COS324: Introduction to Machine Learning

Lecture 17: Neural Networks

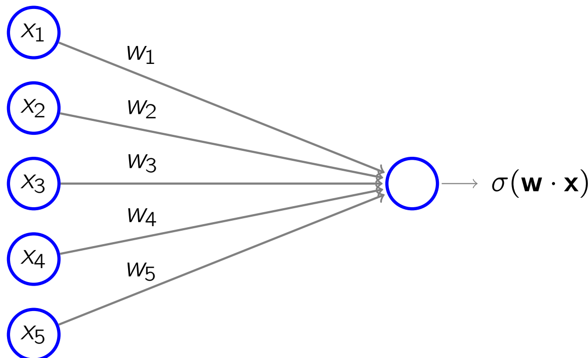
Prof. Elad Hazan & Prof. Yoram Singer

December 7, 2017

Single Neuron

- A neuron is a function of the form $\mathbf{x} \mapsto \sigma(\mathbf{w} \cdot \mathbf{x})$
- Activation function $\sigma : \mathbb{R} \to \mathbb{R}$
- For instance, the sigmoid function

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



Neural Network

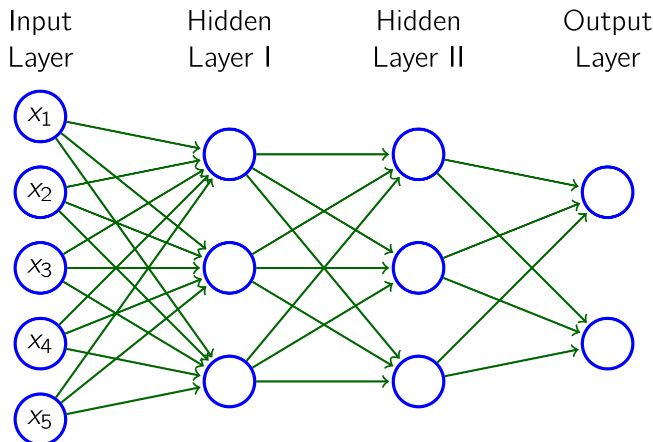
- Obtained by connecting several neurons together
- Focus on feed-forward networks
- Directed (acyclic) graph $G = (V, E)$ and denoting $n = |V|$
- Input nodes: nodes w/o incoming edges $v_1, \dots, v_d$
- Output nodes: nodes w/o outgoing edges $v_{n-l+1}, \dots, v_m$
- Weights associated with each edge $\mathbf{w} : E \to \mathbb{R}$
- Computation defined by NN

$$o[v] = \sigma \left(\sum_{u \to v \in E} \mathbf{w}[u \to v] \, o[u] \right)$$

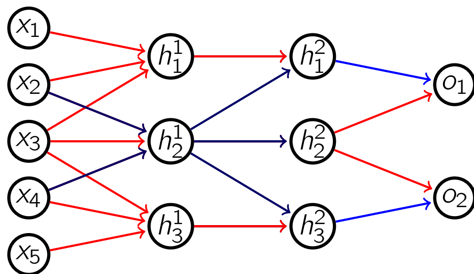
  where for (input) node $j \in [d]$ we define $o[v_j] = x_j$
- Feedfoward NN defines a function $h : \mathbb{R}^d \to \mathbb{R}^l$

Multilayer Neural Networks (MLP)

- Layers $V = \{V_i\}_{i=1}^{r}$ with edges between adjacent layers
- Example: input $d = 5$, depth $r = 3$, size $5 + 6 + 1$



Prediction in MLP



- $\mathbf{x} = (1, -1, 2, 2, 1)$ $\quad \xrightarrow{+} \quad \xrightarrow{-}$
- Activations: $\sigma(z) = \mathrm{sign}(z)\,[|z| - \gamma]_+$
- Define $A \in \mathbb{R}^{3\times5}$ $\quad B \in \mathbb{R}^{3\times3}$ $\quad C \in \mathbb{R}^{2\times3}$ $\quad$ where $M_{j,i} = w[i \to j]$
- $\mathbf{h}^1 = \sigma(A\,\mathbf{x}) = [-1.5, -0.5, -4.5]$ and $\sigma(\mathbf{v}) = (\sigma(v_1), \ldots, \sigma(v_d))$
- $\mathbf{h}^2 = \sigma(B\,\mathbf{h}^1) = [0.5, 0, 3.5]$
- $\mathbf{o} = \sigma(C\,\mathbf{h}^2) = [0, 3]$

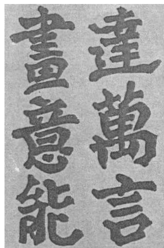
Code is Worth Thousand Pictures

```
% Activation function
def shrink(z, gamma):
    abcap = np.maximum(abs(z), np.zeros(z.shape)) - gamma
    return abcap * np.sign(z)

% Define network arch
x = np.array((5, 1))
h1 = np.zeros((3, 1)) ;   h2 = np.zeros((3, 1))
o = np.zeros((2, 1))

% Define network parameters
A = np.random.randn((len(h1), len(x))) / np.sqrt(len(x))
B = np.random.randn((len(h2), len(h1))) / np.sqrt(len(h1))
C = np.random.randn((len(o), len(h2))) / np.sqrt(len(h2))

% Define functional connectivity
h1 = shrink(np.dot(A, x), 0.5)
h2 = shrink(np.dot(B, h1), 0.5)
o = shrink(np.dot(C, h2), 0.5)
```



Hypothesis Class

- Architecture of NN is $\mathcal{N} = (V, E, \sigma)$
- $\mathbf{w}$ and $\mathcal{N}$ induce hypothesis $h_{(\mathcal{N},\mathrm{w})} : \mathbb{R}^d \to \mathbb{R}^l$
- Set of possible weights $\mathbf{w} \in \mathcal{M}$ defines a hypothesis class

$$\mathcal{H}_{\mathcal{N}} \;=\; \{h_{\mathcal{N},\mathrm{w}} \;:\; \mathbf{w} \in \mathcal{M}\}$$

- Architecture often hand-crafted, reflects insights of problem
- Albeit not a convex set, error can be decomposed:
  - Estimation error of NN (sample complexity): $\sim |E|$
  - Approximation error of NN (expressivenss): many functions approximated by NN if $|V| = O(\exp(d))$ o.w. ?
  - Optimization error of NN (computational complexity): Numerous hardness results, yet SGD often works well

Training NN

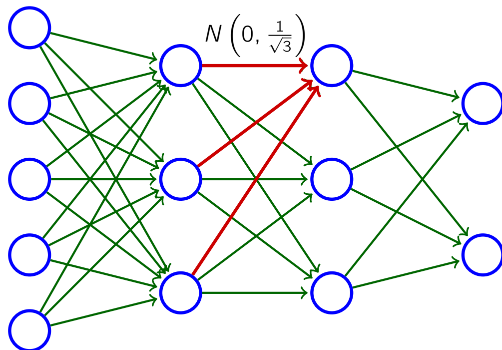
- $h_{\mathcal{N},\mathrm{w}}$ defines a non-convex transformation
- Even if loss is convex ERM becomes non-convex
- Initialization is important: symmetry breaking, scale sensitive
- Gradient-based training takes into account NN's architecture
- back-propagation algorithm is an efficient way to calculate $\nabla \ell \left(h_{(\mathcal{N},\mathrm{w})}(\mathbf{x}), y \right)$ using the chain rule
- Inference and gradient as computations on a graph
- Takes long time to train, yields good results on many tasks
- Many tricks-of-the-trade

Initialization

Need to break symmetry, scale is important

$$\mathbf{w}[u \to v] \sim N\left(0\,,\,\frac{1}{\sqrt{in(v)}}\right)$$

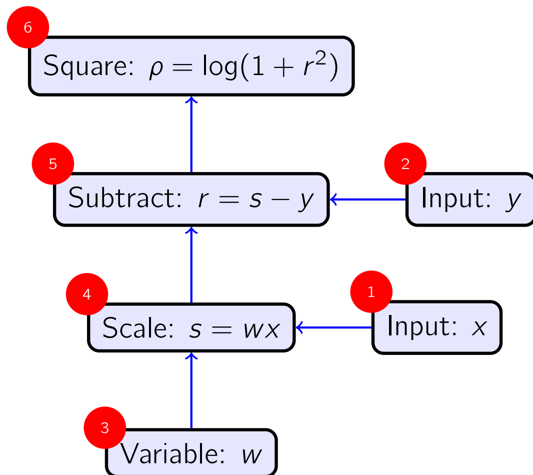
where $in(v) = |\{u : u \to v \in E\}|$



Computation Graph

Graph for one dimensional problem (nodes topologically sorted)

$$\ell(w,(x,y)) = \log\left(1+(wx-y)^2\right) \quad \text{convex: } |wx-y| \le 1$$



Gradient Using Chain Rule

- Fix x, y and define functions

$$\begin{aligned} \rho(z) &= \log(1 + z^2) \\ r_y(z) &= z - y \\ s_x(z) &= xz \end{aligned}$$

- Write ℓ as a function of w

$$\ell(w) = \rho(r_y(s_x(w))) = (\rho \circ r_y \circ s_x)(w) \ .$$

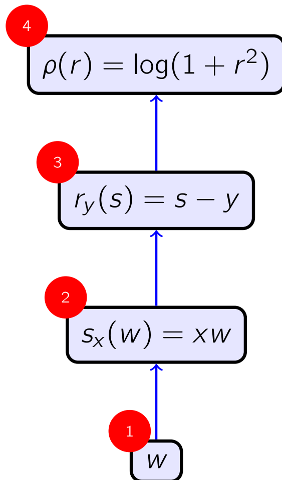
- Chain rule:

$$\begin{aligned} \ell'(w) &= (\rho \circ r_y \circ s_x)'(w) \\ &= \rho'(r_y(s_x(w))) \cdot (r_y \circ s_x)'(w) \\ &= \rho'(r_y(s_x(w))) \cdot r_y'(s_x(w)) \cdot s_x'(w) \end{aligned}$$

Forward Pass: Inference

For $v = 1, \ldots, m$:

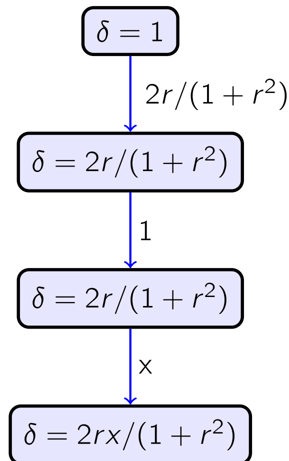
```
  v.output = v.op(v.inputs())
```



Backward Pass: Gradient

- `m->delta = 1`
- For $v = m-1, \dots, 1$:
  - Foreach u s.t. $u \to v \in E$:

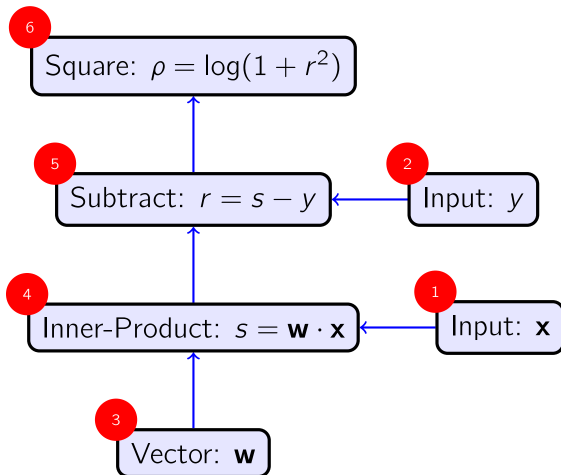
```
u->delta = v->delta * v->deriv(u)
```



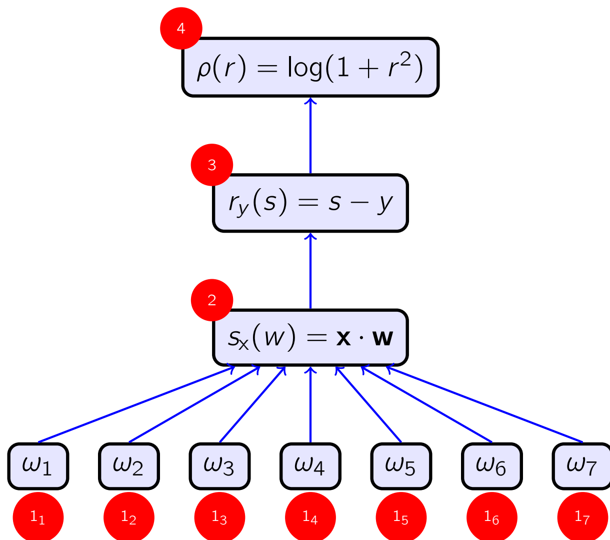
From Scalars to Neurons

Graph for one dimensional problem (nodes topologically sorted)

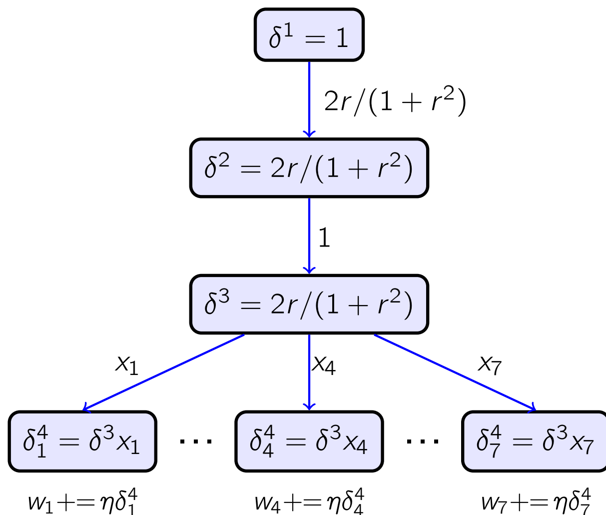
$$\ell(\mathbf{w}, (\mathbf{x}, y)) = \log\left(1 + (\mathbf{w} \cdot \mathbf{x} - y)^2\right)$$



Inference



Backprop



NN Learning Using SGD

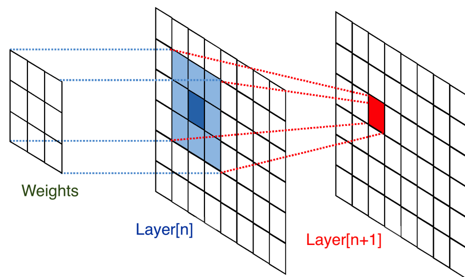
1. Design a network architecture suitable for problem:
   - Number of layers
   - Hidden neurons in each layer
   - Connectivity between layers
   - Activation function

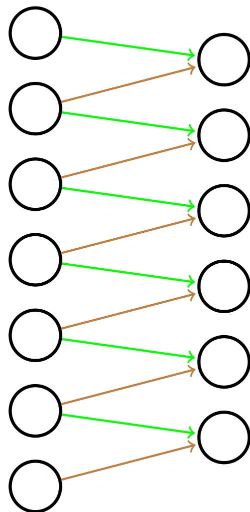
2. Define loss function between y and $\hat{y} = h_{\mathcal{N},\mathrm{w}}(\mathbf{x})$

3. Initialize weights of network

4. SGD – Loop:
   - Obtain a mini-batch of examples
   - Perform forward/inference pass per example
   - Perform backprop for each example
   - Update weights synchronously or asyncronously

Structured Connectivity - ConvNets





- Hidden nodes are connected to subset of nodes in previous layer
- Some of the weights are often shared between neurons

Example: Digit Classification for MNIST

- Task: Handwritten digits recognition
  - Input space: $\mathcal{X} = \{0, 1, \ldots, 255\}^{28\times 28}$
  - Output space: $\mathcal{Y} = \{0, 1, \ldots, 9\}$

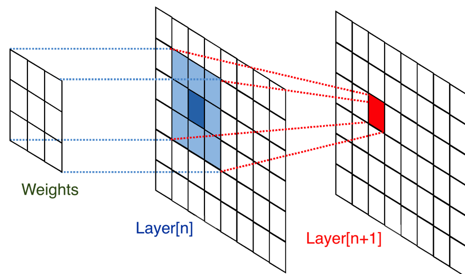
- Multiclass categorization:
  - Hypotheses of the form $f : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|}$
  - Interpret $f(\mathbf{x})$ as scores for each labels
  - Predicted label: $\hat{y} = \arg\max_{j=0}^{9} f_j(\mathbf{x})$

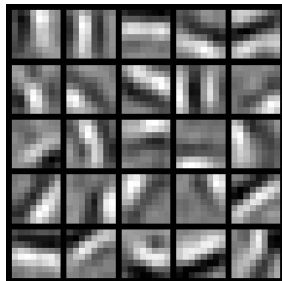
- Architecture:

$$\mathbf{h}_1 = [A\mathbf{x} + \mathbf{b}_1]_+ \quad \mathbf{h}_2 = [B\mathbf{h}_1 + \mathbf{b}_2]_+ \quad \mathbf{f} = C\mathbf{h}_2$$

- Logistic-loss:
  - SoftMax: $\mathbb{P}[i|\mathbf{x}] = \frac{\exp(f_i(\mathrm{x}))}{\sum_j \exp(f_j(\mathrm{x}))}$
  - LogLoss: $\log(\mathbb{P}[y|\mathbf{x}]) = \log\left(\sum_j \exp(f_j(\mathbf{x})\right) - f_y(\mathbf{x})$

Conv Filters for MNIST





Tricks Of Trade

- Input normalization: $\mathbf{x} \to \mathbf{x}/\|\mathbf{x}\|$
- Initialization: normalize by $\sqrt{|\mathrm{in}(v)|}$
- Regularization: weight-decay, dropout
- Mini-batching:
  async with small batches, sync with large batches
- Improved gradient-based methods:
  AdaGrad, Accelerated Gradient Descent, ...
- Learning-rate: $\eta_t = O(|S|/t)$, ...
- Much deeper dive into deep models @ Spring'18 :
  COS-485 "Neural Networks: Theory and Applications"
  Prof. Sebastian Seung