Neural Network

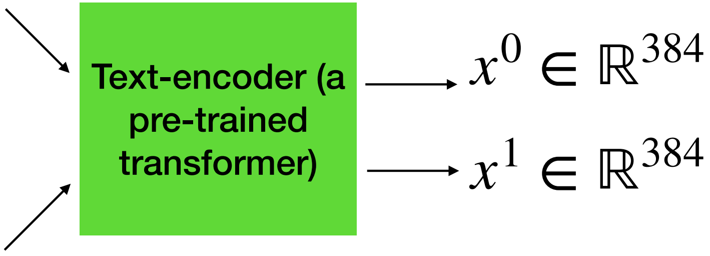
Announcements

Kaggle competition will be released today

Task: sentiment analysis

`r0`: Bromwell High is a cartoon comedy. It ran at the same time as some other programs about school life, such as "Teachers". My 35 years in the teaching profession lead me to believe that Bromwell High's satire is much closer to reality than is "Teachers". The scramble to survive financially, the insightful students who can see right through their pathetic teachers' pomp, the pettiness of the whole situation, all remind me of the schools I knew and their students. When I saw the episode in which a student repeatedly tried to burn down the school, I immediately recalled .... at .... High. A classic line: INSPECTOR: I'm here to sack one of your teachers. STUDENT: Welcome to Bromwell High. I expect that many adults of my age think that Bromwell High is far fetched. What a pity that it isn't!

`r1`: Story of a man who has unnatural feelings for a pig. Starts out with a opening scene that is a terrific example of absurd comedy. A formal orchestra audience is turned into an insane, violent mob by the crazy chantings of it's singers. Unfortunately it stays absurd the WHOLE time with no general narrative eventually making it just too off putting. Even those from the era should be turned off. The cryptic dialogue would make Shakespeare seem easy to a third grader. On a technical level it's better than you might think with some good cinematography by future great Vilmos Zsigmond. Future stars Sally Kirkland and Frederic Forrest can be seen briefly.



Preference $y \in \{0,1\}$ indicate which review is positive

Recap on Boosting

Boosting iteratively learns a new classifier, and add it to the ensemble

Initialize $H_1 = h_1 \in \mathscr{H}$

For t = 1 …

Recap on Boosting

Boosting iteratively learns a new classifier, and add it to the ensemble

$H_t = \sum_{i=1}^{t} \alpha_i h_i$

Initialize $H_1 = h_1 \in \mathscr{H}$

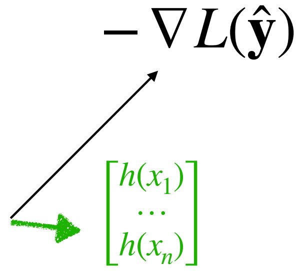
For t = 1 …

Denote $\hat{\mathbf{y}} = \left[H_t(x_1), H_t(x_2), \ldots, H_t(x_n)\right]^\top \in \mathbb{R}^n$

Recap on Boosting

$$\nabla L(\hat{y}) = \begin{bmatrix} \frac{\partial l}{\partial \hat{y}_1} \\ \vdots \\ \frac{\partial l}{\partial \hat{y}_n} \end{bmatrix}$$

Boosting iteratively learns a new classifier, and add it to the ensemble



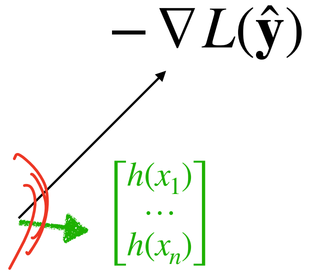
Initialize $H_1 = h_1 \in \mathscr{H}$

For t = 1 …

Denote $\hat{\mathbf{y}} = \left[H_t(x_1), H_t(x_2), \dots, H_t(x_n)\right]^\top \in \mathbb{R}^n$

Recap on Boosting

Boosting iteratively learns a new classifier, and add it to the ensemble



Initialize $H_1 = h_1 \in \mathscr{H}$

For t = 1 …

Denote $\hat{\mathbf{y}} = \left[H_t(x_1), H_t(x_2), \ldots, H_t(x_n)\right]^\top \in \mathbb{R}^n$

Solve the optimization problem: $h_{t+1} = \arg\max_{h \in \mathscr{H}} \left\langle \begin{bmatrix} h(x_1) \\ \cdots \\ h(x_n) \end{bmatrix}, -\nabla L(\hat{\mathbf{y}}) \right\rangle$

Recap on Boosting

Boosting iteratively learns a new classifier, and add it to the ensemble

$-\nabla L(\hat{\mathbf{y}})$

$\begin{bmatrix} h(x_1) \\ \cdots \\ h(x_n) \end{bmatrix}$

Initialize $H_1 = h_1 \in \mathscr{H}$

For t = 1 …

Denote $\hat{\mathbf{y}} = \left[H_t(x_1), H_t(x_2), \ldots, H_t(x_n)\right]^\top \in \mathbb{R}^n$

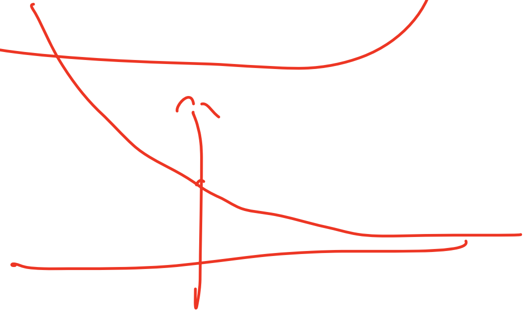
Solve the optimization problem: $h_{t+1} = \arg\max_{h \in \mathscr{H}} \left\langle \begin{bmatrix} h(x_1) \\ \cdots \\ h(x_n) \end{bmatrix}, -\nabla L(\hat{\mathbf{y}}) \right\rangle$

$H_{t+1} = H_t + \alpha h_{t+1}$

Recap on AdaBoost

Adaboost follows this framework with $\ell(\hat{y}, y) = \exp(-\hat{y} \cdot y)$

1. Create a new weighted dataset:



Recap on AdaBoost

Adaboost follows this framework with $\ell(\hat{y}, y) = \exp(-\hat{y} \cdot y)$

1. Create a new weighted dataset:

$\hat{y}_i \leftarrow \mathcal{H}_t(x_i)$

For each x_i, compute $p_i \propto \exp(-\hat{y}_i \cdot y_i)$

Recap on AdaBoost

Adaboost follows this framework with $\ell(\hat{y}, y) = \exp(-\hat{y} \cdot y)$

1. Create a new weighted dataset:

For each x_i, compute $p_i \propto \exp(-\hat{y}_i \cdot y_i)$

Binary classification: $h_{t+1} = \arg\min_{h \in \mathscr{H}} \sum_i p_i \cdot \mathbf{1}\{h(x_i) \neq y_i\}$

Recap on AdaBoost

Adaboost follows this framework with $\ell(\hat{y}, y) = \exp(-\hat{y} \cdot y)$

1. Create a new weighted dataset:

For each x_i, compute $p_i \propto \exp(-\hat{y}_i \cdot y_i)$

Binary classification: $h_{t+1} = \arg\min_{h \in \mathscr{H}} \sum_i p_i \cdot \mathbf{1}\{h(x_i) \neq y_i\}$

2. Add new learner to the ensemble:

Recap on AdaBoost

Adaboost follows this framework with $\ell(\hat{y}, y) = \exp(-\hat{y} \cdot y)$

1. Create a new weighted dataset:

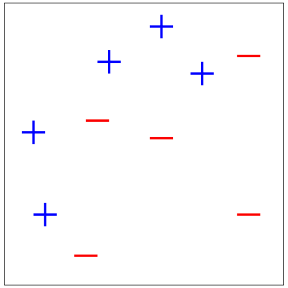
For each x_i, compute $p_i \propto \exp(-\hat{y}_i \cdot y_i)$

Binary classification: $h_{t+1} = \arg\min_{h \in \mathscr{H}} \sum_i p_i \cdot \mathbf{1}\{h(x_i) \neq y_i\}$

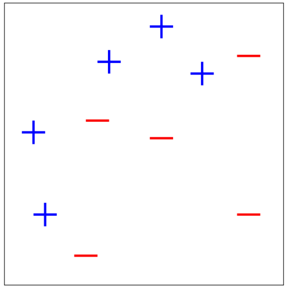
2. Add new learner to the ensemble:

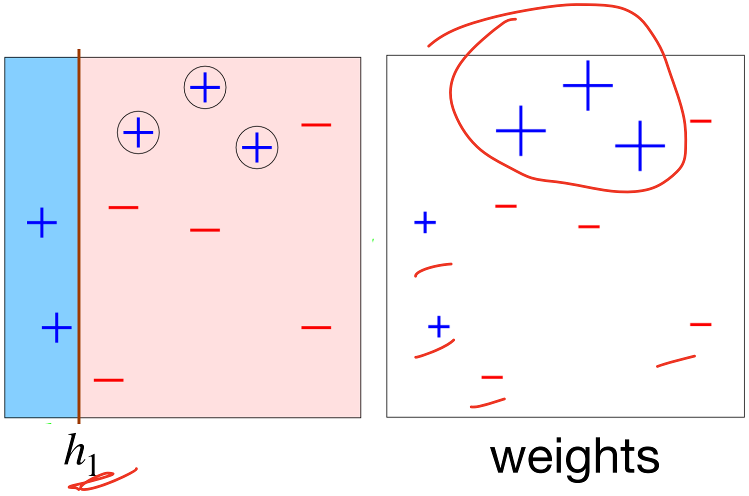
$$H_{t+1} = H_t + \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon} \cdot h_{t+1}$$

Weaker learner: axis-aligned linear decision boundary



Weaker learner: axis-aligned linear decision boundary

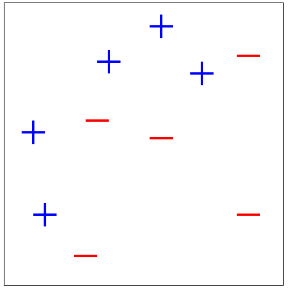


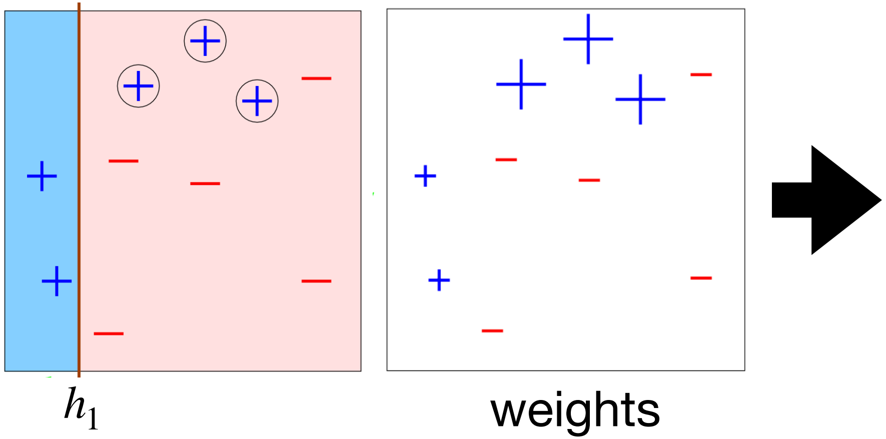


$H_1 = h_1$

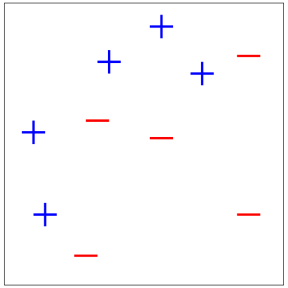
$p_i \propto \exp(-\hat{y}_i y_i)$

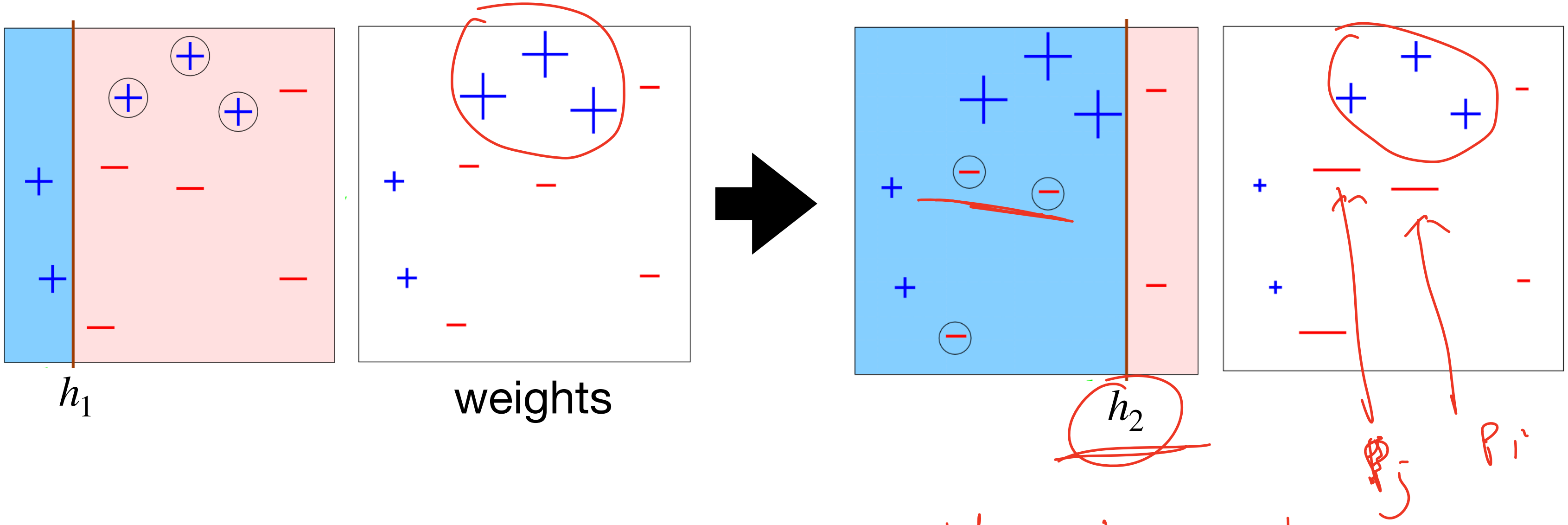
Weaker learner: axis-aligned linear decision boundary





Weaker learner: axis-aligned linear decision boundary



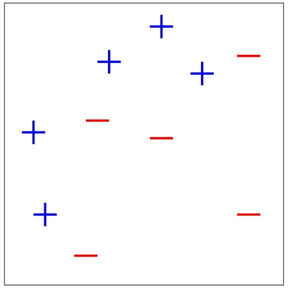


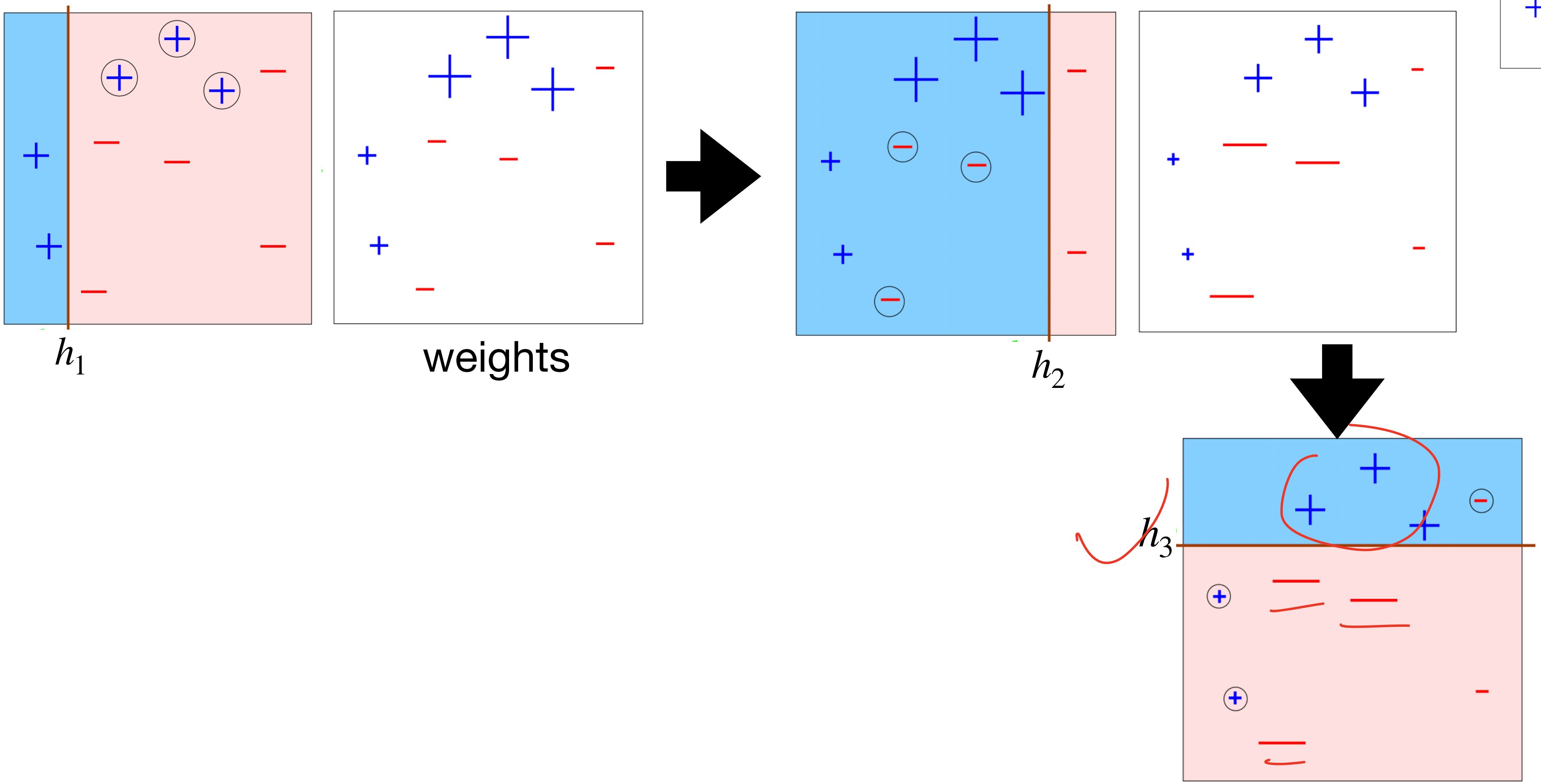
$$H_2 = h_1 + 2h_2$$

$$\hat{y}_i = H_2(x_i)$$

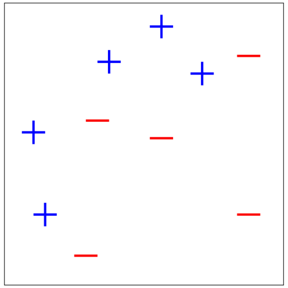
$$p_i \propto \exp(-\hat{y}_i y_i)$$

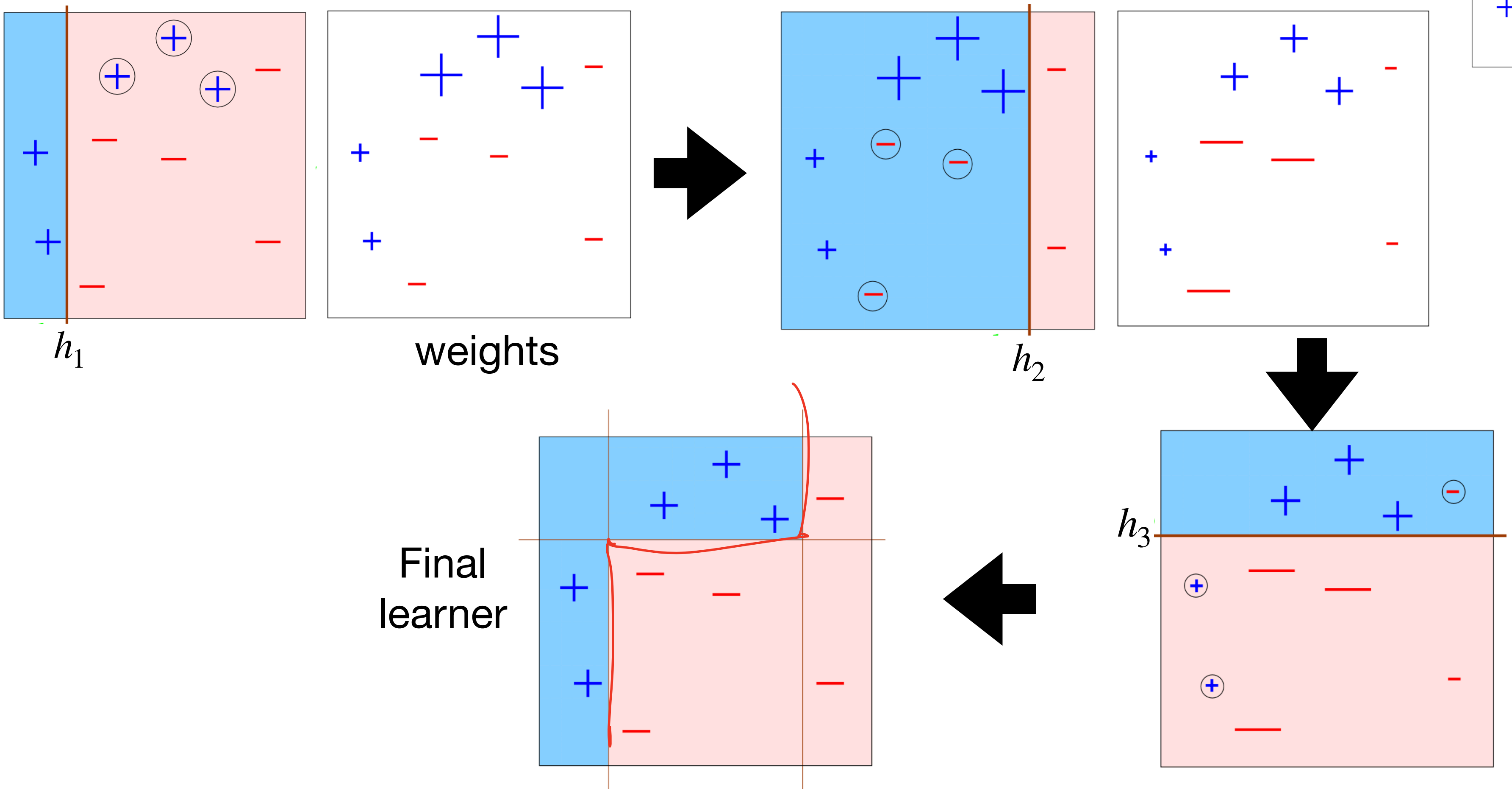
Weaker learner: axis-aligned linear decision boundary





Weaker learner: axis-aligned linear decision boundary





Outline of Today

1. Analysis of Boosting

2. Multilayer feedforward Neural Network

The definition of Weak learning

Each weaker learning optimizes its own data:

$$\widetilde{\mathcal{D}} = \{p_i, x_i, y_i\}, \text{ where } \sum_i p_i = 1,\ p_i \geq 0, \forall i$$

$$h_{t+1} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} p_i \cdot \mathbf{1}(h(x_i) \neq y_i)$$

The definition of Weak learning

Each weaker learning optimizes its own data:

$$\widetilde{\mathscr{D}} = \{p_i, x_i, y_i\}, \text{ where } \sum_i p_i = 1,\, p_i \geq 0, \forall i$$

$$h_{t+1} = \arg\min_{h \in \mathscr{H}} \sum_{i=1}^{n} p_i \cdot \mathbf{1}(h(x_i) \neq y_i)$$

Assume that weaker learner's loss $\epsilon := \sum_{i=1}^{n} p_i \mathbf{1}\{h_{t+1}(x_i) \neq y_i\} \leq \frac{1}{2} - \gamma,\ \gamma > 0$

The definition of Weak learning

Each weaker learning optimizes its own data:

$$\widetilde{\mathscr{D}} = \{p_i, x_i, y_i\}, \text{ where } \sum_i p_i = 1, p_i \geq 0, \forall i$$

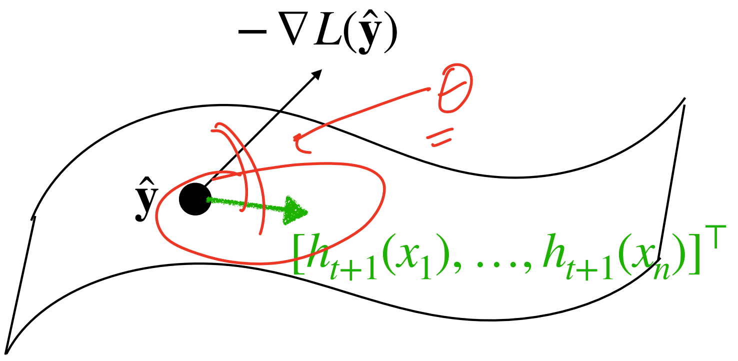
$$h_{t+1} = \arg\min_{h \in \mathscr{H}} \sum_{i=1}^{n} p_i \cdot \mathbf{1}(h(x_i) \neq y_i)$$

Assume that weaker learner's loss $\epsilon := \sum_{i=1}^{n} p_i \mathbf{1}\{h_{t+1}(x_i) \neq y_i\} \leq \frac{1}{2} - \gamma, \ \gamma > 0$

Q: assume $\mathscr{H}$ is symmetric, i.e., $h \in \mathscr{H}$ iff $-h \in \mathscr{H}$, why does the above always hold?

Weaker learnability implies approximating gradient well

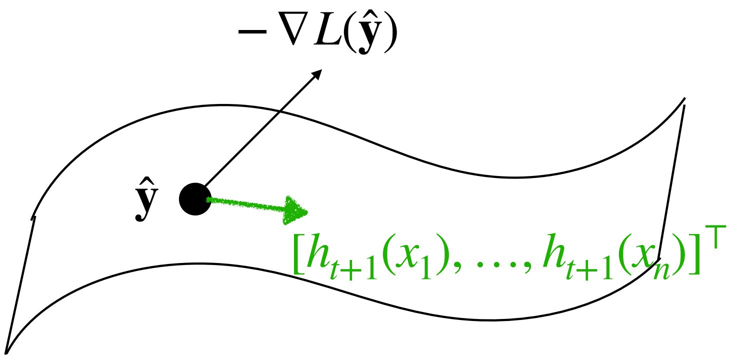
Assume that weaker learner's loss $\epsilon := \sum_{i=1}^{n} p_i \mathbf{1}\{h_{t+1}(x_i) \neq y_i\} \leq \frac{1}{2} - \gamma, \;\; \gamma > 0$



Weaker learnability implies approximating gradient well

Assume that weaker learner's loss $\epsilon := \sum_{i=1}^{n} p_i \mathbf{1}\{h_{t+1}(x_i) \neq y_i\} \leq \frac{1}{2} - \gamma, \ \gamma > 0$

$$(-\nabla L(\hat{\mathbf{y}}))^\top \begin{bmatrix} h_{t+1}(x_1) \\ \cdots \\ h_{t+1}(x_n) \end{bmatrix}$$

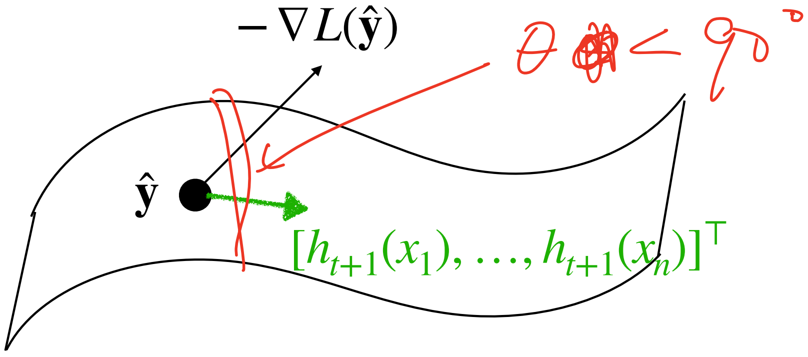


Weaker learnability implies approximating gradient well

Assume that weaker learner's loss $\epsilon := \sum_{i=1}^{n} p_i \mathbf{1}\{h_{t+1}(x_i) \neq y_i\} \leq \frac{1}{2} - \gamma, \;\; \gamma > 0$

$$(-\nabla L(\hat{\mathbf{y}}))^\top \begin{bmatrix} h_{t+1}(x_1) \\ \cdots \\ h_{t+1}(x_n) \end{bmatrix}$$

$$\geq (\sum_{j=1}^{n} w_j \;\;)2\gamma > 0$$

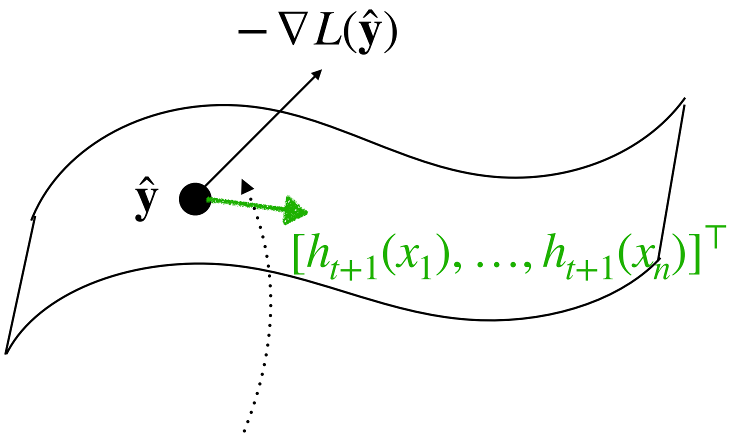


Weaker learnability implies approximating gradient well

Assume that weaker learner's loss $\epsilon := \sum_{i=1}^{n} p_i \mathbf{1}\{h_{t+1}(x_i) \neq y_i\} \leq \frac{1}{2} - \gamma, \;\; \gamma > 0$

$$(-\nabla L(\hat{\mathbf{y}}))^\top \begin{bmatrix} h_{t+1}(x_1) \\ \cdots \\ h_{t+1}(x_n) \end{bmatrix}$$

$$\geq (\sum_{j=1}^{n} w_j \;)2\gamma > 0$$



Within 90 degree, so
improve the objective!

$\ell(\hat{y}, y) = \exp(-\hat{y} \cdot y)$

Formal Convergence of AdaBoost

$(0.95)^{\frac{100}{2}}$

Then after T iterations, for the original exp loss, we have

$$\frac{1}{n}\sum_{i=1}^{n}\exp(-H_T(x_i)\cdot y_i) \le n(\underbrace{1-4\gamma^2}_{<1})^{T/2}$$

$\gamma \leftrightarrow \frac{1}{2} - \gamma$

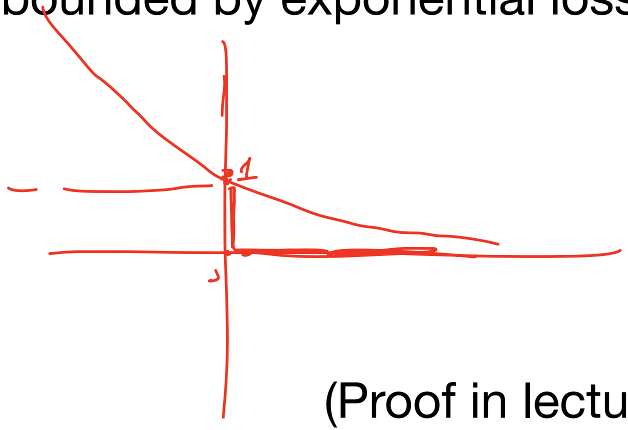
(Proof in lecture note, optional)

Formal Convergence of AdaBoost

Then after T iterations, for the original exp loss, we have

$$\frac{1}{n}\sum_{i=1}^{n}\exp(-H_T(x_i)\cdot y_i) \leq n(1-4\gamma^2)^{T/2}$$

Note zero-one loss is upper bounded by exponential loss



(Proof in lecture note, optional)

Formal Convergence of AdaBoost

Then after T iterations, for the original exp loss, we have

$$\frac{1}{n}\sum_{i=1}^{n}\exp(-H_T(x_i)\cdot y_i) \leq n(1-4\gamma^2)^{T/2}$$

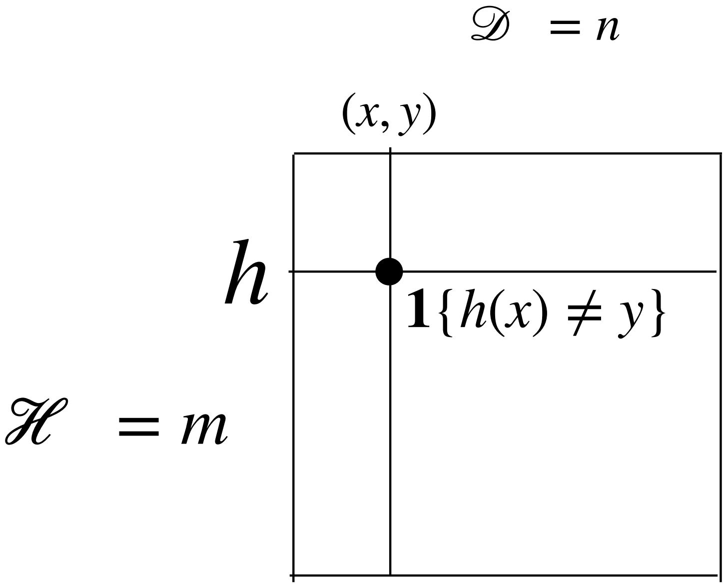
Note zero-one loss is upper bounded by exponential loss

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\{\underline{\text{sign}(H_T(x_i))} \neq y_i\} \leq \frac{1}{n}\sum_{i=1}^{n}\exp(-H_T(x_i)\cdot y_i) \leq n(1-4\gamma^2)^{T/2}$$

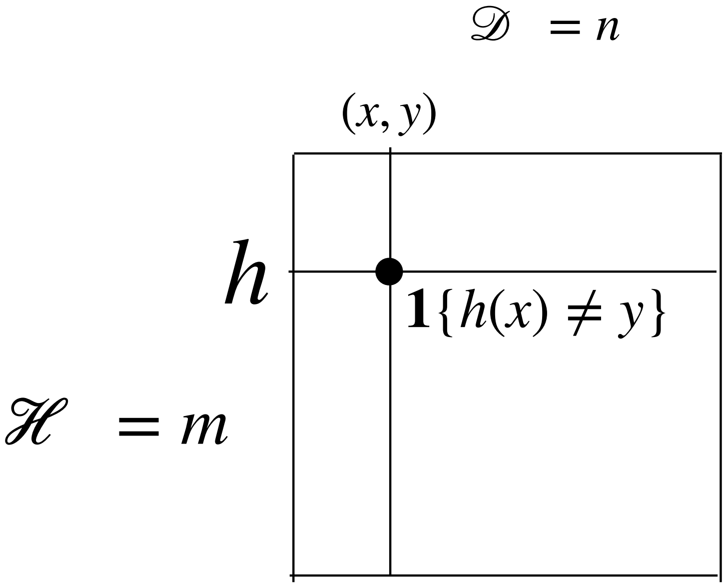
Avg # of mistakes

(Proof in lecture note, optional)

Thinking about Boosting via two player zero sum game



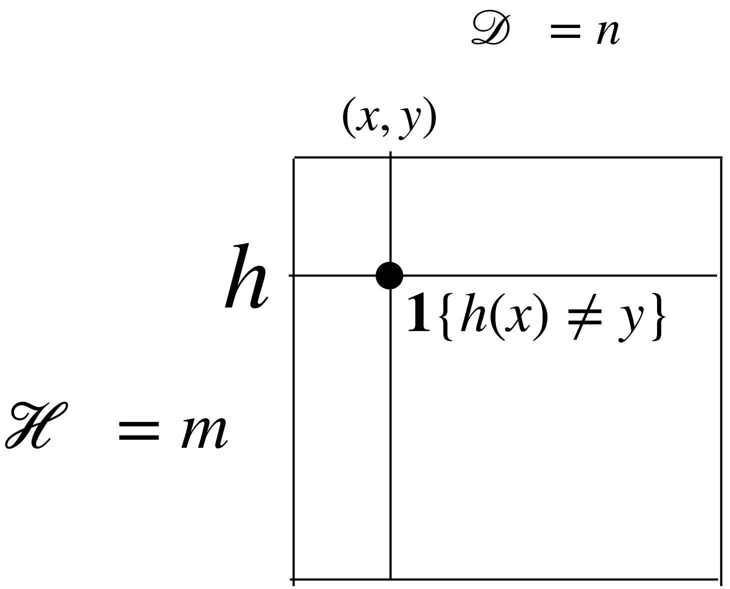
Thinking about Boosting via two player zero sum game



Row player plays hypothesis $h \in \mathscr{H}$

Column player plays example (x, y)

Thinking about Boosting via two player zero sum game

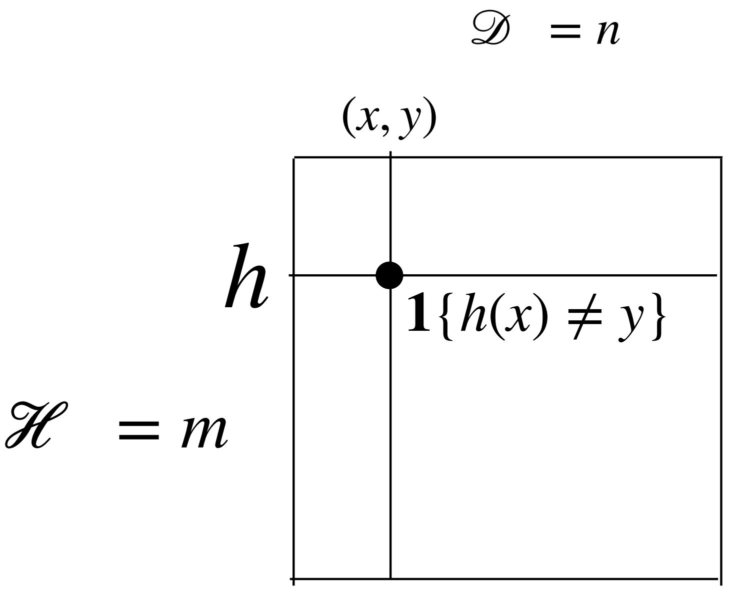


Row player plays hypothesis $h \in \mathscr{H}$

Column player plays example (x, y)

Row player gets loss $\mathbf{1}\{h(x) \neq y\}$

Thinking about Boosting via two player zero sum game



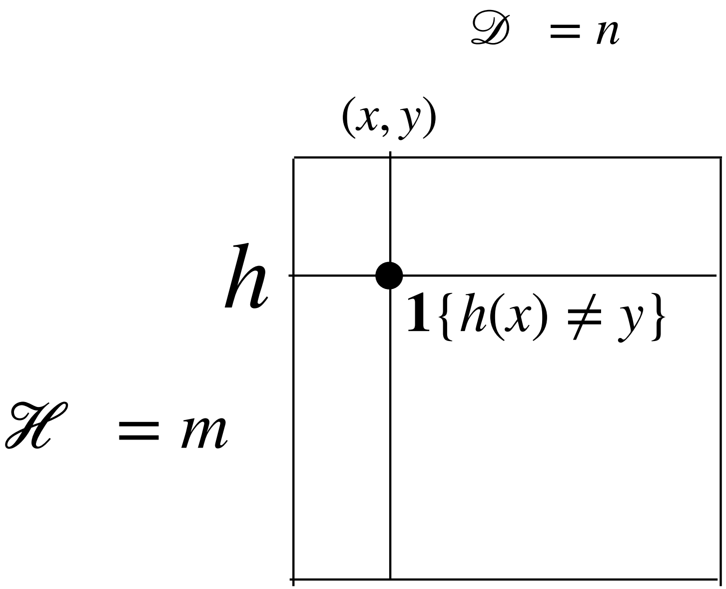
Row player plays hypothesis $h \in \mathscr{H}$

Column player plays example (x, y)

Row player gets loss $\mathbf{1}\{h(x) \neq y\}$

Column player gets loss $-\mathbf{1}\{h(x) \neq y\}$

Thinking about Boosting via two player zero sum game



Row player plays hypothesis $h \in \mathscr{H}$

Column player plays example (x, y)

Row player gets loss $\mathbf{1}\{h(x) \neq y\}$

Column player gets loss $-\mathbf{1}\{h(x) \neq y\}$

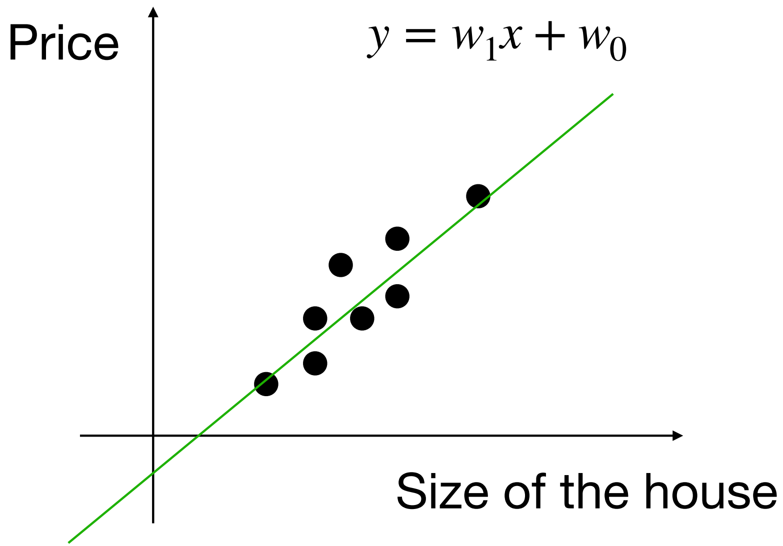
Boosting can be understood as running some specific algorithm to find the Nash equilibrium of the game

Outline of Today

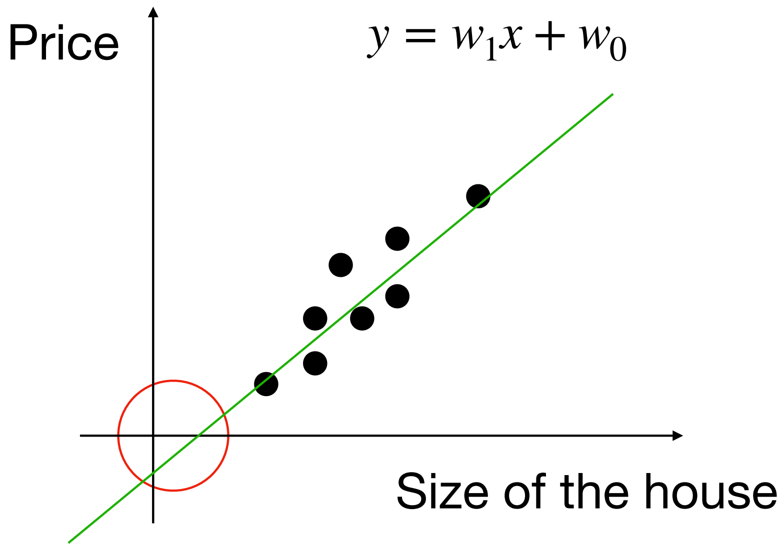
1. Analysis of Boosting

2. Multilayer feedforward Neural Network

Linear Regression Revisit

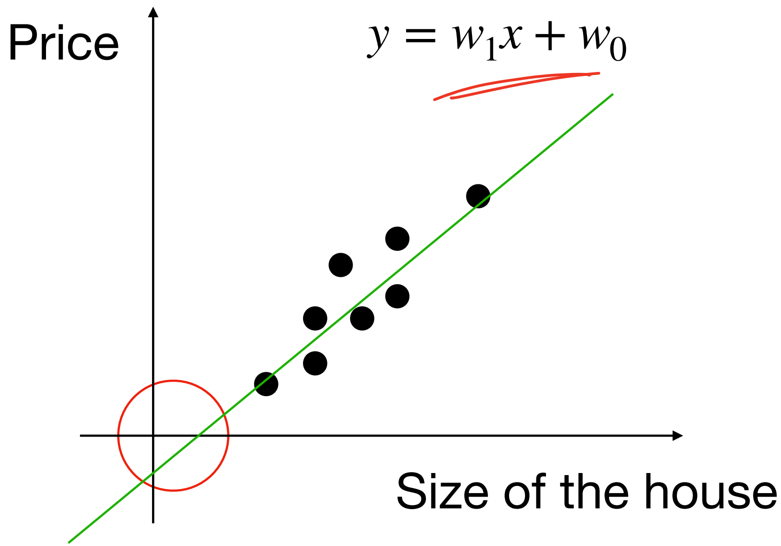


Linear Regression Revisit



Negative part does not make too much sense

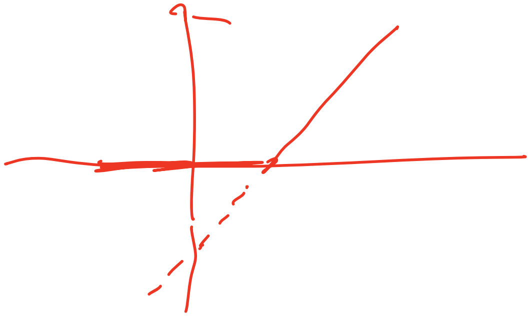
Linear Regression Revisit



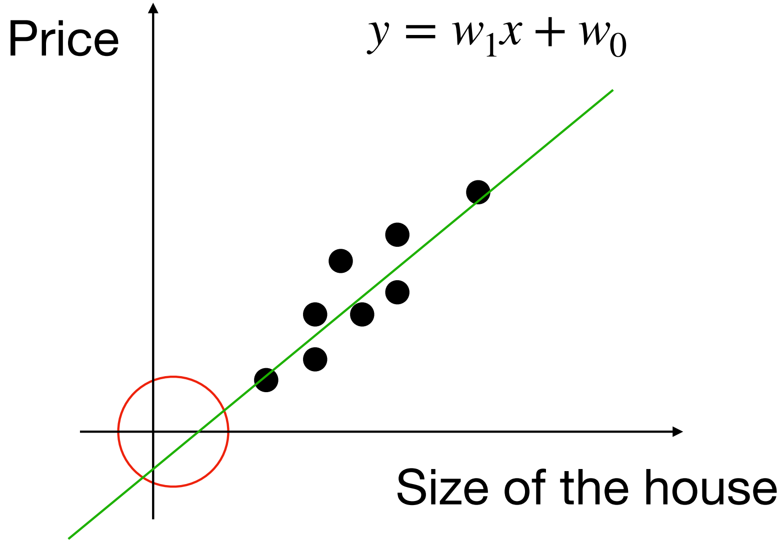
Negative part does not make too much sense

We can fix this with a simple nonlinear function

$$y = \max\{w_1 x + w_0,\ 0\}$$



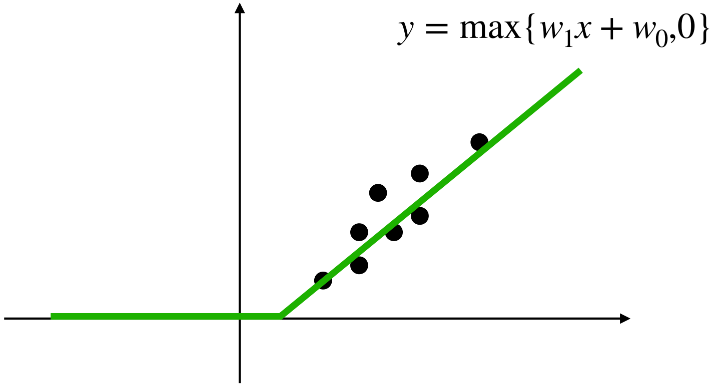
Linear Regression Revisit



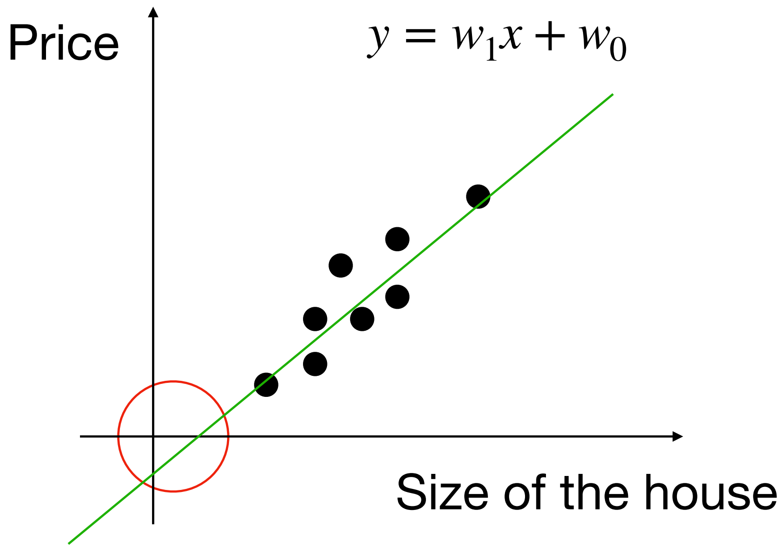
Negative part does not make too much sense

We can fix this with a simple nonlinear function

$$y = \max\{w_1x + w_0,\ 0\}$$



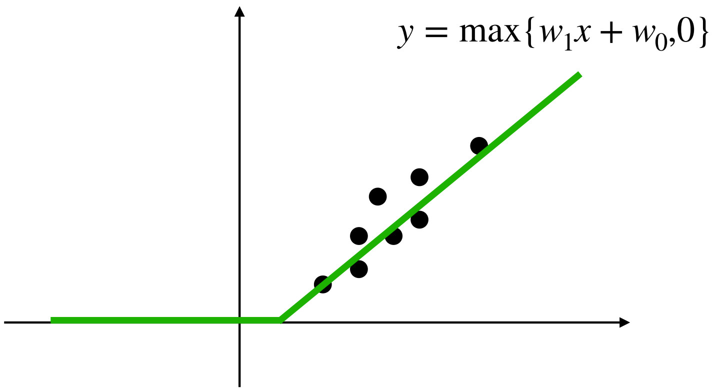
Linear Regression Revisit



Negative part does not make too much sense

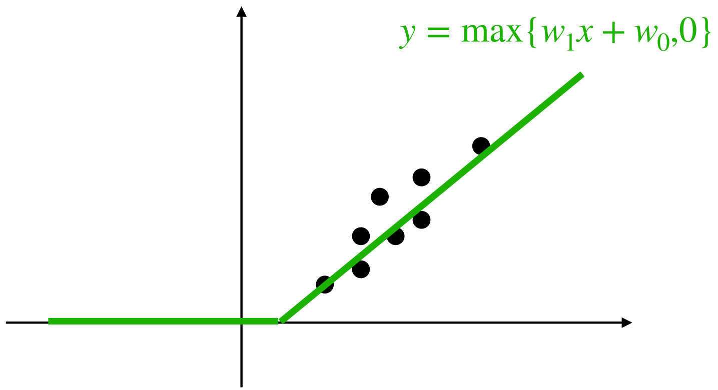
We can fix this with a simple nonlinear function

$$y = \max\{w_1 x + w_0, 0\}$$



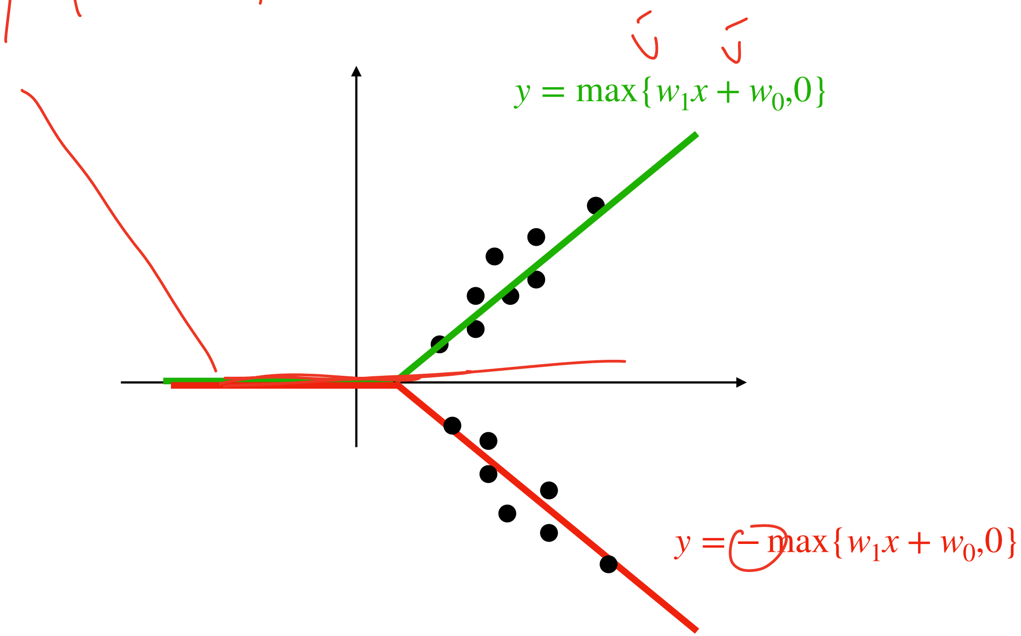
rectified linear unit (ReLU)

A single neuron network

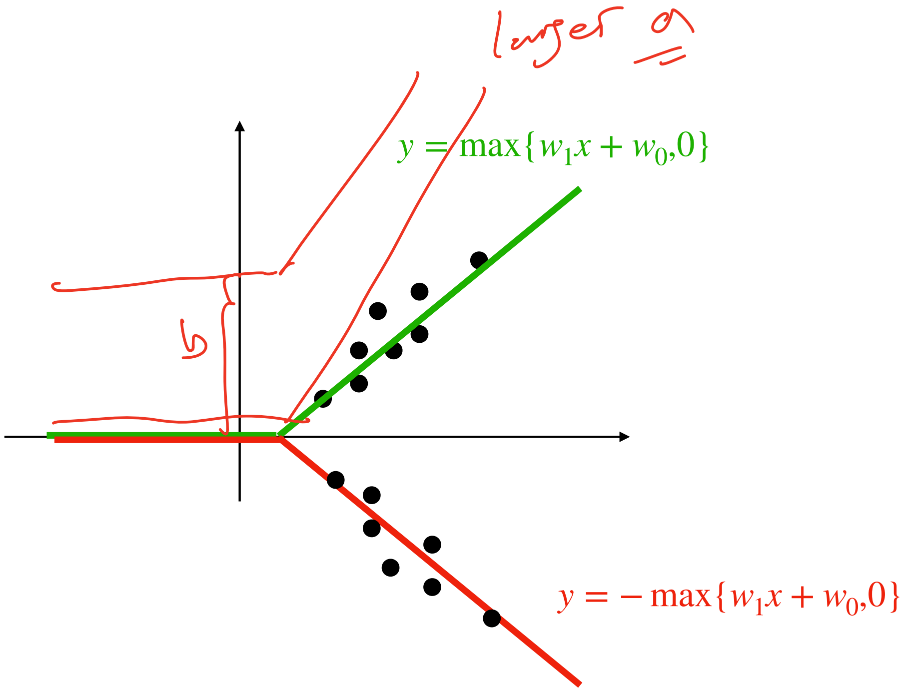


A single neuron network

$y = \max\{-w_1 x - w_0, 0\}$



A single neuron network



$$y = a \max\{w_1 x + w_0, 0\} + b$$

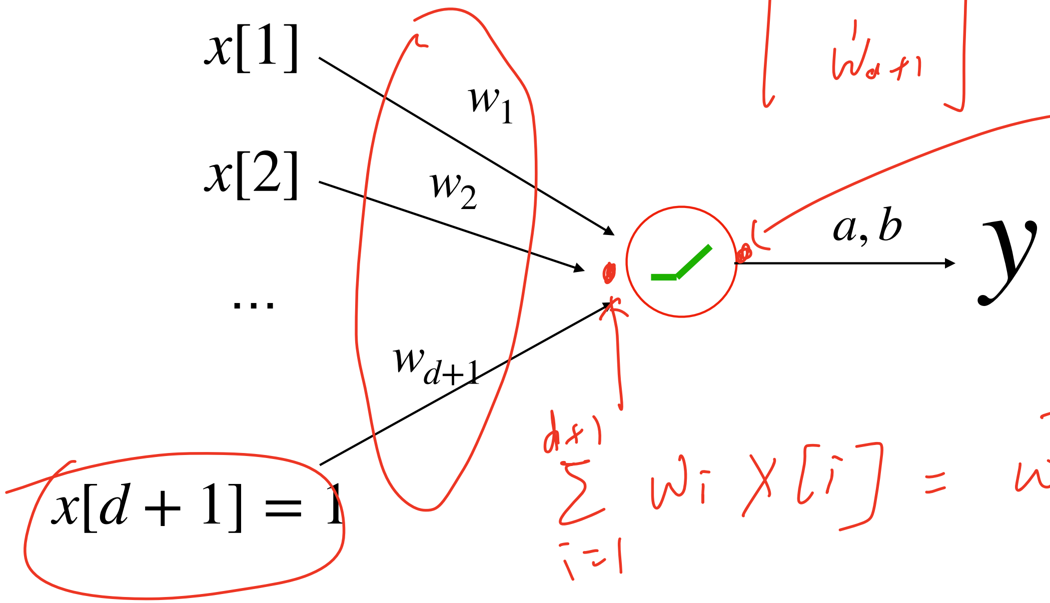
four parameters

a, w_1, w_0, b

A single neuron network

$\cancel{w_i \in \mathbb{R}}$

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{d+1} \end{bmatrix}$$

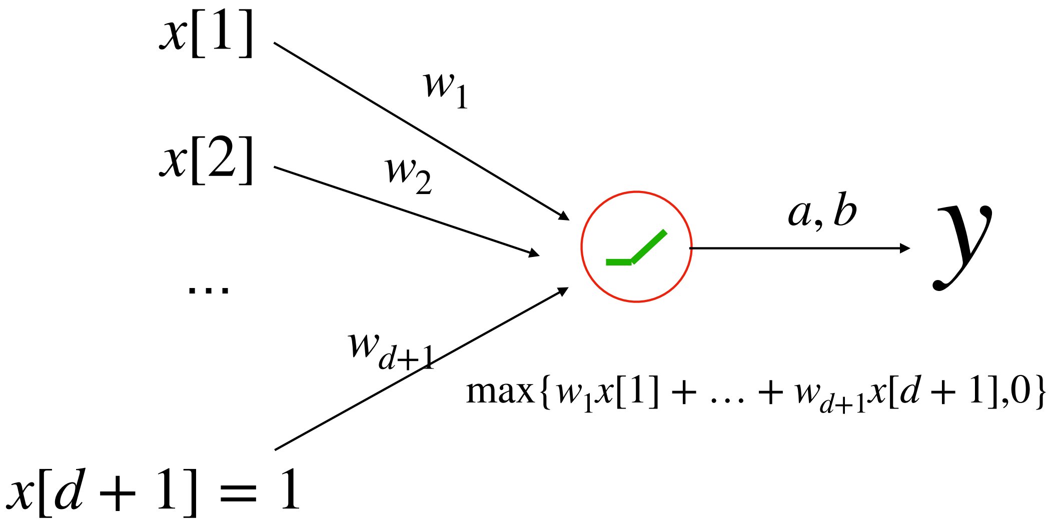


$$\max\{w^T x, 0\}$$

$$y = a \max\{w^T x, 0\} + b$$

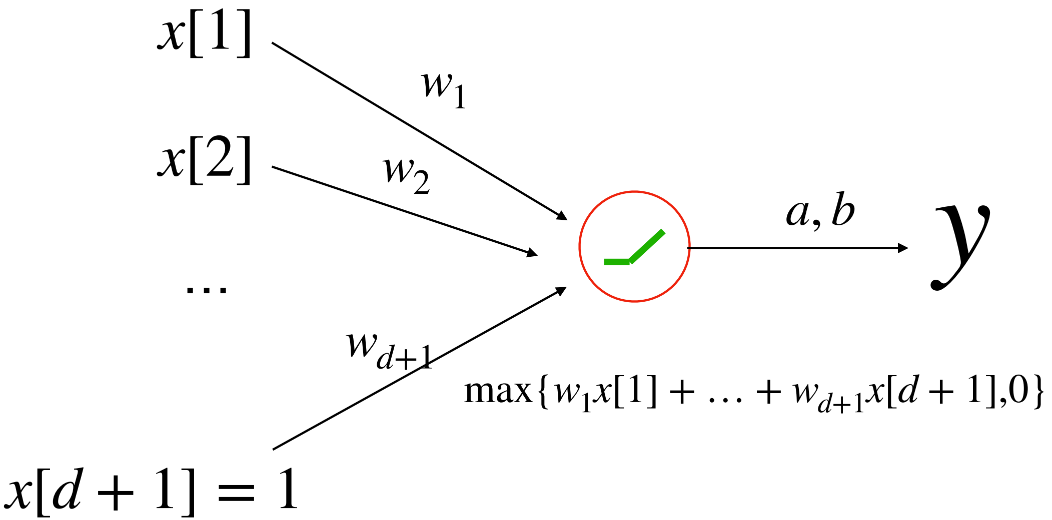
$$\sum_{i=1}^{d+1} w_i x[i] = w^T x$$

A single neuron network

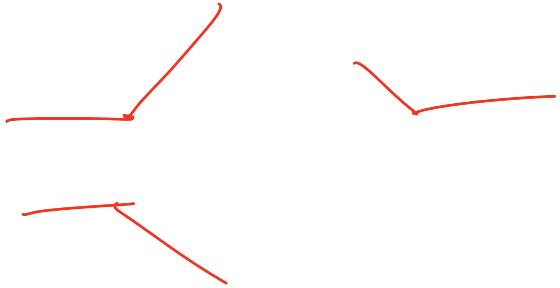


A single neuron network

$\max\{x, 0\} \Leftarrow \mathrm{Relu}(x)$

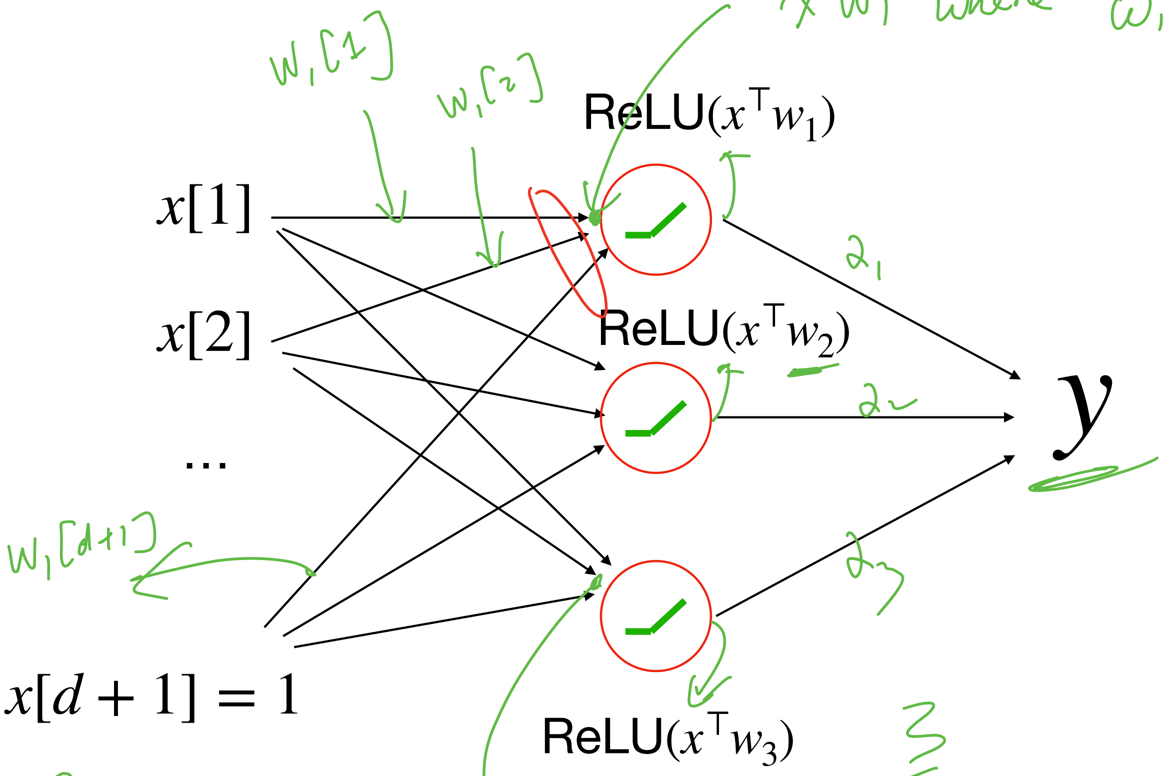


$$y = a\mathrm{ReLU}(w^\top x) + b$$



Let us stack multiple neurons together

$x^T w_1$ where $w_1 = \begin{bmatrix} w_1[1] \\ \vdots \\ w_1[d+1] \end{bmatrix}$



$$y = \sum_{i=1}^{3} a_i \mathrm{ReLU}(x^\top w_i) + b$$

$(d+1)\cdot 3 + 3$

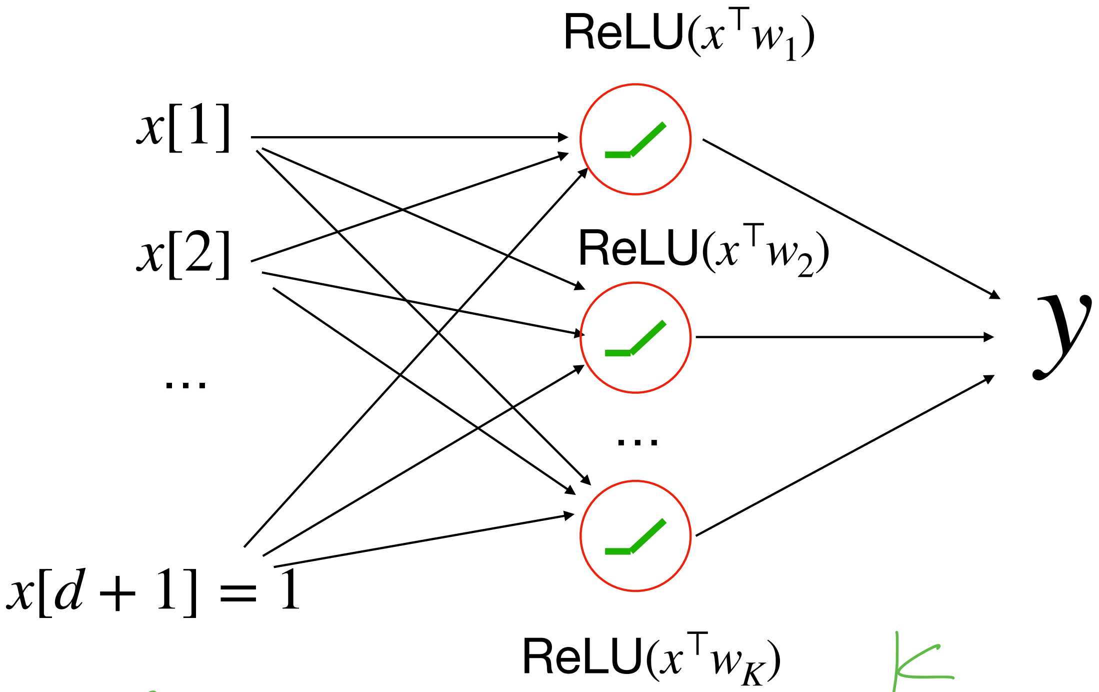
3

$(d+1)\cdot 3$

$x^T w_3$

Let us stack multiple neurons together

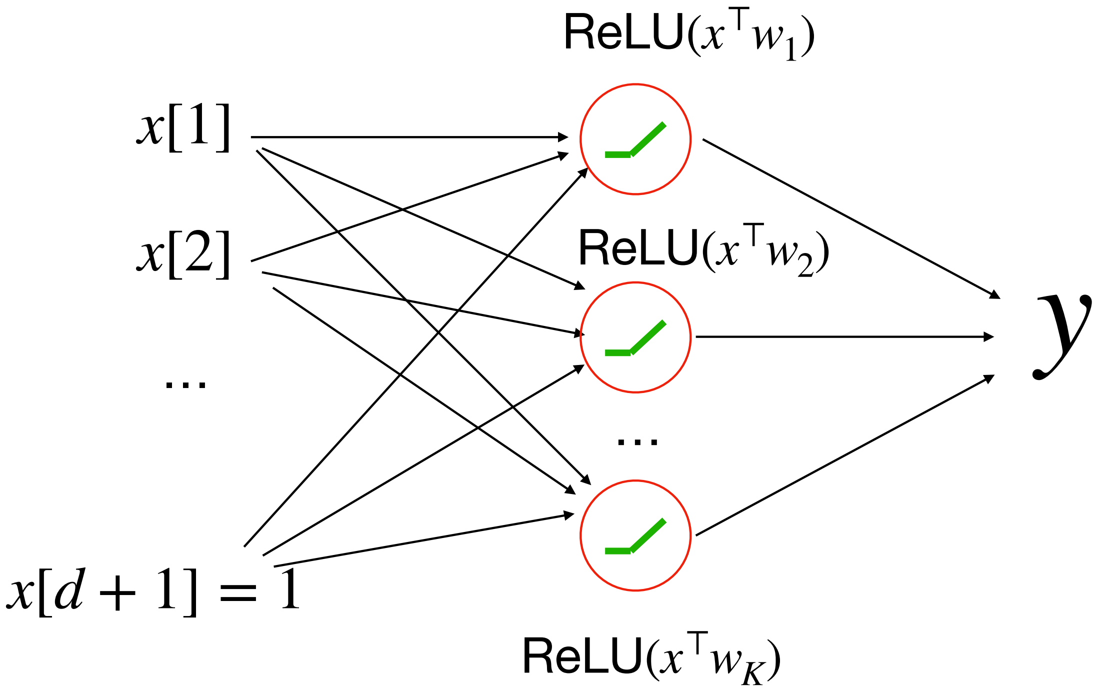
K # of neurons



K

$(d+1) \cdot K$

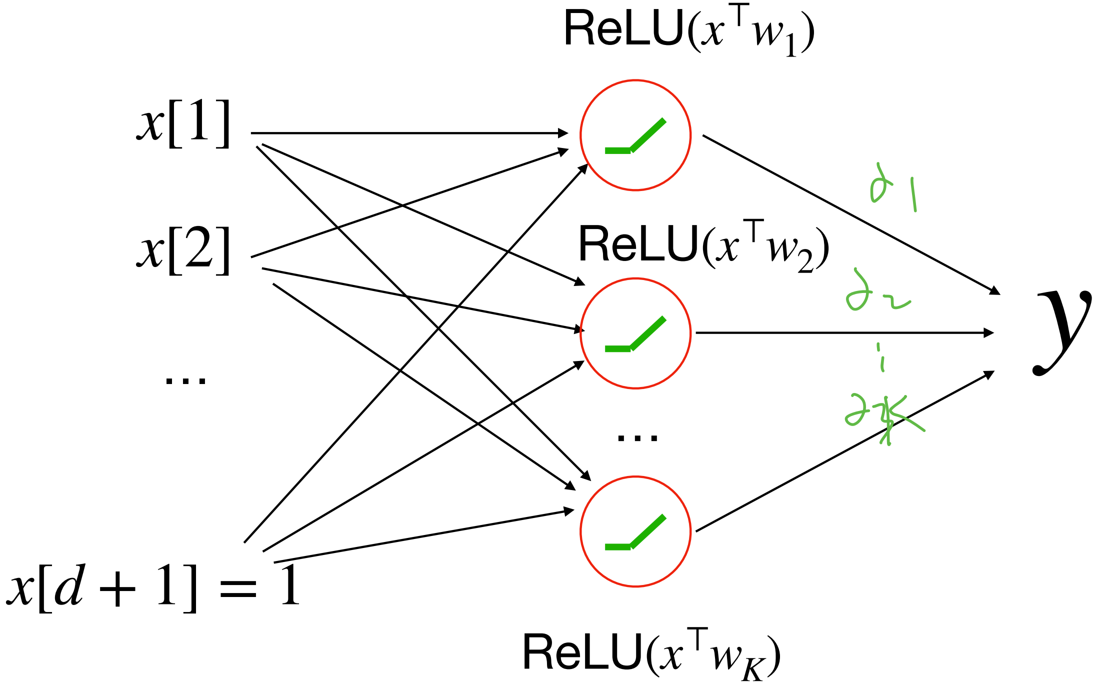
Let us stack multiple neurons together



Vectorized form:

$$\text{Define } W = \begin{bmatrix} (w_1)^\top \\ \cdots \\ (w_K)^\top \end{bmatrix} \in \mathbb{R}^{K \times (d+1)}$$

Let us stack multiple neurons together

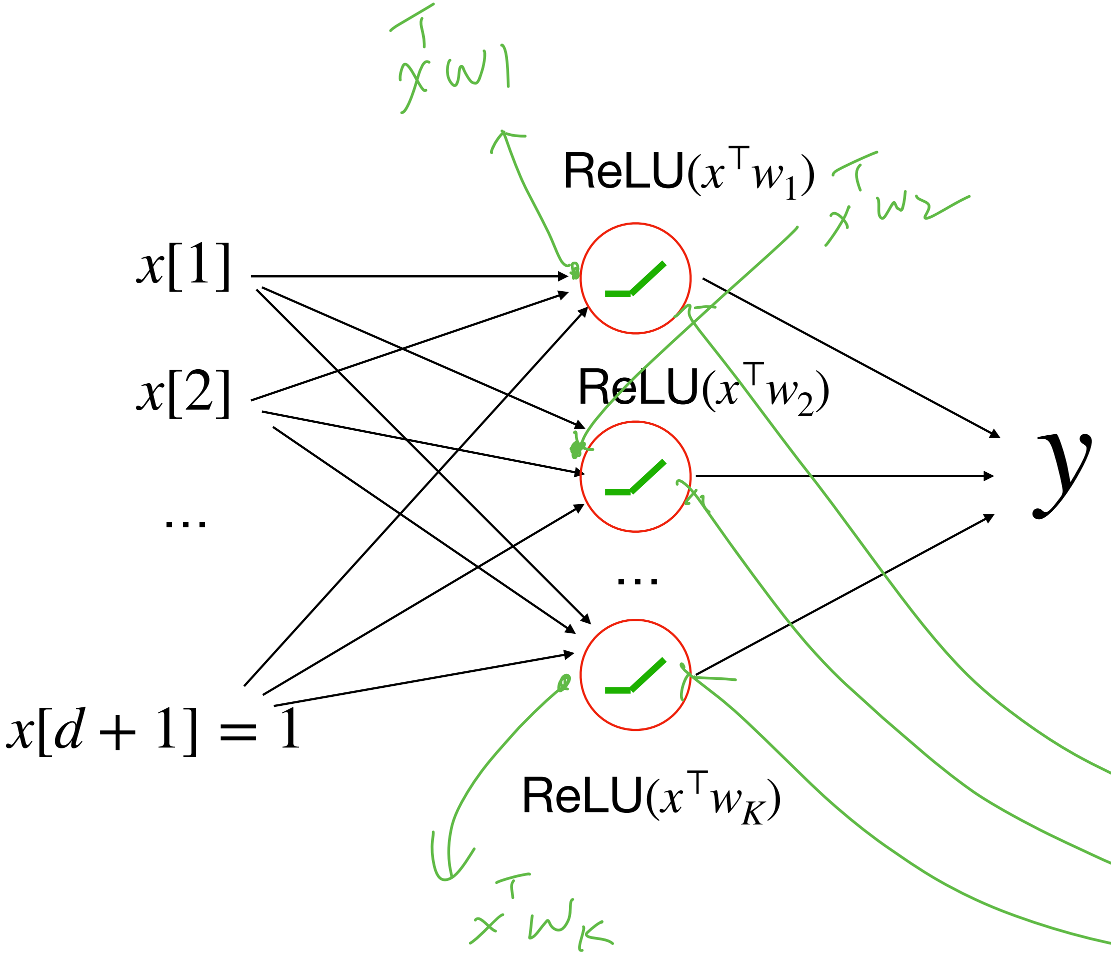


Vectorized form:

$$\text{Define } W = \begin{bmatrix} (w_1)^\top \\ \cdots \\ (w_K)^\top \end{bmatrix} \in \mathbb{R}^{K \times (d+1)}$$

$$\alpha = [a_1, \ldots, a_K]^\top$$

Let us stack multiple neurons together



Vectorized form:

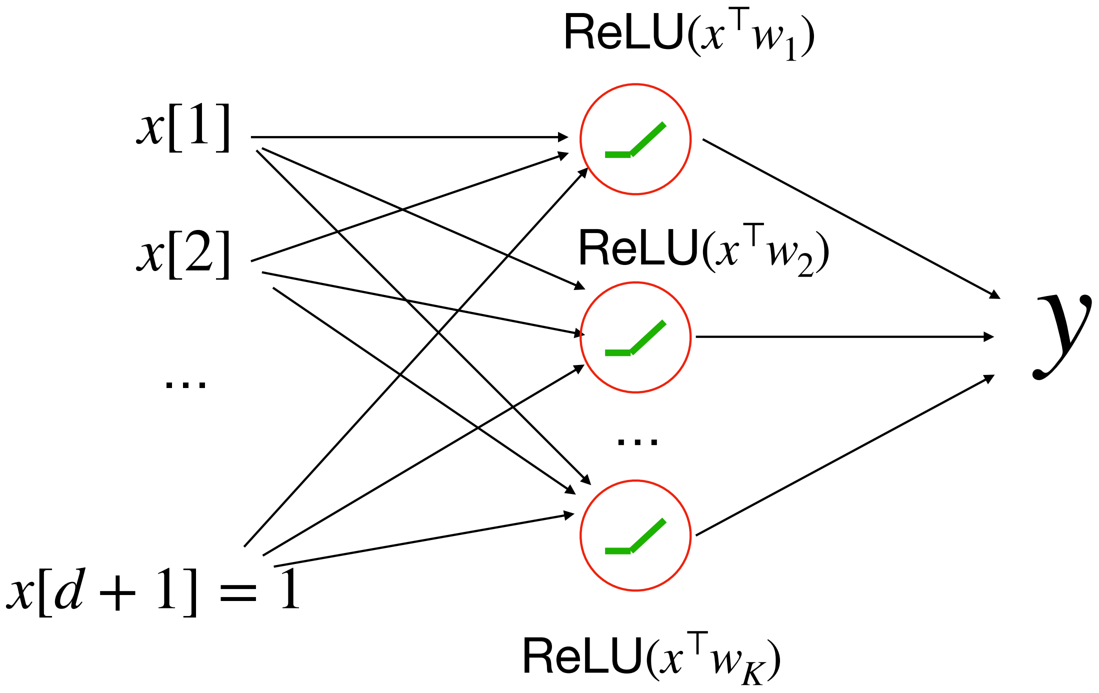
$$\text{ReLu}(z) = \begin{bmatrix} \text{ReLu}(z_1) \\ \text{ReLu}(z_2) \\ \vdots \\ \text{ReLu}(z_d) \end{bmatrix}$$

Define $W = \begin{bmatrix} (w_1)^\top \\ \cdots \\ (w_K)^\top \end{bmatrix} \in \mathbb{R}^{K \times (d+1)}$

$$\alpha = [a_1, \ldots, a_K]^\top$$

$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

Let us stack multiple neurons together



Vectorized form:

Define $W = \begin{bmatrix} (w_1)^\top \\ \cdots \\ (w_K)^\top \end{bmatrix} \in \mathbb{R}^{K \times (d+1)}$

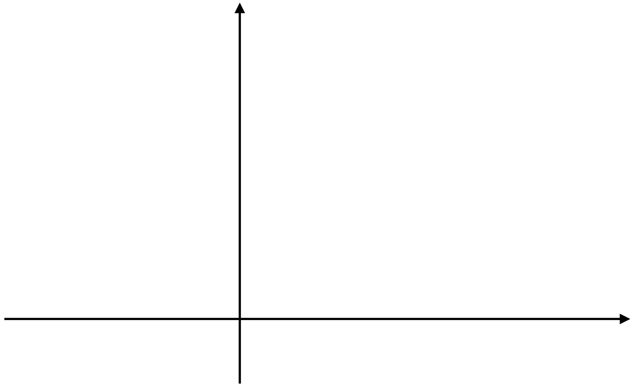
$$\alpha = [a_1, \ldots, a_K]^\top$$

$$y = \alpha^\top (\mathrm{ReLU}(Wx)) + b$$

Learnable feature $\phi(x)$

What does a neural network approximate

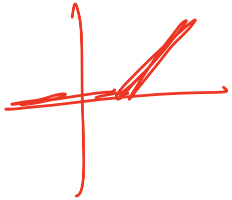
$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

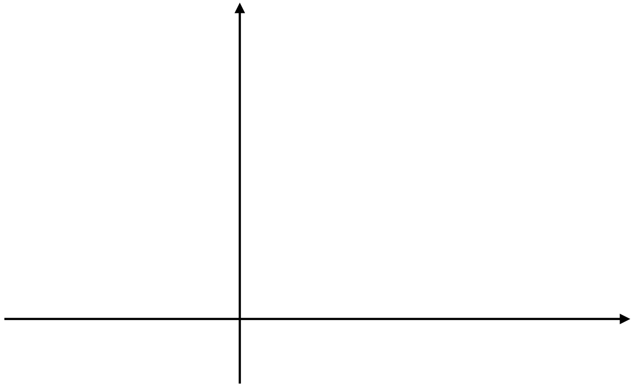


What does a neural network approximate

$$y = \alpha^\top \left(\mathrm{ReLU}(Wx)\right) + b \;=\; \sum_{i=1}^{K} \underbrace{\alpha_i \,\mathrm{Relu}(w_i^\top x)} + b$$

It's a pieces wise linear functions



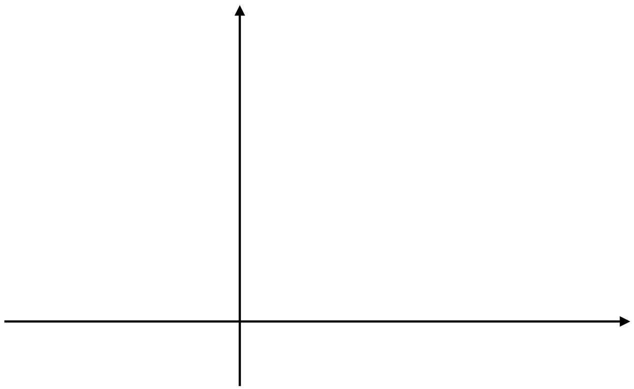


What does a neural network approximate

$$y = \alpha^\top \left(\mathrm{ReLU}(Wx)\right) + b$$

It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):



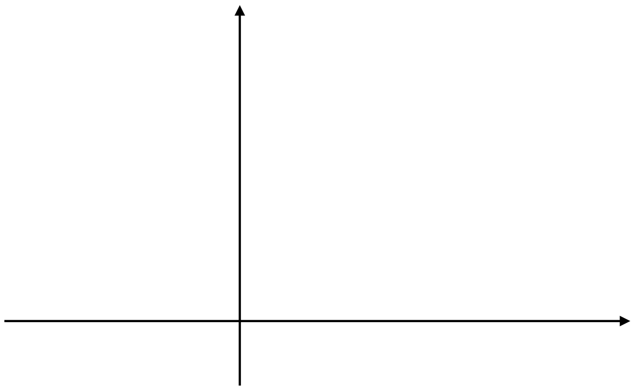
What does a neural network approximate

$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):

$K = 1 : y = a_1 \max\{w_1 x + c_1, 0\}$



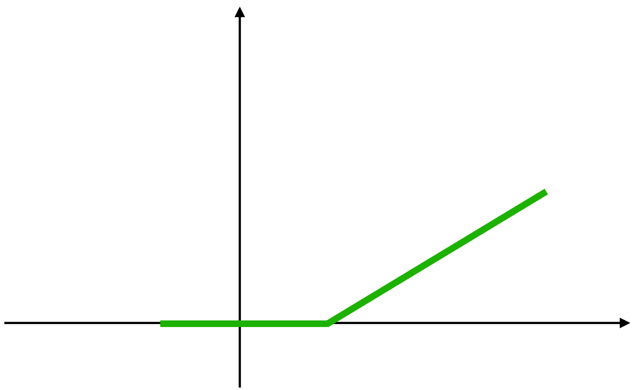
What does a neural network approximate

$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):

$K = 1 : y = a_1 \max\{w_1 x + c_1, 0\}$



What does a neural network approximate

$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

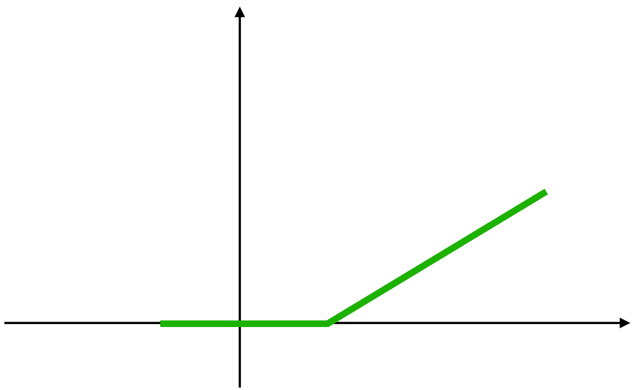
It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):

$$K = 1 : y = a_1 \max\{w_1 x + c_1, 0\}$$

$$K = 2 : y = a_1 \max\{w_1 x + c_1, 0\}$$

$$+ a_2 \max\{w_2 x + c_2, 0\}$$



What does a neural network approximate

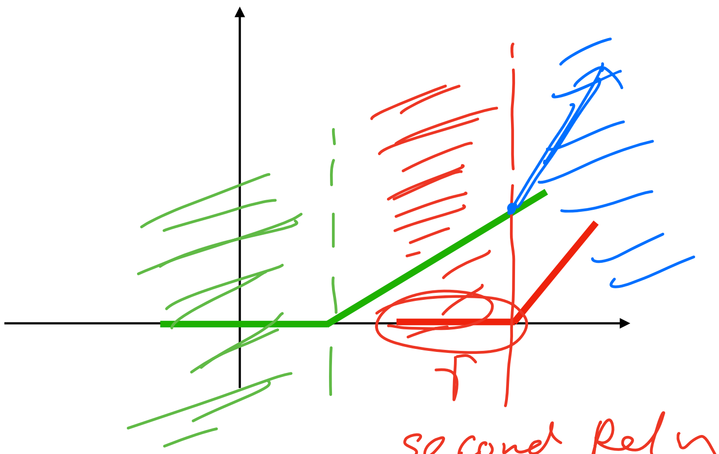
$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):

$$K = 1 : y = a_1 \max\{w_1 x + c_1, 0\}$$

$$K = 2 : y = a_1 \max\{w_1 x + c_1, 0\}$$
$$+ a_2 \max\{w_2 x + c_2, 0\}$$



second Relu

What does a neural network approximate

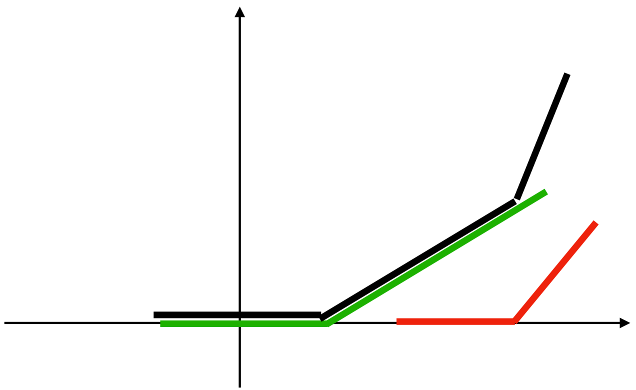
$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):

$$K = 1 : y = a_1 \max\{w_1 x + c_1, 0\}$$

$$K = 2 : y = a_1 \max\{w_1 x + c_1, 0\}$$
$$+ a_2 \max\{w_2 x + c_2, 0\}$$



What does a neural network approximate

$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

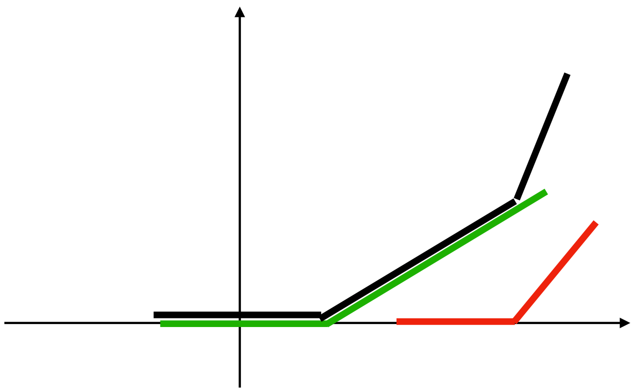
It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):

$$K = 1 : y = a_1 \max\{w_1 x + c_1, 0\}$$

$$K = 2 : y = a_1 \max\{w_1 x + c_1, 0\}$$
$$+ a_2 \max\{w_2 x + c_2, 0\}$$

$$K = 3 : y = a_1 \max\{w_1 x + c_1, 0\}$$
$$+ a_2 \max\{w_2 x + c_2, 0\}$$
$$+ a_3 \max\{w_3 x + c_3, 0\}$$



What does a neural network approximate

$$y = \alpha^\top\left(\mathrm{ReLU}(Wx)\right) + b$$

It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):

$K = 1 : y = a_1 \max\{w_1 x + c_1, 0\}$

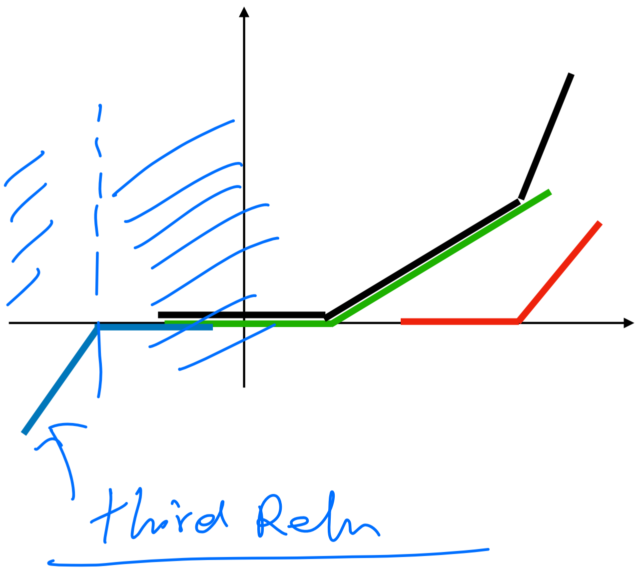
$K = 2 : y = a_1 \max\{w_1 x + c_1, 0\}$

$+a_2 \max\{w_2 x + c_2, 0\}$

$K = 3 : y = a_1 \max\{w_1 x + c_1, 0\}$

$+a_2 \max\{w_2 x + c_2, 0\}$

$+a_3 \max\{w_3 x + c_3, 0\}$



What does a neural network approximate

$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

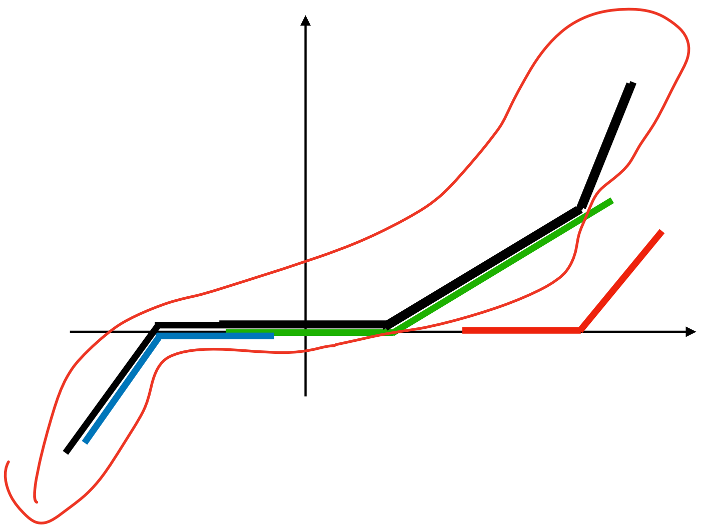
It's a pieces wise linear functions

Consider $d = 1$ case (and assume $b = 0$):

$$K = 1 : y = a_1 \max\{w_1 x + c_1, 0\}$$

$$K = 2 : y = a_1 \max\{w_1 x + c_1, 0\}$$
$$+ a_2 \max\{w_2 x + c_2, 0\}$$

$$K = 3 : y = a_1 \max\{w_1 x + c_1, 0\}$$
$$+ a_2 \max\{w_2 x + c_2, 0\}$$
$$+ a_3 \max\{w_3 x + c_3, 0\}$$



What does a neural network approximate

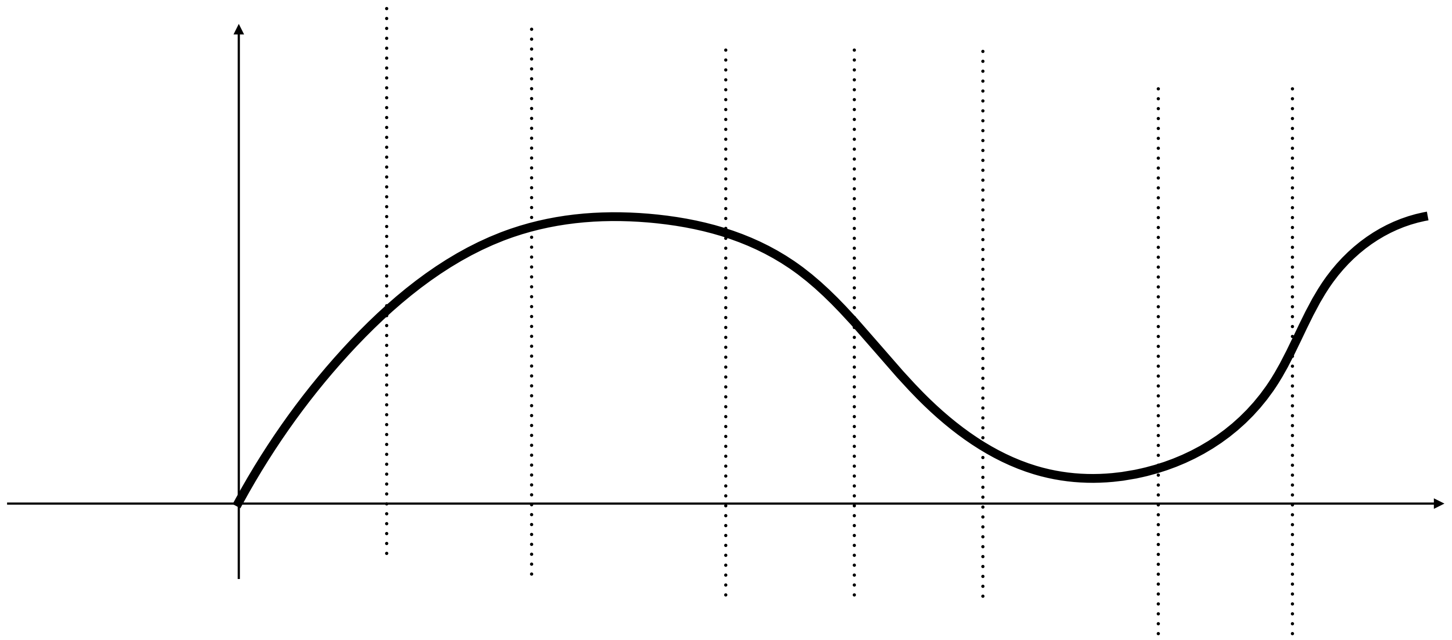
$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

Claim: a wide enough one layer NN can approximate any smooth functions

What does a neural network approximate

$$y = \alpha^\top \left(\mathrm{ReLU}(Wx)\right) + b$$

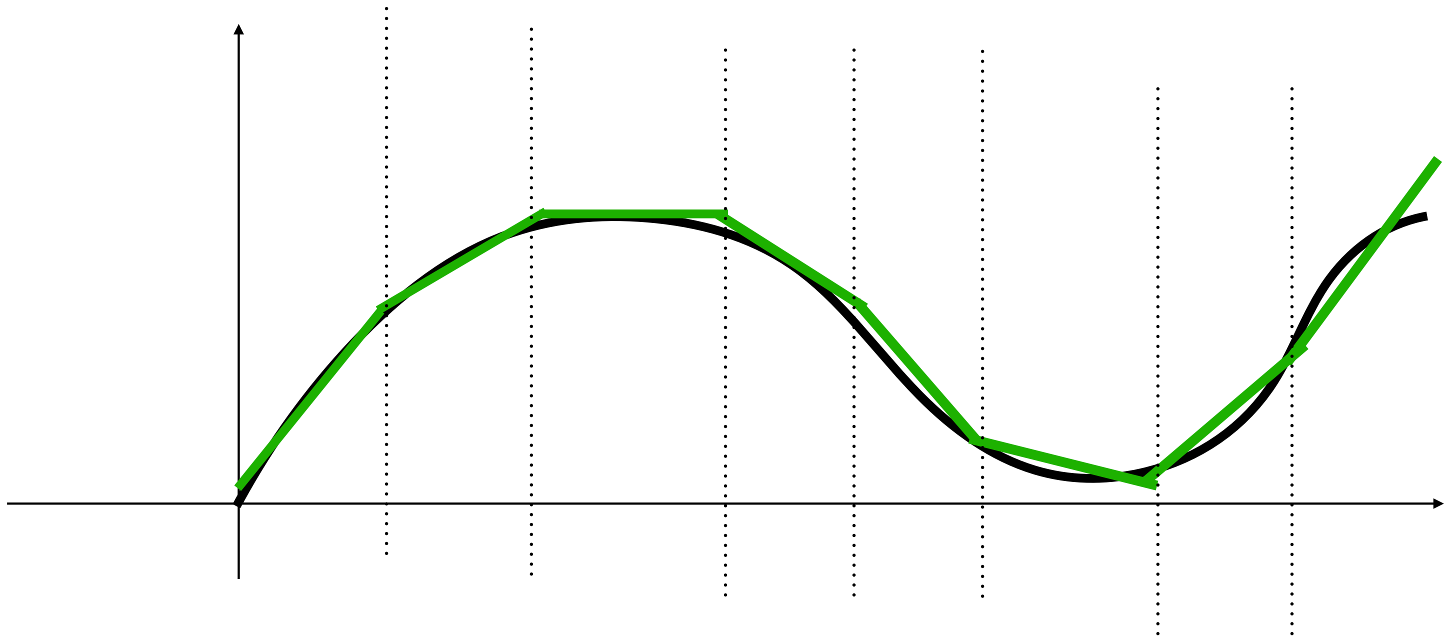
Claim: a wide enough one layer NN can approximate any smooth functions



What does a neural network approximate

$$y = \alpha^\top \left(\mathrm{ReLU}(Wx)\right) + b$$

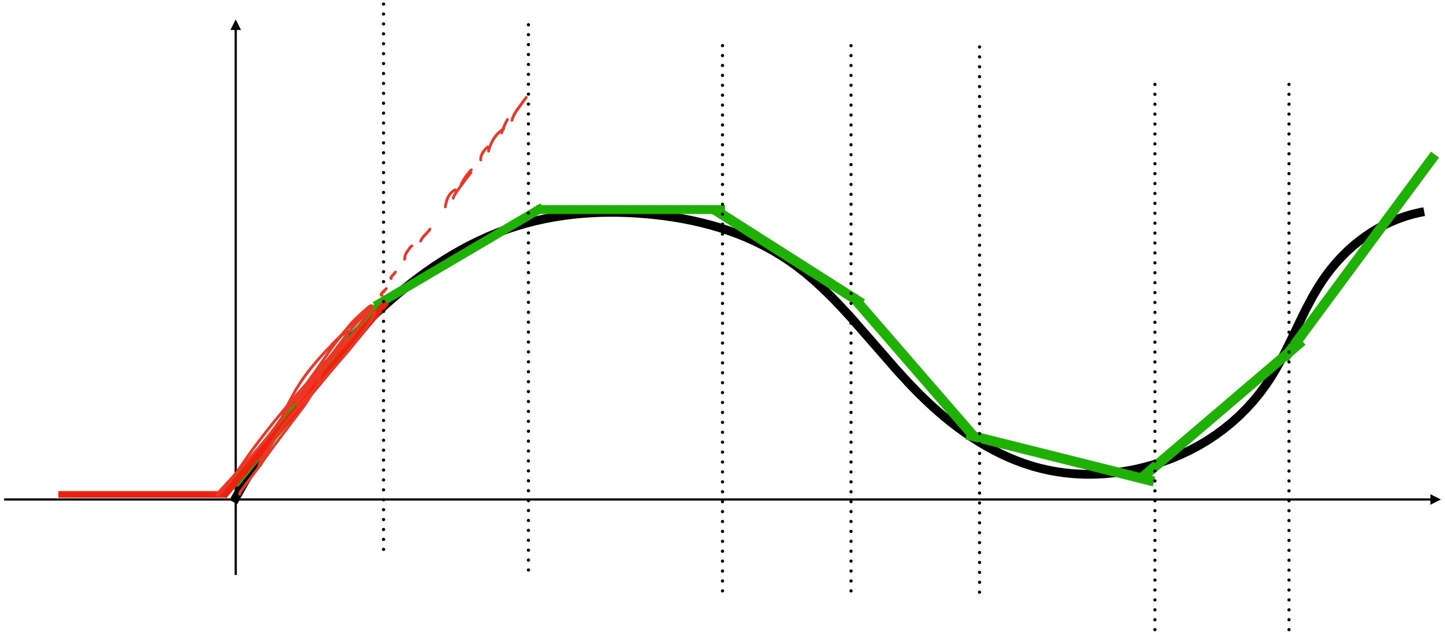
Claim: a wide enough one layer NN can approximate any smooth functions



What does a neural network approximate

$$y = \alpha^\top \left(\mathrm{ReLU}(Wx)\right) + b$$

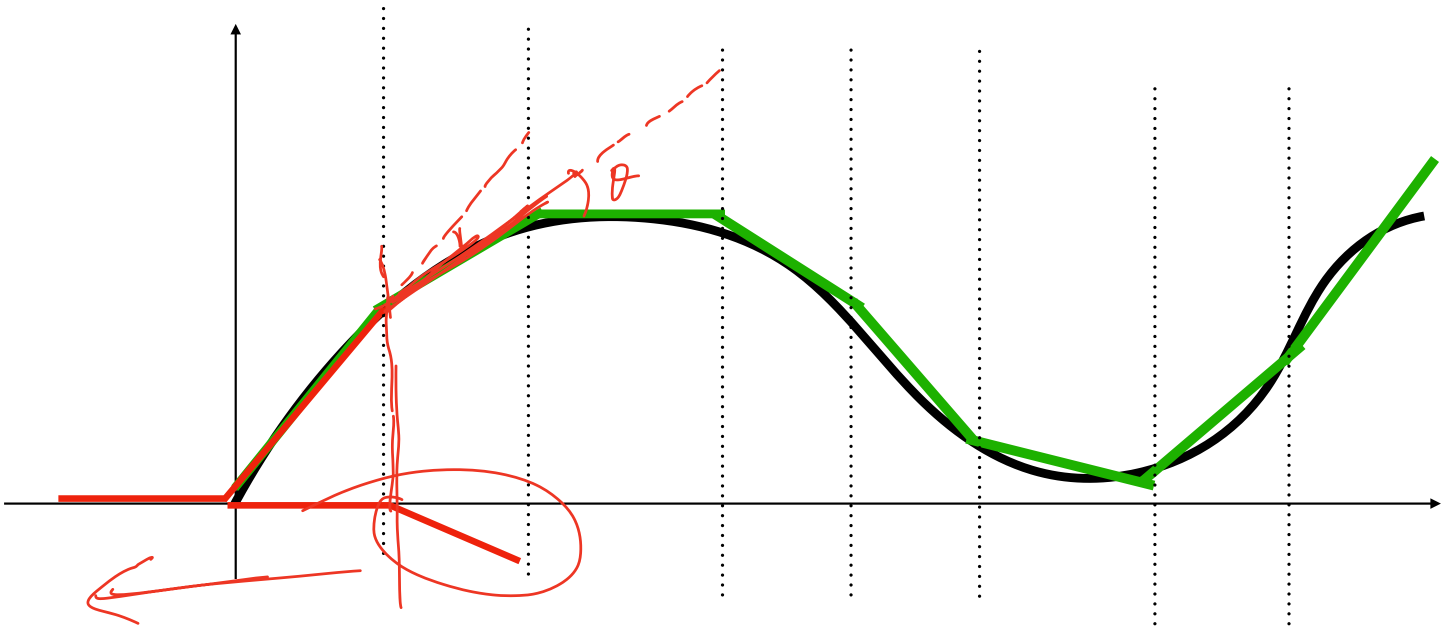
Claim: a wide enough one layer NN can approximate any smooth functions



What does a neural network approximate

$$y = \alpha^\top \left(\mathrm{ReLU}(Wx)\right) + b$$

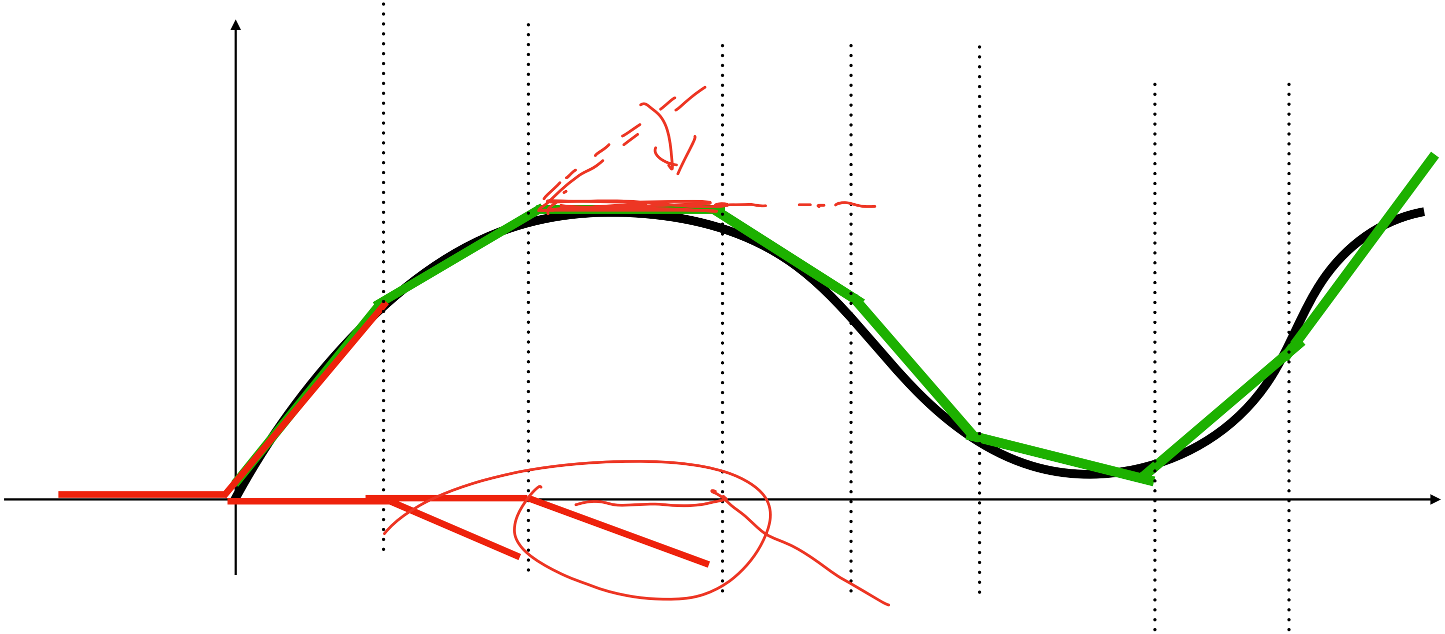
Claim: a wide enough one layer NN can approximate any smooth functions



What does a neural network approximate

$$y = \alpha^\top \left(\mathrm{ReLU}(Wx)\right) + b$$

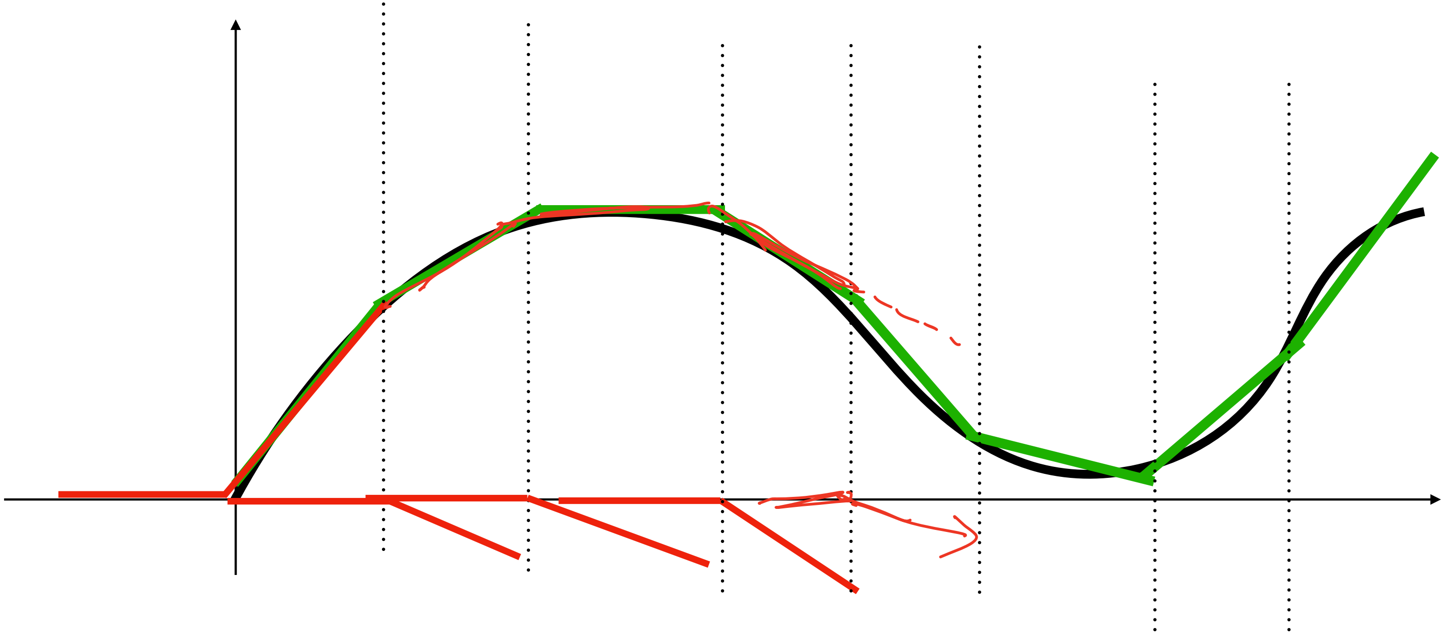
Claim: a wide enough one layer NN can approximate any smooth functions



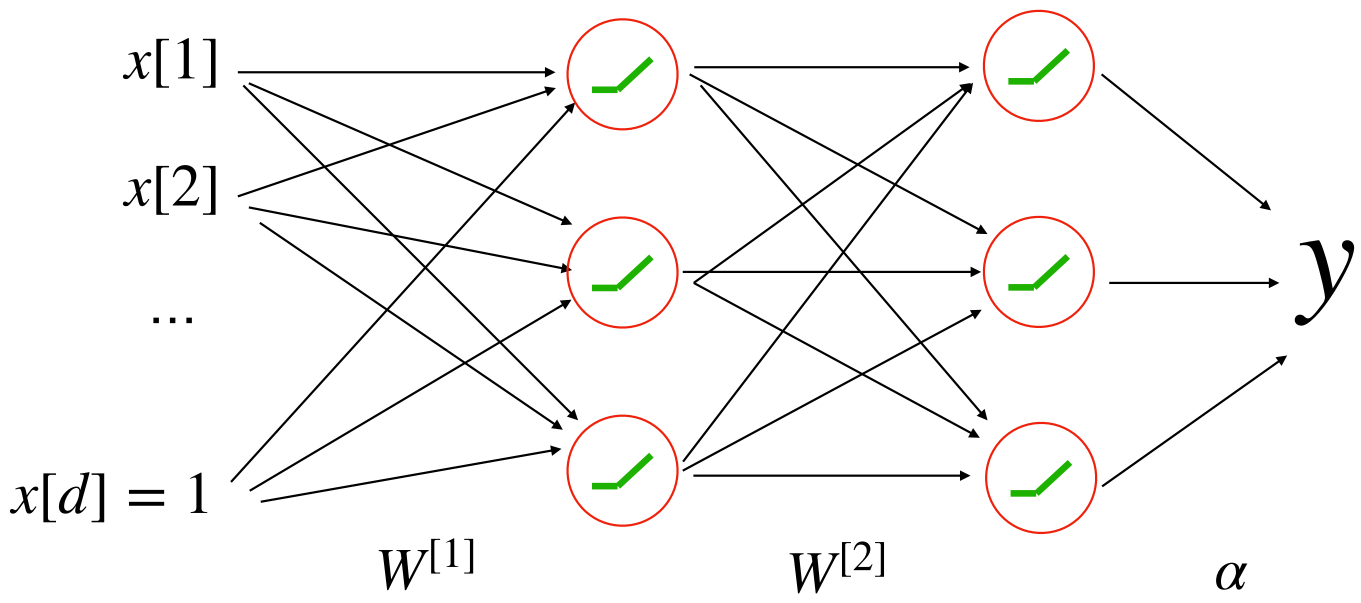
What does a neural network approximate

$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

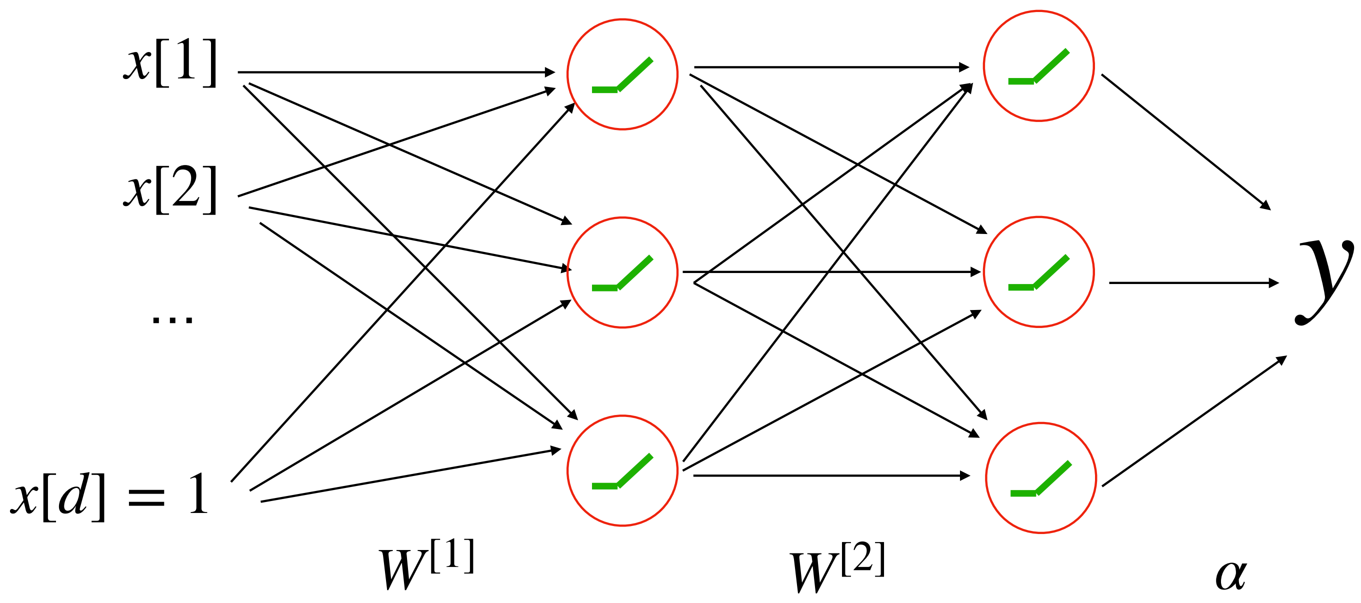
Claim: a wide enough one layer NN can approximate any smooth functions



A multi-layer fully connected neural network

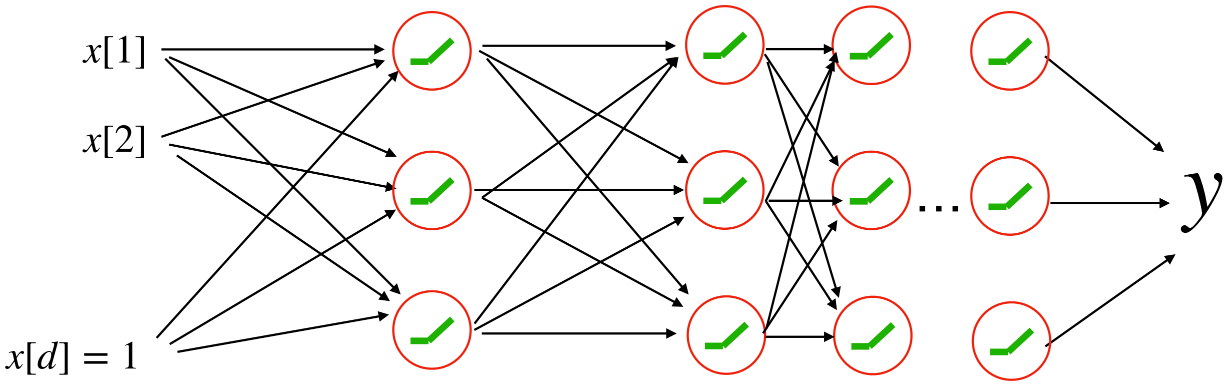


A multi-layer fully connected neural network

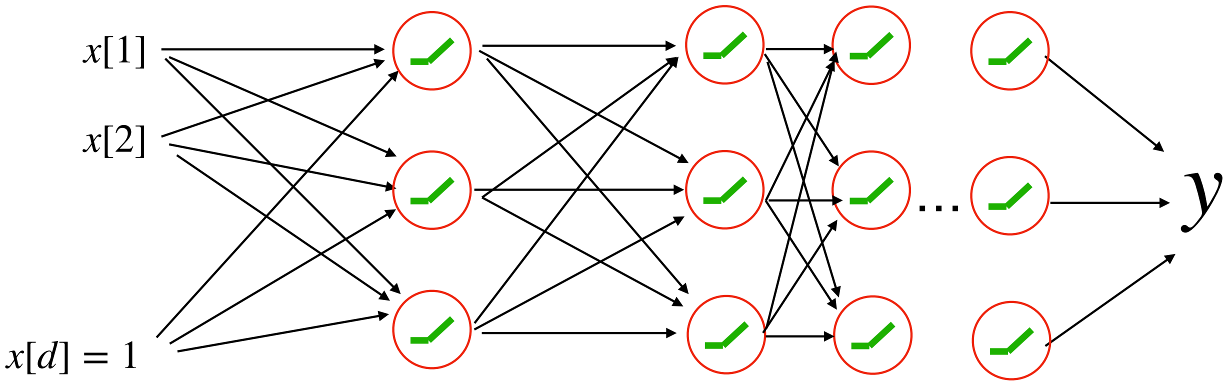


$$y = \alpha^{\top}\mathrm{ReLU}\left(W^{[2]}\mathrm{ReLU}\left(W^{[1]}x\right)\right) + b$$

A multi-layer fully connected neural network

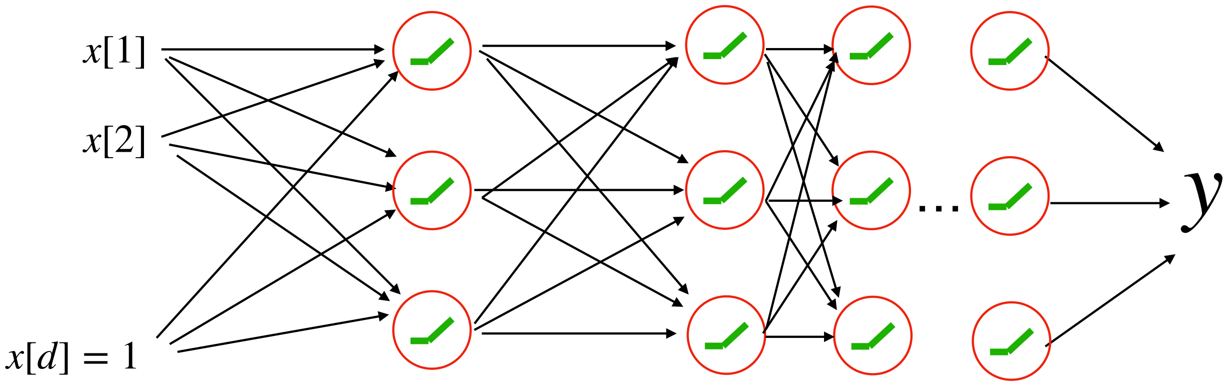


A multi-layer fully connected neural network



Define it by a forward pass:

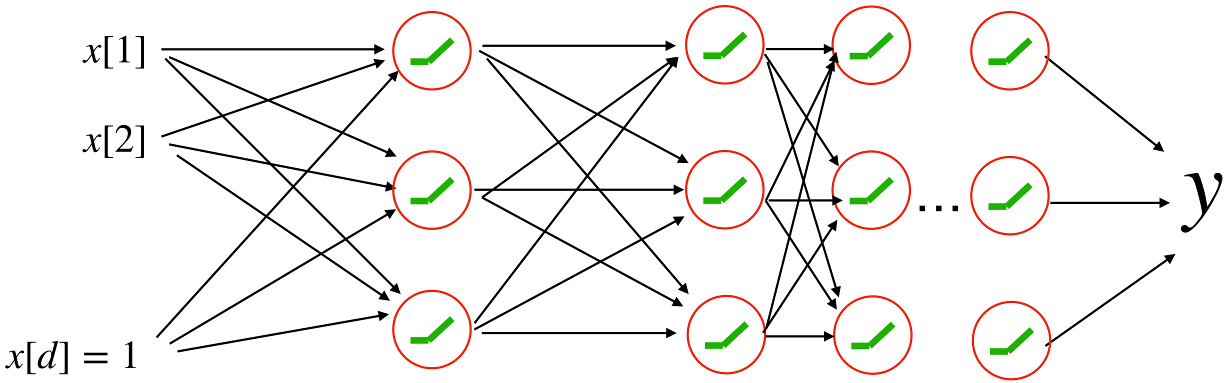
A multi-layer fully connected neural network



Define it by a forward pass:

$$z^{[1]} = x$$

A multi-layer fully connected neural network

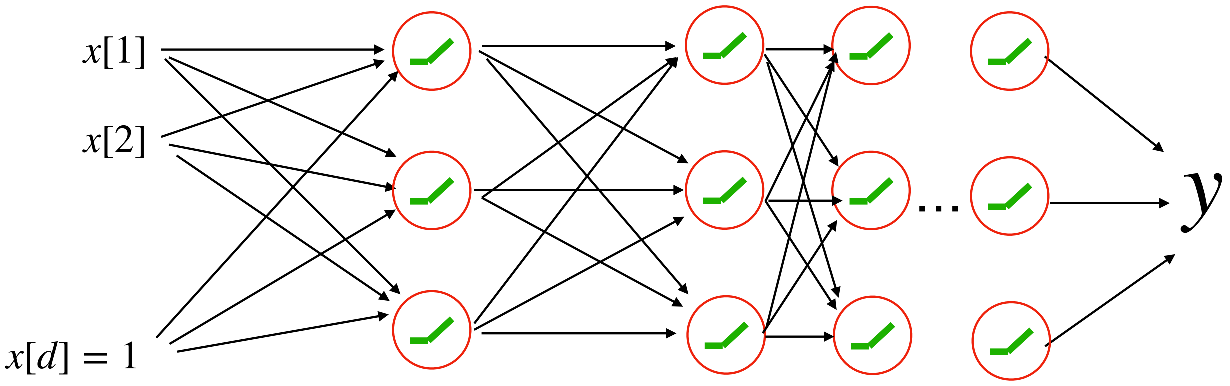


Define it by a forward pass:

$$z^{[1]} = x$$

For t = 1 to T-1:

A multi-layer fully connected neural network



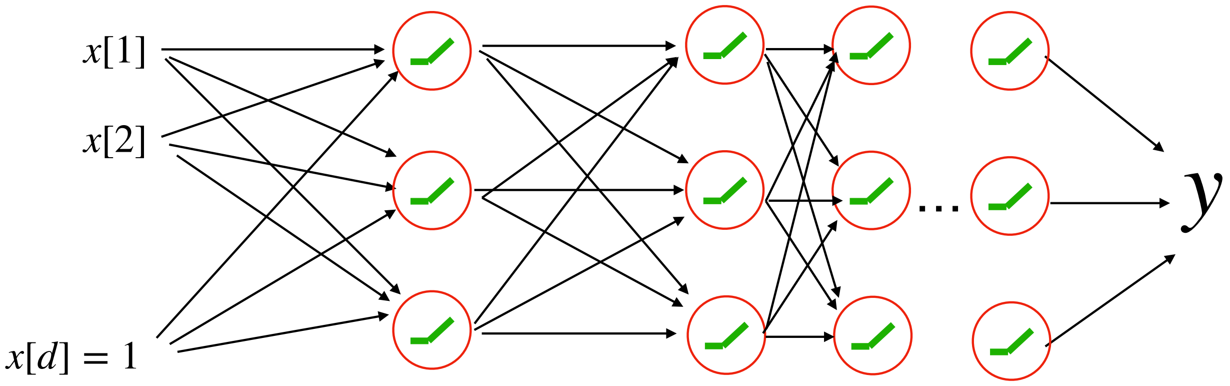
Define it by a forward pass:

$$z^{[1]} = x$$

For t = 1 to T-1:

$$z^{[t+1]} = \text{ReLU}\left(W^{[t]} z^{t}\right)$$

A multi-layer fully connected neural network



Define it by a forward pass:

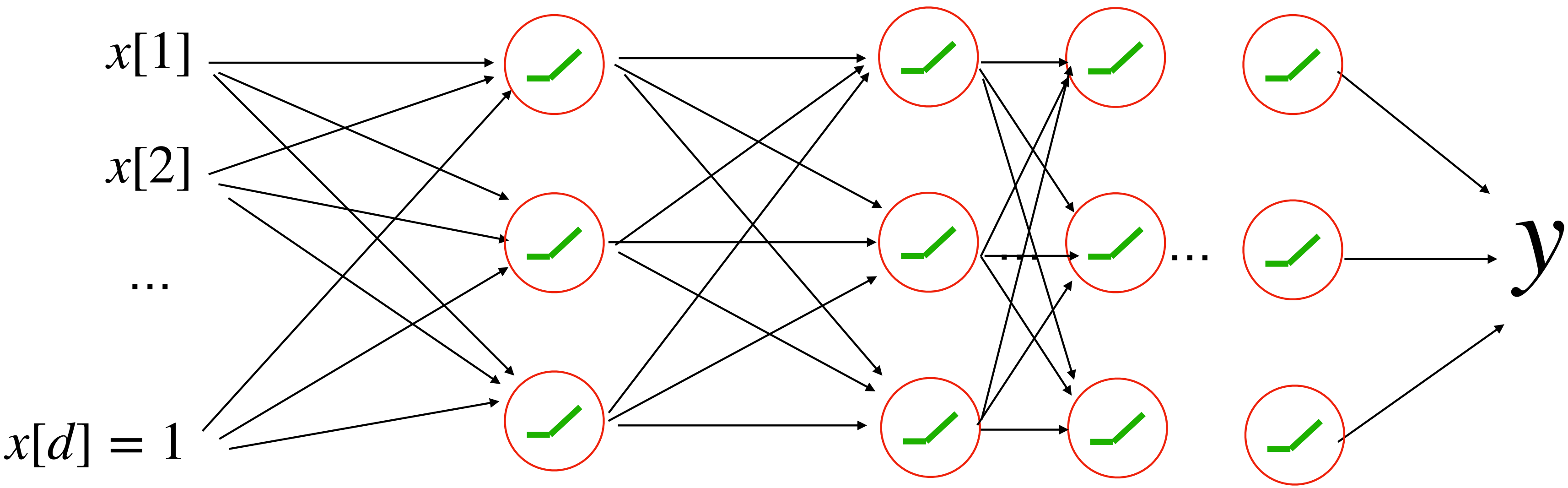
$$z^{[1]} = x$$

For t = 1 to T-1:

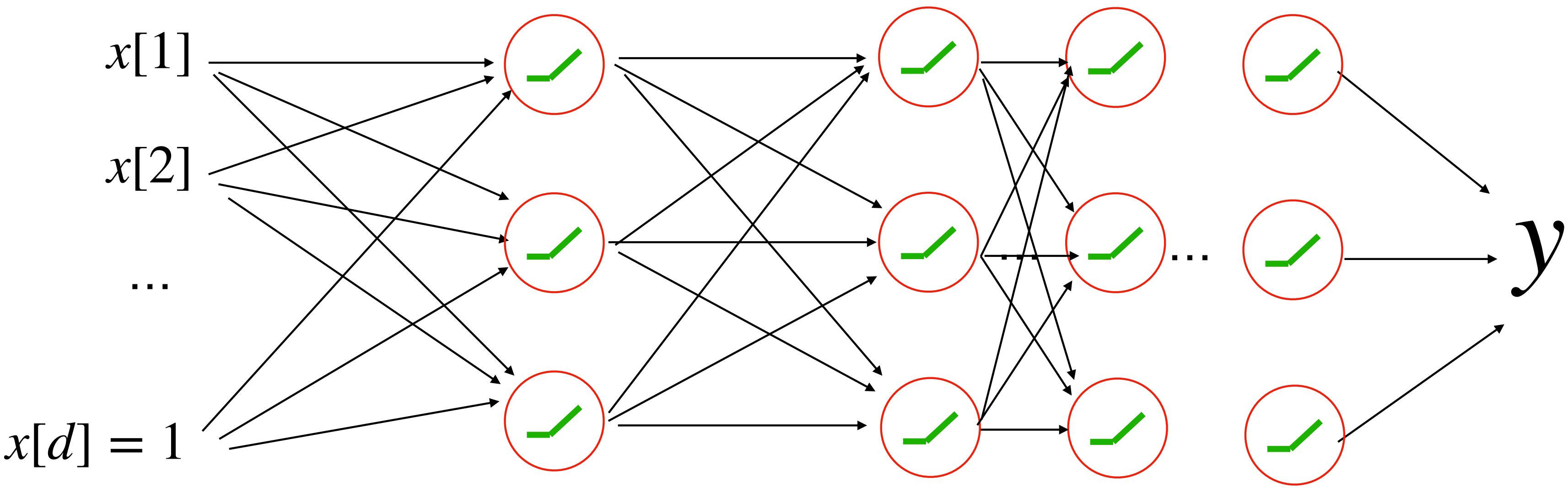
$$z^{[t+1]} = \text{ReLU}\left(W^{[t]} z^{t}\right)$$

$$y = \alpha^\top z^{[T]} + b$$

The benefits of going deep



The benefits of going deep



Allows us to represent complicated functions without making NN too wide

Summary for today

Neural network is universal function approximation

Next lecture: backpropagation for computing gradients efficiently