

# EEP 596: Adv Intro ML || Lecture 14

Dr. Karthik Mohan

Univ. of Washington, Seattle

February 21, 2023

# Logistics

- 1 Please pick a team-mate for your project and find a cool team name by EOD
- 2 Team sign up sheet in the discord channel *Friday*
- 3 Checkpoint submission for mini-project on ~~Sunday~~ night - Early submission to stay on track!
- 4 Kaggle Contest as well to try out your amazing Strategies and Algorithms!
- 5 Anything else?

# Last Time

- a Wrap up on Anomaly Detection

# Last Time

- a Wrap up on Anomaly Detection
- b Deep Learning Applications

# Today

- More Deep Learning facets
- Auto Encoders and types!
- Brain Storming Break-out Exercise
- Deep Learning in NLP
- Sequence to Sequence Models ]

# Deep Learning Reference

## Deep Learning

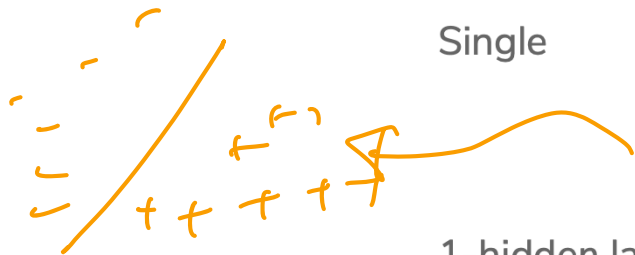
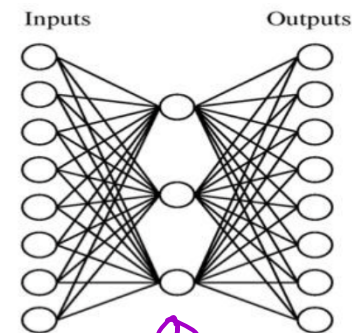
Great reference for the theory and fundamentals of deep learning: Book by Goodfellow and Bengio et al [Bengio et al](#)

# Logistic Regression to Deep Learning: Recap

Logistic

# 2 Layer Neural Network

Two layer neural network (alt. one hidden-layer neural network)



Single

1 hidden layers →

Non-linear Decision boundary  
 Non-linear model

A hand-drawn graph showing a set of data points (represented by '+' signs) separated by a non-linear decision boundary. The boundary is a curve that encloses some points and excludes others.

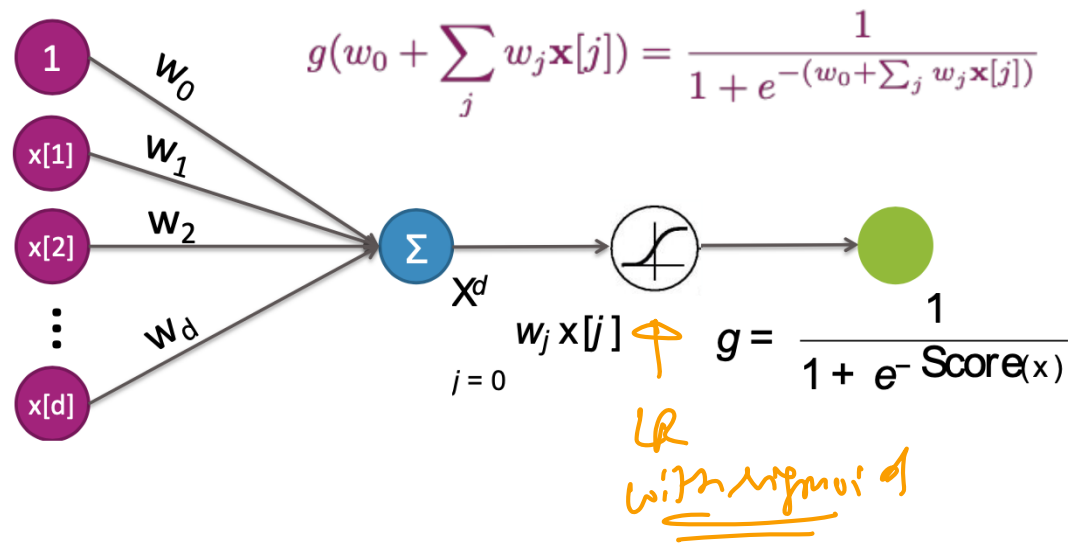
$$out(x) = g\left(w_0 + \sum_j w_j x[j]\right) \quad \text{LR}$$

1-hidden layer

$$out(x) = g\left(w_0 + \sum_k w_k g\left(w_0^{(k)} + \sum_j w_j^{(k)} x[j]\right)\right) \quad \text{1-hidden NN}$$



# Perceptron to Logistic Regression



# Deep Learning: Activations, FFN and more

# Choices for Non-Linear Activation Function

- **Sigmoid**

- Historically popular, but (mostly) fallen out of favor
- Neuron's activation saturates (weights get very large  $\rightarrow$  gradients get small)
- Not zero-centered  $\rightarrow$  other issues in the gradient steps
- When put on the output layer, called "softmax" because interpreted as class probability (soft assignment)

- **Hyperbolic tangent**  $g(x) = \tanh(x)$

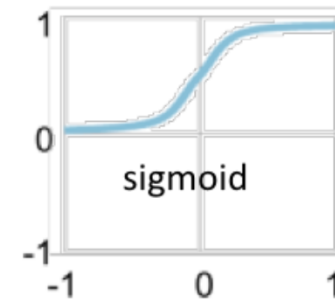
- Saturates like sigmoid unit, but zero-centered

- **Rectified linear unit (ReLU)**  $g(x) = x^+ = \max(0, x)$

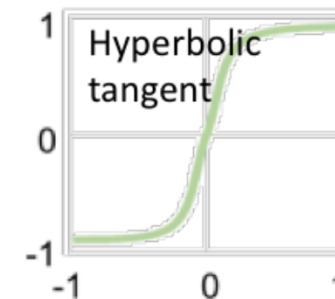
- Most popular choice these days
- Fragile during training and neurons can "die off"... be careful about learning rates
- "Noisy" or "leaky" variants

- **Softplus**  $g(x) = \log(1 + \exp(x))$

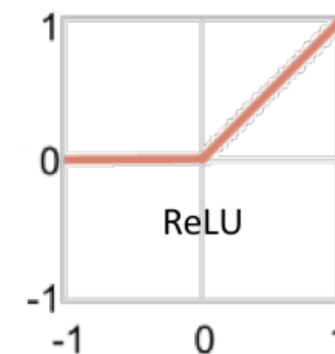
- Smooth approximation to rectifier activation



$\rightarrow$  output layers "only"

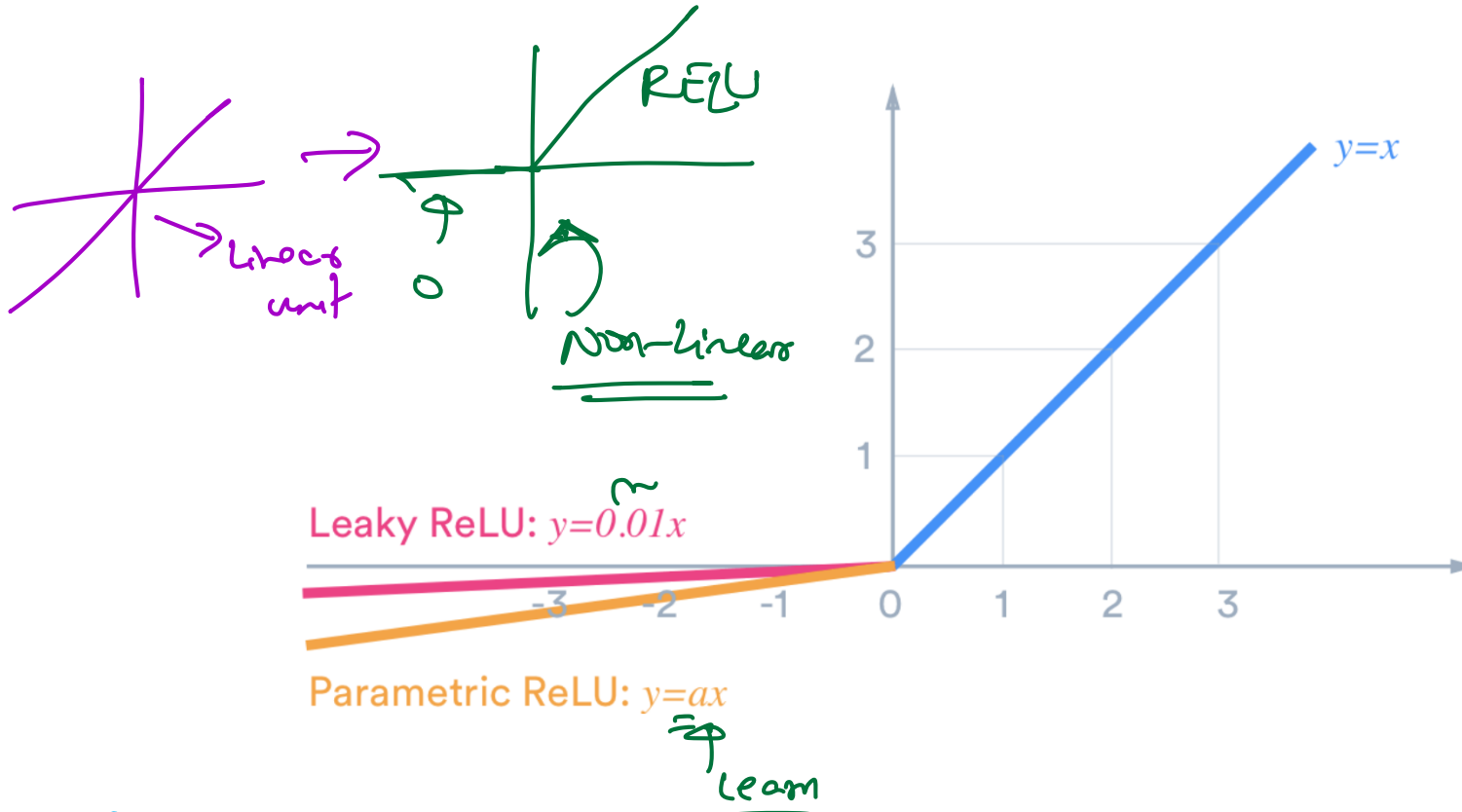


$\rightarrow$  Good choices for hidden layers



# RELU vs Leaky RELU

RELU - Rectified Linear unit



Tensorflow Playground Demo

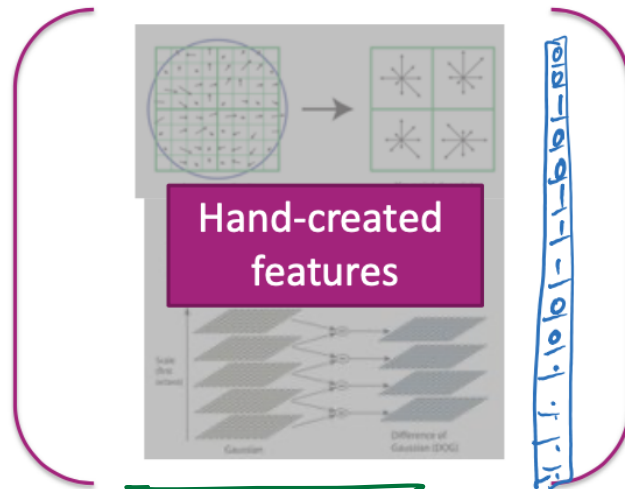
# Computer vision before deep learning

Deep Learning  $\xrightarrow{\text{take off}}$  - CV  
- NLP

Input



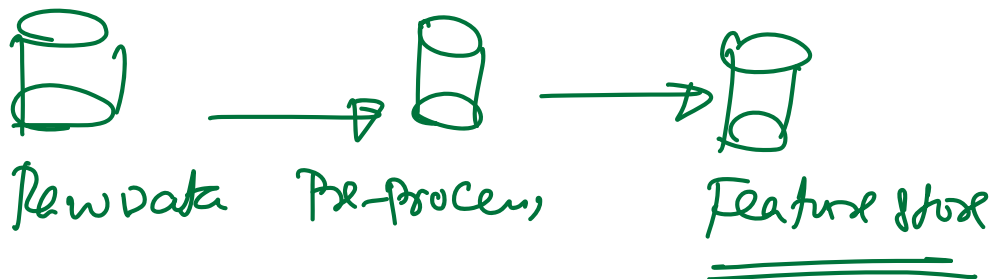
Extract features



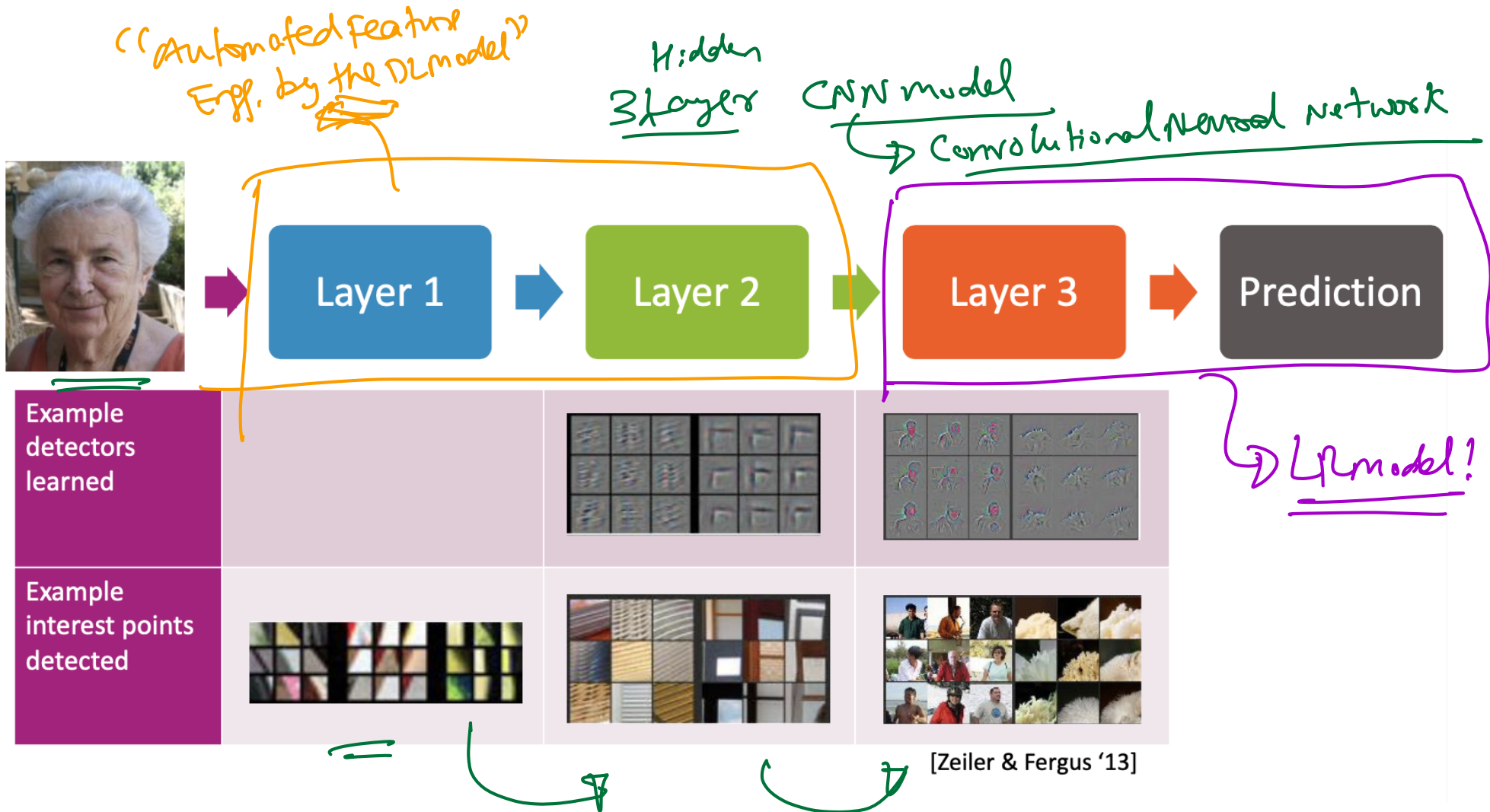
Use simple classifier

e.g., logistic regression, SVMs

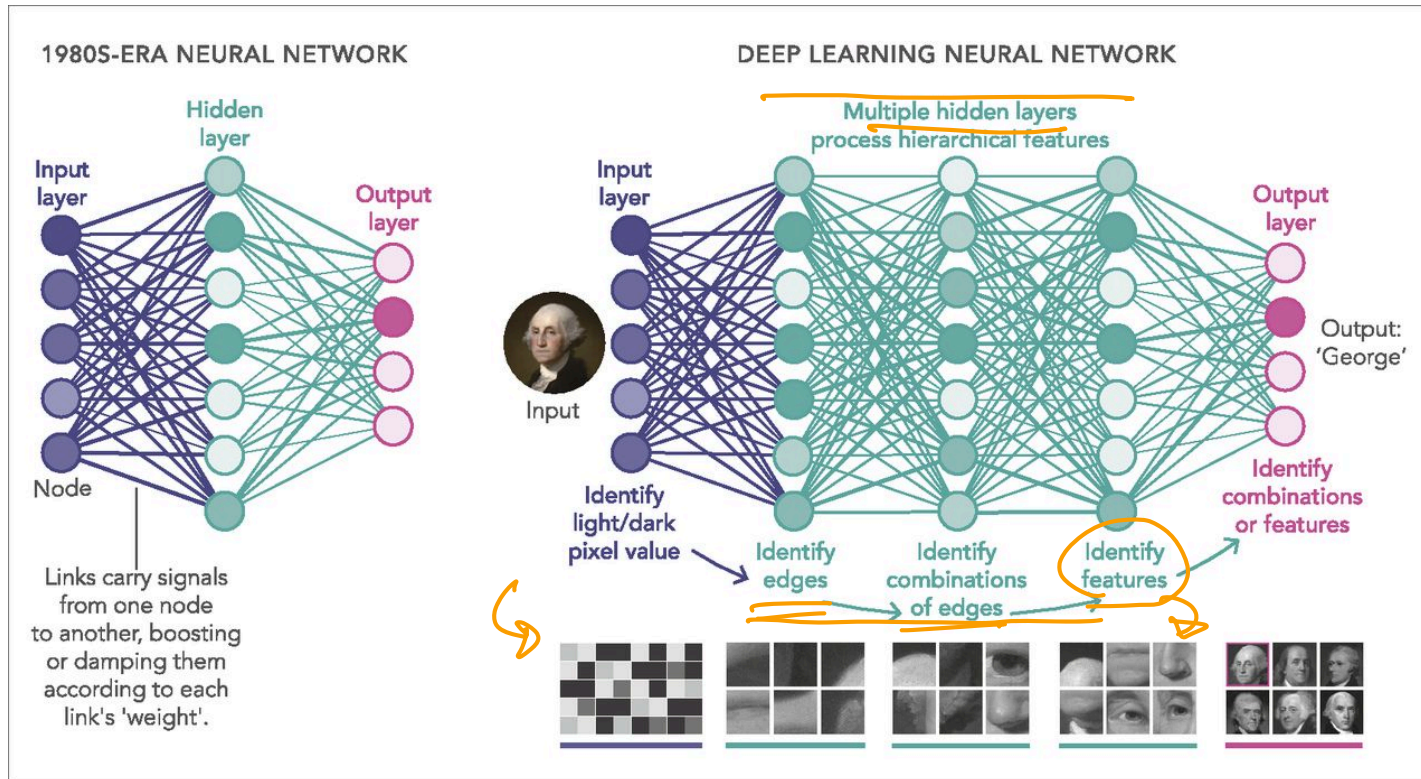
Face?



# Computer vision after deep learning



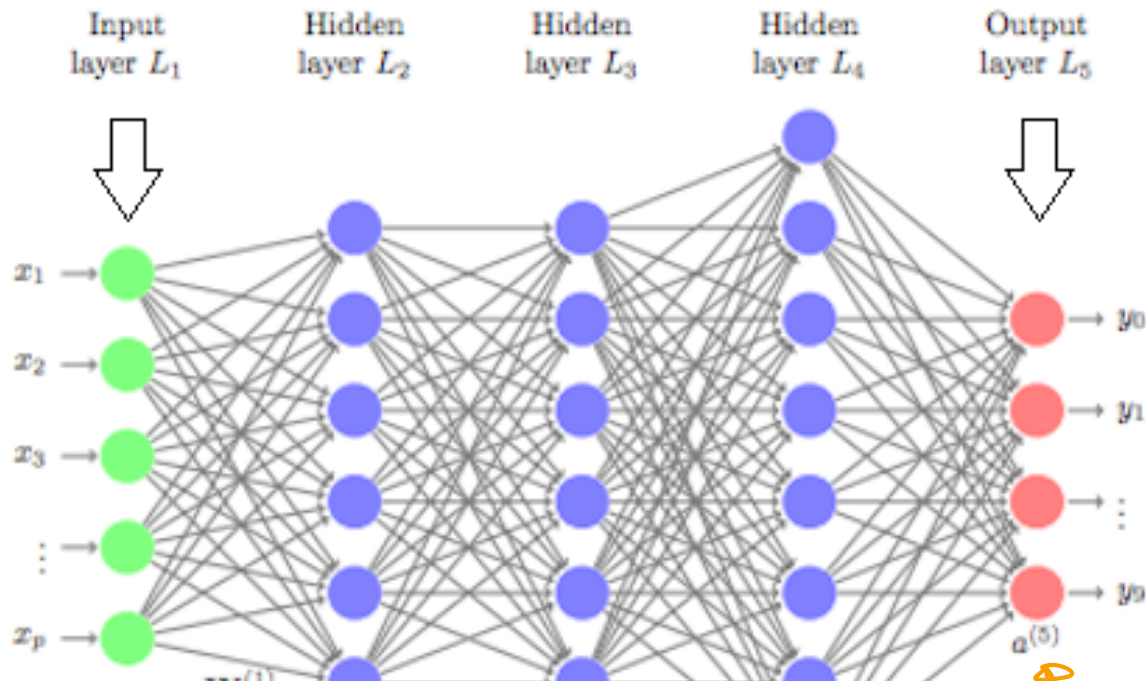
# Feed-forward Deep Learning Architecture Example



Coarse Grained → Fine-grained Features

Simple → Complex Features

# Feed-forward Deep Learning Architecture Example

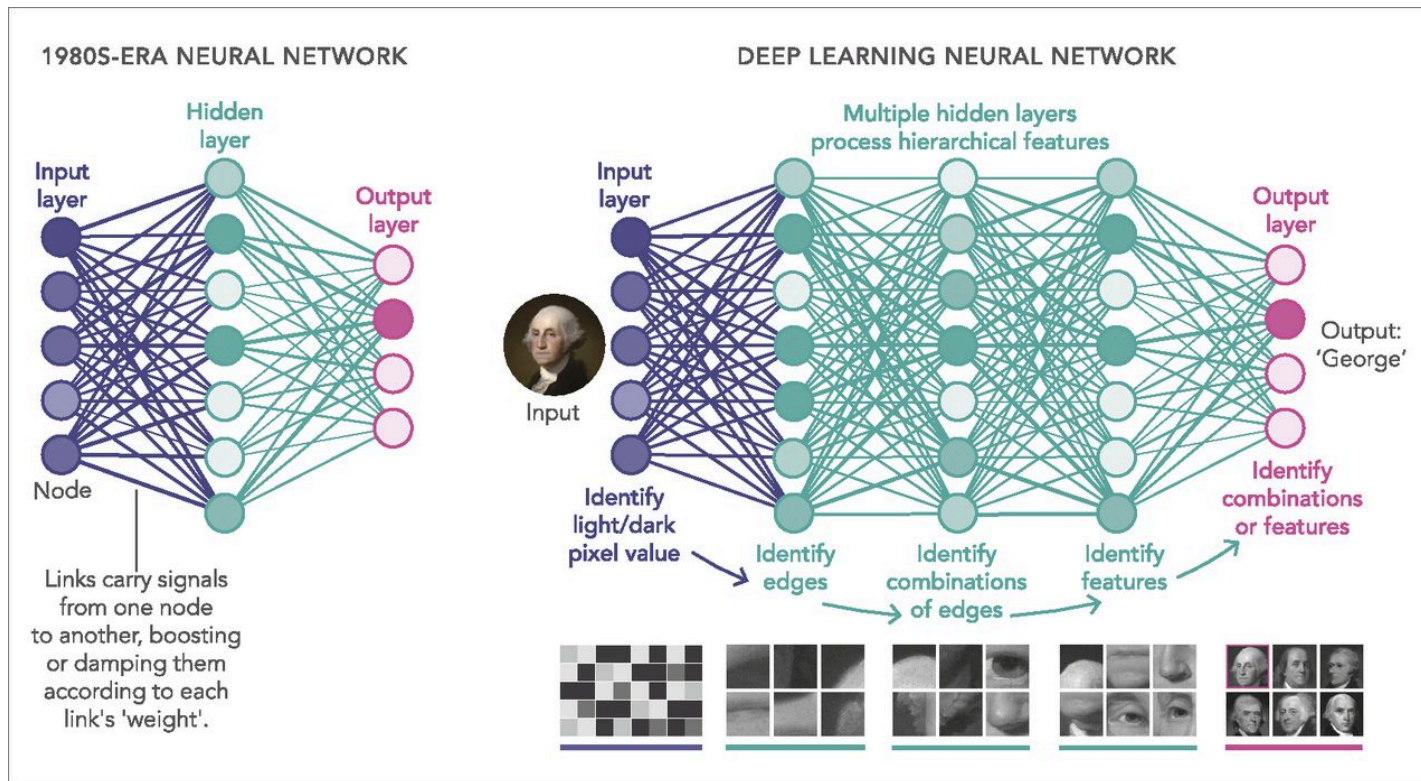


Handwritten notes in orange ink:
 

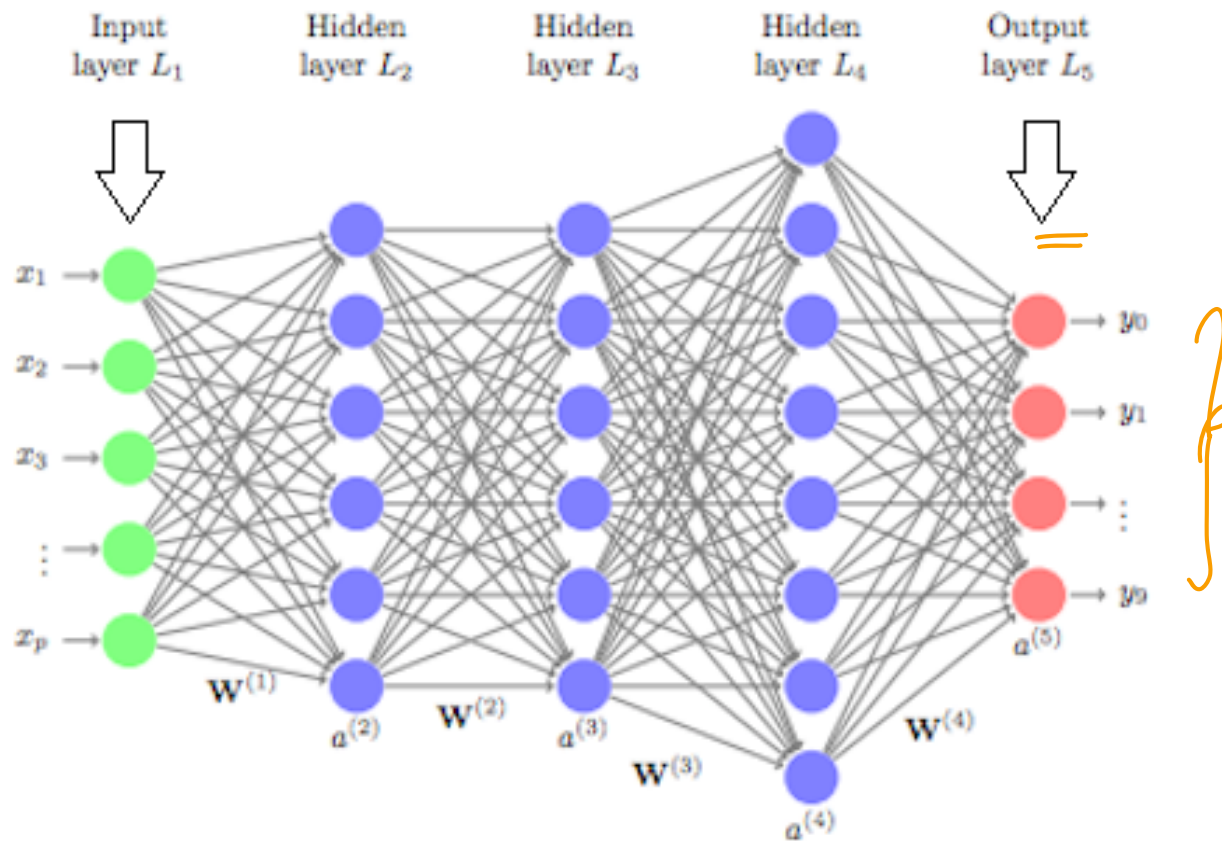
- $W^{(1)}$   $n_1 \times n_2$  #neurons in layer 1
- $D$  in layer 2
- Matrix
- Input
- $H^2$
- $H^2$
- $H^3$
- $K$  Hidden Layers and each layer has  $M$  neurons
- output



# Feed-forward Deep Learning Architecture Example



# Feed-forward Deep Learning Architecture Example

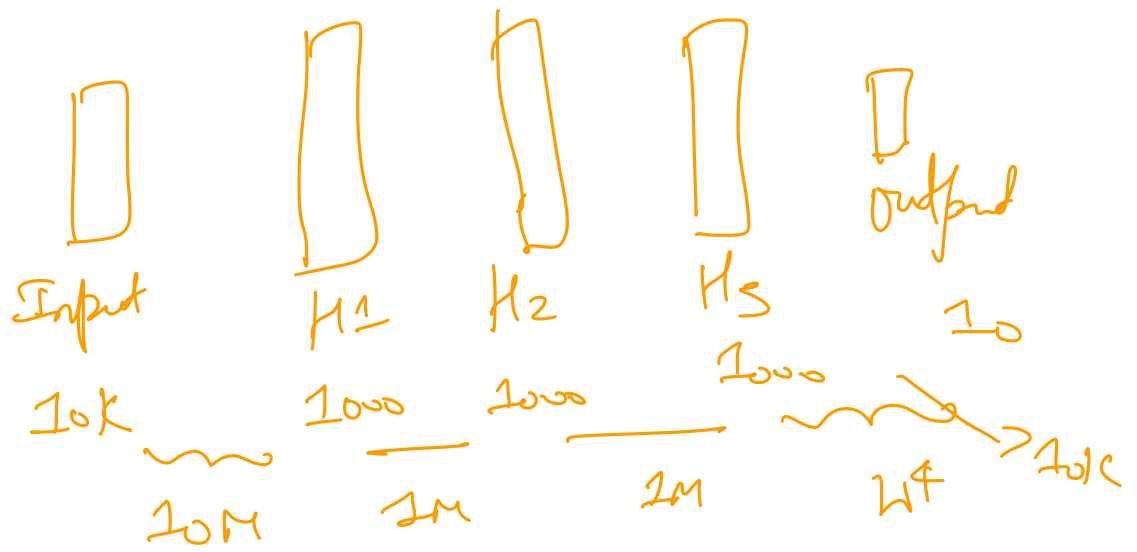


# ICE #1

Compute the number of parameters in DNN model

Consider a DNN model with 3 hidden layers where each hidden layer has 1000 neurons. Let the input layer be raw pixels from a 100x100 image and the output layer has 10 dimensions, let's say for a 10 class image classification example. How many total parameters exist in the DNN model?

- ① 10 million parameters
- ② 11 million parameters
- ③ 12 million parameters
- ④ 13 million parameters



# Training a DNN

## SGD with mini-batch

Same recipe works for DNN as we saw for logistic regression. SGD mini-batch is the staple diet. However there are some **learning rate schedulers** that are known to work better for DNNs - Such as Adagrad and more recently, ADAM. ADAM adapts the learning rate to each individual parameter instead of having a global learning rate.

# Training a DNN

## SGD with mini-batch

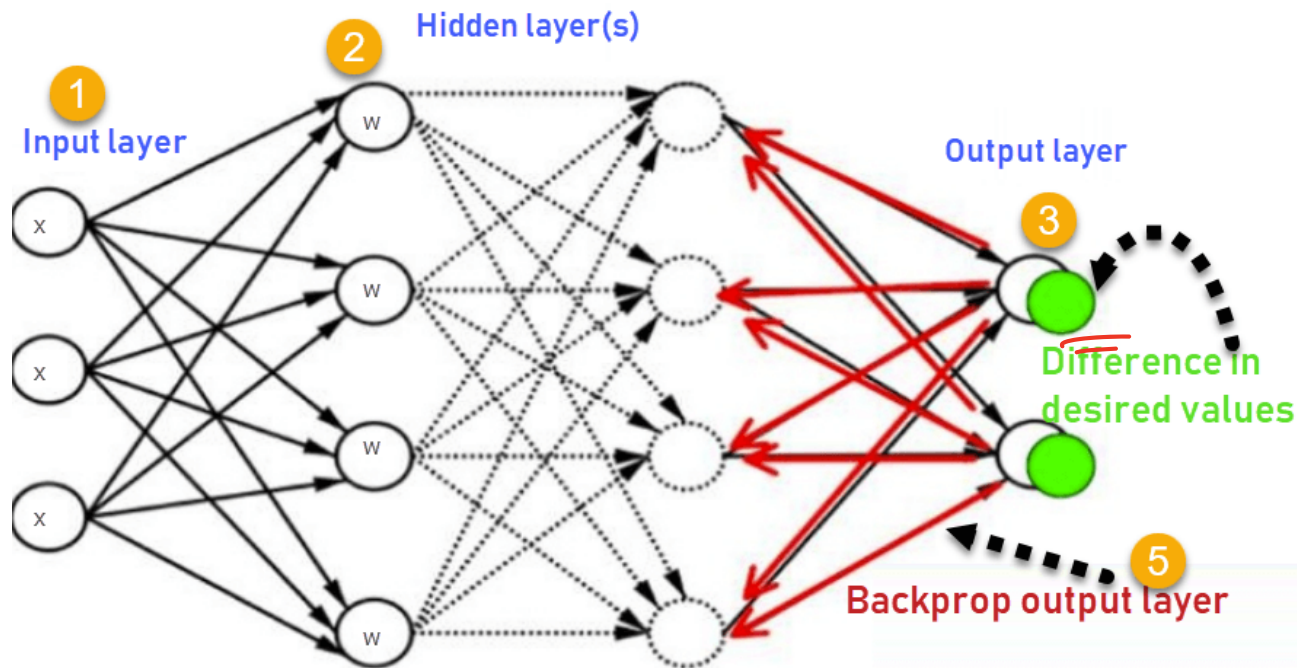
Same recipe works for DNN as we saw for logistic regression. SGD mini-batch is the staple diet. However there are some **learning rate schedulers** that are known to work better for DNNs - Such as Adagrad and more recently, ADAM. ADAM adapts the learning rate to each individual parameter instead of having a global learning rate.

## How do we compute gradient in a DNN?

Back-propagation!

# Forward Propagation vs Back-propagation

Back Prop: - used to compute gradients



Forward Prop: -

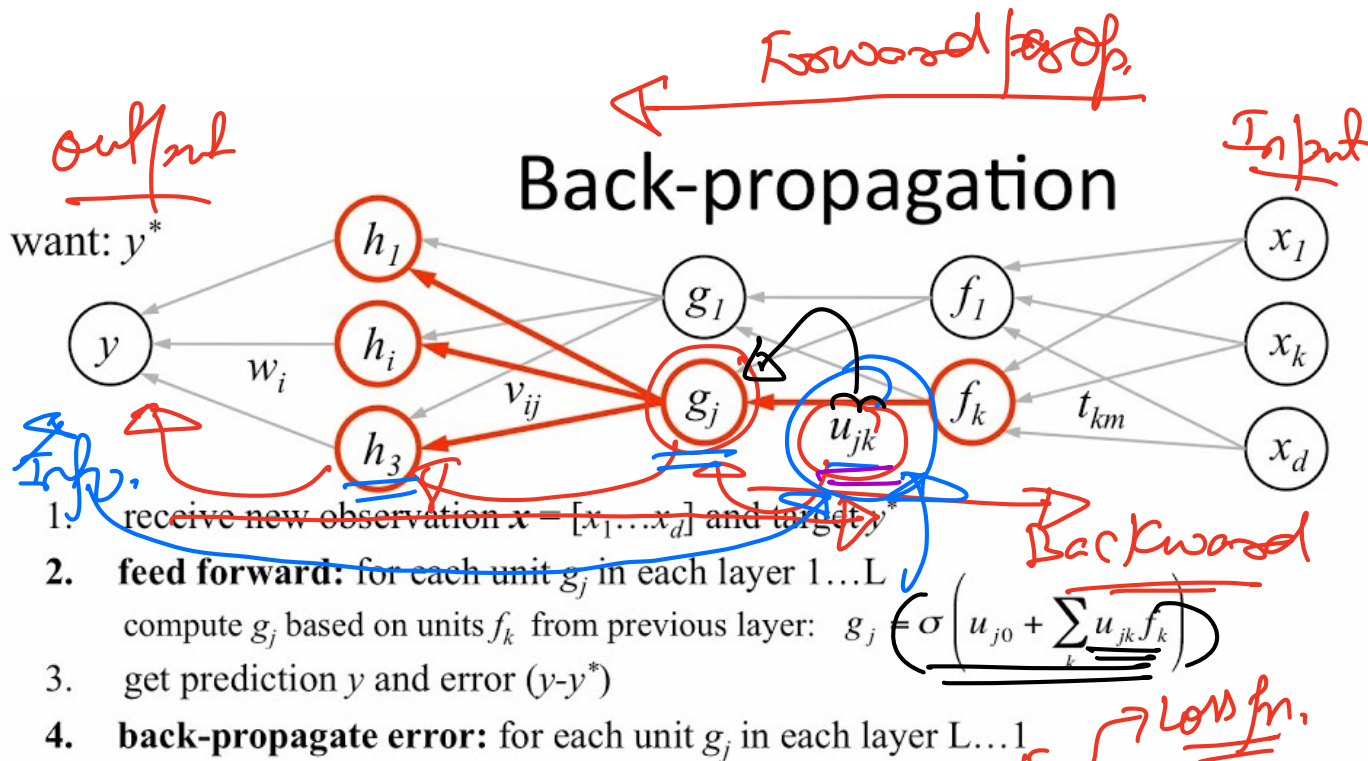
Input: Image

Forward



output: class with prob.

# Back Propagation explained



1. receive new observation  $x = [x_1 \dots x_d]$  and target  $y^*$
2. **feed forward:** for each unit  $g_j$  in each layer  $1 \dots L$  compute  $g_j$  based on units  $f_k$  from previous layer:  $g_j \in \sigma \left( u_{j0} + \sum_k u_{jk} f_k \right)$
3. get prediction  $y$  and error  $(y - y^*)$
4. **back-propagate error:** for each unit  $g_j$  in each layer  $L \dots 1$

(a) compute error on  $g_j$

$$\frac{\partial E}{\partial g_j} = \sum_i \sigma'(h_i) v_{ij} \frac{\partial E}{\partial h_i}$$

should  $g_j$  be higher or lower?    how  $h_i$  will change as  $g_j$  changes    was  $h_i$  too high or too low?

(b) for each  $u_{jk}$  that affects  $g_j$

(i) compute error on  $u_{jk}$

$$\frac{\partial E}{\partial u_{jk}} = \frac{\partial E}{\partial g_j} \sigma'(g_j) f_k$$

do we want  $g_j$  to be higher/lower?    how  $g_j$  will change if  $u_{jk}$  is higher/lower

(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

Learning

Copyright © 2014 Victor Lavrenko



$$g_j = \sigma(u_{j0} + \sum_k u_{jk} f_k)$$

$$\frac{\partial E}{\partial u_{jk}} =$$

$$\frac{\partial E}{\partial g_j} \cdot \frac{\partial g_j}{\partial u_{jk}}$$

Backprop

$$\frac{\partial g_j}{\partial u_{jk}} = \frac{\partial}{\partial u_{jk}} \sigma(u_{j0} + \sum_k u_{jk} f_k)$$

$$= \sigma'(u_{j0} + \sum_k u_{jk} f_k)$$

$$= \sigma'(u_{j0} + \sum_k u_{jk} f_k) f_k$$

# Back Propagation Summary

## Back Prop

Back prop is one of the fundamental backbones of the training modules behind deep learning and beyond (including for example ChatGPT). What exactly is back prop? It is just a way to unravel gradient computation in the neural network. Back prop is how we would **compute the gradient** in a neural network.

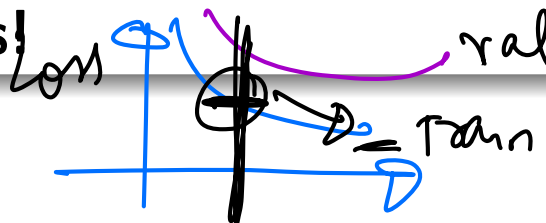
# Back Propagation Summary

## Back Prop

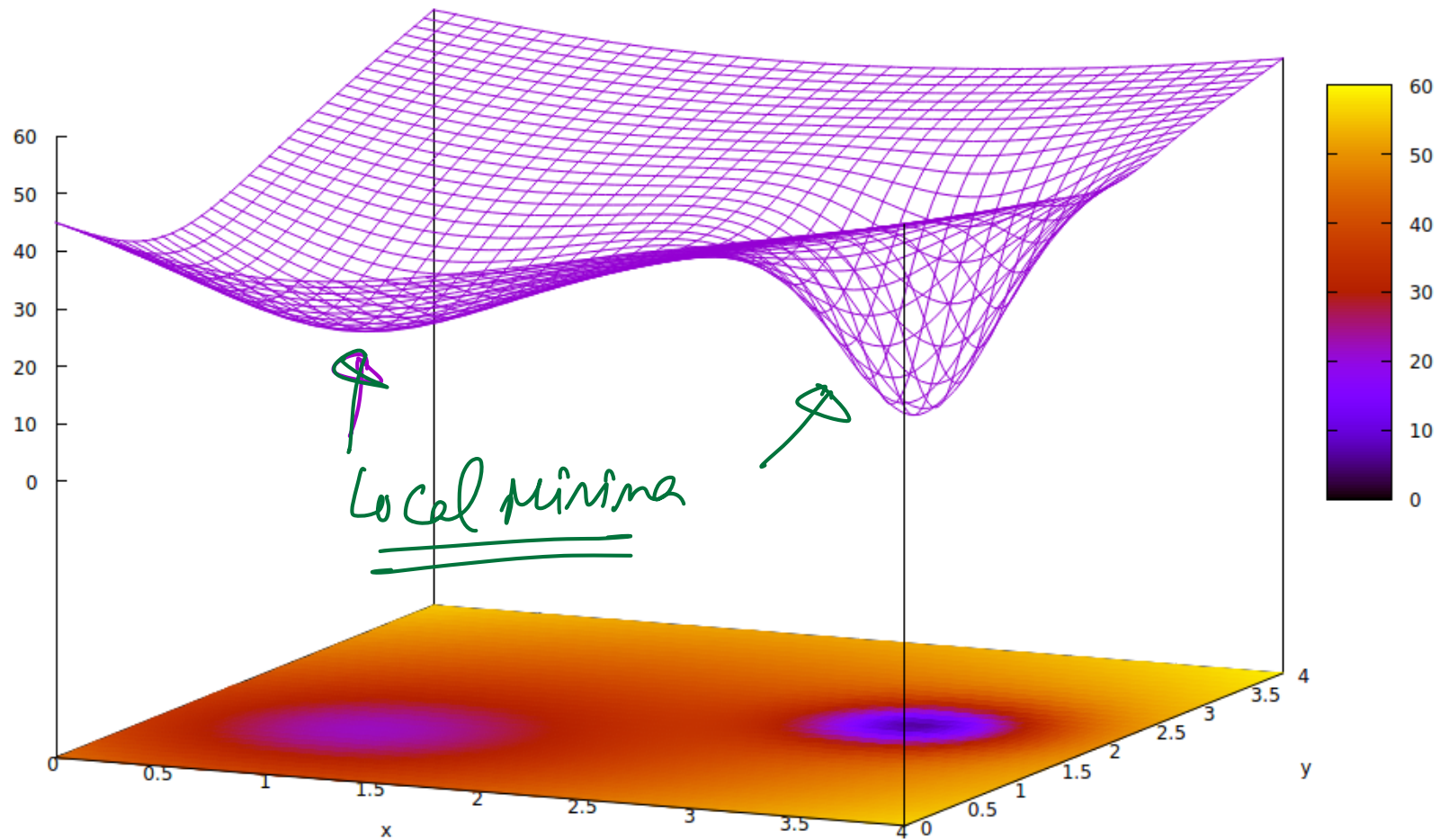
Back prop is one of the fundamental backbones of the training modules behind deep learning and beyond (including for example ChatGPT). What exactly is back prop? It is just a way to unravel gradient computation in the neural network. Back prop is how we would **compute the gradient** in a neural network.

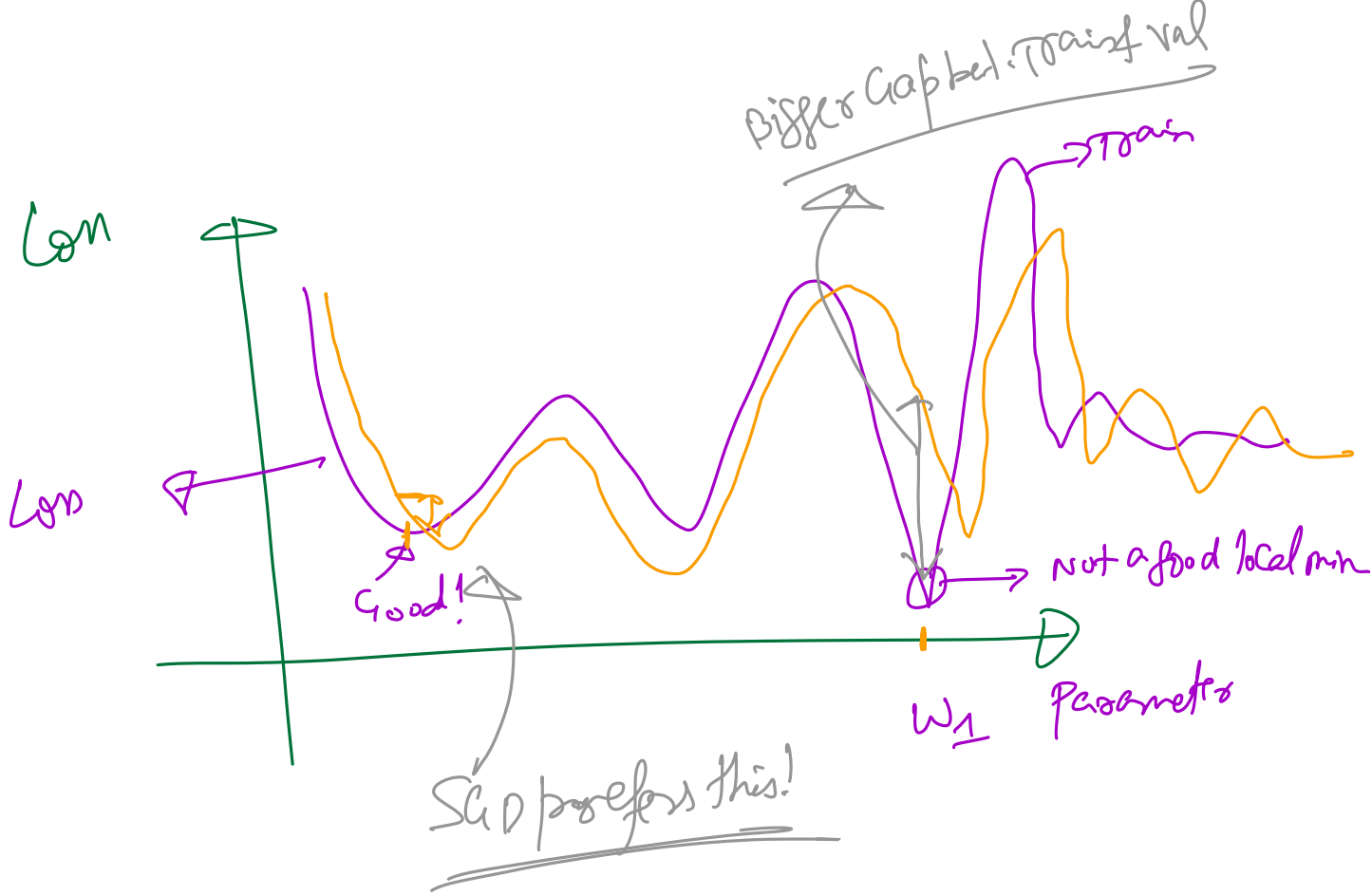
## Back Prop as information flow

It can also be thought of as flow information from the error in the output (the loss function) down to the weights. Update the weights so we don't make **this error** next time around. Back prop is a way to do **gradient descent in neural networks!**



# Good vs Bad Local minima





# Hyper-parameters in Deep Learning



ICE #2: Which of the following is not a hyper-parameter in deep learning?

- ① Learning rate
- ② Number of Hidden Layers
- ③ Number of neurons per hidden layer
- ④ All of the above

# Hyper-parameters in Deep Learning

## Hyper-parameters

- ① Learning rate
- ② Number of Hidden Layers
- ③ Number of neurons per hidden layer

# Hyper-parameters in Deep Learning

## Hyper-parameters

- ① Learning rate
- ② Number of Hidden Layers
- ③ Number of neurons per hidden layer
- ④ Type of non-linear activation function used



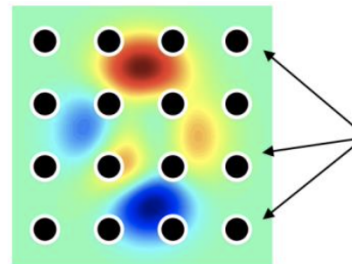
# Hyper-parameters in Deep Learning

## Hyper-parameters

- ① Learning rate
- ② Number of Hidden Layers
- ③ Number of neurons per hidden layer
- ④ Type of non-linear activation function used
- ⑤ Anything else?

# Hyper-parameter tuning methods

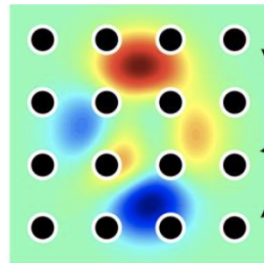
Grid search:



Hyperparameters  
on 2d uniform grid

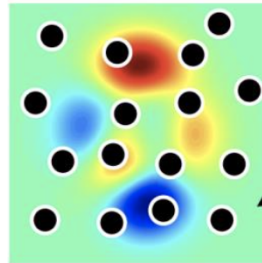
# Hyper-parameter tuning methods

Grid search:



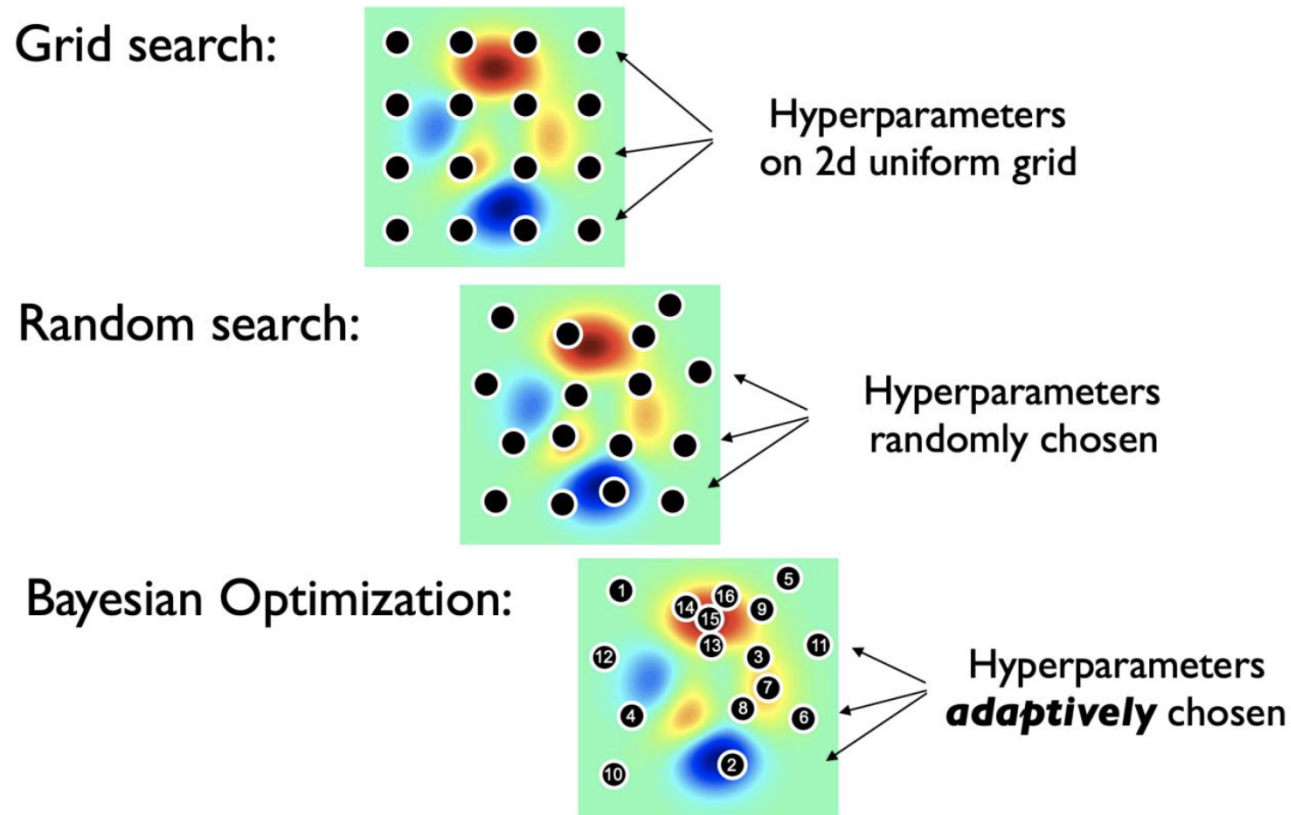
Hyperparameters  
on 2d uniform grid

Random search:



Hyperparameters  
randomly chosen

# Hyper-parameter tuning methods



# Over-fitting in DNNs

## How to handle over-fitting in DNNs

- 1 A DNN model with 100 million parameters and only 100k data points or even a million data points will overfit unless we take care of over-fitting.

# Over-fitting in DNNs

## How to handle over-fitting in DNNs

- ① A DNN model with 100 million parameters and only 100k data points or even a million data points will overfit unless we take care of over-fitting.
- ② Weight regularization can help -  $\ell_1, \ell_2$

# Over-fitting in DNNs

## How to handle over-fitting in DNNs

- ① A DNN model with 100 million parameters and only 100k data points or even a million data points will overfit unless we take care of over-fitting.
- ② Weight regularization can help -  $\ell_1, \ell_2$
- ③ More common over-fitting strategy for DL?

# Over-fitting in DNNs

## How to handle over-fitting in DNNs

- ① A DNN model with 100 million parameters and only 100k data points or even a million data points will overfit unless we take care of over-fitting.
- ② Weight regularization can help -  $\ell_1, \ell_2$
- ③ More common over-fitting strategy for DL?
- ④ Dropouts!



# Over-fitting in DNNs

## How to handle over-fitting in DNNs

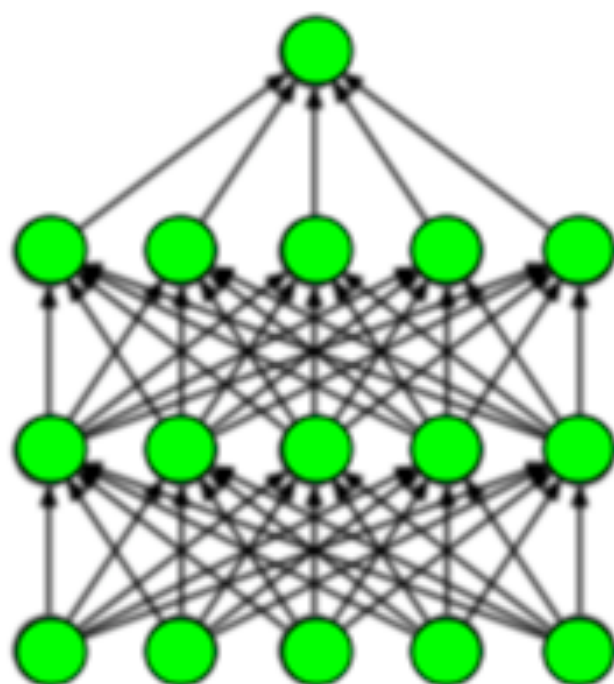
- ① A DNN model with 100 million parameters and only 100k data points or even a million data points will overfit unless we take care of over-fitting.
- ② Weight regularization can help -  $\ell_1, \ell_2$
- ③ More common over-fitting strategy for DL?
- ④ Dropouts!
- ⑤ Early stopping is also a great strategy! Stop training the DL model when the validation error starts increasing. How's this different from regular validation we were doing earlier??

# Over-fitting in DNNs

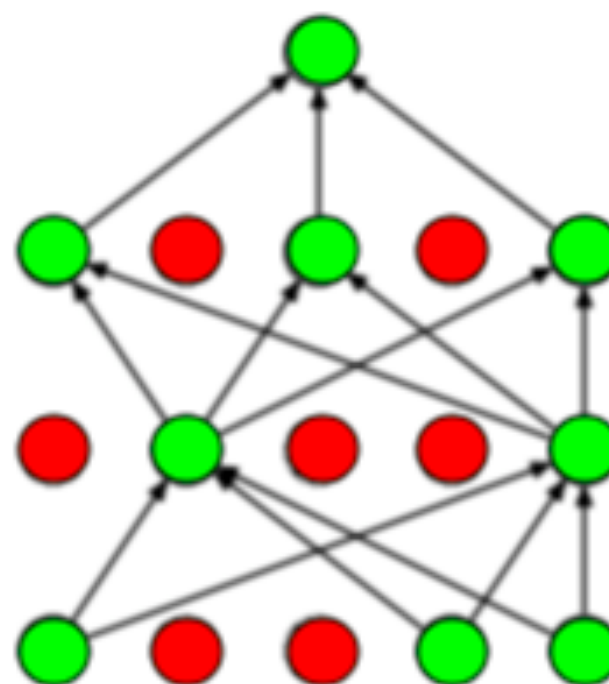
## How to handle over-fitting in DNNs

- 1 A DNN model with 100 million parameters and only 100k data points or even a million data points will overfit unless we take care of over-fitting.
- 2 Weight regularization can help -  $\ell_1, \ell_2$
- 3 More common over-fitting strategy for DL?
- 4 Dropouts!
- 5 Early stopping is also a great strategy! Stop training the DL model when the validation error starts increasing. How's this different from regular validation we were doing earlier??
- 6 Book by Yoshua Bengio has tons of details and great reference for Deep Learning!

# Taking care of Over-fitting: Dropouts



(a) Standard Neural Net



(b) After applying dropout.

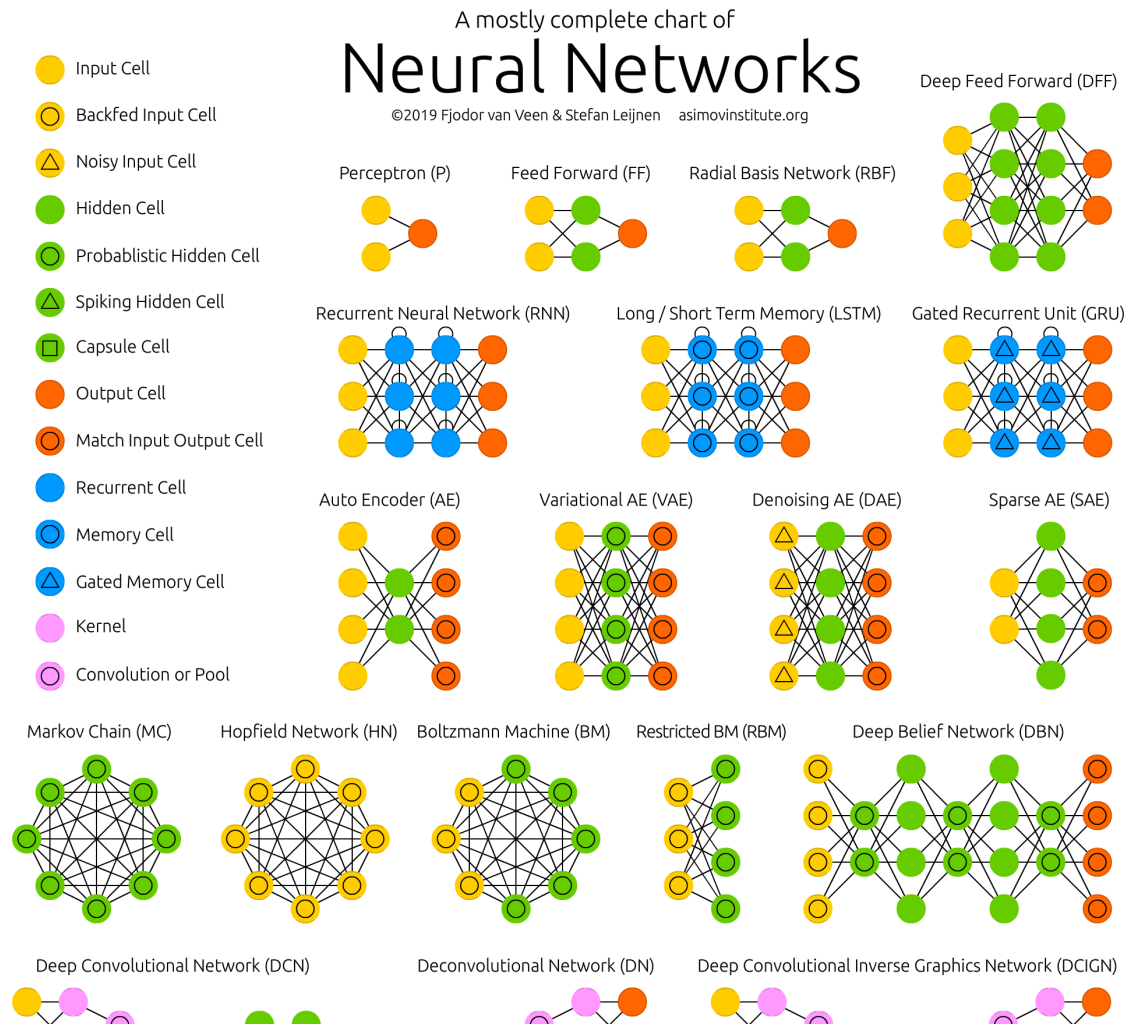
# Tensorflow Playground Demo

Tensorflow Playground Demo

# More DL Architectures

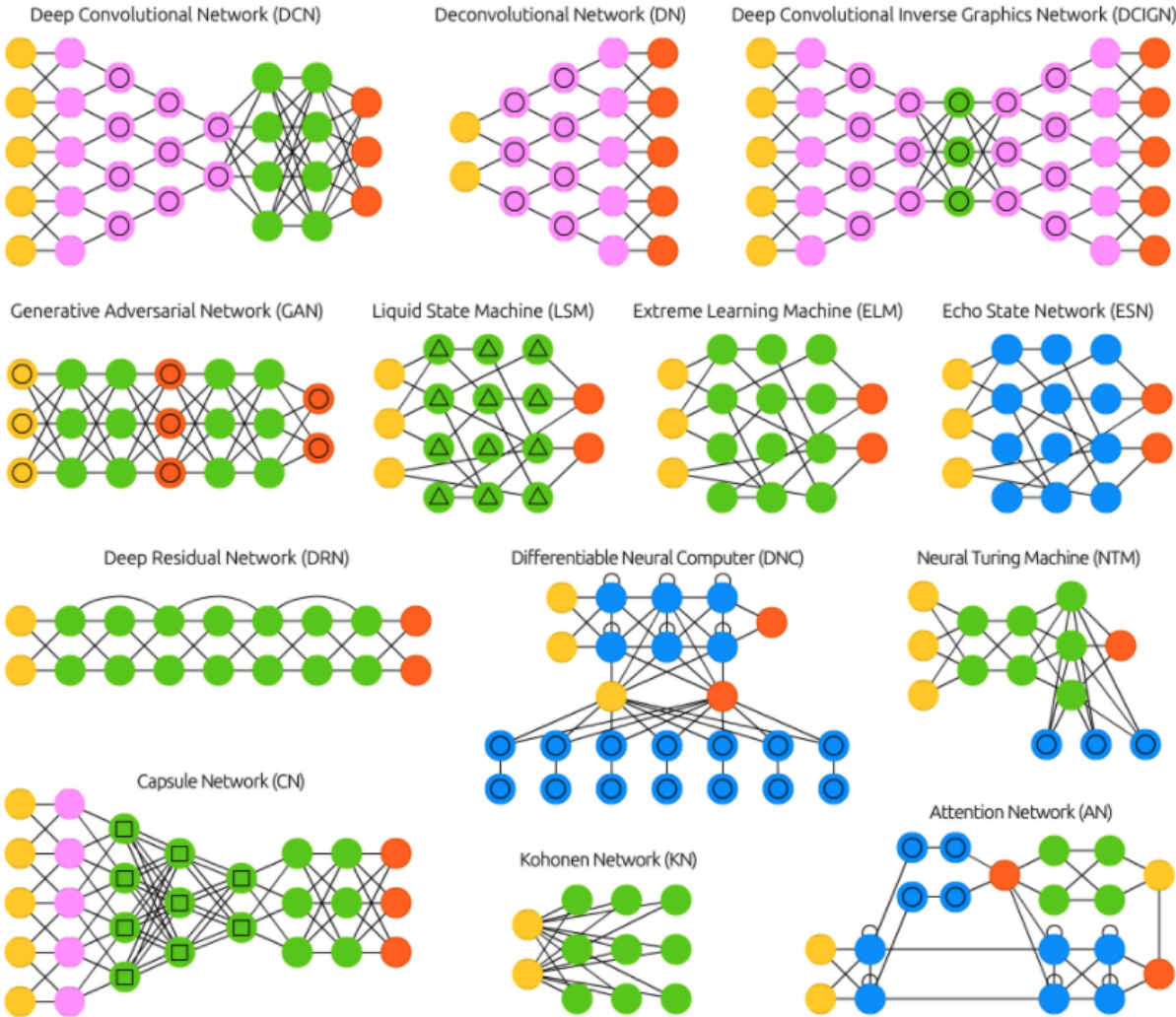
## Neural Networks Zoo

### Zoo Reference

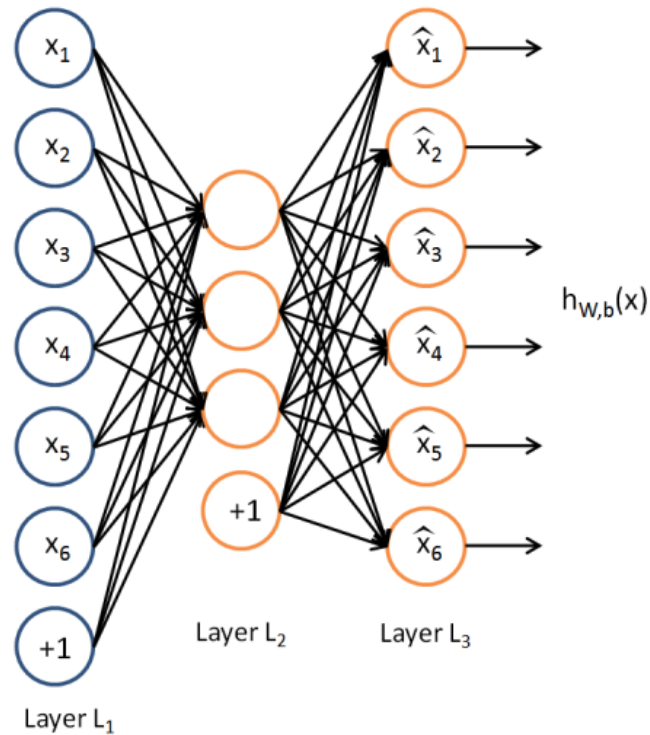


# More DL Architectures

## Neural Networks Zoo



# Auto Encoders



# ICE #3

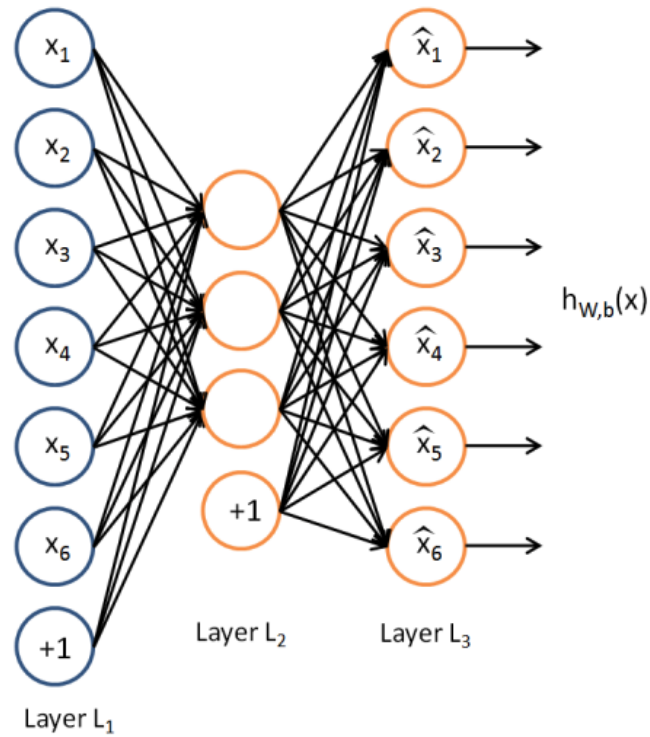
## PCA vs Auto Encoder

Which of the following statements are true ?

- 1 Both PCA and Auto Encoders serve the purpose of dimensionality reduction
- 2 They are both linear models but one uses a neural nets architecture and the other is based on projections
- 3 PCA is robust to outliers while Auto Encoders are not
- 4 Auto Encoders are as better than Glove Embeddings to find low-dim embeddings for words



# PCA vs Auto-Encoders



# AutoEncoders and Dimensionality Reduction

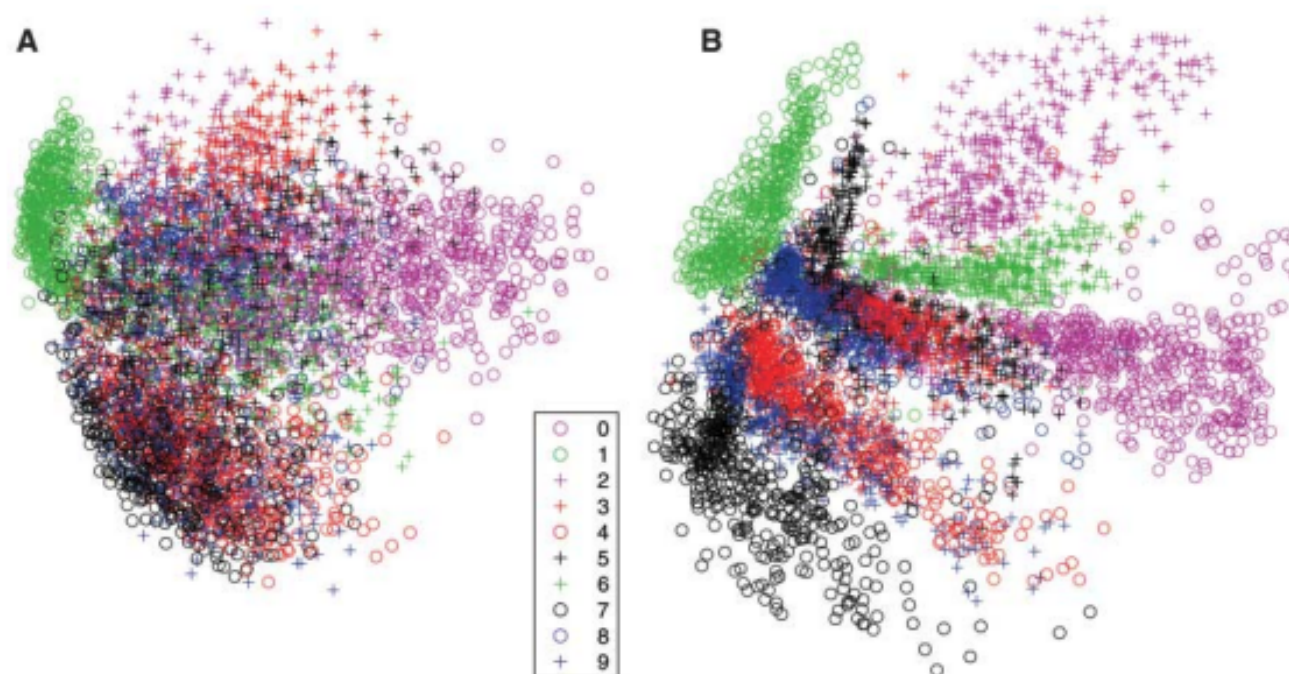
Visualization Performance

Auto Encoder Reference Paper

# AutoEncoders and Dimensionality Reduction

## Reading Reference for AE Dimensionality Reduction

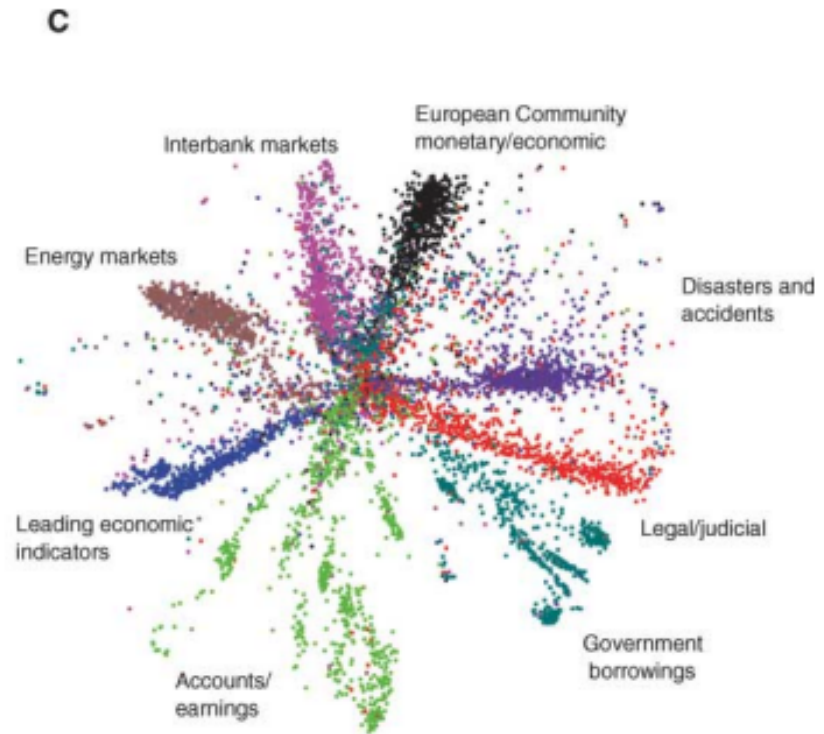
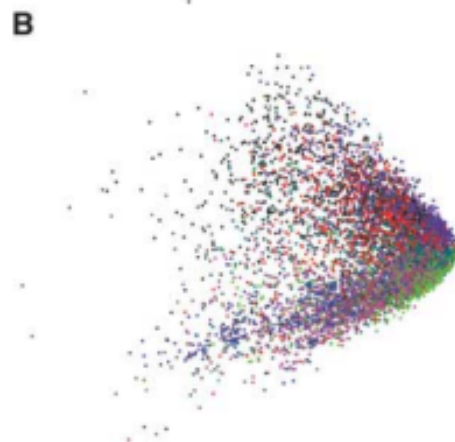
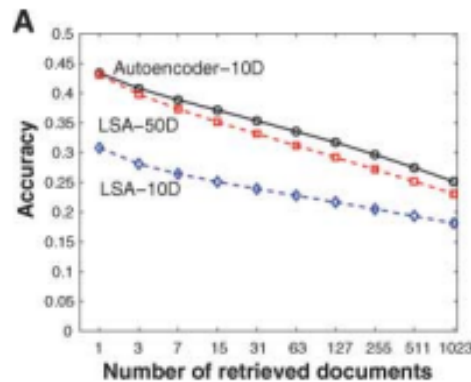
**Fig. 3.** (A) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (B) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (8).



# AutoEncoders and Dimensionality Reduction

## Reading Reference for AE Dimensionality Reduction

**Fig. 4.** (A) The fraction of retrieved documents in the same class as the query when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. (B) The codes produced by two-dimensional LSA. (C) The codes produced by a 2000-500-250-125-2 autoencoder.



# AutoEncoders Summary

- ① Auto-Encoders are a method for dimensionality reduction and can do better than PCA for visualization

# AutoEncoders Summary

- ① Auto-Encoders are a method for dimensionality reduction and can do better than PCA for visualization
- ② Use Neural Networks architecture and hence can encode non-linearity in the embeddings

# AutoEncoders Summary

- ① Auto-Encoders are a method for dimensionality reduction and can do better than PCA for visualization
- ② Use Neural Networks architecture and hence can encode non-linearity in the embeddings
- ③ Anything else?

# AutoEncoders Summary

- ① Auto-Encoders are a method for dimensionality reduction and can do better than PCA for visualization
- ② Use Neural Networks architecture and hence can encode non-linearity in the embeddings
- ③ Anything else?
- ④ Auto Encoders can learn convolutional layers instead of dense layers - Better for images! More flexibility!!



# Removing obstacles in images

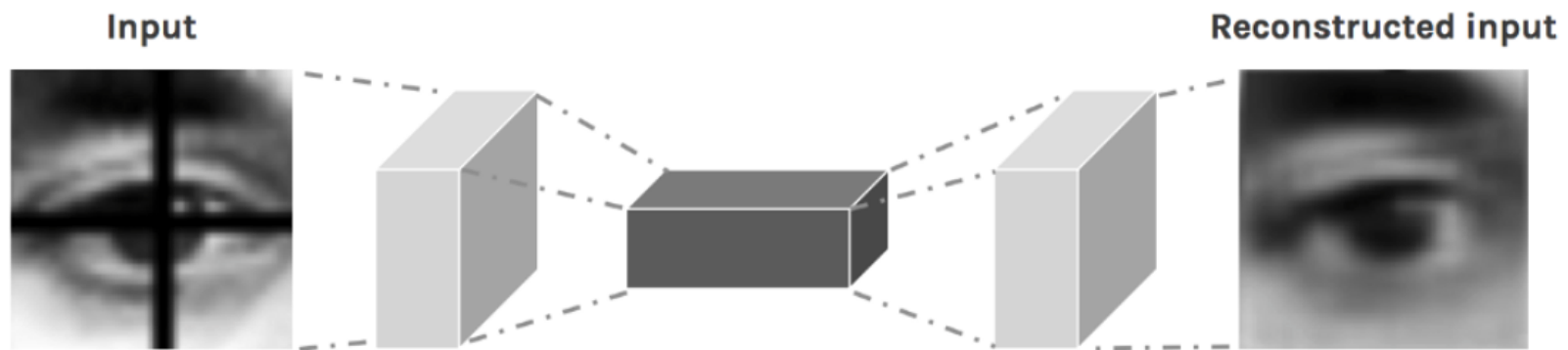


Figure 12: Reconstructed image from missing image [14]

# Removing obstacles in images

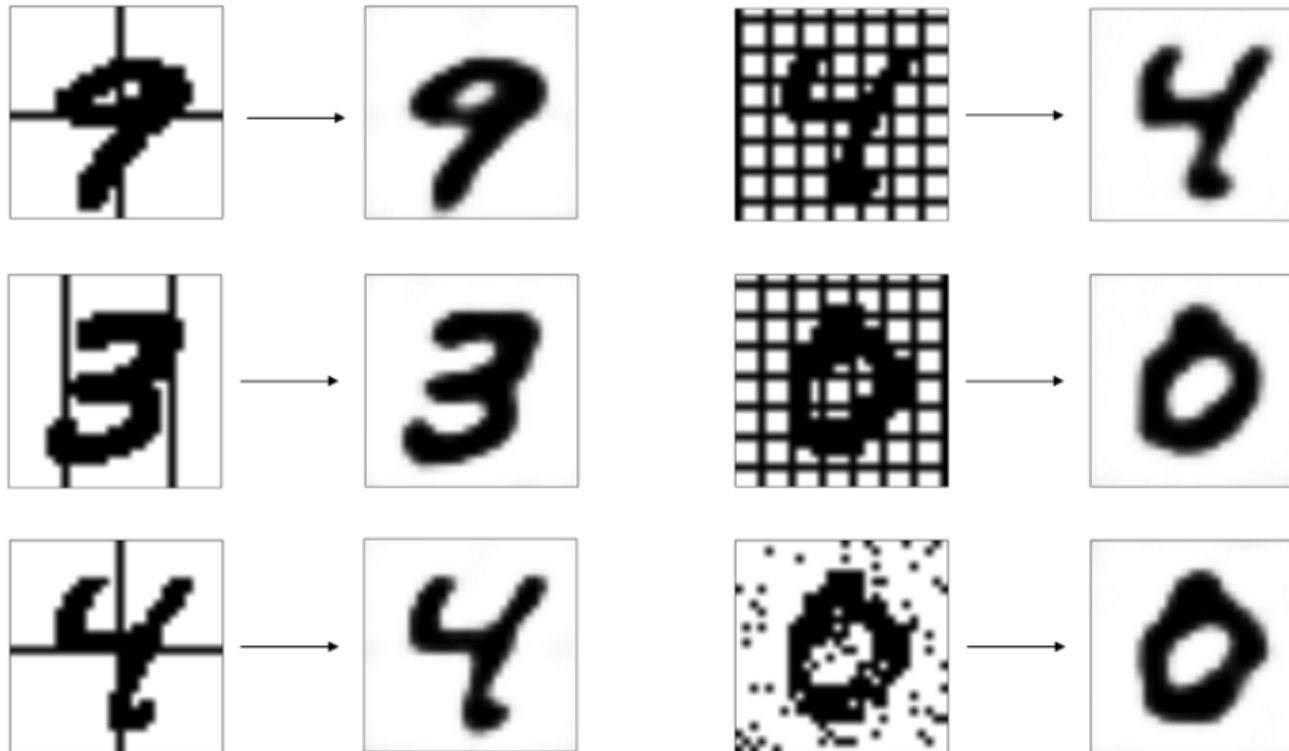


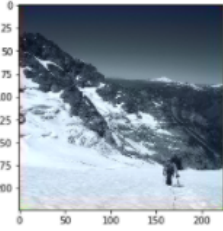
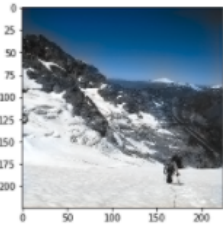



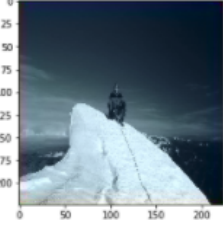
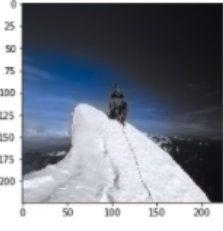



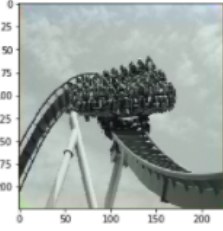
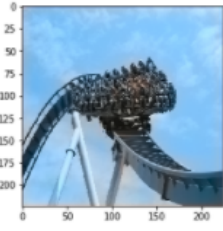

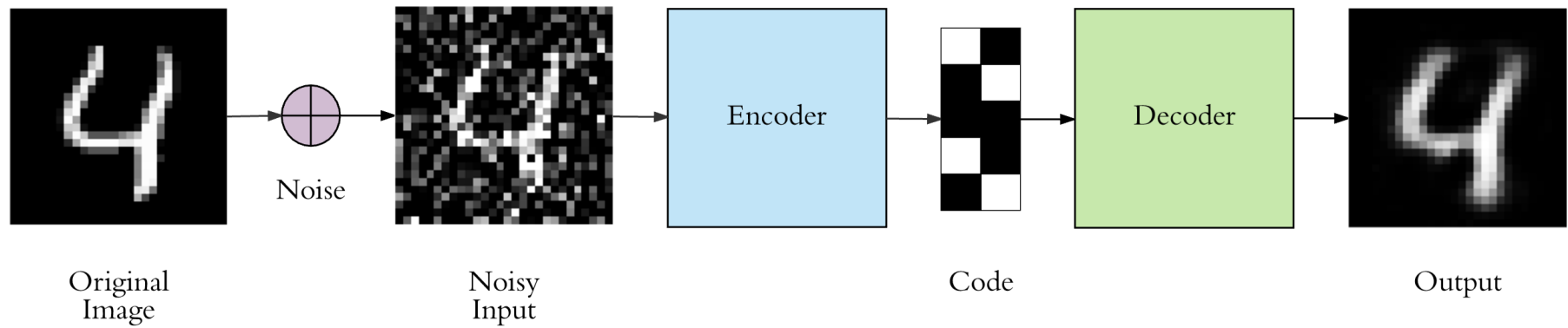


Figure 13: Source [15]

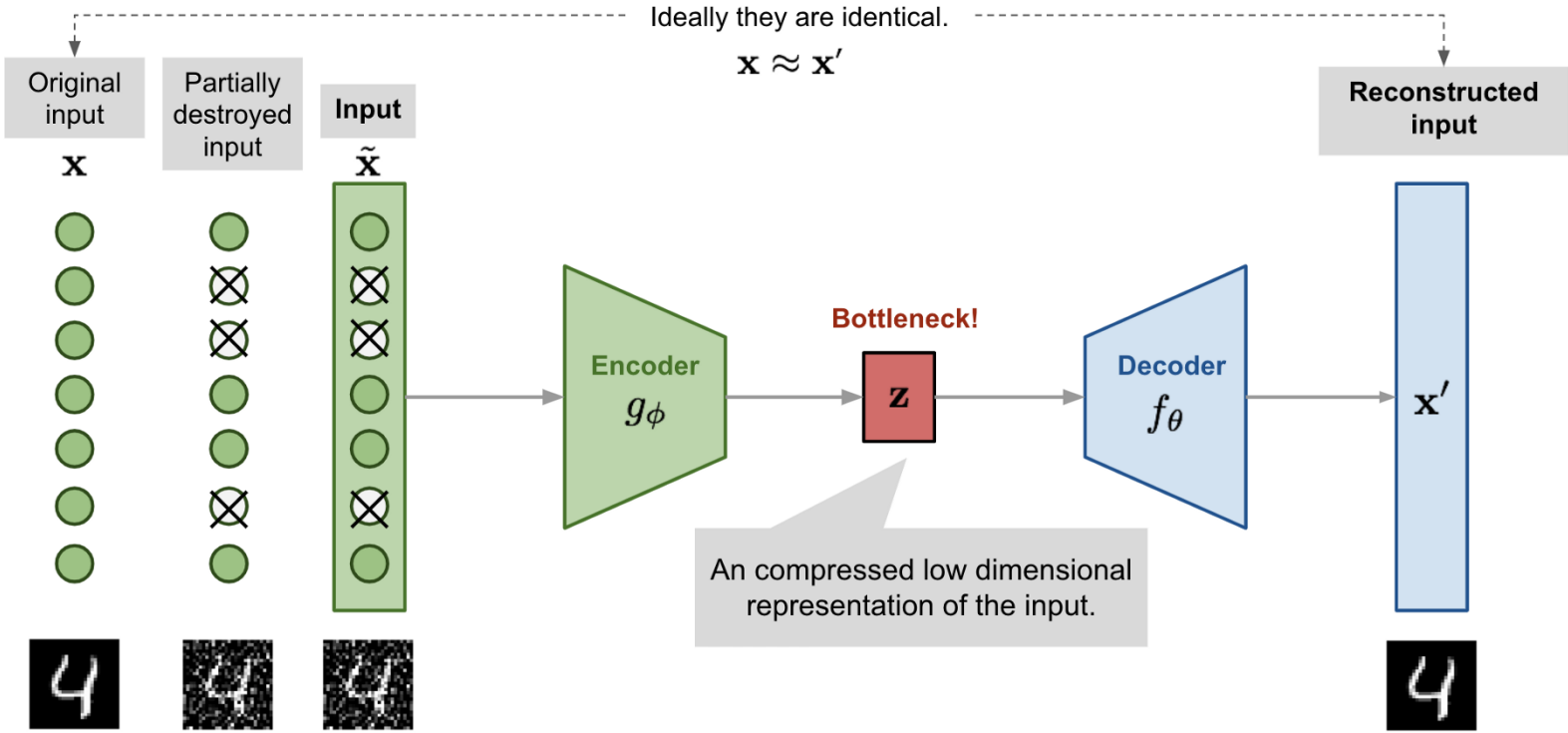
# Coloring Images

Gray Image	Vanilla Autoencoder	Merge Model (YCbCr)	Merge Model (LAB)	Original
				
				
				

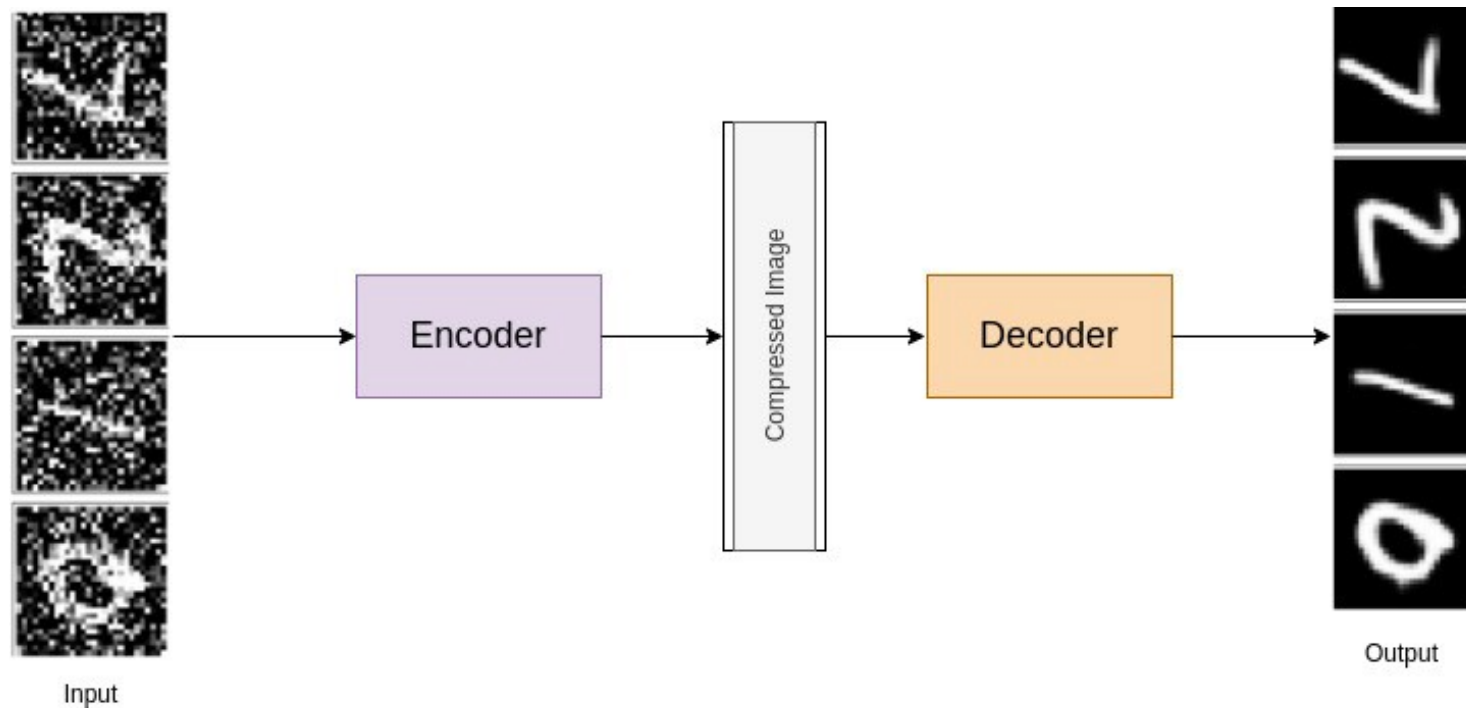
# De-noising Auto Encoders



# De-noising Auto Encoders



# De-noising Auto Encoders



# De-noising Auto Encoders

## Details

- Just like an Auto Encoder

# De-noising Auto Encoders

## Details

- Just like an Auto Encoder
- Difference: Noise is injected in the inputs on purpose but output is a clean data point.



# De-noising Auto Encoders

## Details

- Just like an Auto Encoder
- Difference: Noise is injected in the inputs on purpose but output is a clean data point.
- This forces the Auto Encoder to “de-noise” data, esp. useful for images!

# De-noising Auto Encoders

## Details

- Just like an Auto Encoder
- Difference: Noise is injected in the inputs on purpose but output is a clean data point.
- This forces the Auto Encoder to “de-noise” data, esp. useful for images!
- Esp. useful for a category of objects or images (e.g. digit recognition or face recognition, etc)

# ICE #4

## Unsupervised Learning

Which of these is NOT an example of unsupervised learning?

- ① Perceptron
- ② Auto Encoder
- ③ De-noising Auto Encoder
- ④ K-means++
- ⑤ None of the above
- ⑥ All of the above

# Breakouts Time 1

5 mins

Discuss in your groups what are some real-world applications of any or many of the Auto Encoder Architectures we discussed so far you can think of in your area of work or in a standard context e.g. images.

# Sequence structure in NLP

## Example

I love this car! Positive Sentiment

# Sequence structure in NLP

## Example

I love this car! Positive Sentiment

## Example

I am not sure I love this car! Negative Sentiment

# Sequence structure in NLP

## Example

I love this car! Positive Sentiment

## Example

I am not sure I love this car! Negative Sentiment

## Example

I don't think its a bad car at all! → Positive Sentiment

# Sequence structure in NLP

## Example

I love this car! Positive Sentiment

## Example

I am not sure I love this car! Negative Sentiment

## Example

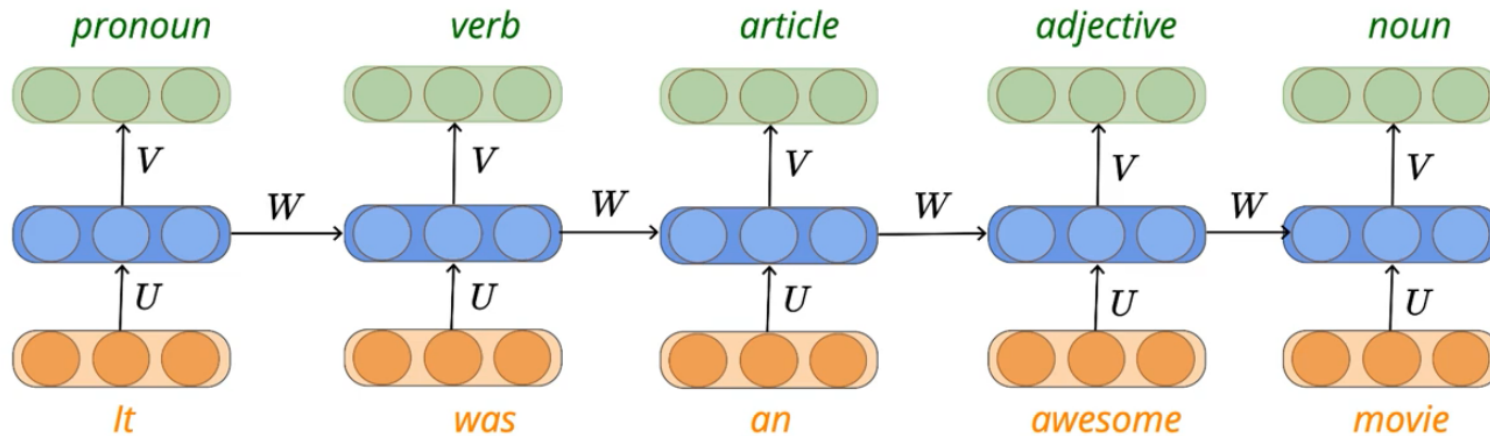
I don't think its a bad car at all! → Positive Sentiment

## Example

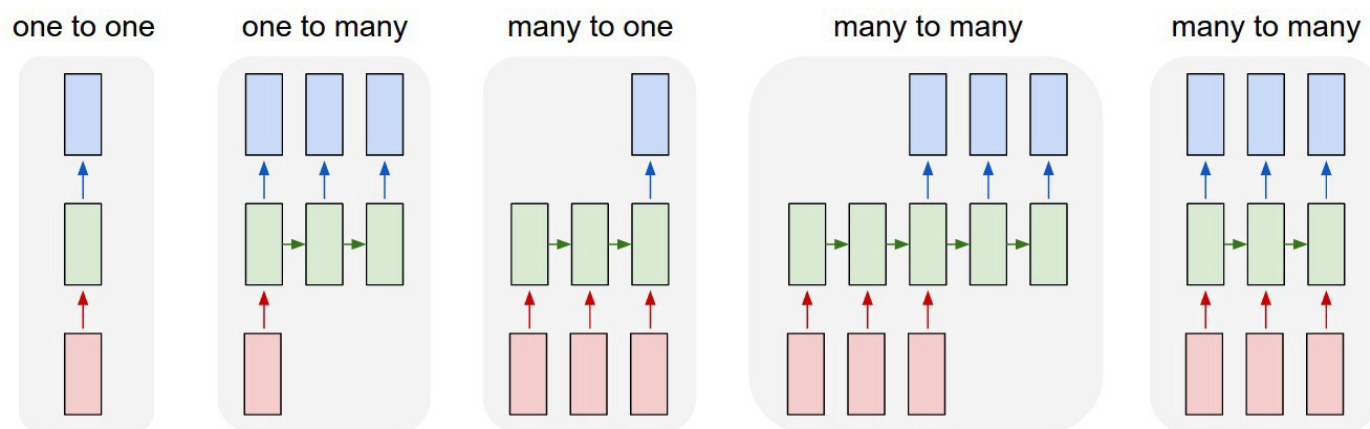
Have to carry the **context(state)** from some-time back to fully understand what's happening!



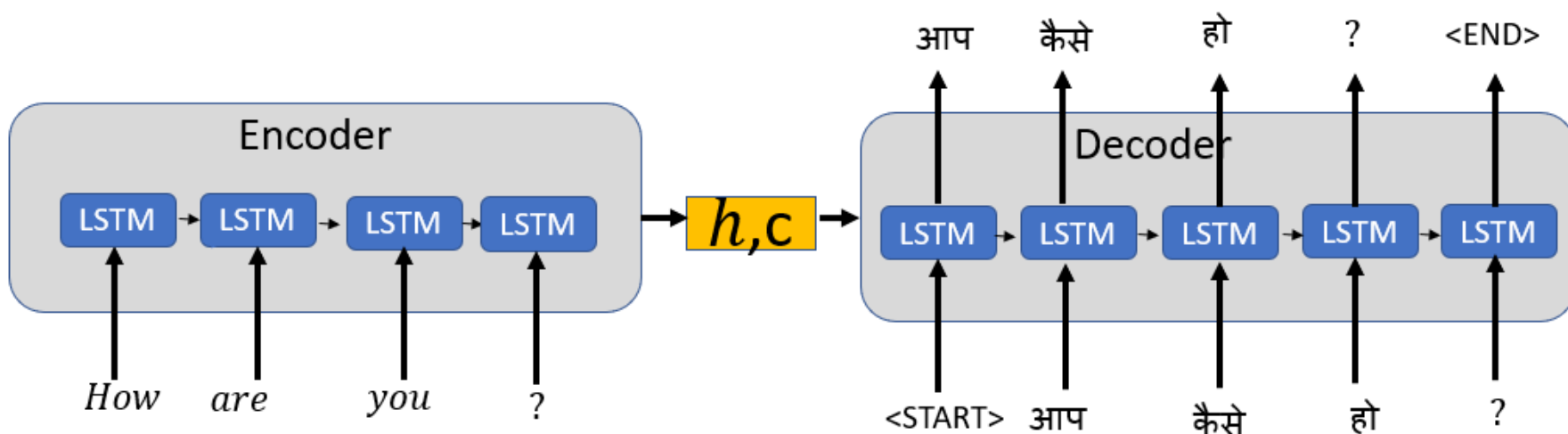
# Sequence to Sequence Model (LSTM) Applications



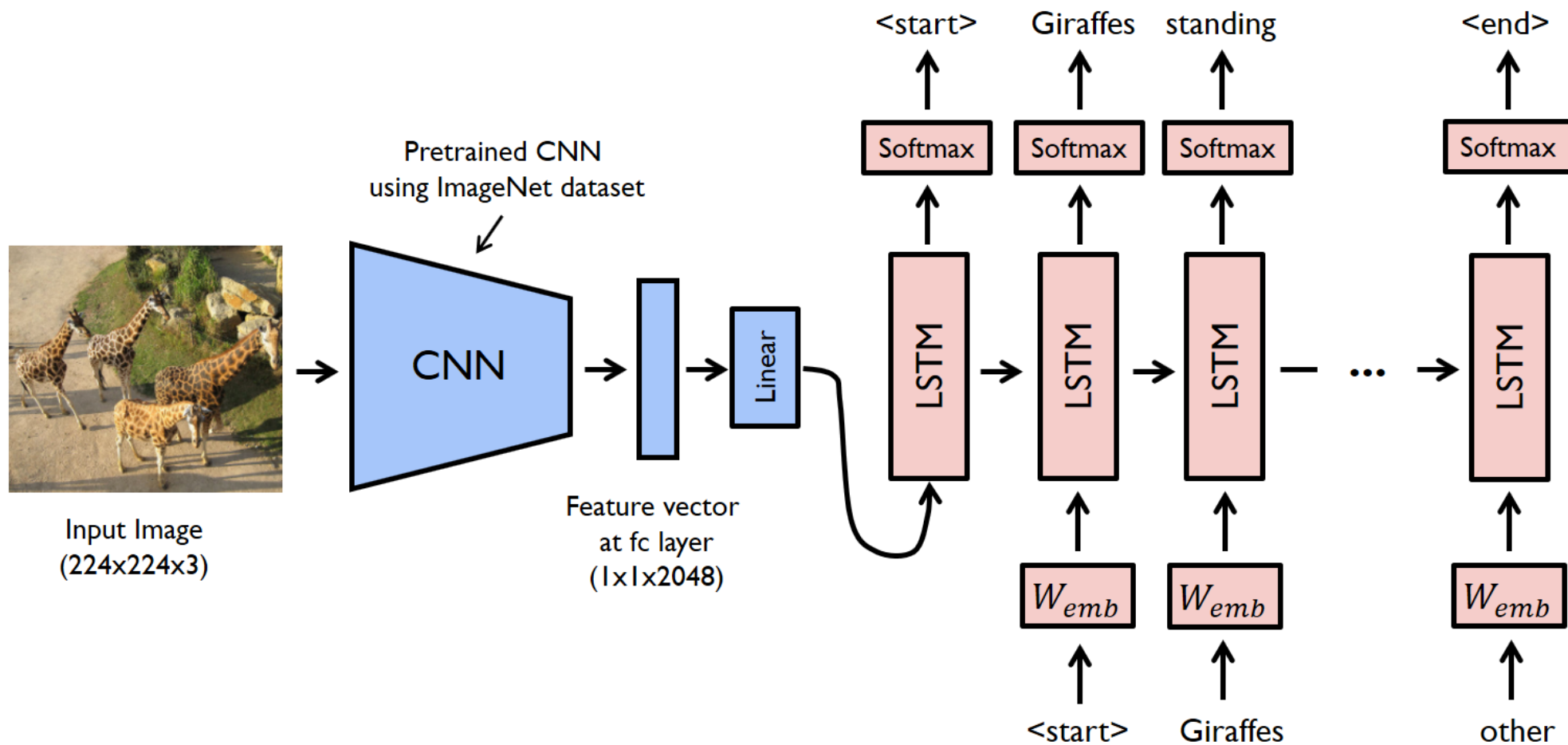
# Sequence to Sequence Model (LSTM) Applications



# Sequence to Sequence Model (LSTM) Applications



# Sequence to Sequence Model (LSTM) Applications



# Breakouts Time #2

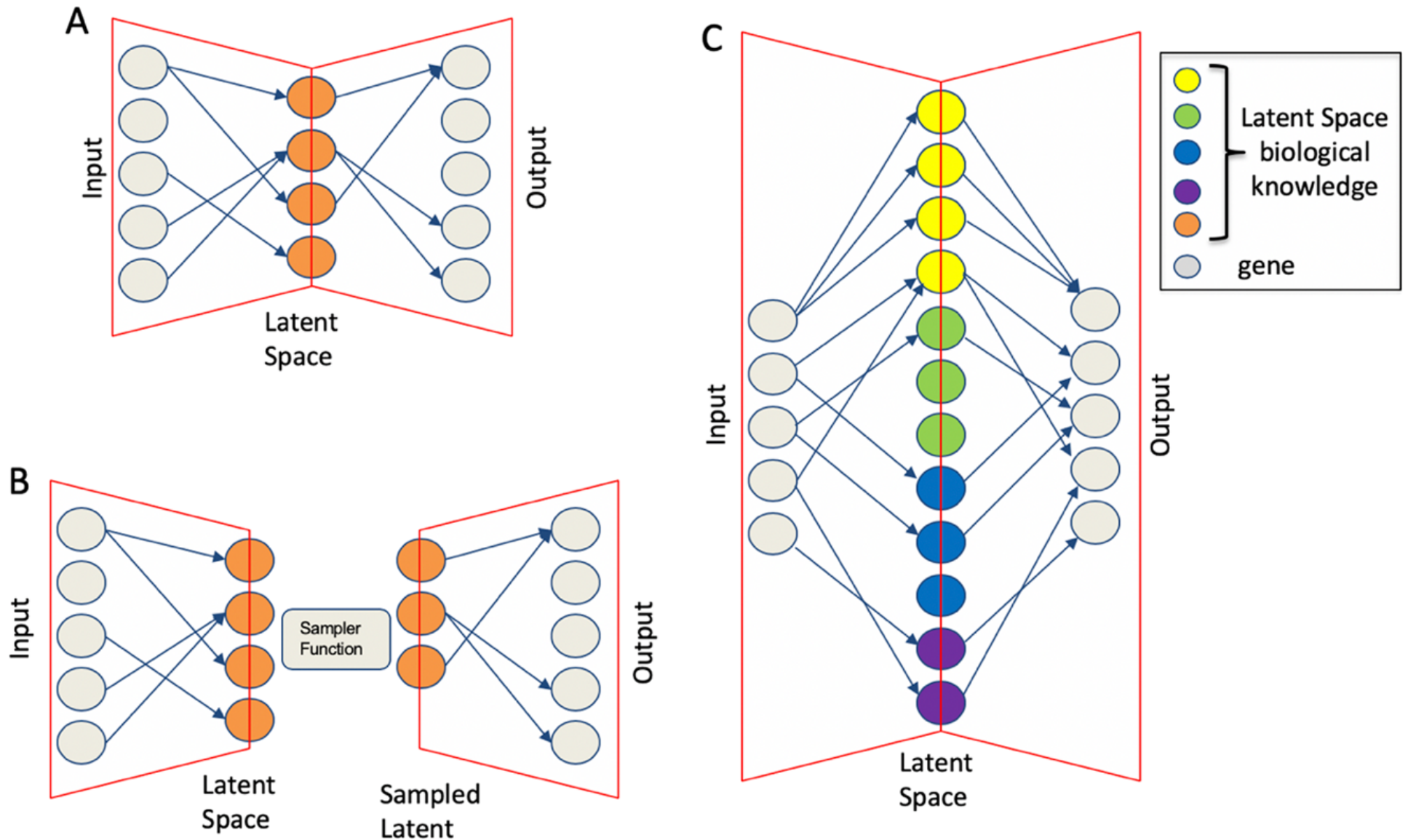
## Auto-complete — 5 mins

Let's say you are tasked with building an in-email auto-completion application, which can help complete partial sentences into full sentences through suggestions (auto-complete). How would you use what we have learned so far to model this? What architecture would you use? What would be your data? And what are some pitfalls or painpoints your model should address?

# Extra Slides

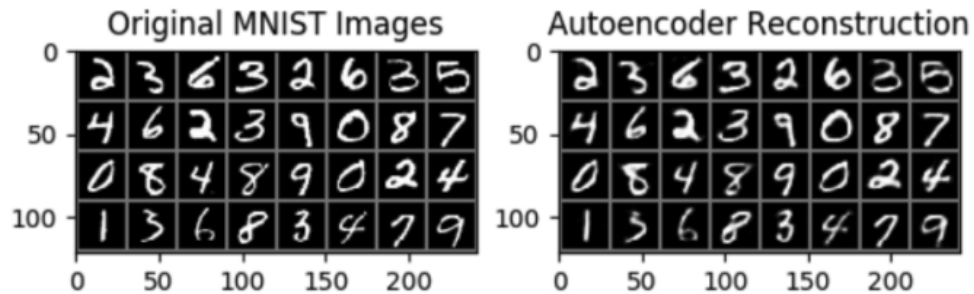
# Sparse Auto Encoders

## Sparse AE



# Sparse Auto Encoders

## Sparse AE Reference



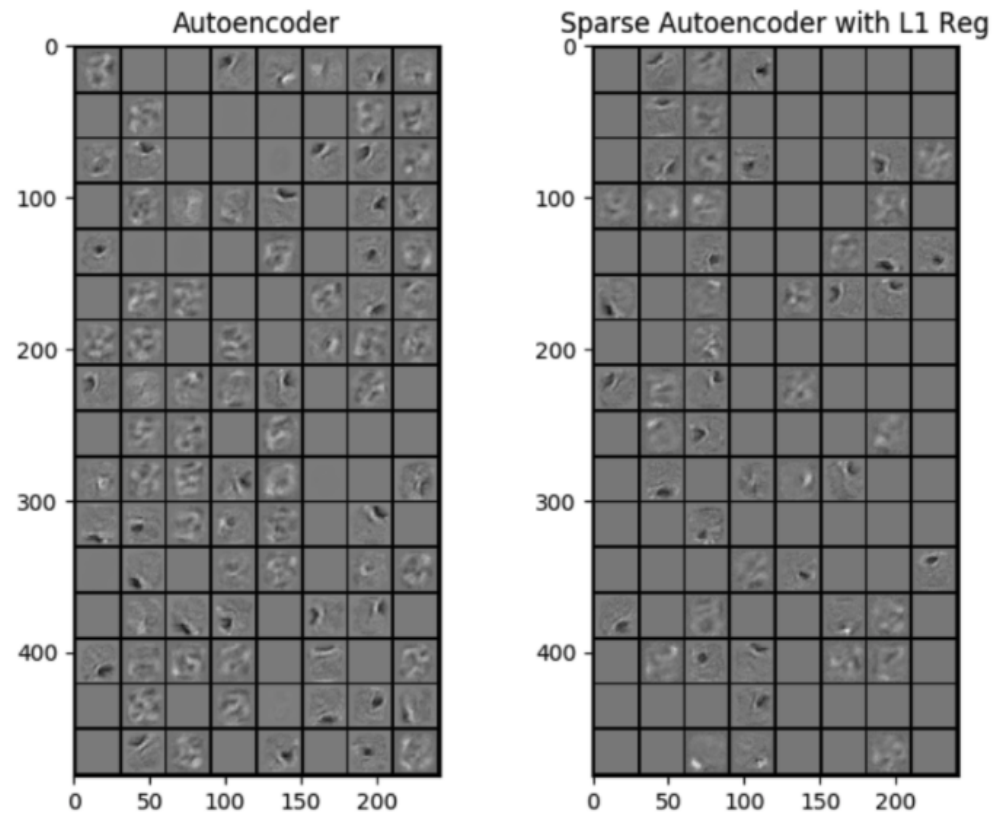
Methods	Best MSE Loss (MNIST or CIFAR-10)
Simple Autoencoder	0.0318 (MNIST)
Sparse Autoencoder (L1 reg)	0.0301 (MNIST)

Experiment Results



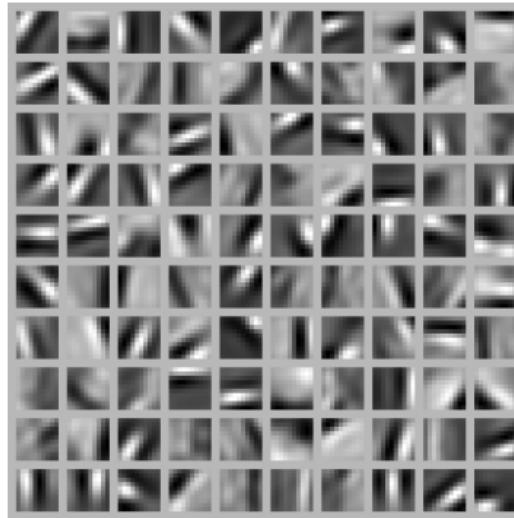
# Sparse Auto Encoders

## Sparse AE Reference



# Sparse Auto Encoders

Input Image that maximizes activations for each neuron in hidden layer!



# Sparse De-noising Auto Encoders

