EEP 596: LLMs: From Transformers to GPT $\|$ Lecture 4

Dr. Karthik Mohan

Univ. of Washington, Seattle

January 22, 2025

Outline for Lecture

- Training and Back-propagation

Outline for Lecture

- Training and Back-propagation
- Over-fitting and Hyper-parameters

Outline for Lecture

- Training and Back-propagation
- Over-fitting and Hyper-parameters
- Other DL architectures

Outline for Lecture

- Training and Back-propagation
- Over-fitting and Hyper-parameters
- Other DL architectures
- Embeddings and Semantic Search

House Keeping Items

- Office Hours and Review Hours → webpage (TeaChip Support)
- Assignment 2 will be a Mini-Project 1 assigned today → 2.5 weeks
- Mini-Project 1 will come in two main parts and cover embeddings and semantic search
- Two people per team to maximize learning - Spreadsheet for team pairing to be shared by TAs → team of 2
- Any questions?

MP1 → Part 1
→ Part 2
→ Kaggle Contest

Semantic Search
Embedding
Streamlit / Demo
Kaggle

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
Deep Learning History

Recap from Last time!

- Perceptron and Logistic Regression

Recap from Last time!

- Perceptron and Logistic Regression
- How can 2-layer NN learn an XOR function

Recap from Last time!

- Perceptron and Logistic Regression
- How can 2-layer NN learn an XOR function
- Concepts of right fit, over-fitting and under-fitting

Recap from Last time!

- Perceptron and Logistic Regression
- How can 2-layer NN learn an XOR function
- Concepts of right fit, over-fitting and under-fitting
- Tensorflow demo

Multi-Layer Perceptron (MLP)

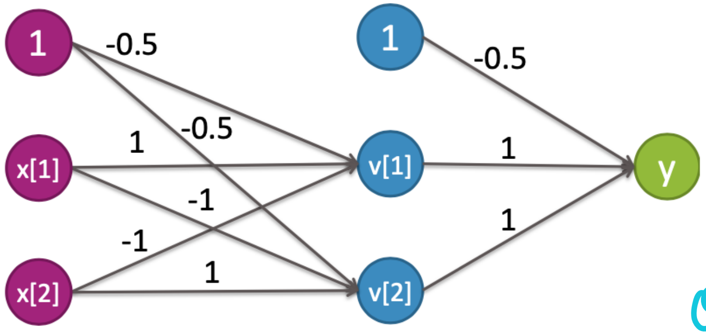
This is a 2-layer neural network

$y = x[1]\ XOR\ x[2] = (x[1]\ AND\ !\,x[2])\ OR\ (!\,x[1]\ AND\ x[2])$

$$v[1] = (x[1]\ AND\ !\,x[2]) \\ = g(-0.5 + x[1] - x[2])$$

$$v[2] = (!\,x[1]\ AND\ x[2]) \\ = g(-0.5 - x[1] + x[2])$$

$$y = v[1]\ OR\ v[2] \\ = g(-0.5 + v[1] + v[2])$$



output

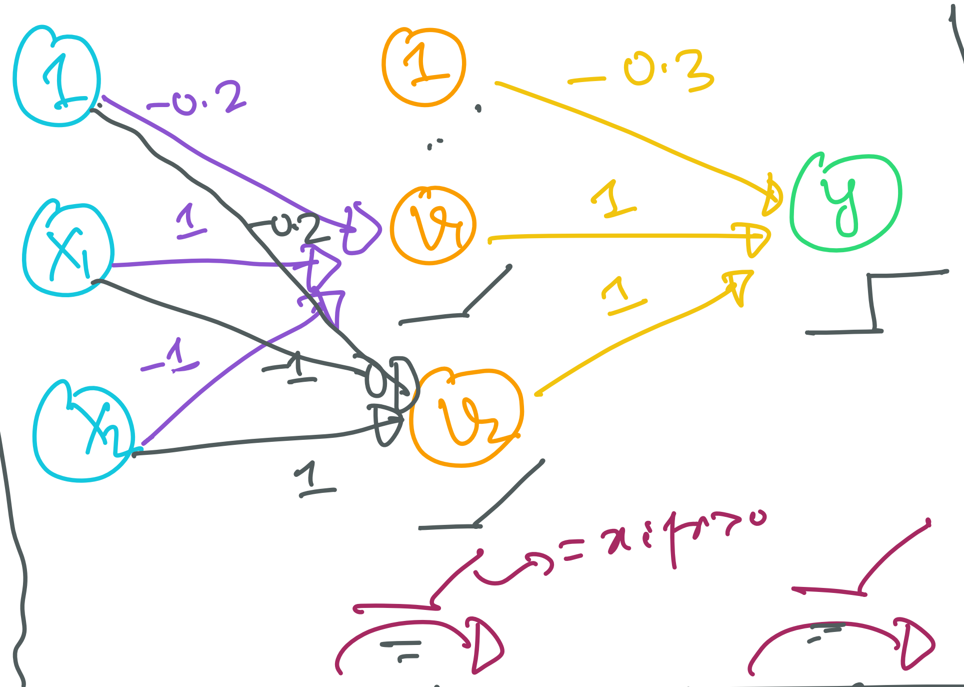
Input

Information Flow

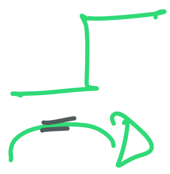
("Forward Propagation")

$\tilde{v}_1 = -0.2 + x_1 - x_2$

$\tilde{v}_2 = -0.2 - x_1 + x_2$



$\tilde{y} = -0.3 + v_1 + v_2$



x_1	x_2	$\tilde{v}_1$	v_1	$\tilde{v}_2$	v_2	$\tilde{y}$	y
0	0	-0.2	0	-0.2	0	-0.3	0
1 ✓	0	0.8	0.8	-1.2	0	0.5	1
0 ✓	1	-1.2	0	0.8	0.8	0.5	1
1	1	-0.2	0	-0.2	0	-0.3	0

$-0.2 + 1 - 1 = -0.2$

$-0.2 - 1 + (1)$

ICE from last lecture ✓

Breakout Session: Would ReLU work?

Work in your breakouts

What weights would give a y value > 0.7 for $(1, 0), (0, 1)$ inputs and a value of $y <$ -0.7 for $(0, 0), (1, 1)$ for the ReLU function?

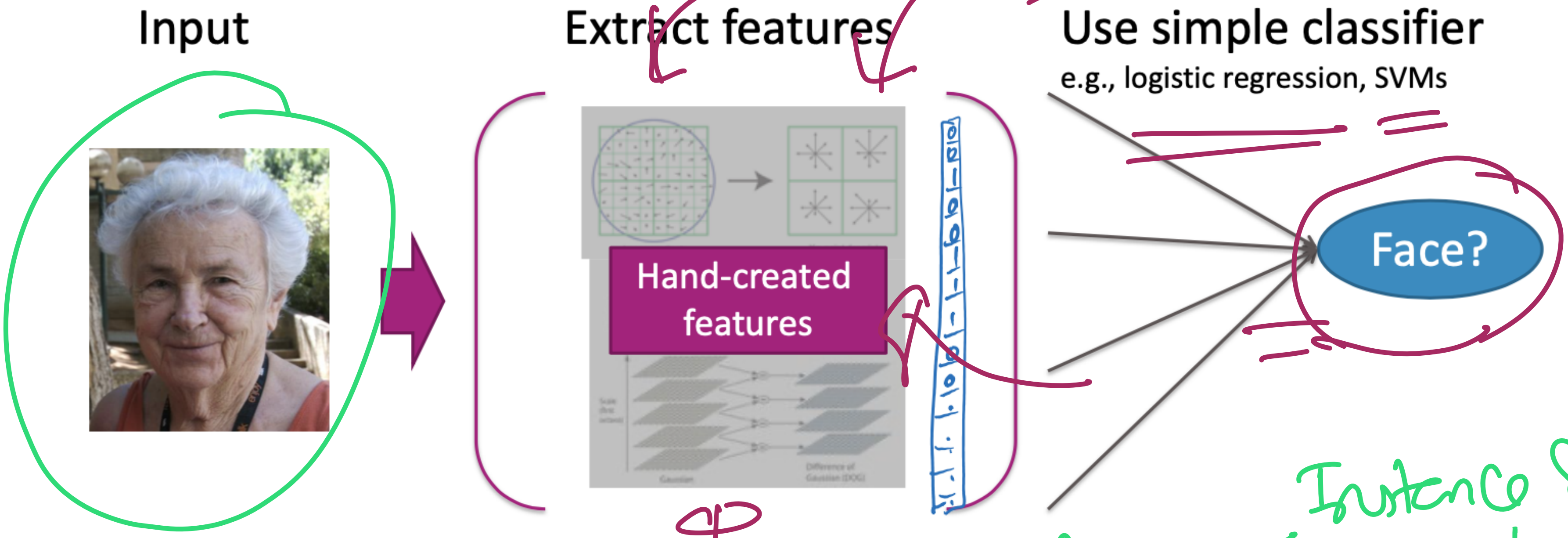
The phenomenon of Overfitting

Computer vision before deep learning

Feature Engineering

Conv.

Pooling



Input

Extract features

Use simple classifier

e.g., logistic regression, SVMs

Hand-created features

Face?

Instance Segm.

Image understanding

Image to text

. . .

CV

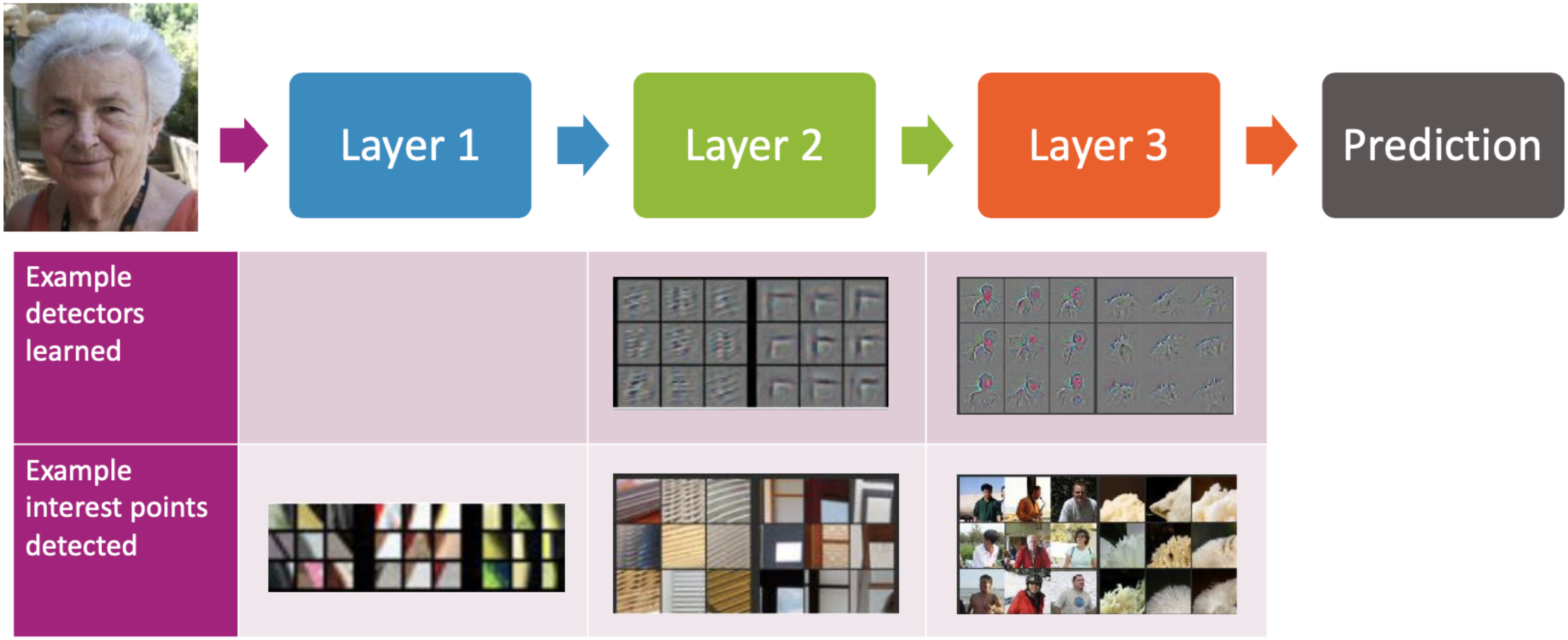
Image Generation

(GANs)

→ Text to Image

→ Image Gen. . .

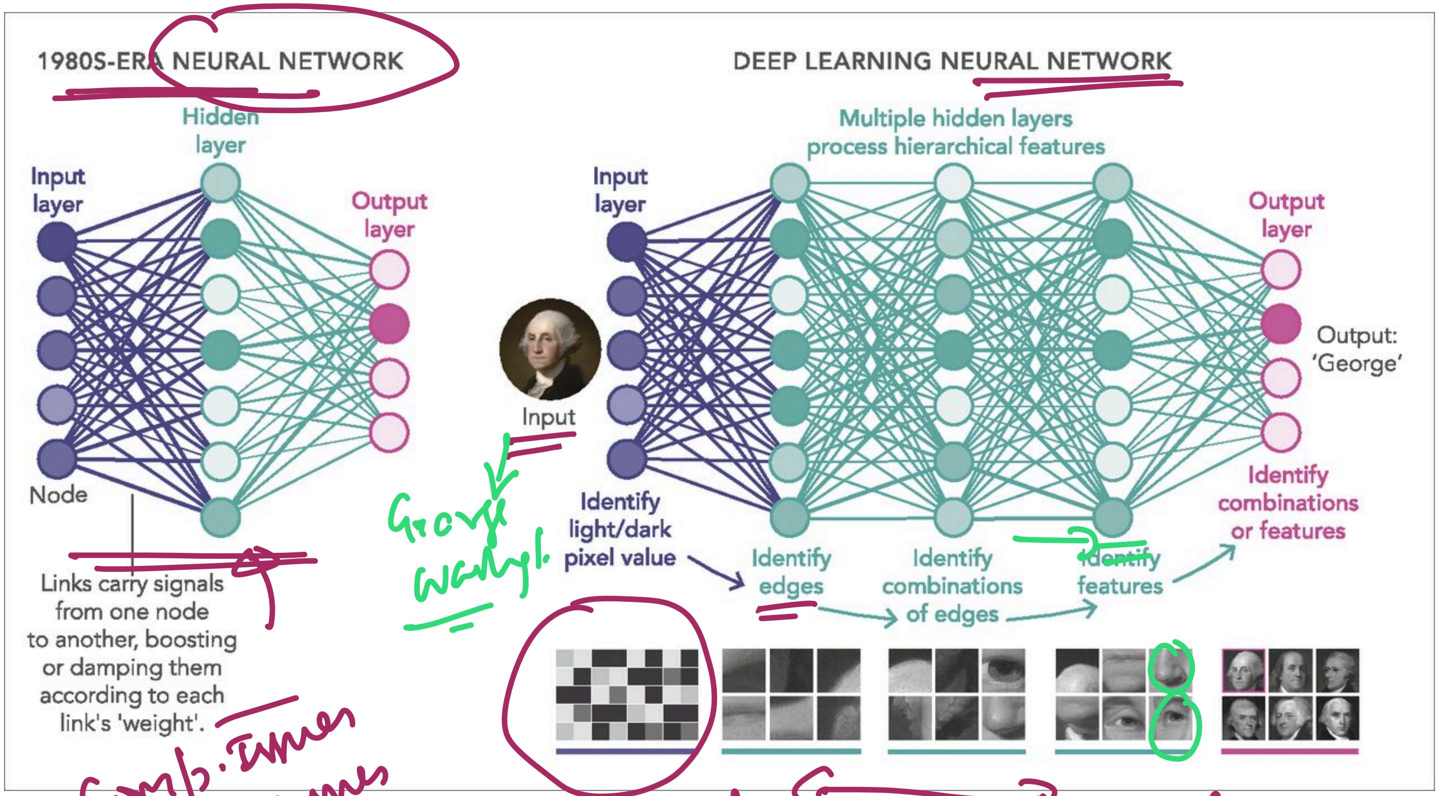
Computer vision after deep learning



[Zeiler & Fergus '13]

DL Can extract fine-tuned features automatically

Feed-forward Deep Learning Architecture Example


1980S-ERA NEURAL NETWORK
Hidden layer
Input layer
Output layer
Node
Links carry signals from one node to another, boosting or damping them according to each link's 'weight'.
DEEP LEARNING NEURAL NETWORK
Multiple hidden layers process hierarchical features
Input layer
Output layer
Input
Output: 'George'
Identify light/dark pixel value
Identify edges
Identify combinations of edges
Identify features
Identify combinations or features
Gray value

Comp. Times
Mem. Times

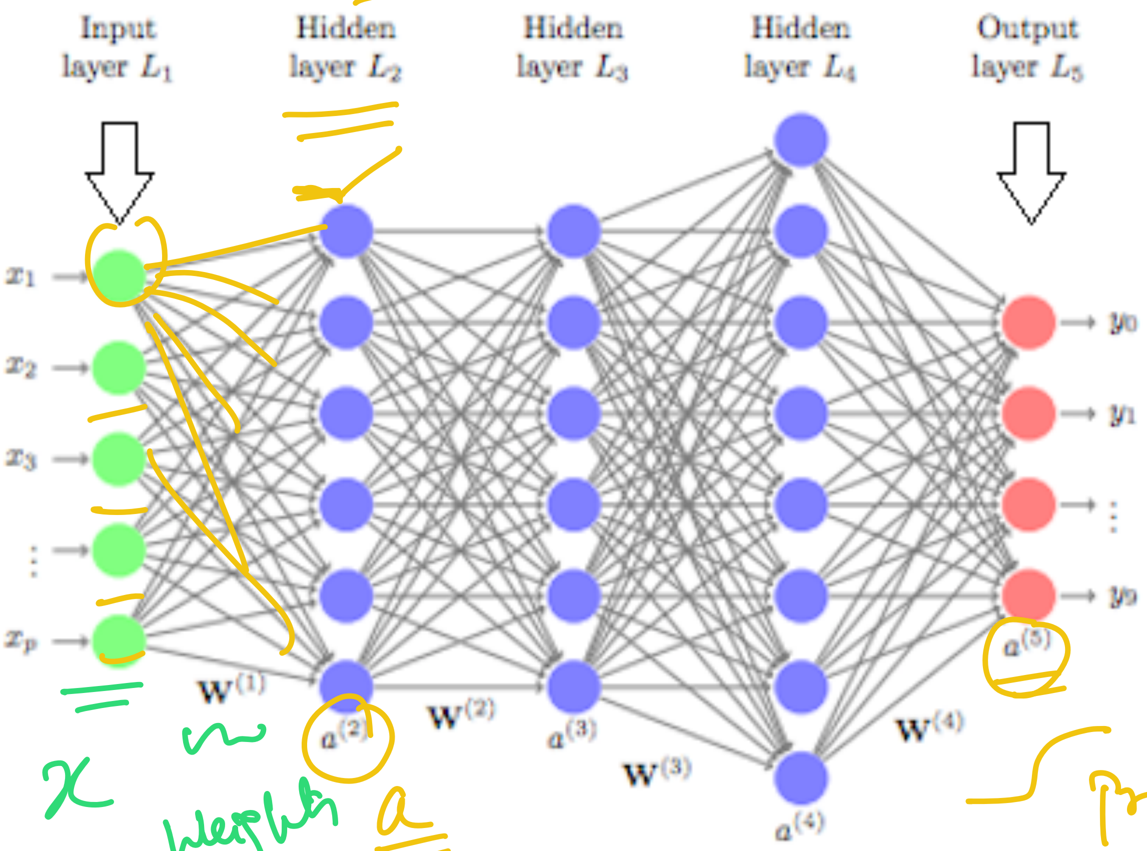
Gray Scale

Layers are Deep

Coarse grained features → Fine grained features

Feed-forward Deep Learning Architecture Example

NN w 3 hidden layers

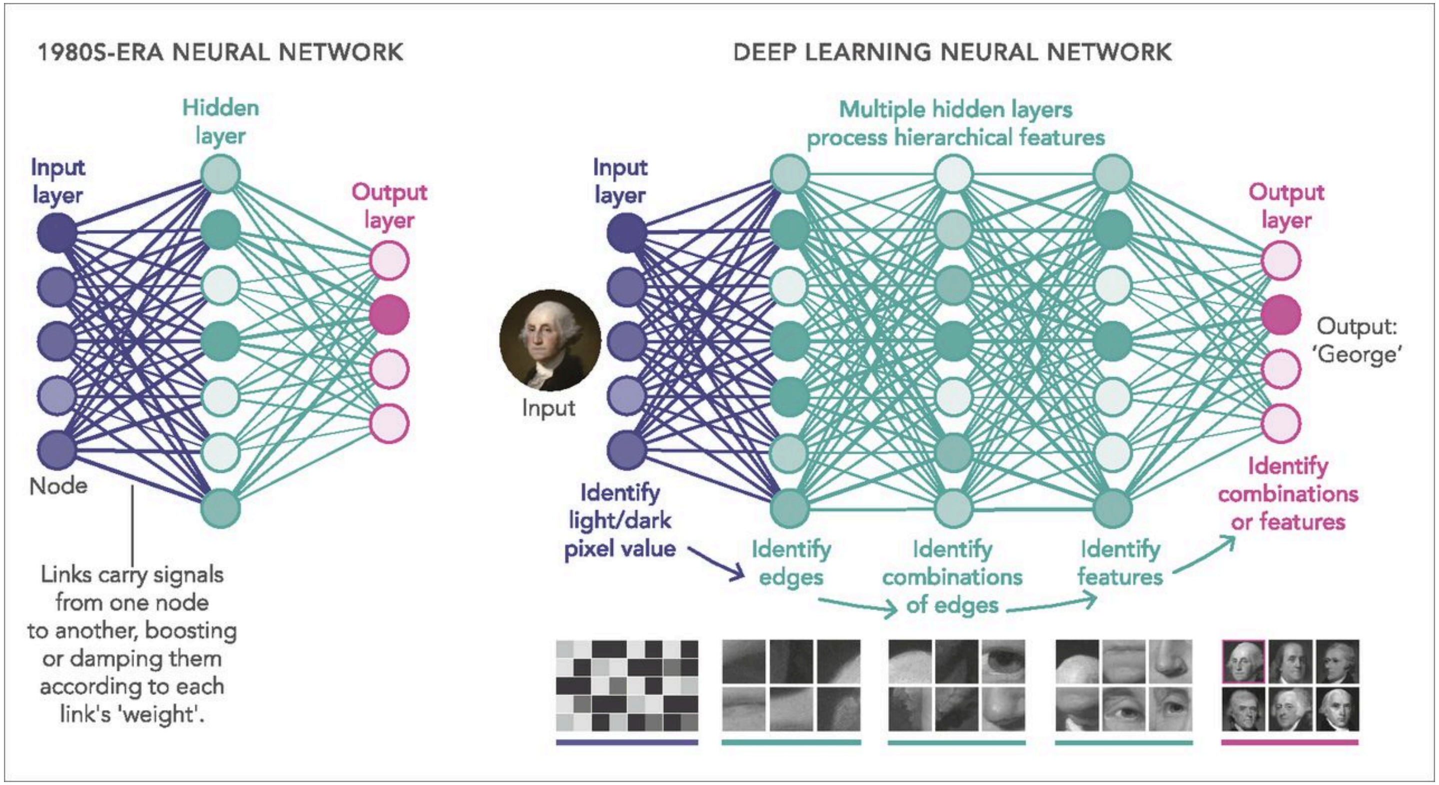


probability vector

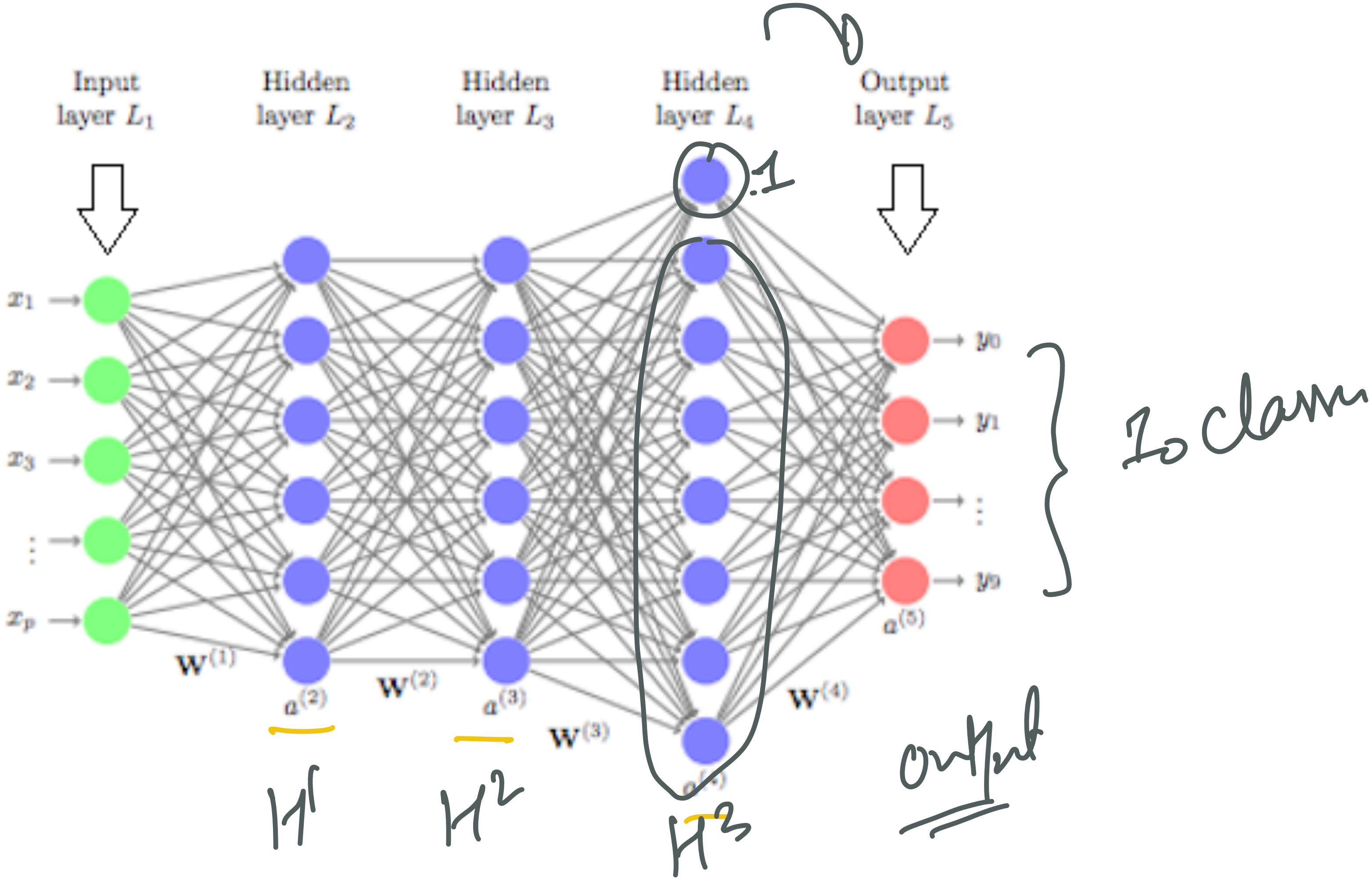
x ~ weights a

W = [ ]

Feed-forward Deep Learning Architecture Example


1980S-ERA NEURAL NETWORK
Input layer
Hidden layer
Output layer
Node
Links carry signals from one node to another, boosting or damping them according to each link's 'weight'.
DEEP LEARNING NEURAL NETWORK
Multiple hidden layers process hierarchical features
Input layer
Output layer
Input
Output: 'George'
Identify light/dark pixel value
Identify edges
Identify combinations of edges
Identify features
Identify combinations or features

Feed-forward Deep Learning Architecture Example

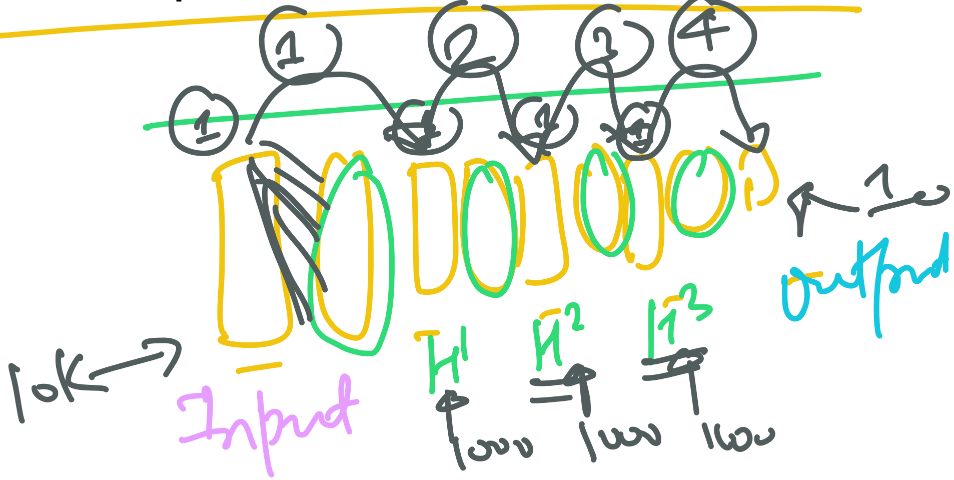


ICE #1

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



③ = 1K × 1K = 1M

④ = 1000 × 10 = 10K

① = (100×100) × 1000 = 10M

② = 1000 × 1000 = 1M

①+②+③+④ = 12M + change

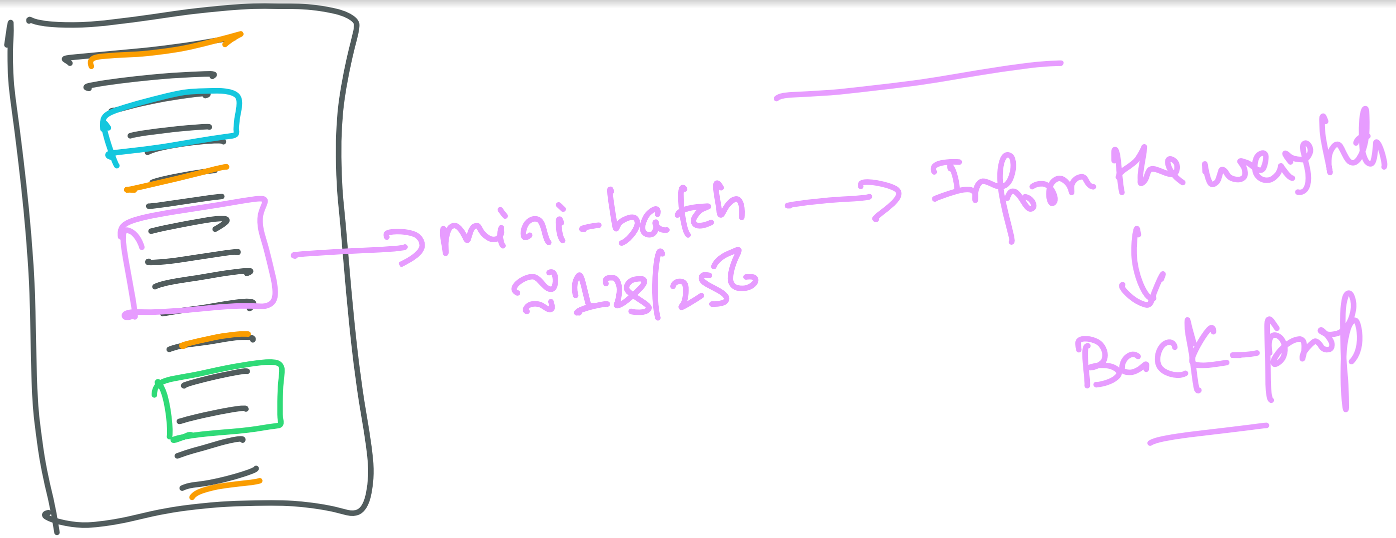
Training a DNN

Stochastic Gradient Descent with Mini-batch

SGD – Main optimizer engine / Training engine for almost any ML model

SGD with mini-batch

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

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.

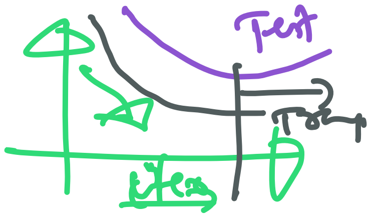
How do we compute gradient in a DNN?

Back-propagation!

Truth Prediction

$$\ell(w) = -\sum_i \left\{ y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i) \right\} / N$$

Data



$\ell(w)$

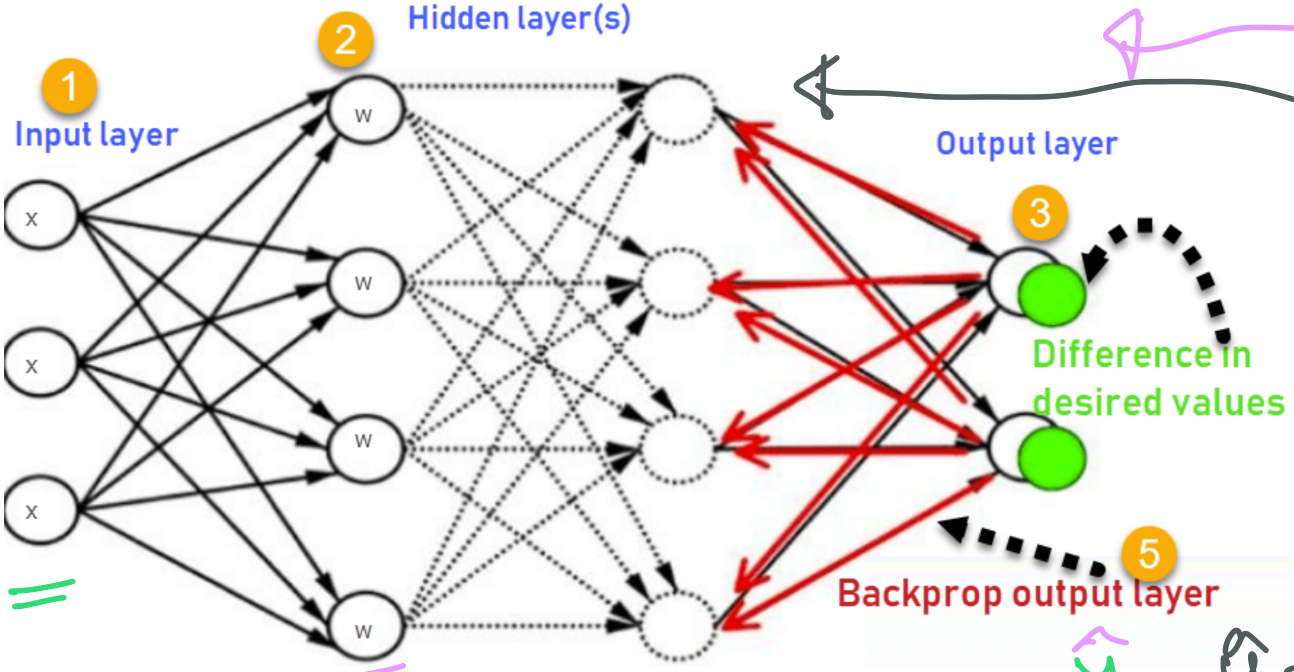
$$w^{k+1} \leftarrow w^k - lr \cdot \nabla \ell(w^k)$$

Gradient – Mathematical quantity that transfers info. for learning weights in a DNN

Start $\rightarrow w^0 \rightarrow w^1 \rightarrow w^2 \rightarrow \cdots \rightarrow w^k$ $\rightarrow$ Trained weights

Forward Propagation vs Back-propagation

Back Prop (Flow of Info / Gradient comp. from Output to Input)



$\hat{y}=0.6, y=1$

X

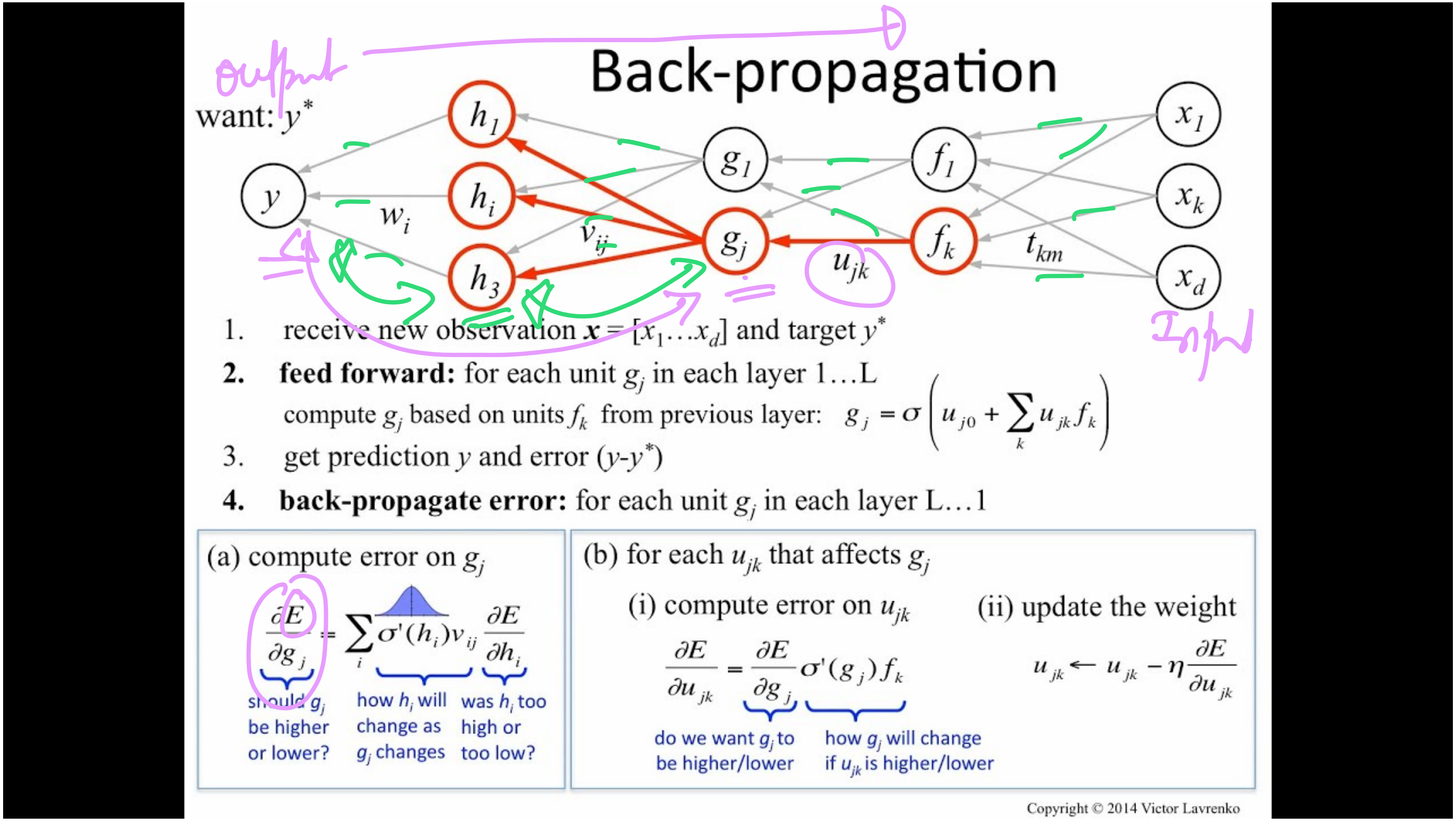
$\hat{y}$

$\hat{y}=0.3, y=1$

X

Forward Prop (Compute outputs | Inputs)

Back Propagation explained



Back-propagation

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

(a) compute error on g_j

$$\underbrace{\frac{\partial E}{\partial g_j}}_{\text{should } g_j \text{ be higher or lower?}} = \sum_i \underbrace{\sigma'(h_i) v_{ij}}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$$

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

(i) compute error on u_{jk}

$$\frac{\partial E}{\partial u_{jk}} = \underbrace{\frac{\partial E}{\partial g_j}}_{\text{do we want } g_j \text{ to be higher/lower}} \underbrace{\sigma'(g_j) f_k}_{\text{how } g_j \text{ will change if } u_{jk} \text{ is higher/lower}}$$

(ii) update the weight

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

Copyright © 2014 Victor Lavrenko

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.

$$f(w) = w^2 \qquad \nabla f(w) = 2w = \frac{df}{dw}$$

$$f(\underbrace{w_1, w_2}) = w_1^2 + w_2^2 \qquad \nabla f(w) = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix} = \begin{bmatrix} 2w_1 \\ 2w_2 \end{bmatrix}$$

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \qquad = 2\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = 2w \to \text{vector}$$

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.

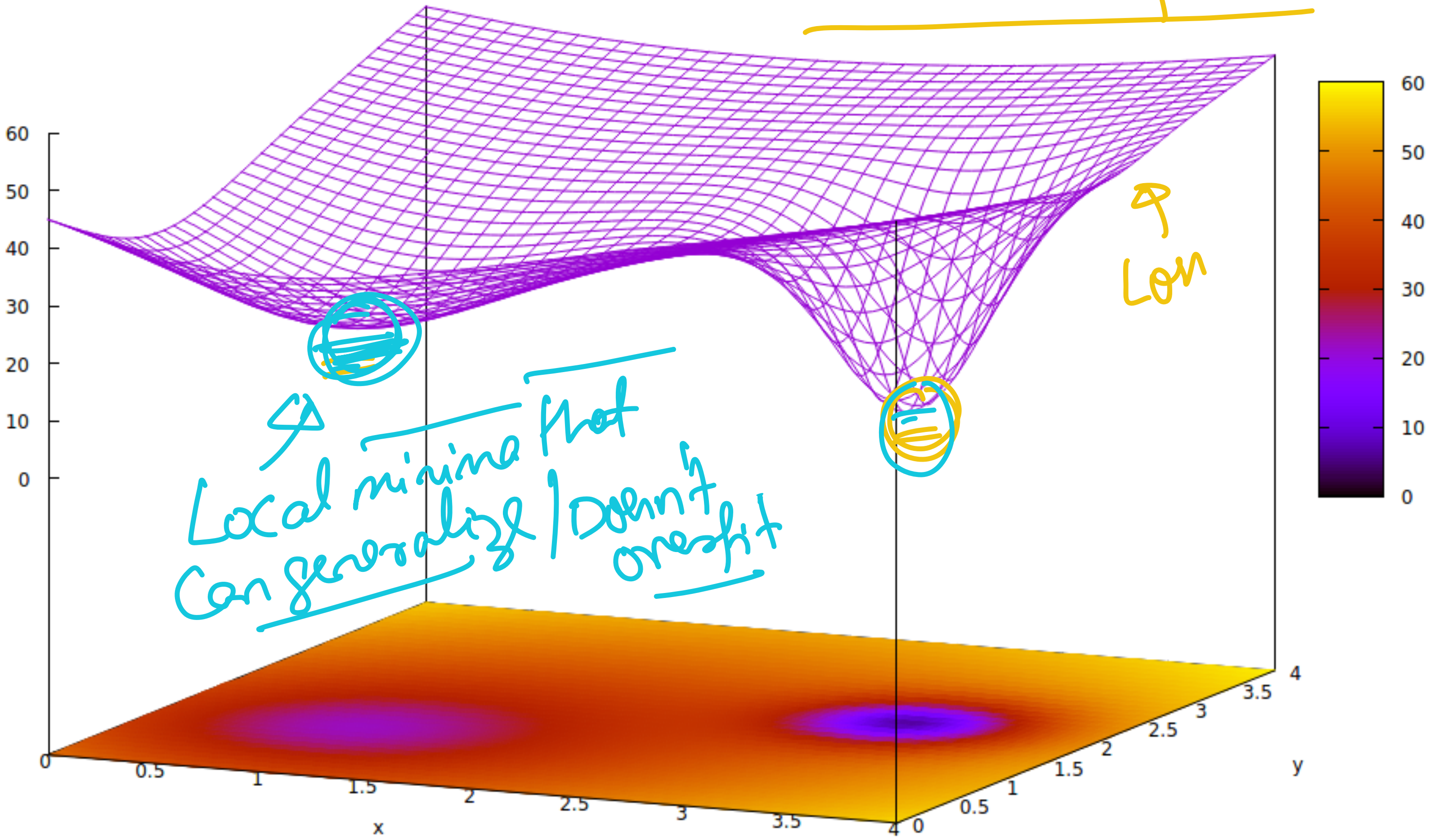
Back Prop as information flow

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

DNN - Highly non-convex function

lr



Loss

Local minima that
Can generalize | Doesn't overfit

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

0.01, 0.1, 0.00001

↑LR → Faster Training

↓LR → Slower Training

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

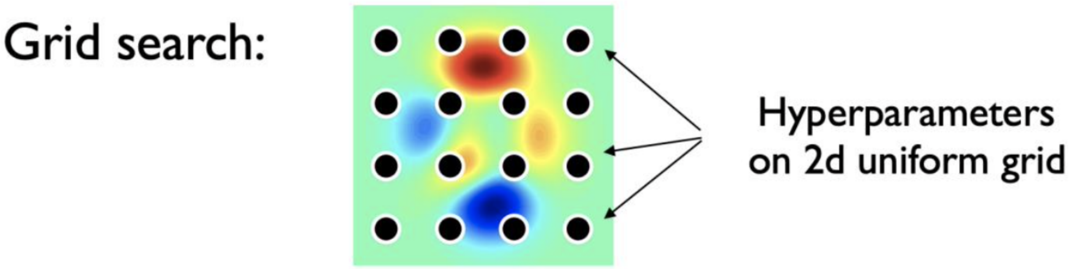
1. Learning rate
2. Number of Hidden Layers
3. Number of neurons per hidden layer
4. Type of non-linear activation function used

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:

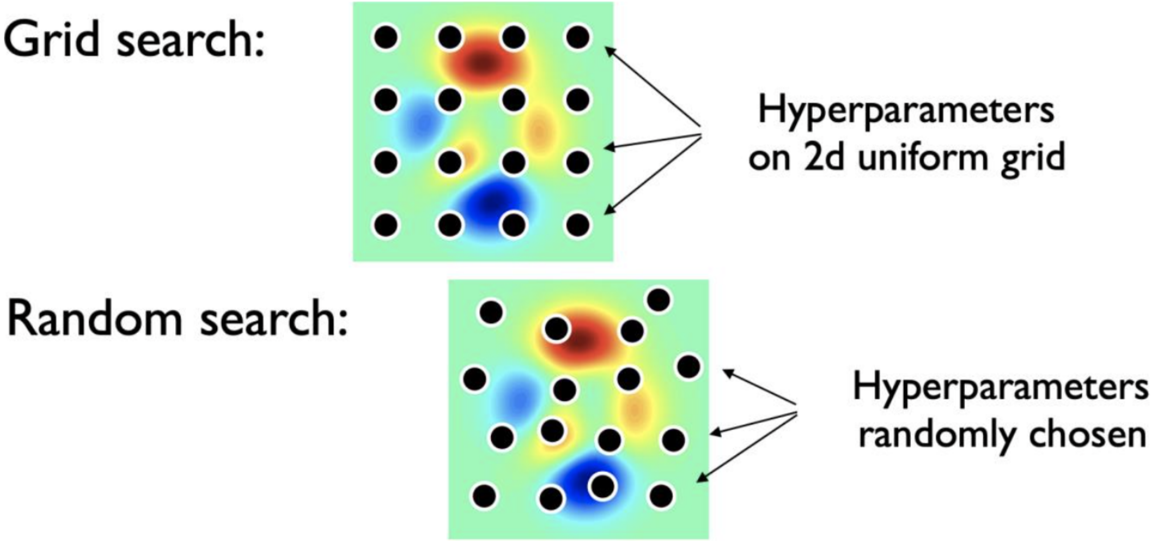
Hyperparameters
on 2d uniform grid

lr = 0.1

lr = 0.01

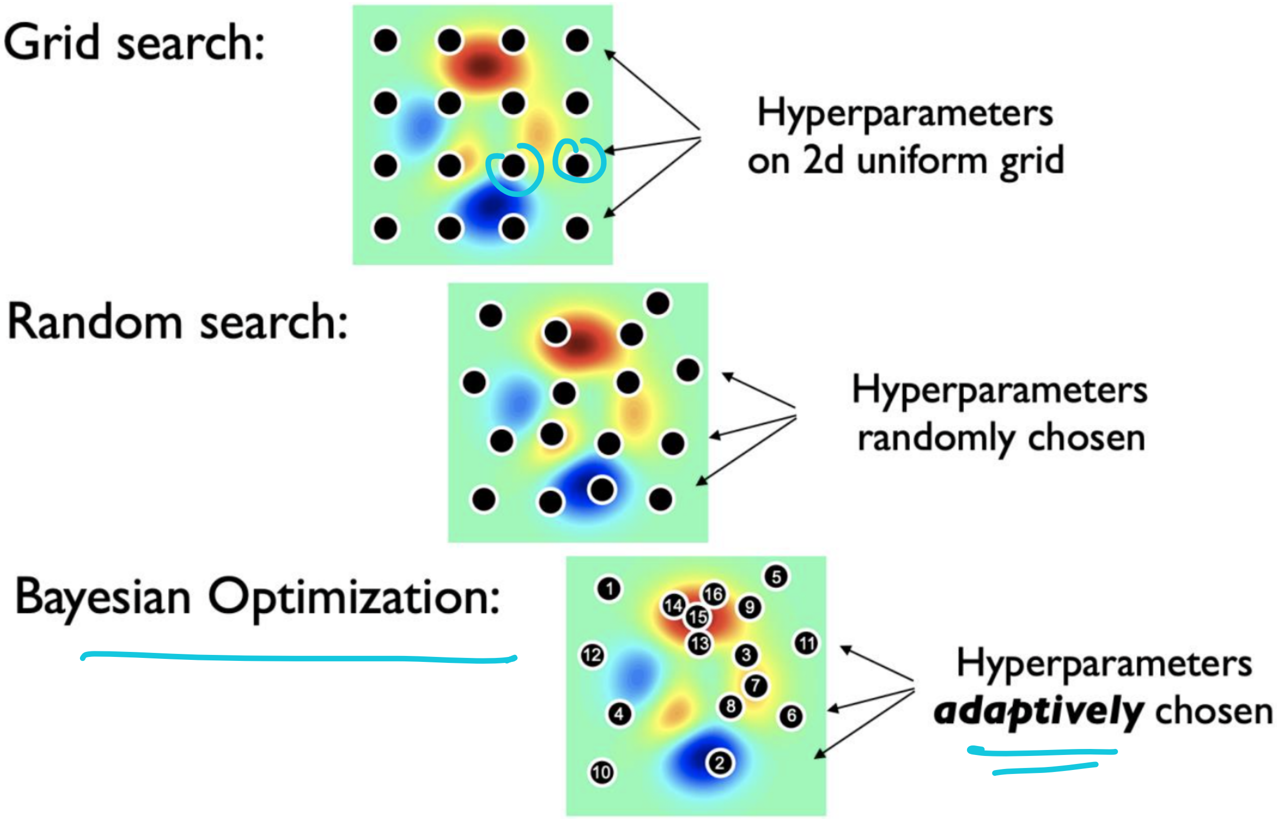
lr = 0.0001

Hyper-parameter tuning methods



Grid search:
Hyperparameters on 2d uniform grid
Random search:
Hyperparameters randomly chosen

Hyper-parameter tuning methods



Grid search:

Hyperparameters on 2d uniform grid

Random search:

Hyperparameters randomly chosen

Bayesian Optimization:

Hyperparameters ***adaptively*** chosen

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 - ℓ_1, ℓ_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 - ℓ_1, ℓ_2
3. More common over-fitting strategy for DL?

Over-fitting in DNNs

How to handle over-fitting in DNNs

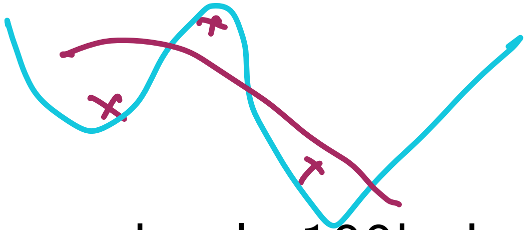
1. A DNN model with 100 million parameters and only 100k data points or even a million data points will overfit unless we take care of over-fitting.
2. Weight regularization can help - ℓ_1, ℓ_2
3. More common over-fitting strategy for DL?
4. Dropouts!

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 - ℓ_1, ℓ_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

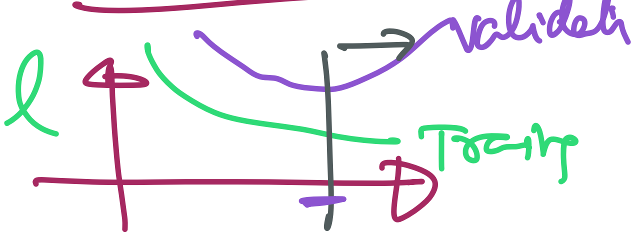


How to handle over-fitting in DNNs

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.

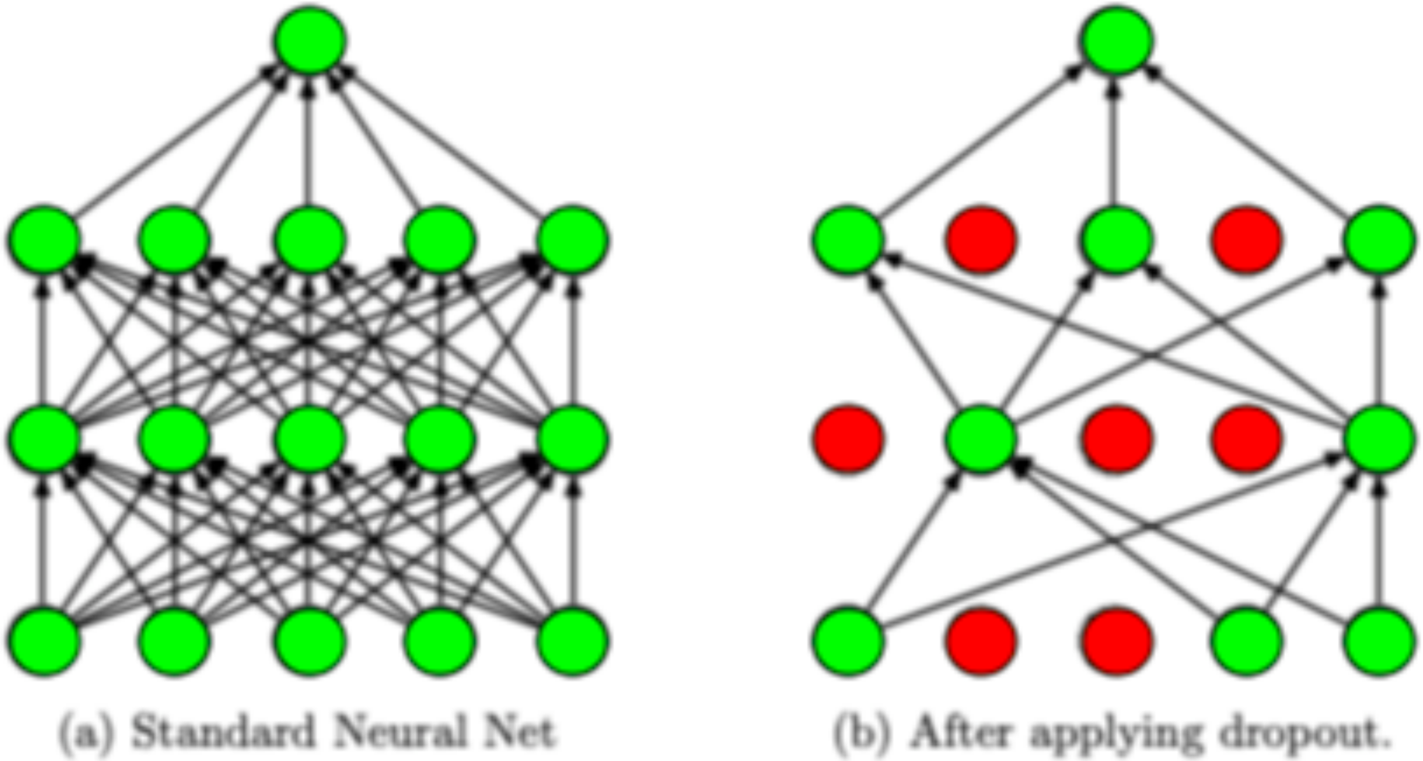
100M Param >>> 100k Data Points

2. Weight regularization can help - ℓ_1, ℓ_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

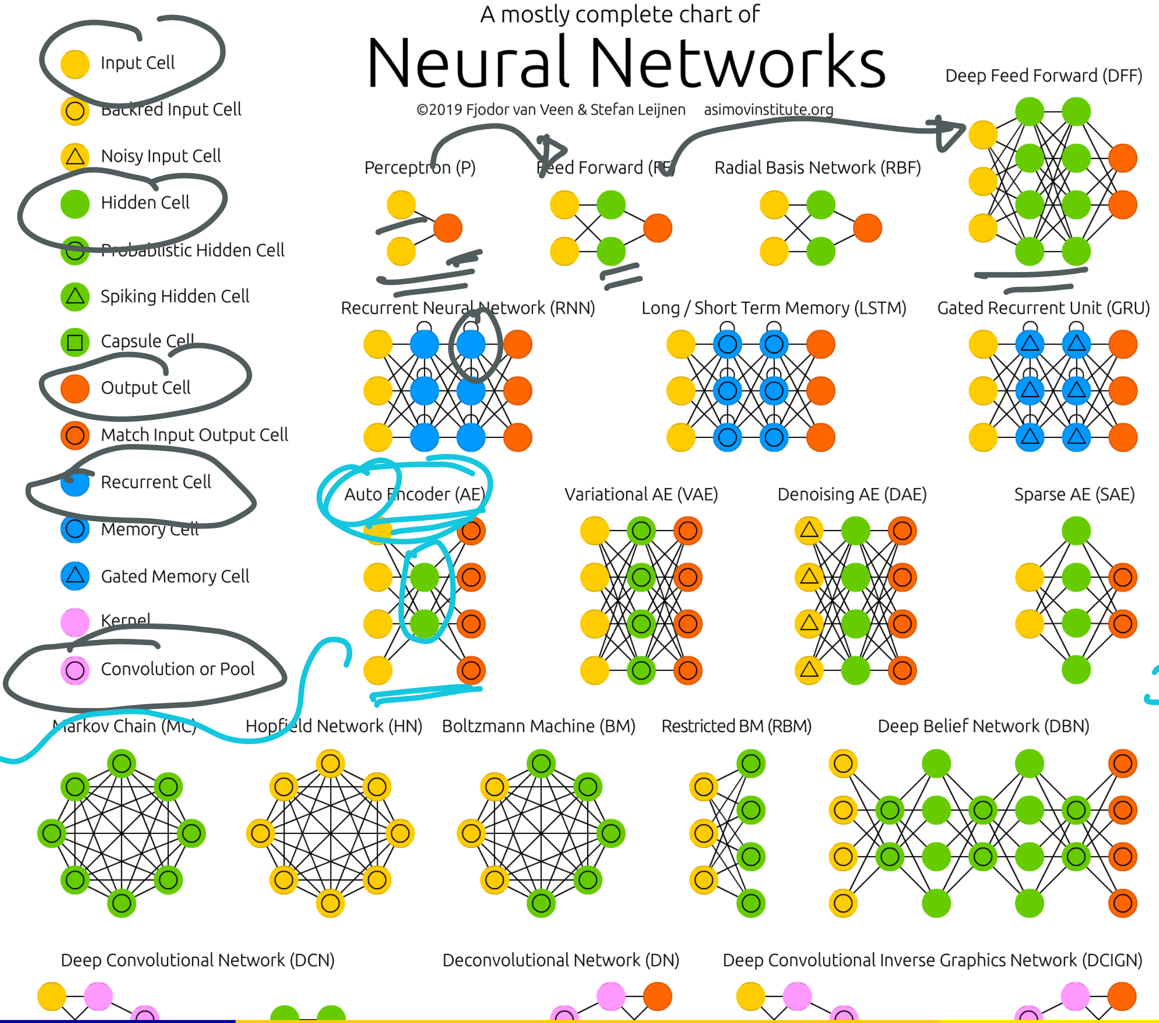
Tensorflow Playground Demo

More DL Architectures

Neural Networks Zoo

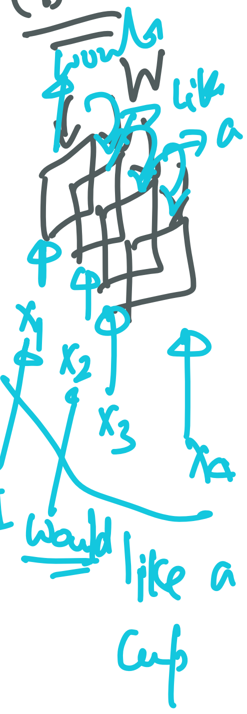
Zoo Reference

Interpretable DNNs



Stable Diffusion

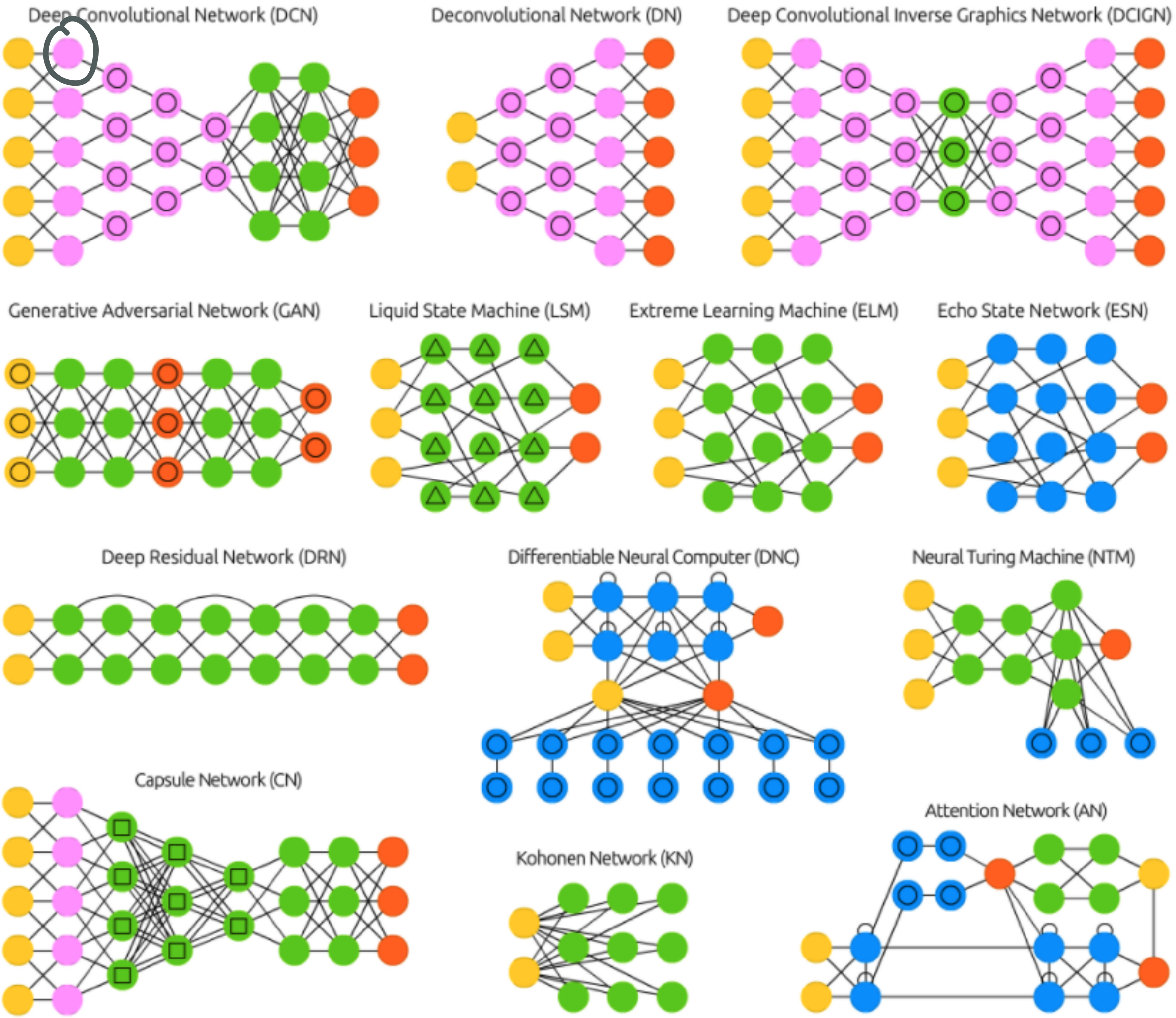
(DNN)



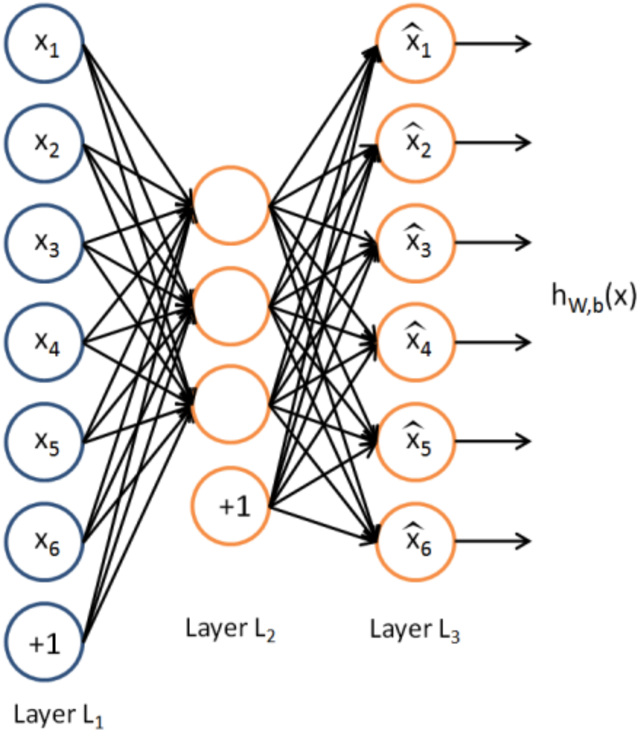
I would like a cup

More DL Architectures

Neural Networks Zoo


Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Differentiable Neural Computer (DNC)
Neural Turing Machine (NTM)
Capsule Network (CN)
Kohonen Network (KN)
Attention Network (AN)

Auto Encoders



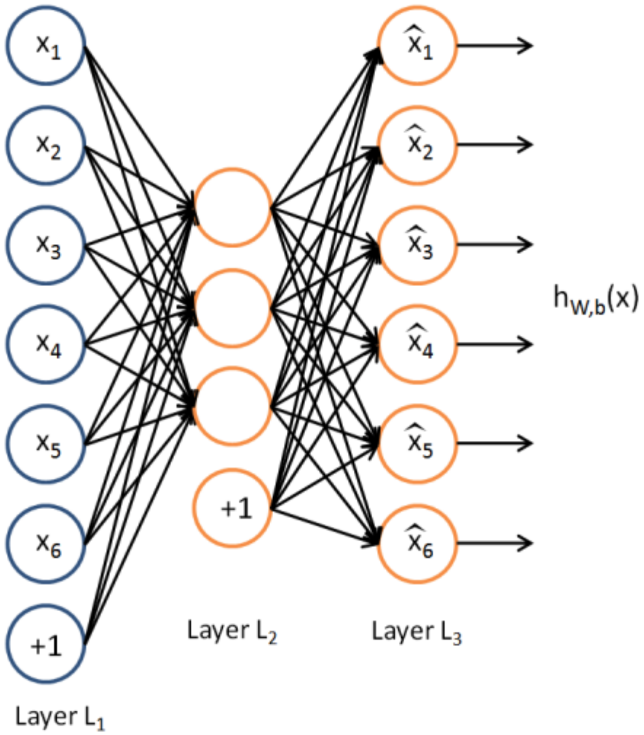
ICE #3

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

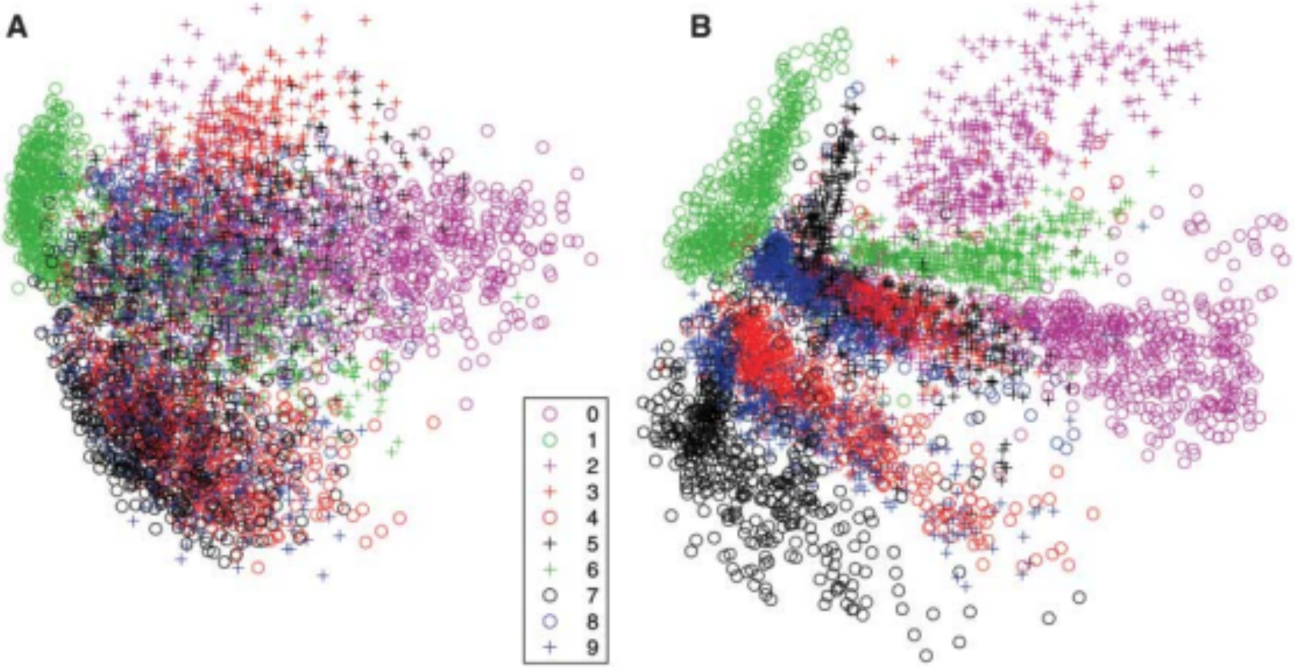
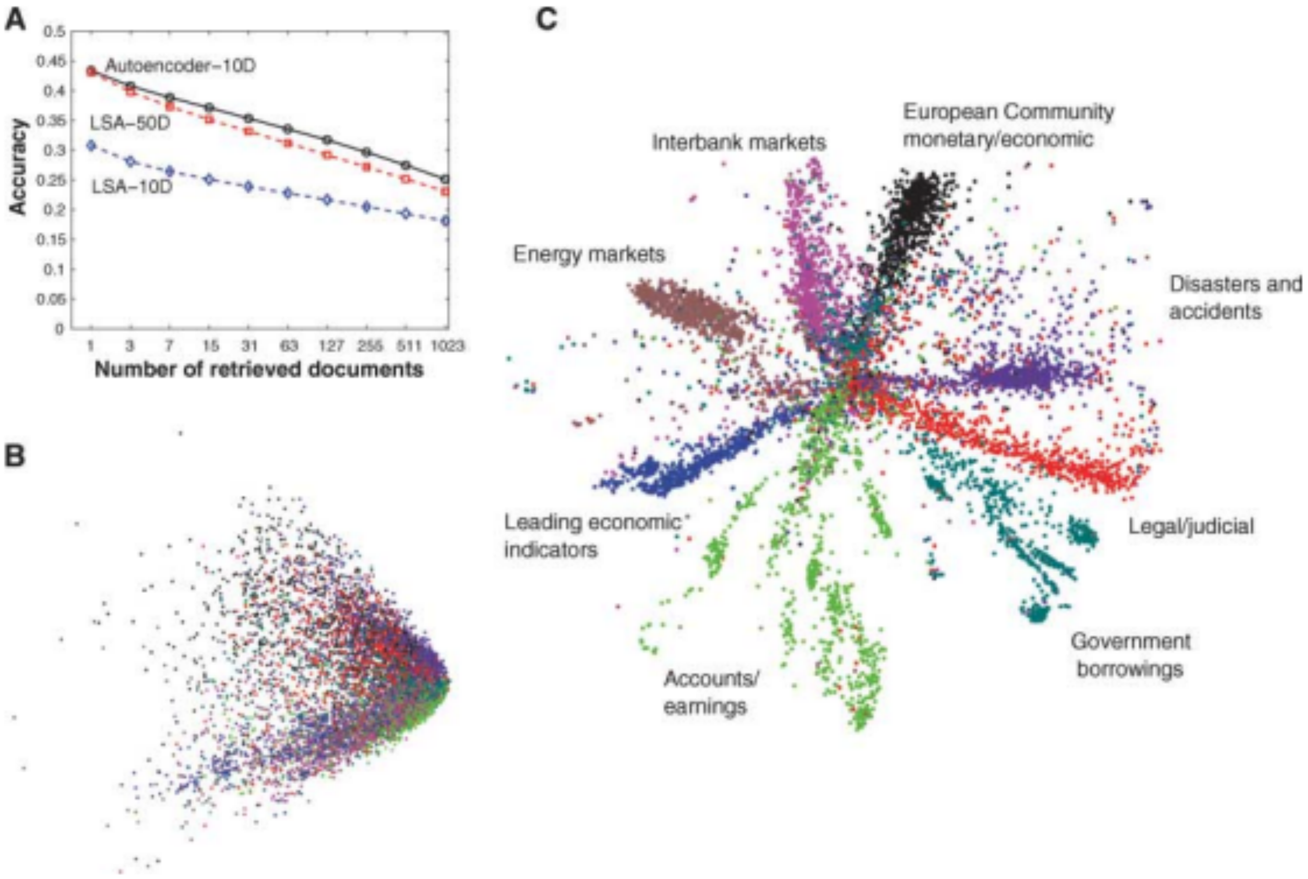


Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).

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.



AutoEncders Summary

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

AutoEncders 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

AutoEncders 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?

AutoEncders 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!!