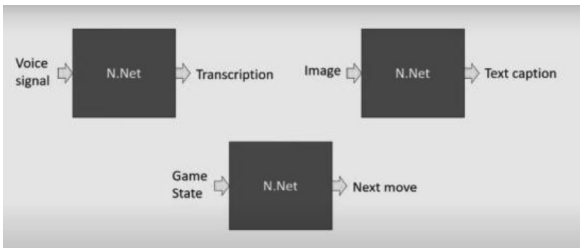
ANN
MLP – Depth Vs Width

Prof. (Dr.) Keyur Rana

Neural Networks - Function

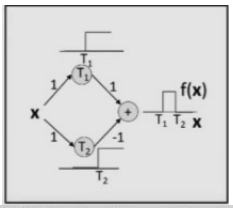


- Boxes take inputs and give Outputs

Deep Networks

- Depth > 2 (i.e. 3 or more layers)

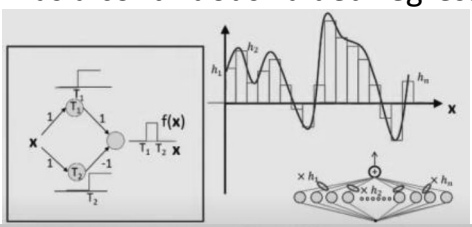
MLP as a continuous valued regression



Combination of T1 and T2 node can generate a **Square Pulse**

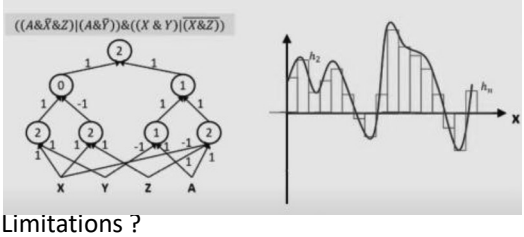
A simple 3-unit MLP with a "summing" output unit can generate a "square pulse" over an input
- Output is 1 only if the input lies between T_1 and T_2
- T_1 and T_2 can be arbitrarily specified

MLP as a continuous valued regression



- **An MLP with many units can model an arbitrary function over an input**
  - To arbitrary precision
    - Simply make the individual pulses narrower
- **This generalizes to functions of any number of inputs**

Multi Layer Perceptron (MLP)



- Limitations ?
- MLP approximate functions

MLP

- How many layers are required for any arbitrary Boolean function?
  - One
  - Why ?
  - Any Boolean function is a just a Truth table
  - Only needed to specify when the output is 1

MLP

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

A compact way to represent a Truth table is - A Disjunctive Normal Form (DNF) expression

Kaurnaugh Map (K-Map)

MLP

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

One perceptron per Clause

6 perceptrons are required

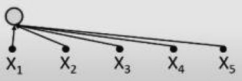
MLP

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$



MLP

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

MLP

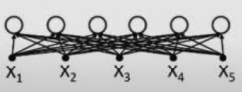
Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

Now Apply OR node to get Final output



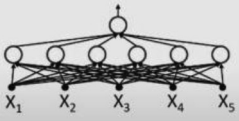
MLP

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

Truth table shows *all* input combinations for which output is 1

$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$



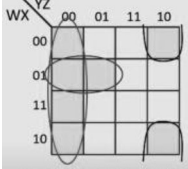
- Any Boolean function can be expressed in this manner
- A one layer MLP is a Universal Boolean Function

MLP

- Any Boolean function can be expressed in this manner
- A one layer MLP is a Universal Boolean Function
- What is the largest number of perceptron required in the single hidden layer for an N-input-variable function?
  - Exponentially large

MLP

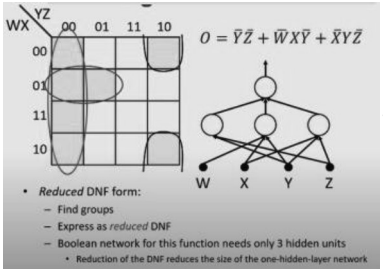
Reducing a Boolean Function



$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$

- *Reduced* DNF form:
  - Find groups
  - Express as reduced DNF

MLP

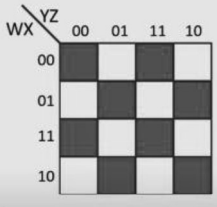


$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$

- *Reduced* DNF form:
  - Find groups
  - Express as *reduced* DNF
  - Boolean network for this function needs only 3 hidden units
    - Reduction of the DNF reduces the size of the one-hidden-layer network

Worst case ?

Largest irreducible DNF?



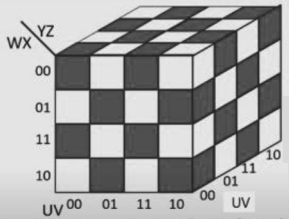
Red=0, white=1

How many Neurons for One hidden layer MLP required ?

8

- What arrangement of ones and zeros simply cannot be reduced further?

Width of a single-layer Boolean MLP



Red=0, white=1

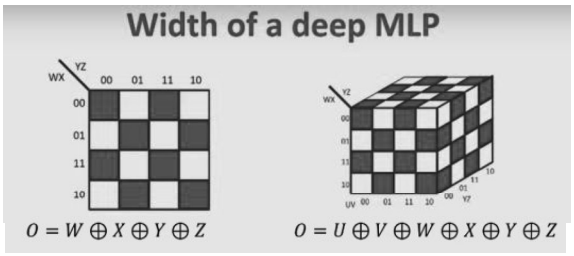
How many Neurons for One hidden layer MLP required ?

32

- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function of 6 variables?

Width of a single layer

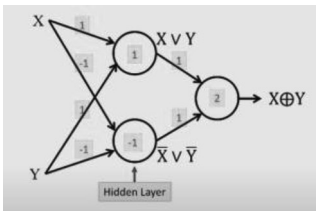
- Can be generalized : It will require 2^{N-1} perceptrons in hidden layer
- Exponential in N
- How many Units if we use Multiple Layers?



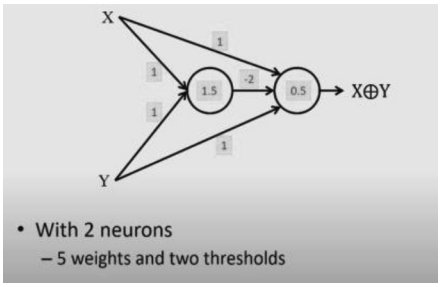
Which function is this?

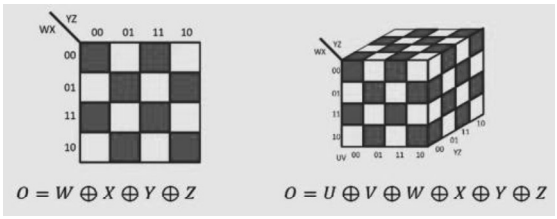
MLP – Ex-OR

- How many neurons do we require to compose an Ex-OR?
  - **– Three**
  - **– It can be done by Two also.**



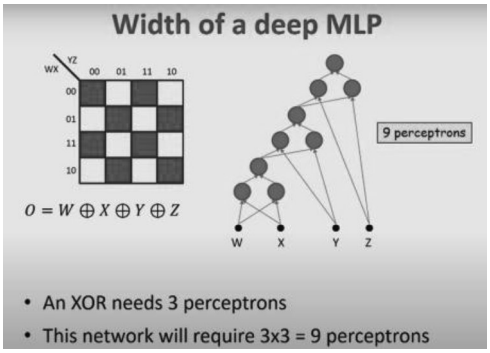
MLP – Ex-OR



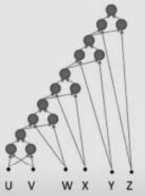


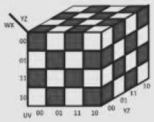
9 15

- How many neurons require for these functions?



Width of a deep MLP





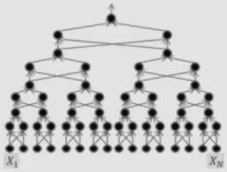
$O = U \oplus V \oplus W \oplus X \oplus Y \oplus Z$

15 perceptrons

- An XOR needs 3 perceptrons
- This network will require 3x5 = 15 perceptrons

- More generally, the Ex-OR of N variables will require 3(N-1) perceptrons (**Deep network**)
  - Linear in nature
- A **Single hidden layer** require $2^{N-1}+1$ perceptrons in all (including O/p unit)
  - Exponential in N

A better representation

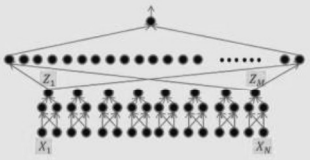


$O = X_1 \oplus X_2 \oplus \cdots \oplus X_N$

- Only $2\log_2 N$ layers
  - By pairing terms
  - 2 layers per XOR

$O = (((((X_1 \oplus X_2) \oplus (X_3 \oplus X_4)) \oplus ((X_5 \oplus X_6) \oplus (X_7 \oplus X_8))) \oplus (((...$

The challenge of depth

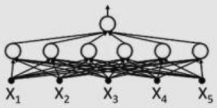


$O = X_1 \oplus X_2 \oplus \cdots \oplus X_N$

$= Z_1 \oplus Z_2 \oplus \cdots \oplus Z_M$

- **I.e. reducing the number of layers below the minimum will result in an exponentially sized network to express the function fully**
- **A network with fewer than the minimum required number of neurons *cannot* model the function**

The actual number of parameters in a network



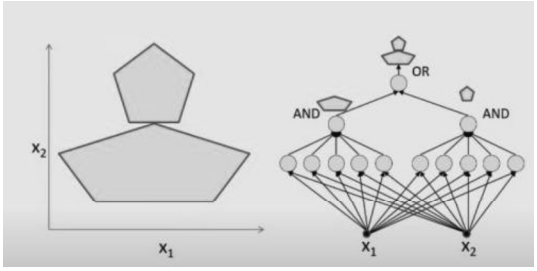
- The actual number of parameters in a network is the number of *connections*
  - In this example there are 30
- This is the number that really matters in software or hardware implementations
- Networks that require an exponential number of neurons will require an exponential or *super*exponential number of weights..

The need for depth

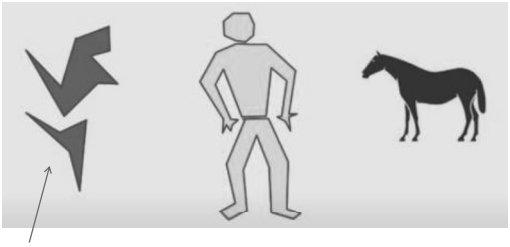
- *Deep* Boolean MLPs that scale *linearly* with the number of inputs …
- … can become exponentially large if recast using only one layer
- It gets worse..
- Having a few extra layers can greatly reduce network size

- Multi-layer perceptrons are *Universal Boolean Machines*
- Even a network with a *single* hidden layer is a universal Boolean machine
  - But a single-layer network may require an exponentially large number of perceptrons
- Deeper networks may require far fewer neurons than shallower networks to express the same function
  - Could be *exponentially* smaller

More Complicated decision boundaries



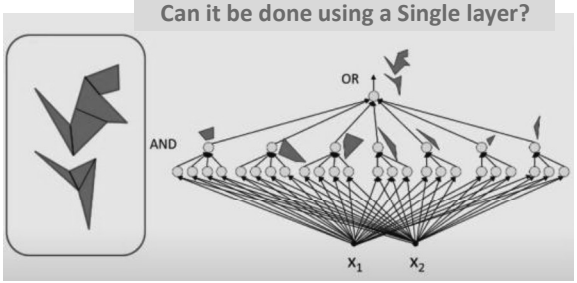
More Complicated decision boundaries



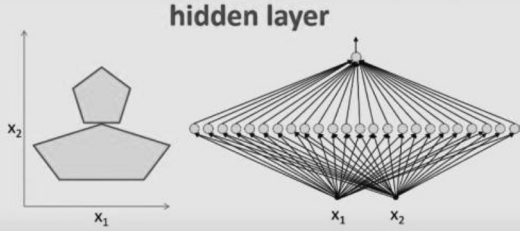
Design a network with two variable input

More Complicated decision boundaries

Can it be done using a Single layer?



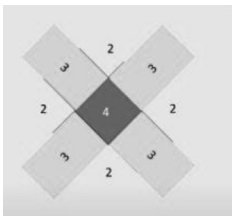
Exercise: compose this with one hidden layer

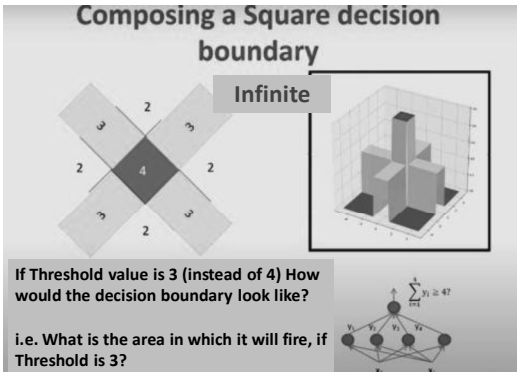


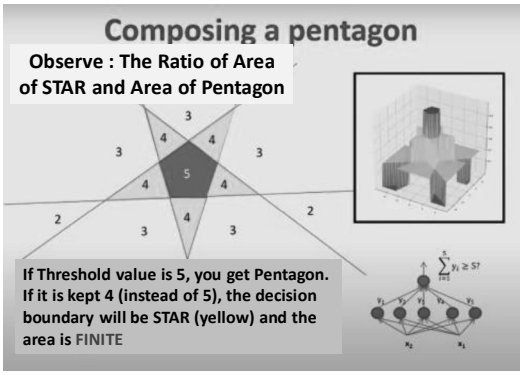
- How would you compose the decision boundary to the left with only *one* hidden layer?

Lets simplify the problem…

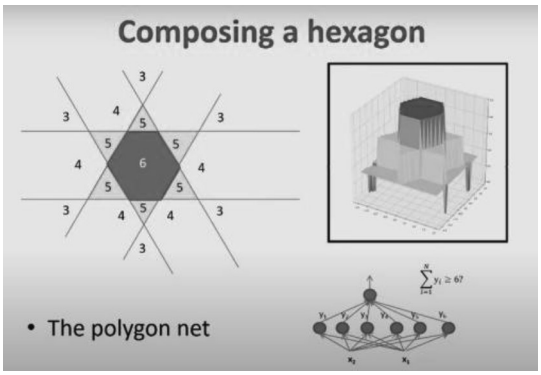
- Lets compute a diamond as a decision boundary.
  - How many neuron do we need in the (one) hidden layer?
  - Four



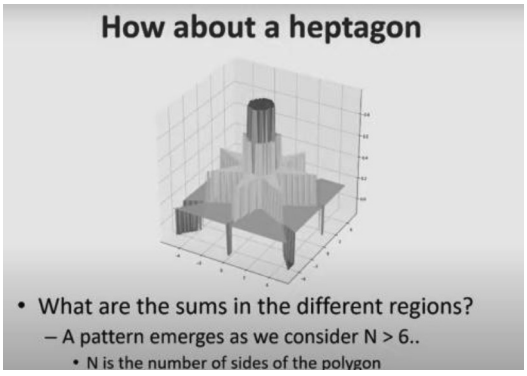




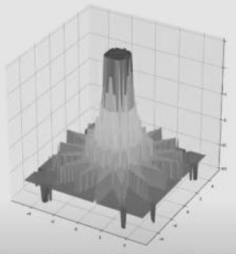
- Observe : The Ratio of Area of STAR and Area of Pentagon
  - Much better than the Area of area having Th=3 to the Area of Dimond (previous case)



- Observe : The Ratio of Area of STAR and Area of Pentagon
  - Much better than the Area of area having Th=3 to the Area of Dimond (previous case)
- The Ratio in case of Hexa gone is better

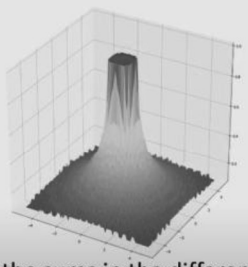


16 sides



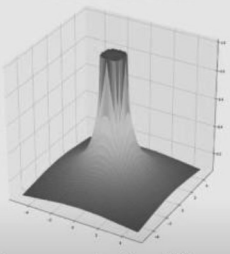
- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

64 sides



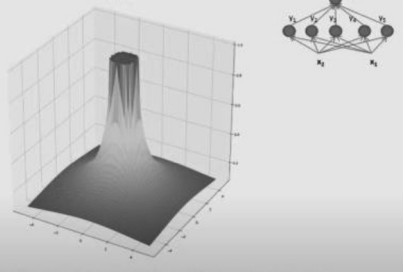
- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

1000 sides



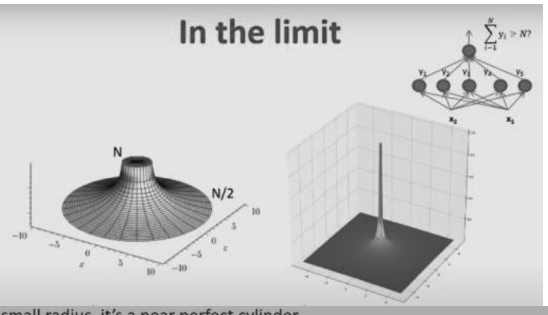
- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

Polygon net



- Increasing the number of sides reduces the area outside the polygon that have $\frac{N}{2} < \sum_i y_i < N$

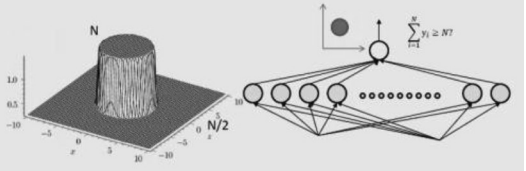
In the limit



For small radius, it's a near perfect cylinder

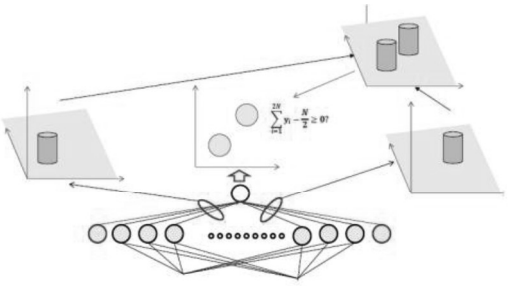
- N in the cylinder, N/2 outside

Composing a circle

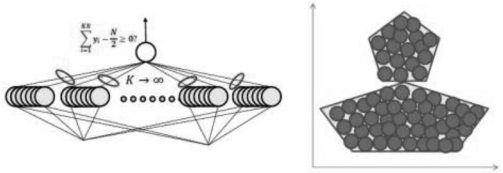


- The circle net
  - Very large number of neurons
  - *Sum is N inside the circle, N/2 outside almost everywhere*
  - Circle can be at any location

Adding circles

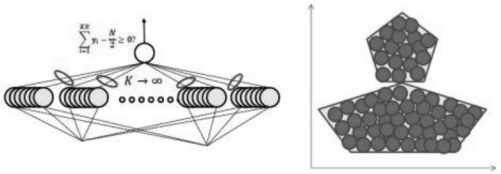


Composing an arbitrary figure



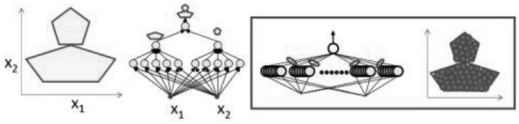
- Just fit in an arbitrary number of circles
  - More accurate approximation with greater number of smaller circles
  - Can achieve arbitrary precision

MLP: Universal classifier



- MLPs can capture any classification boundary
- A one-hidden-layer MLP can model any classification boundary
- **MLPs are universal classifiers**

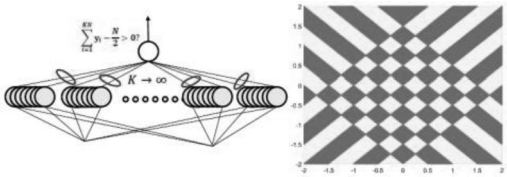
Depth and the universal classifier



- Deeper networks can require far fewer neurons
  - 12 vs. ~infinite hidden neurons in this example

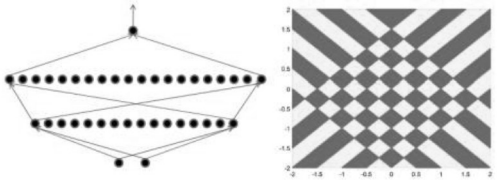
Some more complex patterns…

Optimal depth



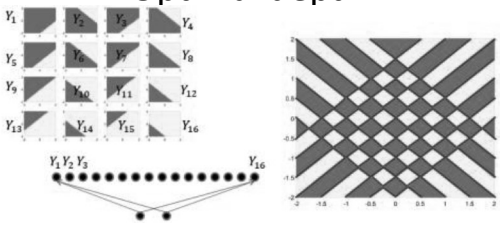
- A naïve one-hidden-layer neural network will require **infinite** hidden neurons

Optimal depth



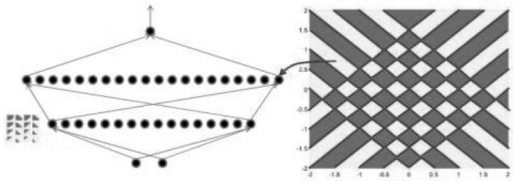
- Two hidden-layer network will require…??
  - 56 hidden neurons and 1 output neuron

Optimal depth



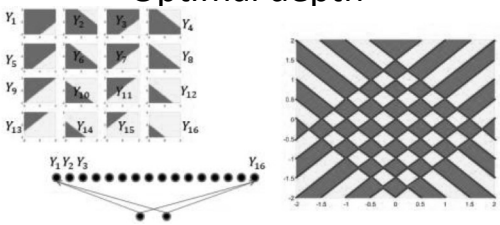
- Two-hidden-layer network: 56 hidden neurons
  - 16 neurons in hidden layer 1

Optimal depth



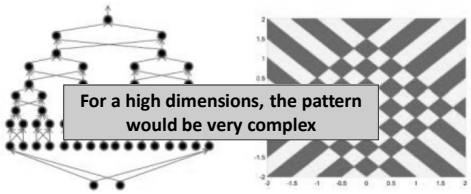
- Two-hidden-layer network: 56 hidden neurons
  - 16 in hidden layer 1
  - in hidden layer 2 **Each neuron in layer 2 is for one Yellow polygon**
  - total neurons, including output neuron

Optimal depth



- But this is just $Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{16}$

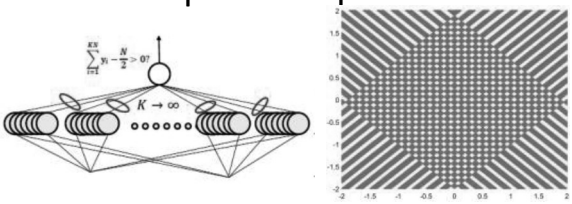
Optimal depth



- But this is just $Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{16}$

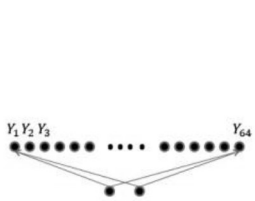
The XOR net will require 16 + 15x3 = 61 neurons
- 46 (16 + 15x2) neurons if we use a two-neuron XOR model

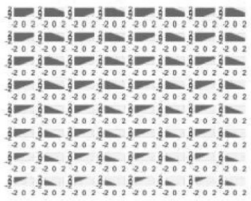
Optimal depth



- A naïve one-hidden-layer neural network will require **infinite** hidden neurons
- Grid formed from **64** lines
  - Network must output 1 for inputs in the yellow regions

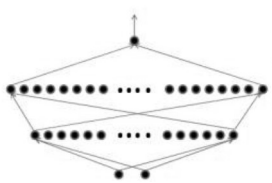
Actual linear units

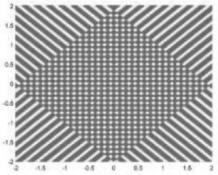




- 64 basic linear feature detectors

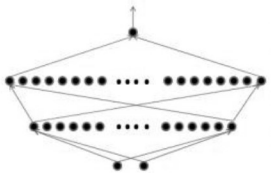
Optimal depth

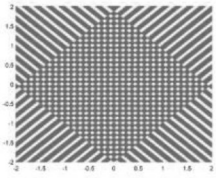




- Two hidden layers: 608 hidden neurons
  - 64 in layer 1
  - 544 in layer 2
- 609 total neurons (including output neuron)

Optimal depth





- XOR network (12 hidden layers): 253 neurons (64 + 63x3)
  - 190 neurons with 2-gate XOR (64 + 64x2)
- The difference in size between the deeper optimal (XOR) net and shallower nets increases with increasing pattern complexity and input dimension

Optimal depth

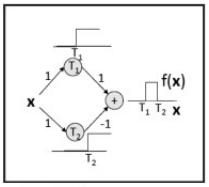
Summary :

- The number of neurons required in a shallow network is potentially exponential in the dimensionality of the input
- The number of neurons grows exponential not with number of individual features but with the number of **statistically independent individual** features

Summary - so far…

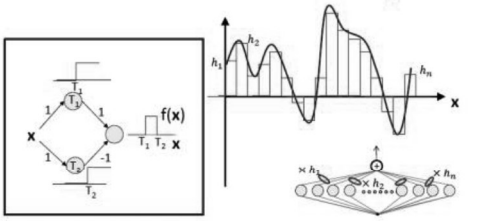
- Multi-layer perceptrons are Universal Boolean Machines
  - Even a network with a single hidden layer is a universal Boolean machine
- Multi-layer perceptrons are Universal Classification Functions
  - Even a network with a single hidden layer is a universal classifier
- But a single-layer network may require an exponentially large number of perceptrons than a deep one
- Deeper networks may require far fewer neurons than shallower networks to express the same function
  - Could be exponentially smaller
  - Deeper networks are more expressive

MLP as a continuous-valued regression



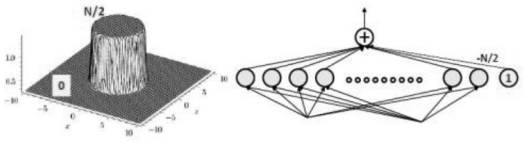
- A simple 3-unit MLP with a "summing" output unit can generate a "square pulse" over an input
  - Output is 1 only if the input lies between T_1 and T_2
  - T_1 and T_2 can be arbitrarily specified

MLP as a continuous-valued regression



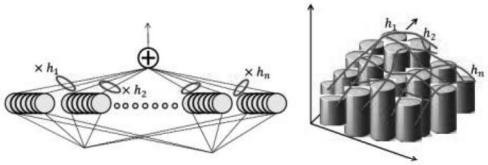
- A simple 3-unit MLP can generate a "square pulse" over an input
- An MLP with many units can model an arbitrary function over an input
  - To arbitrary precision
    - Simply make the individual pulses narrower
- *A one-hidden-layer MLP can model an arbitrary function of a single input*

For higher dimensions



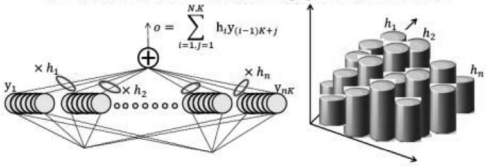
- An MLP can compose a cylinder
  - $N/2$ in the circle, 0 outside

MLP as a continuous-valued function



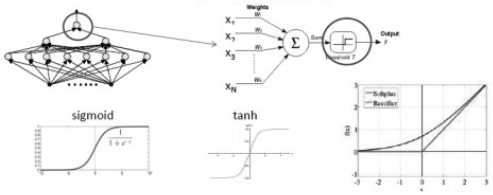
- MLPs can actually compose arbitrary functions in any number of dimensions!
  - Even with only one hidden layer
    - As sums of scaled and shifted cylinders
  - To arbitrary precision
    - By making the cylinders thinner
  - The MLP is a universal approximator!

Caution: MLPs with additive output units are universal approximators



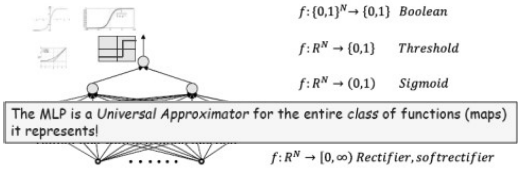
- MLPs can actually compose arbitrary functions
- But explanation so far only holds if the output unit only performs summation
  - i.e. does not have an additional "activation"

"Proper" networks: Outputs with activations



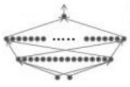
- Output neuron may have actual "activation"
  - Threshold, sigmoid, tanh, softplus, rectifier, etc.
- What is the property of such networks?

The network as a function

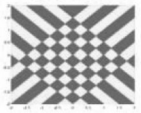


- Output unit with *activation function*
  - Threshold or Sigmoid, or any other
- The network is actually a universal map from the entire domain of input values to the entire range of the output activation
  - All values the activation function of the output neuron

Sufficiency of architecture

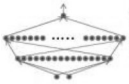


A network with 16 or more neurons in the first layer *is capable of* representing the figure to the right perfectly

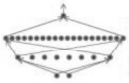


- A neural network *can* represent any function provided it has sufficient *capacity*
  - I.e. sufficiently broad and deep to represent the function
- Not all architectures can represent any function

Sufficiency of architecture



A network with 16 or more neurons in the first layer *is capable of* representing the figure to the right perfectly

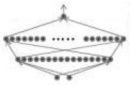


A network with less than 16 threshold-activation neurons in the first layer cannot represent this pattern exactly

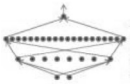
Why?

- A neural network *can* represent any function provided it has sufficient *capacity*
  - I.e. sufficiently broad and deep to represent the function
- Not all architectures can represent any function

Sufficiency of architecture



A threshold-gate network with 16 or more neurons in the first layer *is capable of* representing the figure to the right perfectly

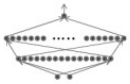


A network with less than 16 threshold-activation neurons in the first layer cannot represent this pattern exactly

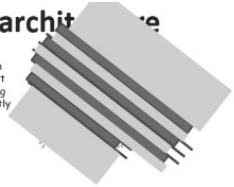
Why?

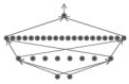
- A network with only 8 threshold neurons in the first layer may capture these 8 boundaries

Sufficiency of archi e



A threshold-gate network with 16 or more neurons in the first layer *is capable of* representing the figure to the right perfectly

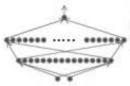




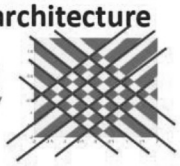
A network with less than 16 threshold-activation neurons in the first layer cannot represent this pattern exactly

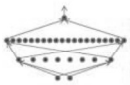
- A network with only 8 threshold neurons in the first layer may capture these 8 boundaries
- That can only give you information about which of these strips the input is in, but not *where* in the strip

Sufficiency of architecture



A threshold-gate network with 16 or more neurons in the first layer *is capable of* representing the figure to the right perfectly

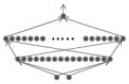




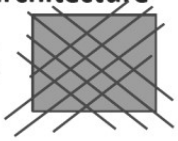
A network with less than 16 threshold-activation neurons in the first layer cannot represent this pattern exactly

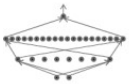
- Even if the 8 first-layer neurons capture *these* boundaries…

Sufficiency of architecture



A threshold-gate network with 16 or more neurons in the first layer *is capable of* representing the figure to the right perfectly

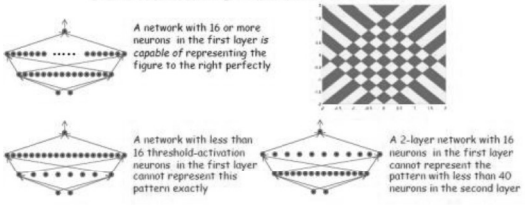




A network with less than 16 threshold-activation neurons in the first layer cannot represent this pattern exactly

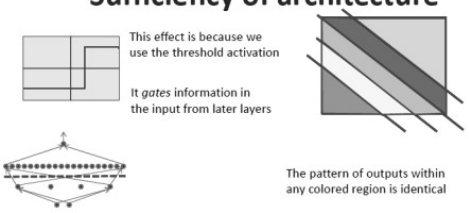
- Even if the 8 first-layer neurons capture *these* boundaries…
- … they can only place you in one of these 25 cells, but cannot inform you of *where* in the cell

Sufficiency of architecture



- Similar restrictions apply to higher layers
- Regardless of depth, every layer must be sufficiently wide in order to capture the function
- Not all architectures can represent any function

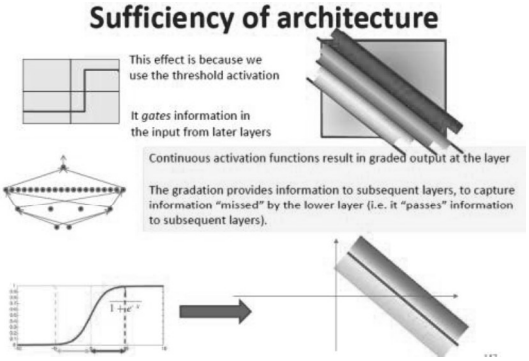
Sufficiency of architecture



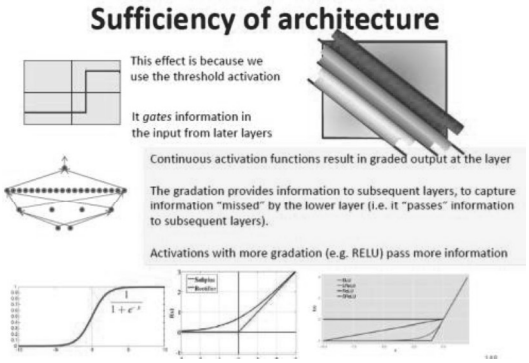
Threshold activation function looses information.
Once you cross the threshold, it does not tell you how far you are from the threshold

- Different kind of activation
- What type of activation function is good?

Sufficiency of architecture



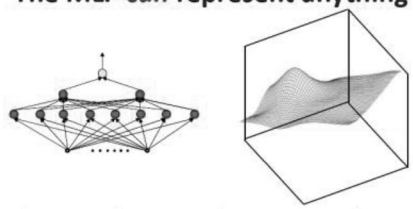
Sufficiency of architecture



Width vs. Activations vs. Depth

- Narrow layers can still pass information to subsequent layers if the activation function is sufficiently graded
- But will require greater depth, to permit later layers to capture patterns

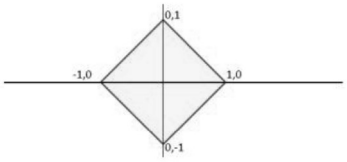
The MLP *can* represent anything



- The MLP *can be constructed* to represent anything
- But *how* do we construct it?

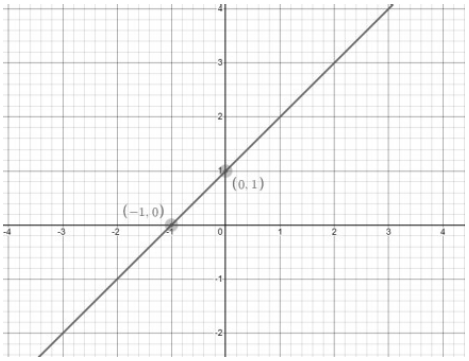
13

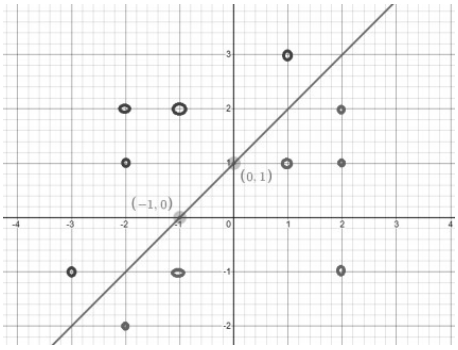
Option 1: Construct by hand

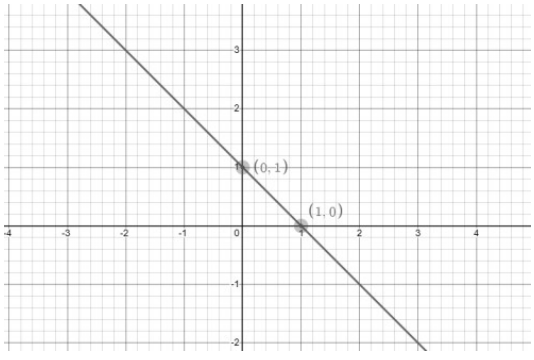


- Given a function, *handcraft* a network to satisfy it
- E.g.: Build an MLP to classify this decision boundary

14





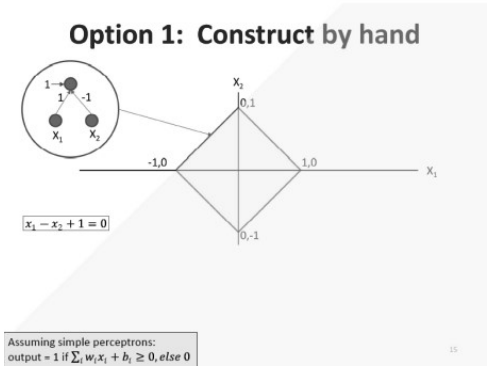


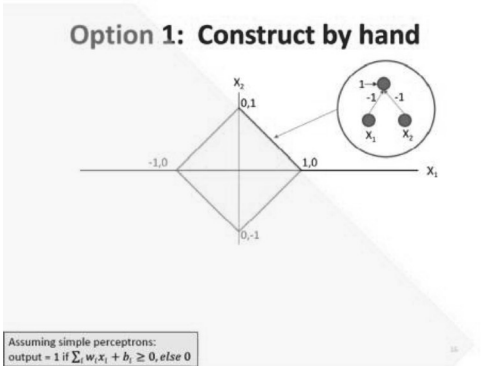
Four lines are…

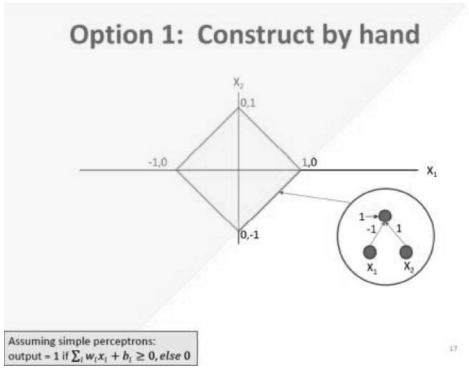
- X1=X2 -1
- X1=-X2+1
- X1=X2+1
- X1=-X2-1

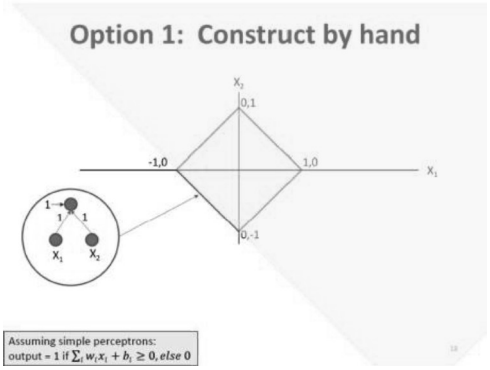
Four Solutions are…

- X1-X2+1=0
- X1+X2-1=-X1-X2+1=0
- X1-X2-1=-X1+X2+1=0
- X1+X2+1=0







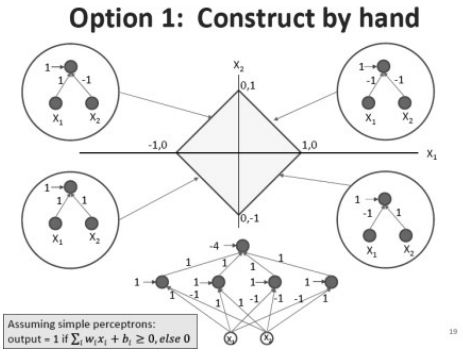


Four lines are…

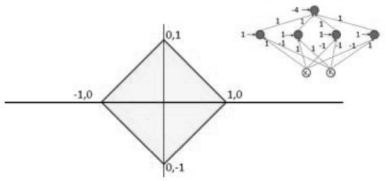
- X1=X2 -1
- X1=-X2+1
- X1=X2+1
- X1=-X2-1

Four Solutions are…

- X1-X2+1=0
- X1+X2-1=-X1-X2+1=0
- X1-X2-1=-X1+X2+1=0
- X1+X2+1=0



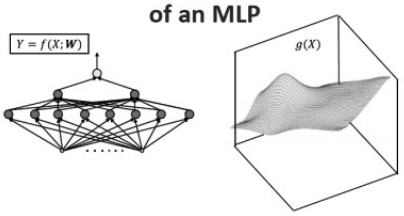
Option 1: Construct by hand



- Given a function, *handcraft* a network to satisfy it
- E.g.: Build an MLP to classify this decision boundary
- Not possible for all but the simplest problems..

20

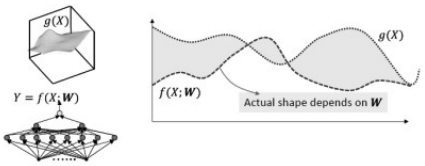
Option 2: Automatic estimation of an MLP



- More generally, *given* the function $g(X)$ to model, we can *derive* the parameters of the network to model it, through computation

21

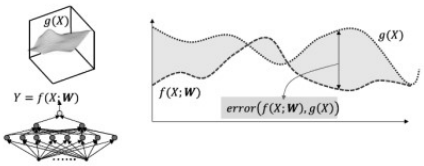
How to learn a network?



- Solution: Estimate parameters to minimize the error between the target function $g(X)$ and the network function $f(X, \boldsymbol{W})$
  - Find the parameter $\boldsymbol{W}$ that minimizes the shaded area

22

How to learn a network?



- The shaded area

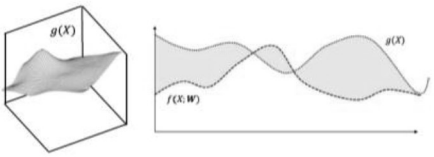
$$totalerr(\boldsymbol{W}) = \int_{-\infty}^{\infty} error(f(X; \boldsymbol{W}), g(X))dX$$

- The optimal $\boldsymbol{W}$

$$\widehat{\boldsymbol{W}} = \underset{\boldsymbol{W}}{\operatorname{argmin}}\ totalerr(\boldsymbol{W})$$

23

Problem: $g(X)$ is unknown



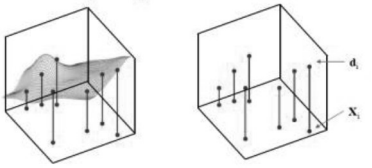
- Function $g(X)$ must be fully specified in order to compute

$$\int_{-\infty}^{\infty} error(f(X; \boldsymbol{W}), g(X))dX$$

  - Known *everywhere*, i.e. for *every* input X
- **In practice we will not have such specification**
- Instead, we would have few samples only (training data)

24

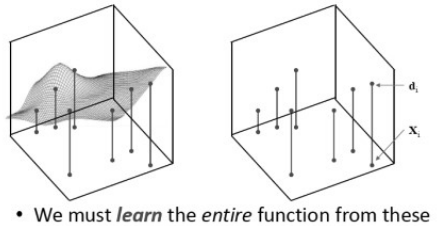
Sampling the function



- *Sample* $g(X)$
  - Basically, get input-output pairs for a number of samples of input X_i
    - Many samples (X_i, d_i), where $d_i = g(X_i) + noise$
- Very easy to do in most problems: just gather training data
  - E.g. set of images and their class labels
  - E.g. speech recordings and their transcription

25

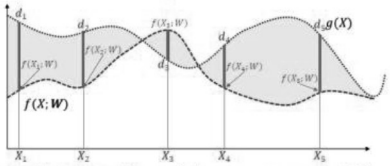
Drawing samples



- We must ***learn*** the *entire* function from these few examples
  - The "training" samples

26

The *Empirical* error



- The *empirical estimate* of the error is the *average* error over the training samples

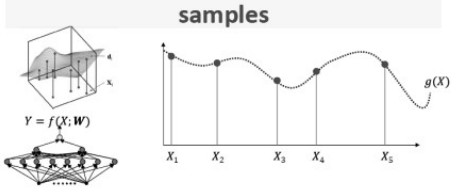
$$EmpiricalError(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} error(f(X_i;\boldsymbol{W}), d_i)$$

- Estimate network parameters to minimize this average error instead

$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}}\, EmpiricalError(\boldsymbol{W})$$

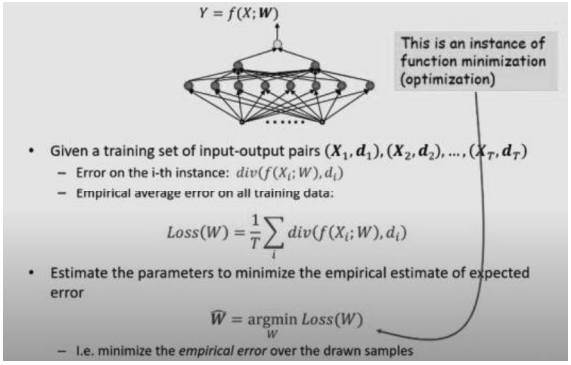
27

Learning the function from training samples



- Aim: Find the network parameters that "fit" the training points exactly $\boldsymbol{W}$: $EmpiricalError(\boldsymbol{W}) = 0$
  - Assuming network architecture is sufficient for such a fit
  - Assuming unique output d at any $\mathbf{X}$
- And hopefully the resulting function is also correct where we *don't* have training samples

28



Summary so far..

- "Learning" a neural network == determining the parameters of the network (weights and biases) required for it to model a desired function
  - The network must have sufficient capacity to model the function
- Ideally, we would like to optimize the network to represent the desired function everywhere
- However this requires knowledge of the function everywhere
- Instead, we draw "input-output" training instances from the function and estimate network parameters to "fit" the input-output relation at these instances
  - And hope it fits the function elsewhere as well

Empirical Risk Management

- Gradient descent algorithm