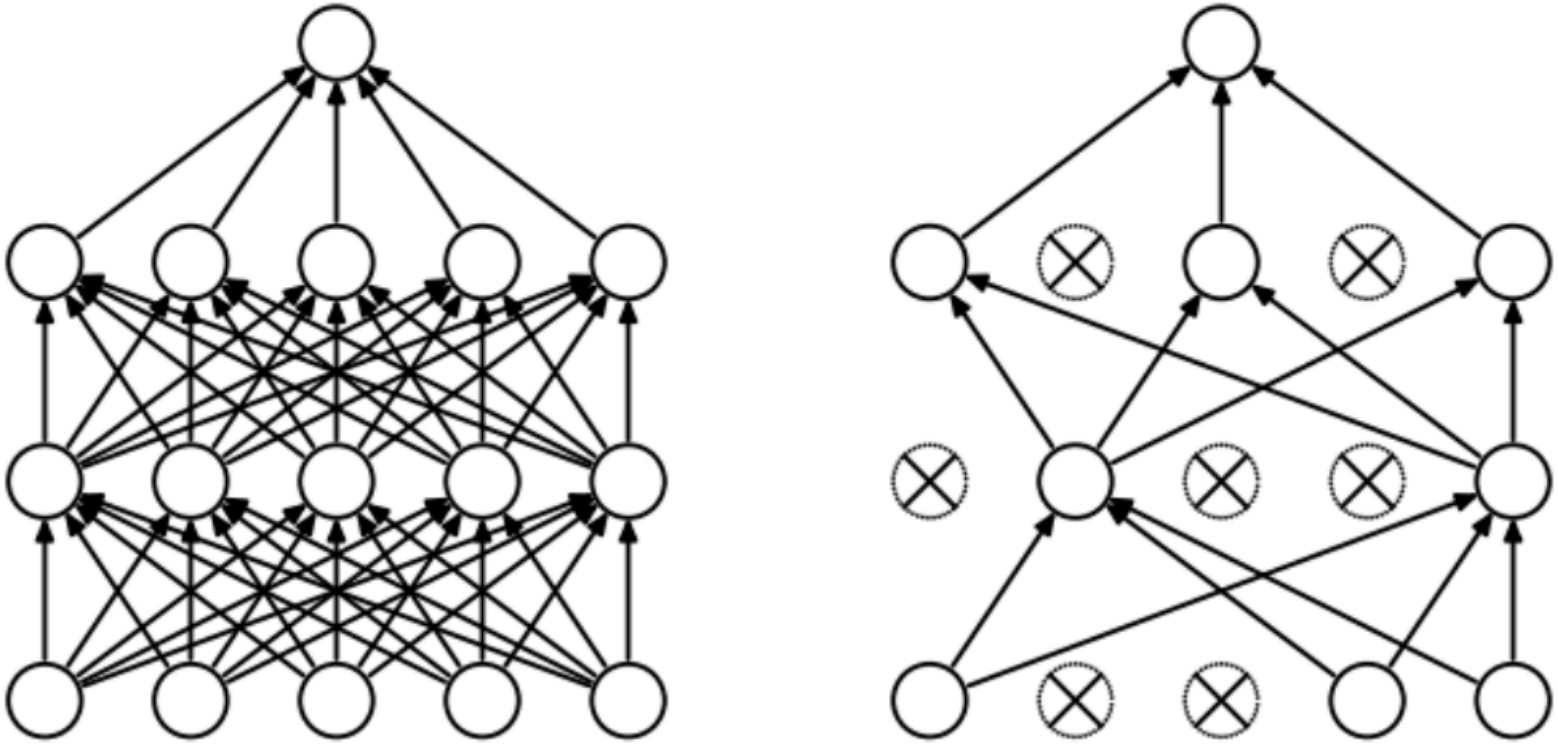
Regularization Parameter

If the weight increases, regularizing term also increases and hence increasing the cost function.

To minimize cost function, weights have to be small

Assumption is that smaller weights will not over-fit. (They will not give high importance to any one neuron)

Idea is as always to prevent over-fitting by making it NOT memorize



(a) Standard Neural Net

(b) After applying dropout.

- Nodes dropped out with some probability, p.
- Each layer can have its own keep_prob (p) for dropping out certain nodes

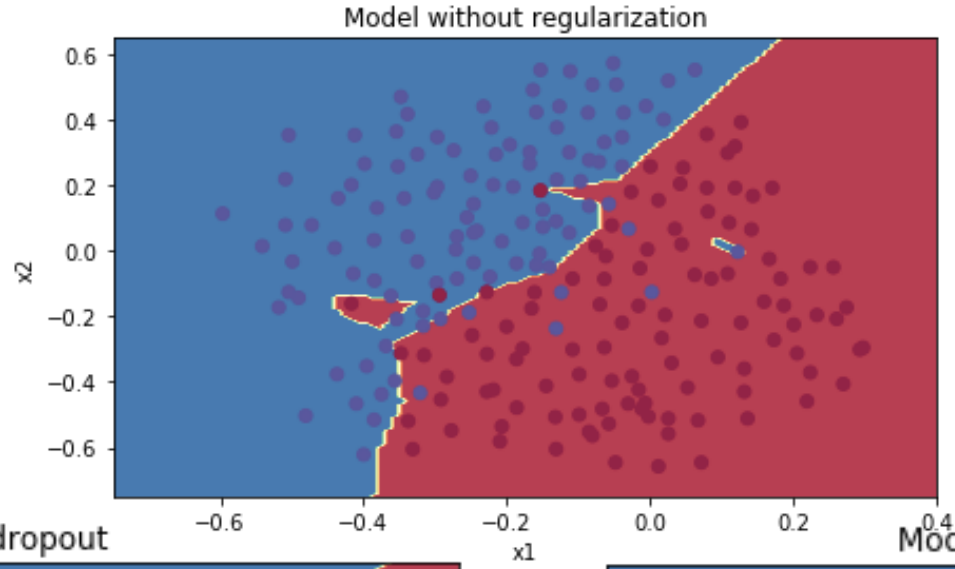
- For each mini-batch, different set of neurons can be dropped out. Hence, different training examples, learn on different neurons
- Weight update for each neuron each drop-out is shared in overall weight matrix
- During test time, data is passed through full networks with weights scaled as per the number of times the appeared in the drop-out
- Alternative to above is, (that is scaling during test time), is to use inverted drop-out where scaling parameter is included in the training itself (inverted, i.e. divided by p)

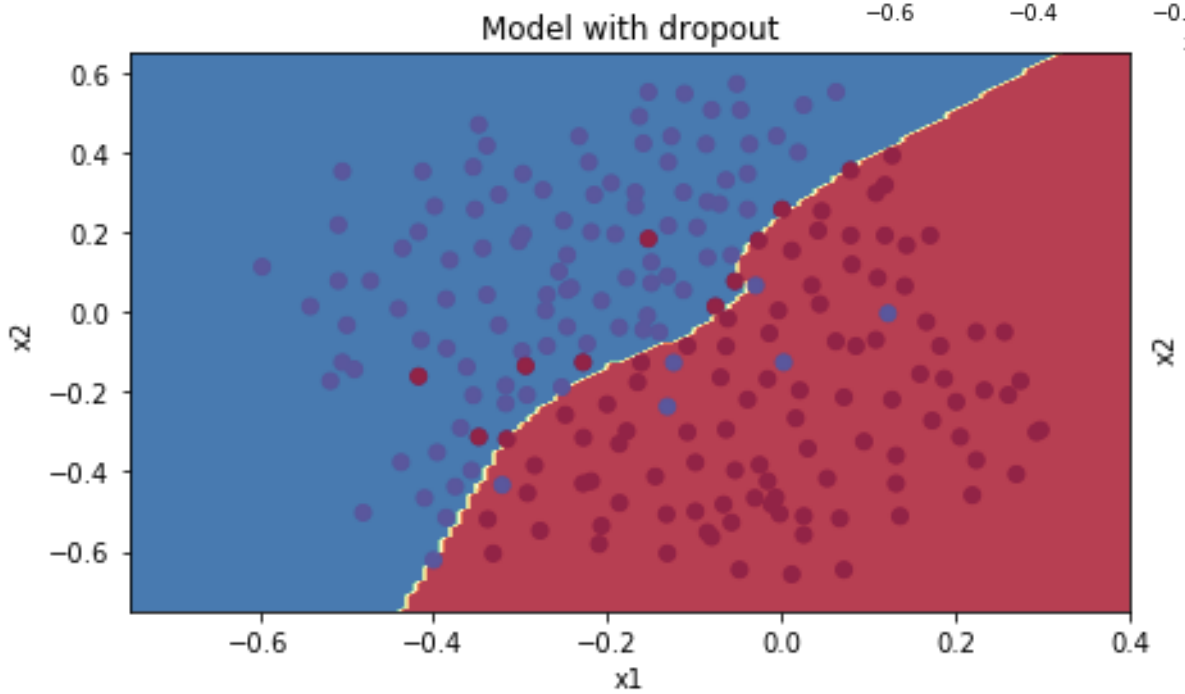
Implementation of Drop-out regularization

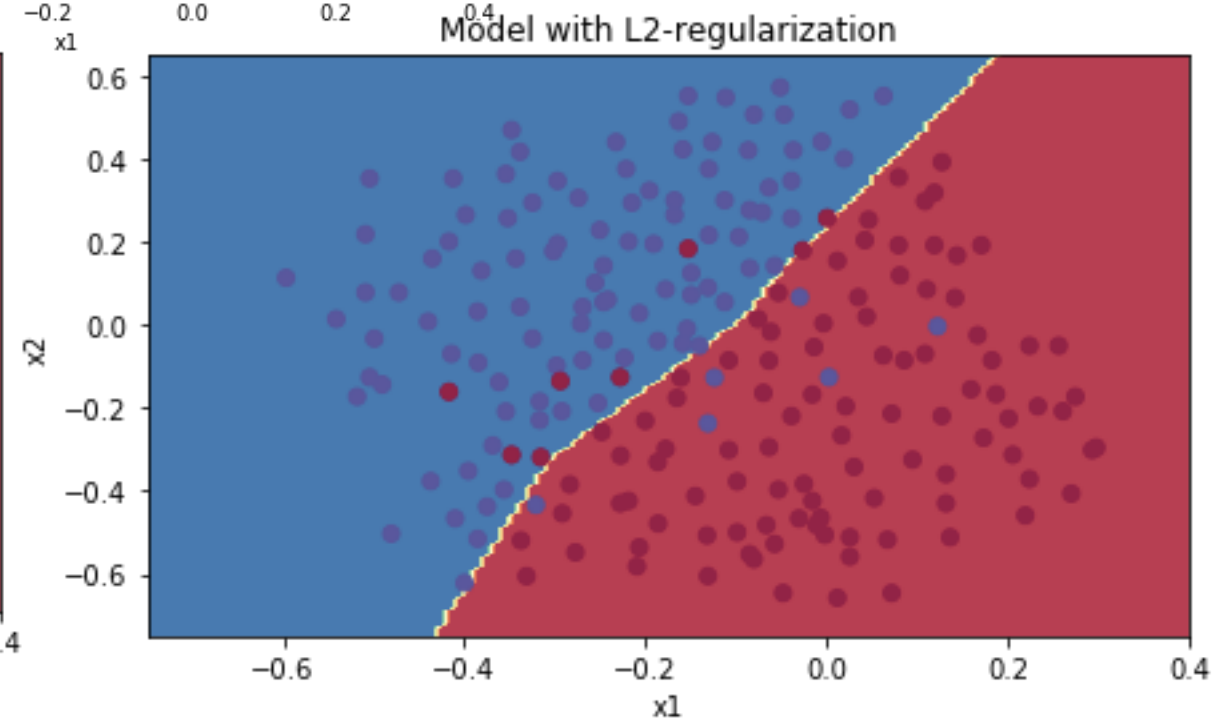
```
# LINEAR -> RELU -> LINEAR -> RELU -> LINEAR -> SIGMOID
Z1 = np.dot(W1, X) + b1
A1 = relu(Z1)
### START CODE HERE ### (approx. 4 lines)         # Steps 1-4 below correspond to the Steps 1-4 described above.
D1 = np.random.rand(A1.shape[0],A1.shape[1])        # Step 1: initialize matrix D1 = np.random.rand(..., ...)
D1 = (D1<keep_prob).astype(int)                     # Step 2: convert entries of D1 to 0 or 1 (using keep_prob as t
A1 = A1*D1                                         # Step 3: shut down some neurons of A1
A1 = A1/keep_prob                                  # Step 4: scale the value of neurons that haven't been shut down
### END CODE HERE ###
Z2 = np.dot(W2, A1) + b2
A2 = relu(Z2)
### START CODE HERE ### (approx. 4 lines)
D2 = np.random.rand(A2.shape[0],A2.shape[1])        # Step 1: initialize matrix D2 = np.random.rand(..., ...)
D2 = (D2<keep_prob).astype(int)                     # Step 2: convert entries of D2 to 0 or 1 (using keep_prob as th
A2 = A2*D2                                         # Step 3: shut down some neurons of A2
A2 = A2/keep_prob                                  # Step 4: scale the value of neurons that haven't been shut down
### END CODE HERE ###
Z3 = np.dot(W3, A2) + b3
A3 = sigmoid(Z3)
```

Inverted drop-out , to avoid scaling at test time

Effectiveness of regularization



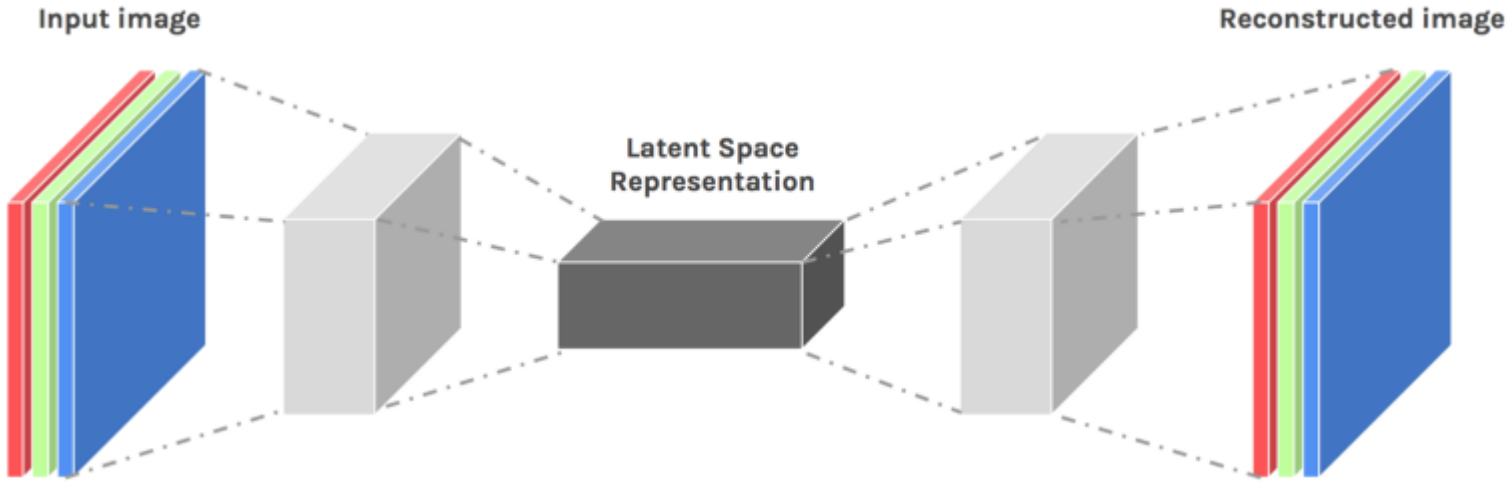




Ref: Programming Assignment in DeepLearing Specialization on CourseEra

AutoEncoders

- An autoencoder neural network is an **unsupervised** Machine learning algorithm
- An autoencoder are neural networks which are trained to copy its input to its output.
- Usually constrained to make their job difficult
- Almost always have encoder (f), and decoder (g) so that model is x=g(f(x))



https://towardsdatascience.com/autoencoders-bits-and-bytes-of-deep-learning-eaba376f23ad

1. Dimensionality reduction

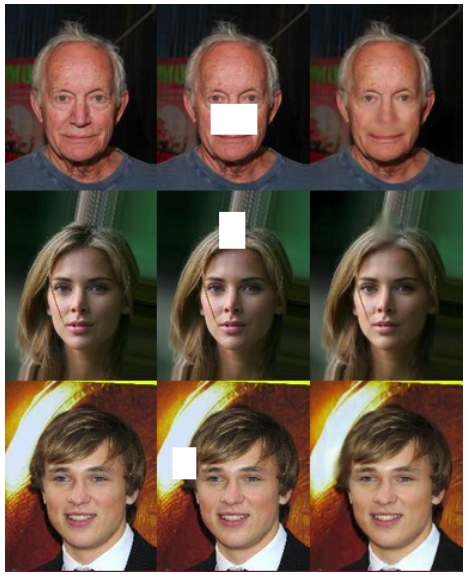
2. Image processing application:
   a) - De-noising
   b) - Auto filling

3. Anomaly detection.

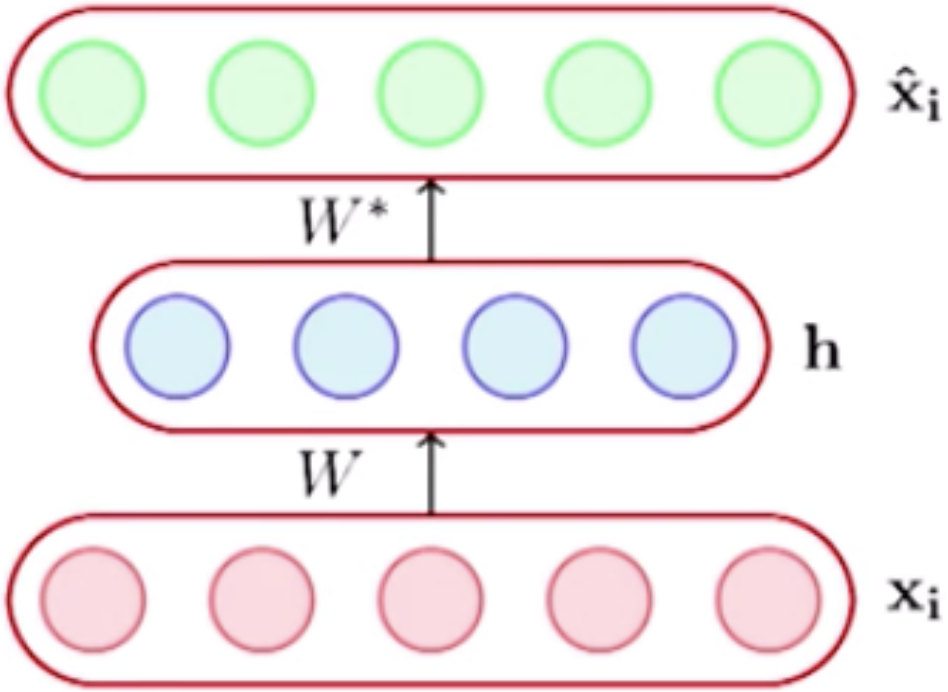
4. Feature generation.

5. Text translation

6. …..



www.cc.gatech.edu



$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

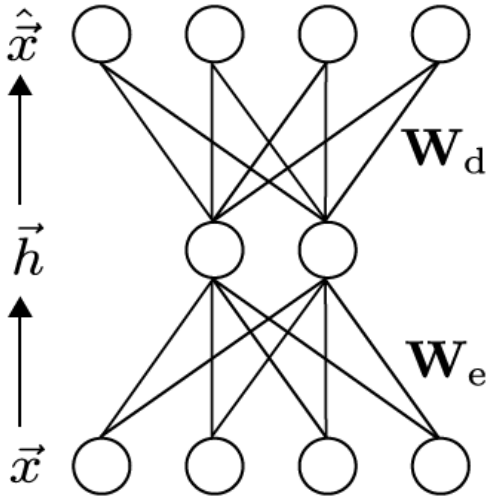


FIGURE 1.3: The standard autoencoder model.

$$\min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{n}(\hat{x}_{ij} - x_{ij})^2$$

$$i.e., \min_{W,W^*,\mathbf{c},\mathbf{b}} \frac{1}{m}\sum_{i=1}^{m}(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})^T(\hat{\mathbf{x}}_\mathbf{i} - \mathbf{x}_\mathbf{i})$$

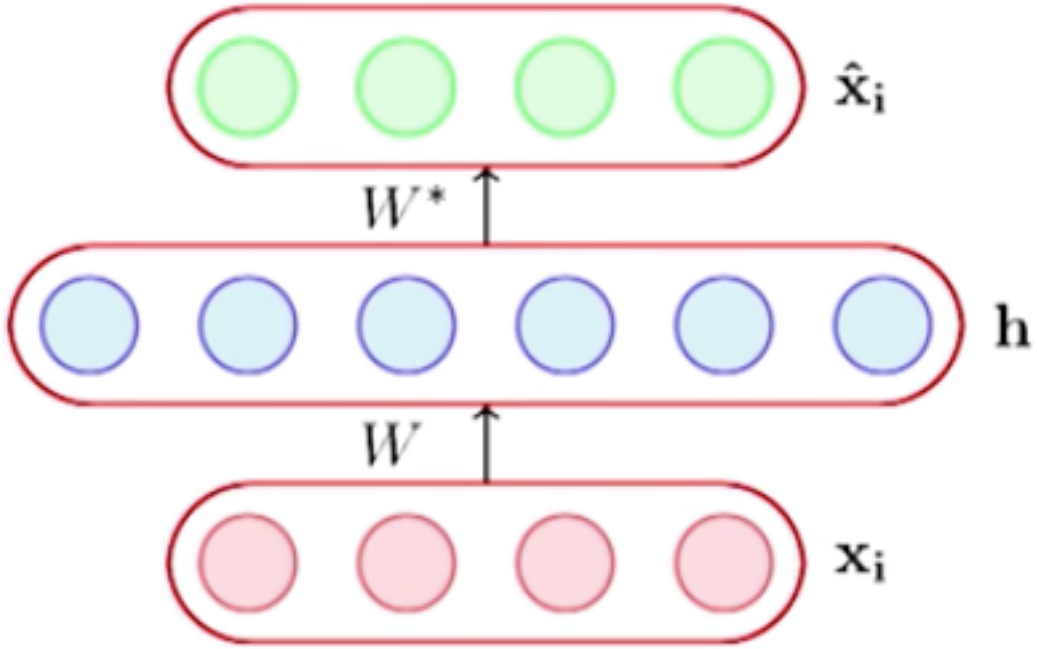
Ref: Video Lecture from NTPEL

PCA is essentially a linear transformation while Auto-encoders can model non linear functions.

PCA features uncorrelated (orthogonal) while autoencoded features might have correlations

PCA is faster and computationally cheaper than autoencoders.

Autoencoder is prone to overfitting due to high number of parameters.

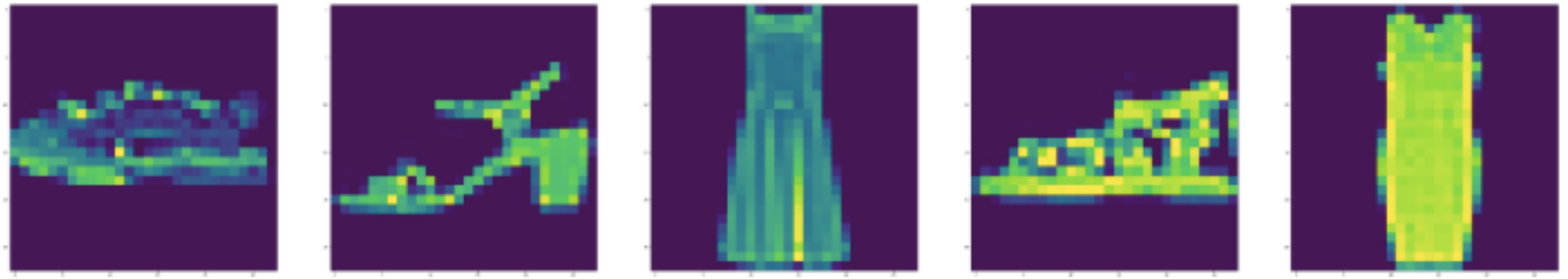


$$\mathbf{h} = g(W\mathbf{x_i} + \mathbf{b})$$

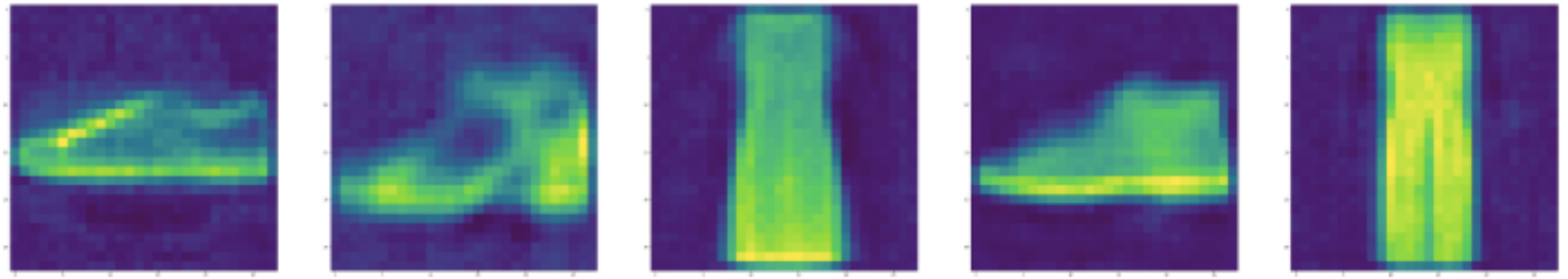
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

$$\dim(\mathbf{h}) \geq \dim(\mathbf{x_i})$$

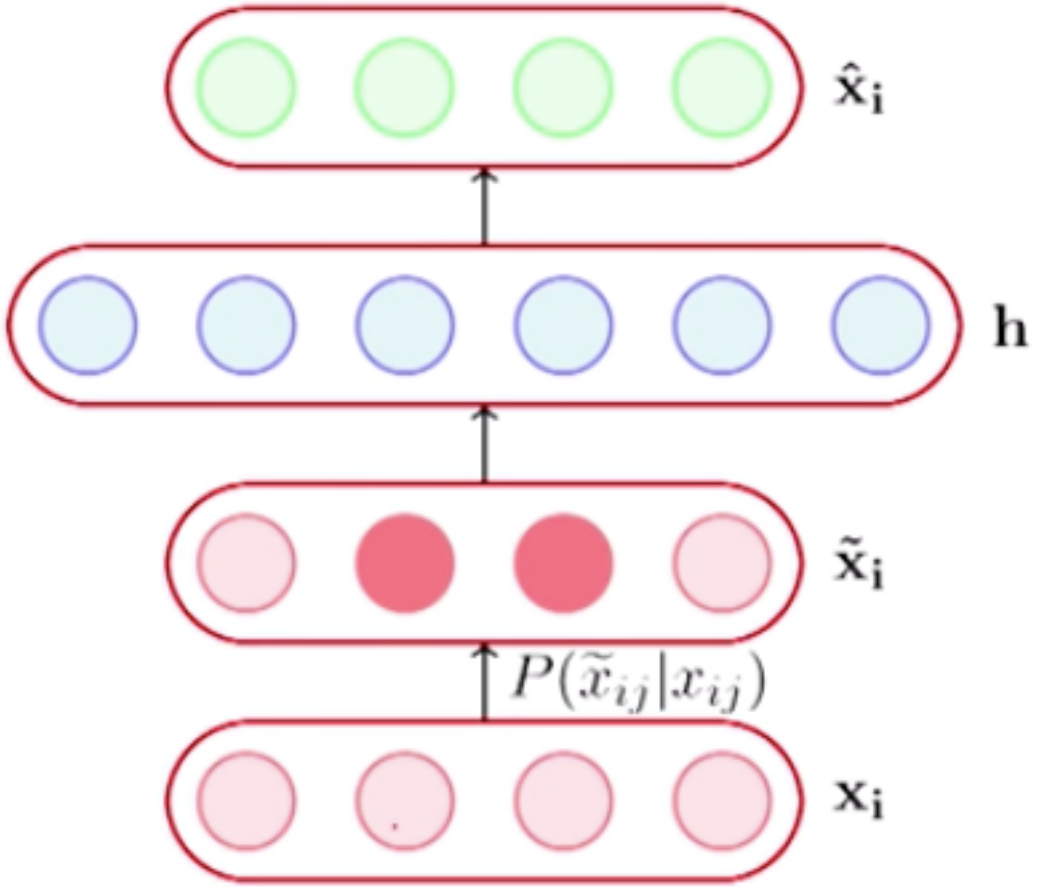
Ref: Video Lecture from NTPEL



Autoencoders compression from 784 pixels to 10 pixels



```
## input layer
input_layer = Input(shape=(784,))

## encoding architecture
encode_layer1 = Dense(1500, activation='relu')(input_layer)
encode_layer2 = Dense(1000, activation='relu')(encode_layer1)
encode_layer3 = Dense(500, activation='relu')(encode_layer2)

## latent view
latent_view   = Dense(10, activation='sigmoid')(encode_layer3)

## decoding architecture
decode_layer1 = Dense(500, activation='relu')(latent_view)
decode_layer2 = Dense(1000, activation='relu')(decode_layer1)
decode_layer3 = Dense(1500, activation='relu')(decode_layer2)

## output layer
output_layer  = Dense(784)(decode_layer3)

model = Model(input_layer, output_layer)
```



$$P(\tilde{x}_{ij} = 0|x_{ij}) = q$$
$$P(\tilde{x}_{ij} = x_{ij}|x_{ij}) = 1 - q$$

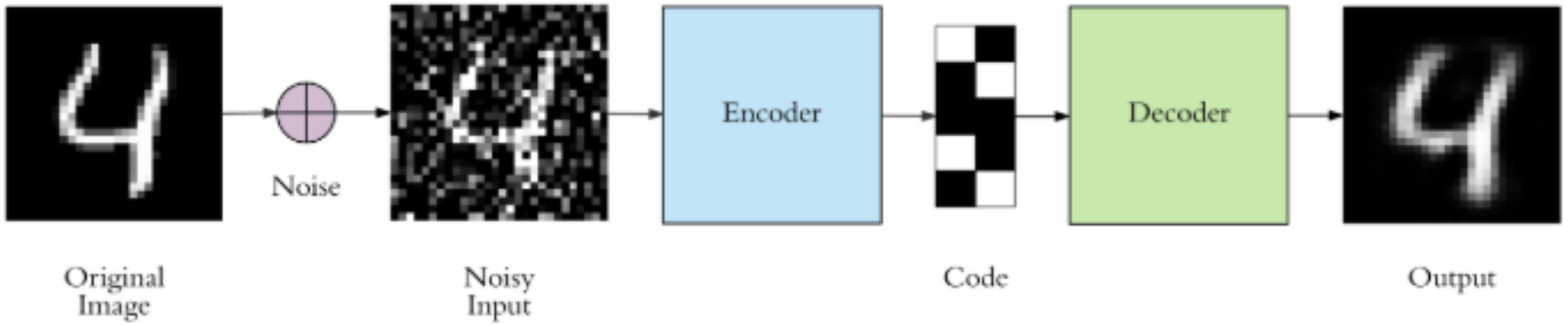
```
autoencoder.fit(x_train, x_train)
```

Denoising autoencoder is trained as:

```
autoencoder.fit(x_train_noisy, x_train)
```

Ref: Video Lecture from NTPEL

REF: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798



Denoising autoencoders can be convolution or fully connected.

- Autoencoders regularized by using a *sparsity constraint* such that only a fraction of the nodes would have nonzero values, called active nodes.
- Penalty term to the loss function such that only a fraction of the nodes become active.
- This forces the autoencoder to represent each input as a combination of small number of nodes, and demands it to discover interesting structure in the data.

Ref: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Average number of times neuron was active for all m examples

$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^{m} h(\mathbf{x}_i)_l$$

Sparsity parameter (Usually very small 0.005)

$$\Omega(\theta) = \sum_{l=1}^{k} \rho \log \frac{\rho}{\hat{\rho}_l} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_l}$$

Regularizing term to be added to the cost function

1) Autoencoders are data-specific (Not appropriate for data which is not similar to train data)

2) Autoencoders are lossy (Not good for compression). Dimensionality reduction is used mainly for

a) Visualization (like t-sne on reduced visuals)
b) Using reduced features to pass on to other ML algorithm such as XG Boost

3) Autoencoders are learned automatically from data examples and no other feature engineering is required. Just need more data.

Embeddings

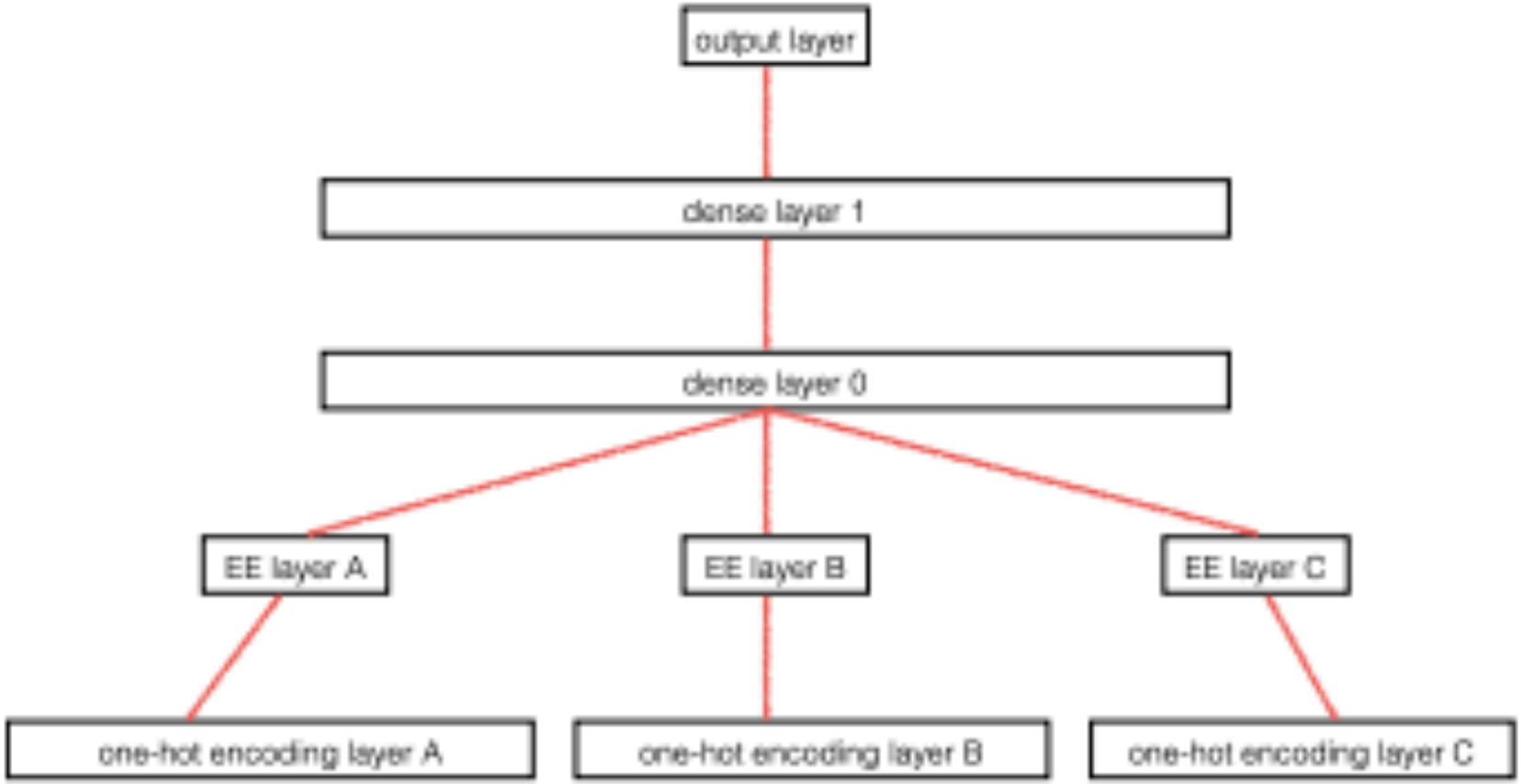
- Two Perspectives

1. Word embeddings for NLP, sentiments, context, etc.
2. Categorical variable in structural data with high cardinality

- In mathematics embedding refers to mapping from space X to Y while preserving some structure of the objects (e.g. their distances).
- In the context of neural networks usually embedding means converting categorical variables into continuous vectors.
- The rise of neural networks in natural language processing is based on the word embedding which puts words with similar meaning closer to each other.

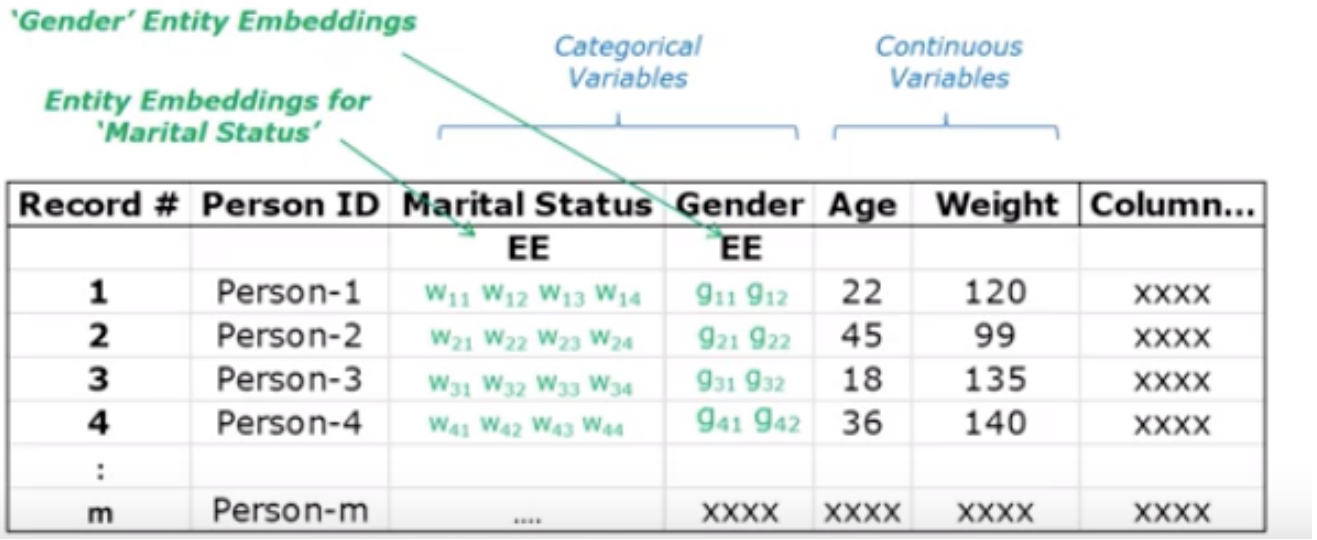
Reference: https://neuro.cs.ut.ee/the-use-of-embeddings-in-openai-five/

- In principle a neural network can approximate any continuous function. The continuous nature of neural networks limits their applicability to categorical variables.
- A common way to circumvent this problem is to use onehot encoding, but it has two shortcomings:
  - Unrealistic number of features and computational requirement.
  - Different values of categorical variables are treated independently by ignoring relationship between each other



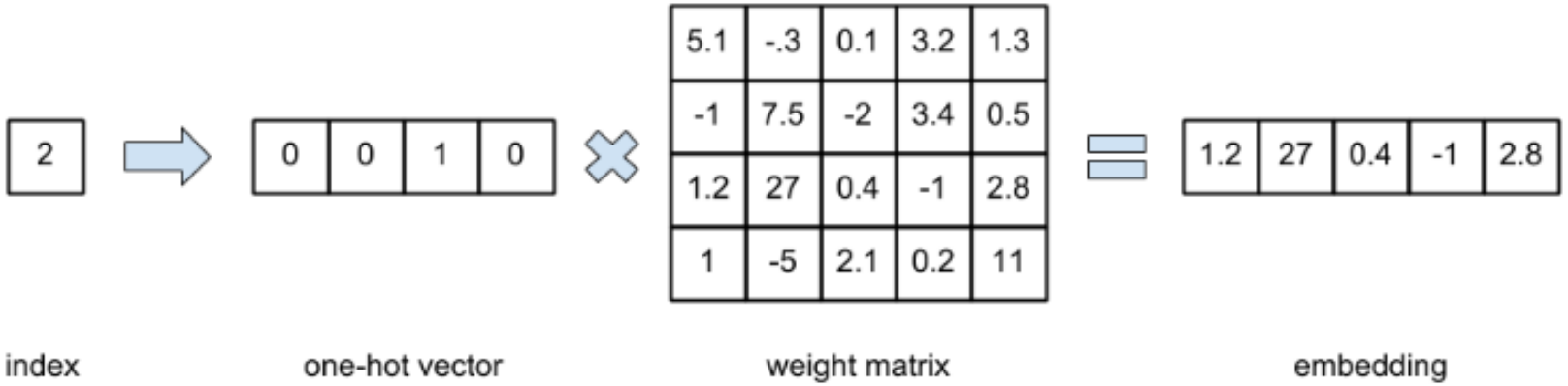
Reference: https://arxiv.org/pdf/1604.06737.pdf

Learning Embeddings: The lookup table



Record #	Person ID	Marital Status	Gender	Age	Weight	Column...
		EE	EE			
1	Person-1	w_{11} w_{12} w_{13} w_{14}	g_{11} g_{12}	22	120	xxxx
2	Person-2	w_{21} w_{22} w_{23} w_{24}	g_{21} g_{22}	45	99	xxxx
3	Person-3	w_{31} w_{32} w_{33} w_{34}	g_{31} g_{32}	18	135	xxxx
4	Person-4	w_{41} w_{42} w_{43} w_{44}	g_{41} g_{42}	36	140	xxxx
:						
m	Person-m	....	xxxx	xxxx	xxxx	xxxx

- Weight matrix is learnt like any other weights in the neural network during training
- Once learnt, this matrix serves as a look up table for converting any category to numeric vector
- Embedding is task specific and will not be valid for other tasks



Reference: https://neuro.cs.ut.ee/the-use-of-embeddings-in-openai-five/

puppy	[0.9, 1.0, 0.0]
dog	[1.0, 0.2, 0.0]
kitten	[0.0, 1.0, 0.9]
cat	[0.0, 0.2, 1.0]

3- Dimensional Embedding. First dimension probably refers to the breed and the second one to the age

- Deep Learning Challenges
  - Overfitting
    - L2 Regularization
    - Drop Out Regularization
  - Vanishing gradient
    - Weight initialization
    - Normalization
  - Speed
    - Momentum gradient descent
- Applications
  - Autoencoders
- Entity Embeddings


INSOF
INTERNATIONAL SCHOOL OF ENGINEERING
INSOFE
Inspire...Educate...Transform.