Machine Learning

Prof. Sergei Gleyzer

Lecture

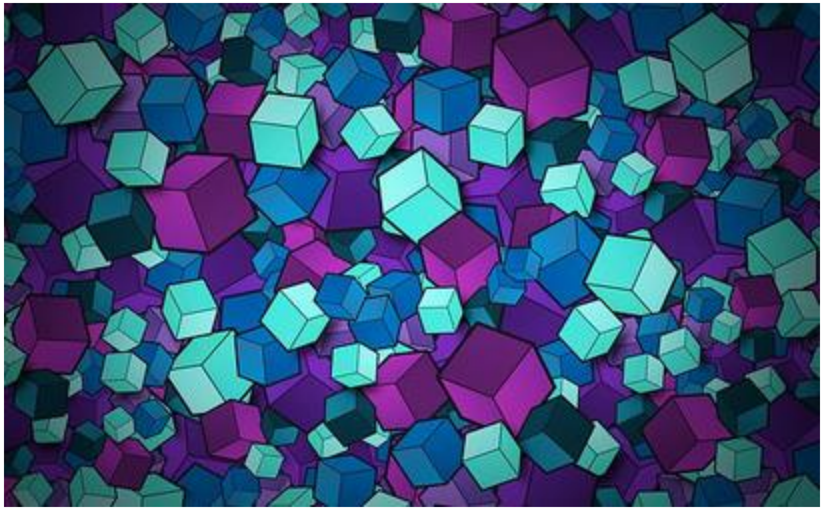
PH451, PH551

February 20, 2025

Announcements

- **Hackathon #1 – due tomorrow**
- **Hands-on #5 – due next Thu.**

Dimensionality Reduction



Why?

- **Better ML algorithm performance**
- **Visualization**
- **Data Compression**
- **Remove Noise**

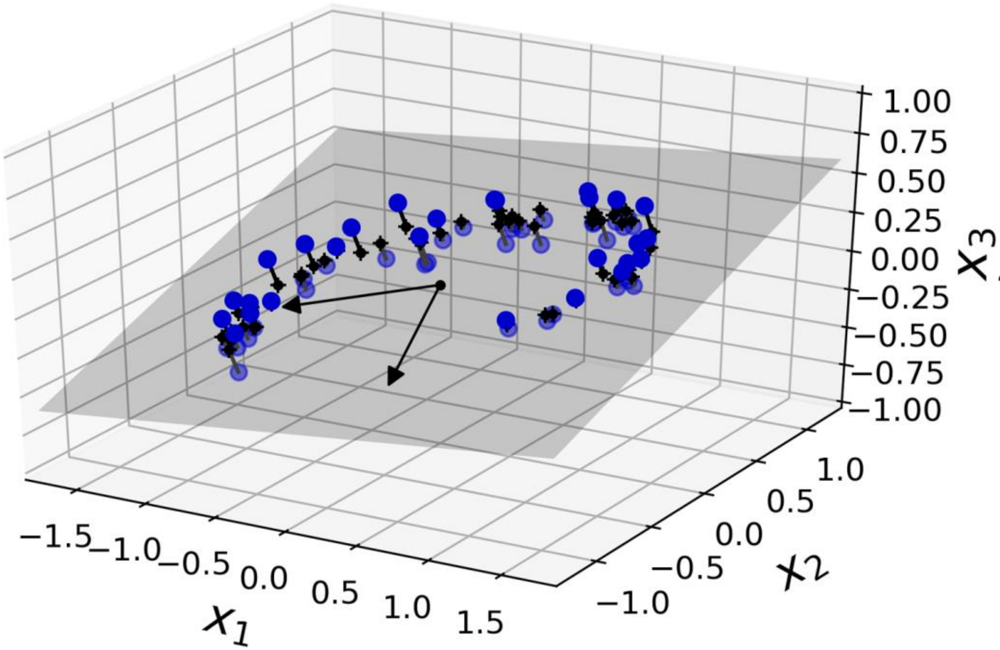
Dimensionality Reduction

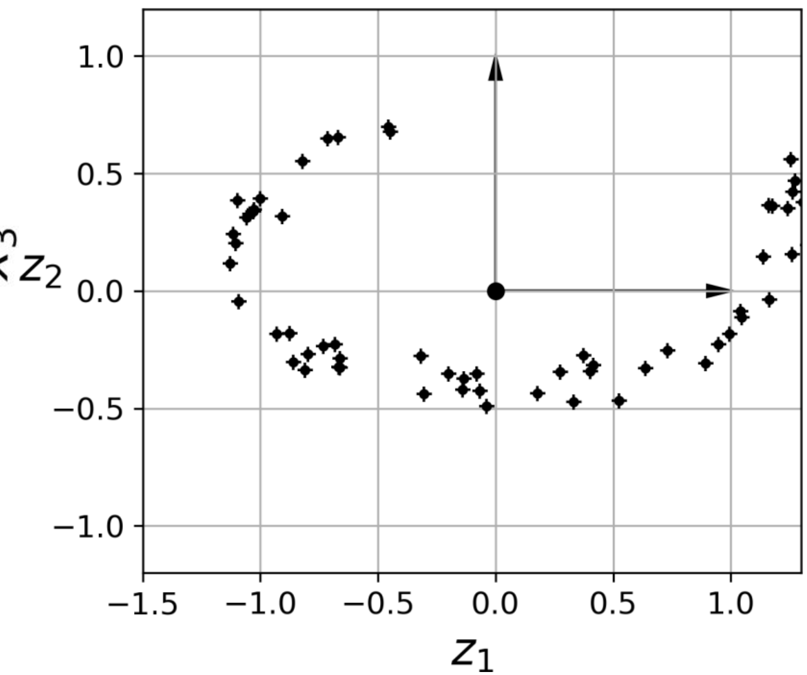
Goal:

- Find the **smallest subspace** of the N-D space that keeps the **most information** about the original data

- **Projection**
- **Manifold Learning**

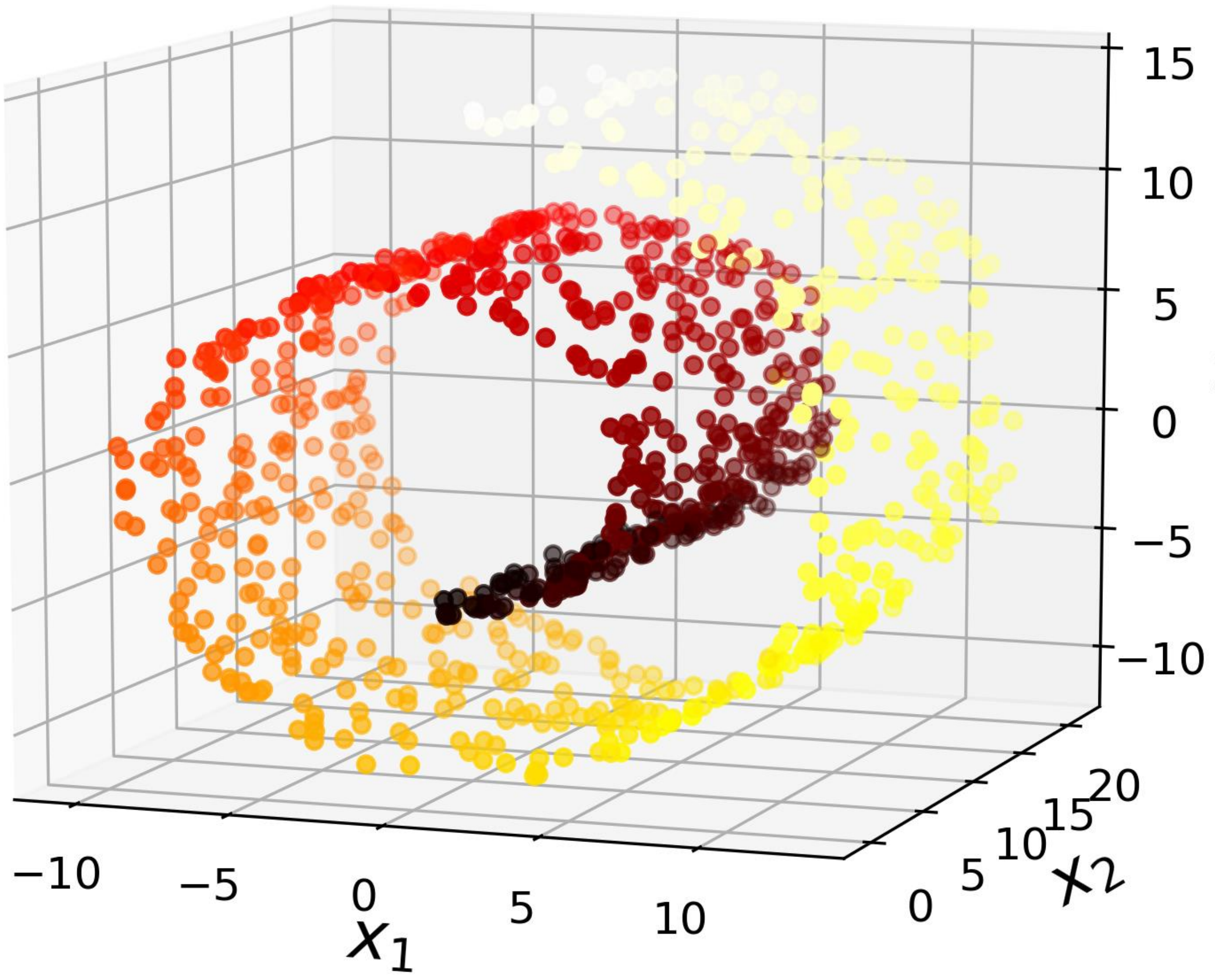
Projection





Lower dimensional subspace projection

What about this case?



2-D Manifold

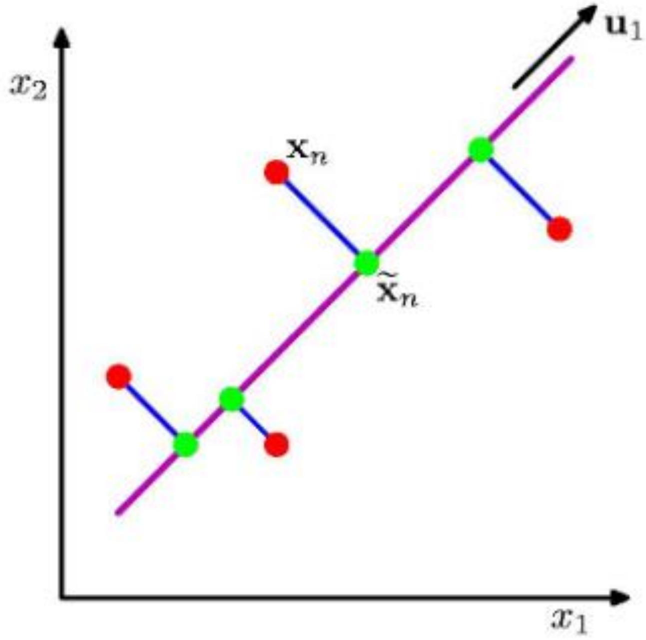
PCA

Principal Components

- Linear Method, Pearson/Hotelling 1901/1933
- Find the **hyperplane closest** to data and project into it
  - Minimize the **squared distance** between original data and projection
  - Orthogonal axes that **maximize remaining variance** (principal components)
  - Find with Singular Value Decomposition (SVD)
  - Ignore components of lesser significance

PCA

Principal Components



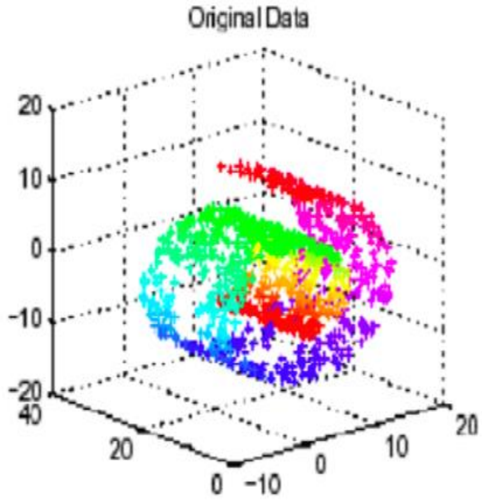
Maximize variance (purple)

Minimize mean squared distance between data points and projections (sum of blue lines)

LLE

Locally Linear Embedding

- **Learn the manifold**
  - Low-dimensional representation that best preserves **local relationships** between data
  - Minimize **squared distance** between instance and linear (weighted) function of its neighbors

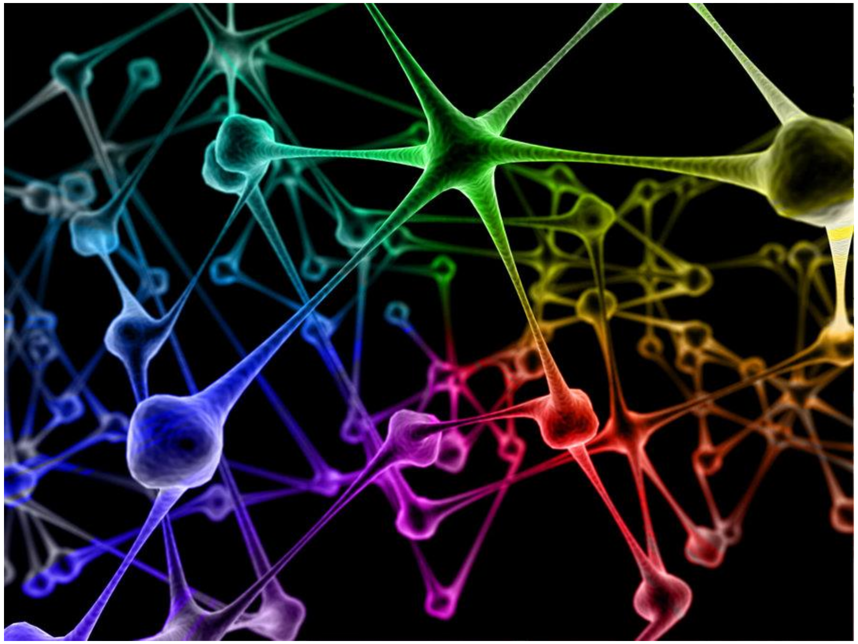


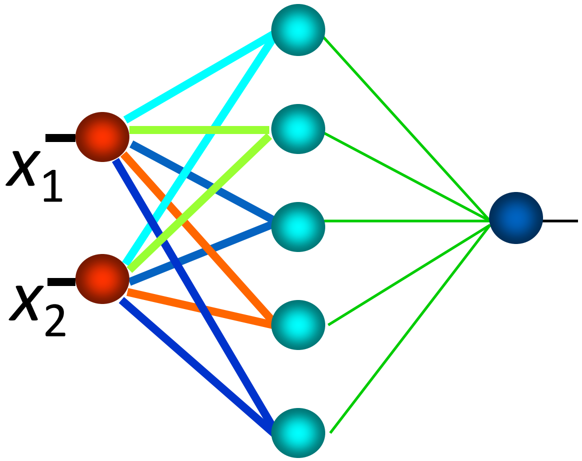
t-SNE

t-Distributed Stochastic Neighbor Embedding

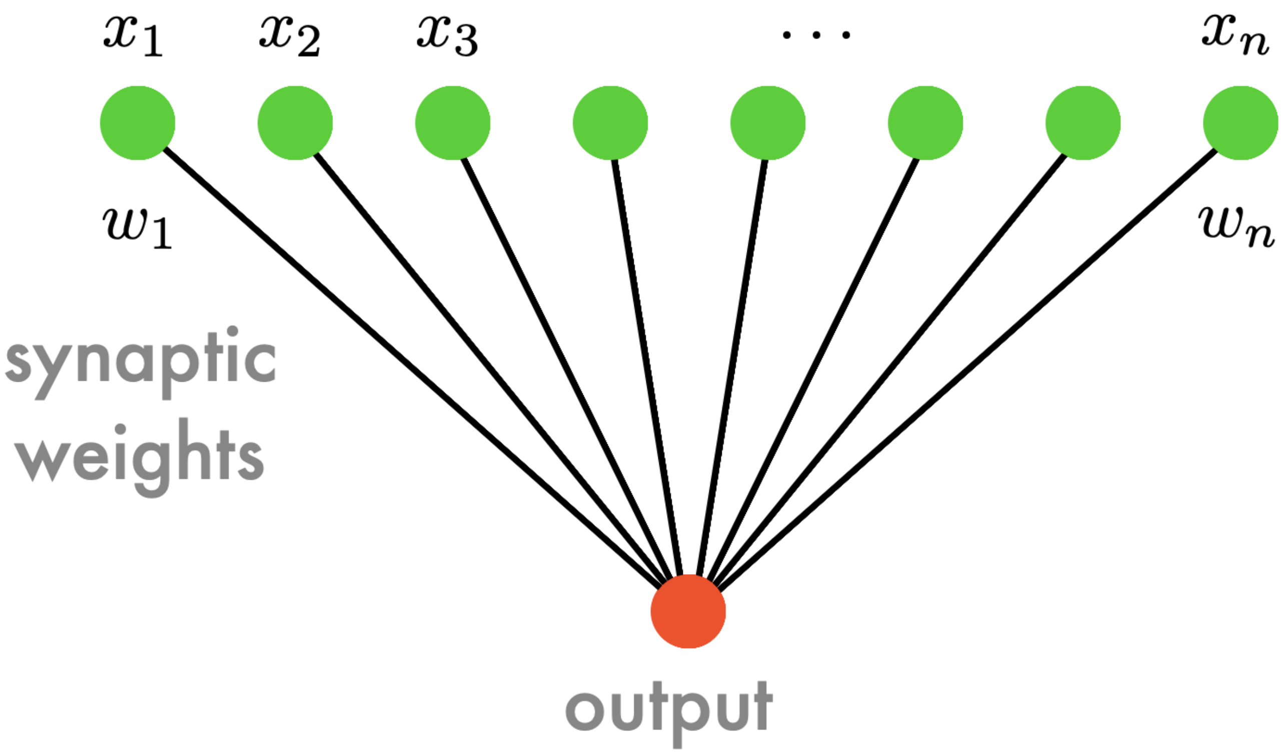
- Van der Maatens and Hinton (2008)
- **Reduce dimensionality**
  - keep similar instances close and different instances apart
  - Measure similarity between points in **high-dimensional** space (Gaussian) and **low-dimensional** space (Student t-distribution) and minimize **Kullback-Liebler** (KL) divergence cost function
    - Recall that KL measures the difference of probability distributions
- **Great for visualization (2D)**

Neural Networks





Perceptron



Frank Rosenblatt, 1957

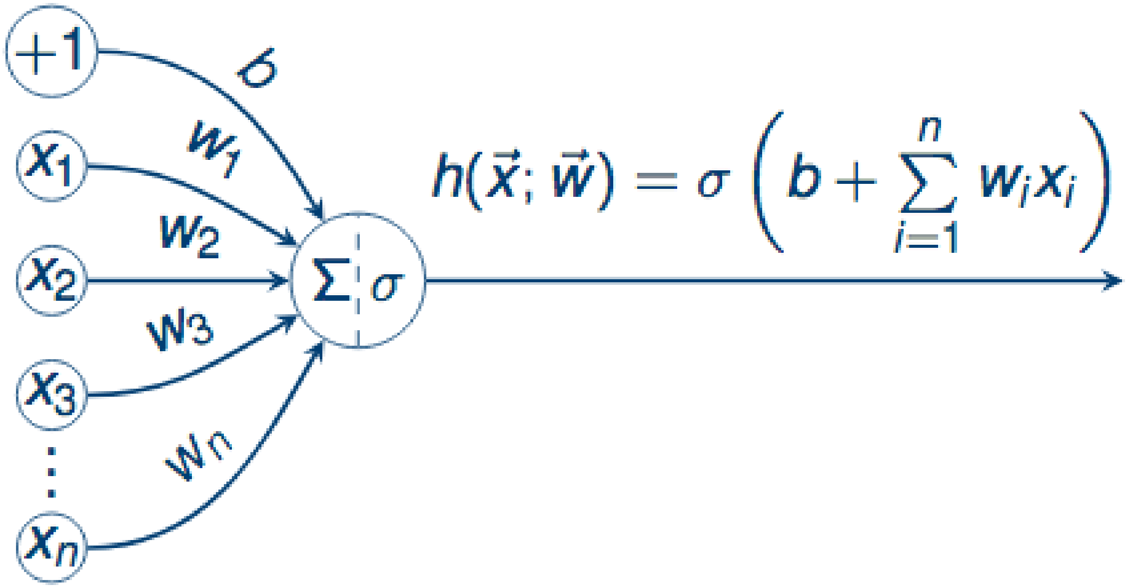
Perceptron Learning

Perceptrons

- **Threshold Logic Unit/Step Function**
  - Linear combination of inputs
  - Classify above threshold
- **Hebbian Learning Rule:**
  - "Fire together, wire together"
- **Linear decision boundary**
  - XoR Classification Problem Minsky and Papert 1969
- **Stack into MultiLayer Perceptrons (MLPs)**

Graphical Representation

Artificial Neuron

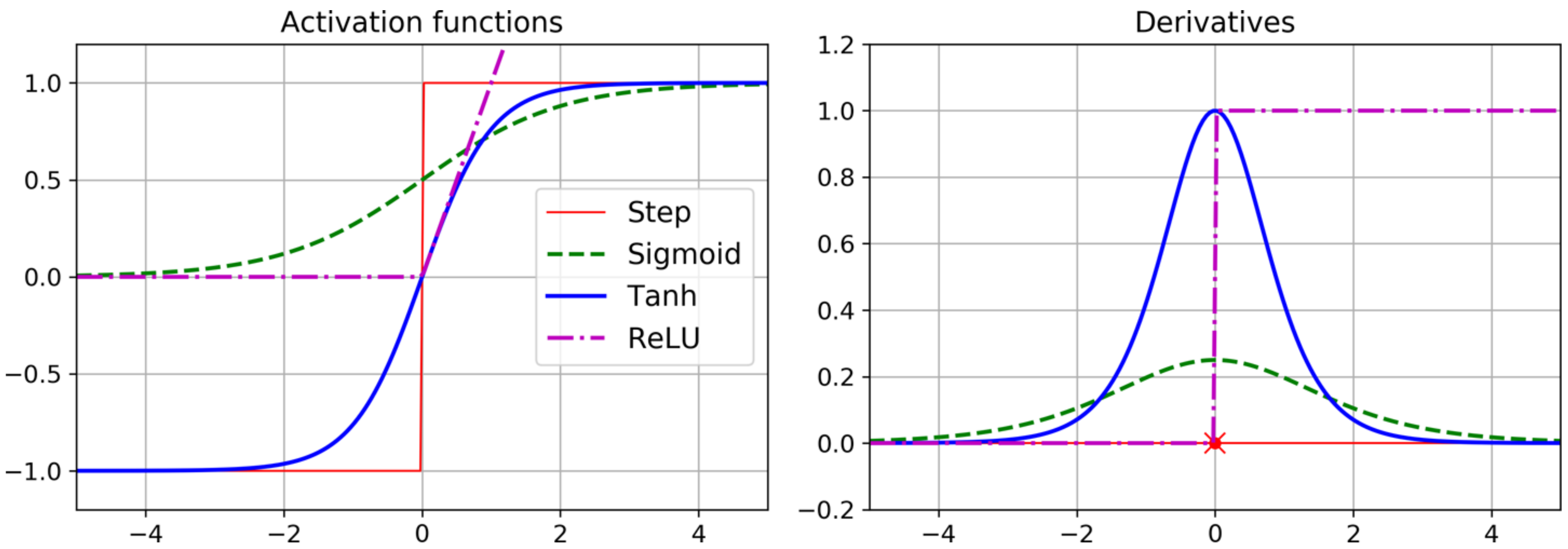


Identity: $\sigma(X) = X$

ReLU: $\sigma(X) = \max(0, x)$

Sigmoidal: $\sigma(X) = [1+\exp(-x)]^{-1}$, $\sigma(X) = \tanh x$

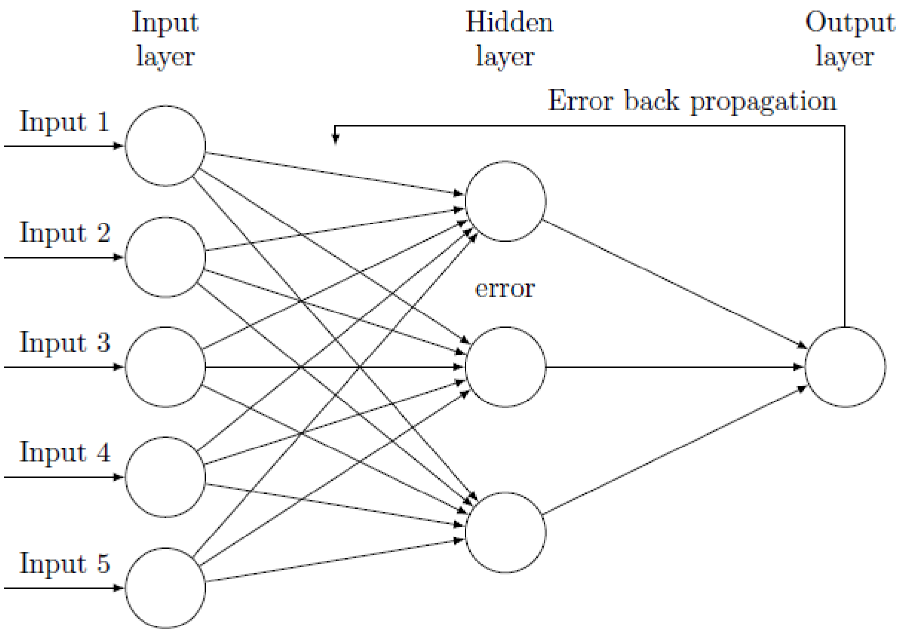
Activation Functions



Adjustable Weights

Compute network weights with

- Error gradients



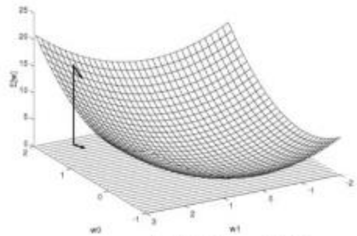
Inputs forward

Errors go backward

- **Rumelhart, Hinton and Williams 1986**

Backpropagation

- **Forward pass**
  - Outputs of all neurons from layer to layer
  - Use Loss Function to measure error
- **Backward pass**
  - Compute gradient of error w.r.t. every weight and bias term until you reach input layer
  - Use Chain Rule
- **Gradient Descent**
  - Update weights with error gradients



$$\nabla E[\mathbf{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n}\right]$$

What we can learn

- **Binary Classification**

$$\log(1 + \exp(-y y_n))$$

- **Multiclass Classification (softmax)**

$$\log \sum_{y'} \exp(y_n[y']) - y_n[y]$$

- **Regression**

$$\frac{1}{2} \|y - y_n\|^2$$

Can Choose

- **Number of hidden layers**
- **Number of neurons per layer**
- **Batch Size**
  - Especially relevant for GPUs
- **Activation Function**
- **Loss Function**
- **Learning Rate**
- **Optimizers**
- **Regularization**

Sigmoids

- **Very popular because of biological systems**
- **Saturates for large positive or negative value**
  - Zero derivatives – vanishing gradients
  - Poor choice for deep networks
- **Still very useful for output nodes**

Vanishing Gradients

- Problem with sigmoid: saturation