From Logistic Regression to Neural Networks (Part 2)

CS114B Lab 5

Kenneth Lai

March 5, 2021

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

Multinomial Logistic Regression

- Separate weights and bias terms for each class c

$$z_c = \left(\sum_{j=1}^{n} w_{jc} x_j \right) + b_c = \mathbf{w}_c \cdot \mathbf{x} + b_c$$

- $\text{softmax}(z_c) = \frac{e^{z_c}}{\sum_{k=1}^{K} e^{z_k}} = P(y = c|\mathbf{x}) = \hat{y}_c$

Multinomial Logistic Regression

- Cross-entropy loss $L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$

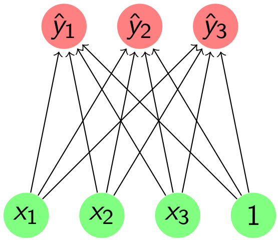
Multinomial Logistic Regression

- Cross-entropy loss $L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$
- Gradient ∇L becomes a matrix, where

Multinomial Logistic Regression

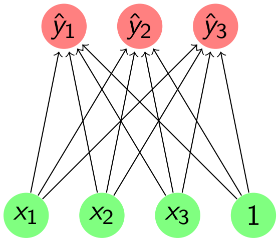
- Cross-entropy loss $L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log \hat{y}_k$
- Gradient ∇L becomes a matrix, where
  - $\dfrac{\partial L}{\partial w_{jk}} = (\hat{y}_k - y_k)x_j$
  - $\dfrac{\partial L}{\partial b_k} = \hat{y}_k - y_k$

Multinomial Logistic Regression



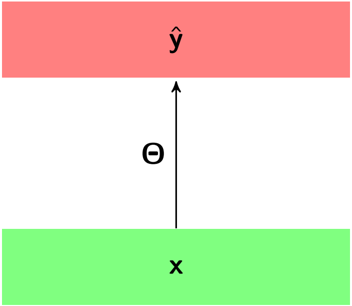
Multinomial Logistic Regression

- Output layer
- Input layer



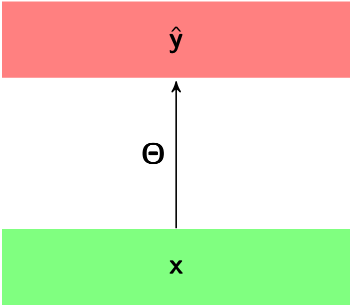
Multinomial Logistic Regression

- Output layer
- Input layer

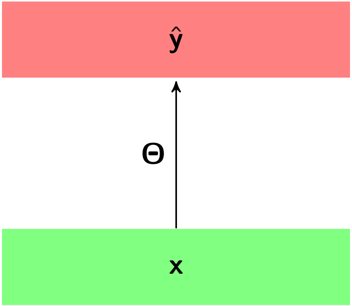


Multinomial Logistic Regression

- Output layer
- Input layer
- $\hat{\mathbf{y}} = \text{softmax}(\mathbf{x} \cdot \boldsymbol{\Theta})$

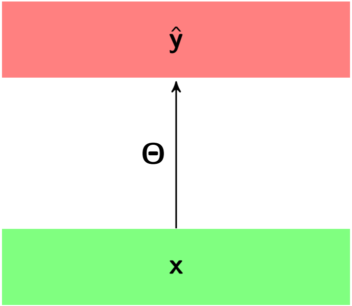


Multinomial Logistic Regression



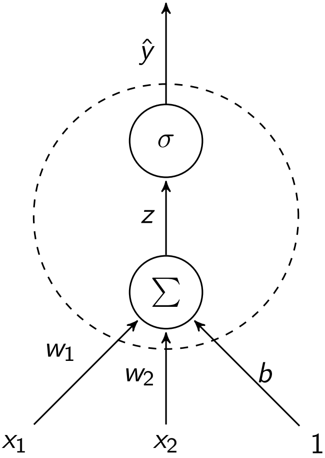
- Output layer
- Input layer
- $\hat{\mathbf{y}} = \text{softmax}(\mathbf{x} \cdot \boldsymbol{\Theta})$
  - We will assume that the dummy feature 1 is part of $\mathbf{x}$

Multinomial Logistic Regression

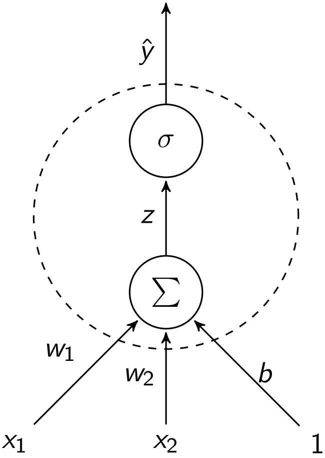


- Output layer
- Input layer
- $\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{x} \cdot \boldsymbol{\Theta})$
  - We will assume that the dummy feature 1 is part of $\mathbf{x}$
- $\nabla L = \frac{1}{m}\left(\mathbf{x}^T \cdot (\hat{\mathbf{y}} - \mathbf{y})\right)$ (for the cross-entropy loss)

Graphical Representation of Logistic Regression



Graphical Representation of a Neuron



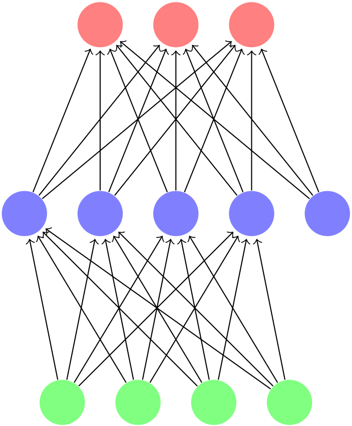
Feedforward Neural Networks

- Suppose that our neurons are grouped into a sequence of layers

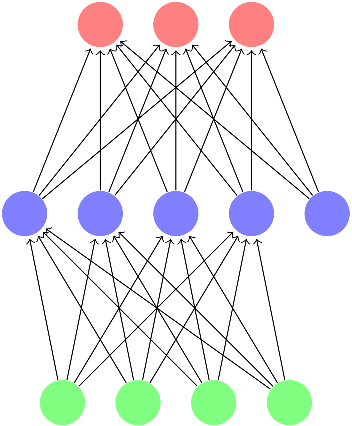
Feedforward Neural Networks

- 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

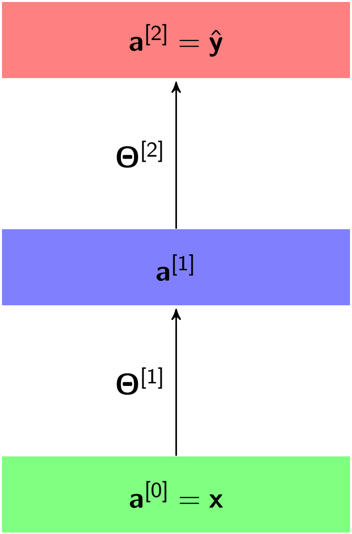


Feedforward Neural Networks



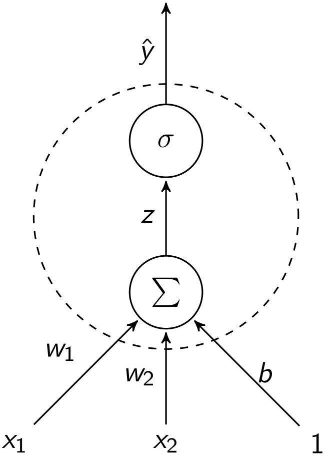
- Output layer
- Hidden layer(s)
- Input layer

Feedforward Neural Networks

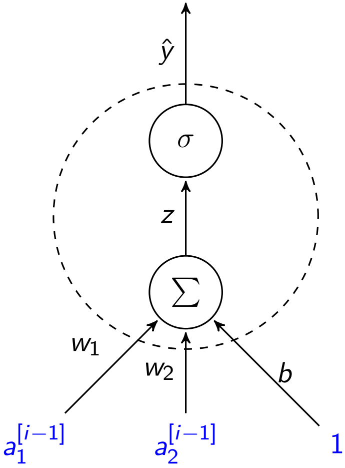


- Output layer
- Hidden layer(s)
- Input layer

Graphical Representation of a Neuron

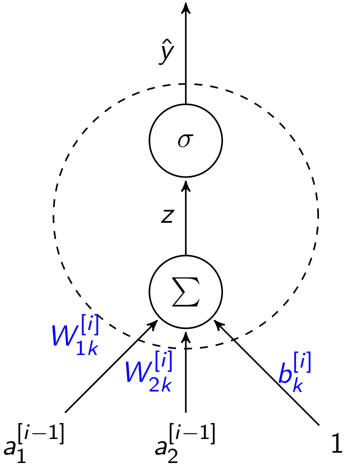


Graphical Representation of a Neuron



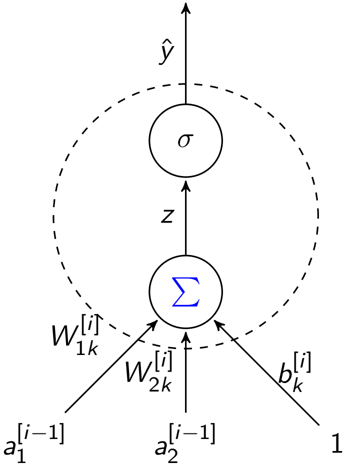
- Inputs to neuron k in layer i = outputs of neurons in layer $i-1$ (and dummy node 1)

Graphical Representation of a Neuron



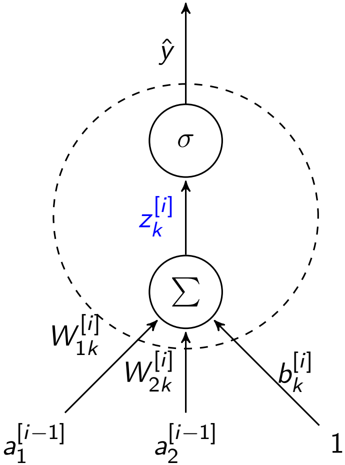
- Weights (and bias term)

Graphical Representation of a Neuron



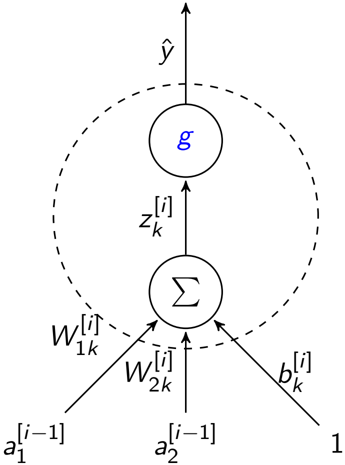
- Sum function $\sum$

Graphical Representation of a Neuron



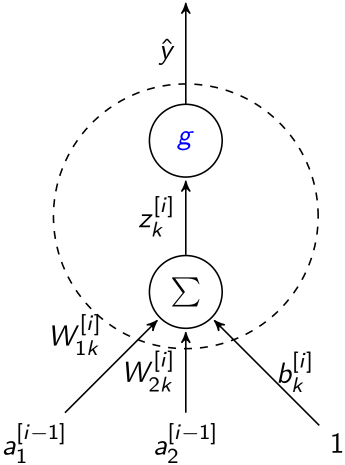
- "Score"

Graphical Representation of a Neuron



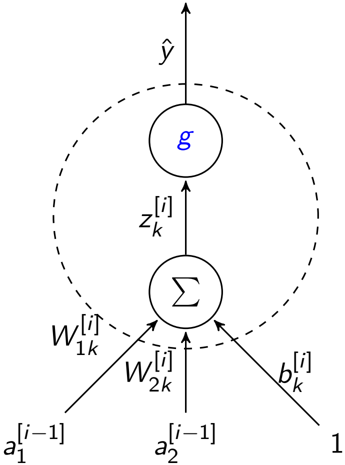
- Activation function g

Graphical Representation of a Neuron



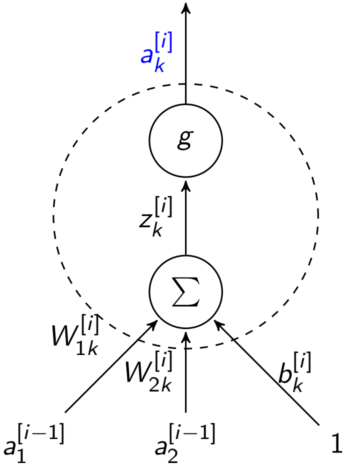
- Activation function g
  - Output neuron: logistic or softmax

Graphical Representation of a Neuron



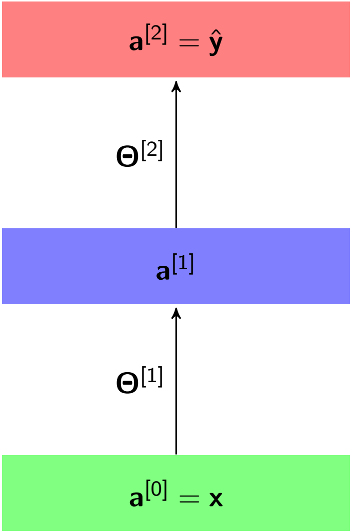
- Activation function g
  - Output neuron: logistic or softmax
  - Hidden neuron: typically logistic, tanh, ReLU, etc.

Graphical Representation of a Neuron



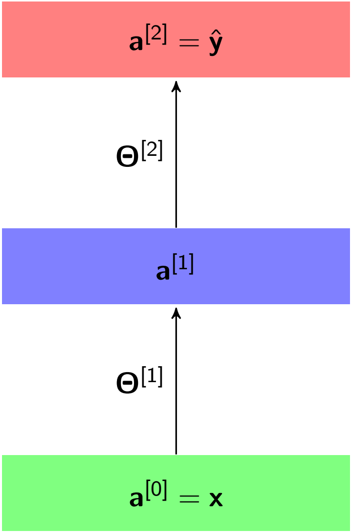
- Activation (output of neuron k in layer i)

Feedforward Neural Networks



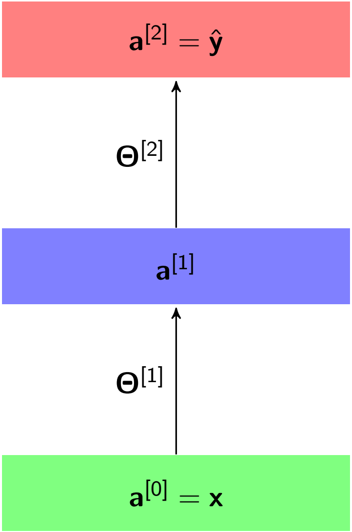
- Output layer
- Hidden layer(s)
- Input layer

Feedforward Neural Networks



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

Feedforward Neural Networks



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

Feedforward Neural Networks

- Warning! A note on notation:

Feedforward Neural Networks

- Warning! A note on notation:
  - These slides, TensorFlow, Keras, MXNet:

Feedforward Neural Networks

- Warning! A note on notation:
  - These slides, TensorFlow, Keras, MXNet:
    - Weight matrices have shapes (inputs, outputs)
    - $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{\Theta}^{[i]})$
      (or $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{W}^{[i]} + \mathbf{b}^{[i]})$)

Feedforward Neural Networks

- Warning! A note on notation:
  - These slides, TensorFlow, Keras, MXNet:
    - Weight matrices have shapes (inputs, outputs)
    - $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{\Theta}^{[i]})$
      (or $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{W}^{[i]} + \mathbf{b}^{[i]})$)
  - Textbooks:

Feedforward Neural Networks

- Warning! A note on notation:
  - These slides, TensorFlow, Keras, MXNet:
    - Weight matrices have shapes (inputs, outputs)
    - $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{\Theta}^{[i]})$
      (or $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{W}^{[i]} + \mathbf{b}^{[i]})$)
  - Textbooks:
    - Weight matrices have shapes (outputs, inputs)
    - $\mathbf{a}^{[i]} = g(\mathbf{\Theta}^{[i]} \cdot \mathbf{a}^{[i-1]})$
      (or $\mathbf{a}^{[i]} = g(\mathbf{W}^{[i]} \cdot \mathbf{a}^{[i-1]} + \mathbf{b}^{[i]})$)

Feedforward Neural Networks

- Warning! A note on notation:
  - These slides, TensorFlow, Keras, MXNet:
    - Weight matrices have shapes (inputs, outputs)
    - $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{\Theta}^{[i]})$
      (or $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{W}^{[i]} + \mathbf{b}^{[i]})$)
  - Textbooks:
    - Weight matrices have shapes (outputs, inputs)
    - $\mathbf{a}^{[i]} = g(\mathbf{\Theta}^{[i]} \cdot \mathbf{a}^{[i-1]})$
      (or $\mathbf{a}^{[i]} = g(\mathbf{W}^{[i]} \cdot \mathbf{a}^{[i-1]} + \mathbf{b}^{[i]})$)
  - PyTorch:

Feedforward Neural Networks

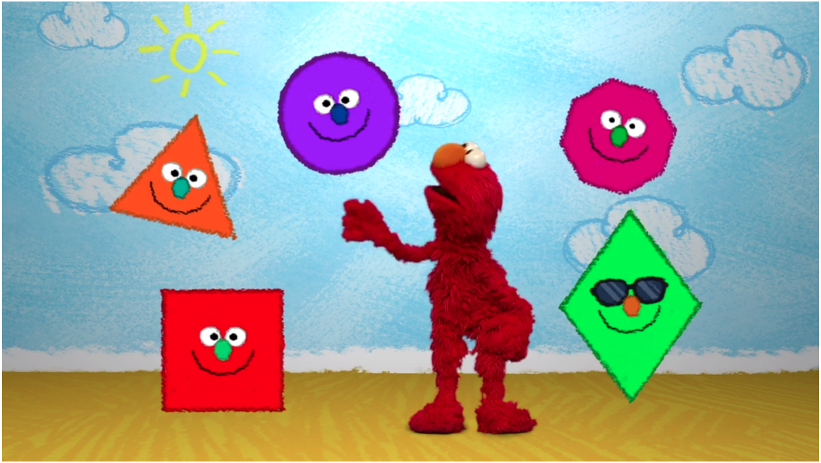
- Warning! A note on notation:
  - These slides, TensorFlow, Keras, MXNet:
    - Weight matrices have shapes (inputs, outputs)
    - $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{\Theta}^{[i]})$
      (or $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot \mathbf{W}^{[i]} + \mathbf{b}^{[i]})$)
  - Textbooks:
    - Weight matrices have shapes (outputs, inputs)
    - $\mathbf{a}^{[i]} = g(\mathbf{\Theta}^{[i]} \cdot \mathbf{a}^{[i-1]})$
      (or $\mathbf{a}^{[i]} = g(\mathbf{W}^{[i]} \cdot \mathbf{a}^{[i-1]} + \mathbf{b}^{[i]})$)
  - PyTorch:
    - Call `torch.nn.Linear(inputs, outputs)`
    - Store weights as shape (outputs, inputs)
    - $\mathbf{a}^{[i]} = g(\mathbf{a}^{[i-1]} \cdot (\mathbf{W}^{[i]})^T + \mathbf{b}^{[i]})$

General Advice

- Know your shapes!

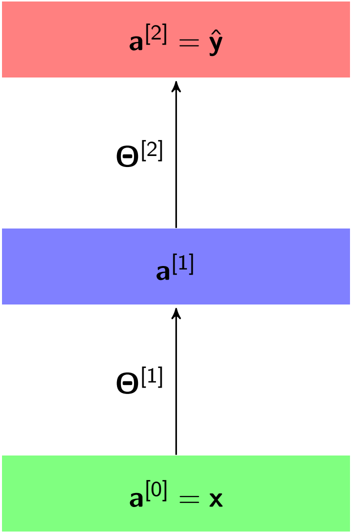
General Advice

- Know your shapes!



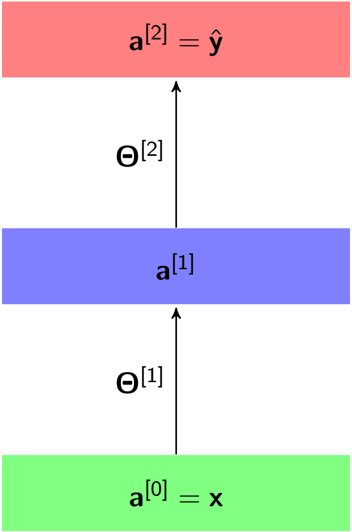
Source

Feedforward Neural Networks



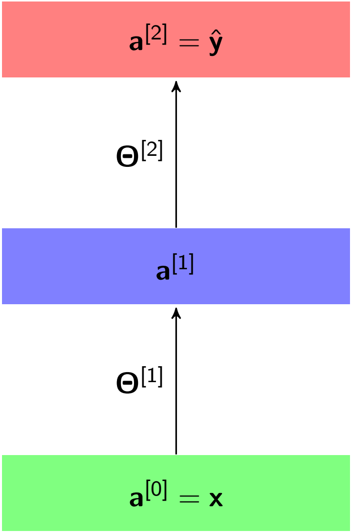
- Output layer
- Hidden layer(s)
- Input layer
- $\mathbf{a}^{[1]} = g(\mathbf{a}^{[0]} \cdot \mathbf{\Theta}^{[1]})$
- $\mathbf{a}^{[2]} = g(\mathbf{a}^{[1]} \cdot \mathbf{\Theta}^{[2]})$

Feedforward Neural Networks



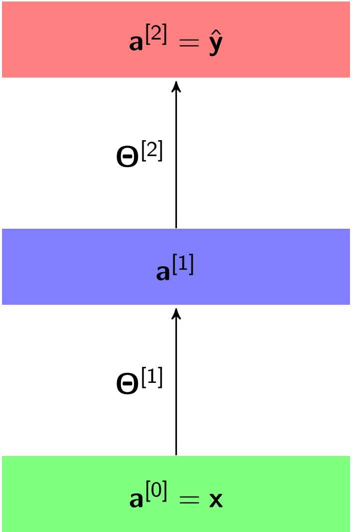
- Output layer
- Hidden layer(s)
- Input layer
- $\mathbf{a}^{[1]} = g(\mathbf{a}^{[0]} \cdot \mathbf{\Theta}^{[1]})$
- $\mathbf{a}^{[2]} = g(\mathbf{a}^{[1]} \cdot \mathbf{\Theta}^{[2]})$
- $(\nabla L)^{[2]} = \frac{1}{m}\Big((\mathbf{a}^{[1]})^T \cdot (\hat{\mathbf{y}} - \mathbf{y})\Big)$
  (for the cross-entropy loss, logistic or softmax function)

Feedforward Neural Networks



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

Advanced: Gradients in Feedforward Neural Networks

- We want to compute $\dfrac{\partial L}{\partial W_{jk}^{[i]}}$

Advanced: Gradients in Feedforward Neural Networks

- We want to compute $\dfrac{\partial L}{\partial W_{jk}^{[i]}}$
- Chain Rule of calculus: $\dfrac{dy}{dx} = \dfrac{dy}{dz}\dfrac{dz}{dx}$

Advanced: Gradients in Feedforward Neural Networks

- We want to compute $\frac{\partial L}{\partial W_{jk}^{[i]}}$
- Chain Rule of calculus: $\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$
- Looking at the graph: $\frac{\partial L}{\partial W_{jk}^{[i]}} = \frac{\partial L}{\partial a_k^{[i]}} \frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} \frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}}$

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $$\frac{\partial L}{\partial W_{jk}^{[i]}} = \frac{\partial L}{\partial a_k^{[i]}} \frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} \frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}}$$

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $\frac{\partial L}{\partial W_{jk}^{[i]}} = \frac{\partial L}{\partial a_k^{[i]}} \frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} a_j^{[i-1]}$
    - $\frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $\frac{\partial L}{\partial W_{jk}^{[i]}} = \frac{\partial L}{\partial a_k^{[i]}} g'(z_k^{[i]}) a_j^{[i-1]}$
    - $\frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$
    - $\frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} = g'(z_k^{[i]})$

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $\frac{\partial L}{\partial W_{jk}^{[i]}} = \frac{\partial L}{\partial a_k^{[i]}} g'(z_k^{[i]}) a_j^{[i-1]}$
    - $\frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$
    - $\frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} = g'(z_k^{[i]})$
    - Let $g'(z_k^{[i]})$ be the derivative of the activation function

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $\frac{\partial L}{\partial W_{jk}^{[i]}} = \frac{\partial L}{\partial a_k^{[i]}} g'(z_k^{[i]}) a_j^{[i-1]}$
    - $\frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$
    - $\frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} = g'(z_k^{[i]})$
    - Let $g'(z_k^{[i]})$ be the derivative of the activation function
    - For the logistic function: $g'(z_k^{[i]}) = a_k^{[i]}(1 - a_k^{[i]})$
    - ...

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $\frac{\partial L}{\partial W_{jk}^{[i]}} = \frac{\partial L}{\partial a_k^{[i]}} g'(z_k^{[i]}) a_j^{[i-1]}$
    - $\frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$
    - $\frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} = g'(z_k^{[i]})$
    - $\frac{\partial L}{\partial a_k^{[i]}} = ?$

Advanced: Gradients in Feedforward Neural Networks

- Note that for a hidden neuron, $a_k^{[i]}$ is an input to each non-bias neuron ℓ in layer $i + 1$

Advanced: Gradients in Feedforward Neural Networks

- Note that for a hidden neuron, $a_k^{[i]}$ is an input to each non-bias neuron ℓ in layer $i+1$
- Chain Rule of multivariable calculus:

$$\frac{df(g_1(x), \ldots, g_n(x))}{dx} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i(x)} \frac{dg_i(x)}{dx}$$

Advanced: Gradients in Feedforward Neural Networks

- Note that for a hidden neuron, $a_k^{[i]}$ is an input to each non-bias neuron ℓ in layer $i+1$
- Chain Rule of multivariable calculus:

$$\frac{df(g_1(x), \ldots, g_n(x))}{dx} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i(x)} \frac{dg_i(x)}{dx}$$

- Express L as a function of $z_\ell^{[i+1]}$: $\dfrac{\partial L}{\partial a_k^{[i]}} = \sum_\ell \dfrac{\partial L}{\partial z_\ell^{[i+1]}} \dfrac{\partial z_\ell^{[i+1]}}{\partial a_k^{[i]}}$

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $$\frac{\partial L}{\partial W_{jk}^{[i]}} = \left(\sum_{\ell} \frac{\partial L}{\partial z_{\ell}^{[i+1]}} \frac{\partial z_{\ell}^{[i+1]}}{\partial a_k^{[i]}} \right) g'(z_k^{[i]}) a_j^{[i-1]}$$
    - $$\frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$$
    - $$\frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} = g'(z_k^{[i]})$$
    - $$\frac{\partial L}{\partial a_k^{[i]}} = \sum_{\ell} \frac{\partial L}{\partial z_{\ell}^{[i+1]}} \frac{\partial z_{\ell}^{[i+1]}}{\partial a_k^{[i]}}$$

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $\dfrac{\partial L}{\partial W_{jk}^{[i]}} = \left(\sum_{\ell} \dfrac{\partial L}{\partial z_{\ell}^{[i+1]}} W_{k\ell}^{[i+1]} \right) g'(z_k^{[i]}) a_j^{[i-1]}$
    - $\dfrac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$
    - $\dfrac{\partial a_k^{[i]}}{\partial z_k^{[i]}} = g'(z_k^{[i]})$
    - $\dfrac{\partial L}{\partial a_k^{[i]}} = \sum_{\ell} \dfrac{\partial L}{\partial z_{\ell}^{[i+1]}} \dfrac{\partial z_{\ell}^{[i+1]}}{\partial a_k^{[i]}}$
    - $\dfrac{\partial z_{\ell}^{[i+1]}}{\partial a_k^{[i]}} = W_{k\ell}^{[i+1]}$

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $$\frac{\partial L}{\partial W_{jk}^{[i]}} = \left(\sum_{\ell} \delta_{\ell}^{[i+1]} W_{k\ell}^{[i+1]} \right) g'(z_k^{[i]}) a_j^{[i-1]}$$
    - $$\frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$$
    - $$\frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} = g'(z_k^{[i]})$$
    - $$\frac{\partial L}{\partial a_k^{[i]}} = \sum_{\ell} \frac{\partial L}{\partial z_{\ell}^{[i+1]}} \frac{\partial z_{\ell}^{[i+1]}}{\partial a_k^{[i]}}$$
    - $$\frac{\partial z_{\ell}^{[i+1]}}{\partial a_k^{[i]}} = W_{k\ell}^{[i+1]}$$
    - $$\frac{\partial L}{\partial z_{\ell}^{[i+1]}} = \delta_{\ell}^{[i+1]}$$

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $\frac{\partial L}{\partial W_{jk}^{[i]}} = \left(\sum_\ell \delta_\ell^{[i+1]} W_{k\ell}^{[i+1]} \right) g'(z_k^{[i]}) a_j^{[i-1]}$
    - $\frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$
    - $\frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} = g'(z_k^{[i]})$
    - $\frac{\partial L}{\partial a_k^{[i]}} = \sum_\ell \frac{\partial L}{\partial z_\ell^{[i+1]}} \frac{\partial z_\ell^{[i+1]}}{\partial a_k^{[i]}}$
    - $\frac{\partial z_\ell^{[i+1]}}{\partial a_k^{[i]}} = W_{k\ell}^{[i+1]}$
    - $\frac{\partial L}{\partial z_\ell^{[i+1]}} = \delta_\ell^{[i+1]}$
    - Let $\delta_\ell^{[i+1]}$ be the "error" in neuron ℓ in layer $i+1$

Advanced: Gradients in Feedforward Neural Networks

- For a hidden neuron:
  - $$\frac{\partial L}{\partial W_{jk}^{[i]}} = \left(\sum_{\ell} \delta_{\ell}^{[i+1]} W_{k\ell}^{[i+1]} \right) g'(z_k^{[i]}) a_j^{[i-1]}$$
    - $\frac{\partial z_k^{[i]}}{\partial W_{jk}^{[i]}} = a_j^{[i-1]}$
    - $\frac{\partial a_k^{[i]}}{\partial z_k^{[i]}} = g'(z_k^{[i]})$
    - $\frac{\partial L}{\partial a_k^{[i]}} = \sum_{\ell} \frac{\partial L}{\partial z_{\ell}^{[i+1]}} \frac{\partial z_{\ell}^{[i+1]}}{\partial a_k^{[i]}}$
    - $\frac{\partial z_{\ell}^{[i+1]}}{\partial a_k^{[i]}} = W_{k\ell}^{[i+1]}$
    - $\frac{\partial L}{\partial z_{\ell}^{[i+1]}} = \delta_{\ell}^{[i+1]}$
    - Let $\delta_{\ell}^{[i+1]}$ be the "error" in neuron ℓ in layer $i+1$
    - What is $\delta_{\ell}^{[i+1]}$?

Advanced: Backpropagation

- We can compute $\dfrac{\partial L}{\partial W_{k\ell}^{[\mathcal{L}]}} = \dfrac{\partial L}{\partial a_{\ell}^{[\mathcal{L}]}} \dfrac{\partial a_{\ell}^{[\mathcal{L}]}}{\partial z_{\ell}^{[\mathcal{L}]}} \dfrac{\partial z_{\ell}^{[\mathcal{L}]}}{\partial W_{k\ell}^{[\mathcal{L}]}}$ for an output neuron ℓ in layer $\mathcal{L}$

Advanced: Backpropagation

- We can compute $\dfrac{\partial L}{\partial W_{k\ell}^{[\mathcal{L}]}} = \dfrac{\partial L}{\partial a_\ell^{[\mathcal{L}]}} \dfrac{\partial a_\ell^{[\mathcal{L}]}}{\partial z_\ell^{[\mathcal{L}]}} \dfrac{\partial z_\ell^{[\mathcal{L}]}}{\partial W_{k\ell}^{[\mathcal{L}]}}$ for an output neuron ℓ in layer $\mathcal{L}$
- If we have already computed $\dfrac{\partial L}{\partial W_{k\ell}^{[i+1]}}$ for some neuron ℓ in layer $i+1$, then we have also computed $\delta_\ell^{[i+1]} = \dfrac{\partial L}{\partial z_\ell^{[i+1]}} = \dfrac{\partial L}{\partial a_\ell^{[i+1]}} \dfrac{\partial a_\ell^{[i+1]}}{\partial z_\ell^{[i+1]}}$

Advanced: Backpropagation

- We can compute $\dfrac{\partial L}{\partial W_{k\ell}^{[\mathcal{L}]}} = \dfrac{\partial L}{\partial a_\ell^{[\mathcal{L}]}} \dfrac{\partial a_\ell^{[\mathcal{L}]}}{\partial z_\ell^{[\mathcal{L}]}} \dfrac{\partial z_\ell^{[\mathcal{L}]}}{\partial W_{k\ell}^{[\mathcal{L}]}}$ for an output neuron ℓ in layer $\mathcal{L}$
- If we have already computed $\dfrac{\partial L}{\partial W_{k\ell}^{[i+1]}}$ for some neuron ℓ in layer $i+1$, then we have also computed $\delta_\ell^{[i+1]} = \dfrac{\partial L}{\partial z_\ell^{[i+1]}} = \dfrac{\partial L}{\partial a_\ell^{[i+1]}} \dfrac{\partial a_\ell^{[i+1]}}{\partial z_\ell^{[i+1]}}$
- We can then use $\delta_\ell^{[i+1]}$ to calculate $\dfrac{\partial L}{\partial W_{jk}^{[i]}} = \left(\sum_\ell \delta_\ell^{[i+1]} W_{k\ell}^{[i+1]} \right) g'(z_k^{[i]}) a_j^{[i-1]}$ for the previous neurons k in layer i

Advanced: Backpropagation

- $$\frac{\partial L}{\partial W_{jk}^{[i]}} = \left(\sum_{\ell} \delta_{\ell}^{[i+1]} W_{k\ell}^{[i+1]} \right) g'(z_k^{[i]}) a_j^{[i-1]}$$

Advanced: Backpropagation

- $$\frac{\partial L}{\partial W_{jk}^{[i]}} = \left(\sum_{\ell} \delta_{\ell}^{[i+1]} W_{k\ell}^{[i+1]} \right) g'(z_k^{[i]}) a_j^{[i-1]}$$
- $$\frac{\partial L}{\partial b_k^{[i]}} = \left(\sum_{\ell} \delta_{\ell}^{[i+1]} W_{k\ell}^{[i+1]} \right) g'(z_k^{[i]})$$

Backpropagation

- For all layers i:
  - $(\nabla L)^{[i]} = \frac{1}{m}\left((\mathbf{a}^{[i-1]})^T \cdot \delta^{[i]}\right)$

Backpropagation

- For all layers i:
  - $(\nabla L)^{[i]} = \frac{1}{m}\left((\mathbf{a}^{[i-1]})^T \cdot \delta^{[i]}\right)$
- For an output layer $\mathcal{L}$:
  - $\delta^{[\mathcal{L}]} = \hat{\mathbf{y}} - \mathbf{y}$
    (for the cross-entropy loss, logistic or softmax function)

Backpropagation

- For all layers i:
  - $(\nabla L)^{[i]} = \frac{1}{m}\left((\mathbf{a}^{[i-1]})^T \cdot \delta^{[i]}\right)$
- 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:
  - $\delta^{[i]} = (\delta^{[i+1]} \cdot (\mathbf{W}^{[i+1]})^T) \odot g'(\mathbf{z}^{[i]})$

Backpropagation

- For all layers i:
  - $(\nabla L)^{[i]} = \frac{1}{m}\left((\mathbf{a}^{[i-1]})^T \cdot \delta^{[i]}\right)$
- 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:
  - $\delta^{[i]} = (\delta^{[i+1]} \cdot (\mathbf{W}^{[i+1]})^T) \odot g'(\mathbf{z}^{[i]})$
    - Let $\odot$ denote the element-wise (Hadamard) product

Backpropagation

- For all layers i:
  - $(\nabla L)^{[i]} = \dfrac{1}{m}\left((\mathbf{a}^{[i-1]})^T \cdot \delta^{[i]}\right)$
- 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:
  - $\delta^{[i]} = (\delta^{[i+1]} \cdot (\mathbf{W}^{[i+1]})^T) \odot g'(\mathbf{z}^{[i]})$
    - Let $\odot$ denote the element-wise (Hadamard) product
    - Also note the use of $\mathbf{W}$ (rather than $\mathbf{\Theta}$)

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:

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 $(\nabla L)^{[i]}$

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

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 $\mathbf{\Theta}^{[i]} = \mathbf{W}^{[i]}, \mathbf{b}^{[i]}$ 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
- $\mathbf{\Theta}^{[i]}_{t+1} = \mathbf{\Theta}^{[i]}_{t} - \eta(\nabla L)^{[i]}$
- Because L is not necessarily convex anymore, we are not guaranteed to reach a global minimum

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
- $\mathbf{\Theta}_{t+1}^{[i]} = \mathbf{\Theta}_t^{[i]} - \eta(\nabla L)^{[i]}$
- 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.