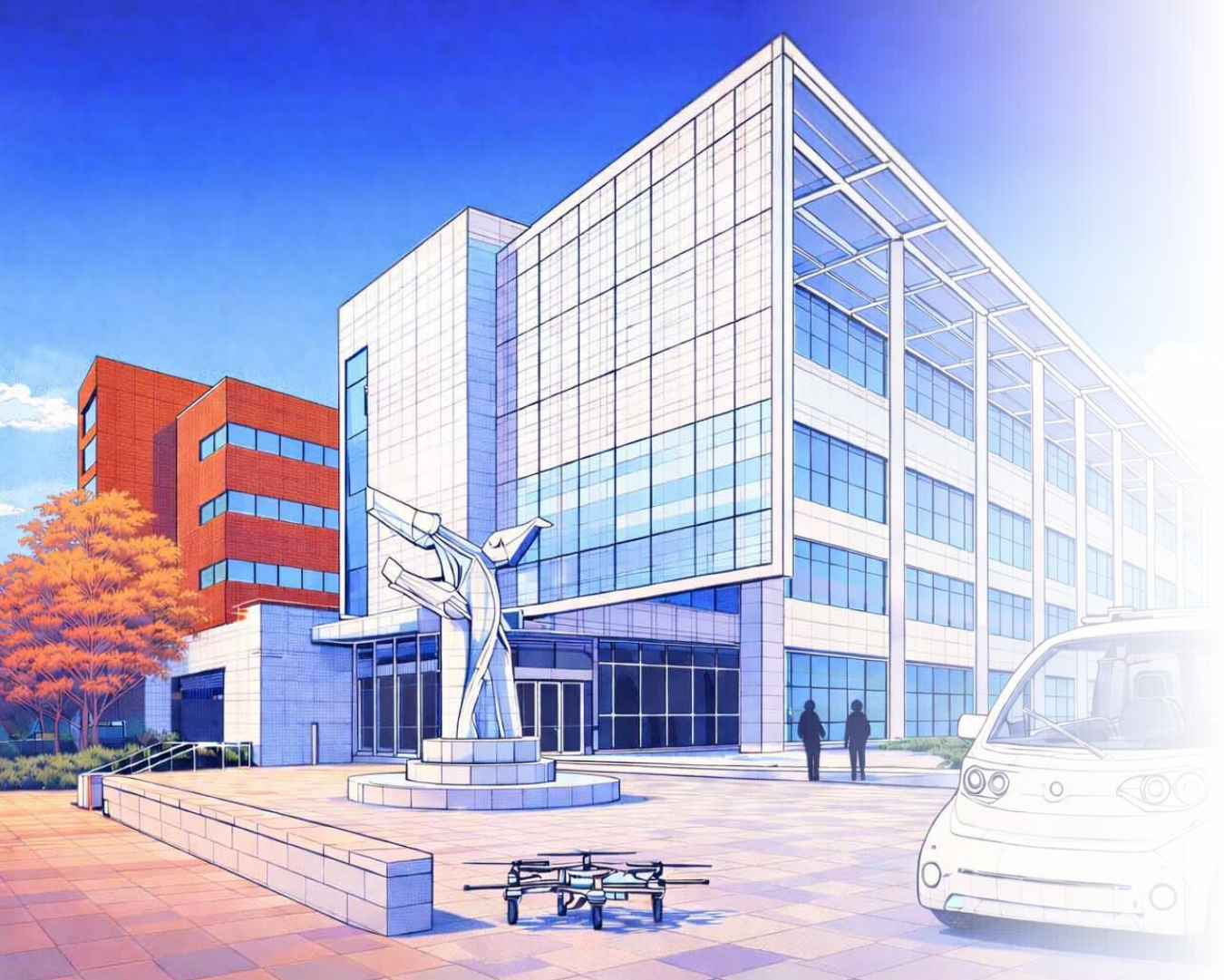


ECE 484
Lecture 5
Perception: Neural Networks

Sayan Mitra

Training

Training Images

Training Labels

Image Features

Training

Learned model

Testing

Test Image

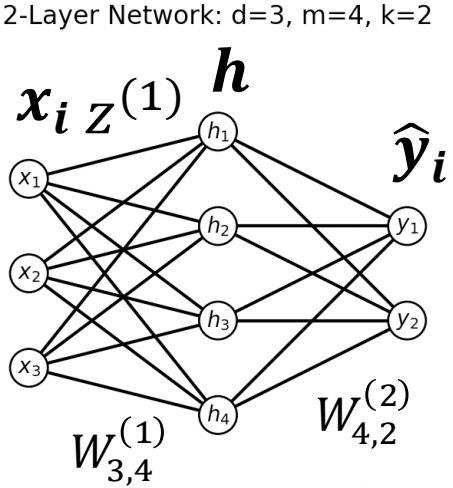
Image Features

Learned model

Prediction

Slide credit: D. Hoiem

Neural Network Forward Propagation



Input $\boldsymbol{x_i}$

Hidden layer **pre-activation**: $\boldsymbol{z}^{(1)} = W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}$.

Hidden layer output: $\boldsymbol{h} = g(\boldsymbol{z}^{(1)})$ g: activation function e.g. $ReLU(.)$; $sigmo$

Output: $\hat{\boldsymbol{y}}_i = W^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$ prediction $\hat{\boldsymbol{y}}_i$ for input $\boldsymbol{x_i}$

Loss function: A scalar function $L(W, \{y_i, x_i, \hat{y}_i\})$ that measures how well a prediction $\hat{\boldsymbol{y}}_i$ matches the ground truth $\boldsymbol{y_i}$.

E.g For pose regression, L2 distance: $L = -\frac{1}{N}\sum_{i=1}^{N} |\hat{\boldsymbol{y}}_i - \boldsymbol{y_i}|_2^2$

Evaluating loss involves computing $\hat{y}_i = f_{NN}(x_i)$ forward pass of NN

Training: algorithm/process of minimizing loss $L(W)$ by changing the weights (W) and the biases (b) of f_{NN}

E.g. gradient descent with back propagation

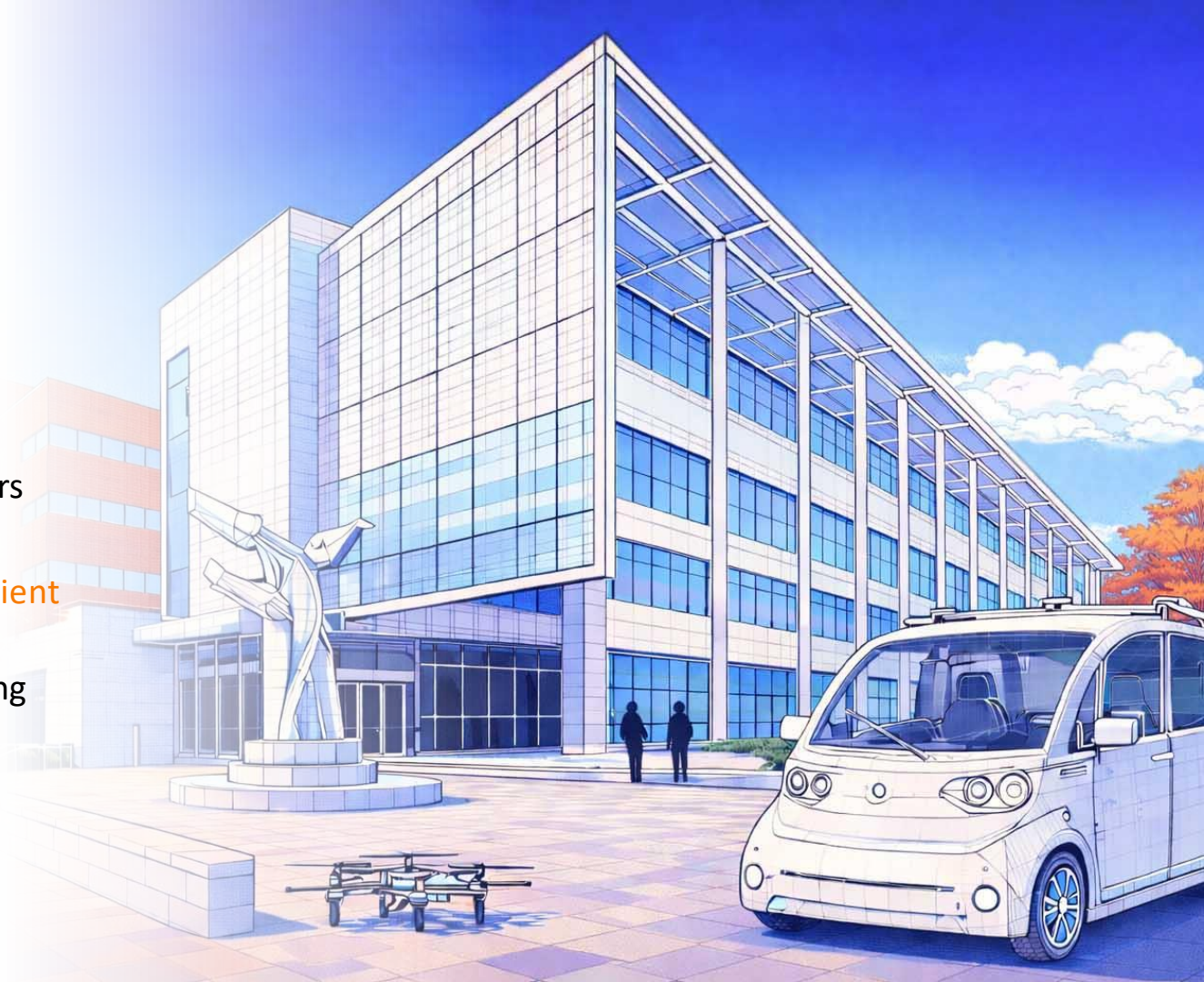
Outline

Linear classifiers

Neural networks

- Universal approximators
- Forward pass
- Backpropagation; Gradient descent
- Exploding and imploding gradients

Coordinate transforms II



Backpropagation and Gradient-Based Training

$$\boldsymbol{h} = f(\boldsymbol{z}^{(1)} = W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})$$
$$\hat{\boldsymbol{y}} = W^{(2)}\boldsymbol{h} + \boldsymbol{b}^{(2)}$$

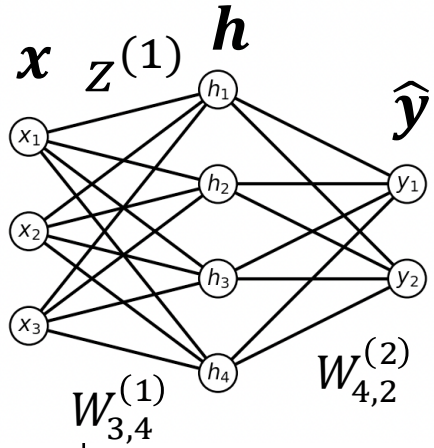
Let $L = -\frac{1}{N}\sum_{i=1}^{N} |\hat{\boldsymbol{y}}_i - \boldsymbol{y_i}|_2^2$

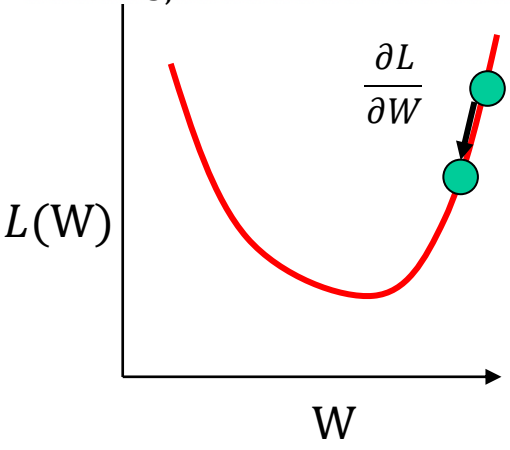
Computing Gradients (Backprop):

- We compute $\frac{\partial L}{\partial W^{(2)}}, \frac{\partial L}{\partial b^{(2)}}, \frac{\partial L}{\partial W^{(1)}}, \frac{\partial L}{\partial b^{(1)}}$ using **chain rule**
- For example:
  - $\frac{\partial L}{\partial \hat{\boldsymbol{y}}} = -\frac{2}{N}(\hat{\boldsymbol{y}}_{\boldsymbol{i}} - \boldsymbol{y_i})^{\boldsymbol{T}}$
  - $\frac{\partial L}{\partial W^{(2)}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}}\frac{\partial \hat{\boldsymbol{y}}}{\partial W^{(2)}} = \left(\frac{\partial L}{\partial \hat{\boldsymbol{y}}}\right) h^{\top}$
- Chain rule implies that derivatives (e.g. $\left(\frac{\partial L}{\partial \hat{\boldsymbol{y}}}\right)$) will be reused

Gradient descent update: $W^{(l)} \coloneqq W^{(l)} - \eta\frac{\partial L}{\partial W^{(l)}}$, $\boldsymbol{b}^{(l)} \coloneqq \boldsymbol{b}^{(l)} - \eta\frac{\partial L}{\partial \boldsymbol{b}^{(l)}}$

2-Layer Network: d=3, m=4, k=2



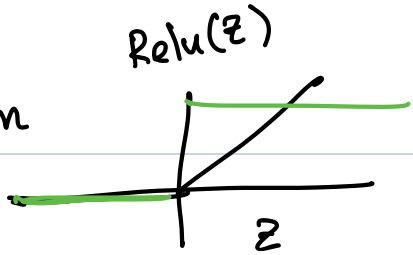


$z = w_1 x + b_1$ preactivation

$h = \text{ReLU}(z)$

$\hat{y} = w_2 h + b_2$ output

Loss $\frac{1}{2}(\hat{y} - y)^2$



Let $x = 2$ $y = 1$ $w_1 = 1$ $b_1 = -1$

$w_2 = 3$ $b_2 = 0$

Forward pass $z = 1$ $h = 1$ $\hat{y} = 3$ $L = 2$

Backward pass

$$\frac{\partial L}{\partial \hat{y}} = (\hat{y} - y) = 2$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} = 2 \cdot h = 2 \cdot 1 = 2$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b_2} = 2 \cdot 1 = 2$$

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h} = 2 \cdot w_2 = 2 \cdot 3 = 6$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h} \cdot \frac{\partial h}{\partial z} = 6 \cdot 1 = 6$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1} = 6 \cdot x = 6 \cdot 2 = 12$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} = 6 \cdot 1 = 6$$

Gradient step with learning rate $\eta = 0.1$

$$w_2 \leftarrow w_2 - \eta \frac{\partial L}{\partial w_2}$$

$$b_2 \leftarrow b_2 - \eta \, \partial L / \partial b_2$$

$$w_1 \leftarrow w_1 - \eta \, \partial L / \partial w_1$$

$$b_1 \leftarrow b_1 - \eta \, \partial L / \partial b_1$$

Recompute $L' = L(y - \hat{y}')$

Backpropagation: Computing gradients with receipts

Forward pass stored: $z = 1, h = 1, \hat{y} = 3$

Backward pass used h to compute $\frac{dL}{dw_2}$, z to know ReLU' = 0 or 1, x to compute $\frac{dL}{dw_1}$

Change b_1 from −1 to −3:

$$z = 1 \cdot 2 - 3 = -1 \Rightarrow h = 0 \Rightarrow \hat{y} = b2 = 0 \Rightarrow L = \frac{1}{2}(0-1)^2 = 0.5$$

Now in backward pass, $ReLU'(z) = 0$ so dL/dw_1 and dL/db_1 become 0

"Dead ReLU": the forward pass (sign of z) decides whether gradients flow

Under and overfitting

- **Underfitting**: training and test error are both *high*
  - Model does an equally poor job on the training and the test set
  - The model is too "simple" to represent the data or the model is not trained well
- **Overfitting**: Training error is *low* but test error is *high*
  - Model fits irrelevant characteristics (noise) in the training data
  - Model is too complex or amount of training data is insufficient

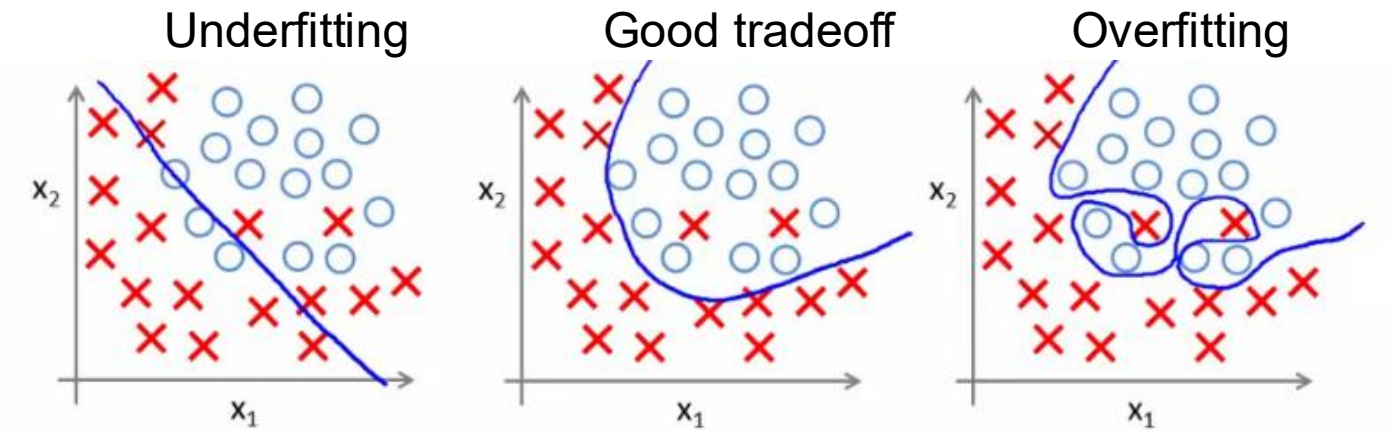


Figure source

Bias-variance tradeoff in ML models

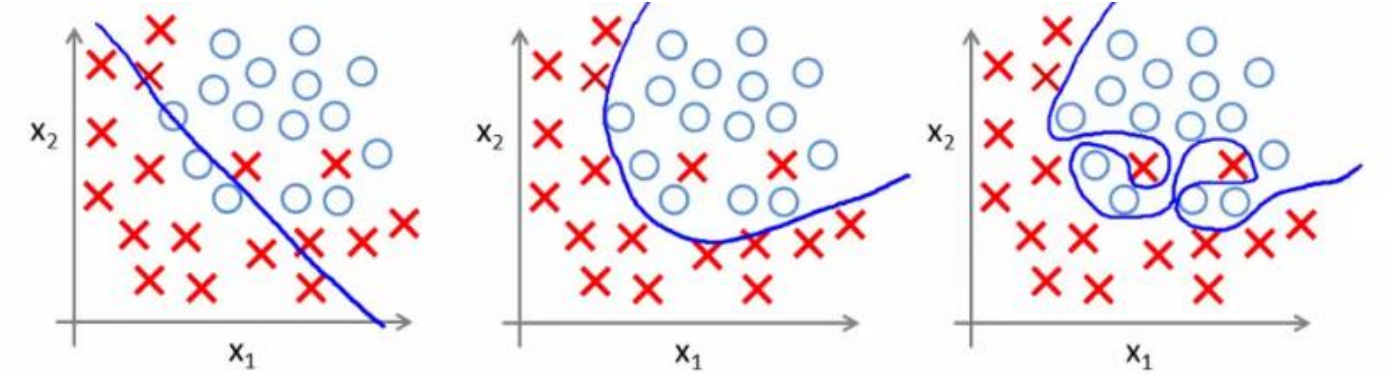
Training loss/ error of learning algorithms has two main components:

- **Bias:** error due to simplicity of models (linear, nubmer of layers, etc.)
- **Variance:** error due to randomness of training set

Bias-variance tradeoff can be tuned with hyperparameters during the **validation phase**

- E.g. hyperparameters: number of layers, activation type, network architecture, etc.

High bias, low variance



Low bias, high variance

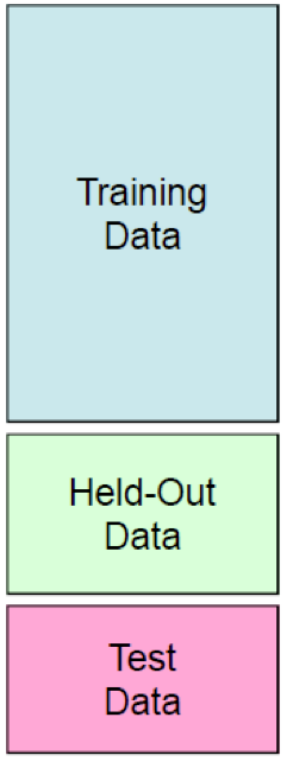
Figure source

Best practices for training classifiers

Goal: obtain a classifier with **good generalization** or performance on never before seen data

1. Learn *model parameters* on the ***training set***
2. Tune *hyperparameters* on the *held out* ***validation set***
3. Evaluate performance on the ***test set***

Crucial: do not peek at the test set when iterating steps 1 and 2!



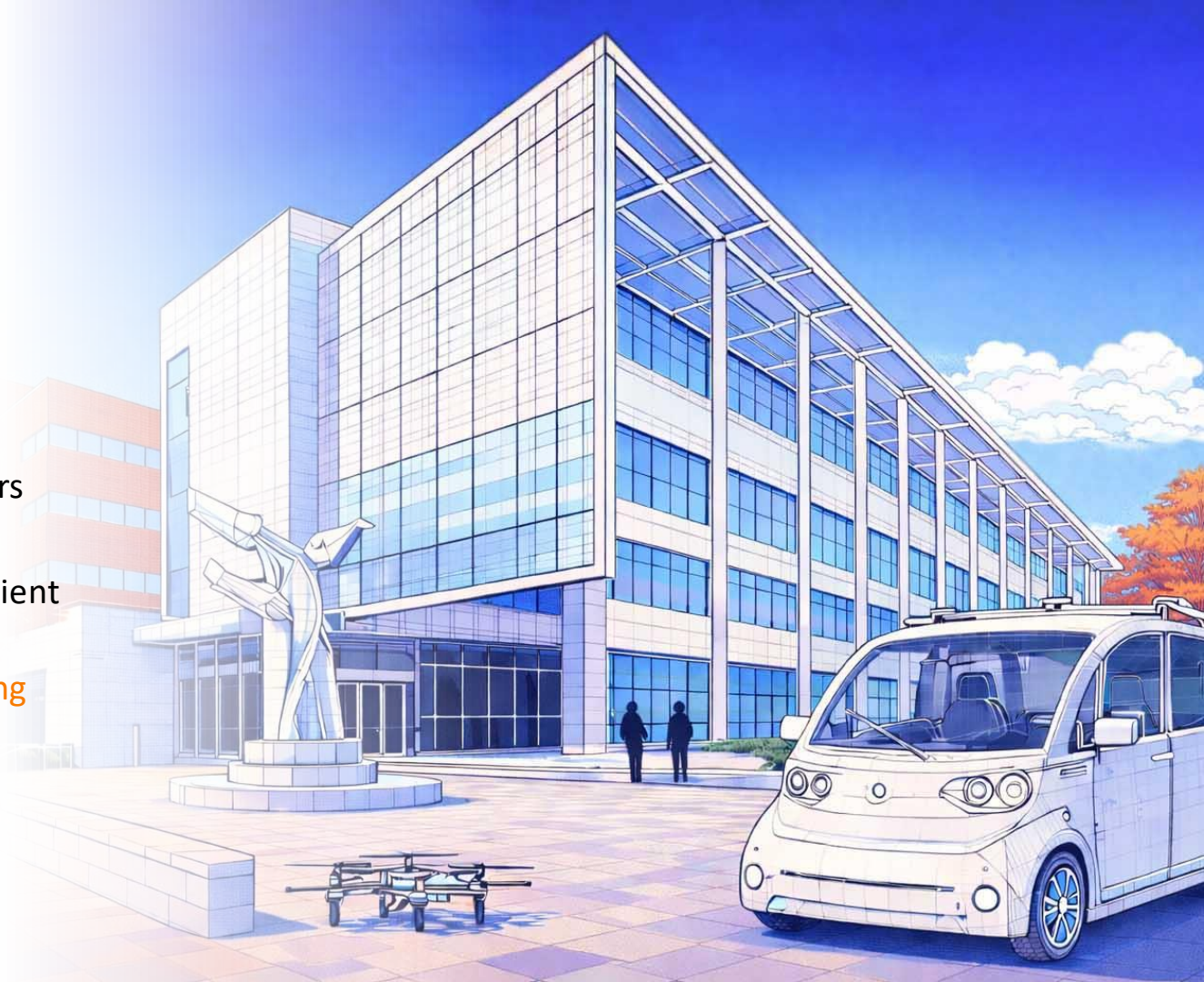
Outline

Linear classifiers

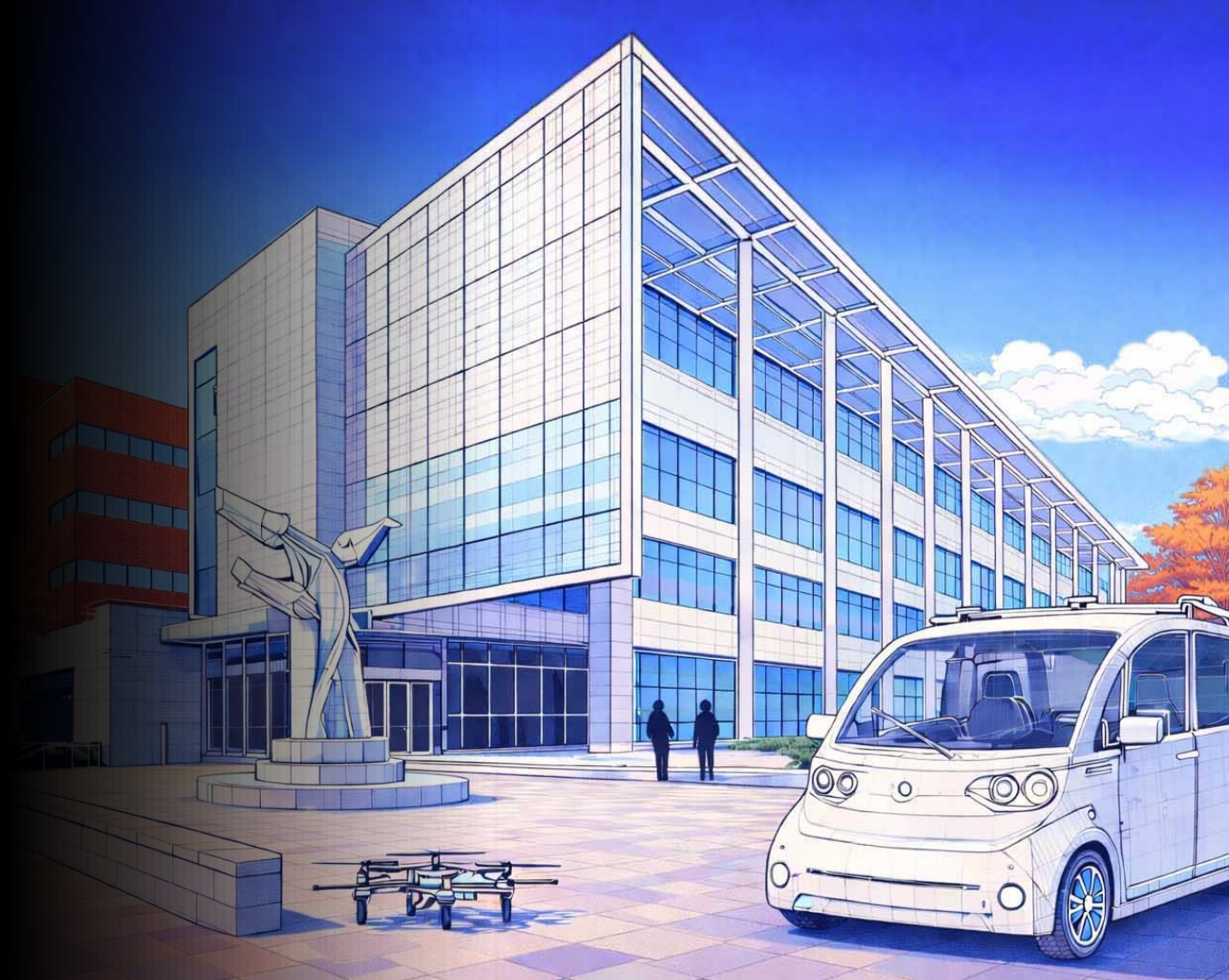
Neural networks

- Universal approximators
- Forward pass
- Backpropagation; Gradient descent
- Exploding and imploding gradients

Coordinate transforms II



Canvas Quiz



Vanishing gradient problem

Consider MLP with:

- Input $x \in \mathbb{R}^d$
- 3 hidden layers, each with **sigmoid**.
- 1 output $\hat{y} \in \mathbb{R}$ for a regression or binary classification.

That is

1. $z^{(1)} = W^{(1)}x + b^{(1)} \in \mathbb{R}^m$
2. $a^{(1)} = \sigma(z^{(1)}) \quad \in \mathbb{R}^m$
3. $z^{(i)} = W^{(i)}a^{(i-1)} + b^{(i)} \in \mathbb{R}^m, i = 2,3,4$
4. $a^{(i)} = \sigma\left(z^{(i)}\right) \quad \in \mathbb{R}^m, i = 2,3$
5. $\hat{y} = \sigma(z^{(4)}) \ \in \mathbb{R}^m$

Loss $\mathrm{L} = \frac{1}{2}(\hat{y} - y)^2$

As the number of layers increase gradient can vanish

$$\frac{\partial L}{\partial \hat{y}} = (\hat{y} - y)$$

$$\frac{\partial L}{\partial z^{(4)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(4)}} = (\hat{y} - y) \frac{\partial \hat{y}}{\partial z^{(4)}} = (\hat{y} - y)\hat{y}(1 - \hat{y})$$

$$\frac{\partial L}{\partial a^{(3)}} = \frac{\partial L}{\partial z^{(4)}} \frac{\partial z^{(4)}}{\partial a^{(3)}} = (\hat{y} - y)\hat{y}(1 - \hat{y}).W^{(4)}$$

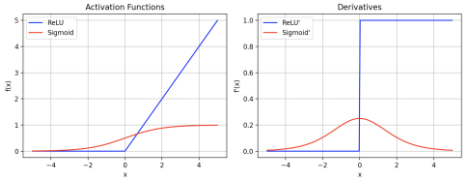
$$\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} = (\hat{y} - y)\hat{y}(1 - \hat{y}).W^{(4)} \odot \sigma'(z^{(3)})$$

$\odot$: Element-wise (Hadamard) product of backprop grad with activation derivative

$$\frac{\partial L}{\partial z^{(\ell)}} = \frac{\partial L}{\partial z^{(\ell+1)}} W^{\ell+1} \sigma'\left(z^{(\ell)}\right)$$

If each $\sigma'\left(z^{(\ell)}\right) \leq 0.25$ the gradient vanishes

$$z^{(1)} = W^{(1)}x + b^{(1)}$$
$$a^{(1)} = \sigma\left(z^{(1)}\right)$$
$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)}$$
$$a^{(2)} = \sigma\left(z^{(2)}\right)$$
$$z^{(3)} = W^{(3)}a^{(2)} + b^{(3)}$$
$$a^{(3)} = \sigma\left(z^{(3)}\right)$$
$$z^{(4)} = W^{(4)}a^{(3)} + b^{(4)}$$
$$\hat{y} = \sigma(z^{(4)})$$
$$L = \frac{1}{2}(\hat{y} - y)^2$$



$$\sigma(z) = \frac{e^z}{1+e^z} \quad \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Exploding gradient problem

Consider MLP with:

- Input $x \in \mathbb{R}^d$
- 3 hidden layers, each with **RELU**.
- 1 output $\hat{y} \in \mathbb{R}$ for a regression or binary classification.

That is

1. $z^{(1)} = W^{(1)}x + b^{(1)} \in \mathbb{R}^m = \alpha I x$ for simplicity $b^{(i)} = 0\ W^{(i)} = \alpha I$
2. $a^{(1)} = ReLU\left(z^{(1)}\right) = \alpha x$ if $a^{(1)}$ is positive in each component
3. $z^{(i)} = \alpha I a^{(i-1)} = \alpha^i x, i = 2,3,4$
4. $a^{(i)} = ReLU\left(z^{(i)}\right) = \alpha^i x, i = 2,3$
5. $\hat{y} = ReLU\left(z^{(4)}\right) = \alpha^4 x$

Loss $\mathrm{L} = \frac{1}{2}(\hat{y} - y)^2$

Exploding gradient continued

$$\frac{\partial L}{\partial z^{(4)}} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z^{(4)}} = \alpha^4 x\ - y.1$$

$$\frac{\partial L}{\partial a^{(3)}} = \frac{\partial L}{\partial z^{(4)}}\frac{\partial z^{(4)}}{\partial a^{(3)}} = (\alpha^4 x\ - y.1)\alpha I$$

$$\frac{\partial L}{\partial z^{(3)}} = \frac{\partial L}{\partial a^{(3)}}\frac{\partial a^{(3)}}{\partial z^{(3)}} = (\alpha^4 x\ - y.1)\alpha.1$$

$$\frac{\partial L}{\partial a^{(2)}} = \frac{\partial L}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial a^{(2)}} = (\alpha^4 x\ - y.1)\alpha\alpha.I = \alpha^2(\alpha^4 x\ - y.1)$$

…

$$\frac{\partial L}{\partial a^{(1)}} = \alpha^3(\alpha^4 x\ - y.1)$$

$$z^{(1)} = W^{(1)}x + b^{(1)} = \alpha I x$$
$$a^{(1)} = ReLU\left(z^{(1)}\right)\ = \alpha x$$
$$z^{(i)} = \alpha I a^{(i-1)}$$
$$a^{(i)} = ReLU\left(z^{(i)}\right)$$
$$\hat{y} = ReLU\left(z^{(4)}\right)$$
$$L = \frac{1}{2}(\hat{y} - y)^2$$

If $\alpha \gg 1$ the factor α^3 explodes in the gradient computation

Caution: Sigmoid activations clip the gradient and can lead to vanishing gradients

ReLU can make the gradients large

Summary

- Neural networks, MLPs overcome the bottleneck of linear classifiers for problems that are not linearly separable
- Bakckprop is gradient descent with smart bookkeeping; utilizes results from forward pass
- Under/over-fitting and bias-variance trade-off in training NN models is handled by hyperparameter tuning on an held-out data set
- For deep networks gradients may vanish or explode impeding training; addressed by **good initialization, normalization**

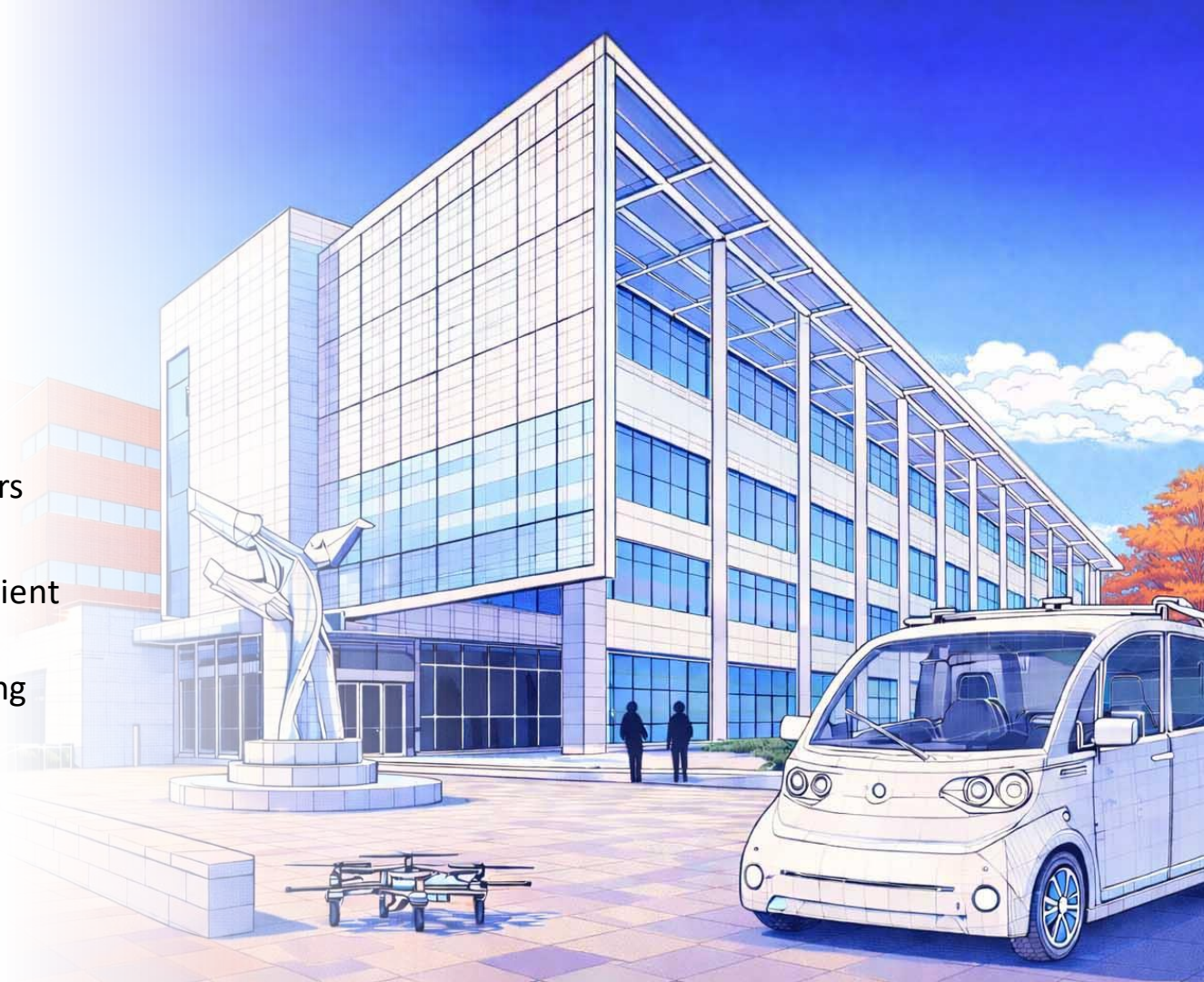
Outline

Linear classifiers

Neural networks

- Universal approximators
- Forward pass
- Backpropagation; Gradient descent
- Exploding and imploding gradients

Coordinate transforms II



ECE484: Coordinate Transformations (Part 2)

Sayan Mitra

Spring 2026

Outline

- Simple translations
- 2D rotations
- Properties of rotation matrices
- 3D rotations
- Full transform and inverse

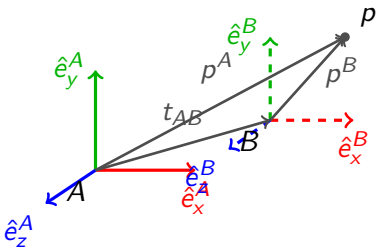
1. Simple Translations

- Frames A and B are parallel (no rotation).
- Let t_{AB} be the position of O_B expressed in A.
- Coordinate transform (translation only):

$$p^A = t_{AB} + p^B.$$

- Inverse transform:

$$p^B = p^A - t_{AB}.$$



2. 2D Rotations

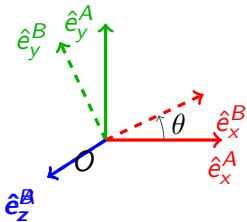
- B is rotated CCW by angle θ about $\hat{e}_z$ from A.
- Rotation $R_{AB}(\theta) \in \mathbb{R}^{2\times 2}$ maps coordinates in B to A.
- Let $\hat{e}_x^B, \hat{e}_y^B$ be the unit axes of frame B, expressed in A:

$$\hat{e}_x^B = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \quad \hat{e}_y^B = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix}.$$

- Rotation matrix $R_{AB}(\theta)$ built by stacking axis coordinates as columns.

$$R_{AB}(\theta) = \begin{bmatrix} \hat{e}_x^B & \hat{e}_y^B \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

- $p^A = R_{AB}(\theta) p^B$.

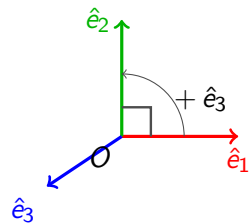


3. Properties of Rotation Matrices 2D and 3D

- Rotation $R \in \mathbb{R}^{3\times 3}$ in 3D lie in $\mathrm{SO}(3)$.

$$\mathrm{SO}(3) = \{R \in \mathbb{R}^{3\times 3} \mid R^\top R = I,\ \det(R) = +1\}.$$

- $R^T R = I$ means orthogonality, i.e., columns are orthogonal vectors.
- $|det(R)| = 1$ means preservation of volume
- $det(R) = +1$ means right-handedness, i.e., preservation of orientation.
- Implications: inverse is transpose, lengths/angles preserved.



$$\|\hat{e}_x^B\| = \|\hat{e}_y^B\| = \|\hat{e}_z^B\| = 1, \quad \hat{e}_i^B \cdot \hat{e}_j^B = 0\ (i \neq j), \quad \hat{e}_x^B \times \hat{e}_y^B = \hat{e}_z^B.$$

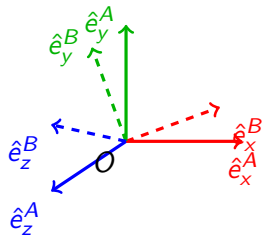
$$R^{-1} = R^\top, \qquad \|Rp\| = \|p\|, \qquad (Rp)\cdot(Rq) = p\cdot q.$$

4. 3D Rotations

- Build R_{AB} by stacking B's axes in A (3D analog of 2D).

$$\hat{e}_x^B = \begin{bmatrix} r_{11} \\ r_{21} \\ r_{31} \end{bmatrix}, \quad \hat{e}_y^B = \begin{bmatrix} r_{12} \\ r_{22} \\ r_{32} \end{bmatrix}, \quad \hat{e}_z^B = \begin{bmatrix} r_{13} \\ r_{23} \\ r_{33} \end{bmatrix}.$$

$$R_{AB} = \begin{bmatrix} \hat{e}_x^B & \hat{e}_y^B & \hat{e}_z^B \end{bmatrix}.$$



- Yaw–pitch–roll parameterization:

$$R = R_z(\psi)R_y(\theta)R_x(\phi).$$

5. Full Transform and Inverse

- Rigid transform combines rotation and translation. $p^A = t_{AB} + R_{AB} p^B$.
- Affine transform that can be represented as a linear map in homogeneous coordinates.
- Homogeneous transform $T_{AB} \in \mathrm{SE}(3)$ is a 4×4 matrix.

$$\begin{bmatrix} p^A \\ 1 \end{bmatrix} = T_{AB} \begin{bmatrix} p^B \\ 1 \end{bmatrix}, \quad T_{AB} = \begin{bmatrix} R_{AB} & t_{AB} \\ 0 & 1 \end{bmatrix}.$$

- Acts on homogeneous coordinates in $\mathbb{R}^4$.
- Inverse has closed form;

$$T_{BA} = T_{AB}^{-1} = \begin{bmatrix} R_{AB}^\top & -R_{AB}^\top t_{AB} \\ 0 & 1 \end{bmatrix}, \qquad T_{AC} = T_{AB} T_{BC}.$$