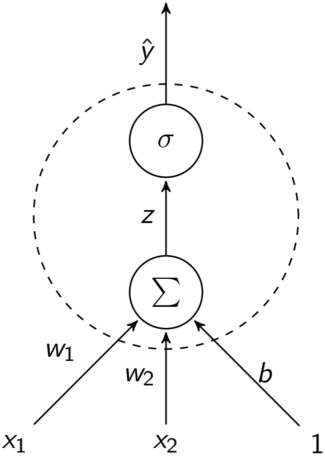
Neural Networks

CS114B Lab 5

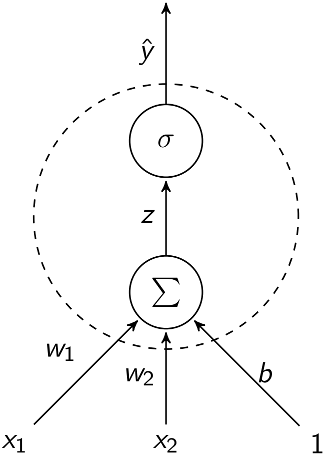
Kenneth Lai

February 17, 2023

Graphical Representation of a Linear Classifier



Graphical Representation of a Neuron



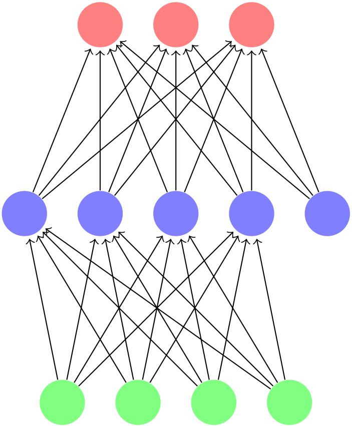
Neural Networks

- (Artificial) neurons are basically linear classifiers!
  - Neurons compute their output based on a linear combination of inputs
  - Outputs are not necessarily classification decisions ($\hat{y}$), but can be inputs to other neurons

Feedforward Neural Networks

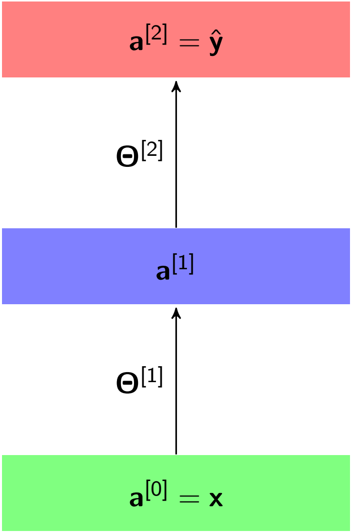
- Suppose that our neurons are grouped into a sequence of layers
- Also suppose that these layers are fully connected (every neuron in one layer is connected to every non-bias neuron in the next layer, and no others)

Feedforward Neural Networks



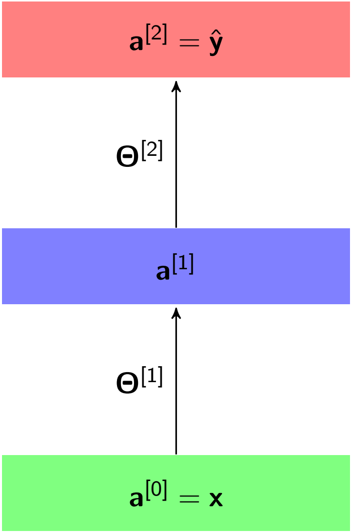
- Output layer
- Hidden layer(s)
- Input layer

Feedforward Neural Networks



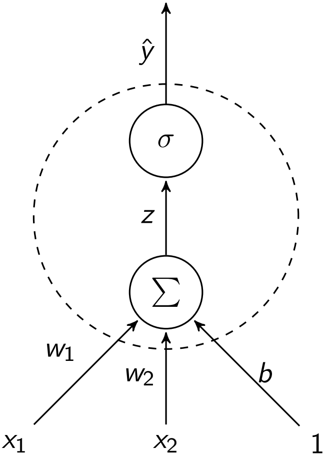
- Output layer
- Hidden layer(s)
- Input layer

Feedforward Neural Networks

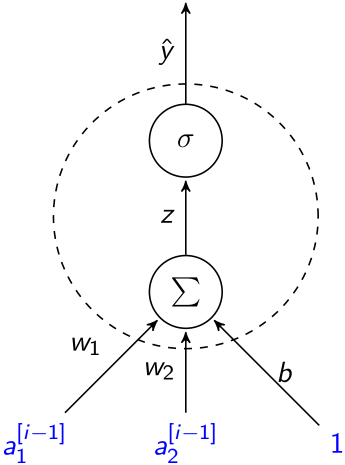


- Output layer
- Hidden layer(s)
- Input layer
  - Outputs of the hidden layers are vectors
    - In particular, they can be seen as intermediate representations of the input

Graphical Representation of a Neuron

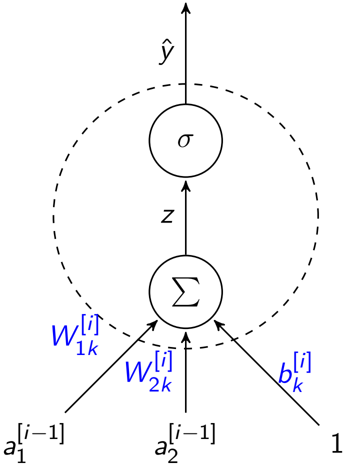


Graphical Representation of a Neuron



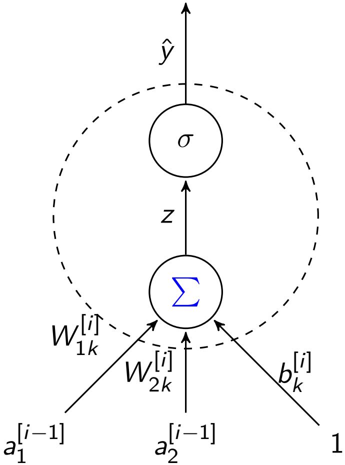
- Inputs to neuron k in layer i = outputs of neurons in layer $i-1$ (and dummy node 1)

Graphical Representation of a Neuron



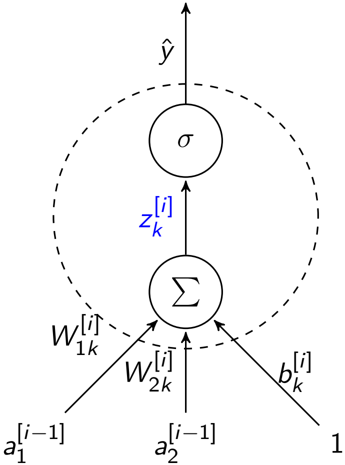
- Weights (and bias term)

Graphical Representation of a Neuron



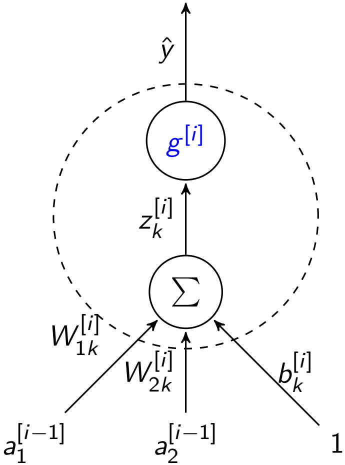
- Sum function $\sum$

Graphical Representation of a Neuron



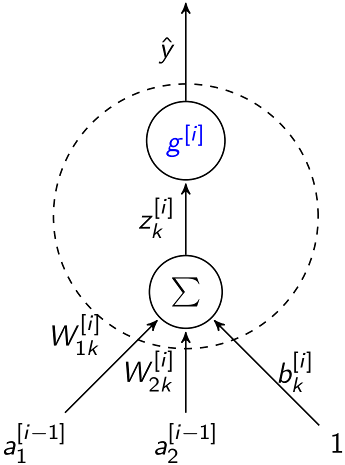
- "Score"

Graphical Representation of a Neuron



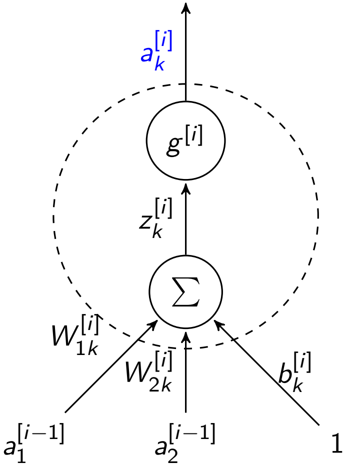
- Activation function $g^{[i]}$

Graphical Representation of a Neuron



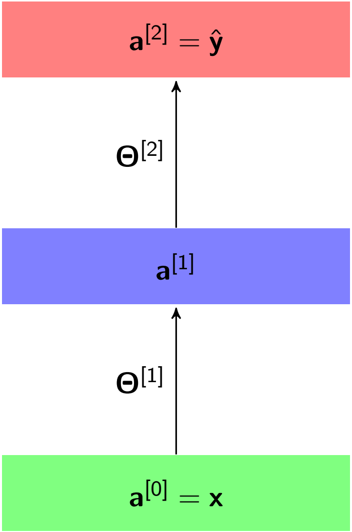
- Activation function $g^{[i]}$
  - Output neuron: logistic or softmax
  - Hidden neuron: typically logistic, tanh, ReLU, etc.

Graphical Representation of a Neuron



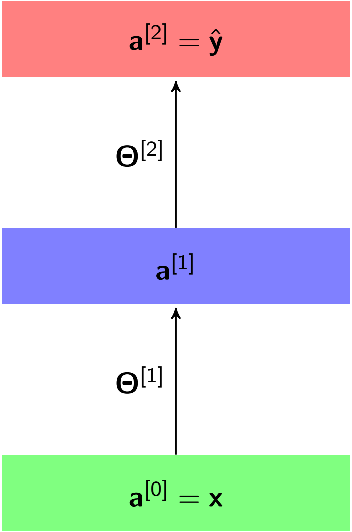
- Activation (output of neuron k in layer i)

Feedforward Neural Networks



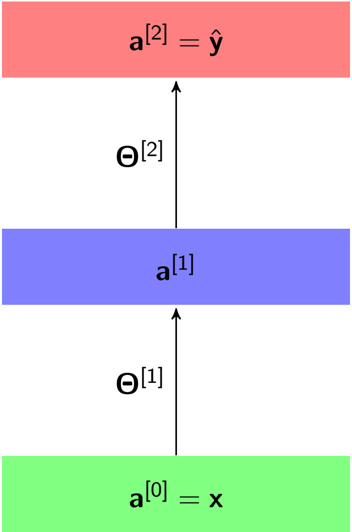
- Output layer
- Hidden layer(s)
- Input layer

Feedforward Neural Networks



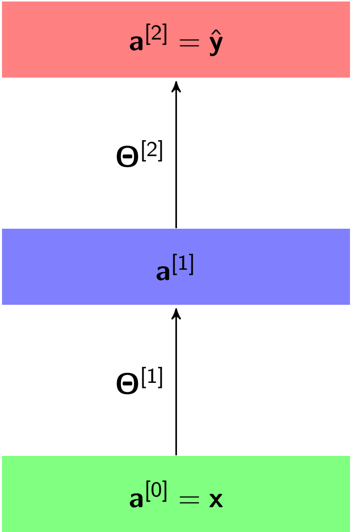
- Output layer
- Hidden layer(s)
- Input layer
- $\mathbf{a}^{[1]} = g^{[1]}(\mathbf{a}^{[0]}\mathbf{\Theta}^{[1]})$
- $\mathbf{a}^{[2]} = g^{[2]}(\mathbf{a}^{[1]}\mathbf{\Theta}^{[2]})$

Feedforward Neural Networks



- Output layer
- Hidden layer(s)
- Input layer
- $\mathbf{a}^{[1]} = g^{[1]}(\mathbf{a}^{[0]}\mathbf{\Theta}^{[1]})$
- $\mathbf{a}^{[2]} = g^{[2]}(\mathbf{a}^{[1]}\mathbf{\Theta}^{[2]})$
- $(\nabla L)^{[2]} = \frac{1}{m}\left((\mathbf{a}^{[1]})^T(\hat{\mathbf{y}} - \mathbf{y})\right)$ (for the cross-entropy loss, logistic or softmax function)
  - Same as for logistic regression, except replace $\mathbf{x}$ with $\mathbf{a}^{[1]}$

Feedforward Neural Networks



- Output layer
- Hidden layer(s)
- Input layer
- $\mathbf{a}^{[1]} = g^{[1]}(\mathbf{a}^{[0]}\mathbf{\Theta}^{[1]})$
- $\mathbf{a}^{[2]} = g^{[2]}(\mathbf{a}^{[1]}\mathbf{\Theta}^{[2]})$
- $(\nabla L)^{[2]} = \frac{1}{m}\left((\mathbf{a}^{[1]})^T(\hat{\mathbf{y}} - \mathbf{y})\right)$ (for the cross-entropy loss, logistic or softmax function)
  - Same as for logistic regression, except replace $\mathbf{x}$ with $\mathbf{a}^{[1]}$
- $(\nabla L)^{[1]} = ?$

Backpropagation

- ...
- (calculus–see supplement slides)
- ...

Backpropagation

- ...
- (calculus–see supplement slides)
- ...
- For all layers i:
  - $(\nabla L)^{[i]} = \dfrac{1}{m}\left((\mathbf{a}^{[i-1]})^T \delta^{[i]}\right)$
    - Note that $\mathbf{a}^{[i-1]}$ includes the dummy feature 1
    - Let $\delta^{[i]}$ be the "error" in layer i

Backpropagation

- ...
- (calculus–see supplement slides)
- ...
- For all layers i:
  - $(\nabla L)^{[i]} = \frac{1}{m}\left((\mathbf{a}^{[i-1]})^T \delta^{[i]}\right)$
    - Note that $\mathbf{a}^{[i-1]}$ includes the dummy feature 1
    - Let $\delta^{[i]}$ be the "error" in layer i
- For an output layer $\mathcal{L}$:
  - $\delta^{[\mathcal{L}]} = \hat{\mathbf{y}} - \mathbf{y}$
    (for the cross-entropy loss, logistic or softmax function)

Backpropagation

- ...
- (calculus–see supplement slides)
- ...
- For all layers i:
  - $(\nabla L)^{[i]} = \frac{1}{m}\left((\mathbf{a}^{[i-1]})^T \delta^{[i]}\right)$
    - Note that $\mathbf{a}^{[i-1]}$ includes the dummy feature 1
    - Let $\delta^{[i]}$ be the "error" in layer i
- For an output layer $\mathcal{L}$:
  - $\delta^{[\mathcal{L}]} = \hat{\mathbf{y}} - \mathbf{y}$
    (for the cross-entropy loss, logistic or softmax function)
- For a hidden layer i:
  - $\delta^{[i]} = (\delta^{[i+1]}(\mathbf{W}^{[i+1]})^T) \odot g'^{[i]}(\mathbf{z}^{[i]})$
    - Let $\odot$ denote the element-wise (Hadamard) product
    - Also note the use of $\mathbf{W}$ (rather than $\boldsymbol{\Theta}$)
    - $\mathbf{W}$ does not include the bias $\mathbf{b}$

Backpropagation

- We can compute $\delta^{[\mathcal{L}]}$ for an output layer $\mathcal{L}$
- Key idea: if we have already computed $\delta^{[i+1]}$ for some layer $i+1$, then we can use it to calculate $\delta^{[i]}$ for the previous layer i

Backpropagation

- For each layer i:
  - Initialize parameters $\mathbf{\Theta}^{[i]} = \mathbf{W}^{[i]}, \mathbf{b}^{[i]}$ randomly (note: not $\mathbf{0}$)

Backpropagation

- For each layer i:
  - Initialize parameters $\mathbf{\Theta}^{[i]} = \mathbf{W}^{[i]}, \mathbf{b}^{[i]}$ randomly (note: not $\mathbf{0}$)
- At each time step t:
  - For each layer i, starting from the output and working backwards:
    - Compute gradient $(\nabla L)^{[i]}$

Backpropagation

- For each layer i:
  - Initialize parameters $\mathbf{\Theta}^{[i]} = \mathbf{W}^{[i]}, \mathbf{b}^{[i]}$ randomly (note: not $\mathbf{0}$)
- At each time step t:
  - For each layer i, starting from the output and working backwards:
    - Compute gradient $(\nabla L)^{[i]}$
  - For each layer i:
    - Move in direction of negative gradient
- $\mathbf{\Theta}^{[i]}_{t+1} = \mathbf{\Theta}^{[i]}_{t} - \eta(\nabla L)^{[i]}$

Backpropagation

- For each layer i:
  - Initialize parameters $\boldsymbol{\Theta}^{[i]} = \mathbf{W}^{[i]}, \mathbf{b}^{[i]}$ randomly (note: not $\mathbf{0}$)
- At each time step t:
  - For each layer i, starting from the output and working backwards:
    - Compute gradient $(\nabla L)^{[i]}$
  - For each layer i:
    - Move in direction of negative gradient
- $\boldsymbol{\Theta}^{[i]}_{t+1} = \boldsymbol{\Theta}^{[i]}_{t} - \eta(\nabla L)^{[i]}$
- Because L is not necessarily convex anymore, we are not guaranteed to reach a global minimum
  - But it works well enough in practice

Further Reading

- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.
- Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press USA.