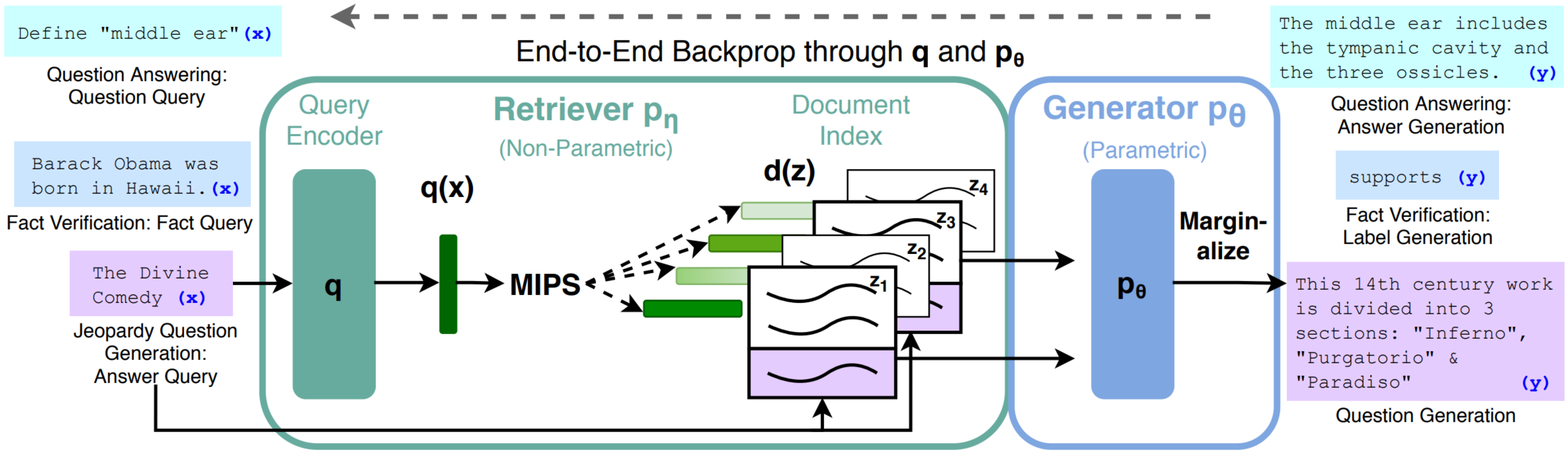
Announcements

- Quiz 4 is due **Monday, October 10 at 8pm**
  - Quiz 5 posted Today evening
- **Project Teams Update**
  - Most likely allowing teams of 2 (will confirm shortly)
  - **Let us know if you prefer to split up and be reassigned**
  - Similar expectations on novel contributions but proportionally less work
- **HW 3 Posted**
  - Due Wednesday, October 19 (2 weeks from today), lease start early!
  - Also, HW 2 late deadline is tonight at 8pm!

Application of KNN



https://ai.facebook.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/

Recap: Ensembles

- Meta-algorithms for combining models to improve their performance

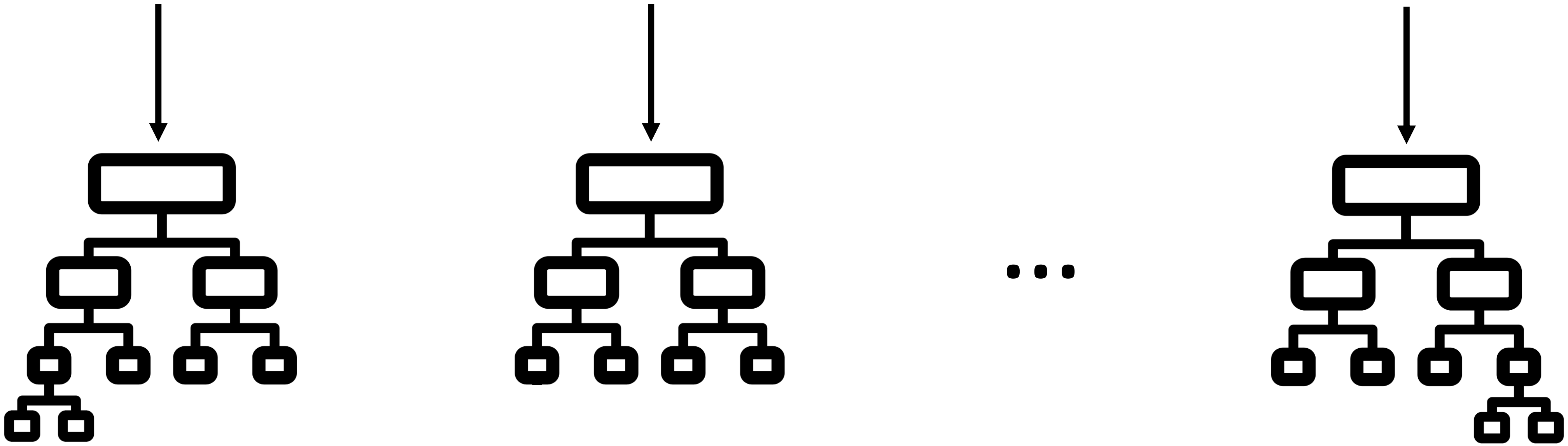
- For an ensemble learning algorithm, two design decisions:
  - How to learn base models?
  - How to combine learned base models?

Recap: Ensemble Design Decisions

- **How to learn the base models $f_1(x), \dots, f_k(x)$?**
  - **Intuition:** Need **diversity**
  - Handcrafted models
  - **Bagging:** Subsample examples and/or features
  - **Boosting:** Iteratively upweight currently incorrect examples

- **How to combine the learned base models?**
  - Average or majority vote
  - Learn a model $g_\beta\big(f_1(x), \dots, f_k(x)\big)$ treating $f_1(x), \dots, f_k(x)$ as “features”

Recap: Ensembles of Decision Trees



Recap: Random Forests

- **Ensemble strategy**
  - Bagging applied to unpruned decision trees
  - Randomly subsample $\sqrt{d}$ features at each split
  - Average random trees

- **Intuition**
  - Unpruned decision trees have high variance
  - Randomness enables us to “average away” excess variance
  - Cannot “overfit” by using too many trees

Recap: Boosting

- **Ensemble strategy**
  - Train depth-limited decision tree on weighted dataset
  - Iteratively upweight incorrectly classified examples

- **Intuition**
  - Depth-limited decision trees have high bias
  - Learning many models increases variance
  - Can overfit by learning too many trees (but often does not in practice)

Lecture 10: Ensembles (Part 2)

CIS 4190/5190

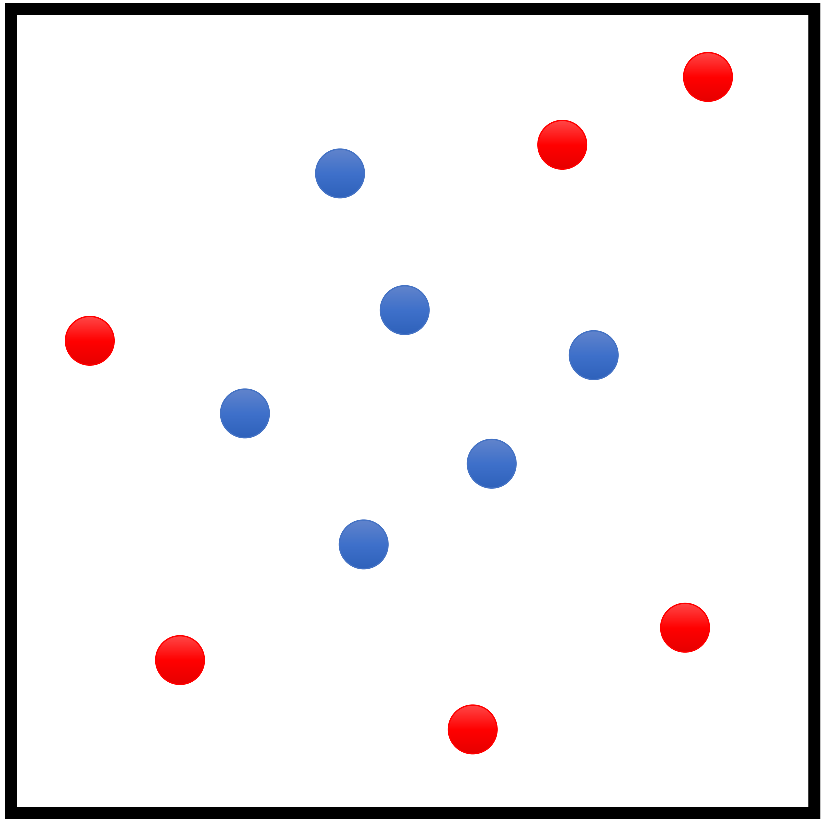
Fall 2022

AdaBoost (Freund & Schapire 1997)

- **Input**
  - Training dataset Z
  - Learning algorithm $\mathrm{Train}(Z, w)$ that can handle weights w
  - Hyperparameter T indicating number of models to train
- **Output**
  - Ensemble of models $F(x) = \sum_{t=1}^{T} \beta_t \cdot f_t(x)$

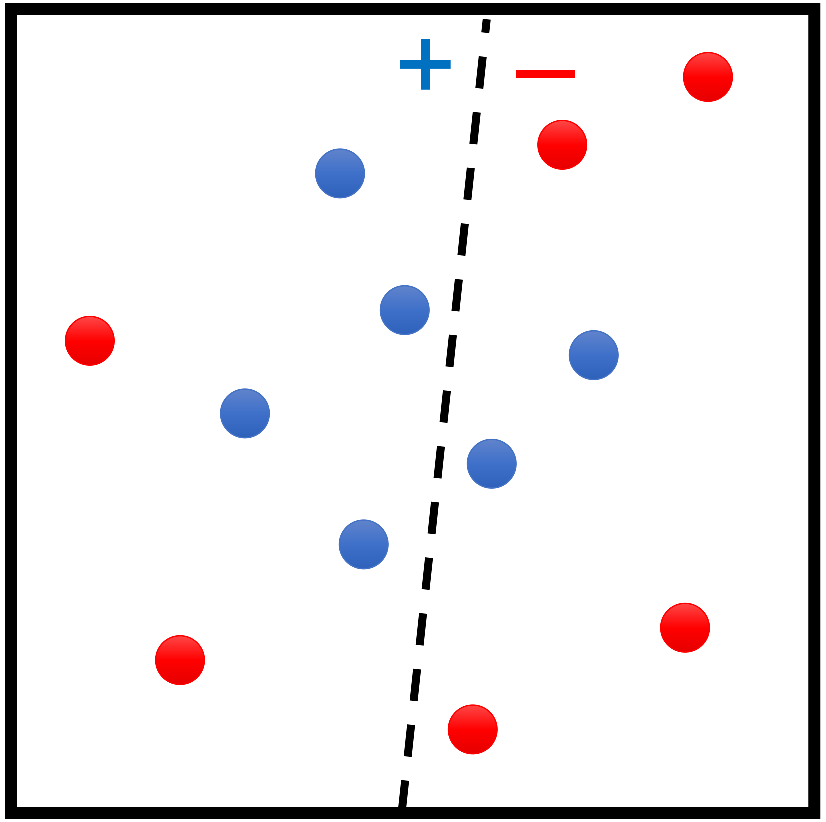
AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$



AdaBoost

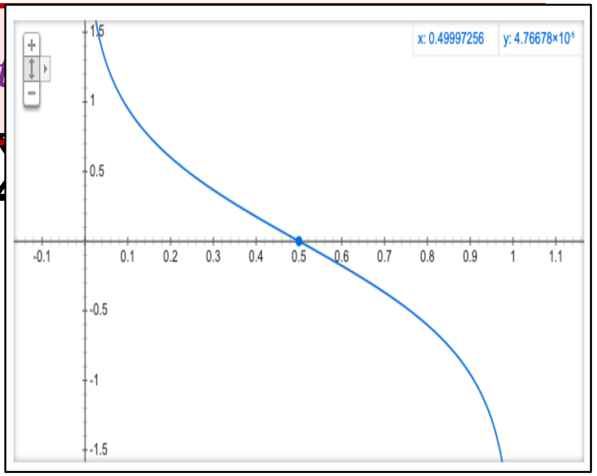
1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



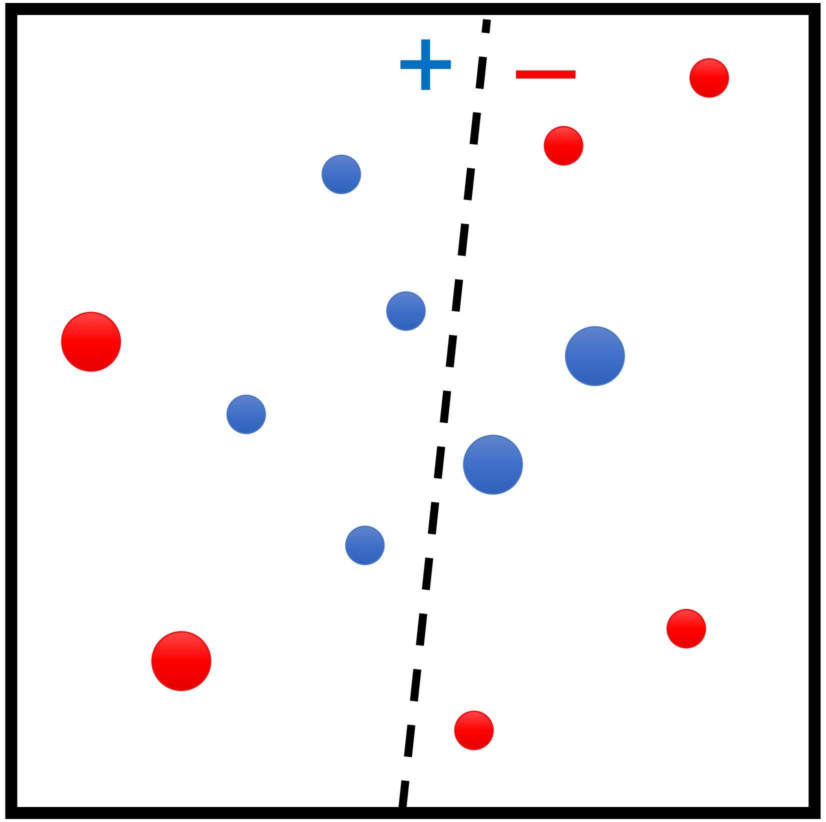
$t = 1$

AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t}$
7. **return** $F(x) = \text{sign}($



β_t becomes larger as
ϵ_t becomes smaller



$t = 1$

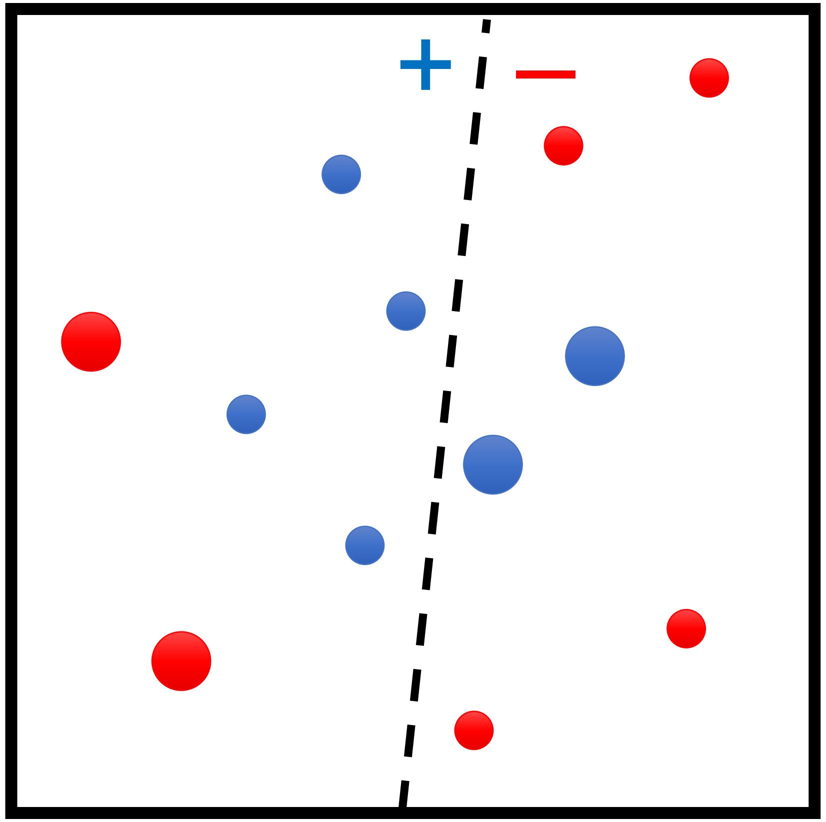
AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$

Use convention $y_i \in \{-1, +1\}$

If correct ($y_i = f_t(x_i)$) then multiply by $e^{-\beta_t}$

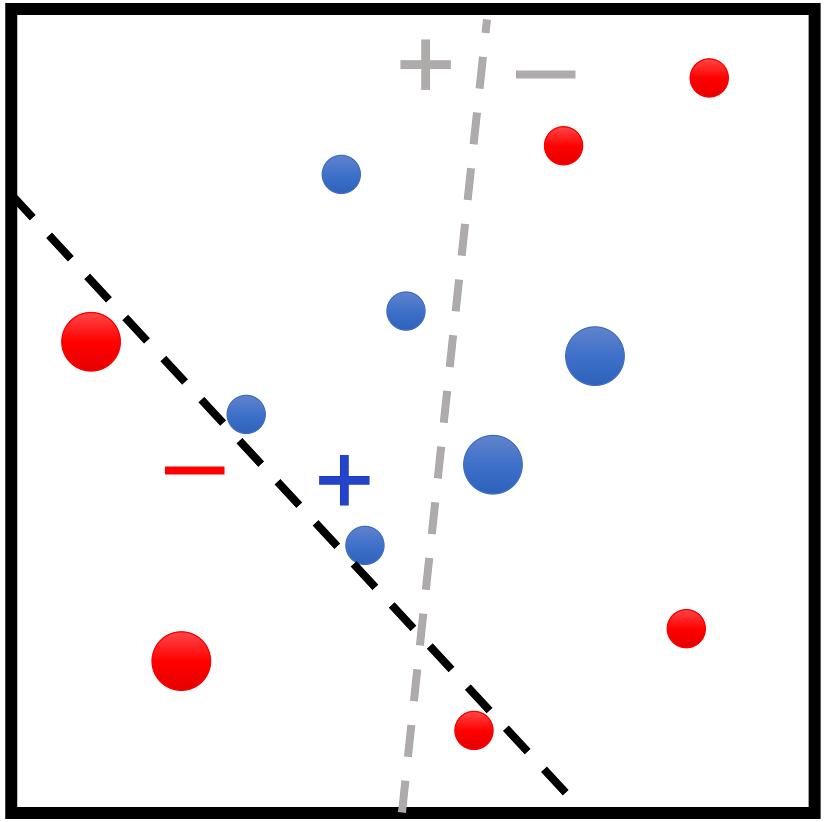
If incorrect ($y_i \neq f_t(x_i)$) then multiply by e^{β_t}



$t = 1$

AdaBoost

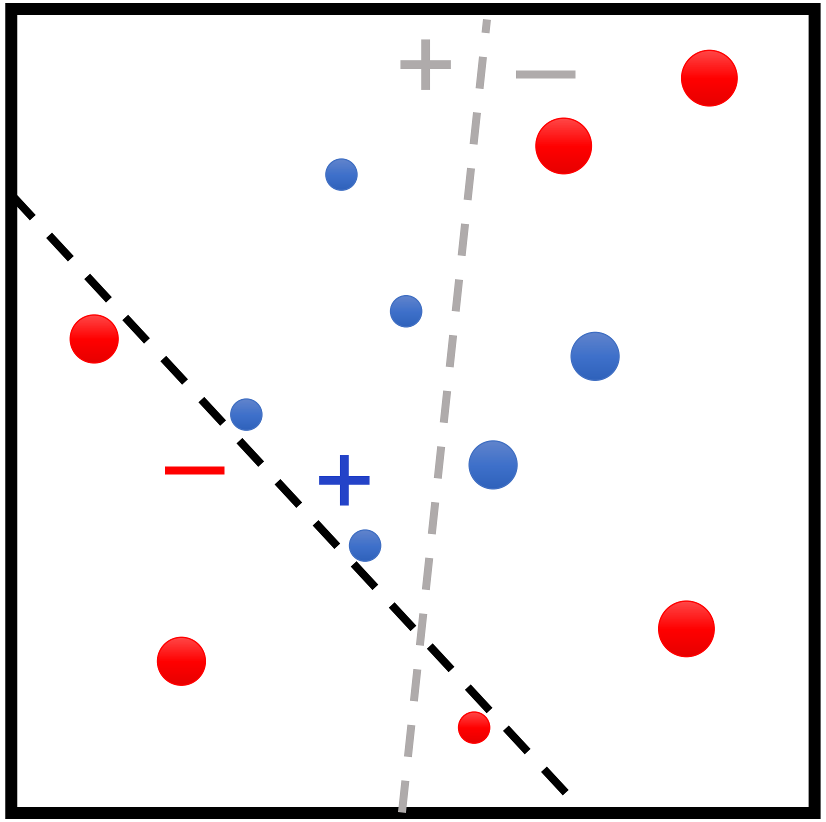
1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\boxed{f_t \leftarrow \mathrm{Train}(Z, w_t)}$
4. $\epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 2$

AdaBoost

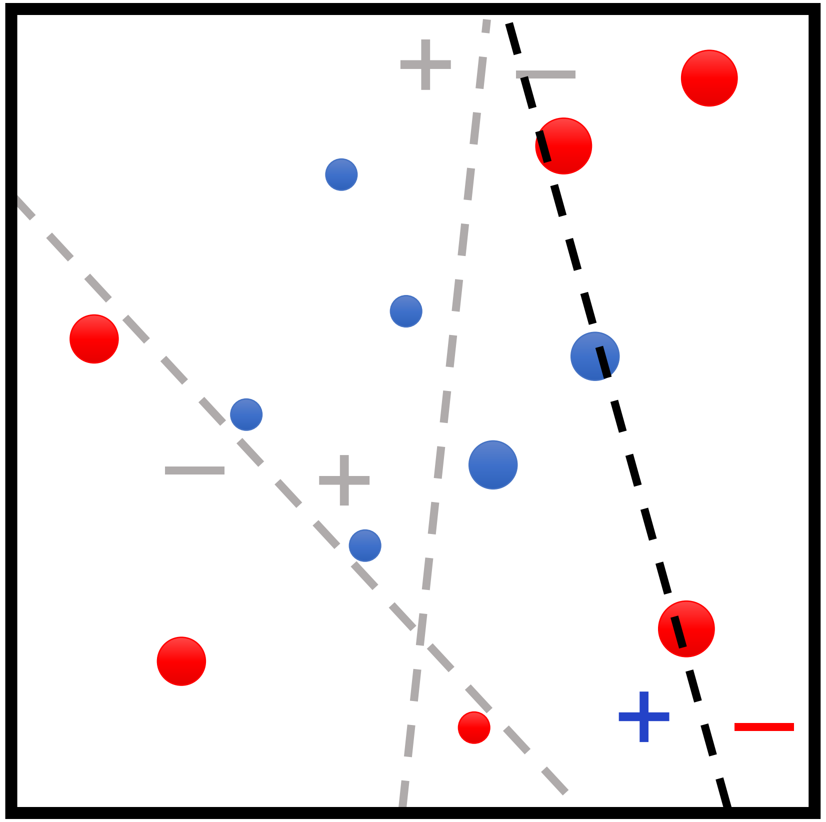
1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad \boxed{w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}}$ (for all i)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 2$

AdaBoost

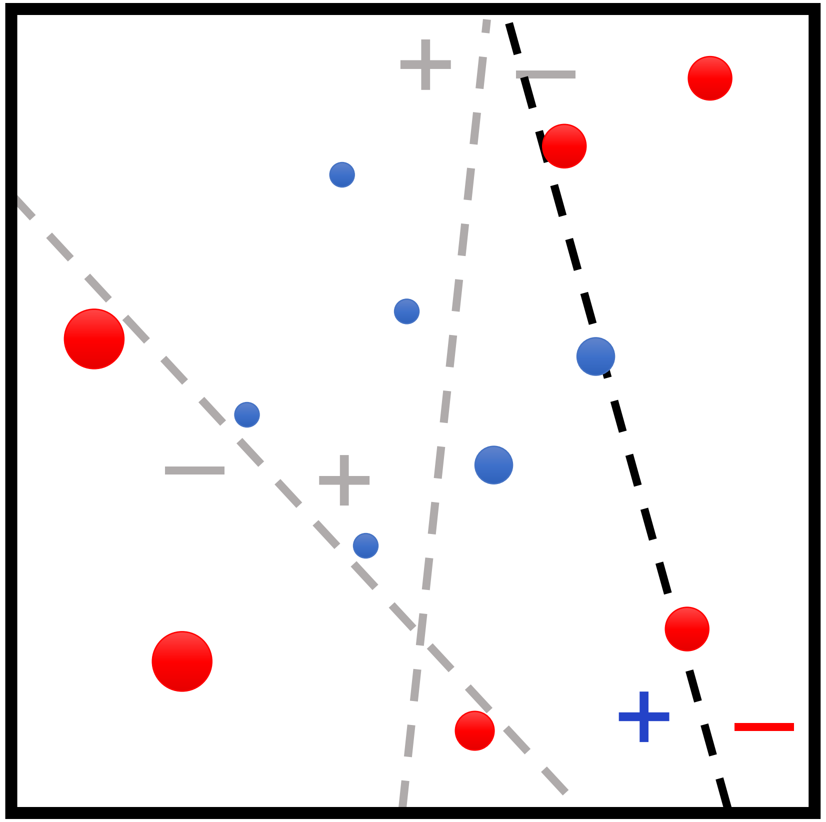
1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$



$t = 3$

AdaBoost

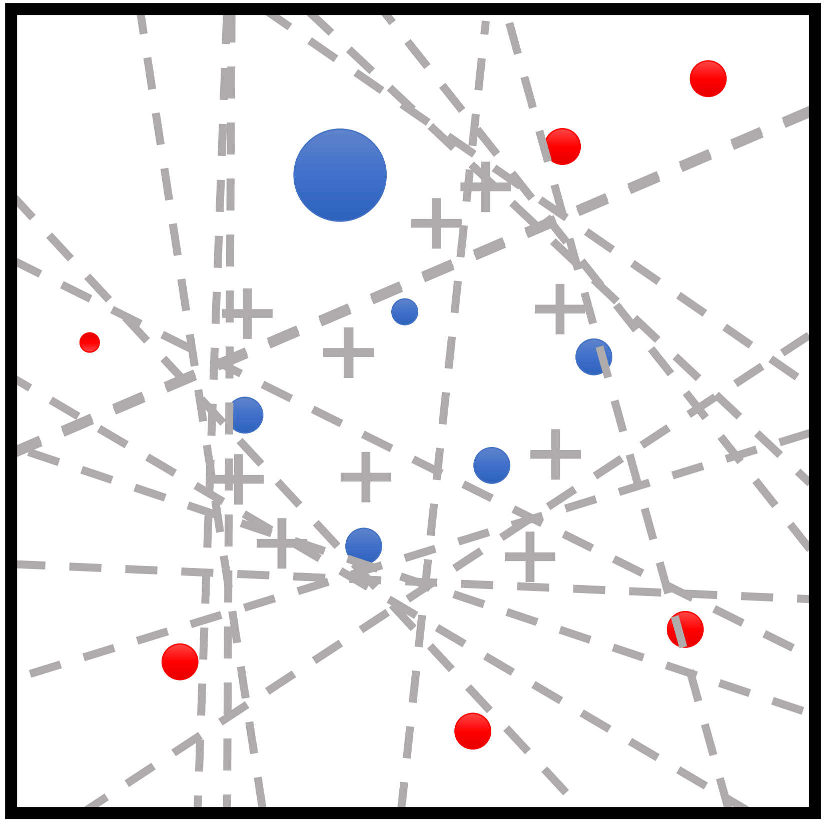
1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad \boxed{w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)} \text{ (for all } i\text{)}}$
7. **return** $F(x) = \mathrm{sign}\left(\sum_{t=1}^{T} \beta_t \cdot f_t(x)\right)$



$t = 3$

AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \dots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \dots, T\}$
3. $\quad f_t \leftarrow \text{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \text{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \text{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$

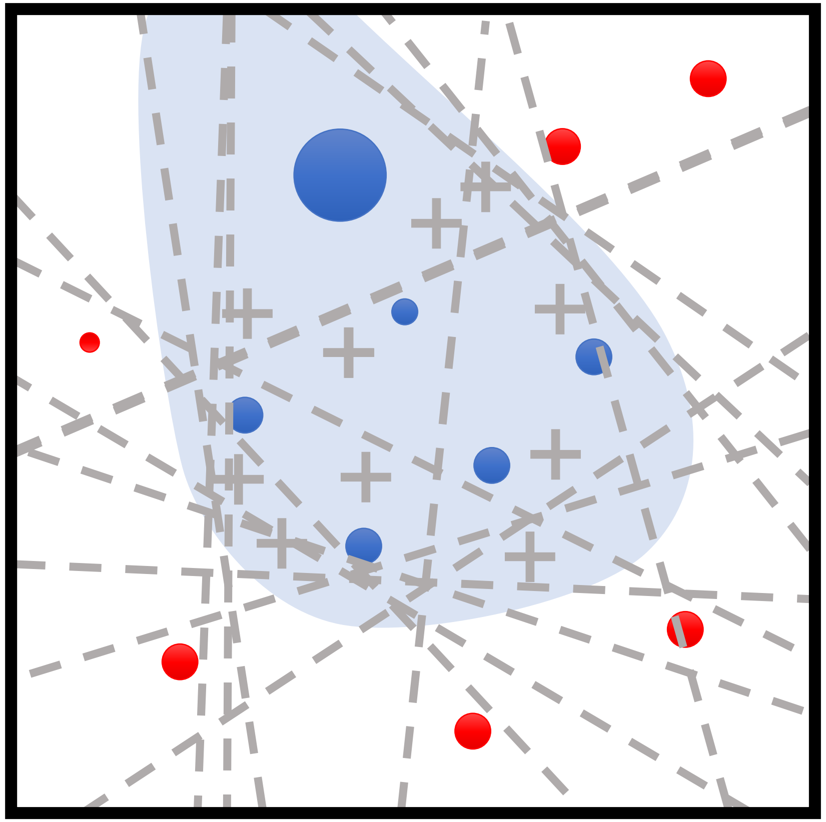


$t = T$

AdaBoost

1. $w_1 \leftarrow \left(\frac{1}{n}, \ldots, \frac{1}{n}\right)$ ($w_{1,i}$ weight for (x_i, y_i))
2. **for** $t \in \{1, \ldots, T\}$
3. $\quad f_t \leftarrow \mathrm{Train}(Z, w_t)$
4. $\quad \epsilon_t \leftarrow \mathrm{Error}(f_t, Z, w_t)$
5. $\quad \beta_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
6. $\quad w_{t+1,i} \propto w_{t,i} \cdot e^{-\beta_t \cdot y_i \cdot f_t(x_i)}$ (for all i)
7. **return** $F(x) = \mathrm{sign}(\sum_{t=1}^{T} \beta_t \cdot f_t(x))$

final model is average of base models weighted by their performance



AdaBoost Summary

- **Strengths:**
  - Fast and simple to implement
  - No hyperparameters (except for T, which is robust in practice)
  - Very few assumptions on base models (except they should be low capacity)
- **Weaknesses:**
  - Can perform poorly when there is insufficient data
  - No way to parallelize
  - **Specific to classification!**

Boosting as Gradient Descent

- Both algorithms: new model = old model + update
- **Gradient Descent:**

$$\theta_{t+1} = \theta_t - \alpha \cdot \nabla_\theta L(\theta_t; Z)$$

- **Boosting:**

$$F_{t+1}(x) = F_t(x) + \beta_{t+1} \cdot f_{t+1}(x)$$

- Here, $F_t(x) = \sum_{i=1}^{t} \beta_i \cdot f_i(x)$

Boosting as Gradient Descent

- Assuming $\beta_t = 1$ for all t, then:

$$F_t(x_i) + f_{t+1}(x_i) = F_{t+1}(x_i)$$

Boosting as Gradient Descent

- Assuming $\beta_t = 1$ for all t, then:

$$F_t(x_i) + f_{t+1}(x_i) = F_{t+1}(x_i) \approx y_i$$

- Rewriting this equation, we have

$$f_{t+1}(x_i) = F_{t+1}(x_i) - F_t(x_i) \approx \underbrace{y_i - F_t(x_i)}_{\text{“residuals”, i.e., error of the current model}}$$

Boosting as Gradient Descent

- In other words, at each step, boosting is training the next model f_{t+1} to approximate the residual:

$$f_{t+1}(x_i) \approx \underbrace{y_i - F_t(x_i)}_{\text{“residuals”, i.e., error of the current model}}$$

- **Idea:** Train f_{t+1} directly to predict residuals $y_i - F_t(x_i)$

- **This strategy works for regression as well!**

Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, \ldots, T\}$:
  - **Step 1:** Train f_{t+1} using dataset

$$Z_{t+1} = \{(x_i, y_i - F_t(x_i))\}_{i=1}^{n}$$

  - **Step 2:** Take

$$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model F_T

Boosting as Gradient Descent

- Consider losses of the form

$$L(F; Z) = \frac{1}{n} \sum_{i=1}^{n} \tilde{L}(F(x_i); y_i)$$

- In other words, sum of individual label-level losses $\tilde{L}(\hat{y}; y)$ of a prediction $\hat{y} = F(x)$ if the ground truth label is y
- For example, $\tilde{L}(\hat{y}; y) = \frac{1}{2}(y - \hat{y})^2$ yields the MSE loss

Boosting as Gradient Descent

- Residuals are the gradient of the squared error $\tilde{L}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$:

$$-\frac{\partial \tilde{L}}{\partial \hat{y}}(F_t(x_i); y_i) = y_i - F_t(x_i) = \mathrm{residual_i}$$

- For general $\tilde{L}$, instead of $\{(x_i, y_i - F_t(x_i))\}_{i=1}^{n}$ we can train f_{t+1} on

$$Z_{t+1} = \left\{\left(x_i, -\frac{\partial \tilde{L}}{\partial \hat{y}}(F_t(x_i); y_i)\right)\right\}_{i=1}^{n}$$

Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, \dots, T\}$:
  - **Step 1:** Train f_{t+1} using dataset

$$Z_{t+1} = \{(x_i, y_i - F_t(x_i))\}_{i=1}^{n}$$

  - **Step 2:** Take

$$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model F_T

Boosting as Gradient Descent

- **Algorithm:** For each $t \in \{1, \dots, T\}$:
  - **Step 1:** Train f_{t+1} using dataset

$$Z_{t+1} = \left\{ \left(x_i, -\frac{\partial \tilde{L}}{\partial \hat{y}} \left(F_t(x_i); y_i \right) \right) \right\}_{i=1}^{n}$$

  - **Step 2:** Take

$$F_{t+1}(x) = F_t(x) + f_{t+1}(x)$$

- Return the final model F_T

Boosting as Gradient Descent

- Casts ensemble learning in the **loss minimization framework**
  - **Model family:** Sum of base models $F_T(x) = \sum_{t=1}^{T} f_t(x)$
  - **Loss:** Any differentiable loss expressed as

$$L(F; Z) = \sum_{i=1}^{n} \tilde{L}(F(x_i), y_i)$$

- Gradient boosting is a general paradigm for training ensembles with specialized losses (e.g., most NLL losses)

Gradient Boosting in Practice

- Gradient boosting with depth-limited decision trees (e.g., depth 3) is one of the most powerful off-the-shelf classifiers available
  - **Caveat:** Inherits decision tree hyperparameters

- XGBoost is a very efficient implementation suitable for production use
  - A popular library for gradient boosted decision trees
  - Optimized for computational efficiency of training and testing
  - Used in many competition winning entries, across many domains
  - https://xgboost.readthedocs.io

Lecture 11: Neural Networks (Part 1)

CIS 4190/5190

Fall 2022

Model Family for Neural Networks

- **Modern view:** Not a single model family

- Instead, a **flexible framework** for **designing** model families

Simple Example of Model Family

- **Feedforward neural network model family (for regression):**

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

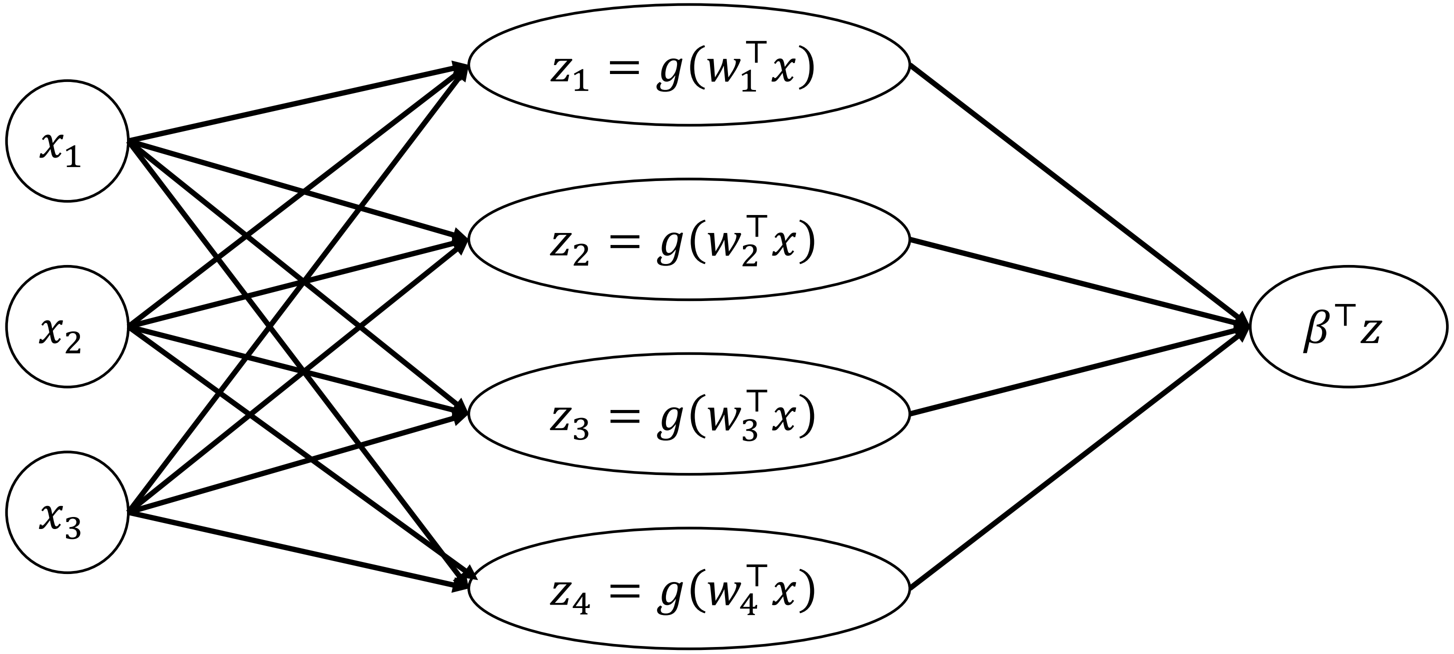
- **Parameters:** Matrix $W \in \mathbb{R}^{d \times k}$ and vector $\beta \in \mathbb{R}^k$
  - k is a hyperparameter called the **number of hidden neurons**

- Here, $g: \mathbb{R} \to \mathbb{R}$ is a given **activation function**
  - It is applied componentwise in $f_{W,\beta}$ (i.e., $g\left(\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}\right) = \begin{bmatrix} g(z_1) \\ g(z_2) \end{bmatrix}$)
  - **Example:** $g(z) = \sigma(z)$ (where σ is the sigmoid function)

Simple Example of Model Family

- **Feedforward neural network model family (for regression):**

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$



Simple Example of Model Family

- **Feedforward neural network model family (for regression):**

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$

- What happens if g is linear? Recovers linear functions!

$$f_{W,\beta}(x) = \beta^\top g(Wx) = \beta^\top Wx = \tilde{\beta}^\top x$$

- **In general:** Linear regression over “features” $g(Wx)$

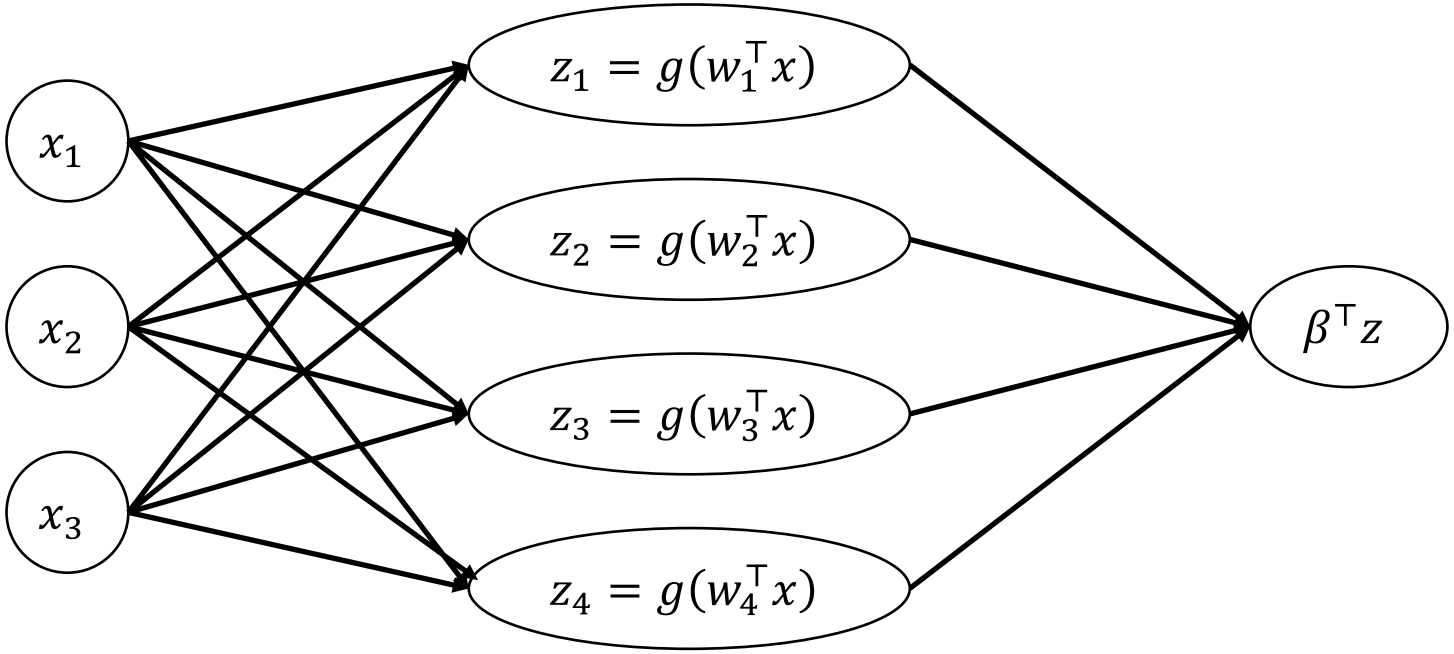
Modern View

- Not a single model family
- Instead, a **flexible framework** for **designing** model families

Modern View

- **Feedforward neural network model family:**

$$f_{W,\beta}(x) = \beta^\top g(Wx)$$



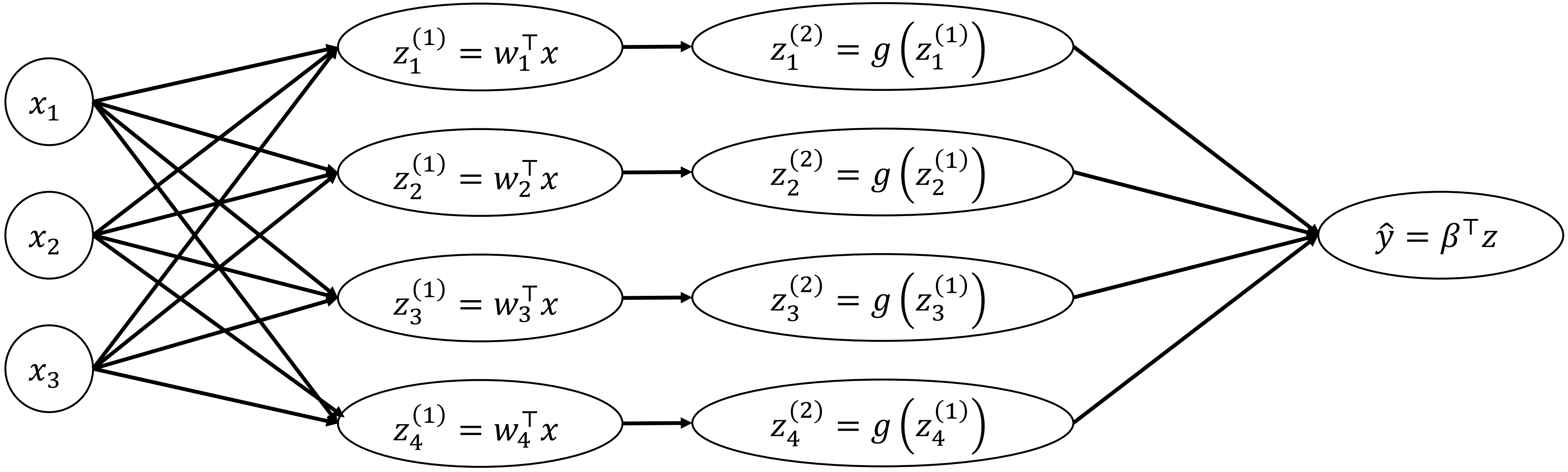
Modern View

Function composition:

$$f \circ g(x) = f\big(g(x)\big)$$

- **Feedforward neural network model family:**

$$f_{W,\beta}(x) = f_\beta\Big(g\big(f_W(x)\big)\Big) = f_\beta \circ g \circ f_W(x)$$



Modern View

- Each **layer** is a parametric function $f_{W_j}\colon \mathbb{R}^k \to R^h$

- Compose sequentially to form model family:

$$f_W = f_{W_m} \circ \cdots \circ f_{W_1}$$

- Equivalently:

$$f_W(x) = f_{W_m}\left(\dots\left(f_{W_1}(x)\right)\dots\right)$$

Modern View

- Each **layer** is a parametric function $f_{W_j}: \mathbb{R}^k \to R^h$

- Can compose layers in other was, e.g., concatenation:

$$f_W(x) = f_{W_1}(x) \oplus f_{W_2}(x)$$

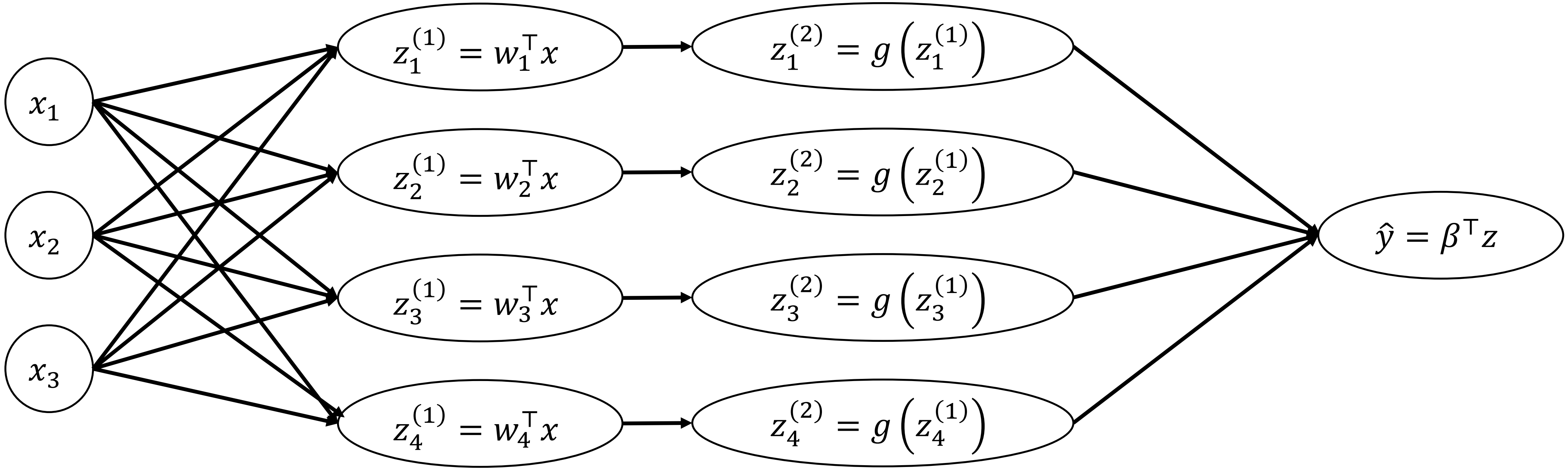
- Here, we have defined

$$[z_1 \quad \cdots \quad z_d]^\top \oplus \left[z_1' \quad \cdots \quad z_{d'}'\right]^\top = \left[z_1 \quad \cdots \quad z_d \quad z_1' \quad \cdots \quad z_{d'}'\right]^\top$$

Modern View

- **Feedforward neural network model family (for regression):**

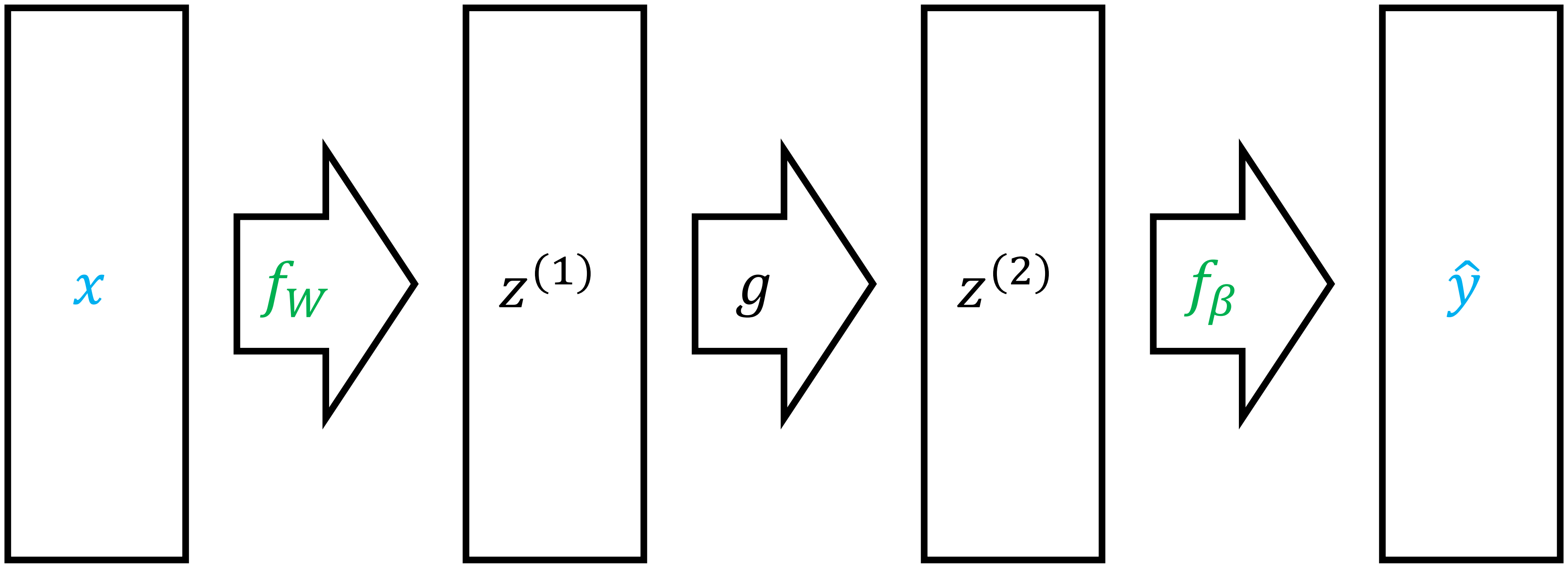
$$f_{W,\beta}(x) = f_\beta \circ g \circ f_W(x)$$



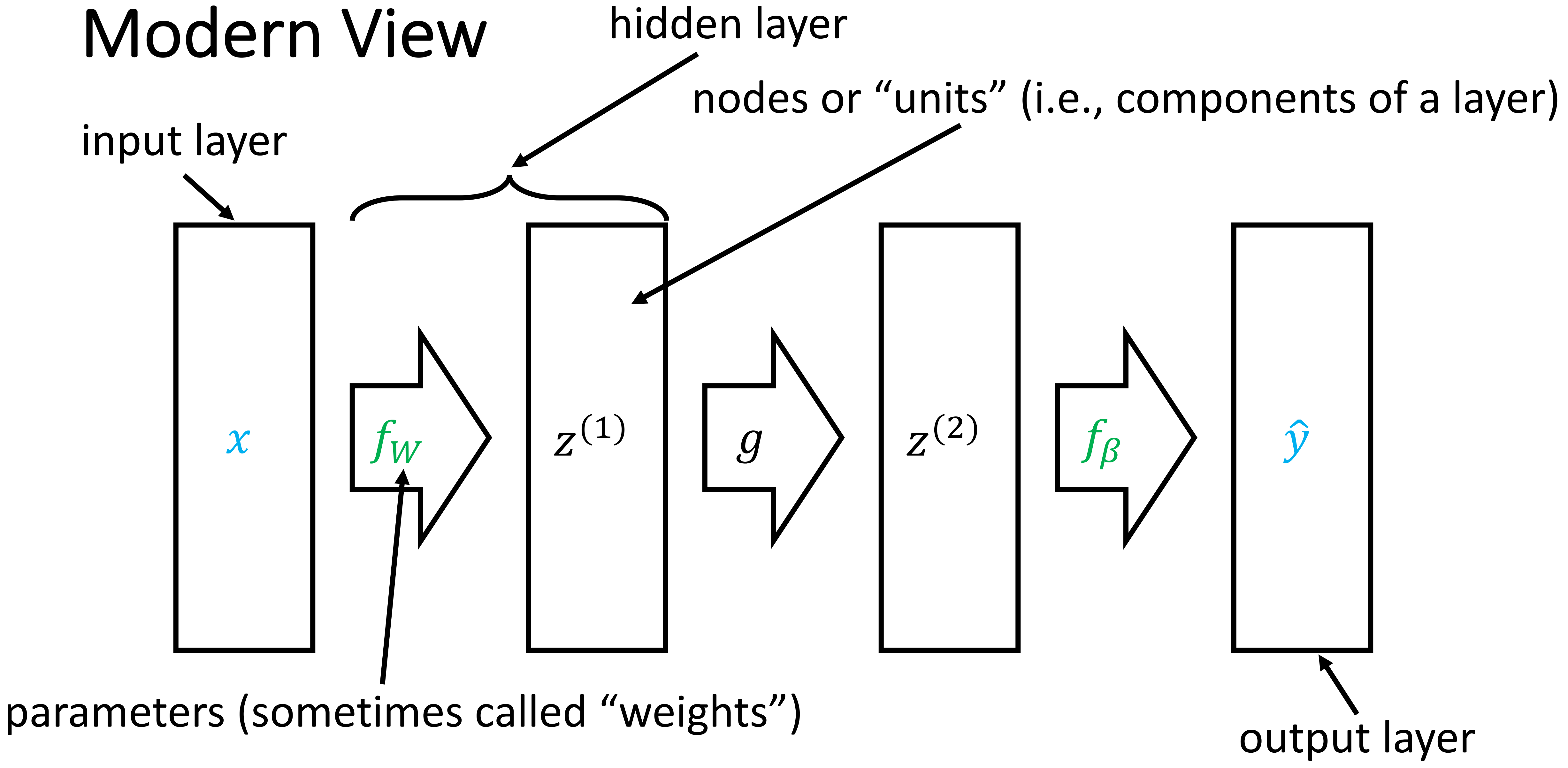
Modern View

- **Feedforward neural network model family (for regression):**

$$f_{W,\beta}(x) = f_\beta \circ g \circ f_W(x)$$



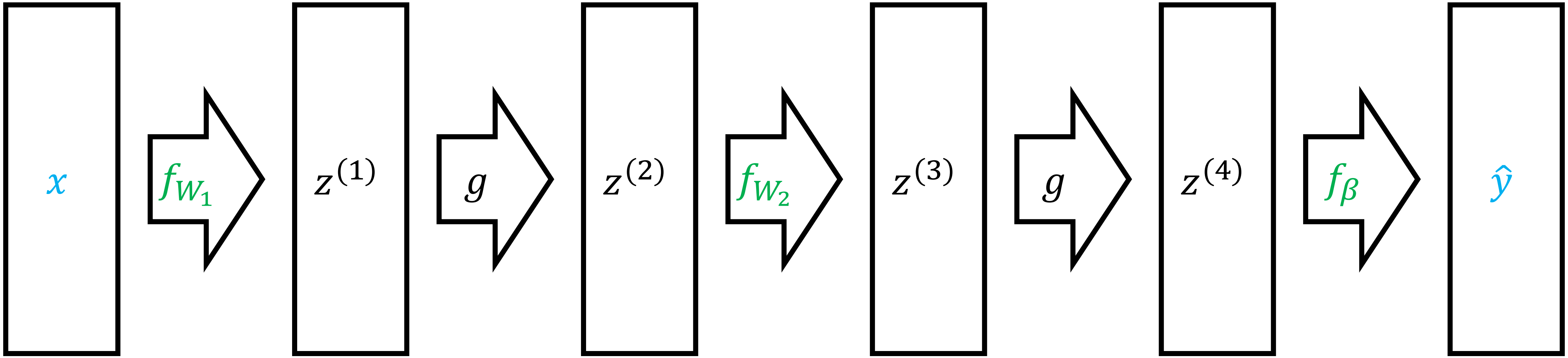
Modern View


hidden layer
nodes or "units" (i.e., components of a layer)
input layer
x
f_W
$z^{(1)}$
g
$z^{(2)}$
f_β
$\hat{y}$
parameters (sometimes called "weights")
output layer

Modern View

- **Neural network with two hidden linear layers:**

$$f_{W_1,W_2,\beta}(x) = f_\beta \circ g \circ f_{W_2} \circ g \circ f_{W_1}(x)$$



Modern View

- **Neural network with two hidden linear layers:**

$$f_{W_1,W_2,\beta}(x) = f_\beta\left(g\left(f_{W_2}\left(g\left(f_{W_1}(x)\right)\right)\right)\right)$$

$$x \xrightarrow{f_{W_1}} z^{(1)} \xrightarrow{g} z^{(2)} \xrightarrow{f_{W_2}} z^{(3)} \xrightarrow{g} z^{(4)} \xrightarrow{f_\beta} \hat{y}$$

Learn successively more “high-level” representations

What About Classification?

- **Recall:** For logistic regression, we choose the likelihood to be

$$p_\beta(Y = 1 \mid x) = \frac{1}{1 + e^{-\beta^\top x}}$$

What About Classification?

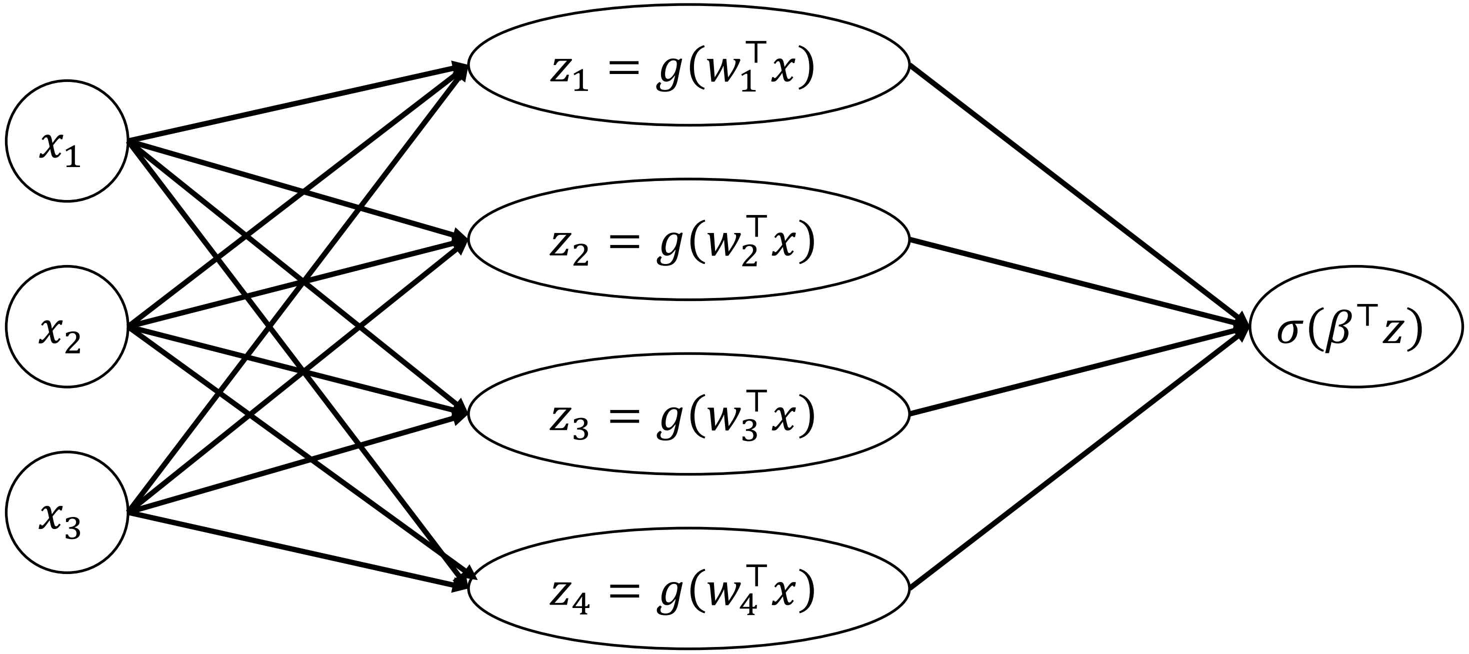
- **Recall:** For logistic regression, we choose the likelihood to be

$$p_\beta(Y = 1 \mid x) = \sigma(\beta^\top x)$$

What About Classification?

- **For binary classification:**

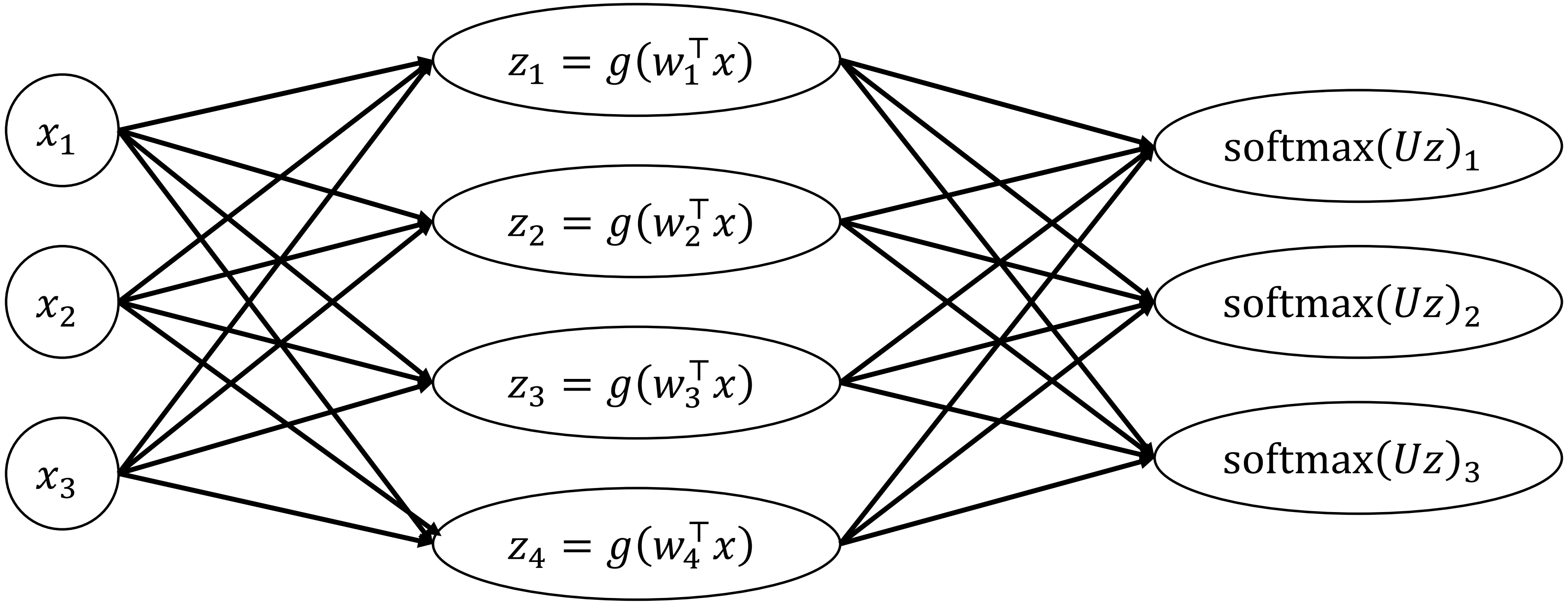
$$p_{W,\beta}(Y = 1 \mid x) = \sigma\left(\beta^\top g(Wx)\right)$$



What About Classification?

- **For multi-class classification:**

$$p_{W,U}(Y = y \mid x) = \text{softmax}\big(Ug(Wx)\big)_y$$



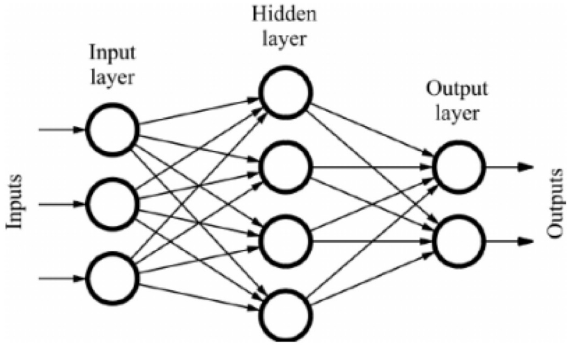
Neural Networks

- **Pros**
  - **“Meta” strategy:** Enables users to **design** model family
  - Design model families that capture **symmetries/structure** in the data (e.g., read a sentence forwards, translation invariance for images, etc.)
  - “Representation learning” (automatically learn features for certain domains)
  - More parameters!

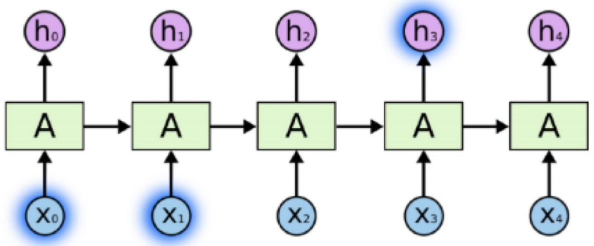
- **Cons**
  - Very hard to train! (Non-convex loss functions)
  - Lots of parameters → need lots of data!
  - Lots of design decisions

Common Architectures

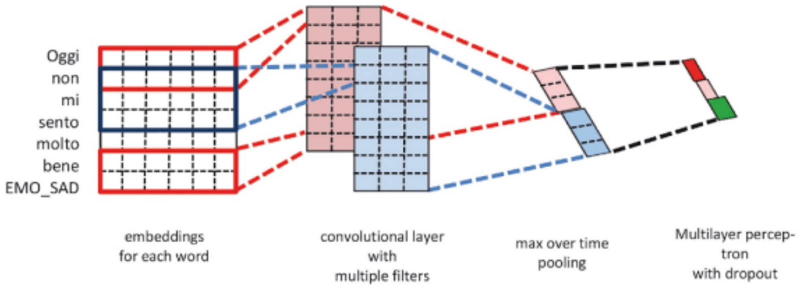
Feed-forward NNs



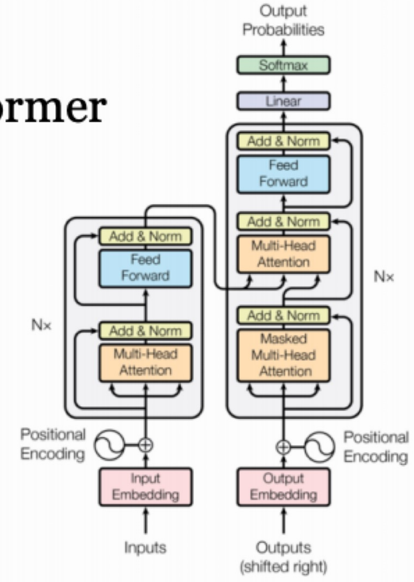
Recurrent NNs



Convolutional NNs



Transformer



Always coupled with word embeddings…