# Neural Network

# Announcements

# 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

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

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

For t = 1 …

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

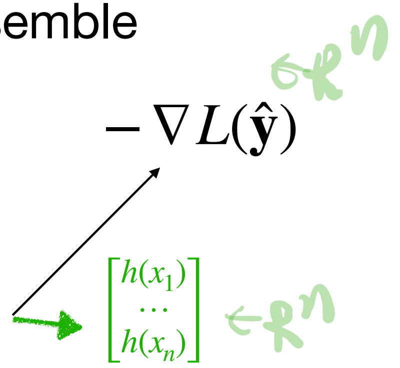
# Recap on Boosting

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

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

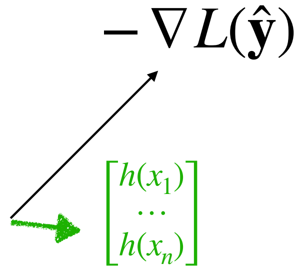
For t = 1 …

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



# Recap on Boosting

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




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

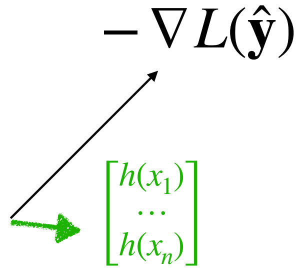
For t = 1 …

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




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:**

$H_t(x_i)$

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

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

$\sum_i p_i = 1$

$p_i \geq 0$

**2. Add new learner to the ensemble:**

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

$\epsilon = \sum_{i: h_{t+1}(x_i) \neq y_i} p_i$

# Outline of Today

1. Analysis of Boosting

2. Multilayer feedforward Neural Network

3. Training a neural network

# The definition of Weak learning

Each weaker learning optimizes its own data:

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

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

# The definition of Weak learning

Each weaker learning optimizes its own data:

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

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

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:

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

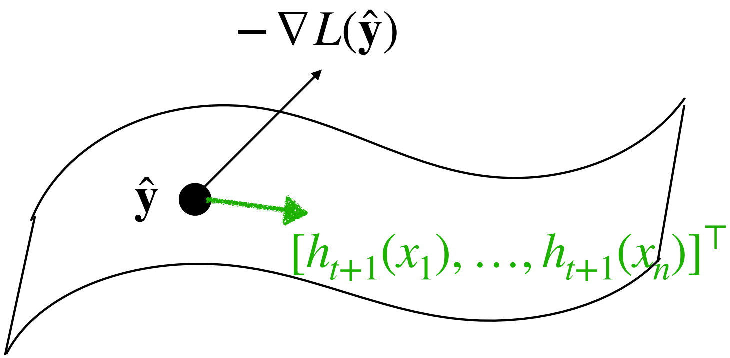
$$h_{t+1} = \arg\min_{h \in \mathscr{H}} \sum_{i=1}^{n} p_i \cdot \mathbf{1}(h(x_i) \neq y_i)$$

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

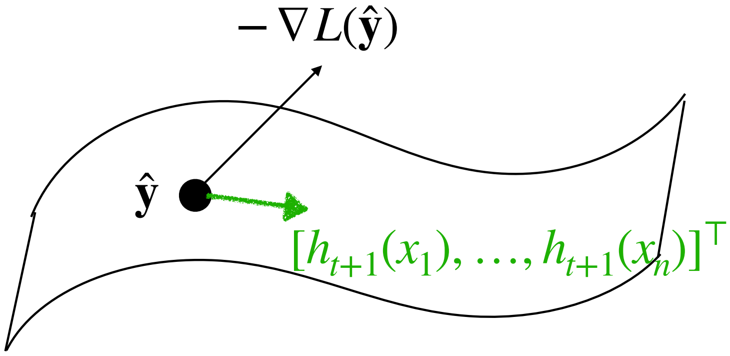
Assume that weaker learner's loss $\epsilon := \sum_{i=1}^{n} p_i \mathbf{1}\{h_{t+1}(x_i) \neq y_i\} \leq \frac{1}{2} - \gamma, \ \ \gamma > 0$



# Weaker learnability implies approximating gradient well

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

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

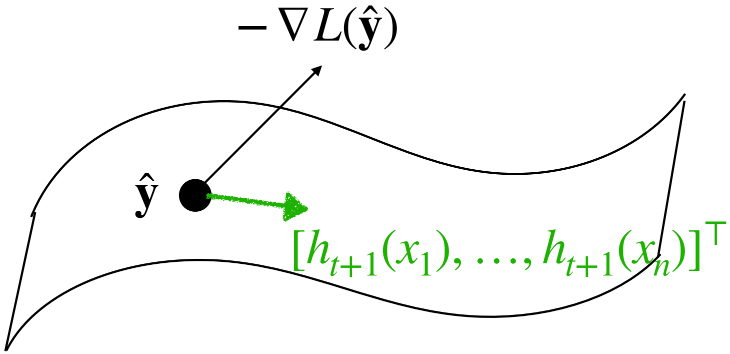


# Weaker learnability implies approximating gradient well

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

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

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

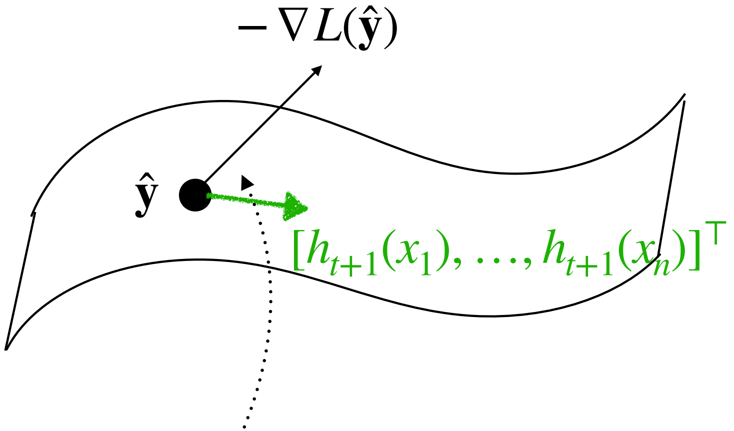


# Weaker learnability implies approximating gradient well

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

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

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




Within 90 degree, so improve the objective!

# Formal Convergence of AdaBoost

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

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

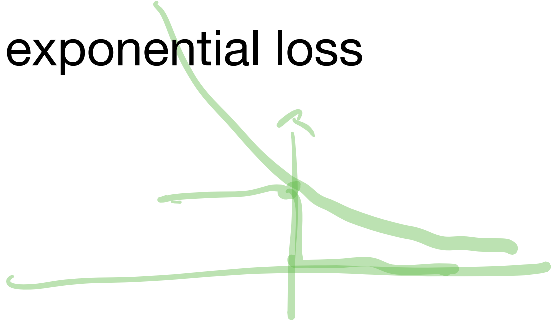
(Proof in lecture note, optional)

# Formal Convergence of AdaBoost

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

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

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

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

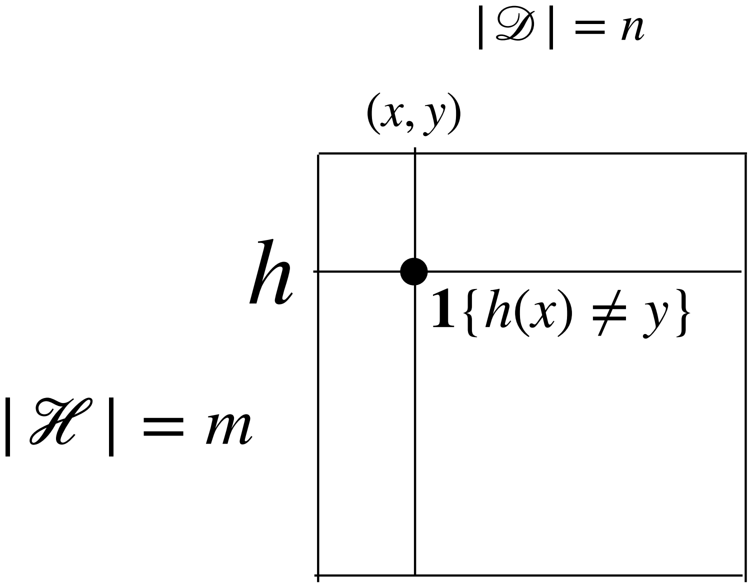
Note zero-one loss is upper bounded by exponential loss

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

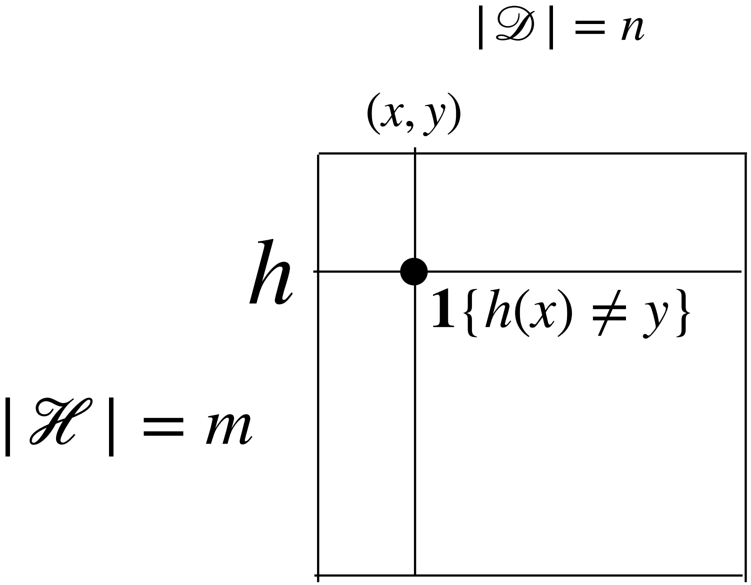
Avg number of mistakes

(Proof in lecture note, optional)

# Thinking about Boosting via two player zero sum game



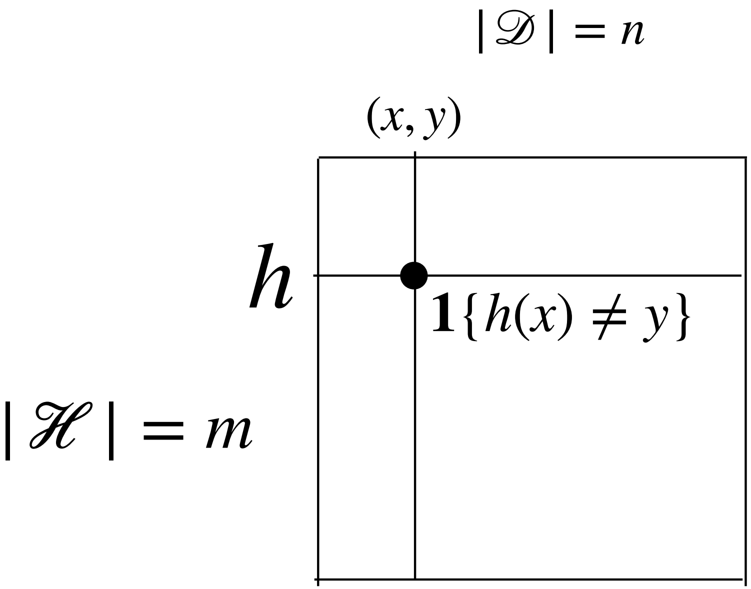
# Thinking about Boosting via two player zero sum game




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

Column player plays example $(x, y)$

# Thinking about Boosting via two player zero sum game

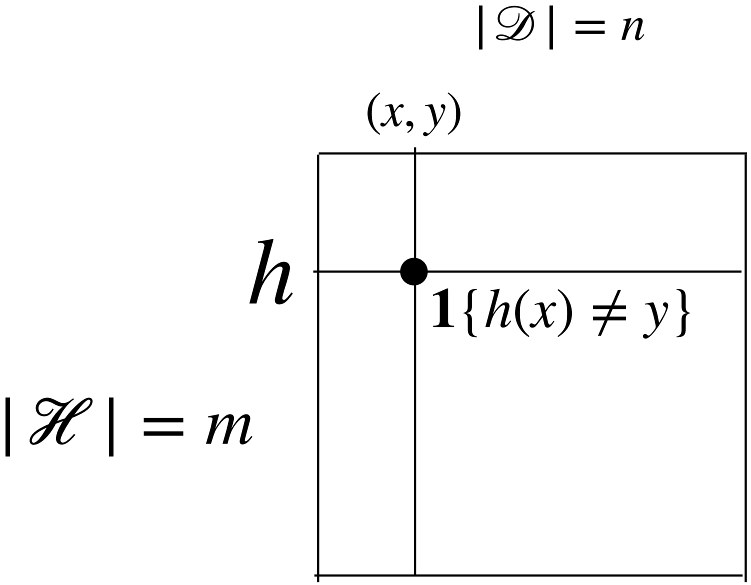



Row player plays hypothesis $h \in \mathcal{H}$

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



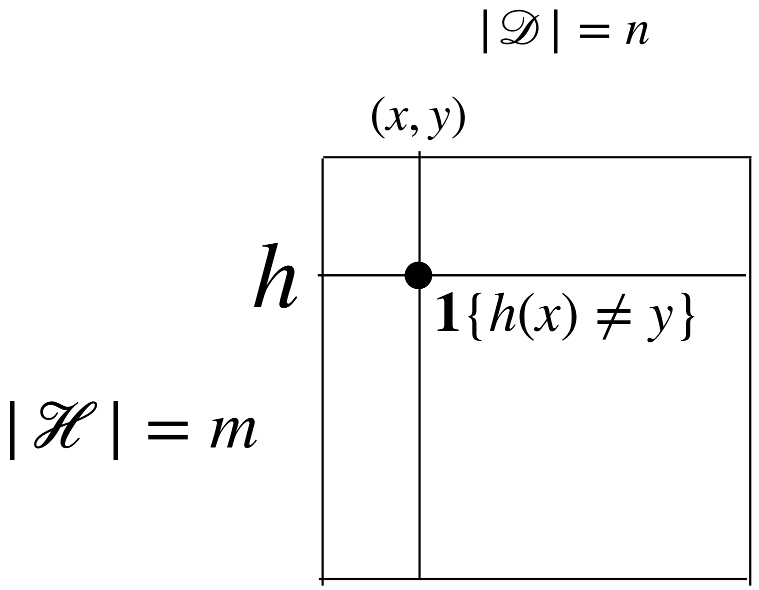

Row player plays hypothesis $h \in \mathcal{H}$

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

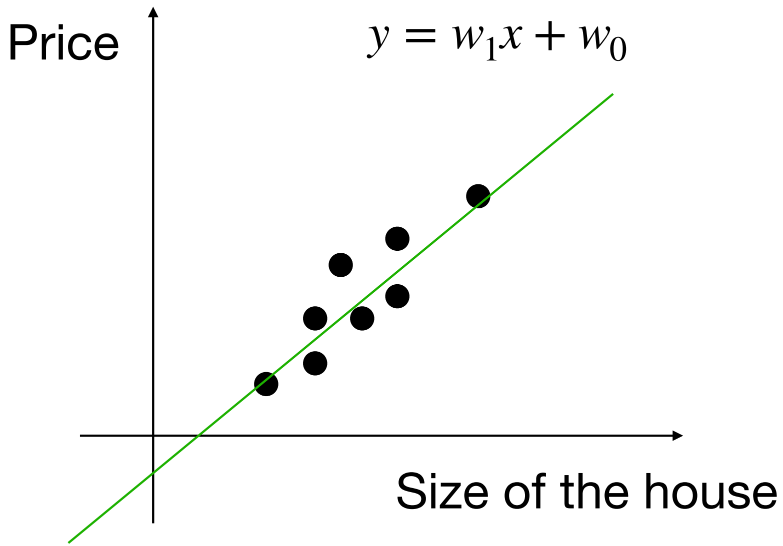
# Outline of Today

1. Analysis of Boosting

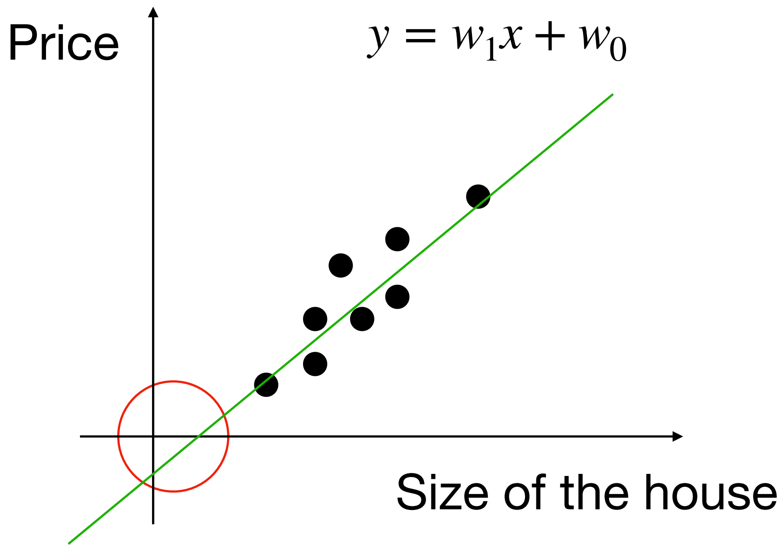
2. Multilayer feedforward Neural Network

3. Training a neural network

# Linear Regression Revisit

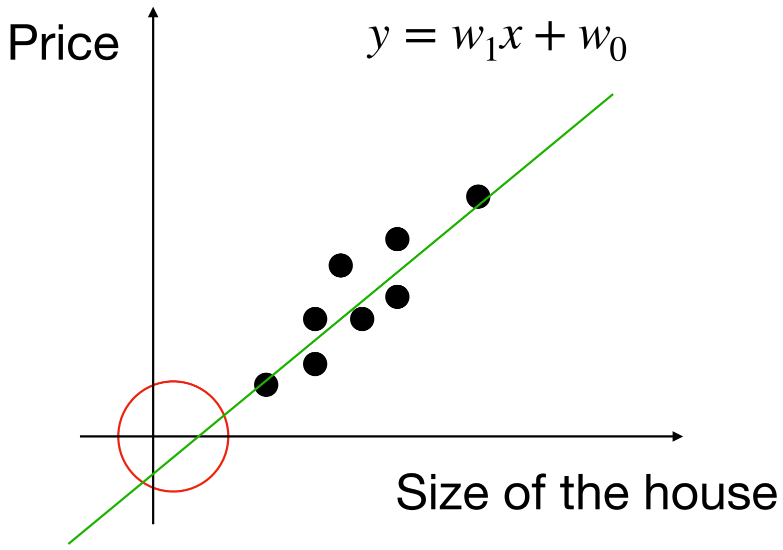


# Linear Regression Revisit




Negative part does not make too much sense

# Linear Regression Revisit

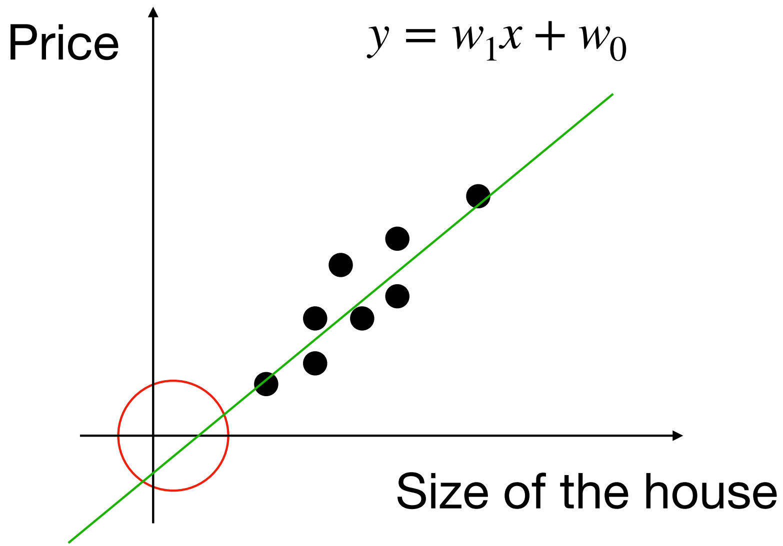



Negative part does not make too much sense

We can fix this with a simple nonlinear function

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

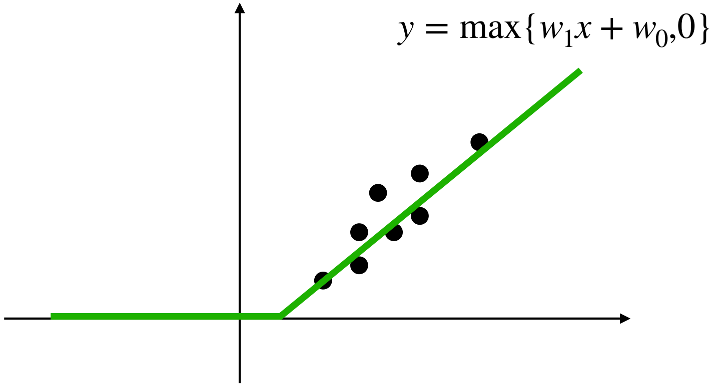
# Linear Regression Revisit



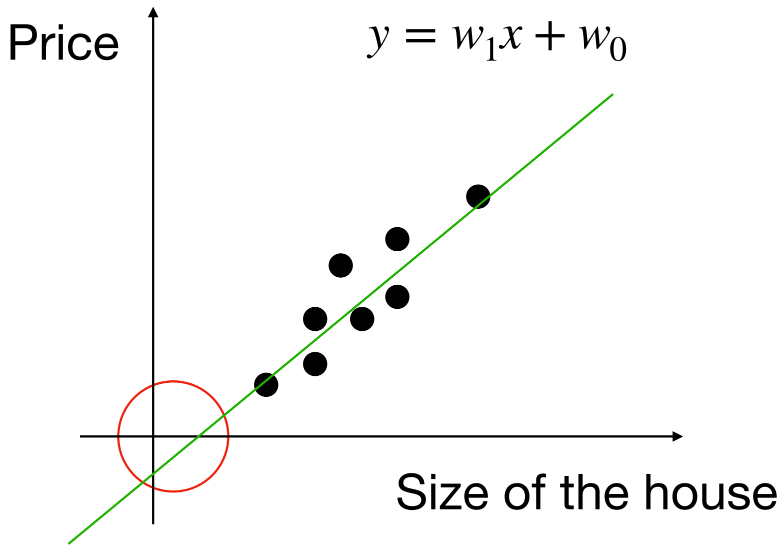

Negative part does not make too much sense

We can fix this with a simple nonlinear function

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



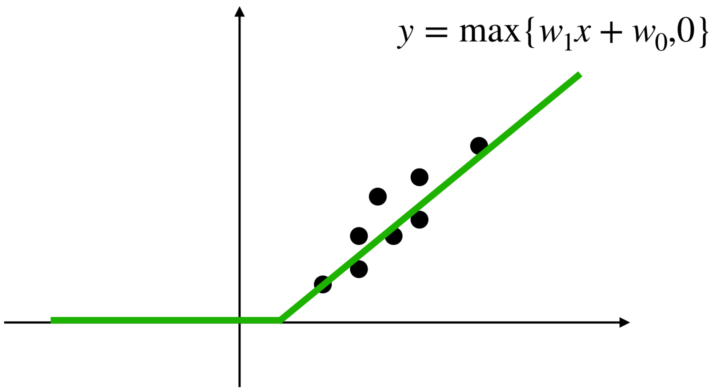
# Linear Regression Revisit




Negative part does not make too much sense

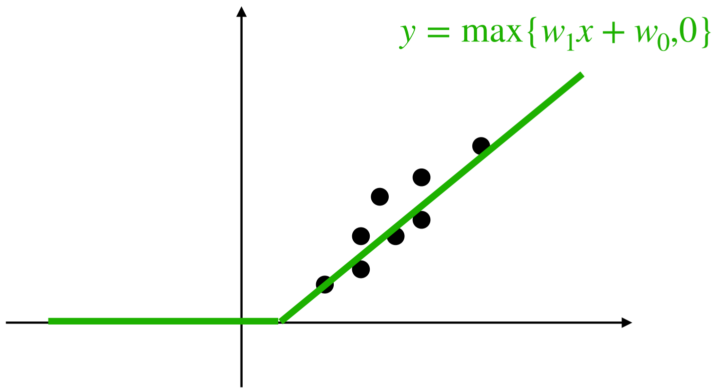
We can fix this with a simple nonlinear function

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

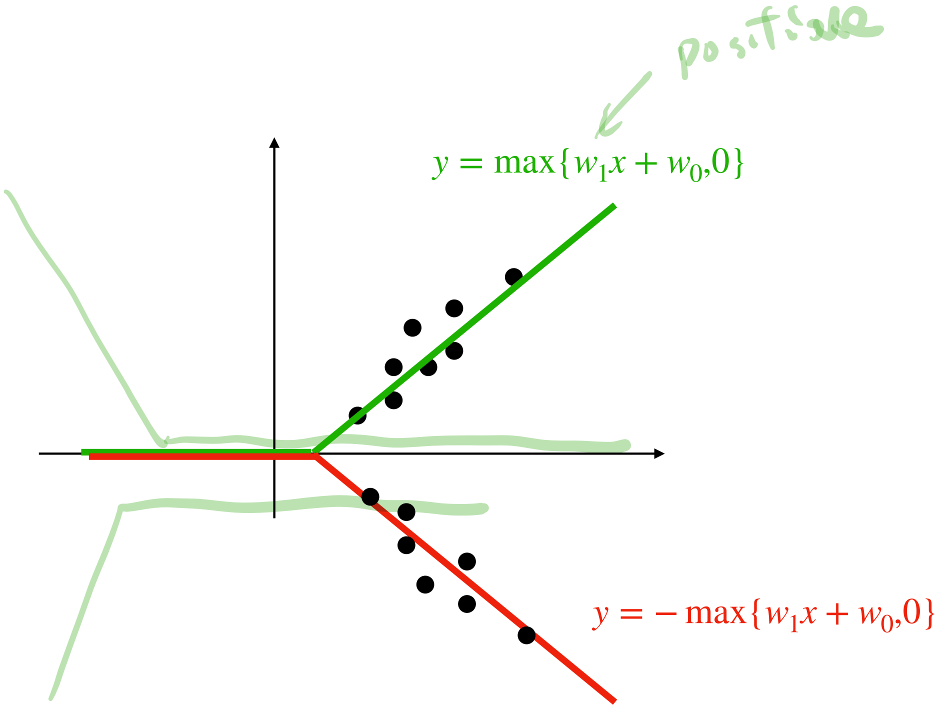



rectified linear unit (ReLU)

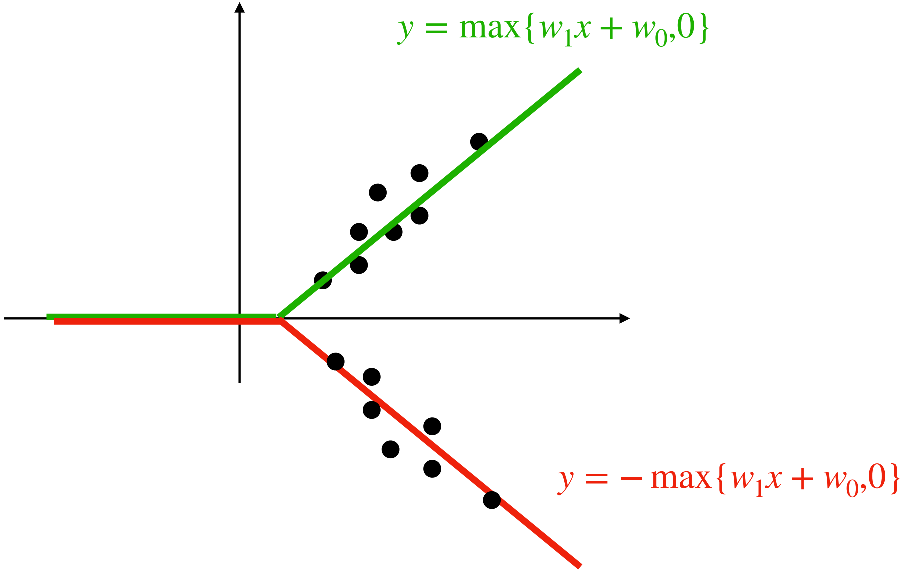
# A single neuron network



# A single neuron network



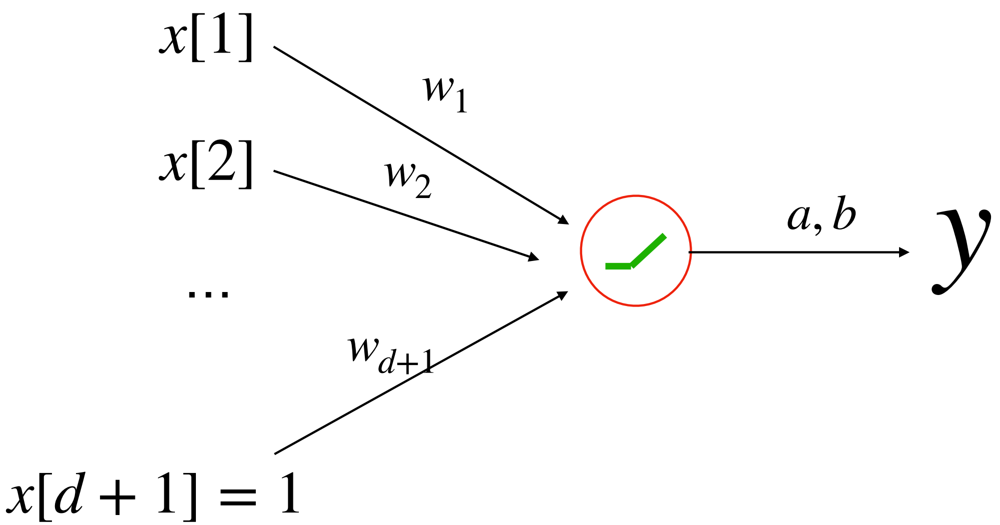
# A single neuron network



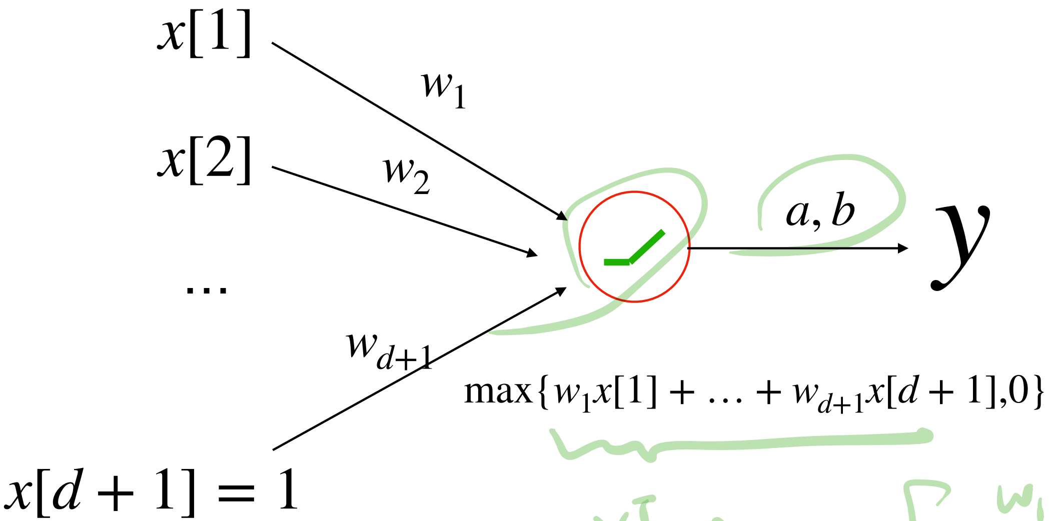

$$y = a \max\{w_1 x + w_0, 0\} + b$$

$(a, w_1, w_0, b)$

# A single neuron network



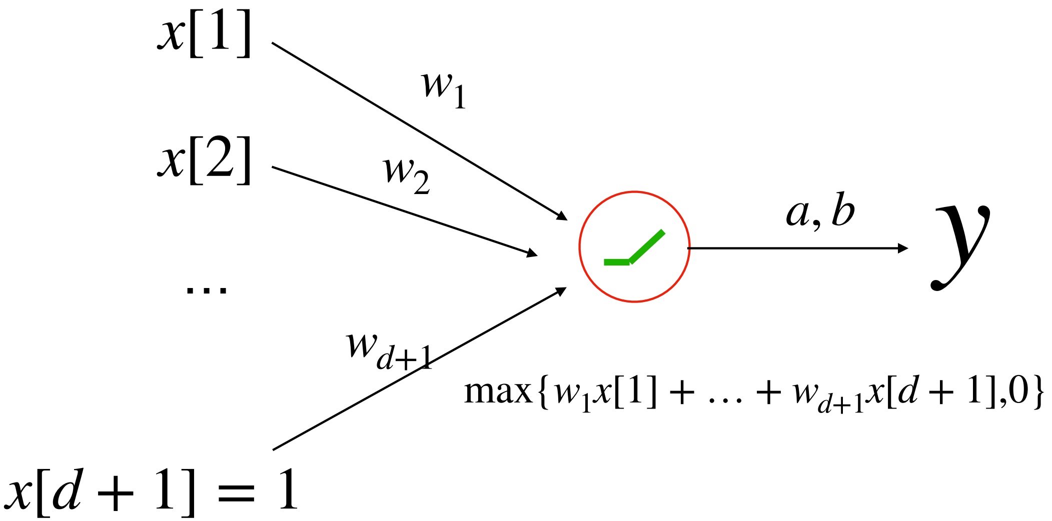
# A single neuron network




$X^T w$, $w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{d+1} \end{bmatrix} \in \mathbb{R}^{d+1}$

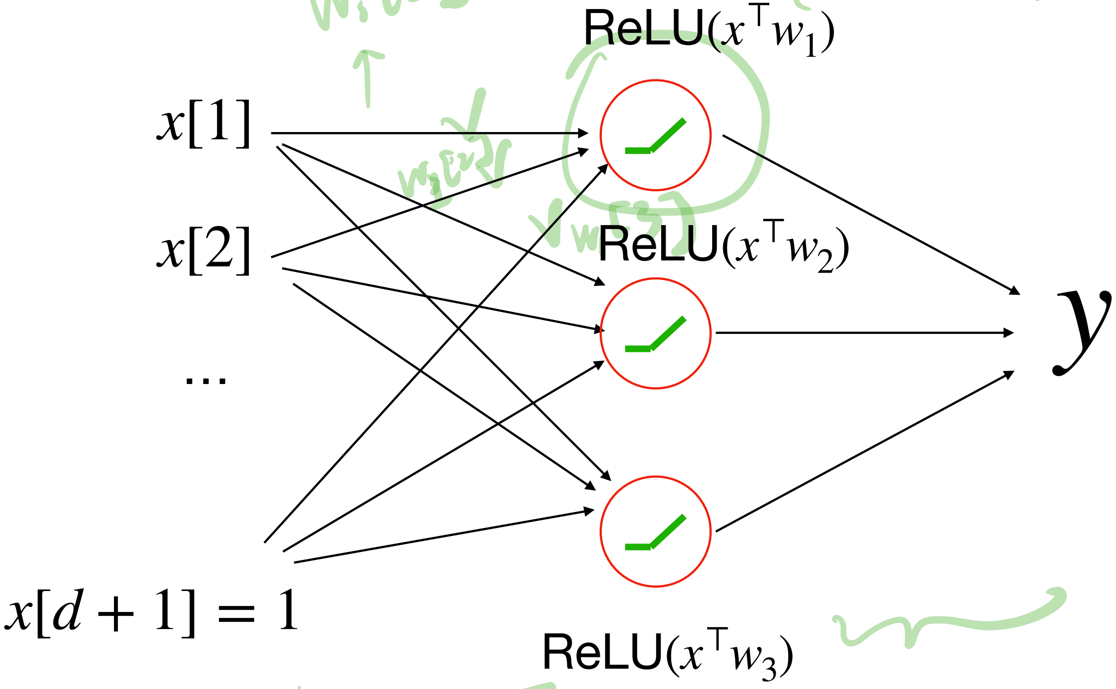
$X \in \mathbb{R}^{d+1}$

# A single neuron network



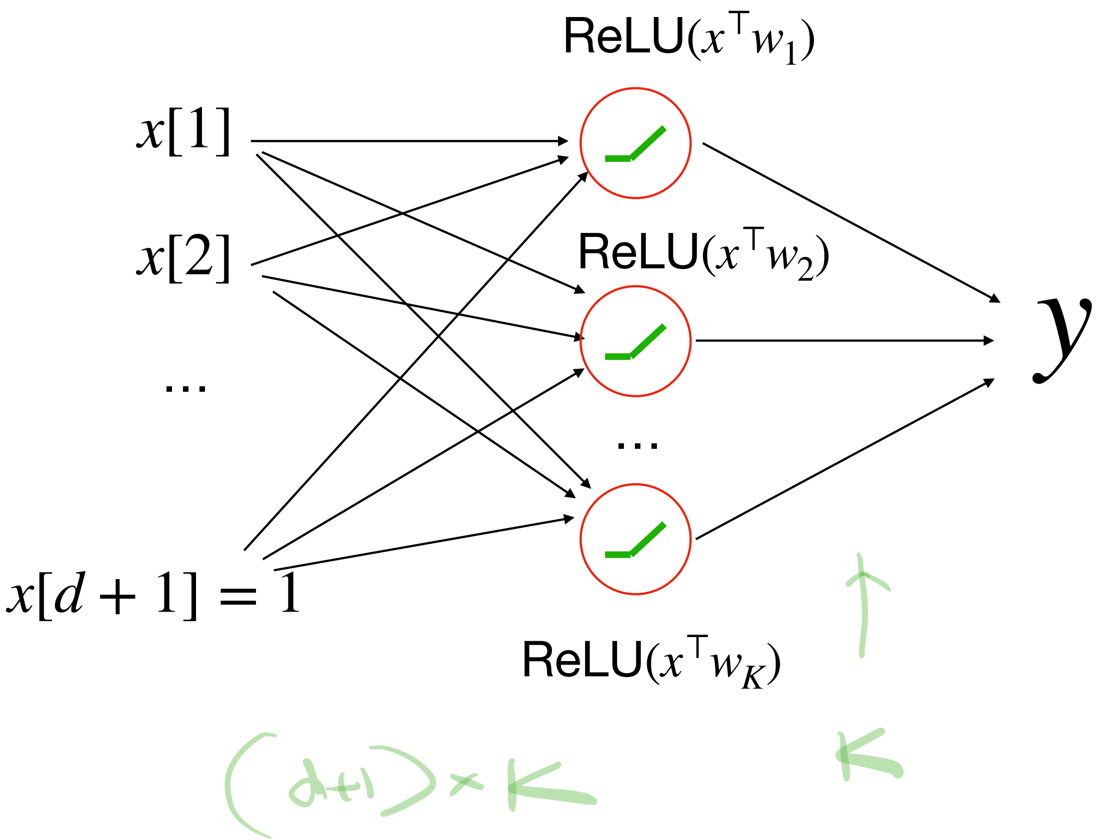

$$y = a\text{ReLU}(w^\top x) + b$$

# Let us stack multiple neurons together

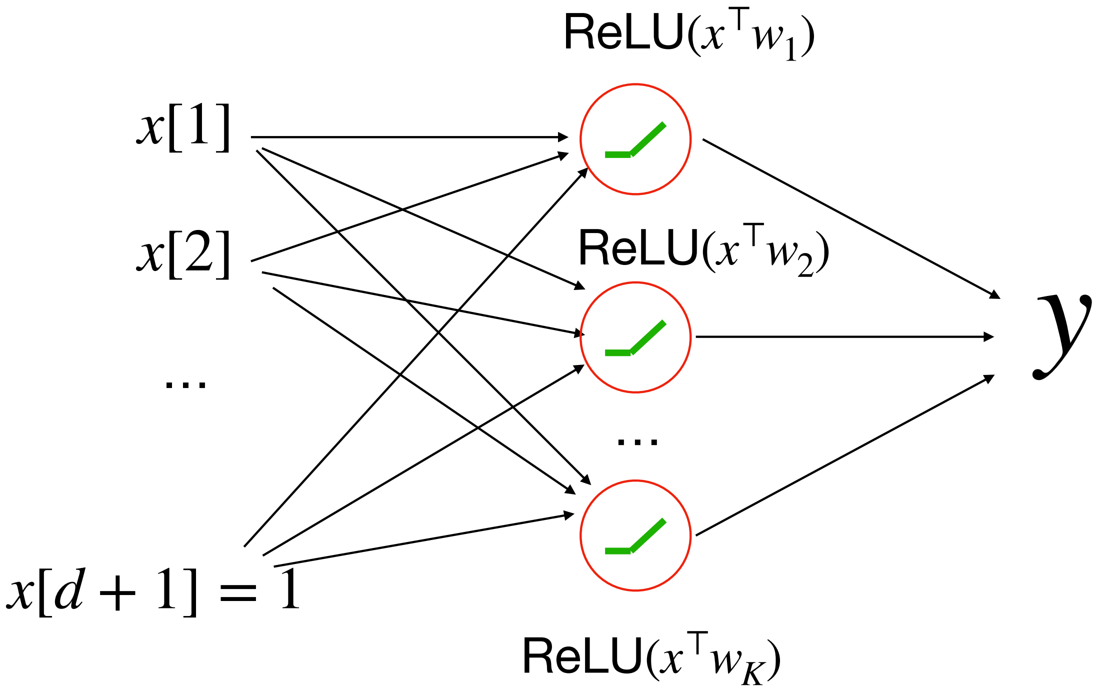



$$y = \sum_{i=1}^{3} a_i \text{ReLU}(x^\top w_i) + b$$

# Let us stack multiple neurons together



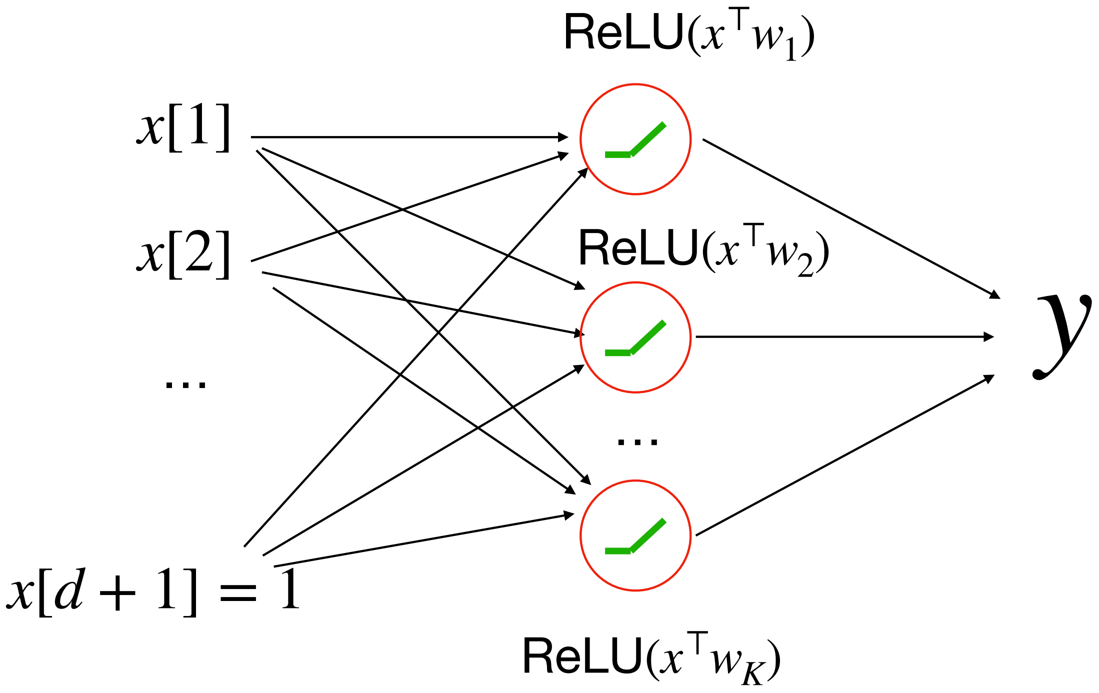
# Let us stack multiple neurons together




Vectorized form:

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

# Let us stack multiple neurons together

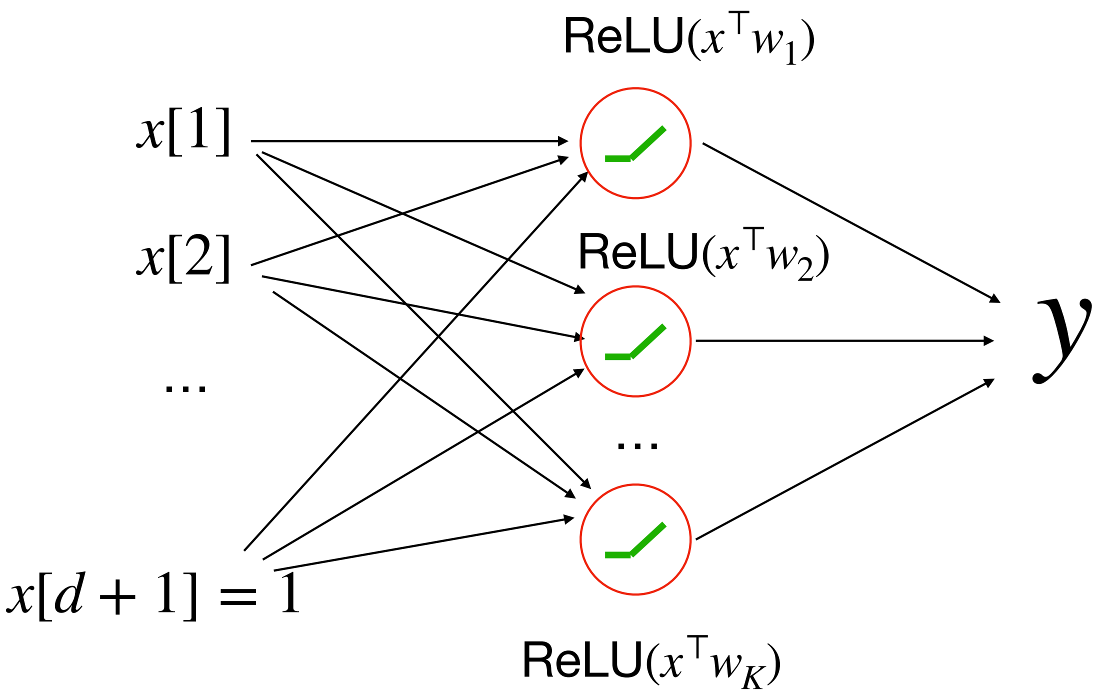



Vectorized form:

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

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

# Let us stack multiple neurons together



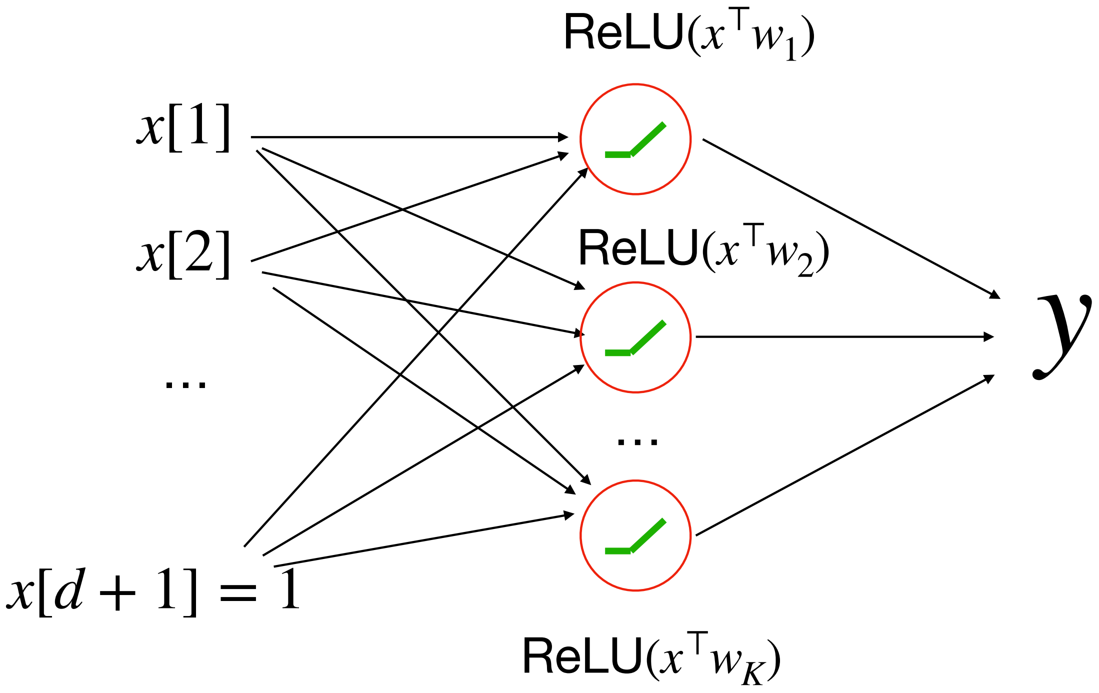

Vectorized form:

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

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

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

# Let us stack multiple neurons together




Vectorized form:

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

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

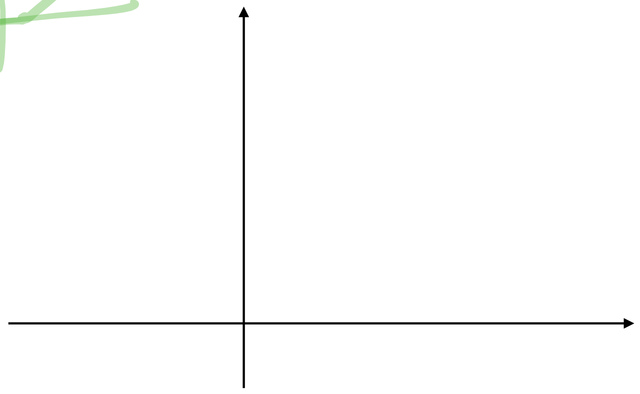
$$y = \alpha^\top \left( \text{ReLU}(Wx) \right) + b$$

Learnable feature $\phi(x)$

# What does a neural network approximate

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

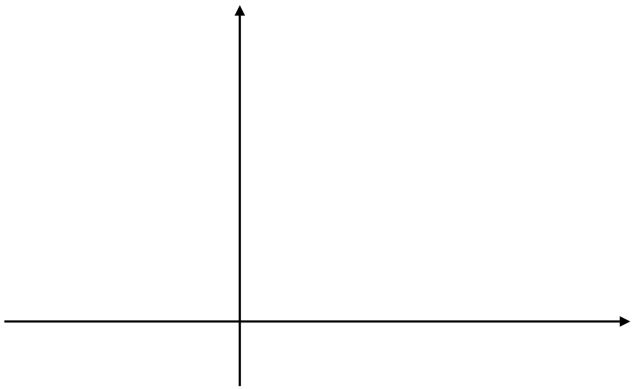
$$= \sum_{i=1}^{K} a_i \, \text{Relu}\left(w_i^T x\right) + b$$



# What does a neural network approximate

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

It's a pieces wise linear functions

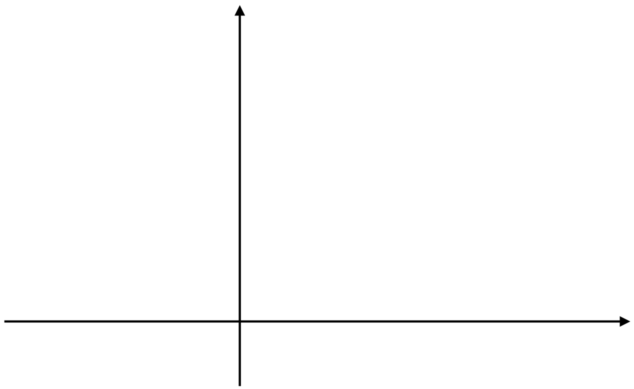


# What does a neural network approximate

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

It’s a pieces wise linear functions

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



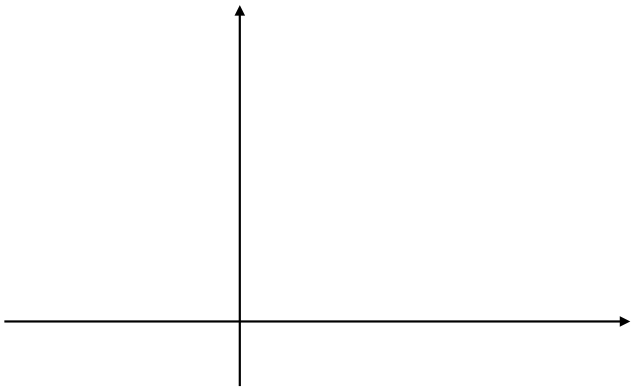
# What does a neural network approximate

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

It's a pieces wise linear functions

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

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



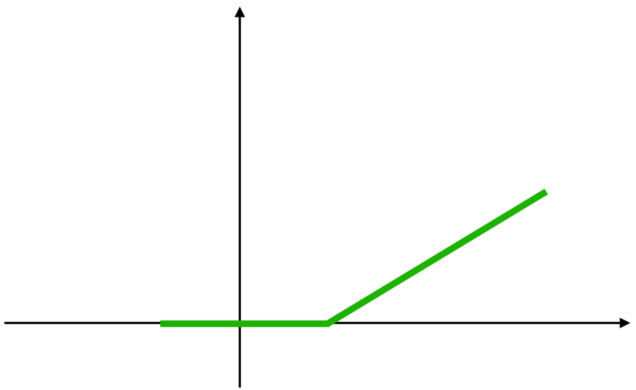
# What does a neural network approximate

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

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

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

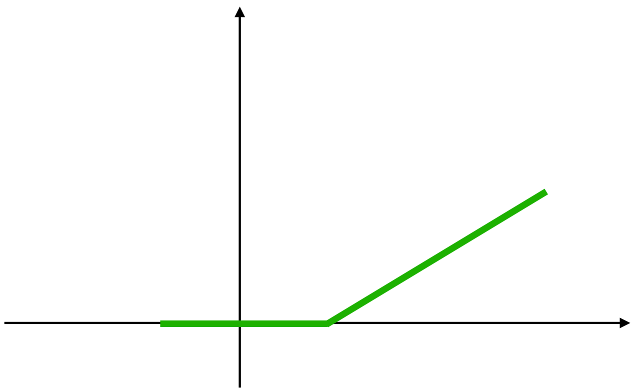
It's a pieces wise linear functions

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

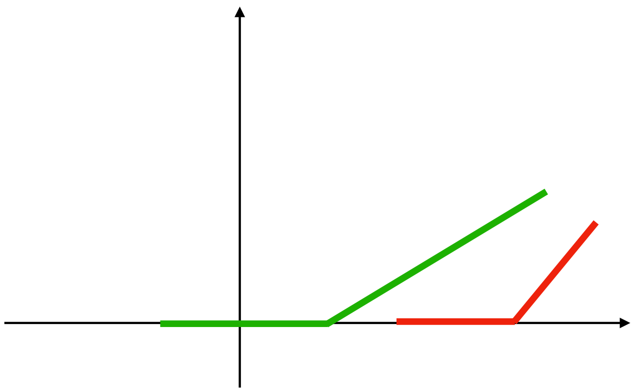
$$y = \alpha^\top \left(\text{ReLU}(Wx)\right) + b$$

It's a pieces wise linear functions

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

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

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



# What does a neural network approximate

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

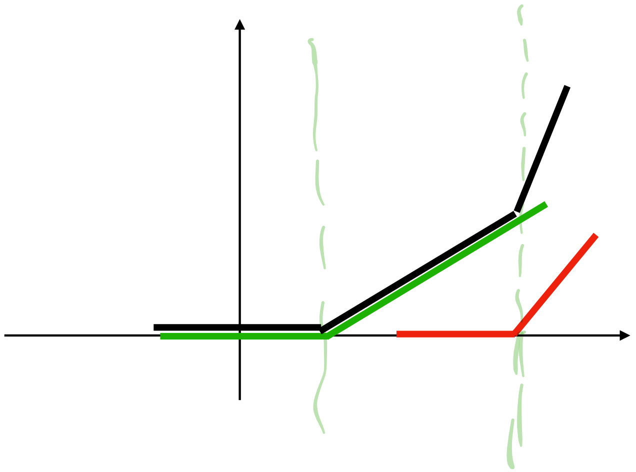
It's a pieces wise linear functions

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

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

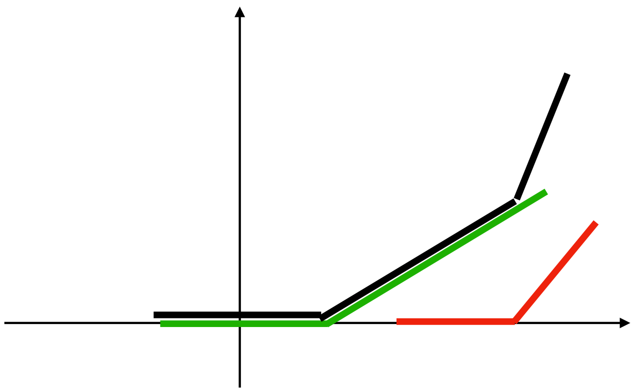
It's a pieces wise linear functions

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

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

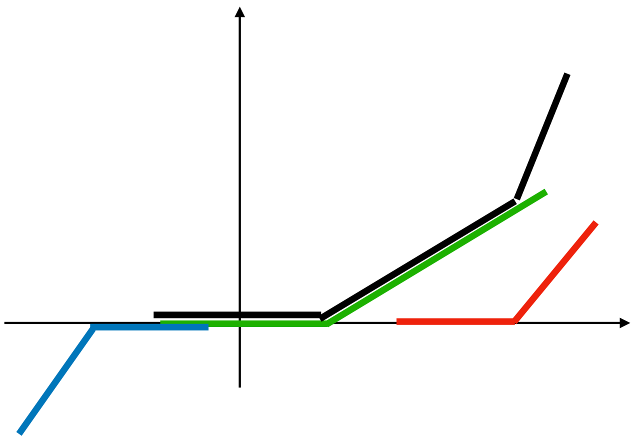
It's a pieces wise linear functions

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

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

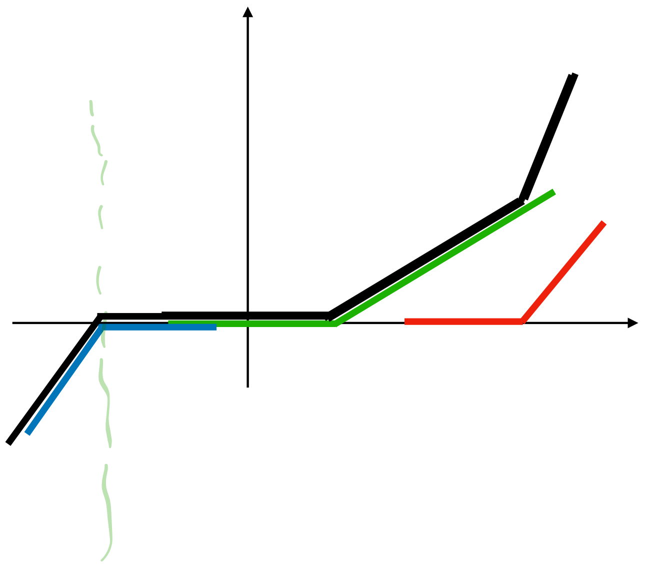
It's a pieces wise linear functions

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

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

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

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



# What does a neural network approximate

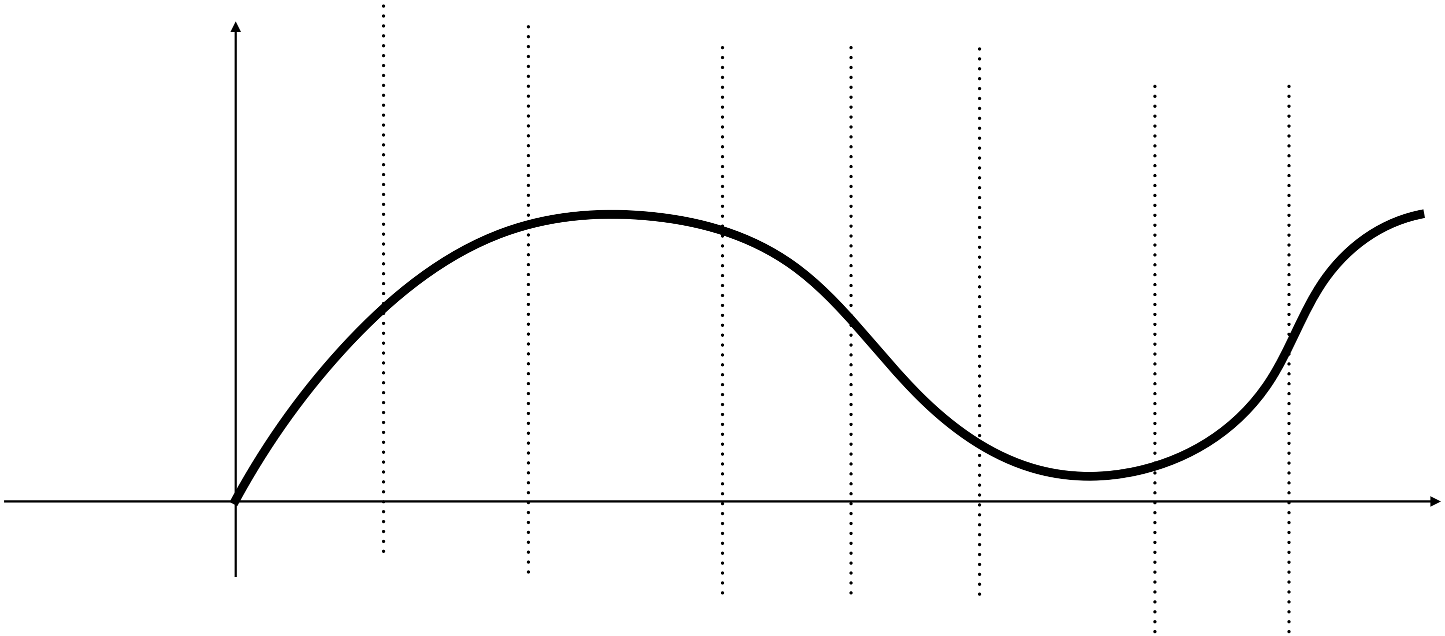
$$y = \alpha^\top \left(\mathrm{ReLU}(Wx)\right) + b$$

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

# What does a neural network approximate

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

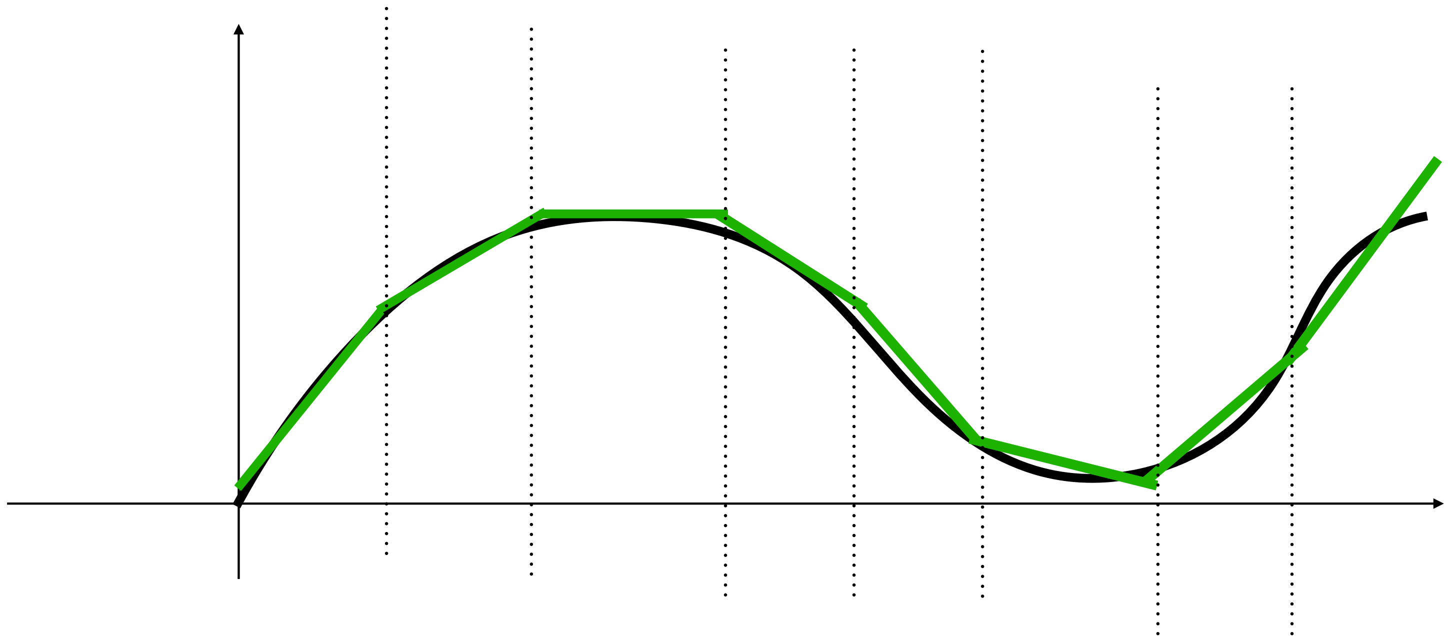
Claim: a wide enough one layer NN can approximate any smooth functions



# What does a neural network approximate

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

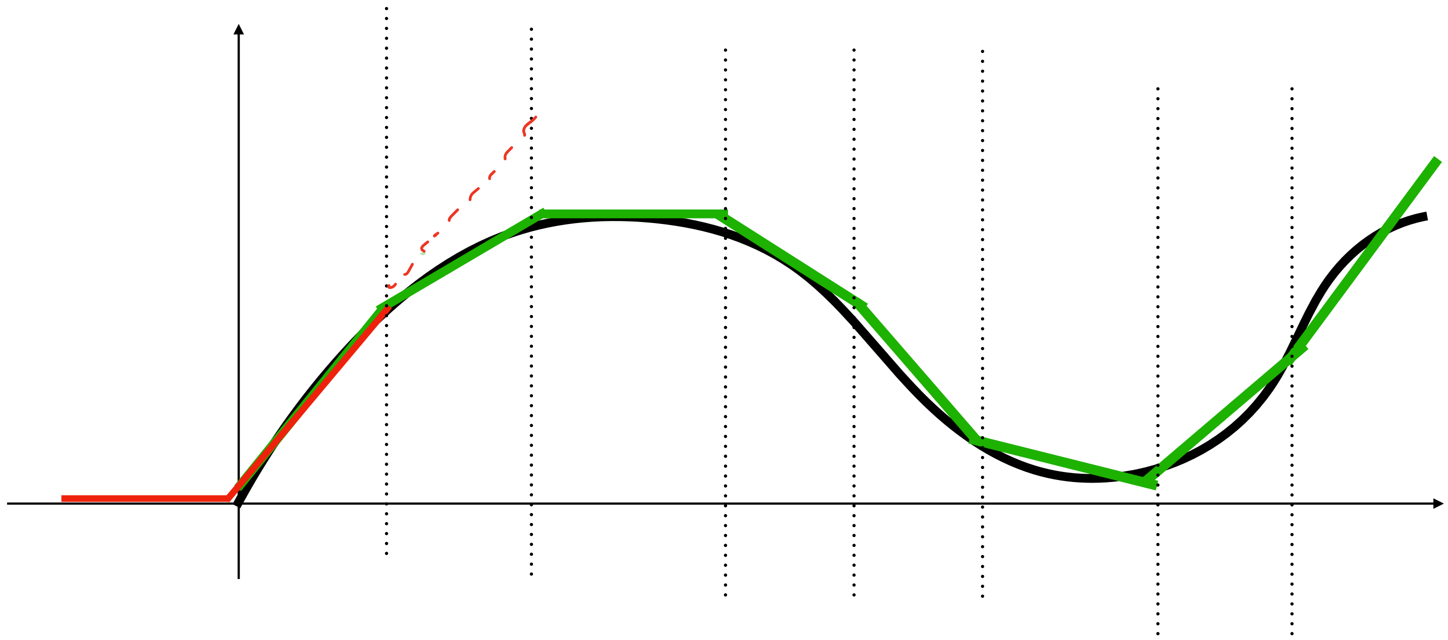
Claim: a wide enough one layer NN can approximate any smooth functions



# What does a neural network approximate

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

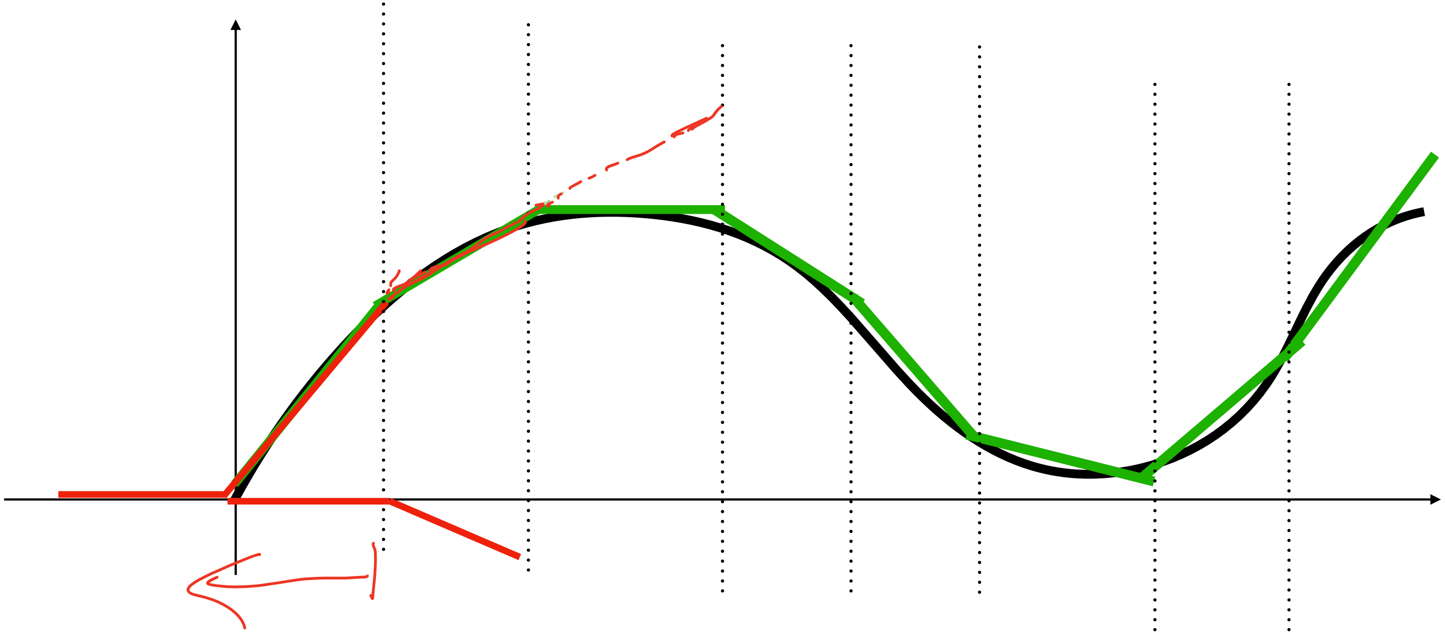
Claim: a wide enough one layer NN can approximate any smooth functions



# What does a neural network approximate

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

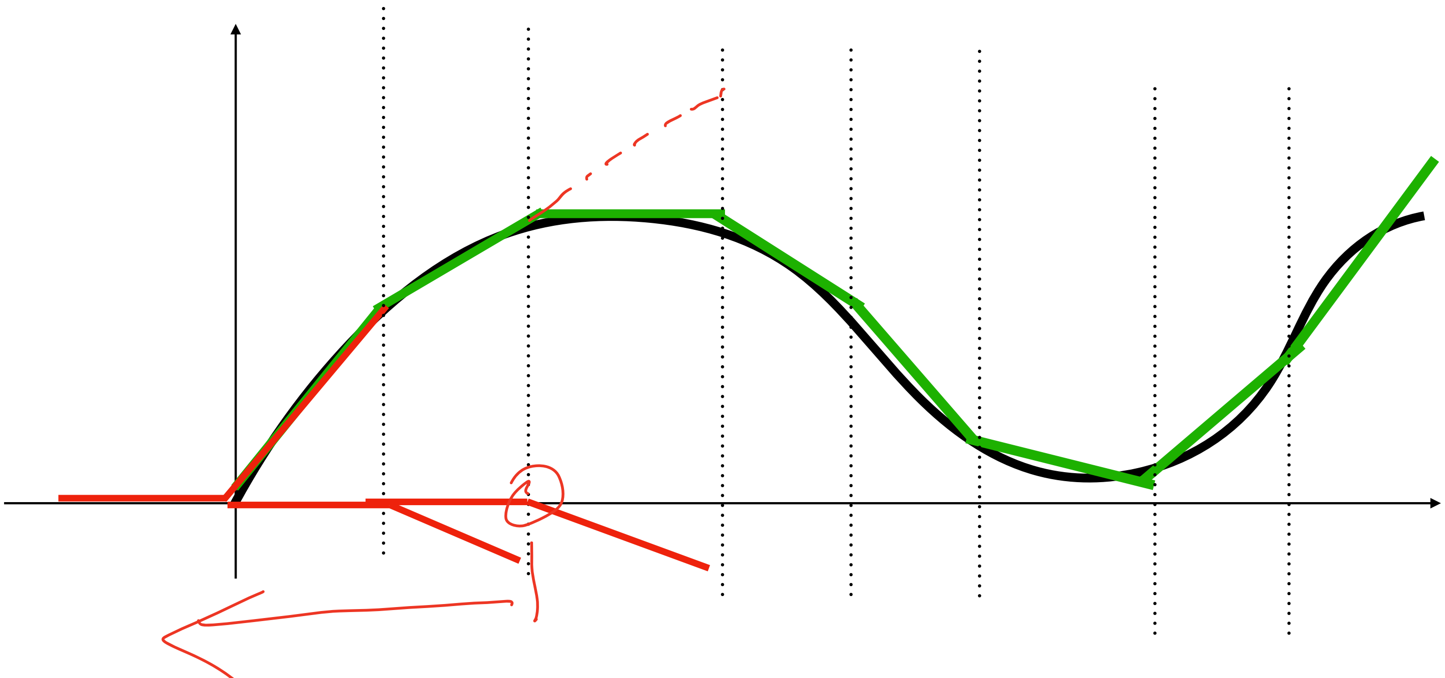
Claim: a wide enough one layer NN can approximate any smooth functions



# What does a neural network approximate

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

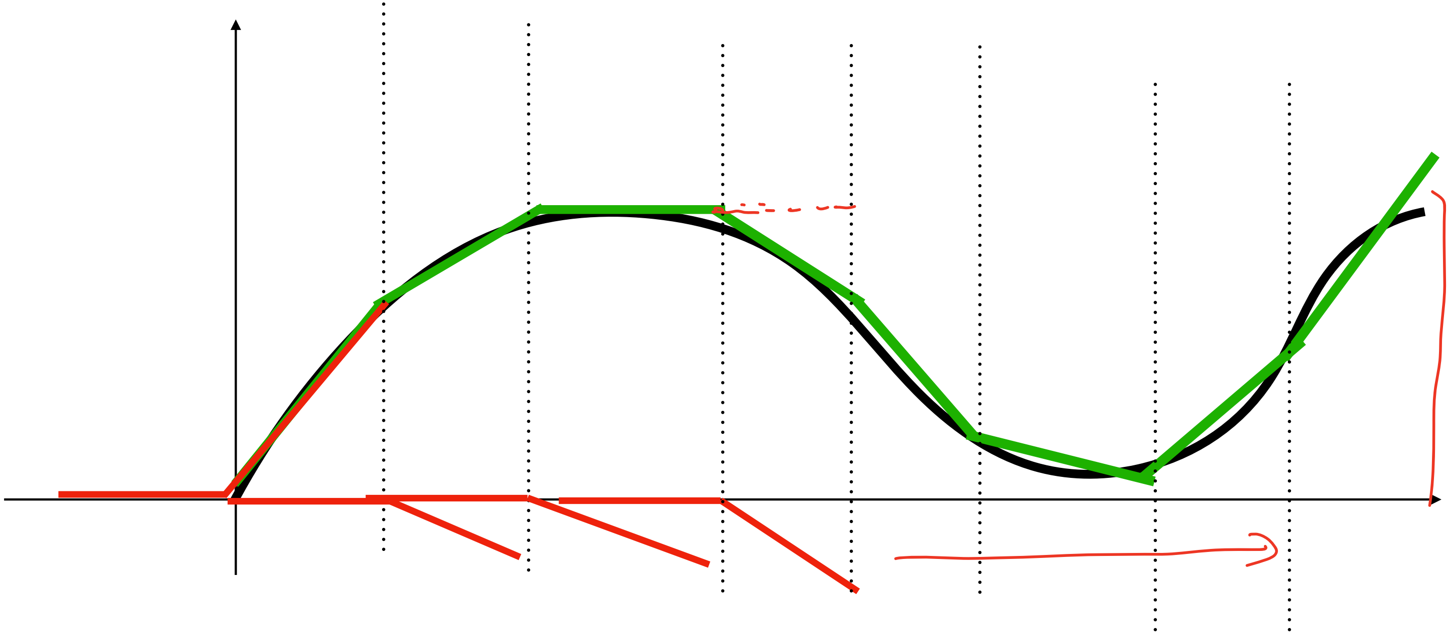
Claim: a wide enough one layer NN can approximate any smooth functions



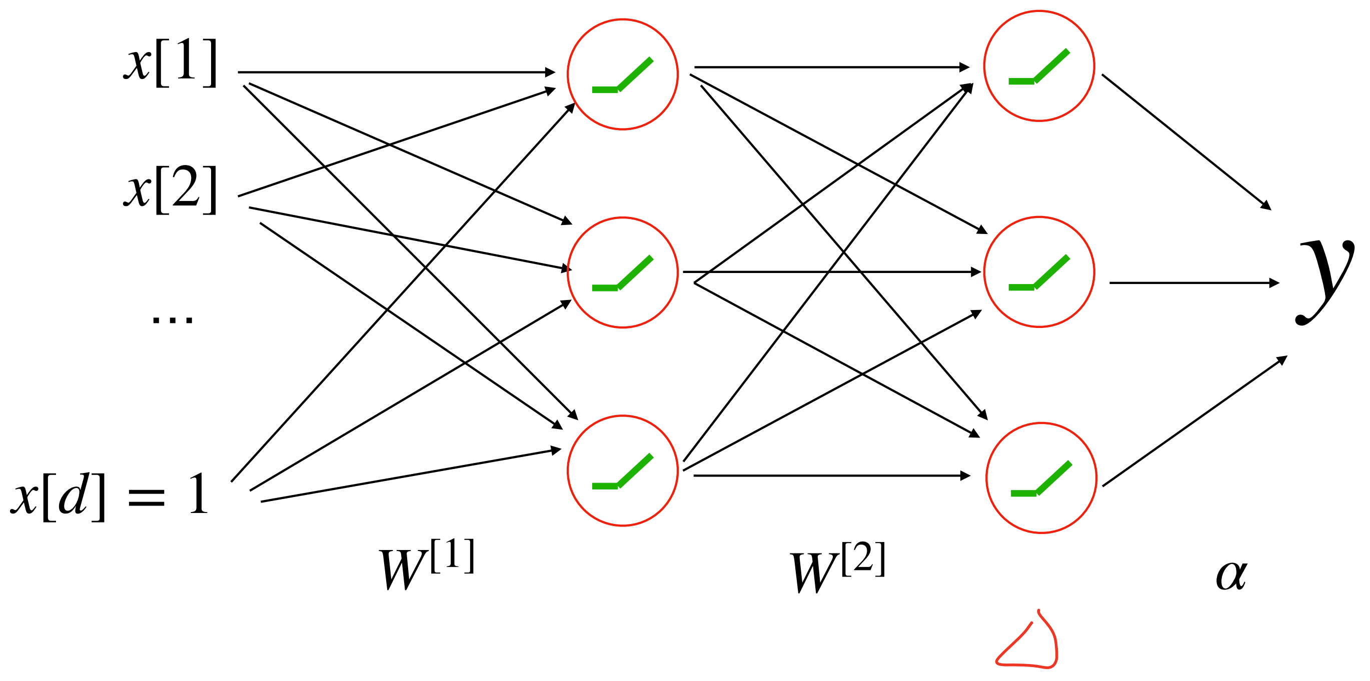
# What does a neural network approximate

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

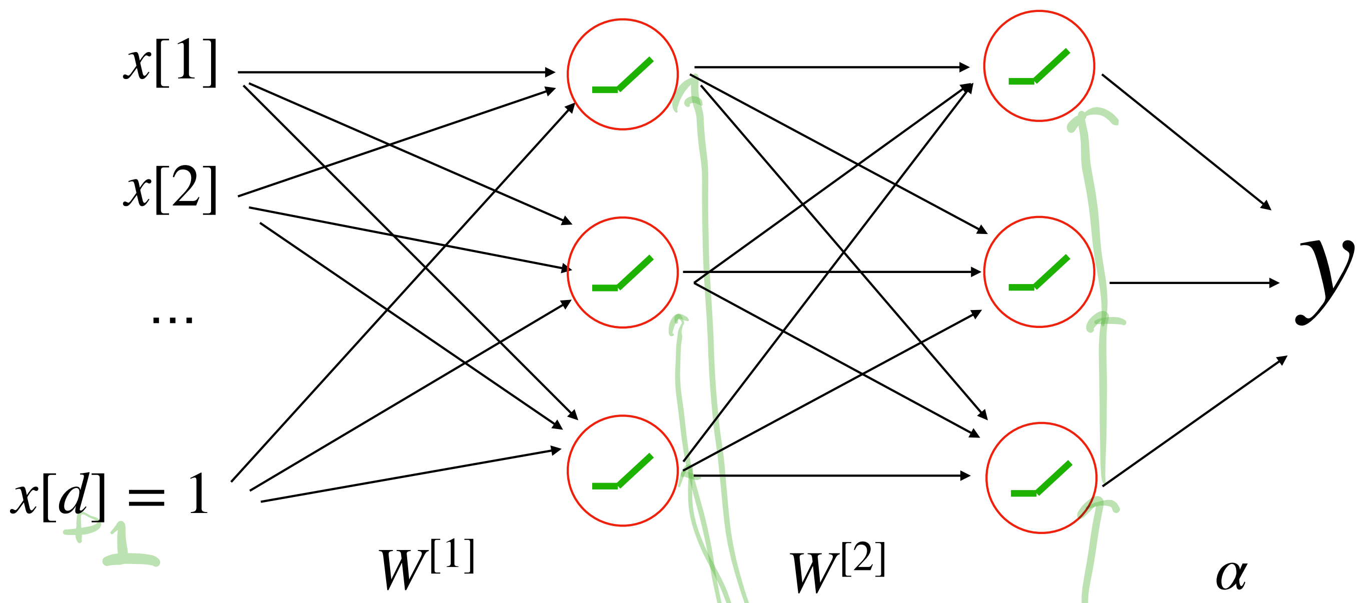
Claim: a wide enough one layer NN can approximate any smooth functions



# A multi-layer fully connected neural network

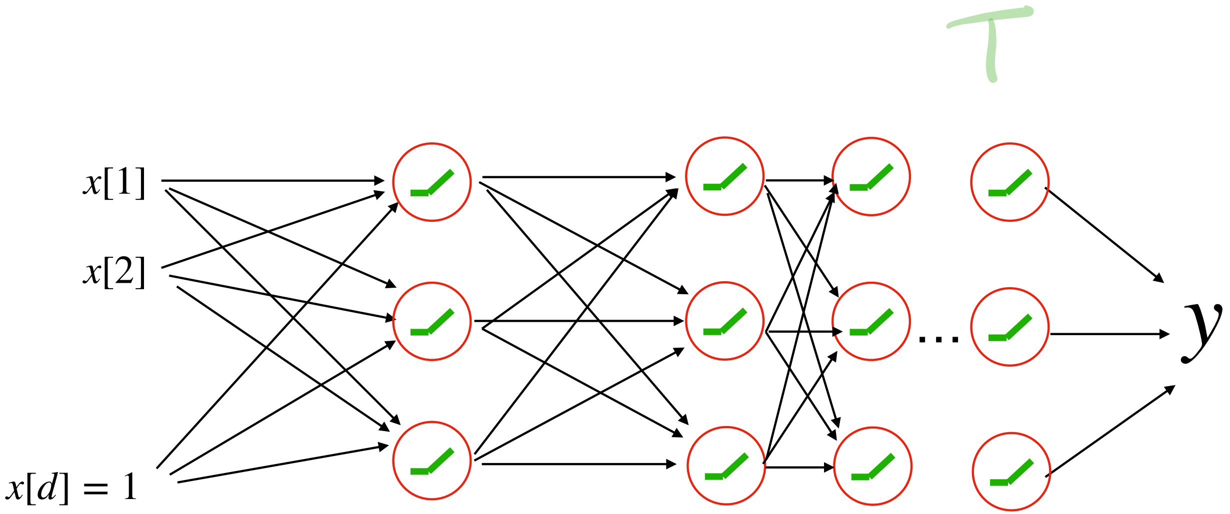


# A multi-layer fully connected neural network

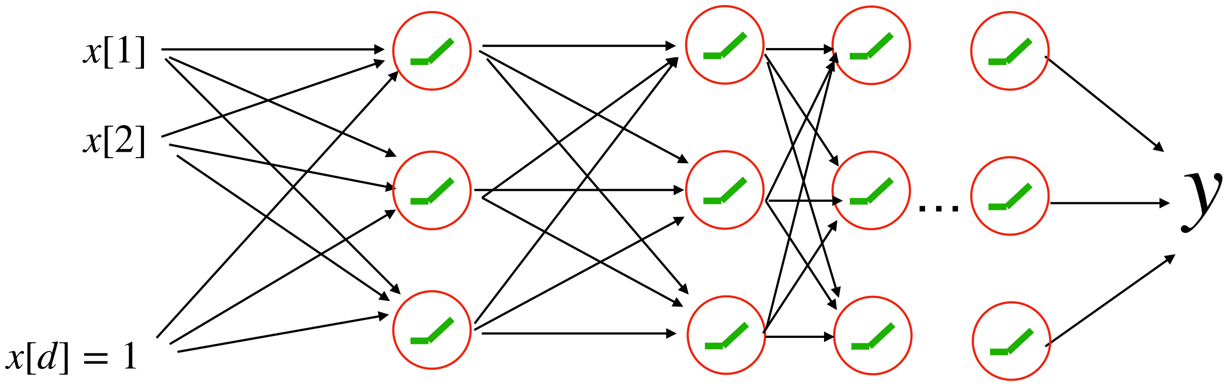



$$y = \alpha^{\top}\mathrm{ReLU}\left(W^{[2]}\mathrm{ReLU}\left(W^{[1]}x\right)\right) + b$$

# A multi-layer fully connected neural network

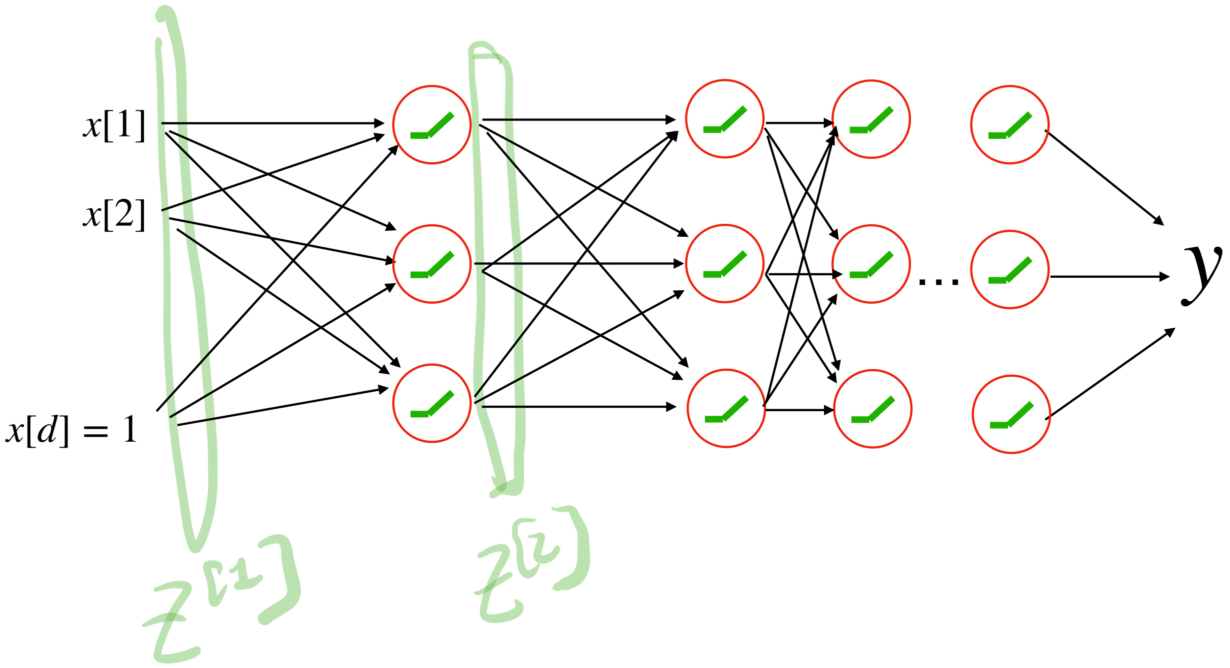


# A multi-layer fully connected neural network




Define it by a forward pass:

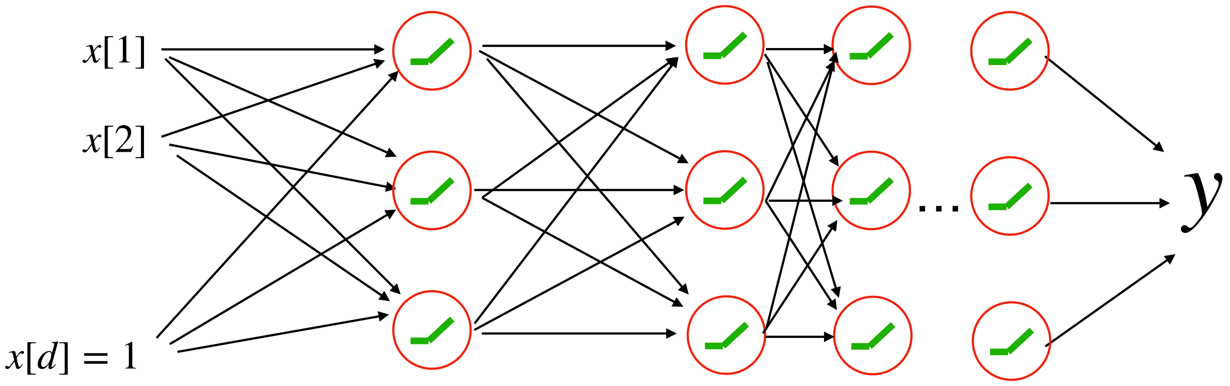
# A multi-layer fully connected neural network




Define it by a forward pass:

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

# A multi-layer fully connected neural network

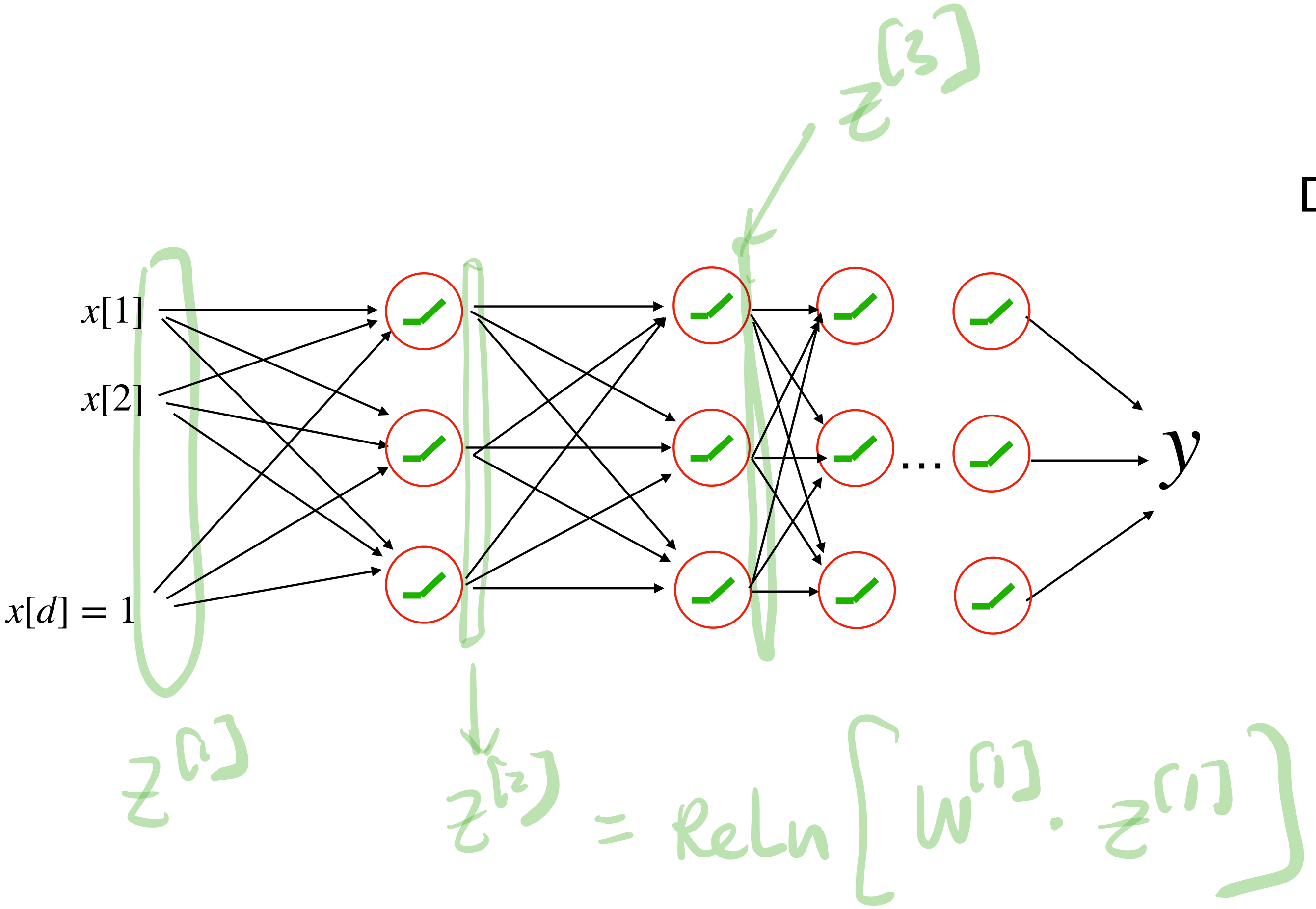



Define it by a forward pass:

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

For t = 1 to T-1:

# A multi-layer fully connected neural network



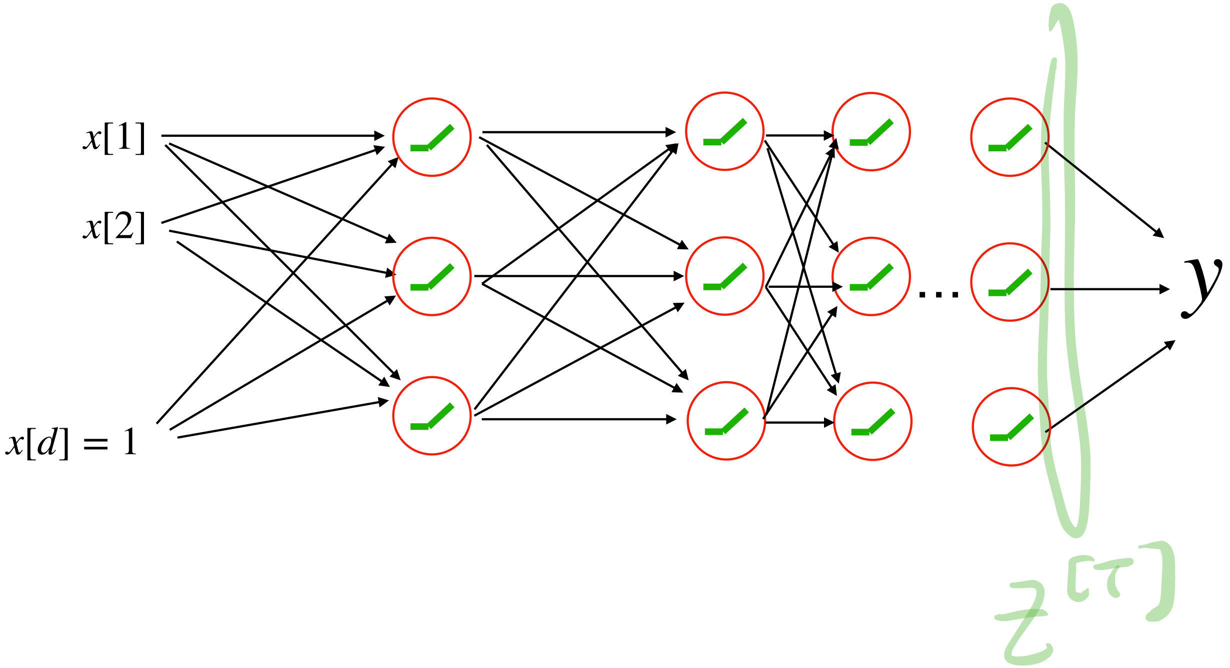

Define it by a forward pass:

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

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:

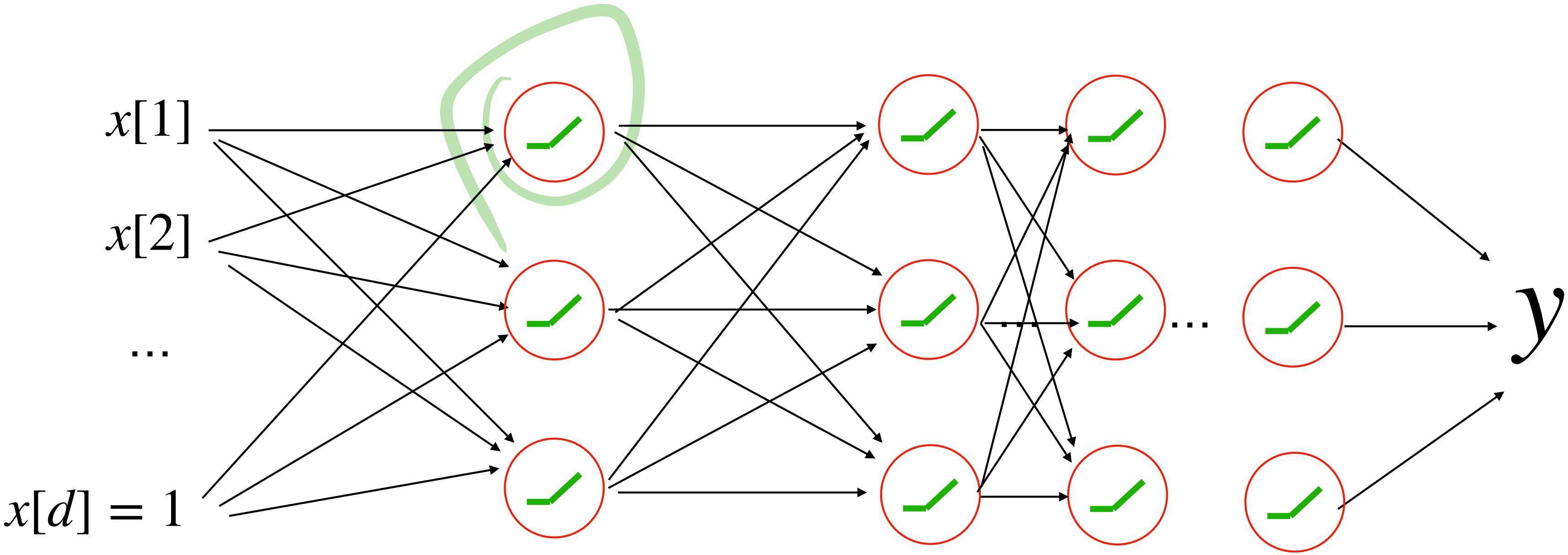
$$z^{[1]} = x$$

For t = 1 to T-1:

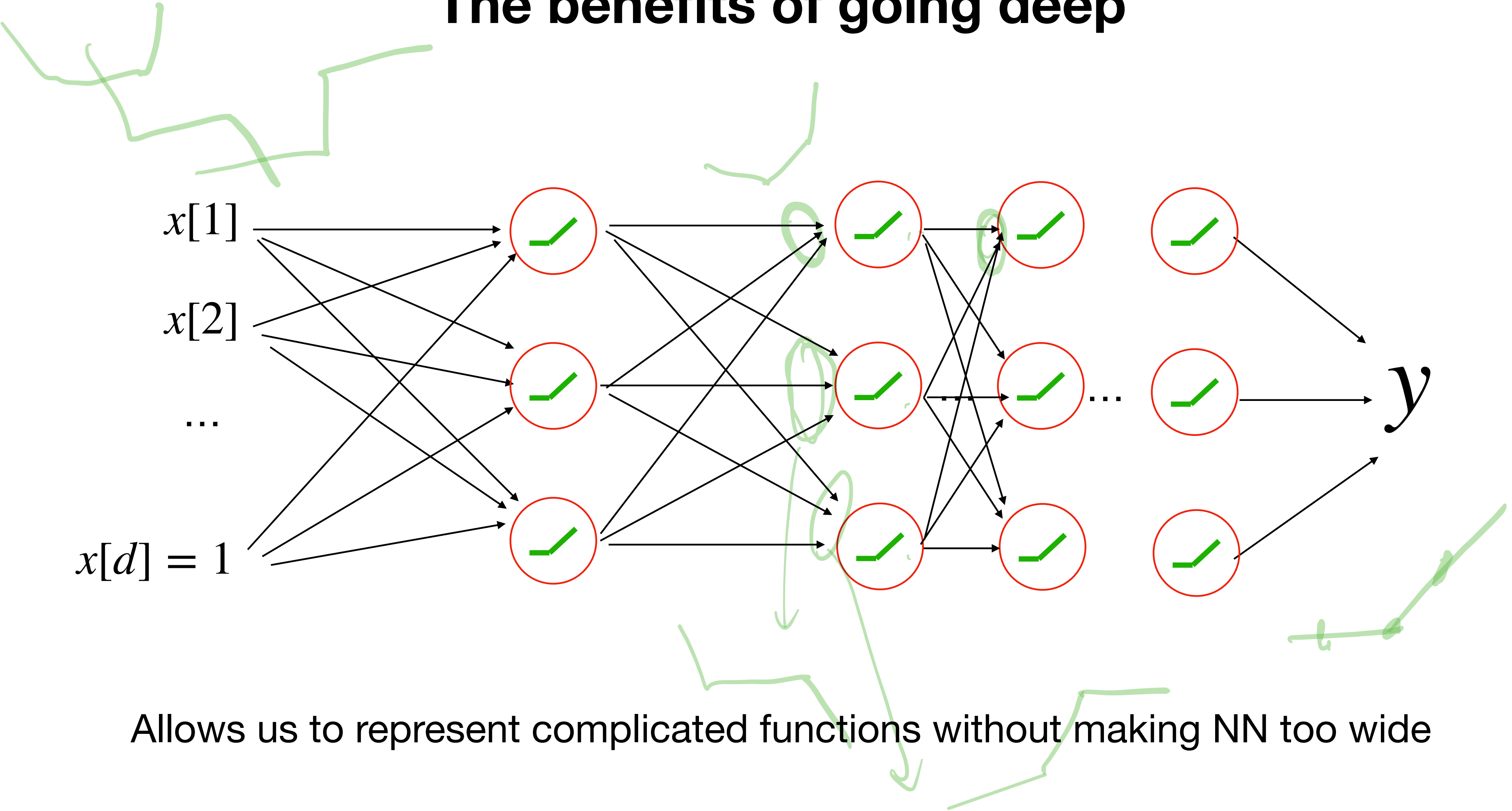
$$z^{[t+1]} = \text{ReLU}\left(W^{[t]} z^{t}\right)$$

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

# The benefits of going deep



# The benefits of going deep




Allows us to represent complicated functions without making NN too wide