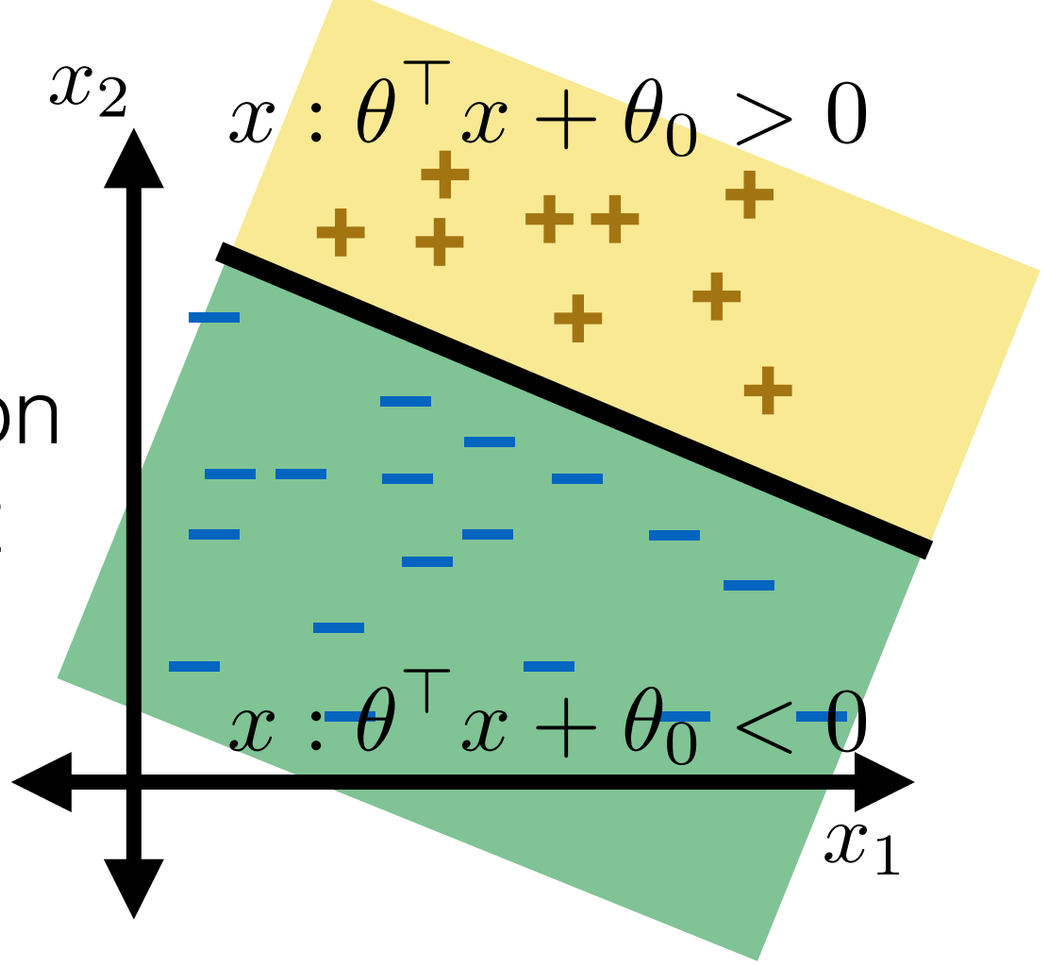
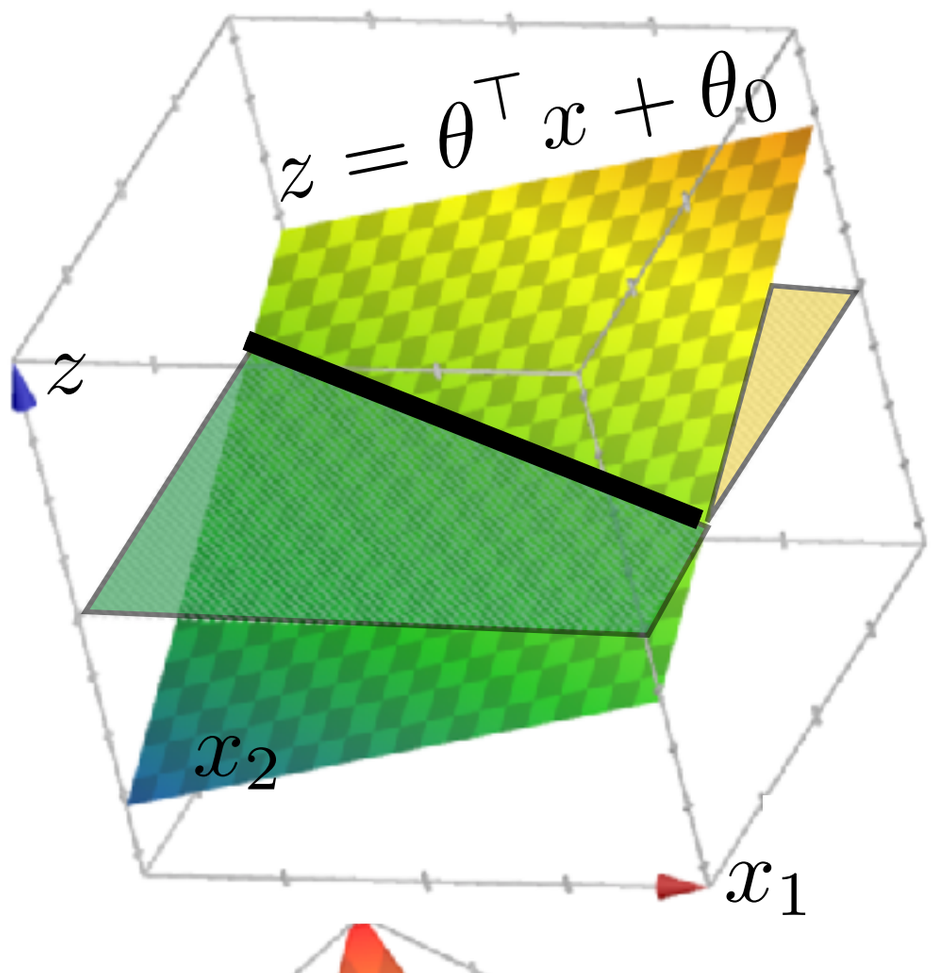
Recall

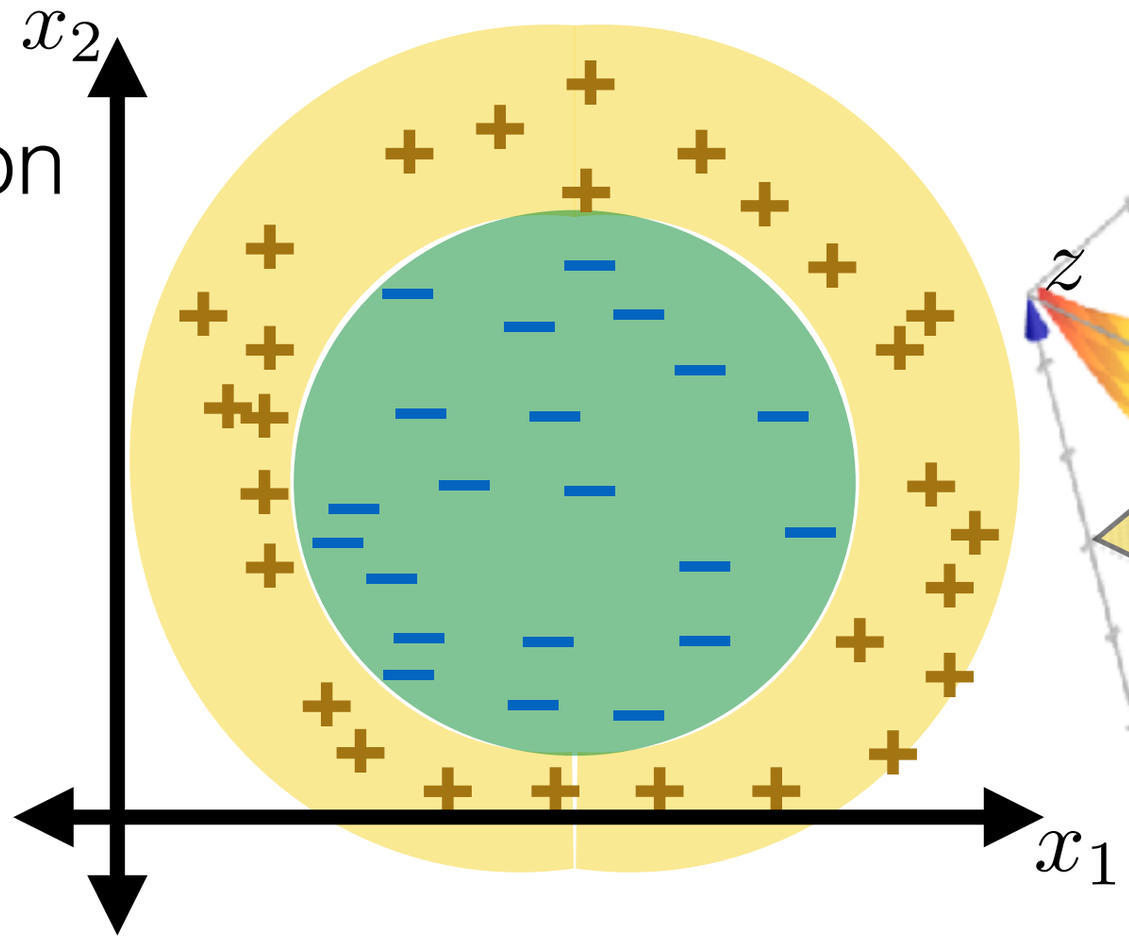
- Linear classification with default features:

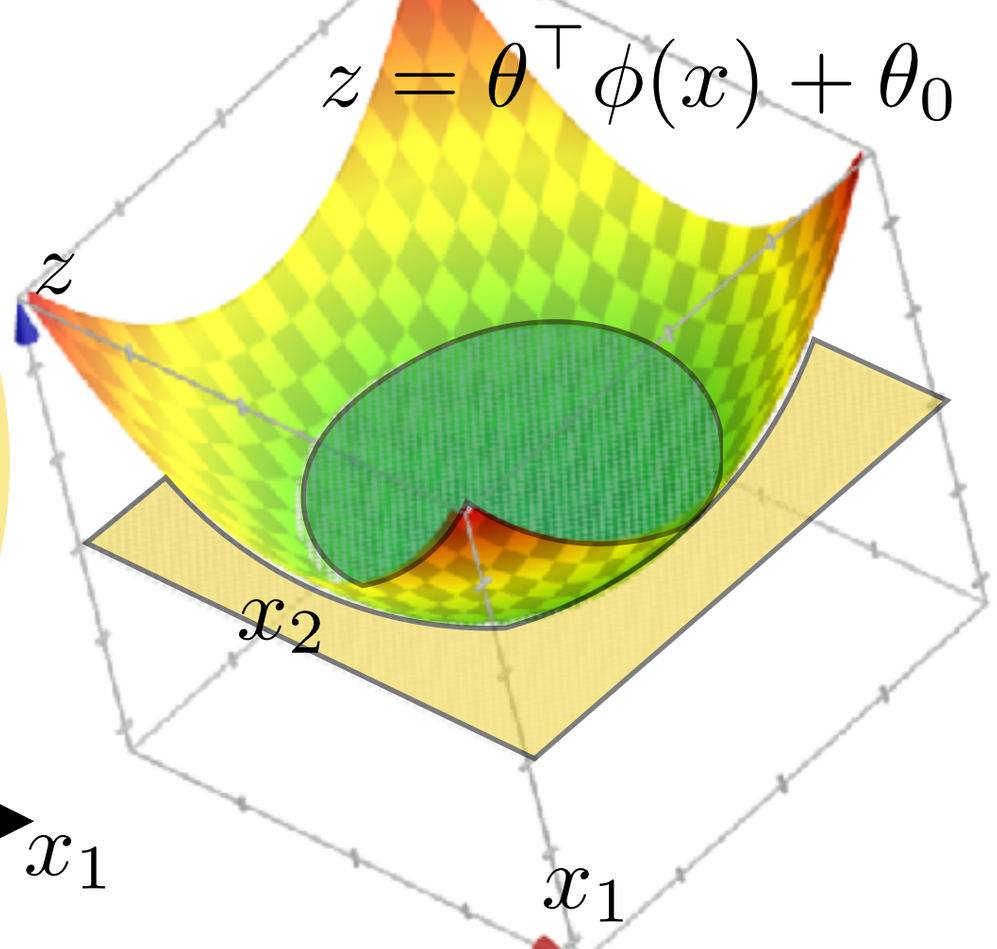




- Linear classification with polynomial features:

$$\phi(x) = [x_1, x_2, x_1^2, x_1 x_2, x_2^2]^\top$$





Lecture 6

★ Recall: (Slide - 1 - Last)

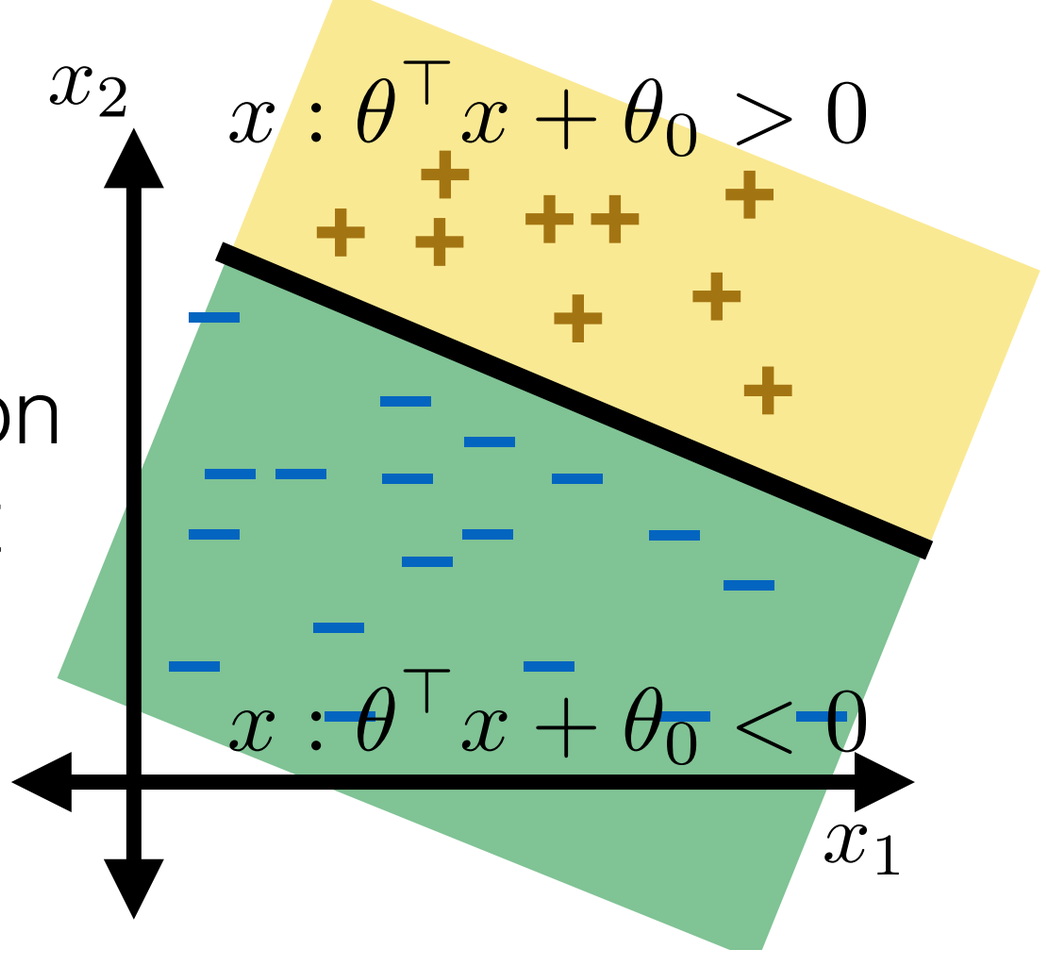
Linear classification with default features & Linear classification with polynomial features

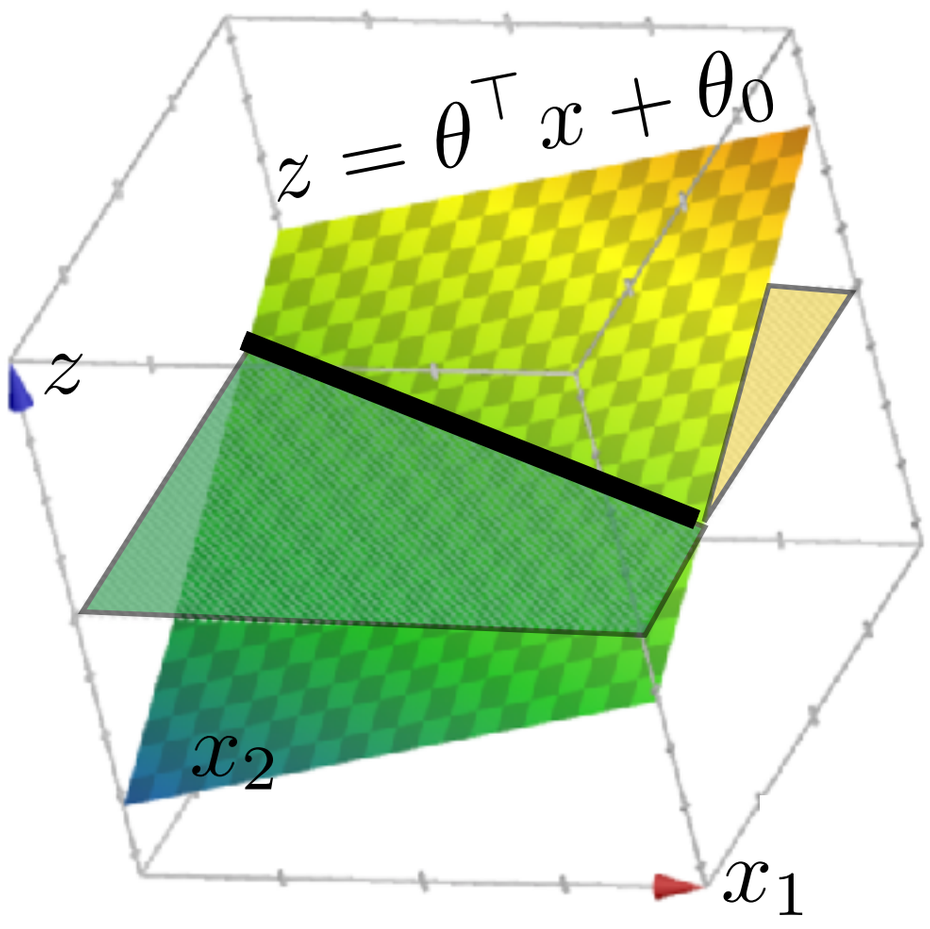
- Notice that for linear data (Figure 1 & 2) (Btw, I mean linear data & not linear classifier. The data is linear itself) $z = \theta^T x + \theta_0$, where $x = [x_1, x_2]$
- For linear classification with polynomial features (Fig 3 & 4) the data is non-linear & $z = \theta^T \phi(x) + \theta_0$ where $\phi(x) = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]$
- Notice that however the data might be, the classification is always linear (Hence, linear classification). i.e. the

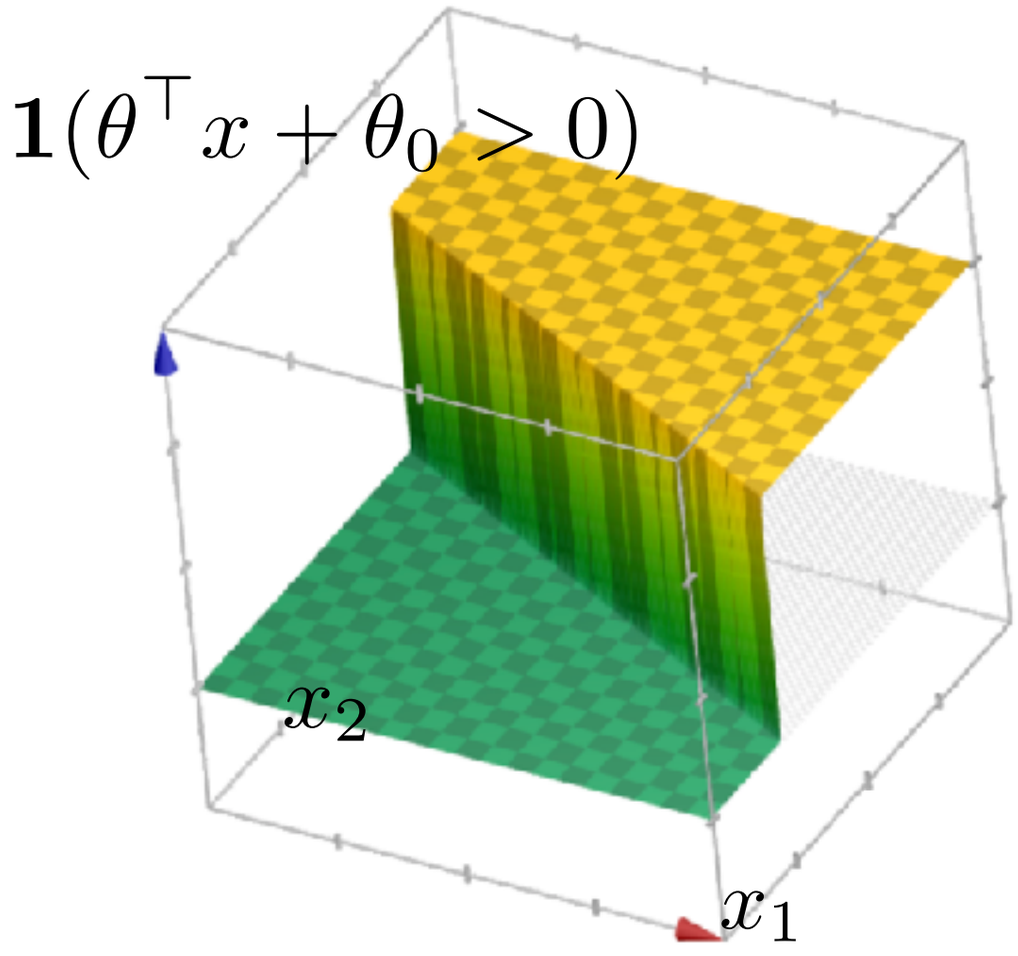
classifying hyperplane is always planar. It's always going to be at z=0. Doesn't matter how many default features are there. Doesn't matter ~~too~~ if they are linear or polynomial basis or anything else. The classifying curve will always be planar (straight/linear).

Recall

- Linear classification with default features:







- We're used to using step functions to classify
- New idea today: we'll use step functions as *features*, with their own parameters

★ Recall: (Slide-2-Last)

(Step functions)

- Imagine you have 2 default features x_1 & x_2 & the data is linear i.e you don't have to apply any other function on the x_1 & x_2 like polynomial basis or something. The feature vector is just $x = [x_1, x_2]$
- So, fig 2. is where you have the plane $z = \theta^T x + \theta_0$ which fits the data & the classifier $z = 0$.

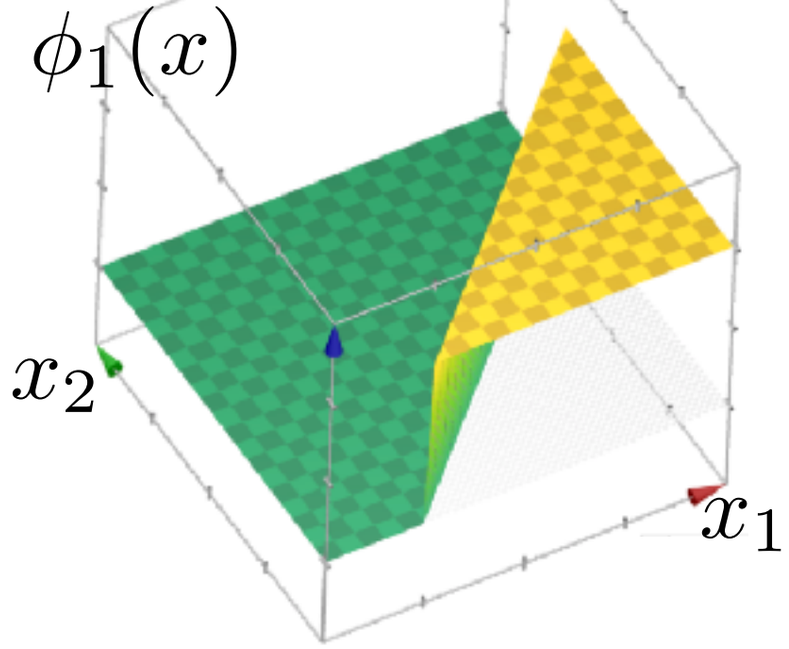
★★

- In fig 3. you can assume the function 'f' where $f = \mathbb{1}(\theta^T x + \theta_0 > 0)$ i.e. $f = 1$ if $\theta^T x + \theta_0 > 0$ & $f = 0$ otherwise. This f is called a step function. We've basically used this function previously for classifying ~~linear~~ data. (Note:- Even for polynomial boundaries, you had $f = 1(\theta^T \phi(x) + \theta_0 > 0)$)
- Thus, this 'f' step function was the classifier which predicted the labels.
- $z = 0$ was the classifier for $z = \theta^T x + \theta_0$ & f was the label predictor where $f = \mathbb{1}(\theta^T x + \theta_0 > 0)$

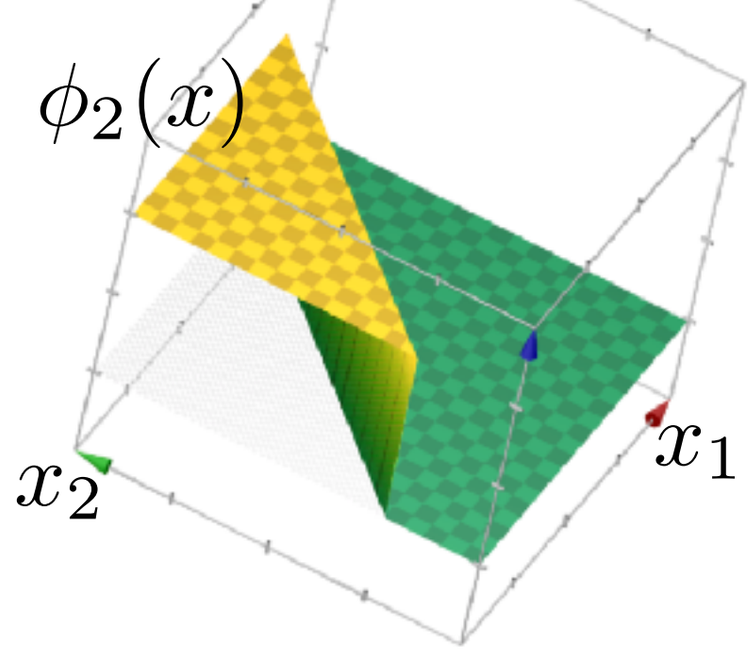
IMP

New features: step functions!

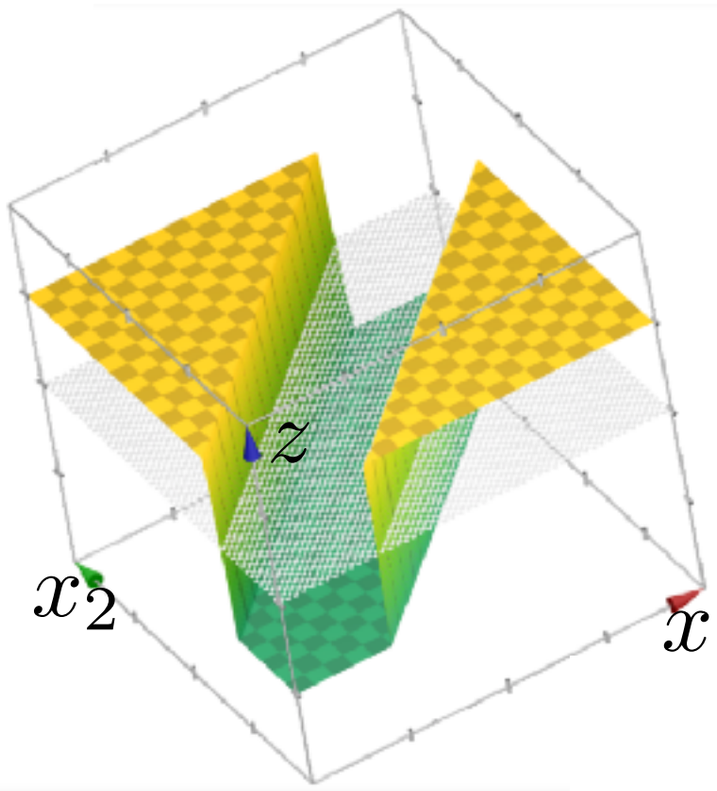
$$\phi_1(x) = \mathbf{1}\{w^\top x + w_0 \geq 0\}$$



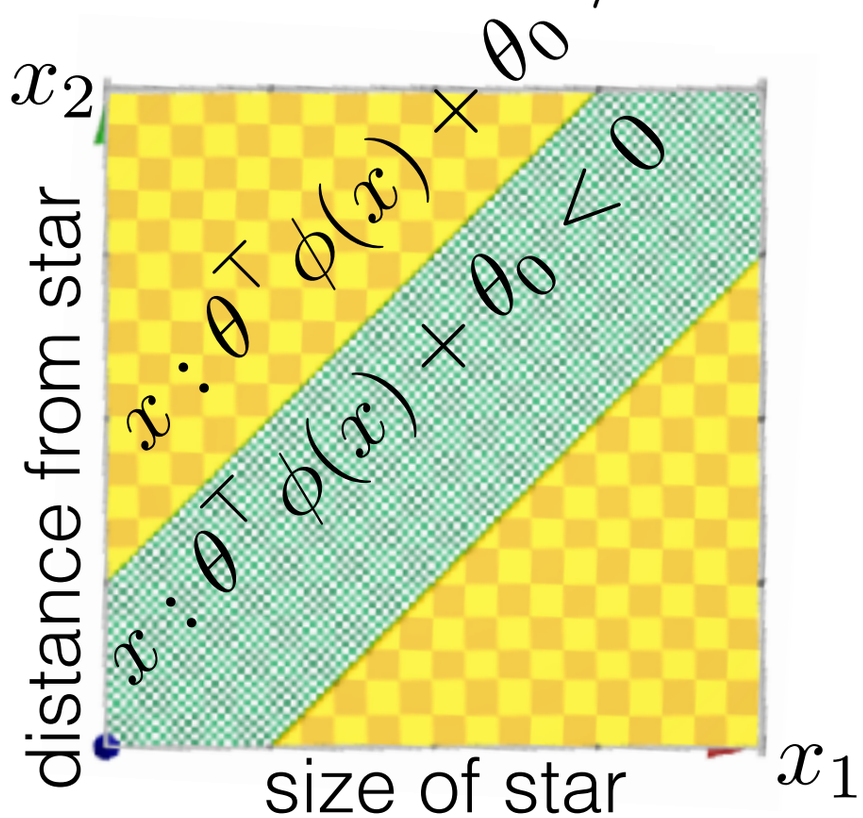
$$\phi_2(x) = \mathbf{1}\{\tilde{w}^\top x + \tilde{w}_0 \geq 0\}$$



$$z = \theta^\top \phi(x) + \theta_0$$
$$= \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \theta_0$$
$$= 1 \cdot \phi_1(x) + 1 \cdot \phi_2(x) + (-0.5)$$

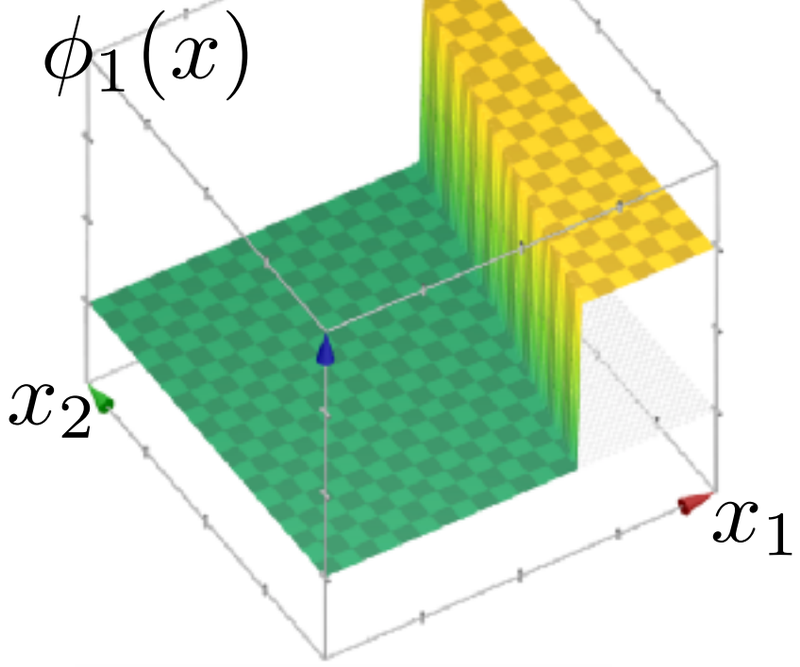


Is an exoplanet in the habitable zone?

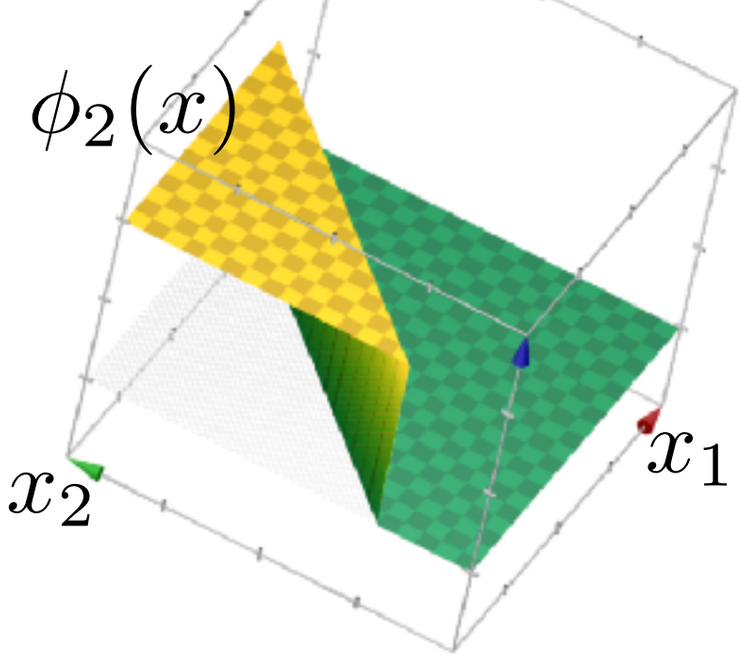


New features: step functions!

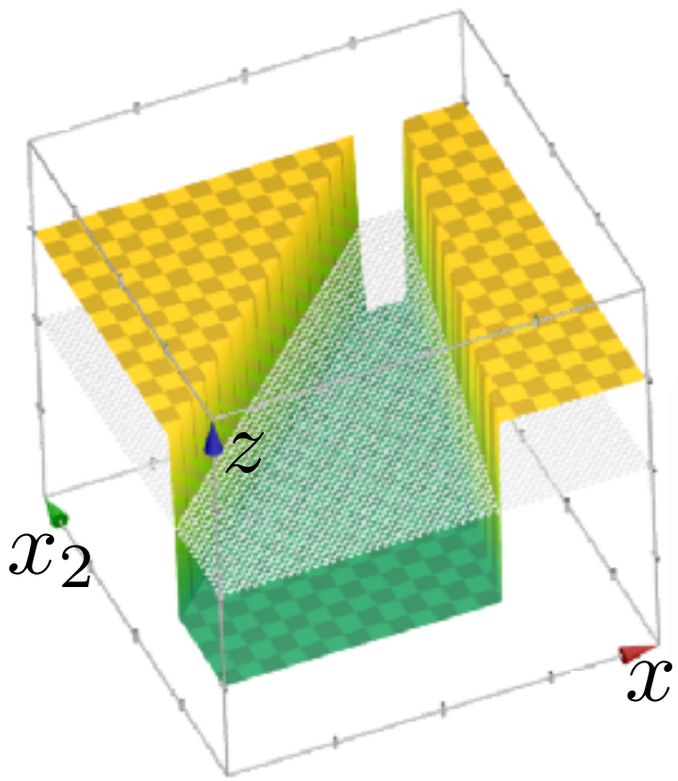
$$\phi_1(x) = \mathbf{1}\{w^\top x + w_0 \geq 0\}$$

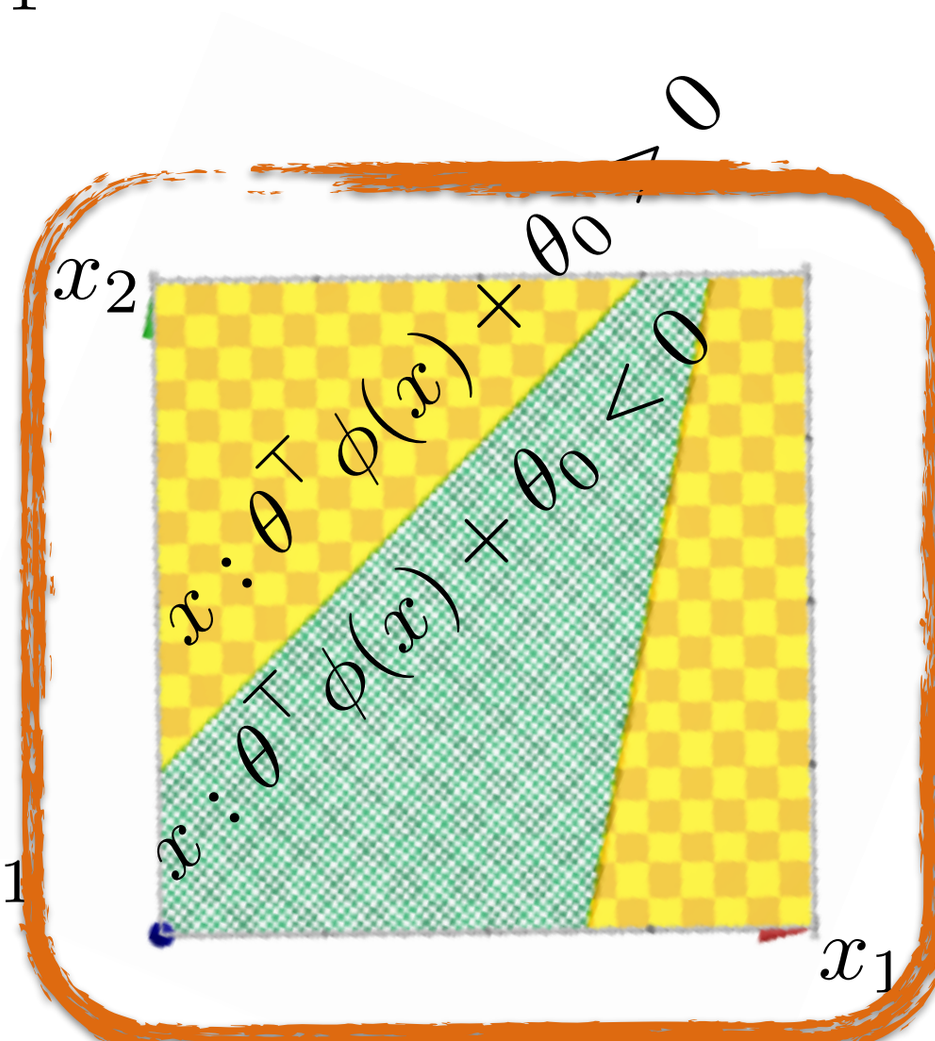


$$\phi_2(x) = \mathbf{1}\{\tilde{w}^\top x + \tilde{w}_0 \geq 0\}$$



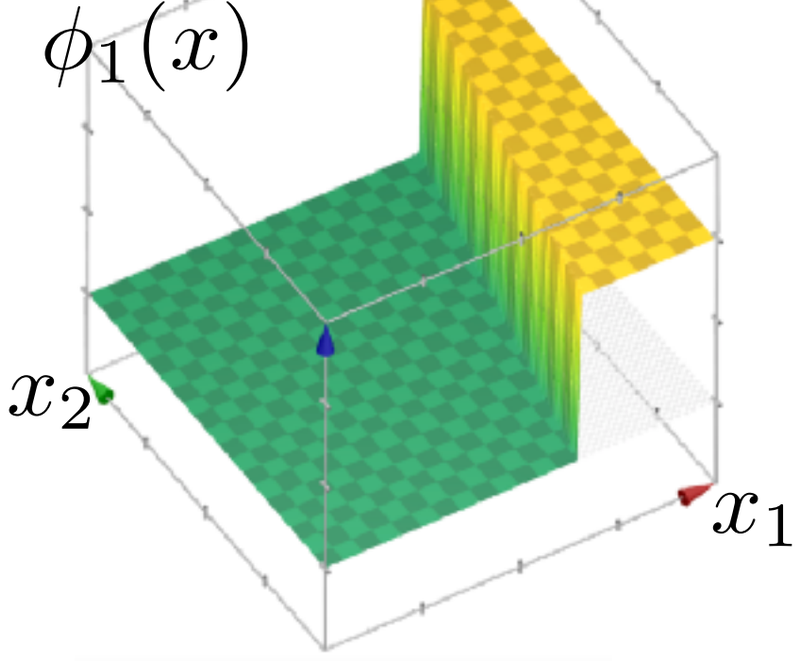
$$z = \theta^\top \phi(x) + \theta_0$$
$$= \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \theta_0$$
$$= 1 \cdot \phi_1(x) + 1 \cdot \phi_2(x) + (-0.5)$$



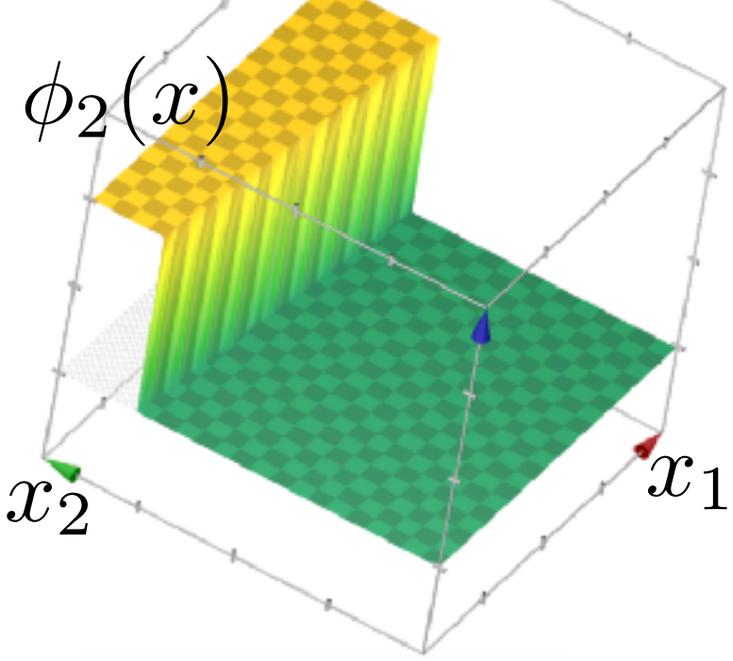


New features: step functions!

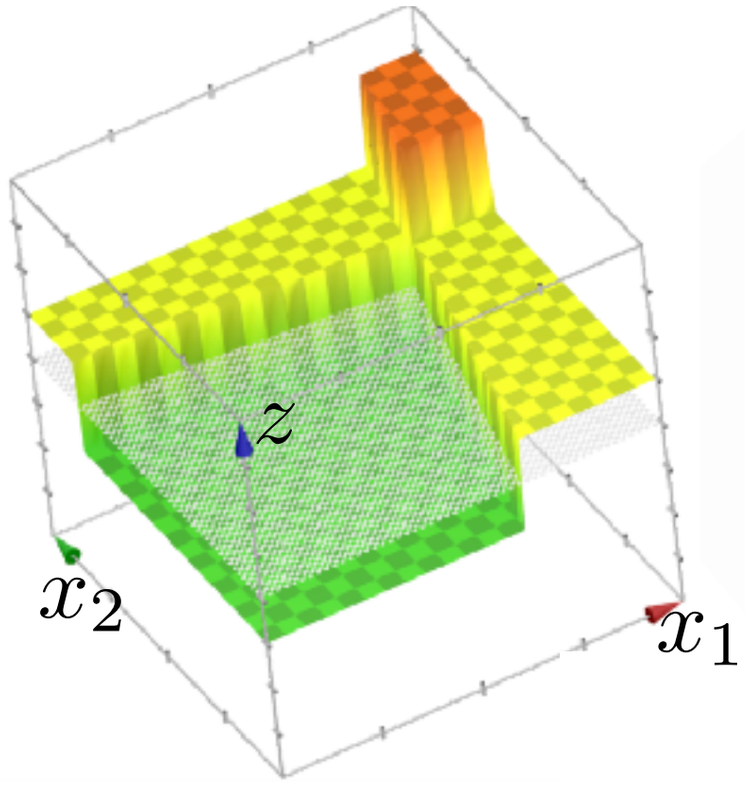
$$\phi_1(x) = \mathbf{1}\{w^\top x + w_0 \geq 0\}$$

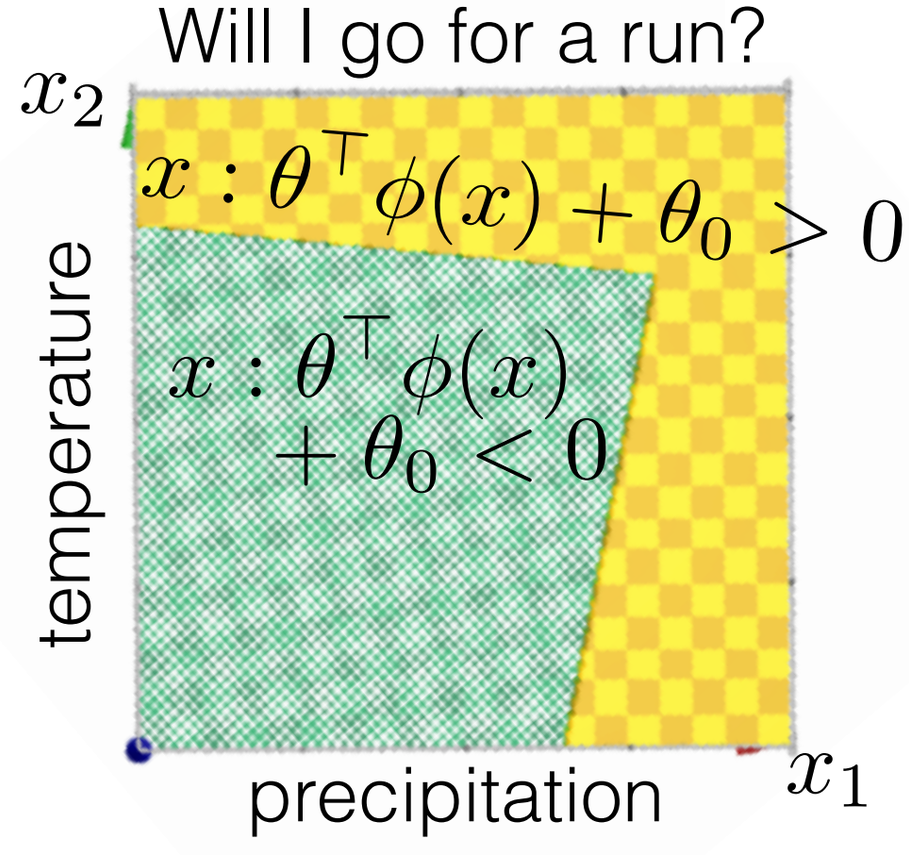


$$\phi_2(x) = \mathbf{1}\{\tilde{w}^\top x + \tilde{w}_0 \geq 0\}$$



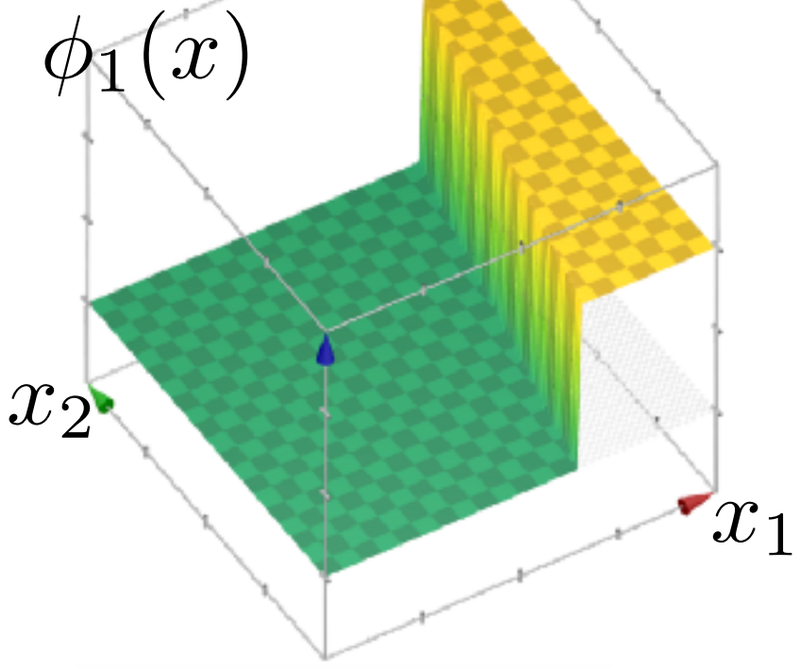
$$z = \theta^\top \phi(x) + \theta_0$$
$$= \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \theta_0$$
$$= 1 \cdot \phi_1(x) + 1 \cdot \phi_2(x) + (-0.5)$$



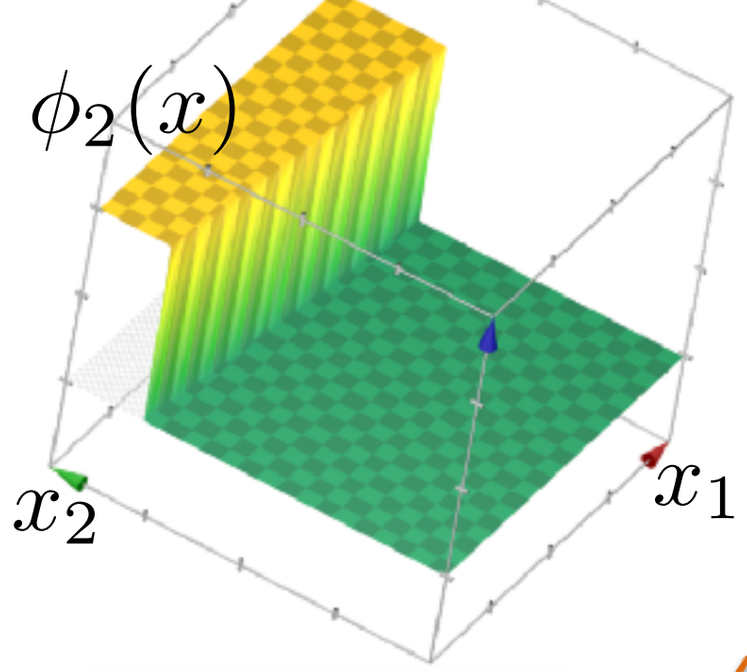


New features: step functions!

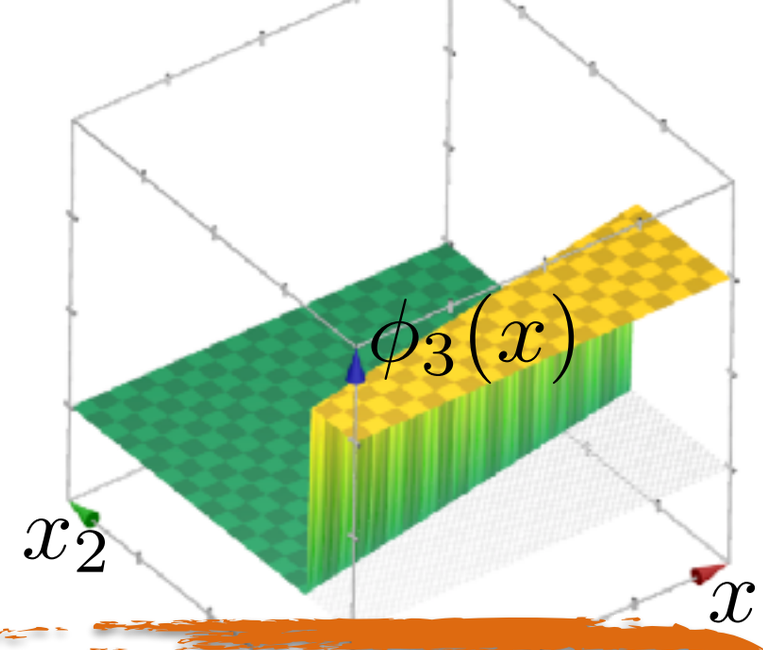
$$\phi_1(x) = \mathbf{1}\{w^\top x + w_0 \geq 0\}$$



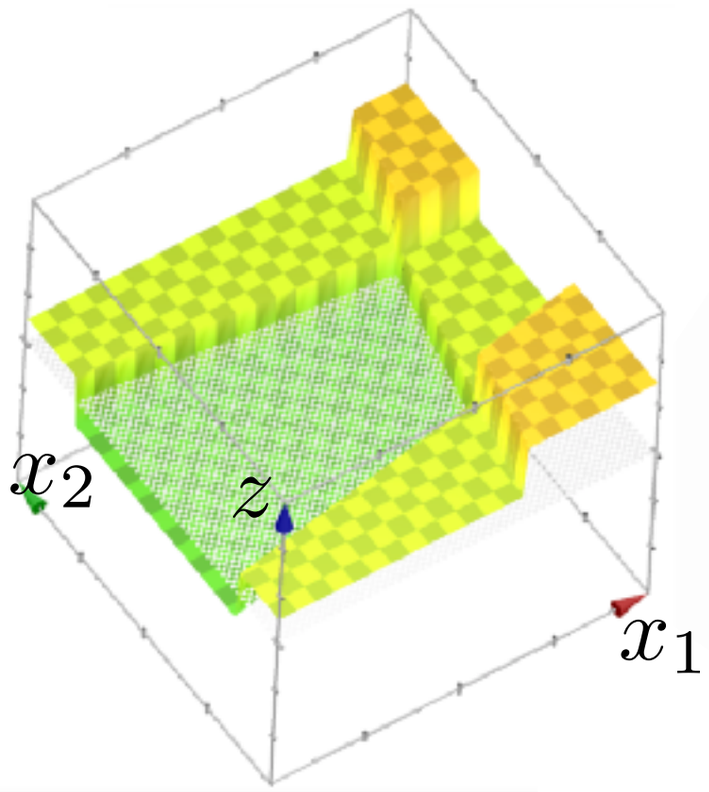
$$\phi_2(x) = \mathbf{1}\{\tilde{w}^\top x + \tilde{w}_0 \geq 0\}$$

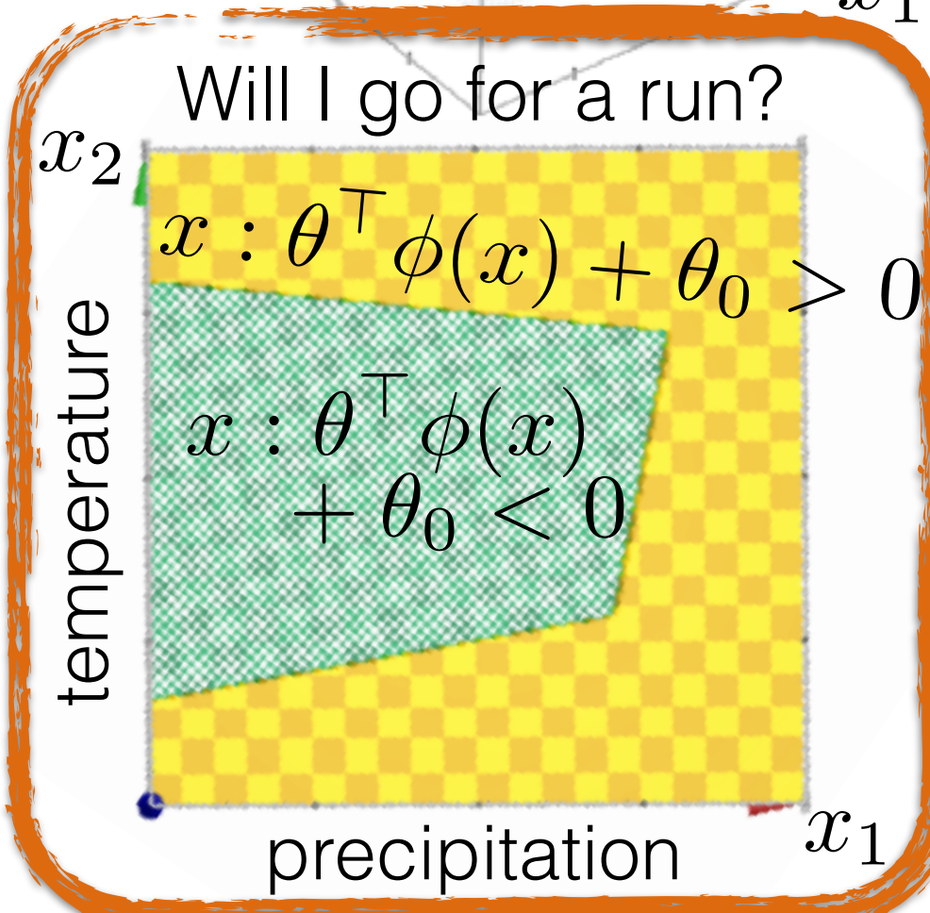


$$\phi_3(x) = \mathbf{1}\{\tilde{\tilde{w}}^\top x + \tilde{\tilde{w}}_0 \geq 0\}$$



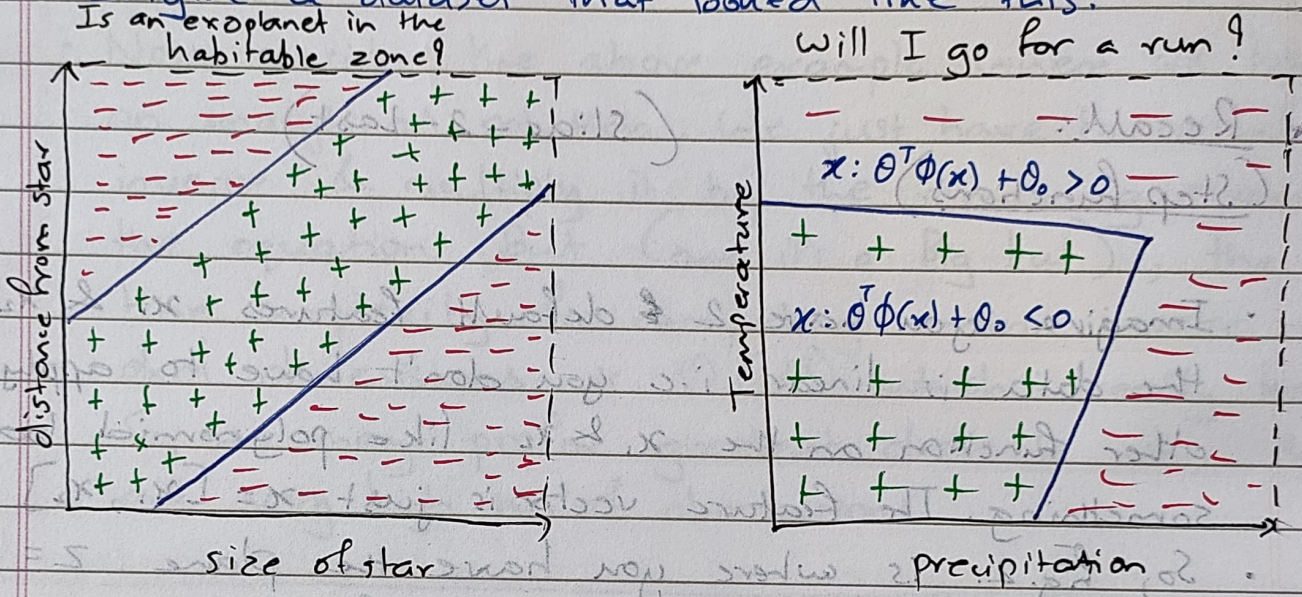
$$z = \theta^\top \phi(x) + \theta_0$$
$$= \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \theta_3 \phi_3(x) + \theta_0$$
$$= 1 \cdot \phi_1(x) + 1 \cdot \phi_2(x) + 1 \cdot \phi_3(x) + (-0.5)$$





★ New features: Step functions (Slide - 4, 5, 6, 7 - Last each)

- Previously, the linear classification that we learnt was very vanilla, even with non-linear boundaries.
- Imagine a dataset that looked like this:



- So, you can see that this data is linearly seperable. But the data isn't directly linear. We will need a non linear function / curve / boundary to fit the data.
- Now, if you see Fig 1. (from this page), you can divide the data into two. This data seems to be divided by 2 classifiers. So, we divide the data in exactly that way.
- In the 1st divided part, we consider the upper classifie Above that classifier is -ve label & below is mix of +ve & -ve. We assume that all labels below are +ve. We train this classifier this way to get the classifier eqn.
- Now, we do the same for the 2nd divided part, to get the eqn for the 2nd classifier.
- Now, we use a function that is some linear combination

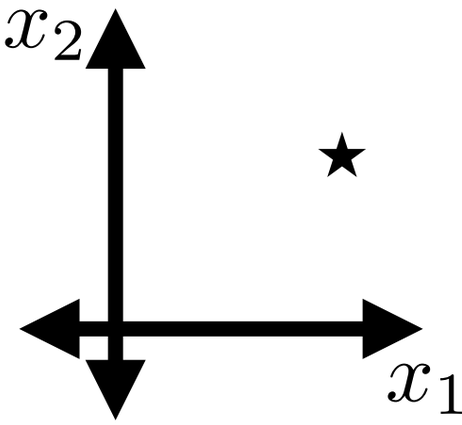
of the classifiers itself) to get the combined classifier.

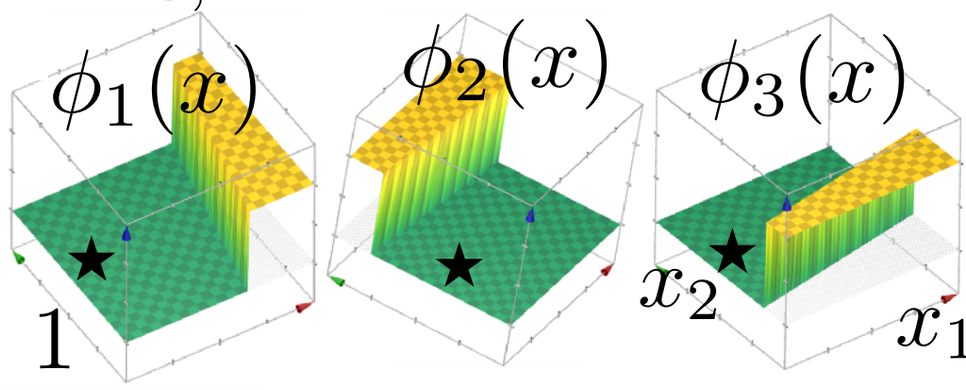
- So, we used our features to train different ~~parts of the data~~ classifier on their respective parts of the data. And then, we use these classifiers' output as new features for the combined classifier.
- There! You just defined a neural network. The first layer was the step functions (different classifiers on their respective chunks of data) & the 2nd layer was the combined classifier.

Let's get some new notation

- 1st layer, constructing the features:
  - Input x (a data point): size $m^{(1)} \times 1$
  - Output $A^{(1)}$ (vector of features): size $n^{(1)} \times 1$
  - The ith feature: $A_i^{(1)} = f^{(1)}(w_i^{(1)\top} x + w_{0,i}^{(1)})$
    - All the features at once:
      - $A^{(1)} = f^{(1)}(W^{(1)\top} x + W_0^{(1)})$
      - $W^{(1)} : m^{(1)} \times n^{(1)}; W_0^{(1)} : n^{(1)} \times 1$

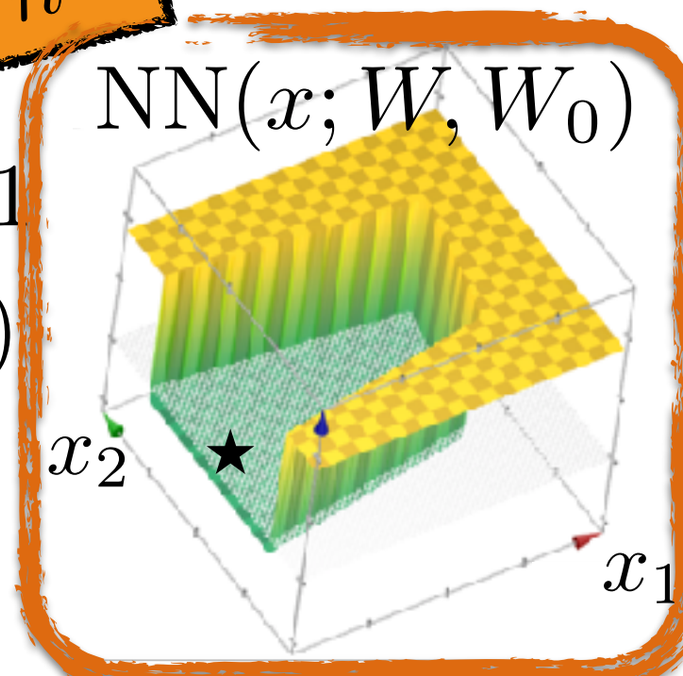
$m^{(1)} = d$





- 2nd layer, assigning a label (or labels):
  - Input (the features): size $m^{(2)} \times 1$
  - Output $A^{(2)}$(vector of labels): size $n^{(2)} \times 1$
  - The ith label: $A_i^{(2)} = f^{(2)}(w_i^{(2)\top} A^{(1)} + w_{0,i}^{(2)})$
    - All: $A^{(2)} = f^{(2)}(W^{(2)\top} A^{(1)} + W_0^{(2)})$
    - $W^{(2)} : m^{(2)} \times n^{(2)}; W_0^{(2)} : n^{(2)} \times 1$

$m^{(2)} = n^{(1)}$



★ Let's get some new notations (Slide - 8 - Last)

- Consider a dataset with only 2 features: x_1 & x_2
  Thus, $x^T = [x_1, x_2]$ & $x \in \mathbb{R}^{d \times 1}$ & $d = 2$ for now.

- Now consider the 1st Layer :-

→ The input to the 1st layer is x (a single data point). The input size is noted by $m^{(1)} \times 1$. The superscript (1) denotes 1st layer. So, you can see that $m^{(1)} = d = 2$ in this case.

→ The output of 1st layer is $A^{(1)}$ (vector of new features) (~~output~~ vector of ~~output of~~ evaluated output of step functions) $A^{(1)}$ has size: $n^{(1)} \times 1$. ('n' has nothing to do with or isn't the no. of data points) Now, $n^{(1)}$ is ~~not~~ really the no. of step functions we use. That really depends on the data. In this case (in this slide), we have 3 step functions. Thus, $n^{(1)} = 3$

$A^{(1)}$ is a column vector with $n^{(1)}$ elements.

$A_i^{(1)}$, $i = 1, 2, 3, \ldots n^{(1)}$ are the elements.

→ Now, $A_i^{(1)} = f^{(1)} \phi_i(x)$ i.e. the evaluating function $f^{(1)}$ applied on $\phi_i(x)$.

→ $\phi_i(x) = w_i^T x + w_{0i}$ (We've just replaced θ with w)
Thus, $\phi_i(x)$ results in a scalar value. In this case for linear classification, $f^{(1)}$ applied on $\phi_i(x)$ is 1 if $\phi_i(x) > 0$ & 0 if $\phi_i(x) < 0$

→ Thus, $A_i^{(1)} = f^{(1)}(\phi_i(x)) = f^{(1)}(w_i^T x + w_{0i})$

→ Now, we write $A^{(1)} = f^{(1)}(W^{(1)T} x + W_0^{(1)})$
where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ of size $d \times 1$

$W_0^{(1)} = \begin{bmatrix} w_{01}^{(1)} \\ w_{02}^{(1)} \\ \vdots \\ w_{0n^{(1)}}^{(1)} \end{bmatrix}$ = collection of all w_{0i} into a column vector of size $n^{(1)} \times 1$.

$W^{(1)} = [w_1^{(1)} \; w_2^{(1)} \; w_3^{(1)} \ldots w_{n^{(1)}}^{(1)}]$ where, $w_i^{(1)}$ is itself a column vector of size $d \times 1$.

So, $W^{(1)}$ is collection of all $w_i^{(1)}$.
$W^{(1)}$ has size $d \times n^{(1)}$ i.e. $m^{(1)} \times n^{(1)}$

→ ~~Now, $A^{(1)}$ has same size as $(W^{(1)T} x + W_0^{(1)})$ which is $n^{(1)} \times 1$.~~

→ ~~Now, we are applying~~ Now, $(W^{(1)T} + W_0^{(1)})$ has size $n^{(1)} \times 1$ & then we are applying the $f^{(1)}$

function on that vector.

→ A function applied on a vector can mean different things. It's not necessarily applied on each element. It may be different. BUT, in this case, $f^{(1)}$ is applied element-wise i.e. on each element.

→ Thus, $A^{(1)}$ will have same size as $(W^{(1)T}x + W_0^{(1)})$ which is $n^{(1)} \times 1$.

• Now, consider the 2nd Layer:-

→ Now, the input to the 2nd layer is going to be output of the 1st layer. So input for 2nd layer is $A^{(1)}$. The size of input of 2nd Layer is denoted by $m^{(2)} \times 1$. (Basically, input size is $m^{(layer)} \times 1$ & output size is $n^{(layer)} \times 1$). So, you can see that $m^{(2)} = n^{(1)} = 3$ in this case.

→ Now the output of 2nd layer will be called $A^{(2)}$ (Basically output is always $A^{(layer)}$) This $A^{(2)}$ is going to be a vector of labels.

→ Till now, we have only predicted 1 label in linear classification (Basically, we answer only 1 question). But, from the same data, we can predict multiple set of labels (Basically, answer many Yes/No questions based on a single dataset). Generally & especially in this case, we only predict one label. Thus, $A^{(2)}$ will have only one value.

→ The size of $A^{(2)}$ is given by $n^{(2)} \times 1$. As, we saw in the previous point, $n^{(2)} = 1$. Thus, $A^{(2)}$ has size 1×1.

→ This entire function consisting of these 2 layers is called the neural network and is denoted by $NN(x; W, W_0)$

→ Now, let's forget for the moment that we're only predicting 1 ~~bot~~ label. We'll assume an arbitrary $n^{(2)}$. So, $A^{(2)}$ has size $n^{(2)} \times 1$.

→ Thus, $A^{(2)}$ is collection of $A_i^{(2)}$ where $A_i^{(2)}$ is a label. So, we can write $A_i^{(2)}$ as:

$$A_i^{(2)} = f^{(2)}\left(w_i^{(2)T} A^{(1)} + w_{0i}^{(2)}\right)$$

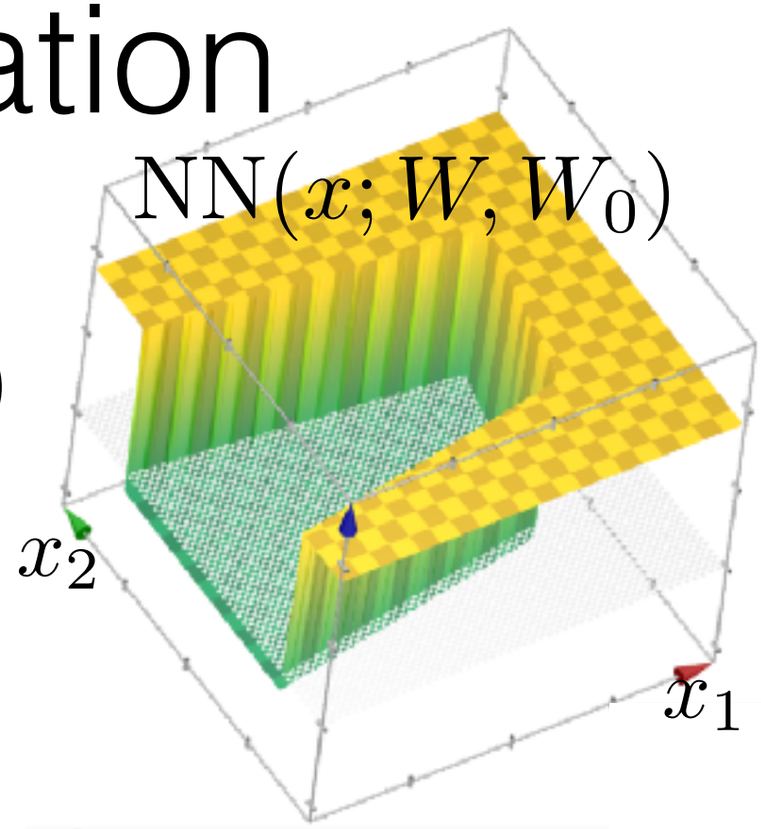
So, we collect $A_i^{(2)}$ together into $A^{(2)}$ & collect all $w_i^{(2)}$ & $w_{0i}^{(2)}$ into their respective matrices: $W^{(2)}$ & $W_0^{(2)}$ where $W^{(2)}$ has size $m^{(2)} \times n^{(2)} = n^{(1)} \times n^{(2)}$ & $W_0^{(2)}$ has size $n^{(2)} \times 1$.

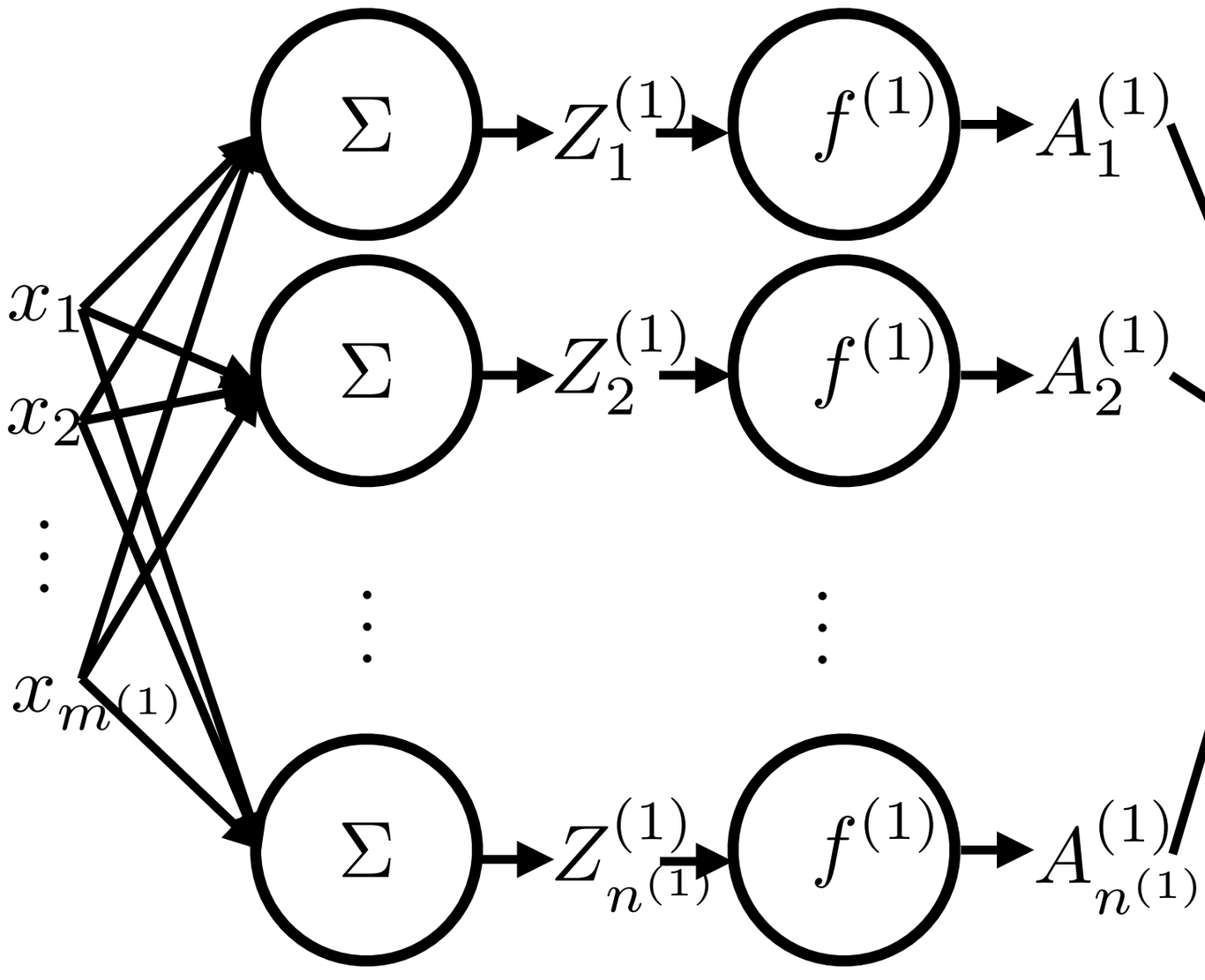
→ So, now we can write $A^{(2)}$ as:

$$A^{(2)} = f^{(2)}\left(W^{(2)T} A^{(1)} + W_0^{(2)}\right)$$

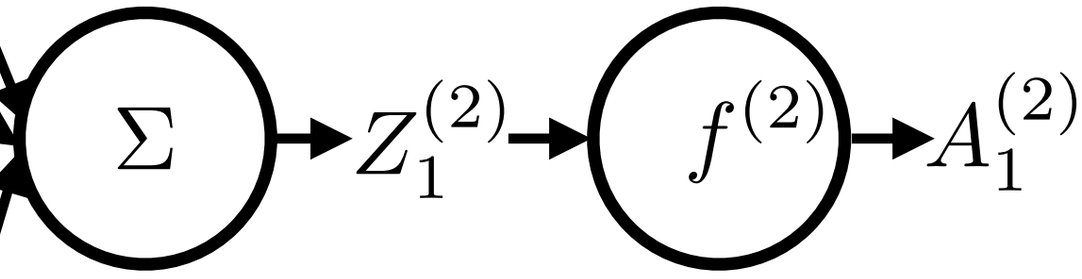
Function graph representation

- 1st layer: $A^{(1)} = f^{(1)}(W^{(1)\top} x + W_0^{(1)})$
- 2nd layer: $A^{(2)} = f^{(2)}(W^{(2)\top} A^{(1)} + W_0^{(2)})$
- Whole thing: $A^{(2)} = \mathrm{NN}(x; W, W_0)$





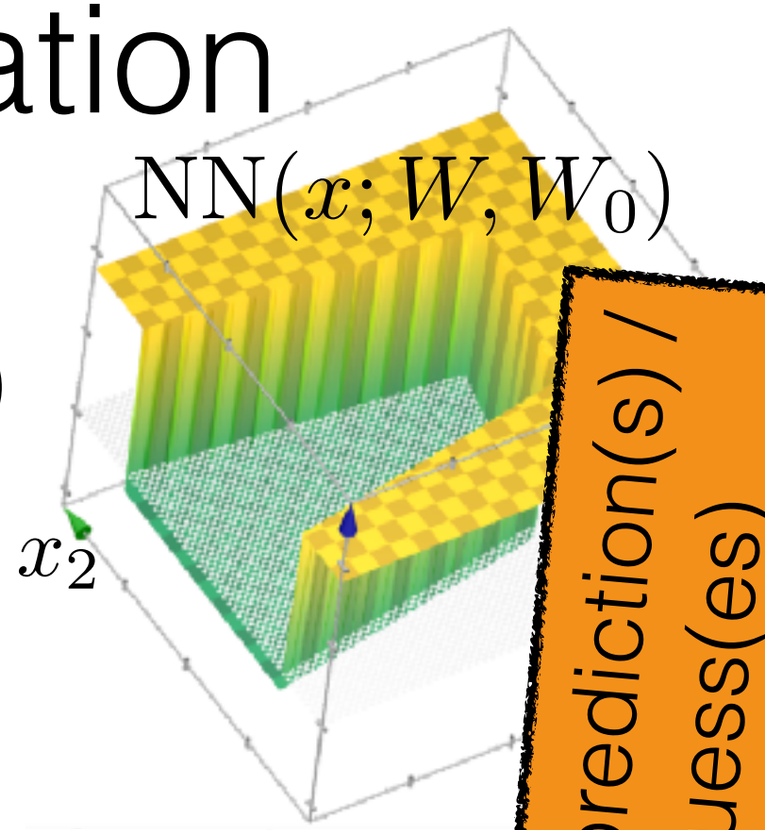
- Circle: function evaluation

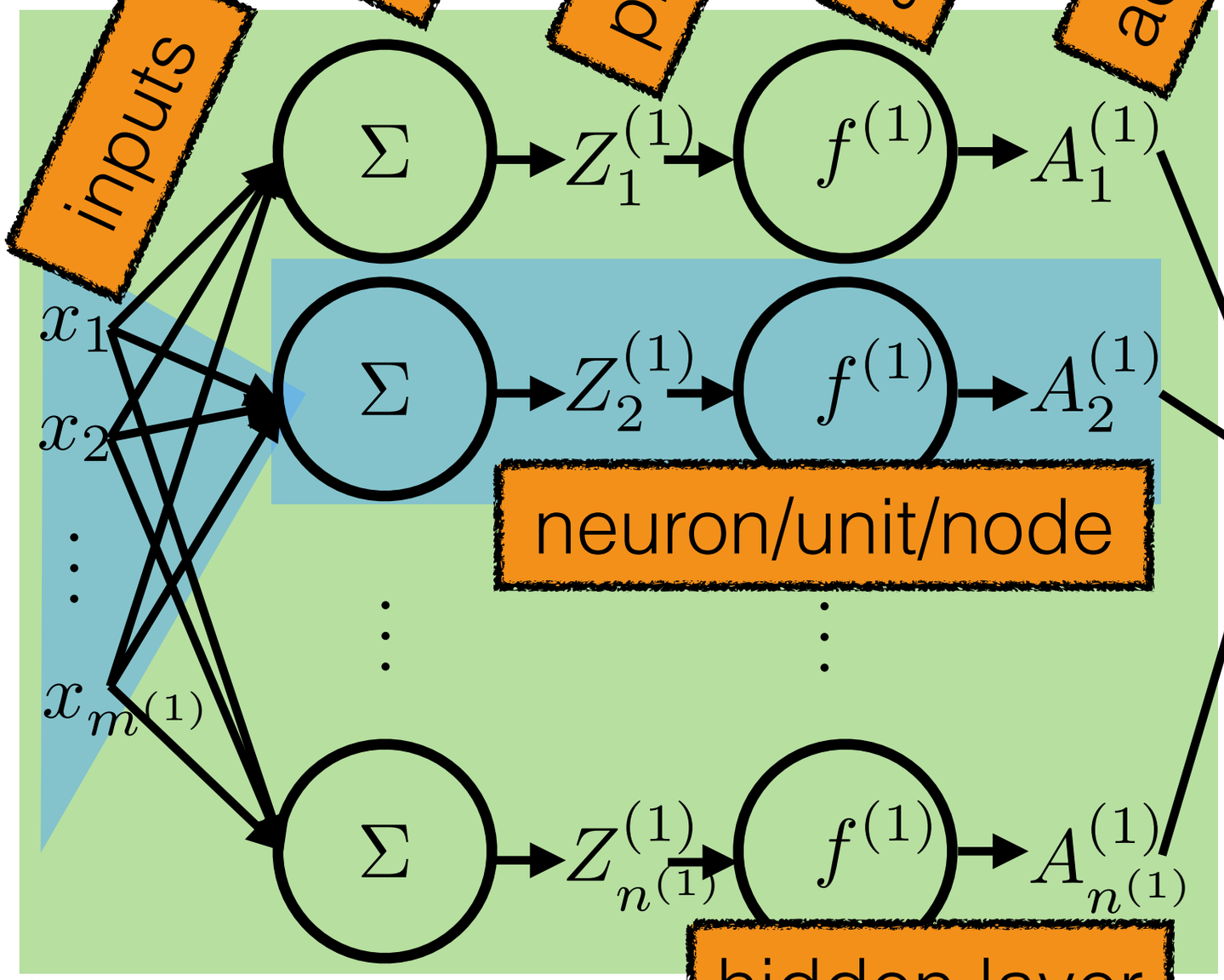


- Forward vs. backward
- A feed-forward neural network

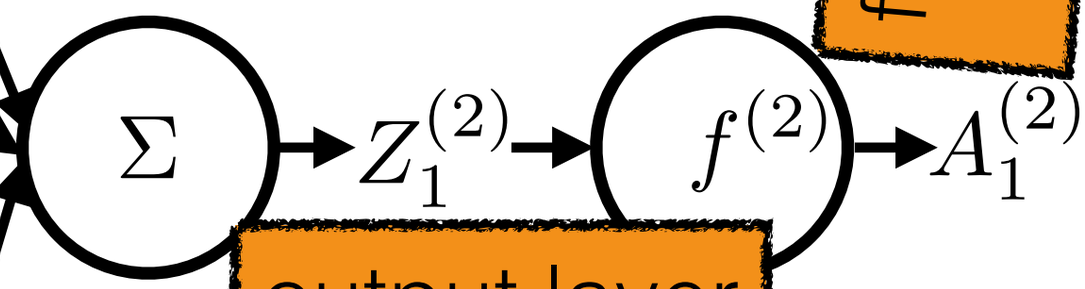
Function graph representation

- 1st layer: $A^{(1)} = f^{(1)}(W^{(1)\top} x + W_0^{(1)})$
- 2nd layer: $A^{(2)} = f^{(2)}(W^{(2)\top} A^{(1)} + W_0^{(2)})$
- Whole thing: $A^{(2)} = \mathrm{NN}(x; W, W_0)$





- Circle: function evaluation



- Forward vs. backward
- A feed-forward neural network
- Fully connected

* Function Graph Representation (slide - 9 - 128, 141)

- Circular notation or something written in a circle is a function evaluation. For ex; in this slide the Σ is put in a circle to denote / indicate the dot product. Since we sum all the individual products, we've put the Σ symbol. Similarly, $f^{(1)}$ is also put in a circle.
- The $z_i^{(1)}$ is the output of 'Σ' function & is input to the '$f_\theta^{(1)}$' function. Both $z_i^{(1)}$ & $A_i^{(1)}$ are scalars

- These neural networks also have directionality. It may be forward or backward or hybrid.
  Backward direction is used for inferrence (Bayesian statistics i.e. we may infer some initial things based on the final results)
- This particular network is a feed forward network because all directions go forward. For ex: Recurrent neural networks aren't feed forward.

- $f^{(1)}$ is called the activation function because it sort of decides whether $A_i^{(1)}$ will be active or not (1 or 0). So, $A_i^{(1)}$ is called activation & $z_i^{(1)}$ is called pre-activation

- A neuron / node / unit is a sequence that goes from inputs to activation. A layer is a collection of such neurons.

- This is also a fully connected network because the inputs to all neurons in a layer are the same.

Problem setup

1. Choose a hypothesis class. E.g.,

$$h(x; W, W_0) = \mathrm{NN}(x; W, W_0)$$

   - 1st layer: $A^{(1)} = f^{(1)}(W^{(1)\top} x + W_0^{(1)})$
   - 2nd layer: $A^{(2)} = f^{(2)}(W^{(2)\top} A^{(1)} + W_0^{(2)})$

2. Choose a loss. E.g. for classification: 0-1 loss, asymmetric, negative log likelihood
3. Learn the parameters. E.g. gradient descent or SGD
4. Predict on new data using these parameters

Issues:

- Derivatives are zero (or undefined) if we use the step function activation, so (S)GD won't do what we want
- What if I want to do regression?
- What if I want to use NLL loss?

★ Problem Setup (Slide-10-Last)

1) Choose a hypothesis class like linear classifiers, Neural networks, etc.
2) Choose a type of loss: 0-1 loss, assymetric loss, negative log likelihood loss, mean squared
3) Learn the parameters using methods like gradient descent or stochastic gradient descent.
4) Predict on new data using these parameters.

Issues:-

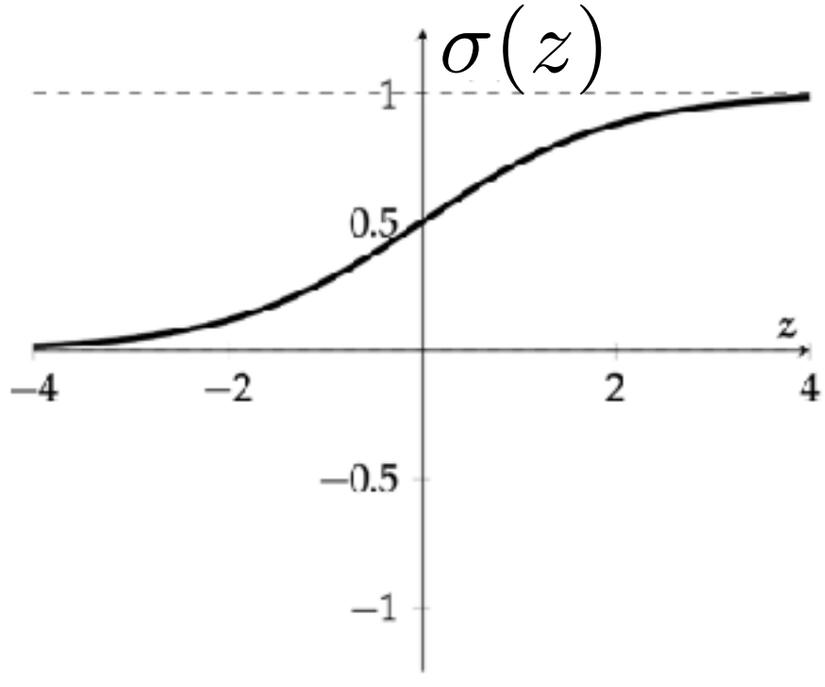
- Derivatives are 0 or undefined for step functions
- Can't directly use regression because of the 0-1 evaluating functions $f^{(1)}$ & $f^{(2)}$
- Can't plug 0 or 1 directly in NLL. We want likelihood

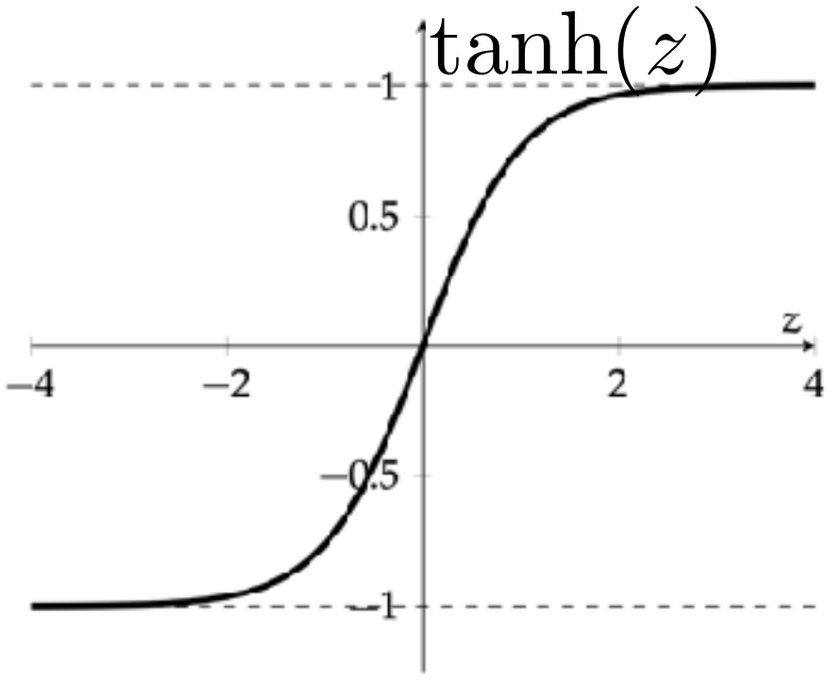
NOTE:- To get around these issues, you could just use different activation functions!

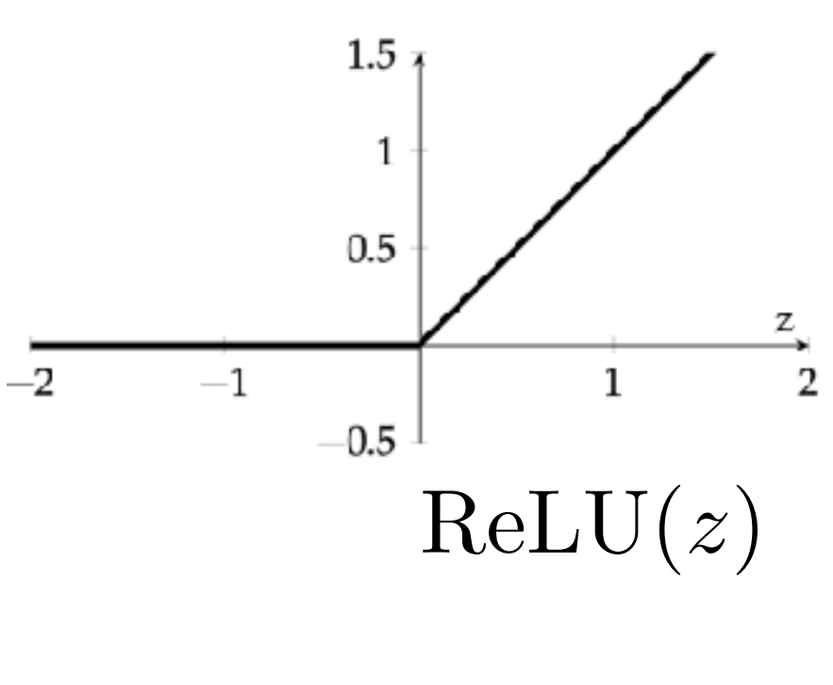
Different activation functions

1. Hypotheses $h(x; W, W_0) = \mathrm{NN}(x; W, W_0)$
   - 1st layer: $A^{(1)} = f^{(1)}(W^{(1)\top} x + W_0^{(1)})$
   - 2nd layer: $A^{(2)} = f^{(2)}(W^{(2)\top} A^{(1)} + W_0^{(2)})$

- What if I want to do regression? $f^{(2)}(z) = z$
- What if I want to use NLL loss? $f^{(2)}(z) = \sigma(z)$
- Need non-zero derivatives for (S)GD: Above & $f^{(1)}(z) =$







* Different activation functions (Slide - 11 - Last)

- To use regression, you could just diminate the 0-1 evaluation from $f^{(2)}$ such that $f^{(2)}(z) = z$ i.e. input = output. (Basically, make $f^{(2)}$ useless)

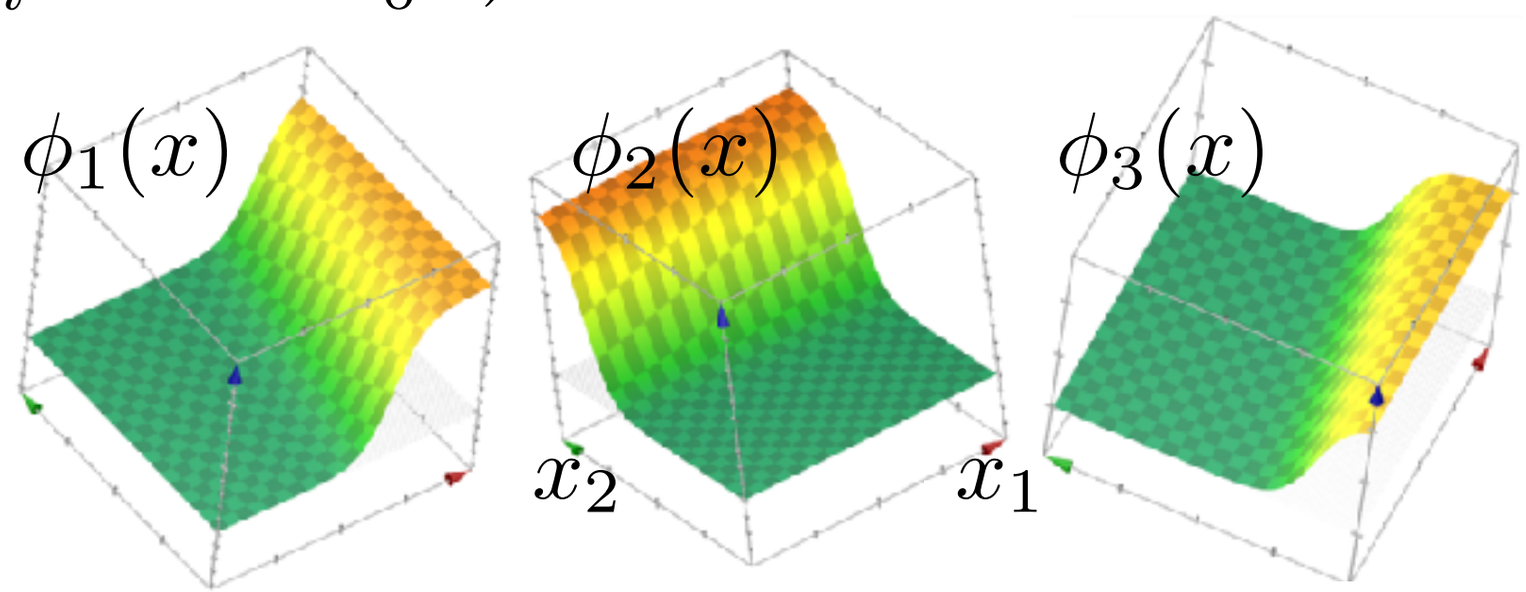
- For NLL loss, you need values between 0 & 1. So, you could just use sigmoid. $f^{(2)}(z) = \sigma(z)$

- For gradient descent or S.G.D., we need non-zero derivatives & need $f^{(2)}$ to be differentiable & also $f^{(1)}$ need to be differentiable.
- $f^{(2)}$ ~~is the actual~~ would be the actual regression function if we decided to do regression. $f^{(1)}$ would be a function to choose raw ~~data~~ feature combinations to get new features.

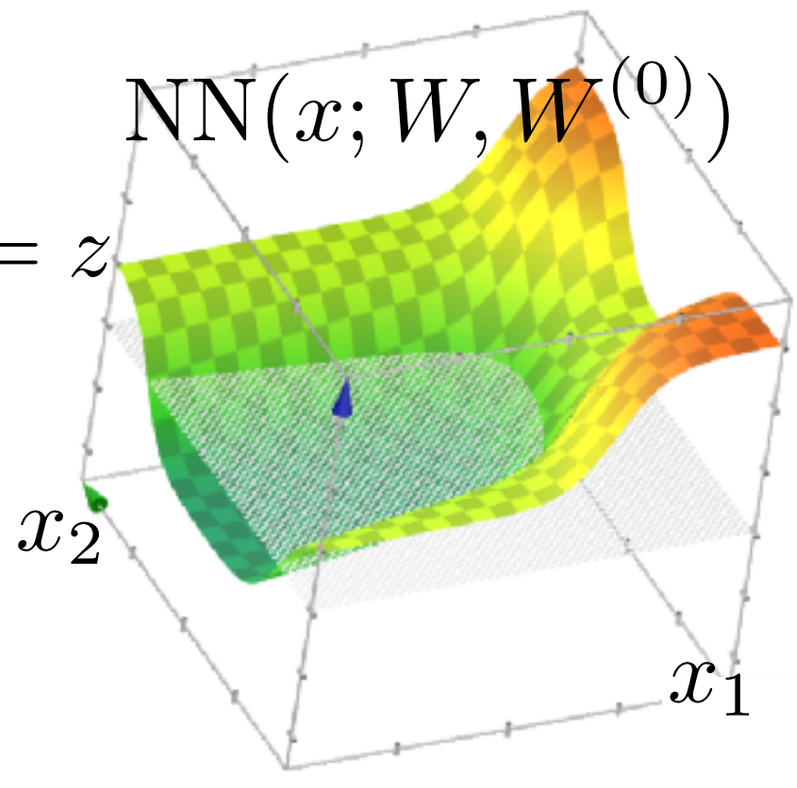
- To choose $f^{(1)}$ for gradient descent, you could choose any of these or any other good functions: $f^{(2)}(z) = z$, $f^{(2)}(z) = \sigma(z)$, $f^{(2)}(z) = \tanh(z)$, $f^{(2)}(z) = ReLU(z)$, etc.

Choices of activation function

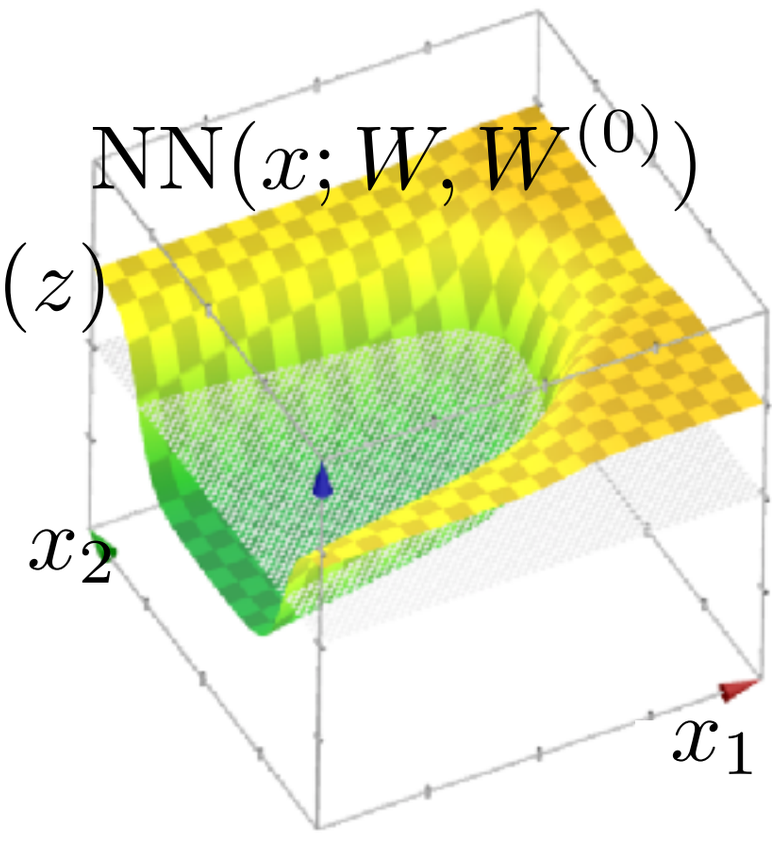
- 1st layer: $A_i^{(1)} = f^{(1)}(w_i^{(1)\top} x + w_0^{(1)})$
  - Choose $f^{(1)}(z) = \sigma(z)$



- 2nd layer: $A_i^{(2)} = f^{(2)}(w_i^{(2)\top} A^{(1)} + w_0^{(2)})$
  - Choose $f^{(2)}(z) = z$



  - Choose $f^{(2)}(z) = \sigma(z)$



★ Choices of activation function (Slide - 12 - 182)

- If we choose $f^{(1)}(z) = \sigma(z)$ as our 1st activation function (activation function for 1st layer) we get these nice smooth differentiable curves as weights.

Note:- Yes! The $\phi_i(x)$ are the weights. The step functions are the weights.

- Now, for the 2nd layer if we choose $f^{(2)}(z) = z$ then we get this nice curve shown in the left fig in the slide. Note that these values (weights) aren't discrete. So, in the overlap of the ~~step functions~~ weights (or the region where the ~~step functions~~ / weights overlap) the values of $f^{(2)}(z)$ are higher. In the non overlapping region, but where we still predict +1, we get values that are lower than the overlap, but still

high enough to predict +1.

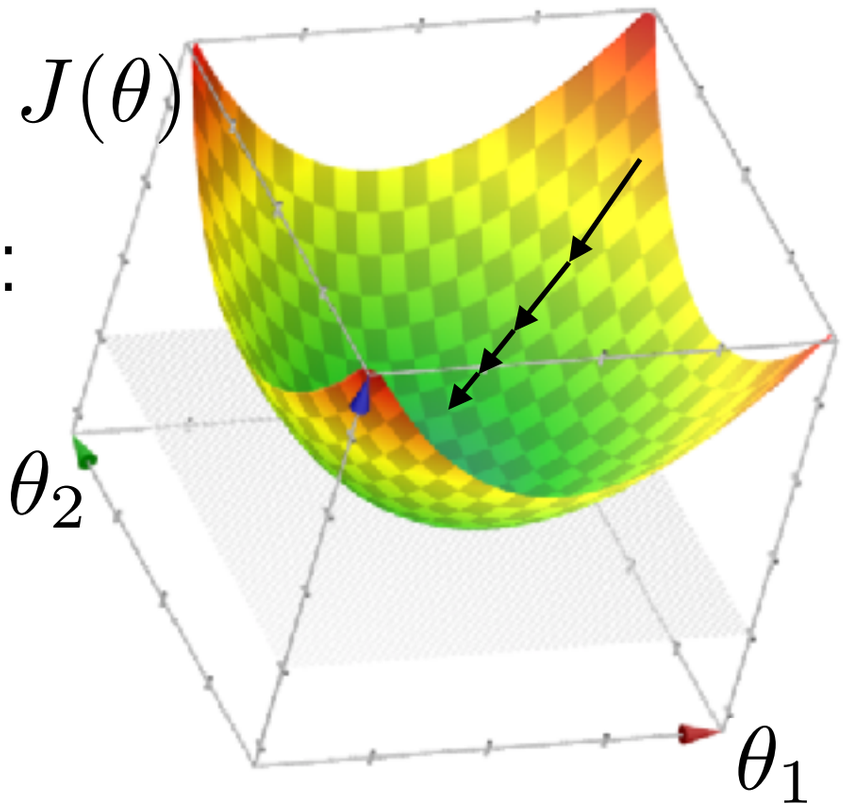
- So, you can see that $f^{(2)}(z) = z$ predicts correctly, but doesn't explain about high values or how to interpret their meaning in the overlapping region.

- BUT, if you use $f^{(2)}(z) = \sigma(z)$, you can interpret their values as probabilities & you can interpret that a point lying in the overlap of the weights has a much higher probability of getting predicted +1 than the non-overlapping (& yet +1 giving values) points.

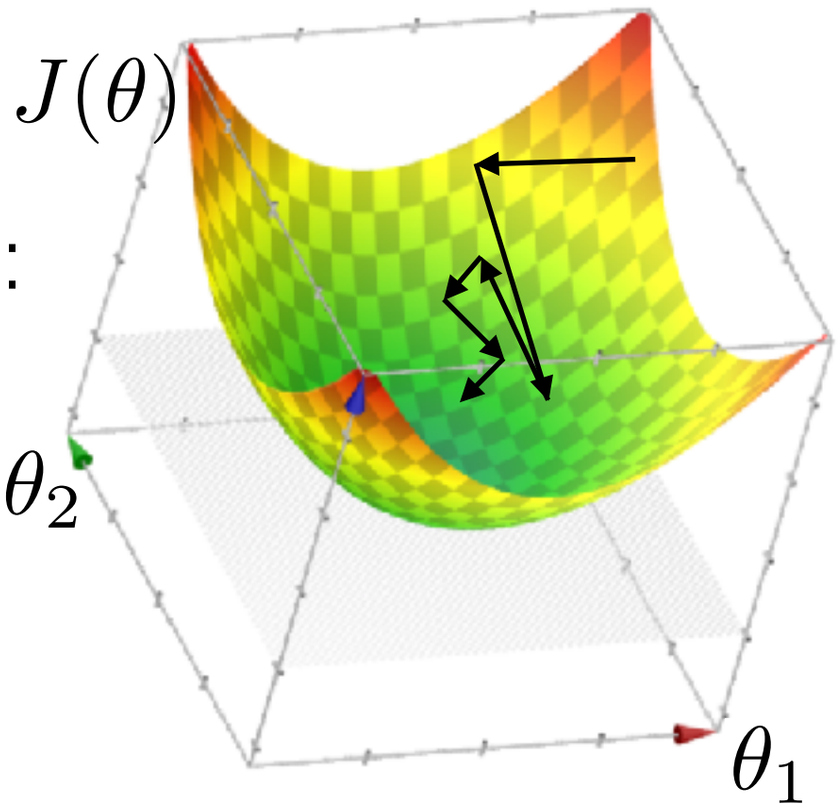
Learning the parameters

- Objective: $J(W, W_0) = \frac{1}{n_{\text{data}}} \sum_{i=1}^{n_{\text{data}}} L(h(x^{(i)}; W, W_0), y^{(i)})$

- GD:



- SGD:



- **Theorem**: (Roughly) if the objective is nice and convex, GD and SGD perform well
- **Big challenge**: the NN objective is a (very) non-convex function of the parameters (except in e.g. 1 layer)
- Huge bag of tricks to optimize / regularize

Lecture 6* continued:-

* Learning the parameters (Slide- 14-200)

- The challenge with Neural Nets:-

  → The objective function in the neural networks is super complex (not the imaginary, na type) & it's not very "nice" which typically G.D. & S.G.D. require.

  → They are very not convex and have multiple local optimums which is a problem for optimization algorithms.

  → Also, NNs are super high dimensional due to their very nature which makes them hard to visualize (Remember that visualization helps a LOT with figuring out the nature of the data)

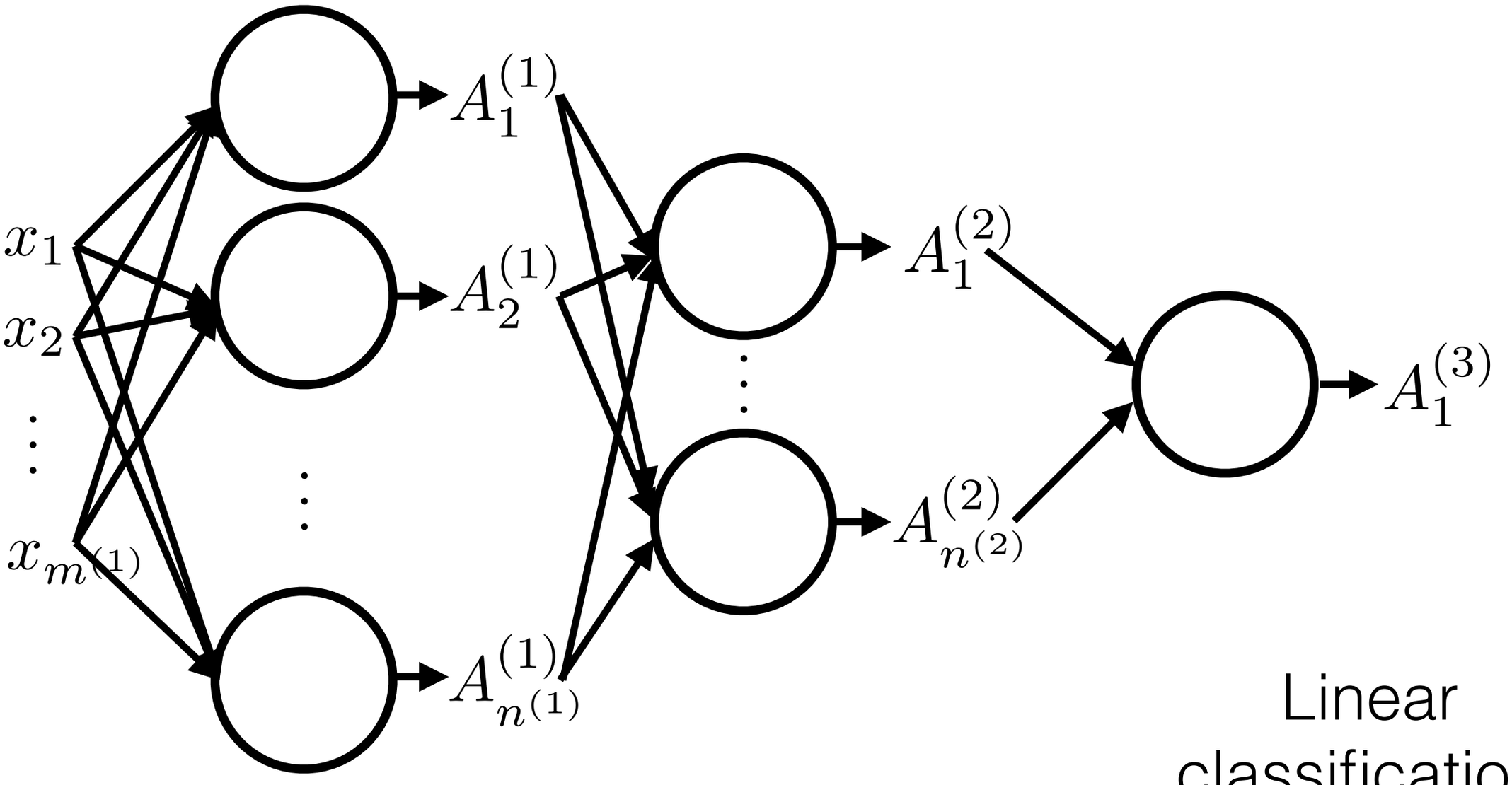
- Hence, we need to employ lots of tricks & methods

to optimize & regularize the NNs.

- Sometimes, it's even not clear if you want a global optimum ~~because of regularization~~. It relates to regularization & over fitting.

More layers!

- Why stop at 2 layers?



- Just one layer: linear classification/regression with default features