[course site]

#DLUPC

Day 2 Lecture 4

Multilayer Perceptrons MLP

Xavier Giro-i-Nieto
xavier.giro@upc.edu

Associate Professor
Universitat Politècnica de Catalunya
Technical University of Catalonia

Acknowledgements

Kevin McGuinness
kevin.mcguinness@dcu.ie

Research Fellow
Insight Centre for Data Analytics
Dublin City University

Eva Mohedano
eva.mohedano@insight-centre.org

Postdoctoral Researcher
Insight Centre for Data Analytics
Dublin City University

Elisa Sayrol
elisa.sayrol@upc.edu

Associate Professor
ETSETB TelecomBCN
Universitat Politècnica de Catalunya

Antonio Bonafonte
antonio.bonafonte@upc.edu

Associate Professor
ETSETB TelecomBCN
Universitat Politècnica de Catalunya

Video lecture

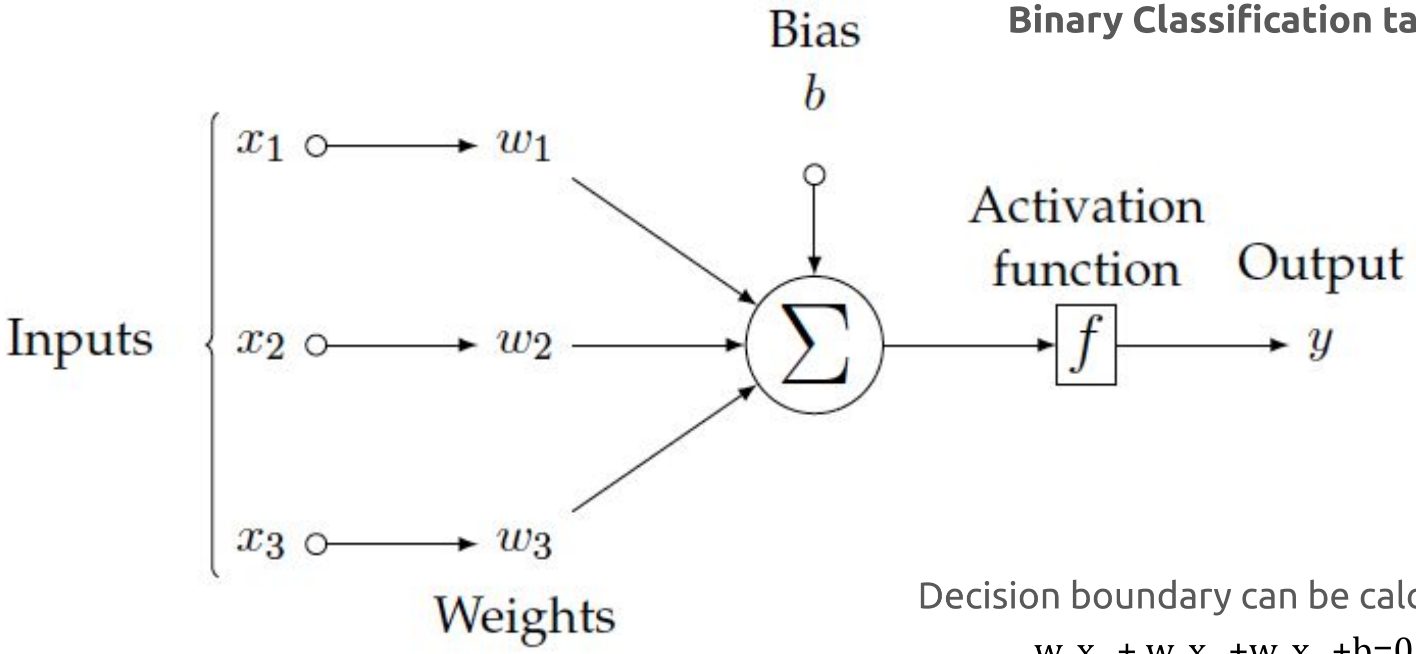
Overview

- **Limitations the perceptron**
- Multilayer perceptrons
- Dynamic Programming
- Implementation details

Single Neuron

If the weighted sum of the input exceeds a threshold the neuron fires a signal.

Binary Classification task

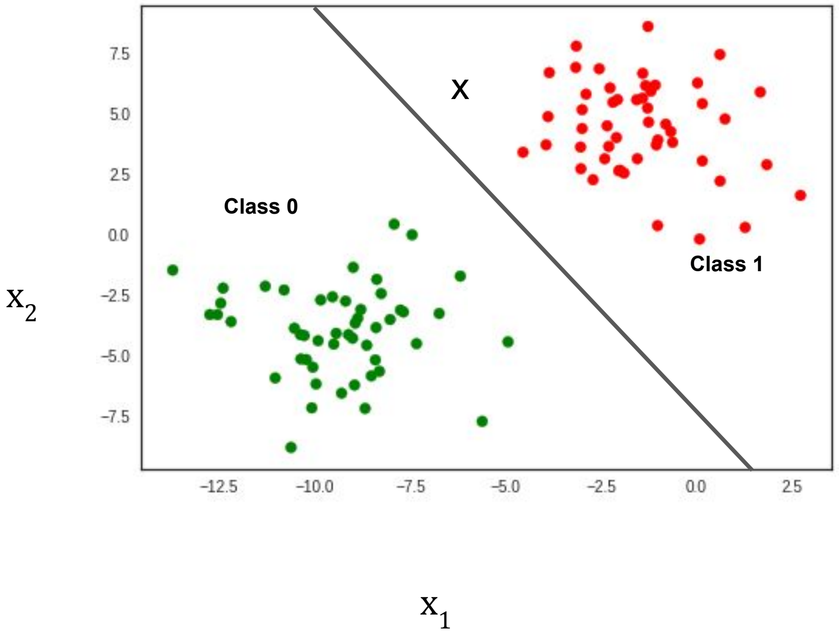


Decision boundary can be calculated by:

$$w_1x_1 + w_2x_2 + w_3x_3 + b = 0$$

Linear decision decision boundary

2D input space data



$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

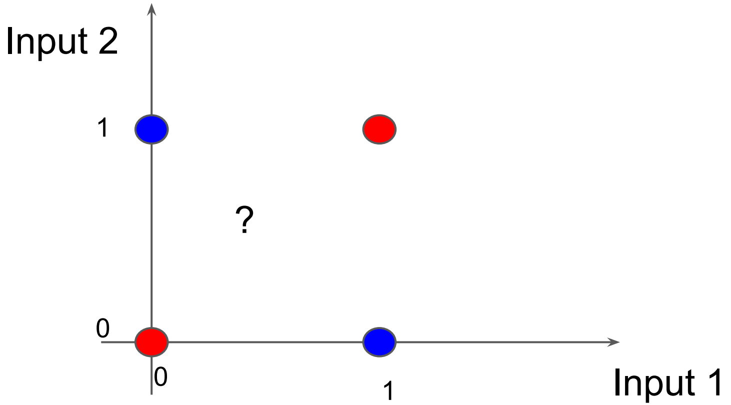
The XOR problem

Limitation: Data might be **non linearly separable**

→ One single neuron is not enough

XOR logic table

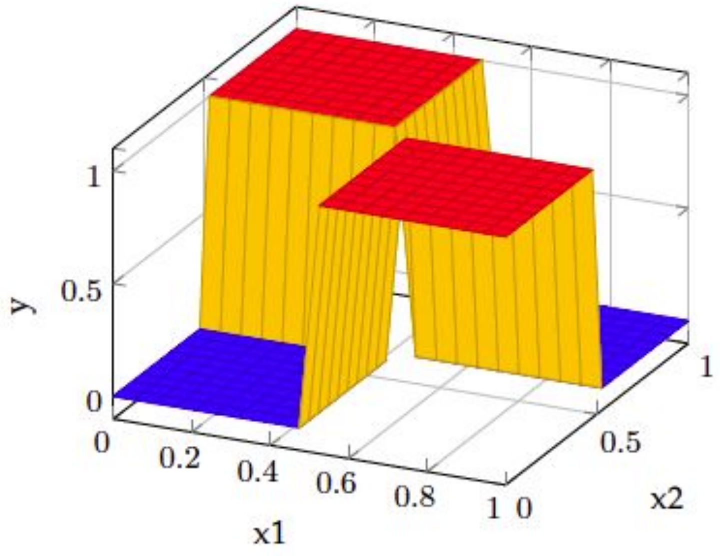
Input 1	Input 2	Desired Output
0	0	0
0	1	1
1	0	1
1	1	0

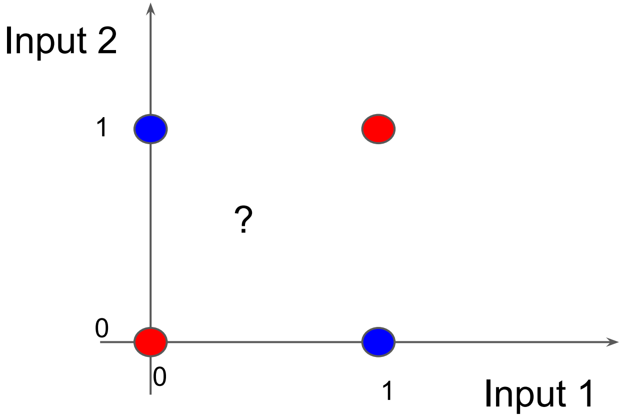


The XOR problem

Limitation: Data might be **non linearly separable**

→ One single neuron is not enough



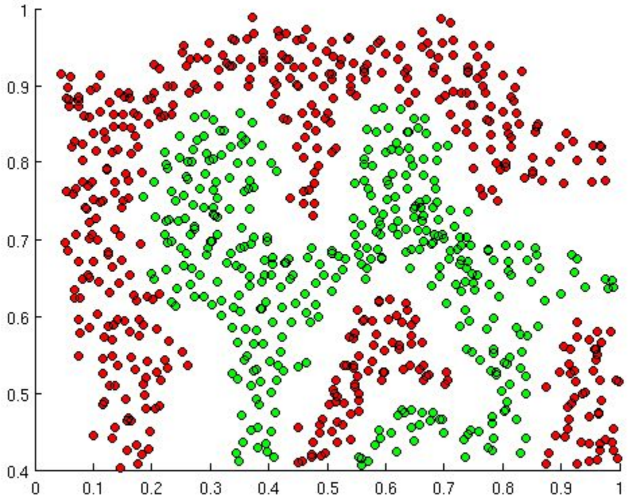


Non-linear decision boundaries

Linear models can only produce linear decision boundaries

Real world data often needs a non-linear decision boundary

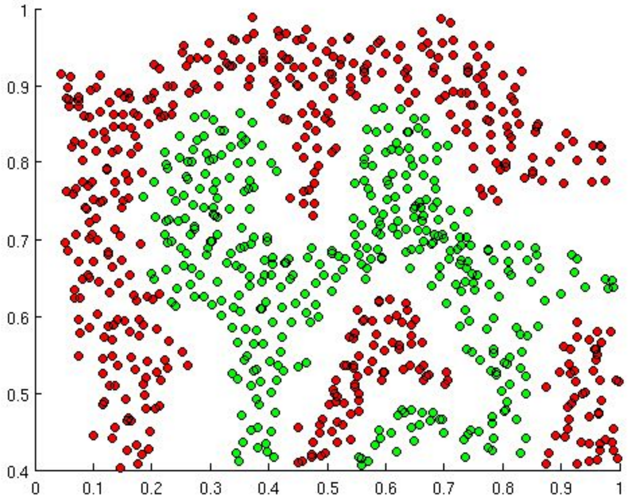
- Images
- Audio
- Text



Non-linear decision boundaries

What can we do?

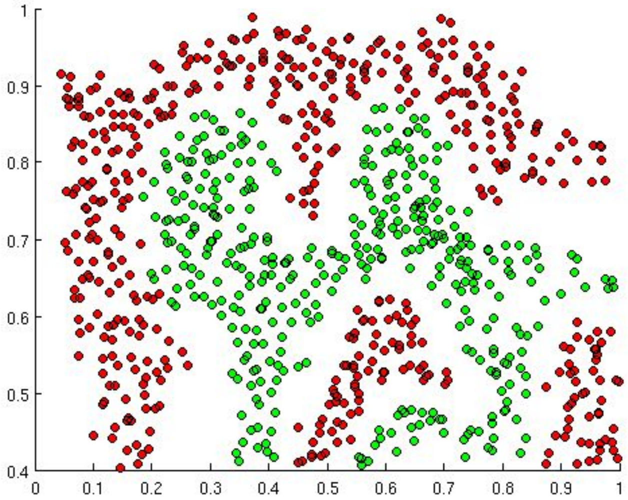
1. Use a non-linear classifier
   - Decision trees (and forests)
   - K nearest neighbours
2. Engineer a suitable representation
   - One in which features are more linearly separable
   - Then use a linear model
3. Engineer a kernel
   - Design a kernel $\mathbf{K(x_1, x_2)}$
   - Use kernel methods (e.g. SVM)
4. Learn a suitable representation space from the data
   - Deep learning, deep neural networks
   - Boosted cascade classifiers like Viola Jones also take this approach



Non-linear decision boundaries

What can we do?

1. Use a non-linear classifier
   - Decision trees (and forests)
   - K nearest neighbours
2. Engineer a suitable representation
   - One in which features are more linearly separable
   - Then use a linear model
3. Engineer a kernel
   - Design a kernel $\mathbf{K(x_1, x_2)}$
   - Use kernel methods (e.g. SVM)
4. Learn a suitable representation space from the data
   - **Deep learning, deep neural networks**
   - Boosted cascade classifiers like Viola Jones also take this approach



Overview

- Limitations the perceptron
- **Multilayer Perceptrons**
- Dynamic Programming
- Implementation details

Multilayer Perceptrons - MLPs

When each node in each layer is a linear combination of **all inputs from the previous layer** then the network is called a multilayer perceptron (MLP)

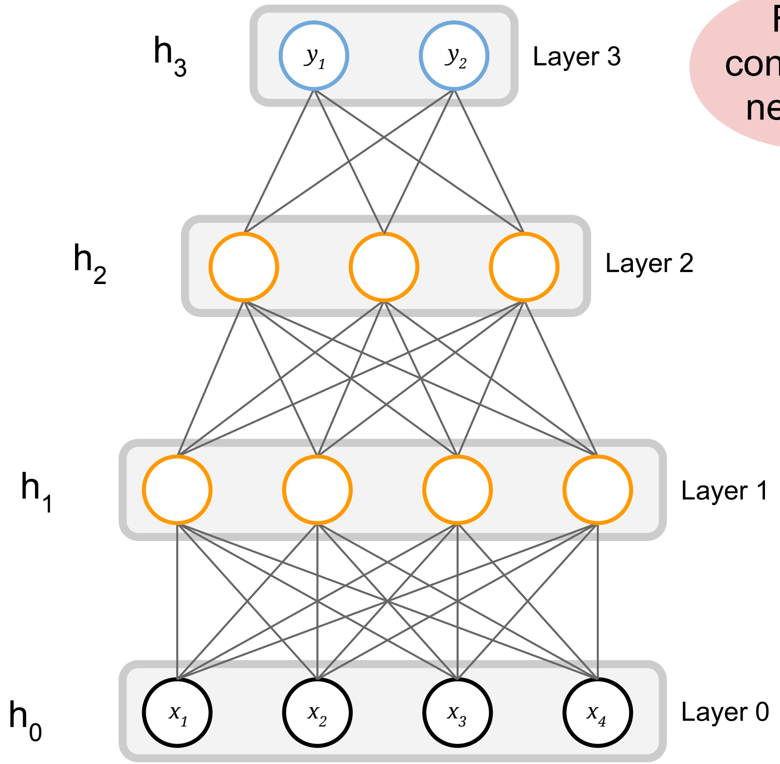
Weights can be organized into matrices.

Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$



Fully connected network

Multilayer Perceptrons - MLPs

W_1

w_{11}	w_{12}	w_{13}	w_{14}
w_{21}	w_{22}	w_{23}	w_{24}
w_{31}	w_{32}	w_{33}	w_{34}
w_{41}	w_{42}	w_{43}	w_{44}

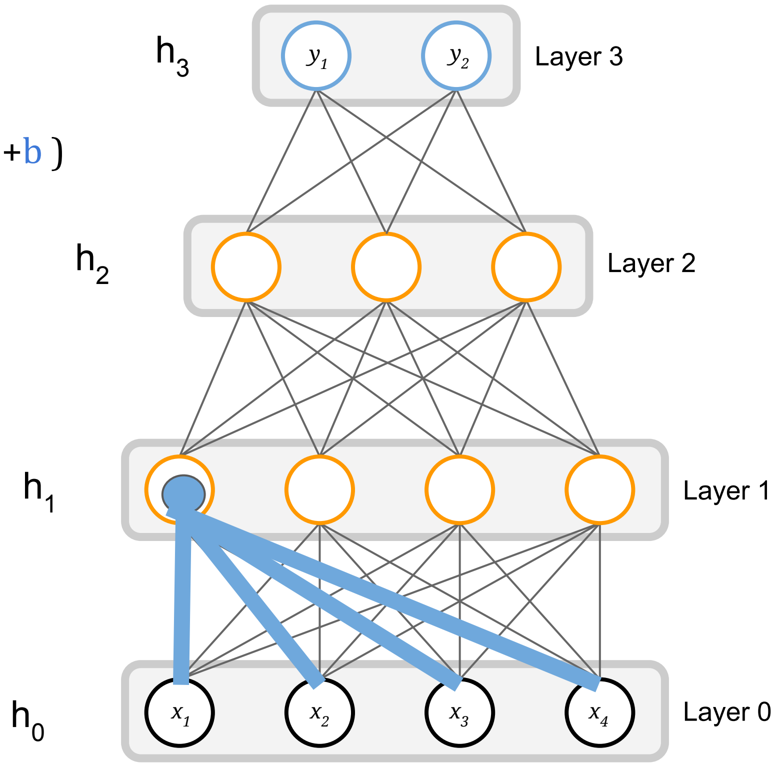
h_0

x_1
x_2
x_3
x_4

b_1

b_1
b_2
b_3
b_4

$h_{11} = g(\mathbf{w}\mathbf{x} + b)$



Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

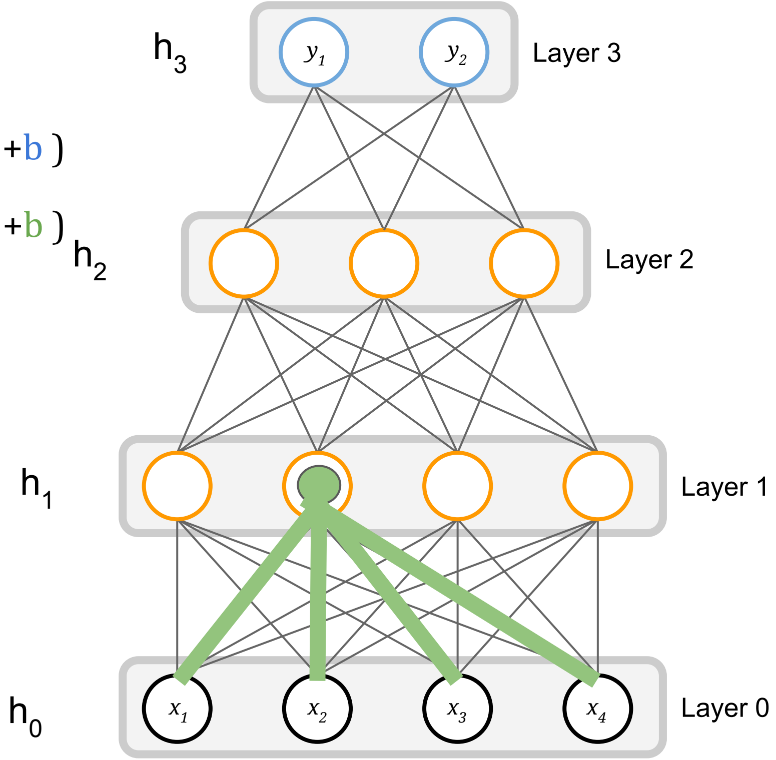
$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$

Multilayer Perceptrons - MLPs

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \quad h_0 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad b_1 = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$$h_{11} = g(\mathbf{w}\mathbf{x} + b)$$

$$h_{12} = g(\mathbf{w}\mathbf{x} + b)$$



Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$

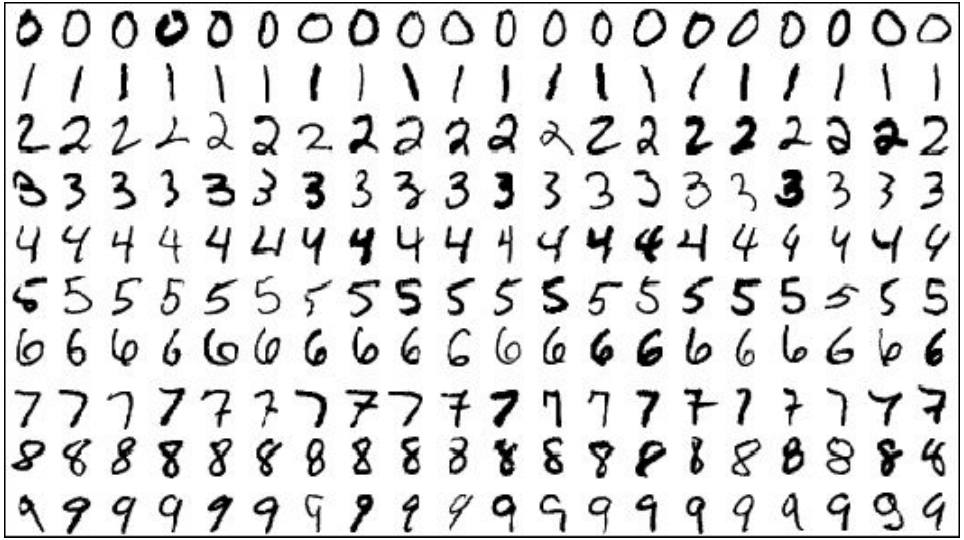
Task: MNIST digit classification

MNIST

- Popular dataset of handwritten digits
- 60,000 training examples
- 10,000 test examples
- 10 classes (digits 0-9)
- http://yann.lecun.com/exdb/mnist/
- 28x28 grayscale images (784D)

Objective

- Learn a function y = f(x) that predicts the digit from the image
- Measure accuracy on test set

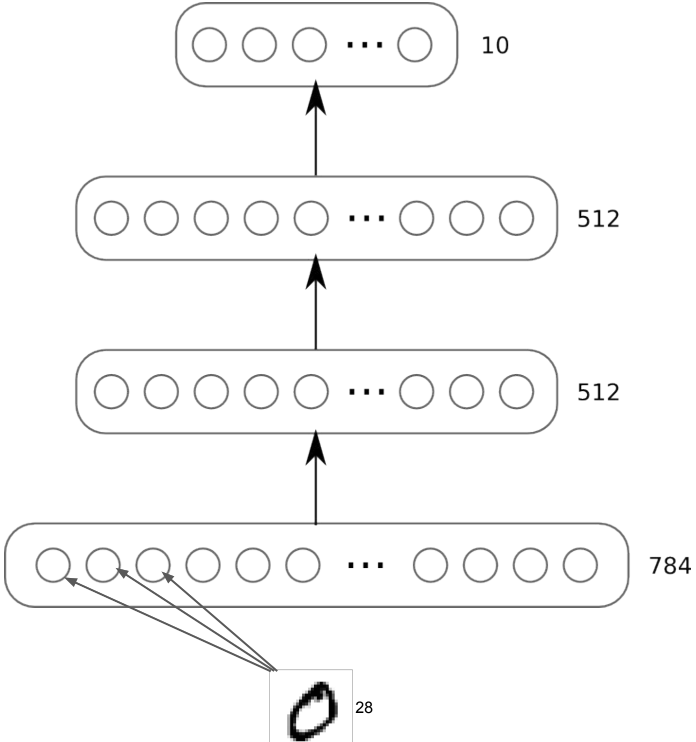


Multilayer Perceptrons - MLPs

How many parameters contains the following MLP ?

Model

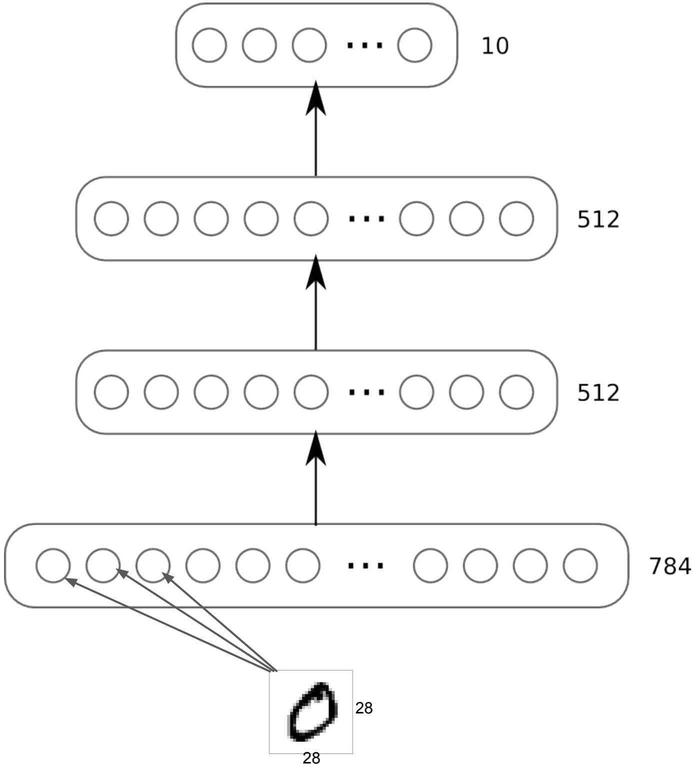
- 3 layer neural network (2 hidden layers)
- Tanh units (activation function)
- 512-512-10
- **Softmax** on top layer



Multilayer Perceptrons - MLPs

How many parameters contains the following MLP ?

Layer	#Weights	#Biases	Total
1	784 x 512	512	401,920
2	512 x 512	512	262,656
3	512 x 10	10	5,130
			669,706



Multilayer Perceptrons - MLPs

Which of the statements below is true?

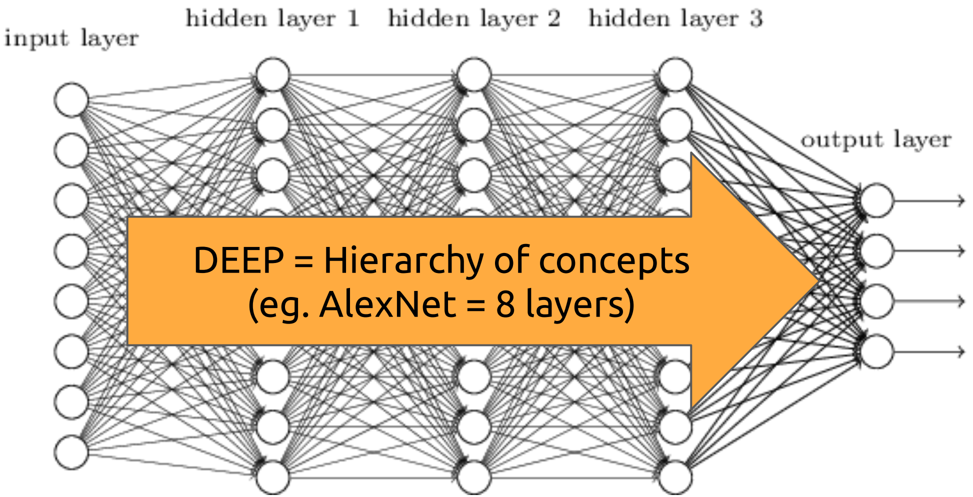
A. Multi-layer perceptrons and CNNS are the same kind of networks
B. Each node in a given layer is connected to all inputs from the previous layer
C. In multi-layer perceptrons, the hidden layers are connected only to the input layer
D. There are no hidden layers in the Multi-layer perceptron only in deep networks

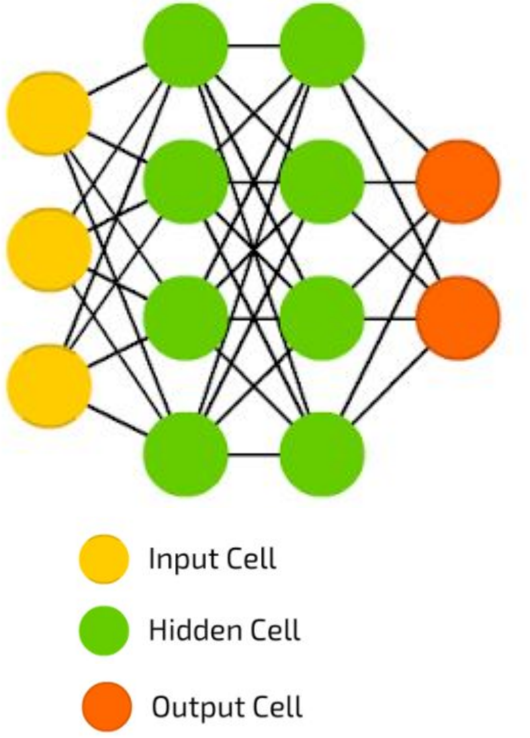
Multilayer Perceptrons - MLPs

Which of the statements below is true?

A. Multi-layer perceptrons and CNNS are the same kind of networks
B. Each node in a given layer is connected to all inputs from the previous layer
C. In multi-layer perceptrons, the hidden layers are connected only to the input layer
D. There are no hidden layers in the Multi-layer perceptron only in deep networks

Deep Neural Networks (DNN)





F. Van Veen, [“The Neural Network Zoo”]() (2016)

Neural Networks
Volume 2, Issue 5, 1989, Pages 359-366

Original contribution

Multilayer feedforward networks are universal approximators

Kurt Hornik, Maxwell Stinchcombe, Halbert White [1]

⊞ Show more

https://doi.org/10.1016/0893-6080(89)90020-8

Abstract

This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.

ELSEVIER

Neural Networks
Volume 2, Issue 5, 1989, Pages 359-366

Original contribution

Multilayer feedforward networks are universal approximators

Kurt Hornik, Maxwell Stinchcombe, Halbert White [1]

Show more

https://doi.org/10.1016/0893-6080(89)90020-8

Get rights and content

Abstract

This paper rigorously establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.

- Needs a “finite number of hidden neurons”: finite may be extremely large
- How to find the parameters (weights, biases) of these neurons ?

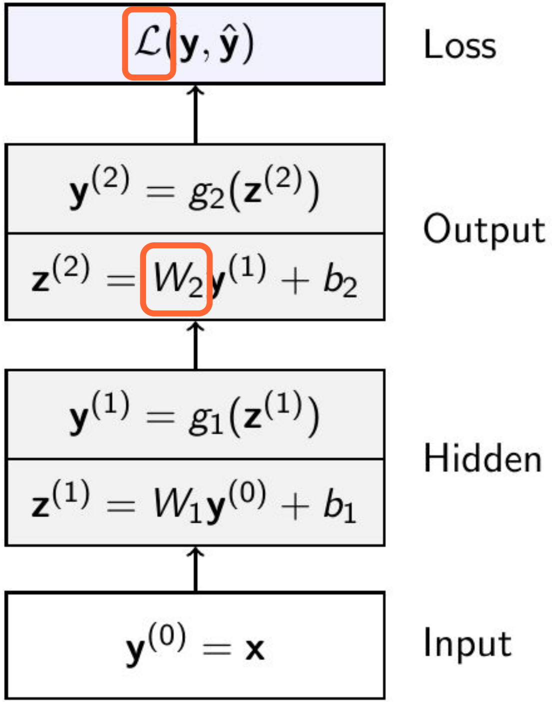
Overview

- Limitations the perceptron
- Multilayer Perceptrons
- **Dynamic Programming**
- Implementation details

Dynamic programming

Need to calculate gradients wrt. parameters:

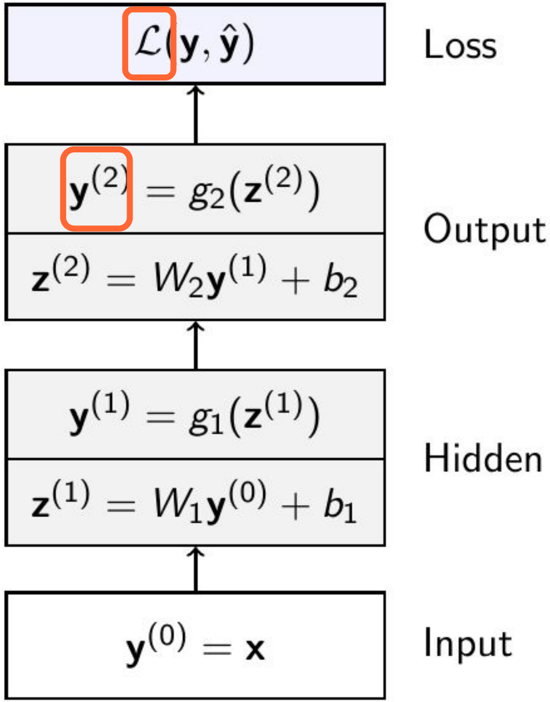
$$\boxed{\frac{\partial \mathcal{L}}{\partial W_2}} = \quad \ldots ?$$



Dynamic programming

Need to calculate gradients wrt. parameters:

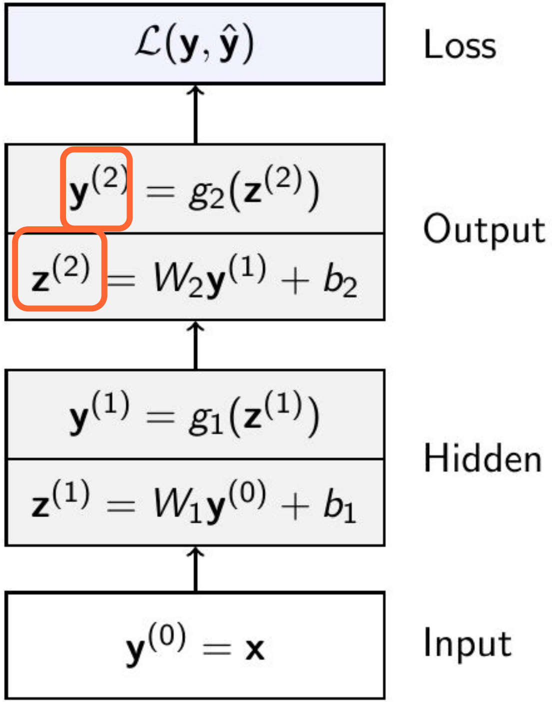
$$\frac{\partial \mathcal{L}}{\partial W_2} = \boxed{\left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \right.} \quad \ldots ?$$



Dynamic programming

Need to calculate gradients wrt. parameters:

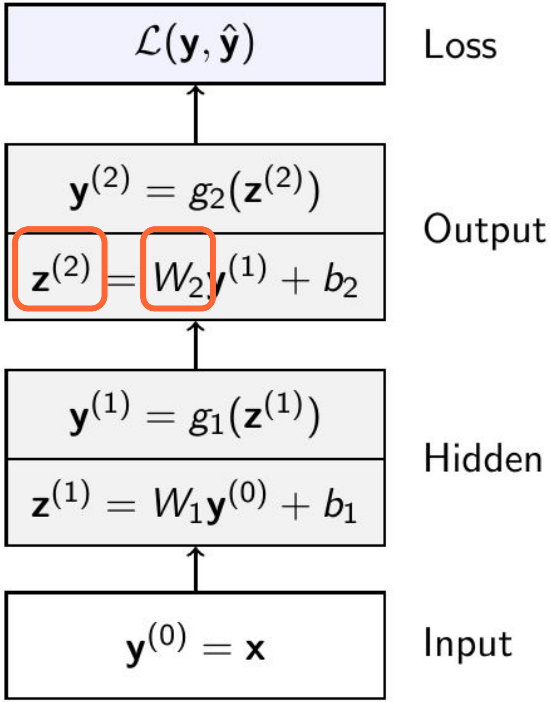
$$\frac{\partial \mathcal{L}}{\partial W_2} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \boxed{\frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}}} \right) \ldots ?$$



Dynamic programming

Need to calculate gradients wrt. parameters:

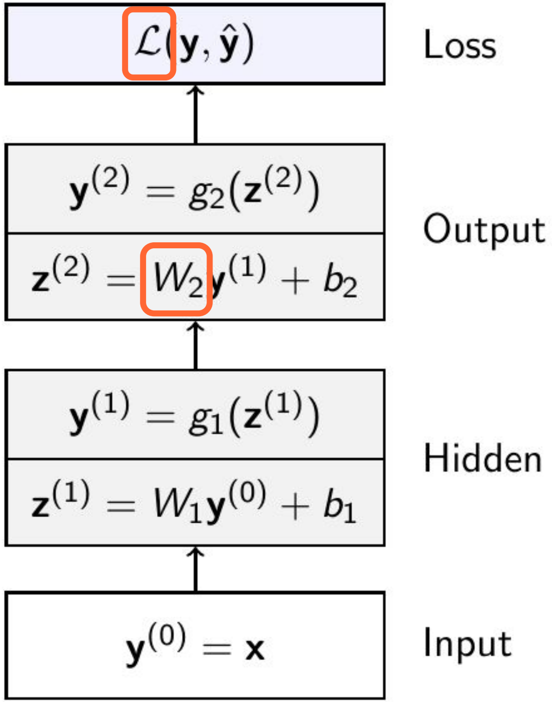
$$\frac{\partial \mathcal{L}}{\partial W_2} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}} \right) \boxed{\frac{\partial \mathbf{z}^{(2)}}{\partial W_2}}$$



Dynamic programming

Need to calculate gradients wrt. parameters:

$$\boxed{\frac{\partial \mathcal{L}}{\partial W_2}} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}} \right) \frac{\partial \mathbf{z}^{(2)}}{\partial W_2}$$

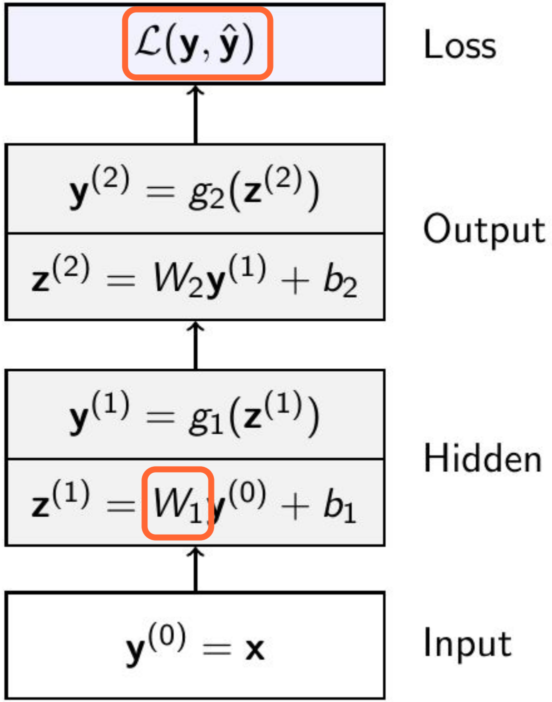


Dynamic programming

Need to calculate gradients wrt. parameters:

$$\frac{\partial \mathcal{L}}{\partial W_2} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}} \right) \frac{\partial \mathbf{z}^{(2)}}{\partial W_2}$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial W_1}} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}} \right) \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{y}^{(1)}} \frac{\partial \mathbf{y}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial W_1}$$

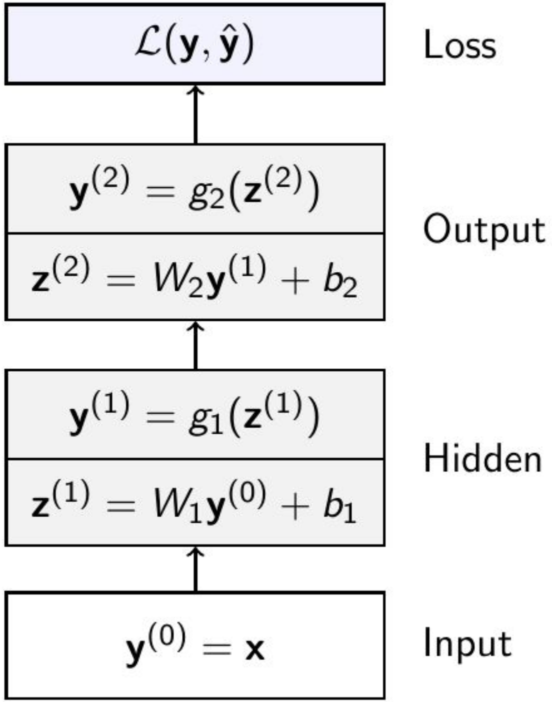


Dynamic programming

Need to calculate gradients wrt. parameters:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W_2} &= \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}} \right) \frac{\partial \mathbf{z}^{(2)}}{\partial W_2} \\
\frac{\partial \mathcal{L}}{\partial W_1} &= \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}} \right) \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{y}^{(1)}} \frac{\partial \mathbf{y}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial W_1}
\end{aligned}$$

- Can re-use the bit in parenthesis when computing gradient wrt. W_1.



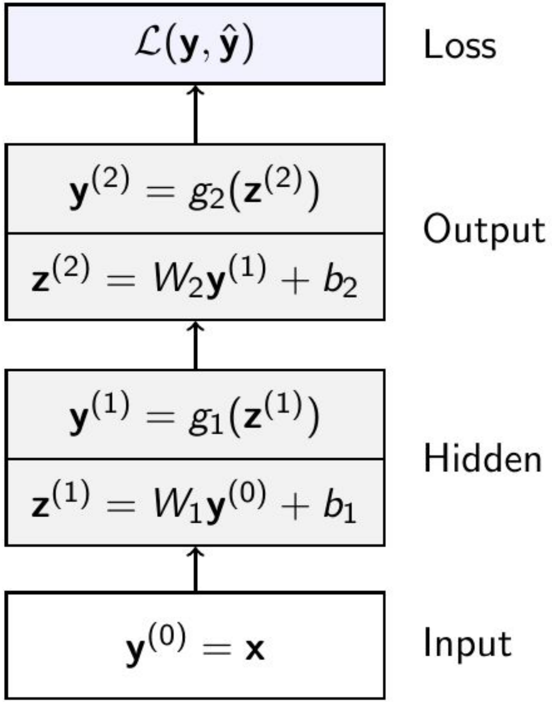
Dynamic programming

Need to calculate gradients wrt. parameters:

$$\frac{\partial \mathcal{L}}{\partial W_2} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}}\right) \frac{\partial \mathbf{z}^{(2)}}{\partial W_2}$$

$$\frac{\partial \mathcal{L}}{\partial W_1} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{y}^{(2)}} \frac{\partial \mathbf{y}^{(2)}}{\partial \mathbf{z}^{(2)}}\right) \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{y}^{(1)}} \frac{\partial \mathbf{y}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial W_1}$$

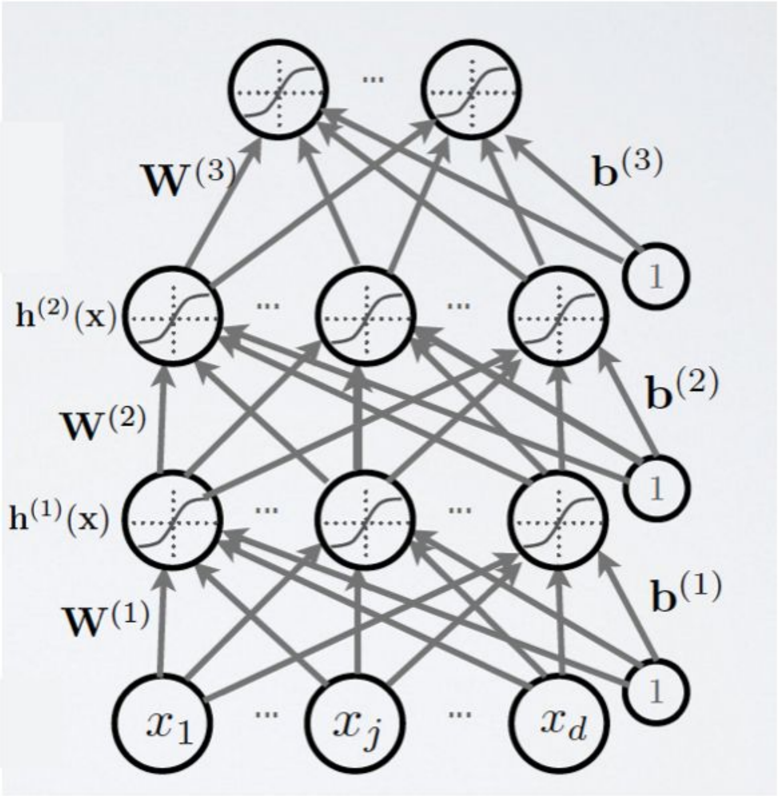
- Can re-use the bit in parenthesis when computing gradient wrt. W_1.



Overview

- Limitations the perceptron
- Multilayer Perceptrons
- Dynamic Programming
- **Implementation details**

Software implementation



Slide Credit: Hugo Laroche NN course

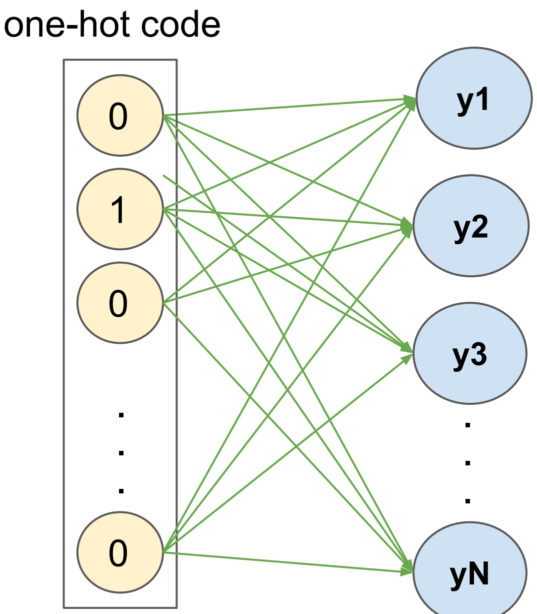
PYTORCH

```
NUM_INPUTS=100
HIDDEN_SIZE=1024
NUM_OUTPUTS=20

mlp = nn.Sequential(
    nn.Linear(NUM_INPUTS, HIDDEN_SIZE),
    nn.Tanh(),
    nn.Linear(HIDDEN_SIZE, HIDDEN_SIZE),
    nn.Tanh(),
    nn.Linear(HIDDEN_SIZE, NUM_OUTPUTS),
    nn.LogSoftmax(dim=1)
)
```

Software implementation: Discrete inputs

We usually take one-hot codes as discrete tokens. Can we use a *Linear layer* to process it?



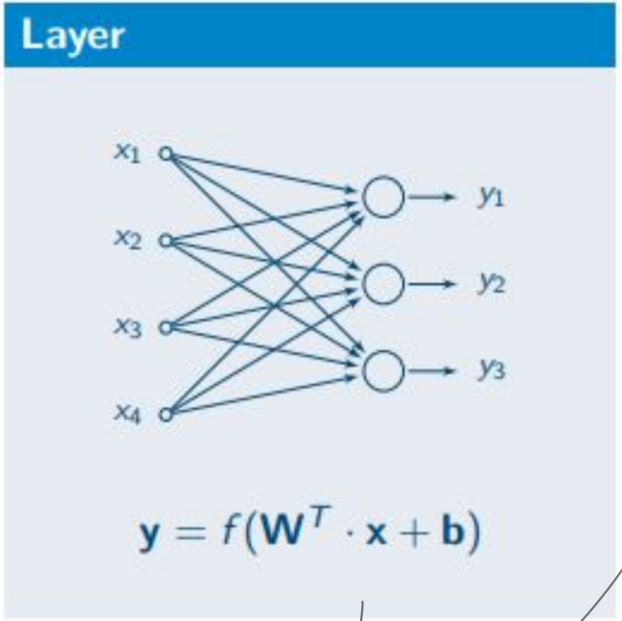

```
VOCAB_SIZE = 10000
HIDDEN_SIZE=100
# mapping a Vocabulary of size 10.000 to HIDDEN_SIZE projections
emb_1 = nn.Linear(VOCAB_SIZE, HIDDEN_SIZE)
```

```
# forward example [10, 10000] tensor
code = [1] + [0] * 9999
# copy 10 times the same code [1 0 0 0 ... 0]
x = torch.FloatTensor([code] * 10)
print('Input x tensor size: ', x.size())
y = emb_1(x)
print('Output y embedding size: ', y.size())
```

```
Input x tensor size:  torch.Size([10, 10000])
Output y embedding size:  torch.Size([10, 100])
```

Software implementation: Discrete inputs

Each x as an Integer 0 or 1



CLASS `torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None, max_norm=None, norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None)` [SOURCE]

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters:

- **num_embeddings** (*int*) – size of the dictionary of embeddings
- **embedding_dim** (*int*) – the size of each embedding vector
- **padding_idx** (*int, optional*) – If given, pads the output with the embedding vector at `padding_idx` (initialized to zeros) whenever it encounters the index.
- **max_norm** (*float, optional*) – If given, each embedding vector with norm larger than `max_norm` is renormalized to have norm `max_norm`.
- **norm_type** (*float, optional*) – The p of the p-norm to compute for the `max_norm` option. Default `2.`
- **scale_grad_by_freq** (*boolean, optional*) – If given, this will scale gradients by the inverse of frequency of the words in the mini-batch. Default `False`.
- **sparse** (*bool, optional*) – If `True`, gradient w.r.t. `weight` matrix will be a sparse tensor. See Notes for more details regarding sparse gradients.

Variables:

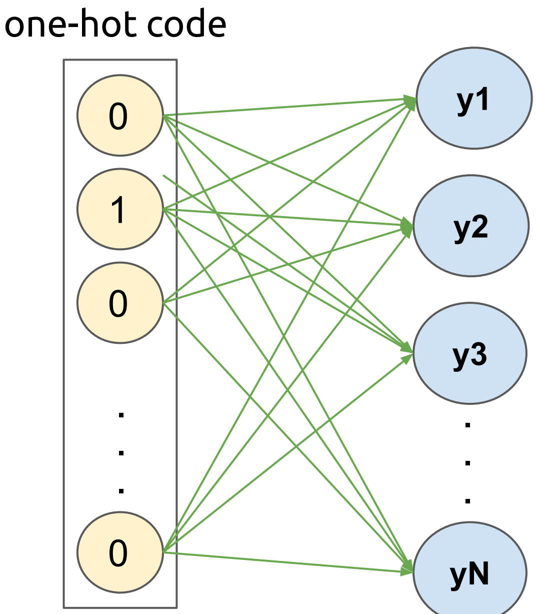
weight (*Tensor*) – the learnable weights of the module of shape (num_embeddings, embedding_dim) initialized from $\mathcal{N}(0, 1)$

Shape:

- Input: LongTensor of arbitrary shape containing the indices to extract
- Output: (*, *embedding_dim*), where * is the input shape

Software implementation: Discrete inputs

PYTORCH



Embedding layer makes an efficient lookup operation, not a full matrix multiplication (**just select one-hot index column from weight matrix!**)

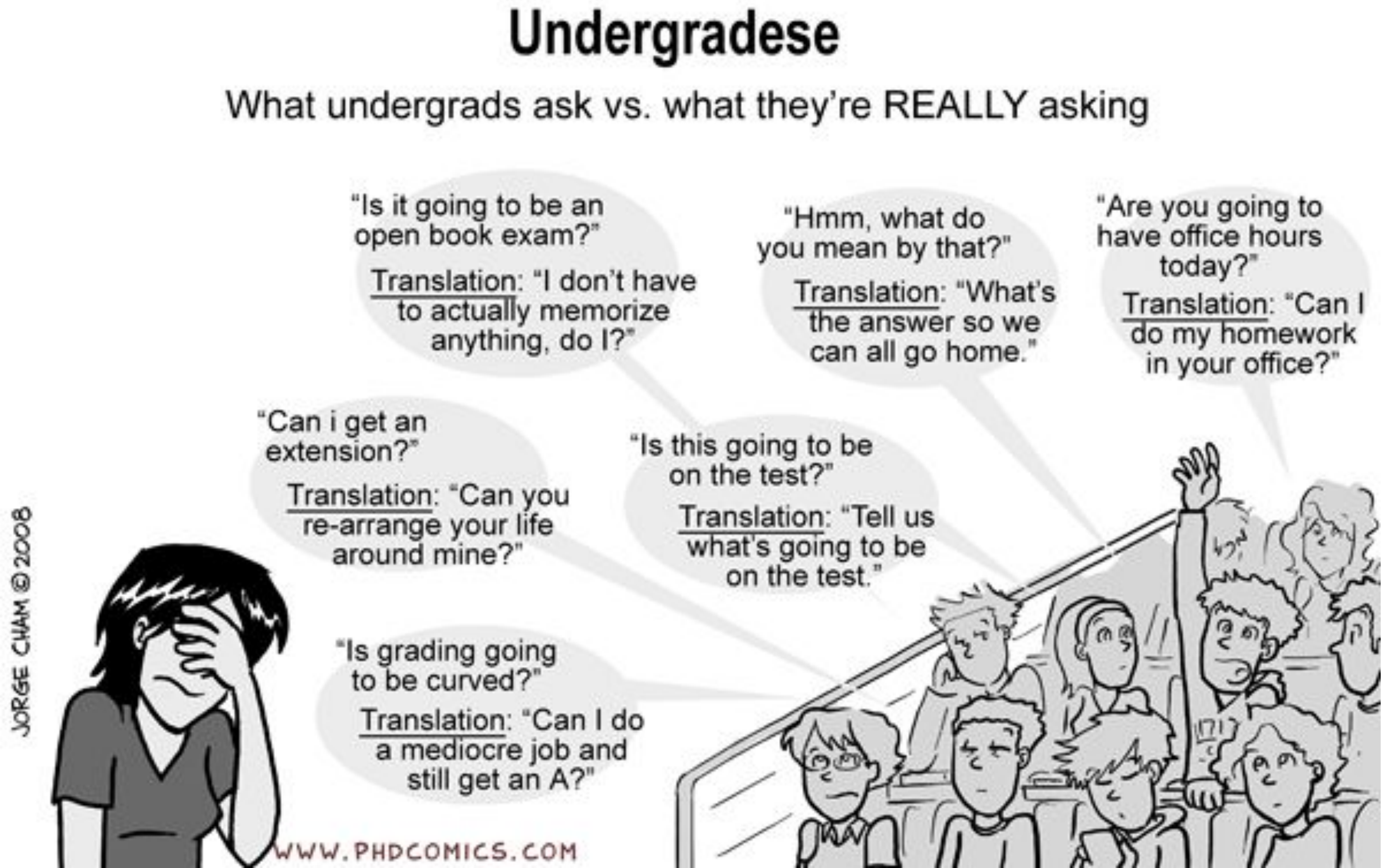
```
VOCAB_SIZE = 10000
HIDDEN_SIZE=100
# mapping a Vocabulary of size 10.000 to HIDDEN_SIZE projections
emb_2 = nn.Embedding(VOCAB_SIZE, HIDDEN_SIZE)
```

```
# Just make a long tensor with zero-index
x = torch.zeros(10, 1).long()
print('Input x tensor size: ', x.size())
y = emb_2(x)
print('Output y embedding size: ', y.size())
```

```
Input x tensor size:  torch.Size([10, 1])
Output y embedding size:  torch.Size([10, 1, 100])
```

Overview

- Limitations the perceptron
- Multilayer Perceptrons
- Dynamic Programming
- Implementation details


Undergradese
What undergrads ask vs. what they're REALLY asking
"Is it going to be an open book exam?"
Translation: "I don't have to actually memorize anything, do I?"
"Hmm, what do you mean by that?"
Translation: "What's the answer so we can all go home."
"Are you going to have office hours today?"
Translation: "Can I do my homework in your office?"
"Can i get an extension?"
Translation: "Can you re-arrange your life around mine?"
"Is this going to be on the test?"
Translation: "Tell us what's going to be on the test."
"Is grading going to be curved?"
Translation: "Can I do a mediocre job and still get an A?"
JORGE CHAM © 2008
WWW.PHDCOMICS.COM