Neural Networks (Deep Learning)

Cornell CS 3/5780 · Spring 2026

1. Nonlinear Methods

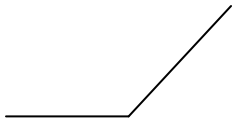
- **Review: Kernels**:
  - Kernel methods lift linear models into nonlinear predictors.
  - Nonlinear features: $\phi(x)$, where $\phi(x)$ is a feature map.
  - Kernel function: (feature map is implicit and high dimensional)
    $$k(x, x') = \phi(x)^\top \phi(x')$$
  - Training complexity: $> n^2$ (kernel matrix) and test complexity: $O(n)$.
- **Today: Neural Networks**
  - Multilayer Perceptron invented at Cornell by Frank Rosenblatt (1963)
  - Learns the feature mapping **explicitly** from data.
  - Can capture structure by stacking layers of "neurons".

2. Neural Networks as Learnable Features

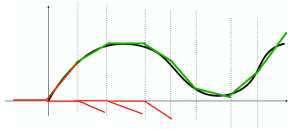
- **Single Neuron Network**: Given input $x \in \mathbb{R}^{d+1}$ with bias $x[d+1] = 1$:
  $$y = a\,\sigma(w^\top x) + b$$
  where $\sigma : \mathbb{R} \to \mathbb{R}$ is an *activation function*.
- Common activations:
  - **ReLU**: $\sigma(a) = \max(a, 0)$
  - **Sigmoid**: $\sigma(a) = \frac{e^a}{1+e^a}$
  - **Tanh**: $\sigma(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
- **Stacking Multiple Neurons**
  - Example with K neurons: $y = \sum_{i=1}^{K} a_i, \mathrm{ReLU}(x^\top w_i) + b$
  - Vectorized: Let $W \in \mathbb{R}^{K \times d}$ and let $a \in \mathbb{R}^K$
    $$y = a^\top \mathrm{ReLU}(Wx) + b$$
- This defines a **learnable feature map** $\phi(x) = \mathrm{ReLU}(Wx)$.

3. What Does a Neural Network Approximate?

- For one hidden layer:
  $$y = a^\top \mathrm{ReLU}(Wx) + b$$
  - This is a **piecewise linear** function.
  - In 1D (with $b = 0$):
    $$y = a_1 \max(w_1 x + c_1, 0) + a_2 \max(w_2 x + c_2, 0) + \ldots$$



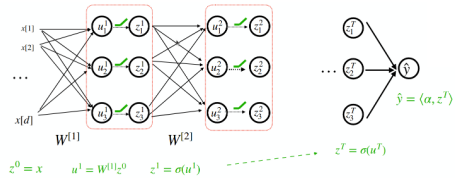
- **Claim:** A sufficiently wide one-layer neural network can approximate **any smooth function**.



4. Multi-Layer Fully Connected Neural Network

- Going Deep: depth allows hierarchical feature composition.
  - Can represent complex functions without extremely wide layers.
- Forward pass:
  - Input: $z^{(0)} = x$
  - For layers $t = 1 \dots T$:

$$u^{(t)} = W^{(t)} z^{(t-1)}$$
$$z^{(t)} = \sigma(u^{(t)})$$



  - Output:

$$y = a^\top z^{(T)} + b$$

- Computation cost: Linear in number of edges + number of nodes.

5. Training Neural Networks with Gradients

- Given model:

$$h(x) = a^\top \sigma(W^{(L)} \dots \sigma(W^{(2)} \sigma(W^{(1)} x)) \dots) + b$$

- Loss: for a single data point $\ell(h(x), y)$
- Compute gradients:
  - $\frac{\partial \ell}{\partial W_{ij}^{(k)}}$ (for all i, j, k), $\frac{\partial \ell}{\partial a_j}$ (for all j), $\frac{\partial \ell}{\partial b}$
- Basic idea: use the chain rule
  - Example: scalar feature, one layer, squared loss

$$z = wx, \quad h = \sigma(z), \quad \hat{y} = a \cdot h + b, \quad \ell(\hat{y}, y) = (y - \hat{y})^2$$

    What are $\frac{\partial \ell}{\partial w}$, $\frac{\partial \ell}{\partial a}$, $\frac{\partial \ell}{\partial b}$?
  - Next lecture: *backpropagation*, a more efficient algorithm.

6. Training Neural Networks with Mini-Batch SGD

- Parameters: $\theta = W^{(1)}, W^{(2)}, \dots, W^{(L)}, a, b$
- Procedure:
  1. Shuffle dataset.
  2. Split into batches of size B.
  3. For each batch D_j:

$$\theta \leftarrow \theta - \eta \frac{1}{B} \sum_{(x,y) \in D_j} \nabla_\theta \ell(h(x), y)$$

- Practical notes:
  - Mini-batches give **unbiased gradient estimates**.
  - Nonconvex optimization: solution depends on initialization.
  - Momentum helps escape saddle point, Adagrad also works well.
  - SGD/Adagrad generalize well even for large deep networks.