EEP 596: LLMs: From Transformers to GPT || Lecture 3

Dr. Karthik Mohan

Univ. of Washington, Seattle

January 11, 2024

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

Office Hours

Michael :- M 11am?

Shreenid :- F 5:30PM

Karthik :- T after class

Quiz section

Every Friday :- 5PM - 5:30PM

Outline for Lecture

notability

- Training and Back-propagation
- Over-fitting and Hyper-parameters
- Other DL architectures
- Deep Learning, Embeddings and Vector Search

Back-ground/ Pre-Reqs

Pine Cone

Logistics :- Assignment 1 due this weekend

Outline for Lecture

- Training and Back-propagation
- Over-fitting and Hyper-parameters
- Other DL architectures
- Deep Learning, Embeddings and Vector Search
- Working with Embeddings

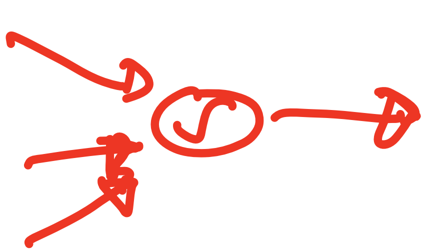
Deep Learning References

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 Sentence Embeddings

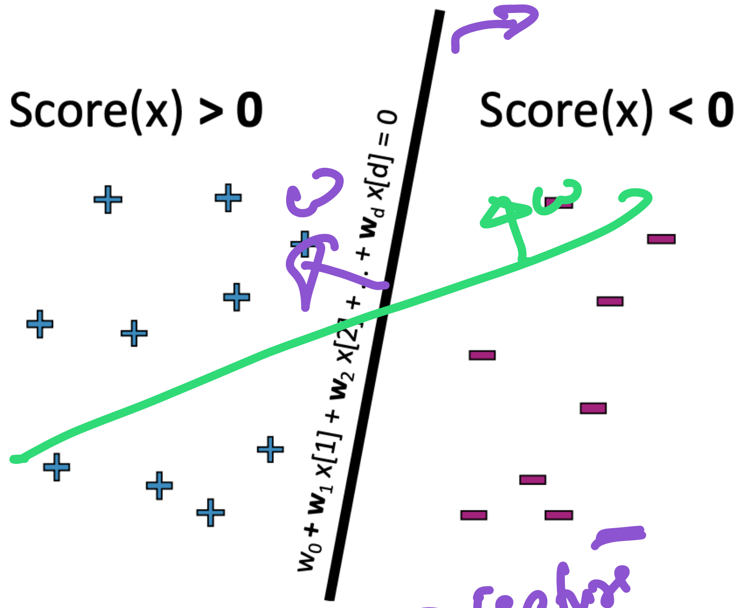
Recap from last lecture

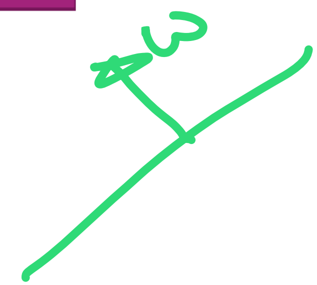
- Perceptron = Artificial Neuron
- OR/AND functions
- XOR
- Activation Functions



Perceptron

Score(x) = $w_0 + \mathbf{w}_1\, x[1] + \mathbf{w}_2\, x[2] + \ldots + \mathbf{w}_d\, x[d]$





Perceptron

= Single Artificial Neuron

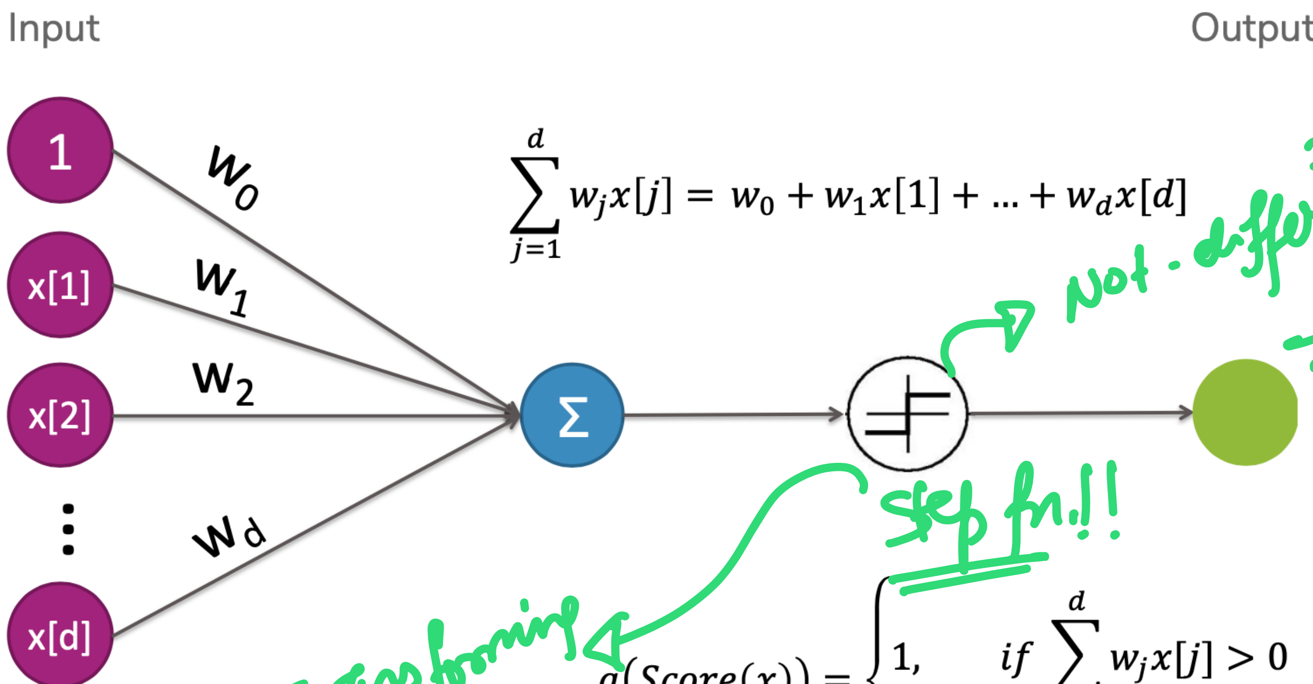
represented by $w \in \mathbb{R}^d$

w is a d-dim. vector

$w = [\ \vdots\ ]$ } d dim. → Feature Dimension

↳ 100 / 1000 / ...

Perceptron



$$\sum_{j=1}^{d} w_j x[j] = w_0 + w_1 x[1] + \ldots + w_d x[d]$$

Not-differentiable at 0

Step fn.!!

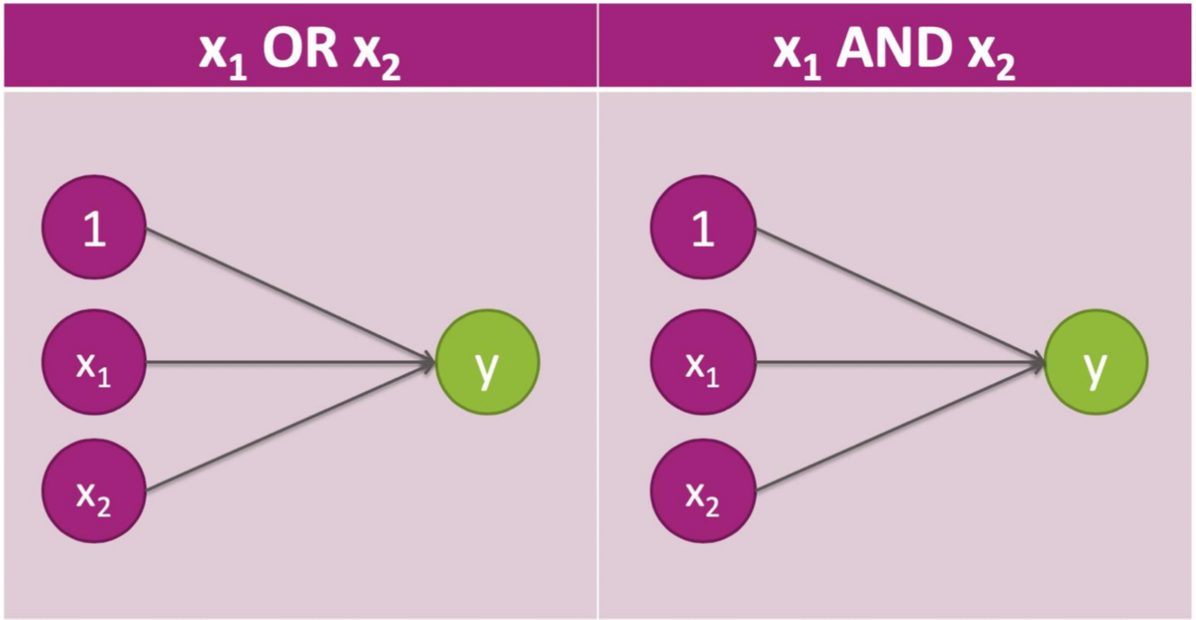
$$g(Score(x)) = \begin{cases} 1, & if \sum_{j=1}^{d} w_j x[j] > 0 \\ 0, & otherwise \end{cases}$$

Transforming fn.

= Activation Function

OR and AND Functions

What can a perceptrons represent?

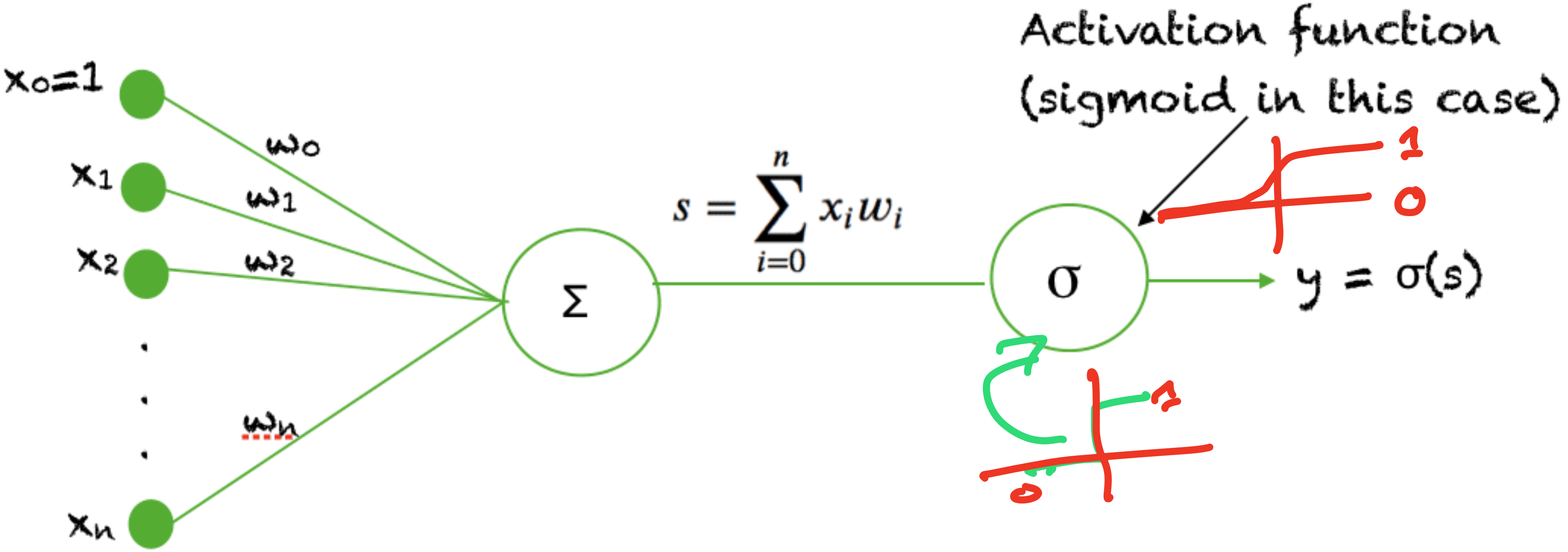


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

LR represented Graphically

Linear Model
OR
AND
Spam Filtering



XOR through Multi-layer perceptron

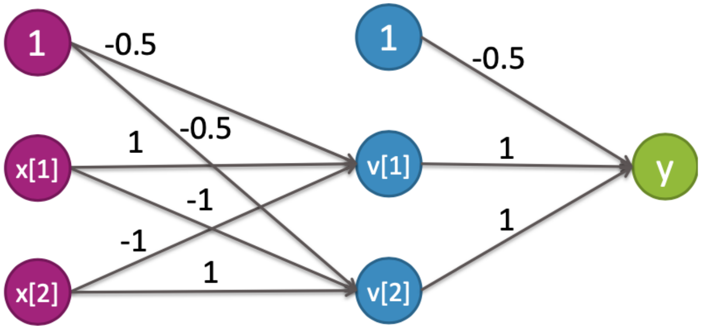
This is a 2-layer neural network

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

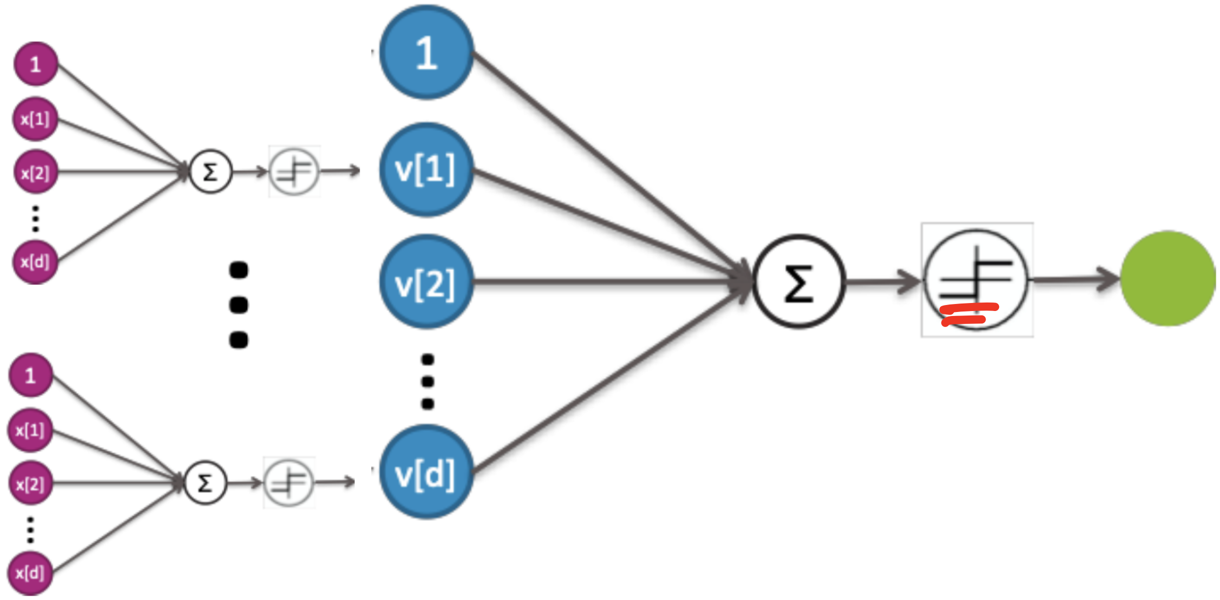
$v[1] = (x[1] \; AND \; ! \, x[2])$
$\quad = g(-0.5 + x[1] - x[2])$

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

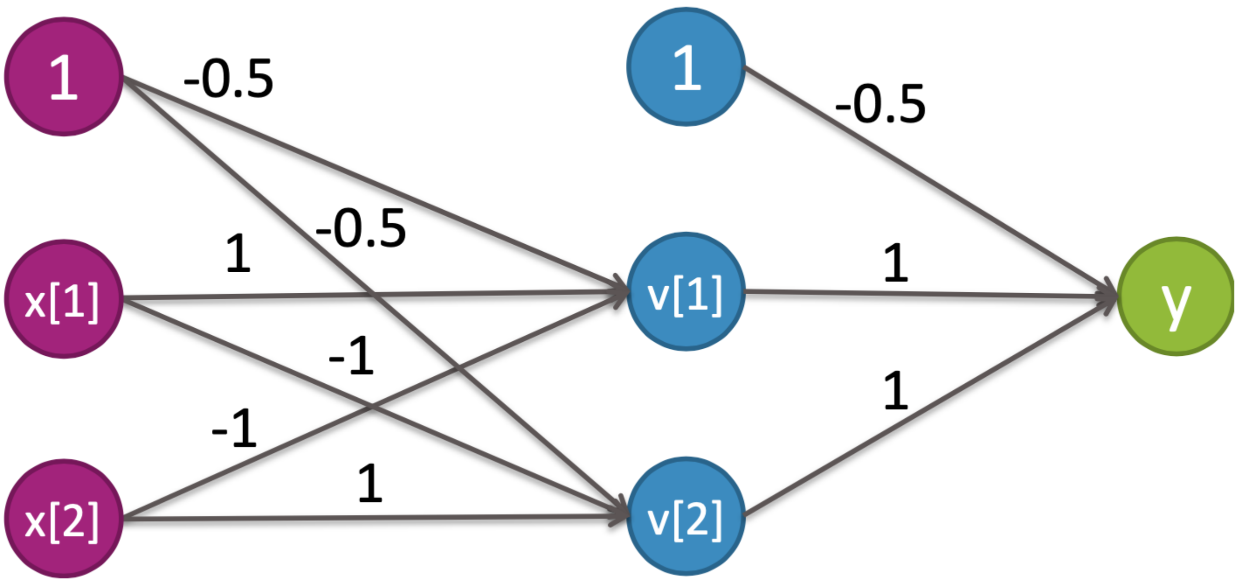
$y = v[1] \; OR \; v[2]$
$\quad = g(-0.5 + v[1] + v[2])$



Multi-Layer Perceptron (MLP)

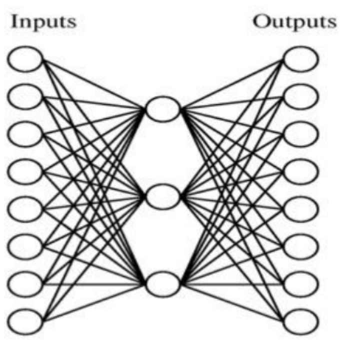


Multi-Layer Perceptron (MLP)



2 Layer Neural Network

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



Single

$$out(x) = g\left(w_0 + \sum_j w_j x[j]\right)$$

1-hidden layer

Activation

$$out(x) = g\left(w_0 + \sum_k w_k g\left(w_0^{(k)} + \sum_j w_j^{(k)} x[j]\right)\right)$$

Linear ≡ 1 layer NN

Deep Learning: Activations, FFN and more

Choices for Non-Linear Activation Function

→ output

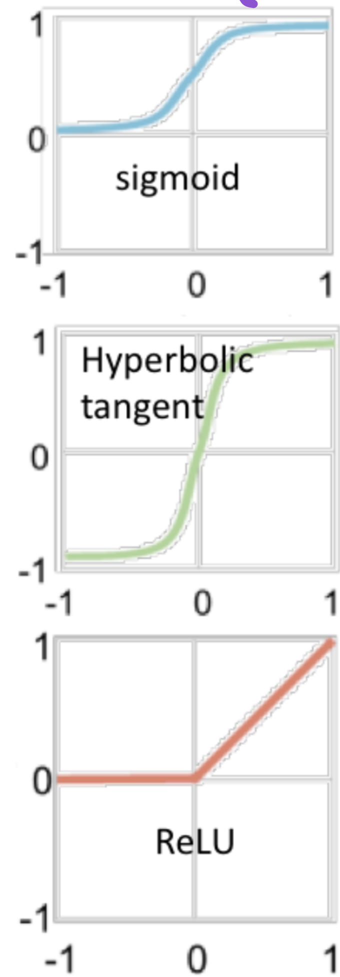
•Sigmoid
-Historically popular, but (mostly) fallen out of favor
•Neuron's activation saturates
(weights get very large -> gradients get small)
•Not zero-centered -> other issues in the gradient steps
-When put on the output layer, called "softmax" because interpreted as class probability (soft assignment)

•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

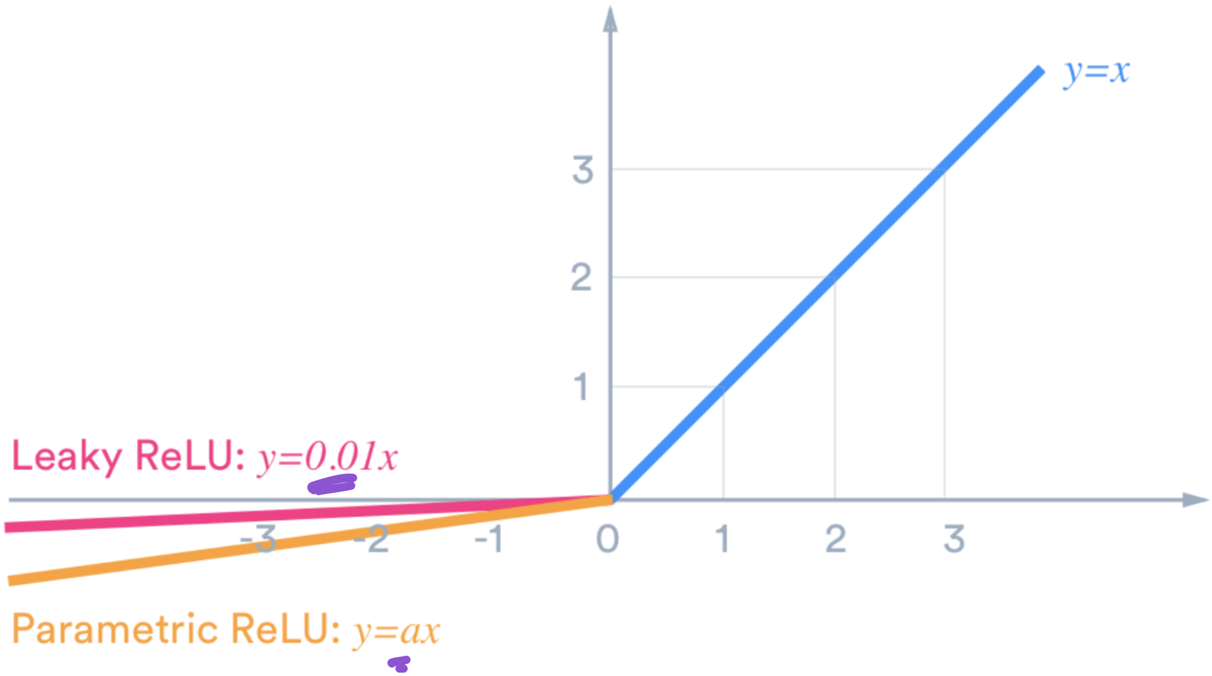
LeakyReLU
SELU

•Softplus $g(x) = \log(1+\exp(x))$
-Smooth approximation to rectifier activation



→ Hidden Layer Activation

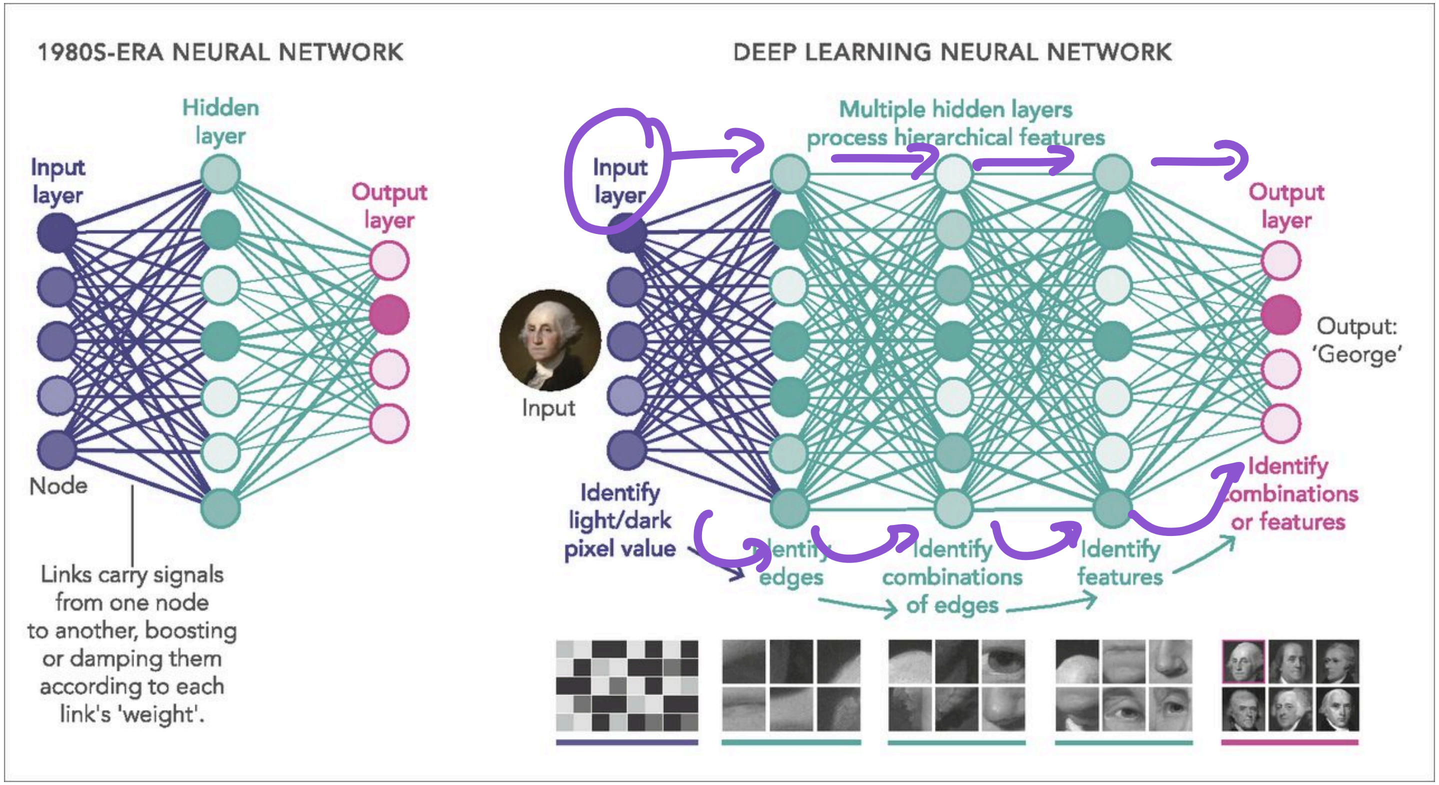
RELU vs Leaky RELU



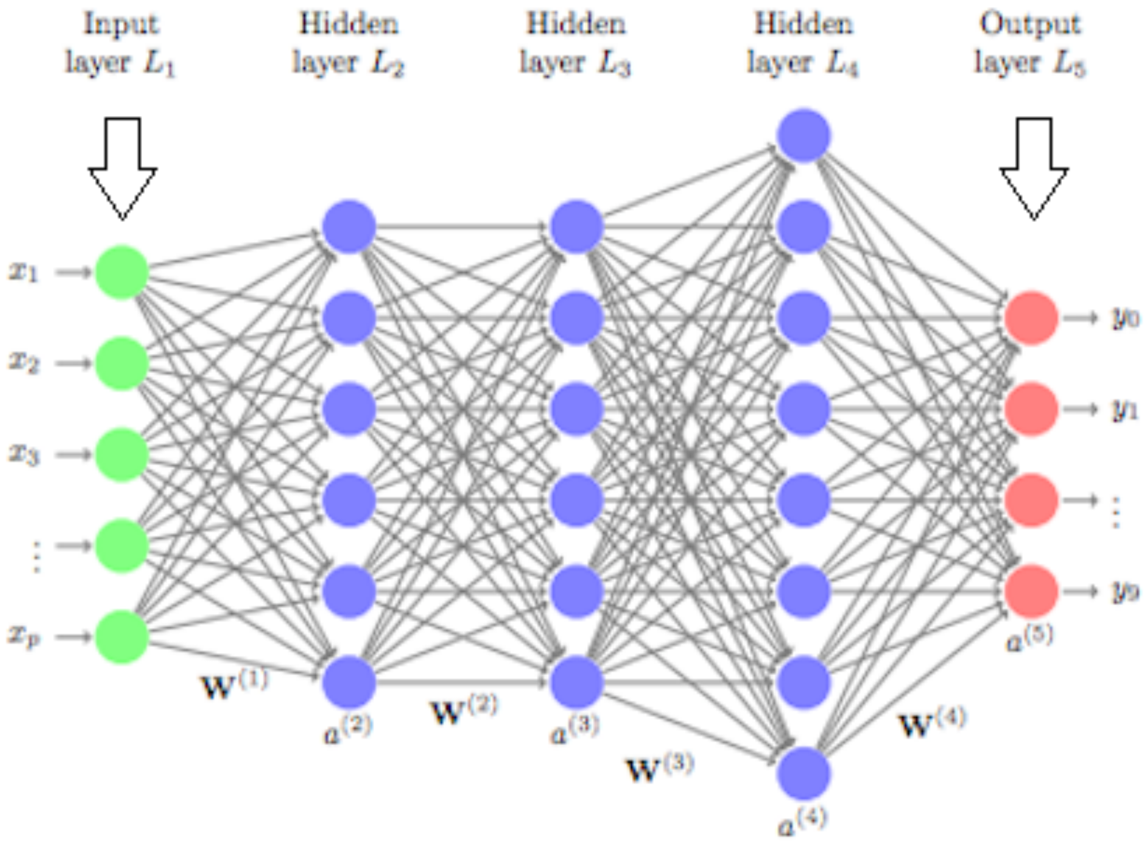
Tensorflow Playground Demo

Tensorflow Playground Demo

Feed-forward Deep Learning Architecture Example



Feed-forward Deep Learning Architecture Example



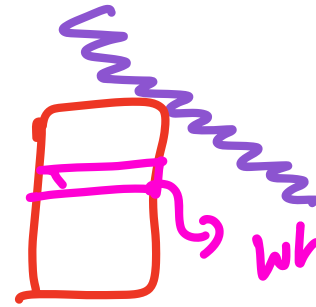
Training a DNN

Stochastic Gradient Descent

ML 2023

SGD with mini-batch

→ mini-batch of data



→ What can I learn from 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.

Update for learning : $w^{t+1} \leftarrow w^t - lr \times \nabla l(w^t)$

Epoch = 1 pass through Data

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?

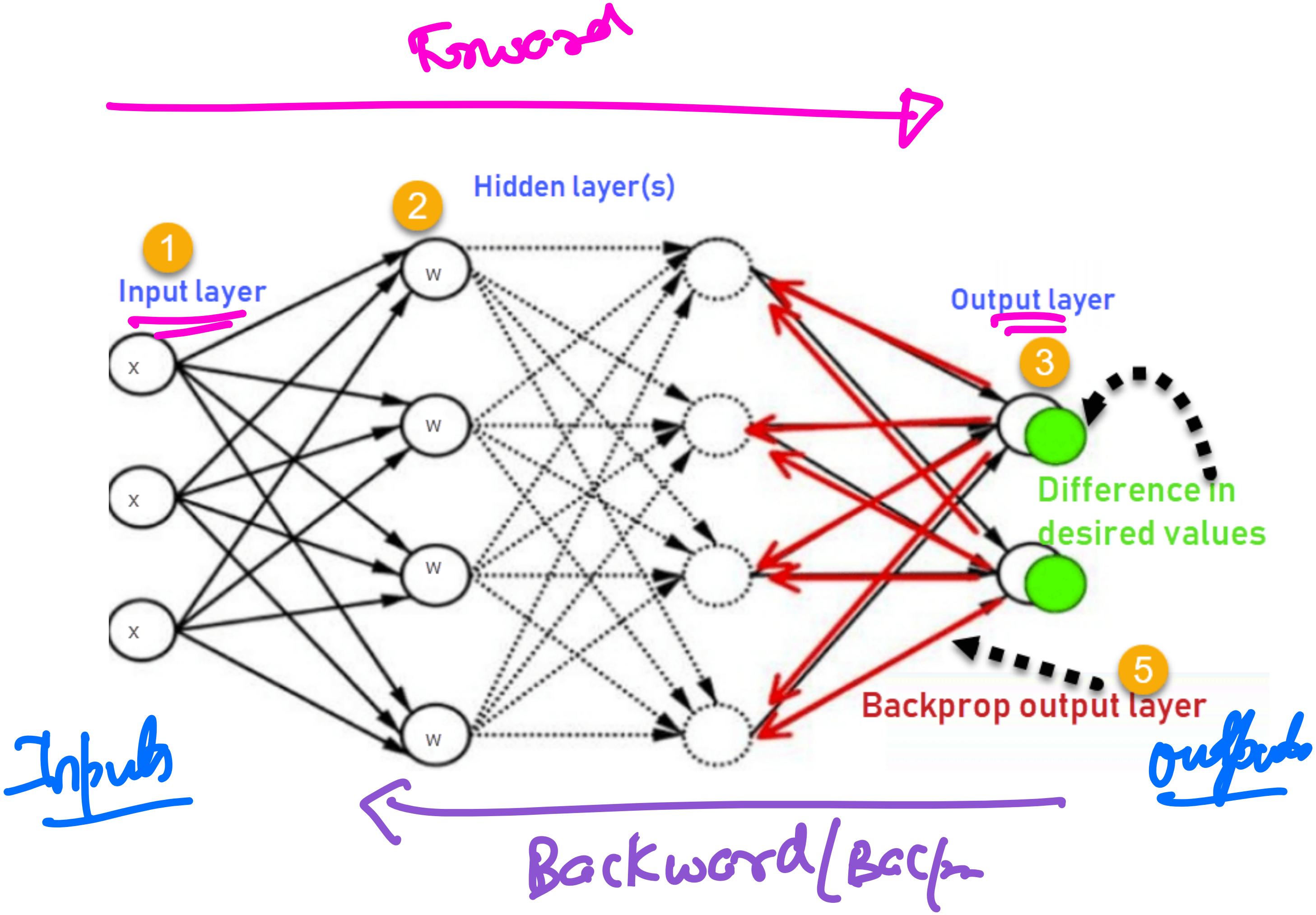
Back-propagation!

$f(x) = x^2 \quad \nabla f = 2x$

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

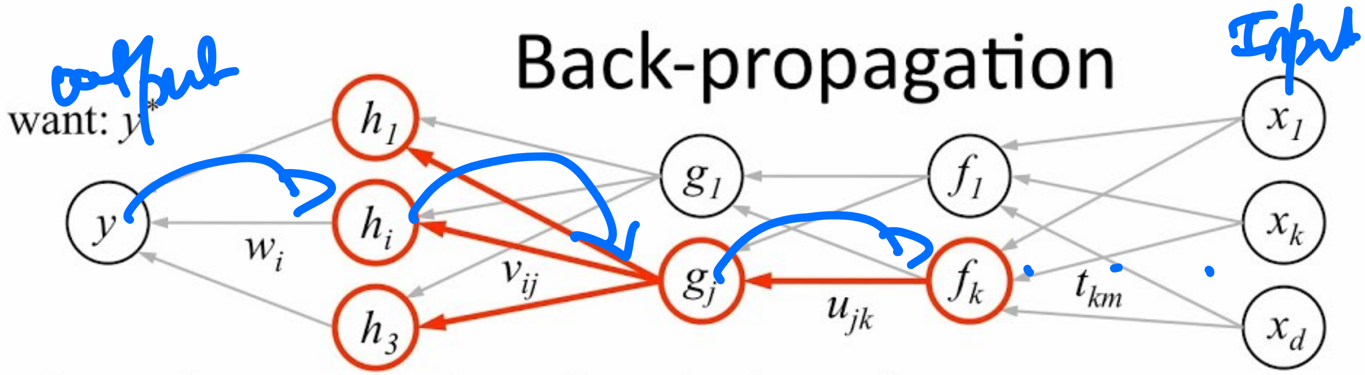
Forward Propagation vs Back-propagation

(Fam)



Back Propagation explained

Back-propagation



1. receive new observation $\boldsymbol{x} = [x_1 \ldots x_d]$ 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 $(y\text{-}y^*)$
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

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

AutoGrad uses BackProp

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

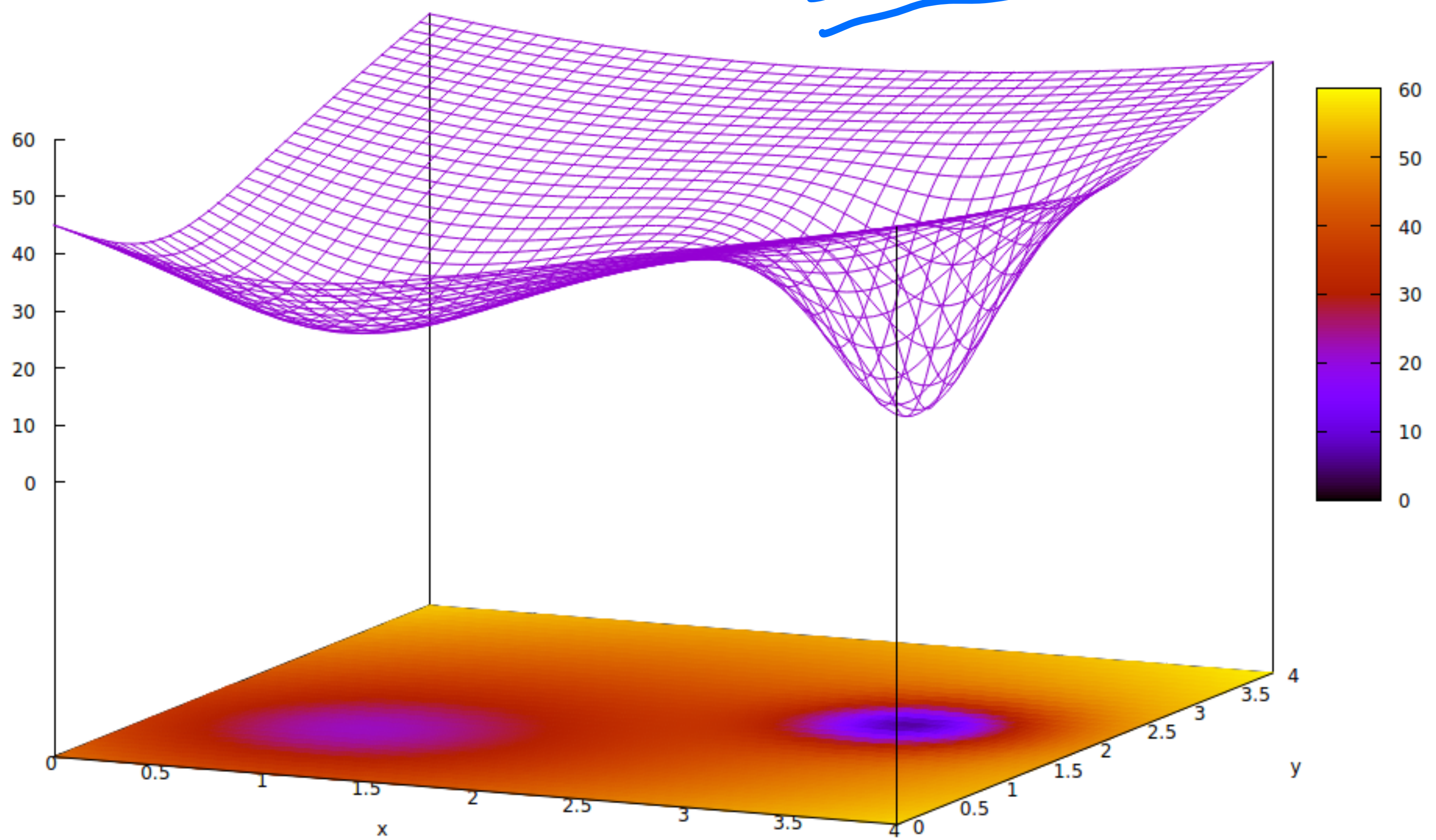
Back prop is one of the fundamental backbones of the training modules behind deep learning and beyond (including for example ChatGPT). What exactly is back prop? It is just a way to unravel gradient computation in the neural network. Back prop is how we would **compute the gradient** in a neural network.

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

SGD → Gets you to good local minima



Hyper-parameters in Deep Learning

ICE #1: 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

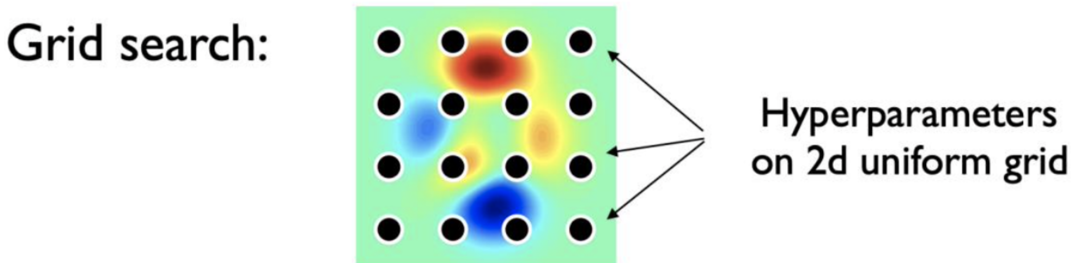
1. Learning rate
2. Number of Hidden Layers
3. Number of neurons per hidden layer
4. Type of non-linear activation function used

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?

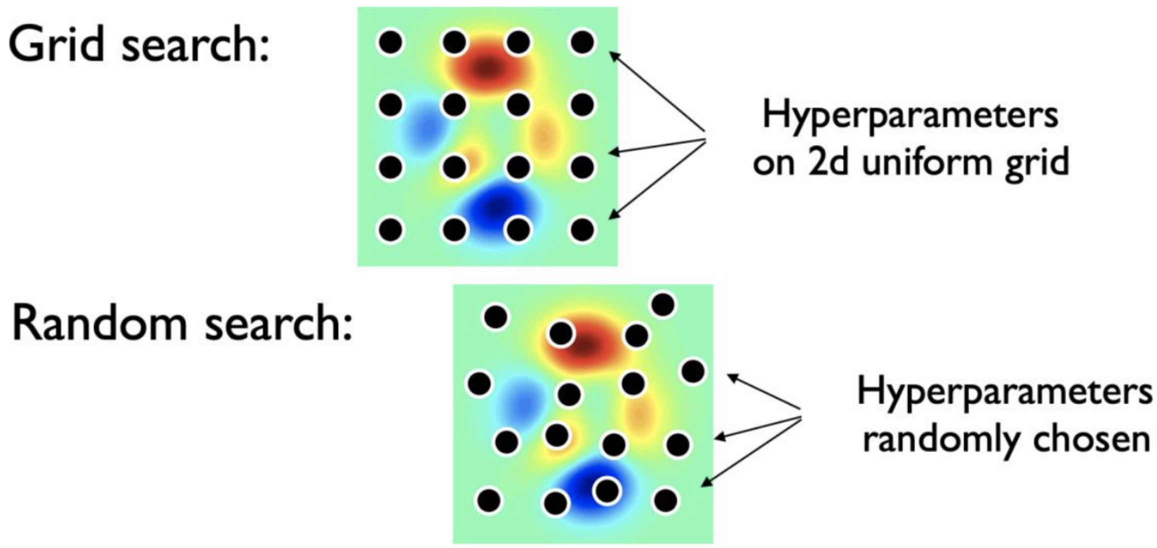
Hyper-parameter tuning methods



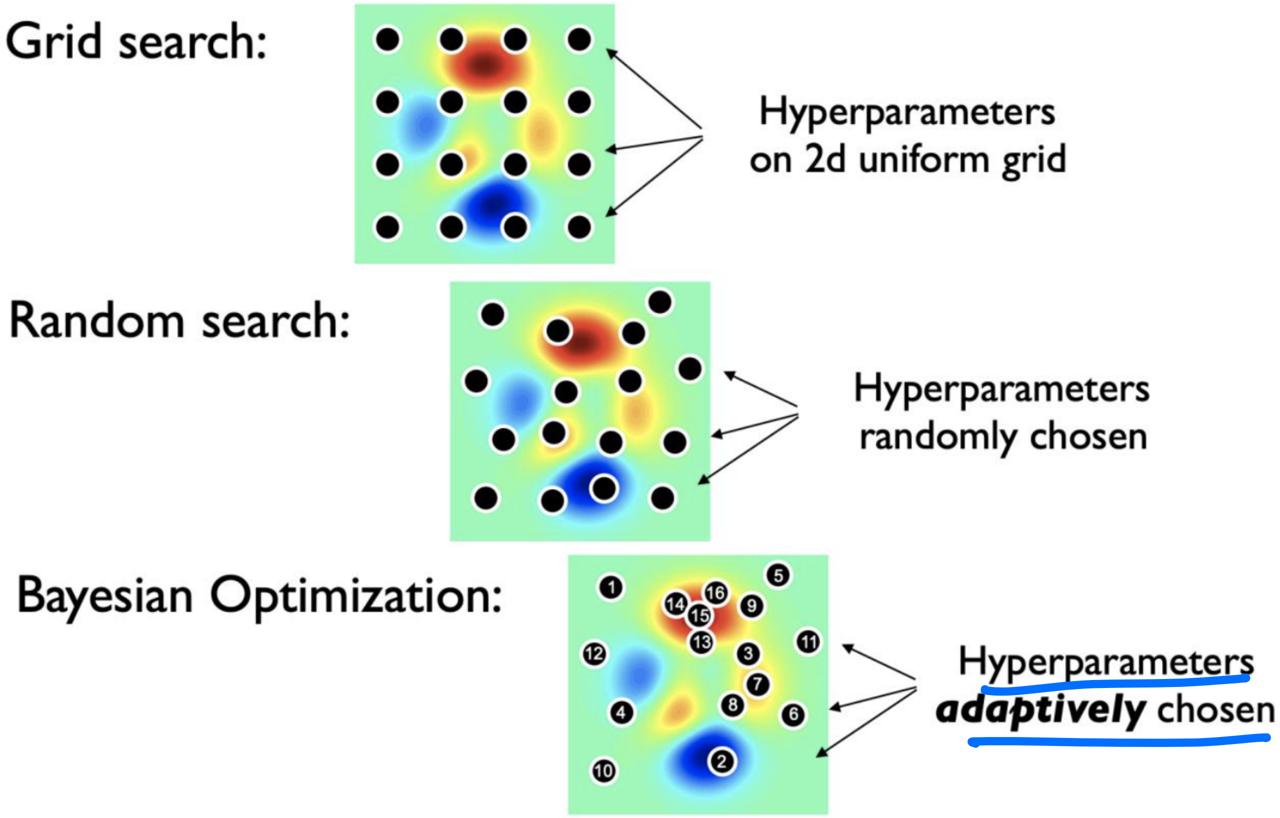
0 ← lr → ∞

0 0.001 0.1 1 10 100

Hyper-parameter tuning methods



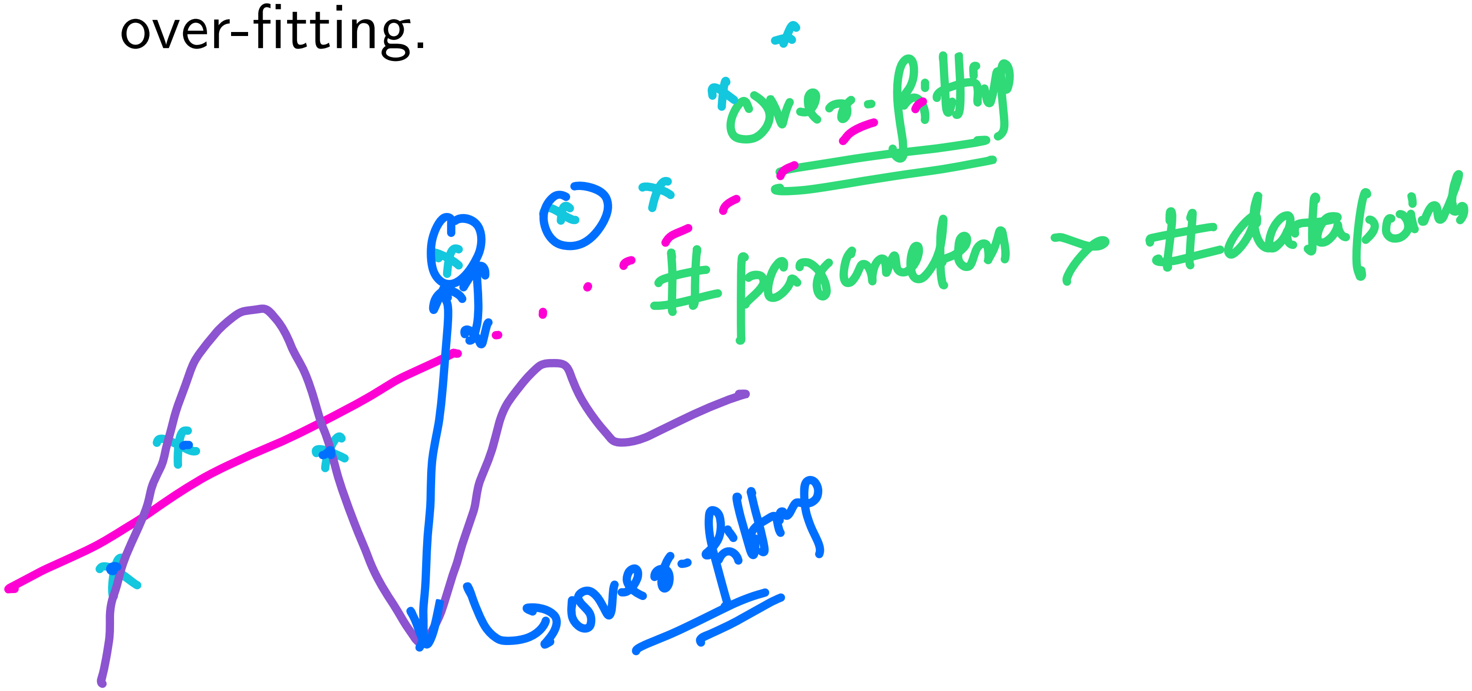
Hyper-parameter tuning methods



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 - 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

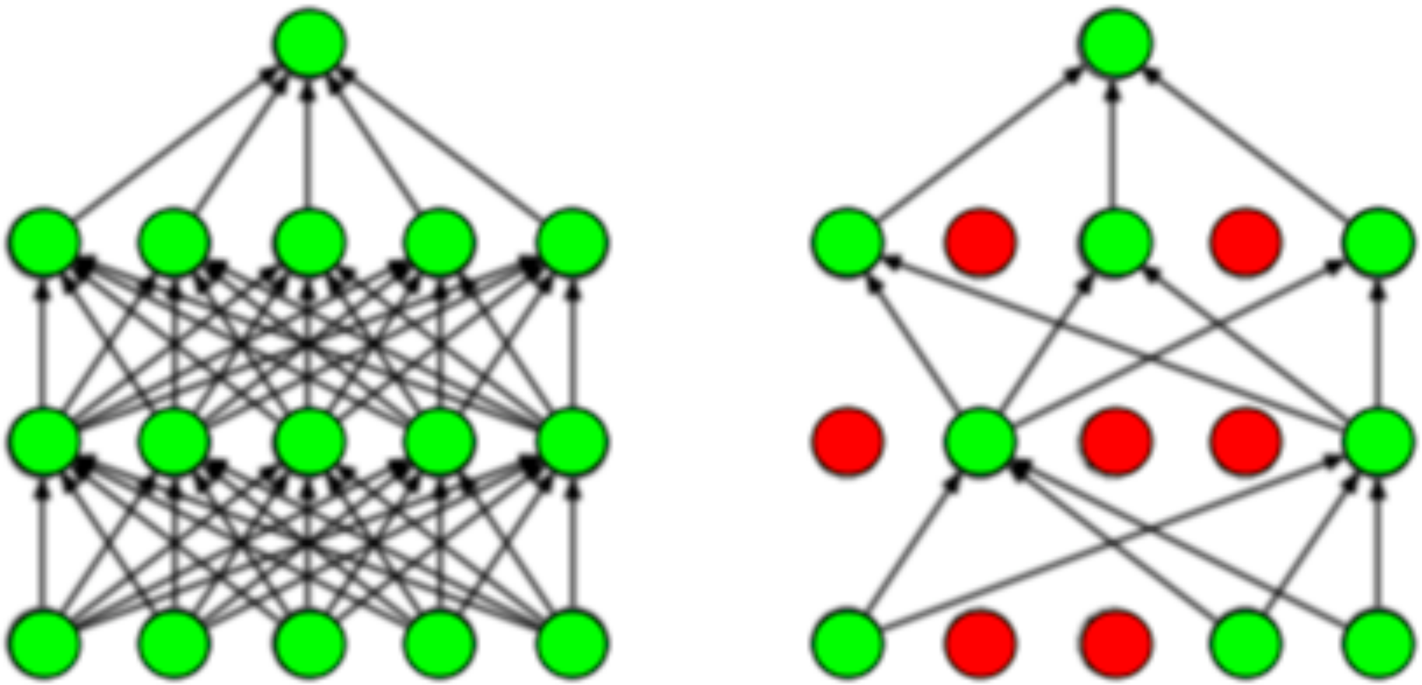
1. A DNN model with 100 million parameters and only 100k data points or even a million data points will overfit unless we take care of over-fitting.
2. Weight regularization can help - ℓ_1, ℓ_2
3. More common over-fitting strategy for DL?
4. Dropouts!
5. Early stopping is also a great strategy! Stop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

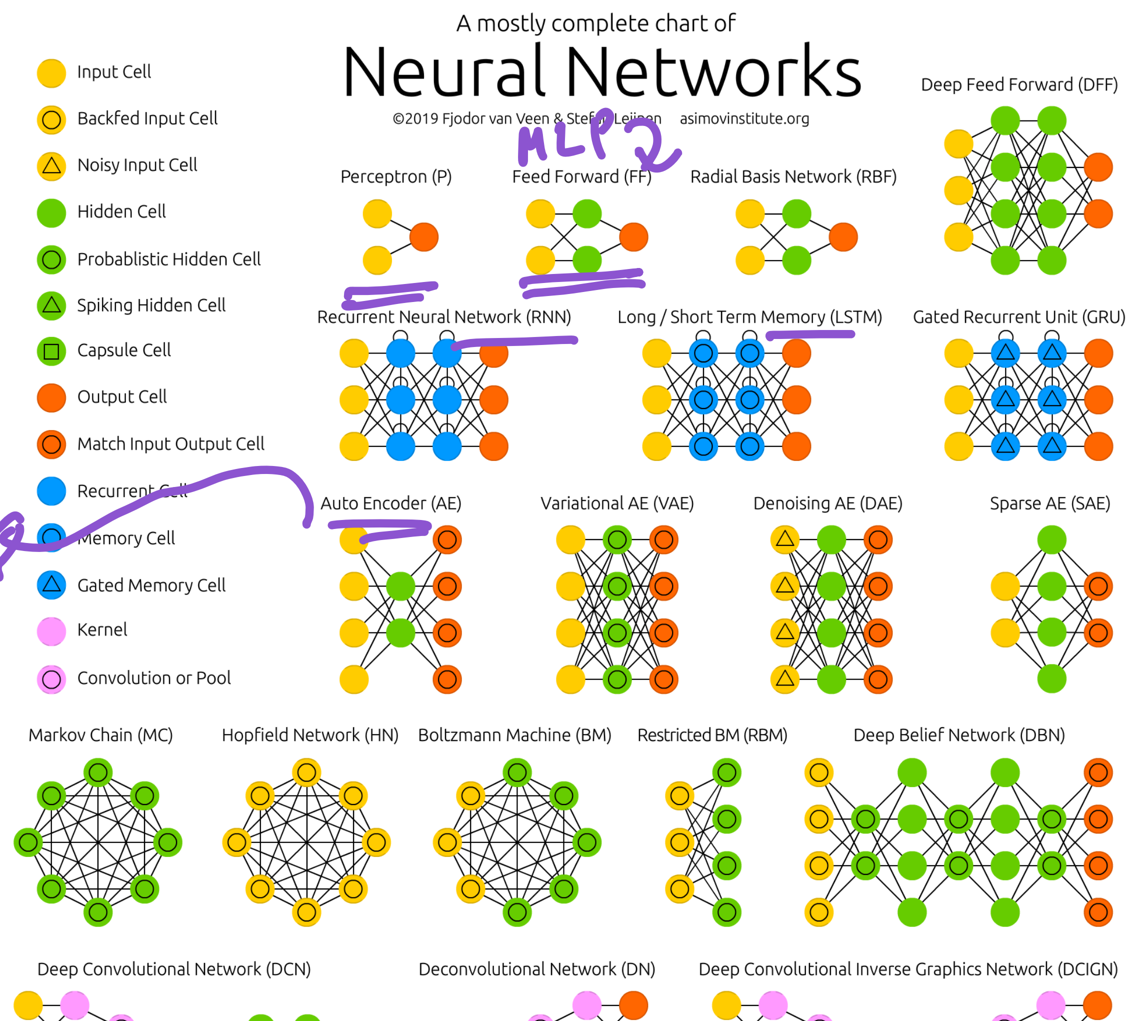
Tensorflow Playground Demo

Tensorflow Playground Demo

Neural Networks Zoo

Zoo Reference

RNN & LSTM
— First AI models in "Generative AI"



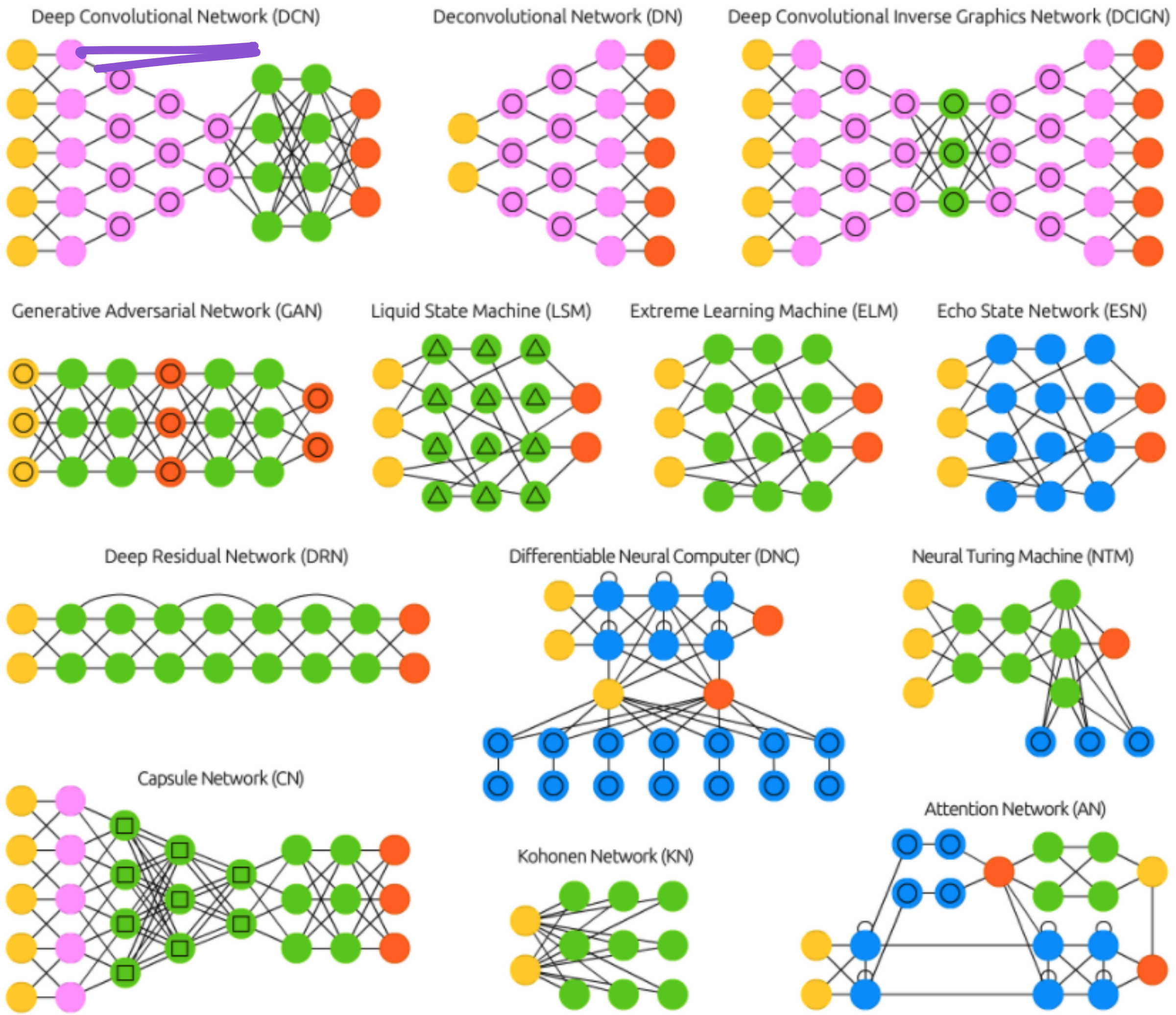
MLP 2

Stable Diffusion

More DL Architectures

Neural Networks Zoo

→ Image CNN



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! $\rightarrow$ Positive Sentiment

Sequence structure in NLP

Example

I love this car! Positive Sentiment

Example

I am not sure I love this car! Negative Sentiment

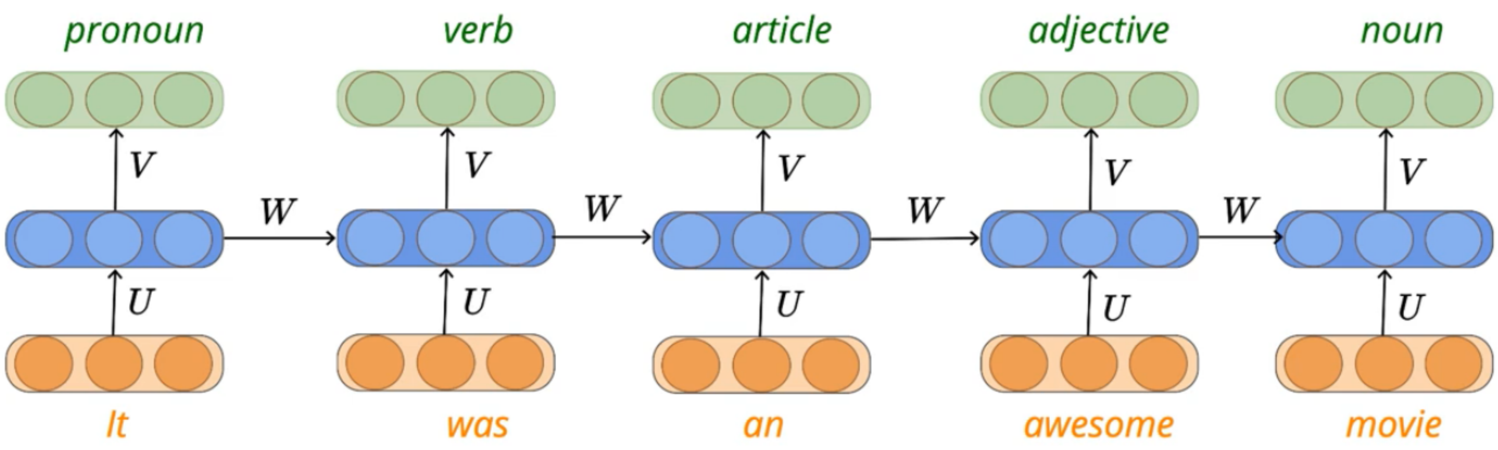
Example

I don't think its a bad car at all! $\rightarrow$ Positive Sentiment

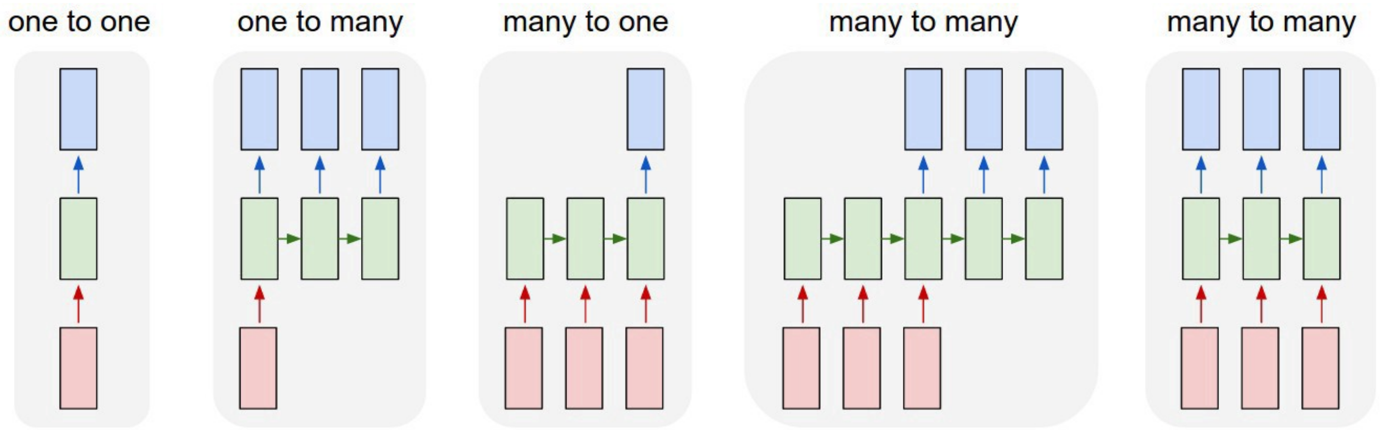
Example

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

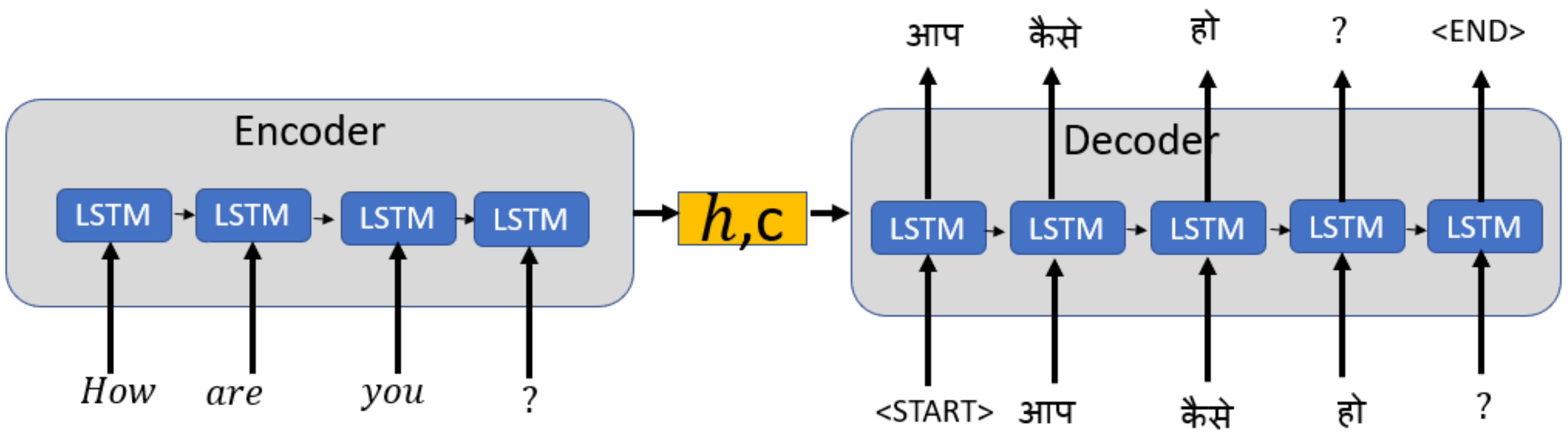
Sequence to Sequence Model (LSTM) Applications



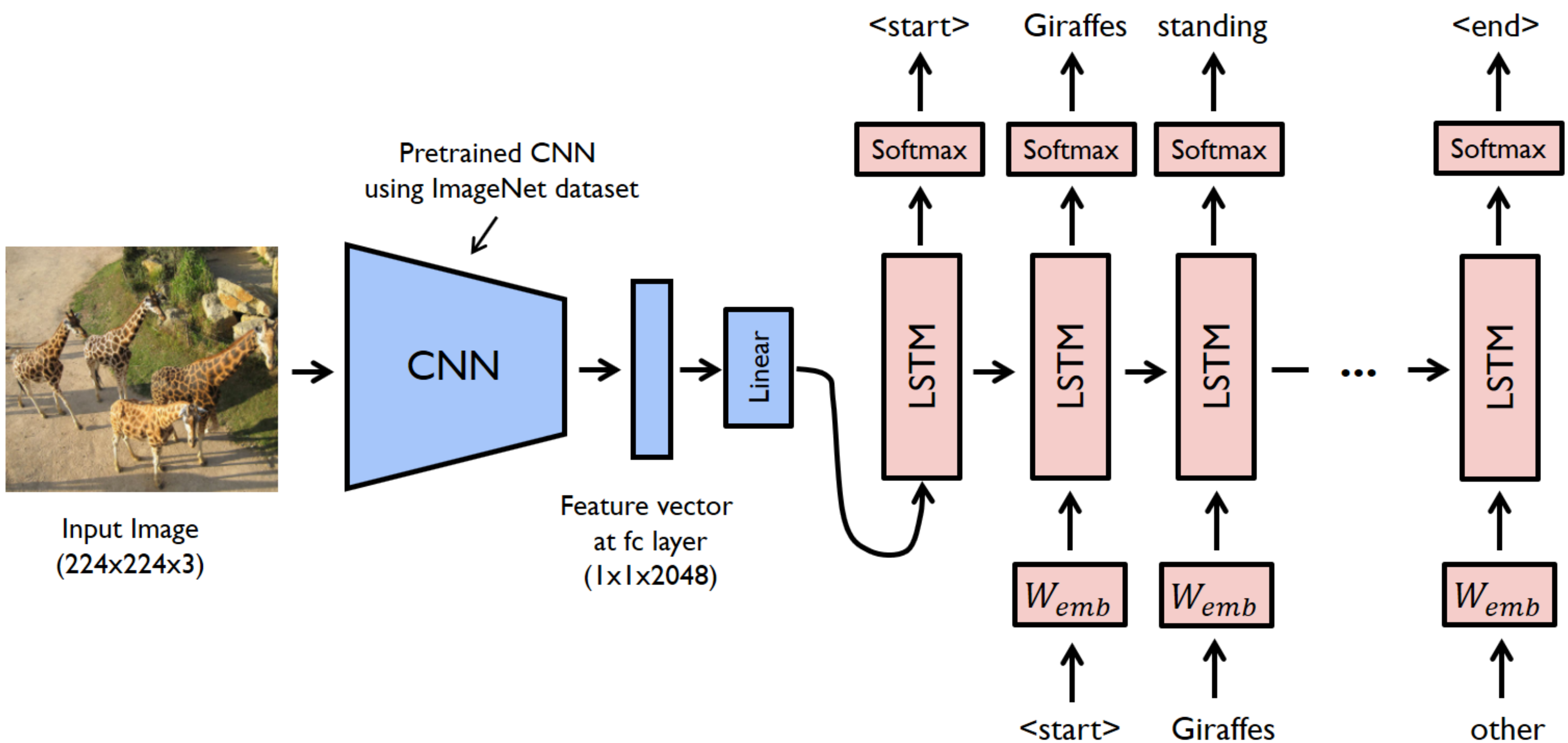
Sequence to Sequence Model (LSTM) Applications



Sequence to Sequence Model (LSTM) Applications



Sequence to Sequence Model (LSTM) Applications



Breakouts Time #1

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?