11-695: Competitive Engineering
Feed-forward Neural Networks

Spring 2018

Outline

CarnegieMellon

1. Feed-forward Neural Networks
2. Back-propagation
3. Regularizations
4. Augmented Connections

Neural Nets: A Modeling Perspective

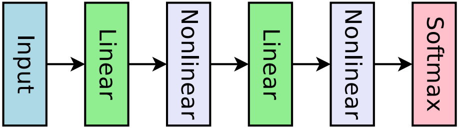
CarnegieMellon

- **The problem:** want to learn a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$
- **The "data" solution:**
  - Collect *data* $\left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), ..., \left(\mathbf{x}^{(N)}, \mathbf{y}^{(N)}\right)\right\}$
  - Come up with a set of *hypotheses* $\mathcal{H} = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_{|\mathcal{H}|}\}$
  - Come up with a loss function $\mathcal{L}$.
  - Find $\mathbf{f}^* = \operatorname{argmin}_{\mathbf{f} \in \mathcal{H}} \mathcal{L}(\mathbf{f})$.

Neural Nets: A Modeling Perspective

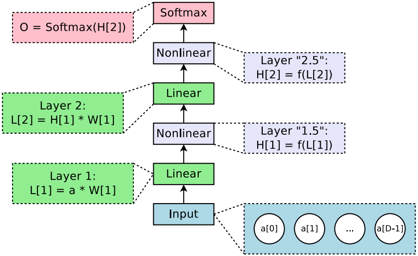
CarnegieMellon

- **The problem:** want to learn a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$
- **The "data" solution:**
  - Collect *data* $\left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), ..., \left(\mathbf{x}^{(N)}, \mathbf{y}^{(N)}\right)\right\}$
  - Come up with a set of *hypotheses* $\mathcal{H} = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_{|\mathcal{H}|}\}$



  - Come up with a loss function $\mathcal{L}$.
  - Find $\mathbf{f}^* = \operatorname{argmin}_{\mathbf{f} \in \mathcal{H}} \mathcal{L}(\mathbf{f})$.

Neurons, Layers, Synapses



- **Input:** $a \in \mathbb{R}^{1 \times M}$. a_i is called a feature.
- **Layer 1:** (Layer 2 and above are similar)
  - $\mathbf{W}^{(1)} \in \mathbb{R}^{M \times |L_1|}$. $\mathbf{W}^{(1)}_{i,j}$ is called a weight, or a synapse, or a parameter.
  - $L^{(1)} \stackrel{\text{def}}{=} a \cdot \mathbf{W}^{(1)}$. $L^{(1)}_i$ is called a pre-activation.
  - $H^{(1)} \stackrel{\text{def}}{=} f(L_1)$. $H^{(1)}_i$ is called a neuron, or an activated value.
    - f is called an activation function, or a non-linearity.
- The number of layers is called the network's depth.

Activation Functions

- Make the network non-linear.

$$
\begin{aligned}
H^{(D)} &= f\left(H^{(D-1)} \cdot \mathbf{W}^{(D)}\right) \\
&= f\left(f\left(H^{(D-2)} \cdot \mathbf{W}^{(D-2)}\right) \cdot \mathbf{W}^{(D)}\right) \\
&= f\left(f\left(\cdots f(a \cdot \mathbf{W}^{(1)}) \cdots\right) \cdot \mathbf{W}^{(D)}\right) \\
&= a \cdot \underbrace{\mathbf{W}^{(1)} \cdot \mathbf{W}^{(2)} \cdots \mathbf{W}^{(D)}}_{\mathbf{W}} \qquad \text{(without } f\text{)} \\
&= a \cdot \mathbf{W}
\end{aligned}
$$

  - Without $\mathbf{f}$, $H^{(\text{final layer})}$ is just a linear transformation of a.
  - Your "neural net" is as weak as a linear model.

Some Popular Activation Functions

CarnegieMellon

- Sigmoid: $f(x) = \dfrac{1}{1 + \exp\{-x\}}$
  - Old-school. Nobody really cares, except:
  - $f(x) \in (0, 1)$. Can be used for *gating* purposes.

Some Popular Activation Functions

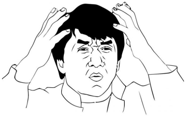
Carnegie Mellon

- Sigmoid: $f(x) = \dfrac{1}{1 + \exp\{-x\}}$
  - Old-school. Nobody really cares, except:
  - $f(x) \in (0, 1)$. Can be used for *gating* purposes.
- Hyperbolic tangent: $f(x) = \tanh(x) = \dfrac{\exp\{x\} - \exp\{-x\}}{\exp\{x\} + \exp\{-x\}}$
  - Better than Sigmoid, but also quite old, so nobody cares, except:
  - $f(x) \in [-1, 1]$. Can be used for *clamping* purposes.

Some Popular Activation Functions

Carnegie Mellon

- Sigmoid: $f(x) = \dfrac{1}{1 + \exp\{-x\}}$
  - Old-school. Nobody really cares, except:
  - $f(x) \in (0, 1)$. Can be used for *gating* purposes.
- Hyperbolic tangent: $f(x) = \tanh(x) = \dfrac{\exp\{x\} - \exp\{-x\}}{\exp\{x\} + \exp\{-x\}}$
  - Better than Sigmoid, but also quite old, so nobody cares, except:
  - $f(x) \in [-1, 1]$. Can be used for *clamping* purposes.
- Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$.
  - Fashionable. Everyone will love it for while. Just use it, or:
  - Many variations: LeakyReLU, PReLU, CReLU, SeLU...



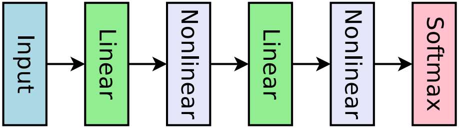
Outline

Carnegie Mellon

Neural Nets So Far

CarnegieMellon

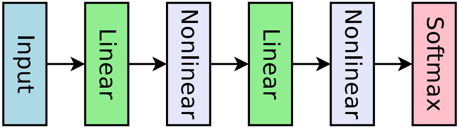
- **The problem:** want to learn a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$
- **The "data" solution:**
  - Collect *data* $\left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), ..., \left(\mathbf{x}^{(N)}, \mathbf{y}^{(N)}\right)\right\}$
  - Come up with a set of *hypotheses* $\mathcal{H} = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_{|\mathcal{H}|}\}$



Neural Nets So Far

CarnegieMellon

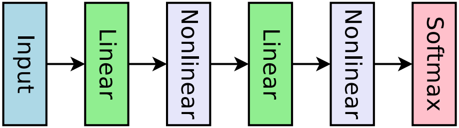
- **The problem:** want to learn a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$
- **The "data" solution:**
  - Collect *data* $\left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), ..., \left(\mathbf{x}^{(N)}, \mathbf{y}^{(N)}\right)\right\}$
  - Come up with a set of *hypotheses* $\mathcal{H} = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_{|\mathcal{H}|}\}$



  - Come up with a loss function $\mathcal{L}$.
    - Can be anything, depending on the problem

Neural Nets So Far

- **The problem:** want to learn a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$
- **The “data” solution:**
  - Collect *data* $\left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}\right), ..., \left(\mathbf{x}^{(N)}, \mathbf{y}^{(N)}\right)\right\}$
  - Come up with a set of *hypotheses* $\mathcal{H} = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_{|\mathcal{H}|}\}$



  - Come up with a loss function $\mathcal{L}$.
    - Can be anything, depending on the problem
  - Find $\mathbf{f}^* = \text{argmin}_{\mathbf{f} \in \mathcal{H}} \mathcal{L}(\mathbf{f})$
    - How?
    - Gradient Descent (or one of its variations)
    - Use back-prop to compute gradients

Neural Nets are Composite Functions

Carnegie Mellon

- Recall that the final layer of a neural net is:

$$\begin{aligned} H^{(D)} &= f\left(H^{(D-1)} \cdot \mathbf{W}^{(D)}\right) \\ &= f\left(f\left(H^{(D-2)} \cdot \mathbf{W}^{(D-2)}\right) \cdot \mathbf{W}^{(D)}\right) \\ &= f\left(f\left(\cdots f(a \cdot \mathbf{W}^{(1)}) \cdots\right) \cdot \mathbf{W}^{(D)}\right) \end{aligned}$$

- Applying the loss function $\mathcal{L}$:

$$\mathcal{L}\left(H^{(D)}\right) = \mathcal{L}\left(f\left(f\left(\cdots f(a \cdot \mathbf{W}^{(1)}) \cdots\right) \cdot \mathbf{W}^{(D)}\right)\right)$$

- We need: $\dfrac{\partial \mathcal{L}}{\partial \mathbf{W}^{(1)}}, \dfrac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}}, \ldots, \dfrac{\partial \mathcal{L}}{\partial \mathbf{W}^{(D)}}$

Chain Rule and the Back-prop *idea*

- If $f : \mathbb{R}^N \to \mathbb{R}$
  - reads: f is a real function of N variables
- and $g_1, g_2, ..., g_N : \mathbb{R} \to \mathbb{R}$, then:

$$\frac{\partial f}{\partial x} = \sum_{i=1}^{N} \frac{\partial f}{\partial g_i} \cdot \frac{\partial g_i}{\partial x}$$

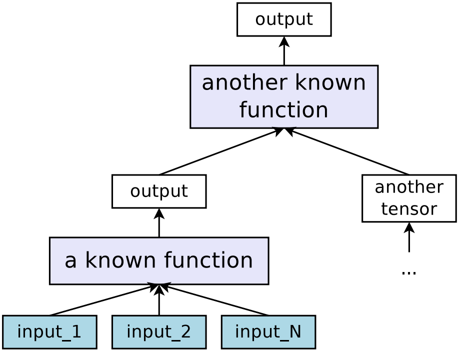
Chain Rule and the Back-prop *idea*

- If $f : \mathbb{R}^N \to \mathbb{R}$
  - reads: f is a real function of N variables
- and $g_1, g_2, ..., g_N : \mathbb{R} \to \mathbb{R}$, then:

$$\frac{\partial f}{\partial x} = \sum_{i=1}^{N} \frac{\partial f}{\partial g_i} \cdot \frac{\partial g_i}{\partial x}$$

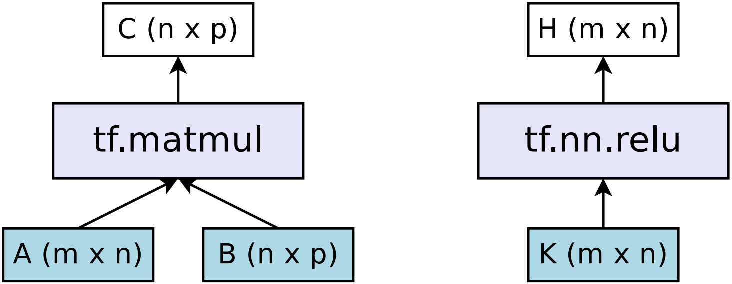
- This is what happens for each neuron at each layer.
- You can reverse this process to compute each $\dfrac{\partial \mathcal{L}}{\partial \mathbf{W}^{(i)}}$

Modern Implementation of Back-prop



- Only need to compute local gradients of the functions

Examples



- For matrix multiplication:

$$\frac{\partial \mathcal{L}}{\partial A} = \frac{\partial \mathcal{L}}{\partial C} \cdot B^\top \quad ; \quad \frac{\partial \mathcal{L}}{\partial B} = A^\top \cdot \frac{\partial \mathcal{L}}{\partial C}$$

- For ReLU:

$$\frac{\partial \mathcal{L}}{\partial K} = \frac{\partial \mathcal{L}}{\partial H} \cdot \mathbf{1}[K \geq 0]$$

Outline

Carnegie Mellon

ℓ_p Regularization

Carnegie Mellon

- Change the activation function $\mathcal{L}$ into

$$\mathcal{L}^{(\mathrm{reg})} = \mathcal{L} + \beta \cdot \sum_{i=1}^{D} \left\| \mathbf{W}^{(i)} \right\|_p^2$$

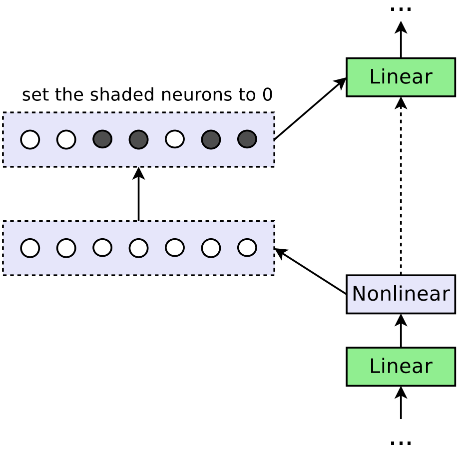
- β has to be chosen
- ℓ_1 regularization encourages sparsity

$$\mathcal{L}^{(\mathrm{reg})} = \mathcal{L} + \beta \cdot \sum_{i=1}^{D} \left| \mathbf{W}^{(i)} \right|$$

- ℓ_2 regularization encourages small weights

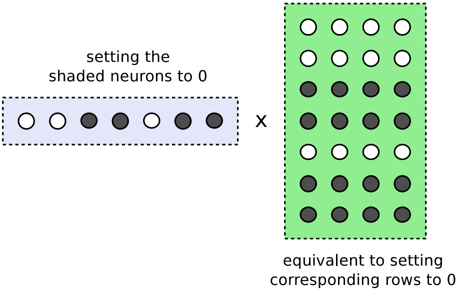
$$\mathcal{L}^{(\mathrm{reg})} = \mathcal{L} + \beta \cdot \sum_{i=1}^{D} \left\| \mathbf{W}^{(i)} \right\|^2$$

DropOut



- At train time: With probability p
  - Randomly set some neurons to 0.
  - Multiply the rest by $1 - p$.

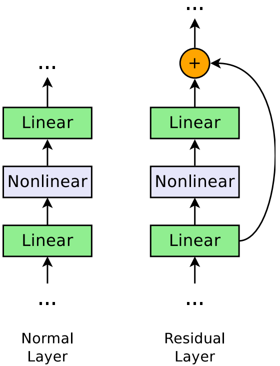
DropOut: A Variational Interpretation



- Normal: $L^{(i)} = H^{(i)} \cdot \mathbf{W}^{(i)}$
- Dropout: $L^{(i)} = \text{Drop}(H^{(i)}) \cdot \mathbf{W}^{(i)}$
- Equivalently: $L^{(i)} = H^{(i)} \cdot g(\mathbf{W}^{(i)})$
- g zeros out some rows and scale the other rows of $\mathbf{W}^{(i)}$.

Outline

Residual Connections (Kaiming He et al, 2015)



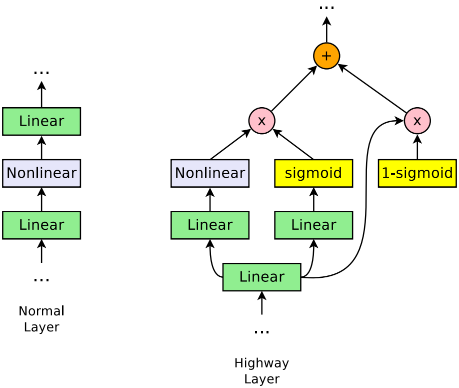
- Normal layer:

$$L^{(i)} \leftarrow L^{(i-1)} \cdot \mathbf{W}^{(i)} \quad ; \quad H^{(i)} \leftarrow f(L^{(i-1)})$$

- Residual layer:

$$L^{(i)} \leftarrow L^{(i-1)} \cdot \mathbf{W}^{(i)} \quad ; \quad H^{(i)} \leftarrow f\left(L^{(i-1)}\right) + L^{(i-1)}$$

Highway Connections (Rupesh Srivastava et al, 2016)

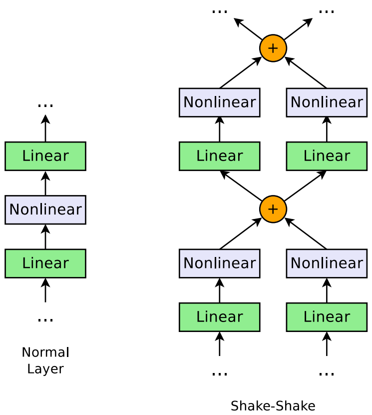


- Highway layer:

$$L^{(i)} \leftarrow L^{(i-1)} \cdot \mathbf{W}^{(i)} \quad ; \quad c^{(i)} \leftarrow \text{Sigmoid}\left(L^{(i-1)} \cdot \mathbf{W}_c^{(i)}\right)$$

$$H^{(i)} \leftarrow c^{(i)} \otimes f\left(L^{(i-1)}\right) + (1 - c^{(i)}) \otimes L^{(i-1)}$$

Shake-Shake (Xavier Gastaldi, 2017)



- Clone the main network
- Use different parameters
- Average each layer