Section 5: Forward-Backward Propagation

Ruofan Ma

Gov2018 2024 Spring

February 28, 2024

Motivation

- Why do we need neural networks?

Motivation

- Why do we need neural networks?
  - A problem that is intractable with raw input data may be solvable with basis-transformed data (e.g., LDA)

Motivation

- Why do we need neural networks?
  - A problem that is intractable with raw input data may be solvable with basis-transformed data (e.g., LDA)
  - This transformation, often, is guided by domain-specific knowledge.
  - Neural networks simultaneously solve for our model parameters and the best basis transformations.

Motivation

- Why do we need neural networks?
  - A problem that is intractable with raw input data may be solvable with basis-transformed data (e.g., LDA)
  - This transformation, often, is guided by domain-specific knowledge.
  - Neural networks simultaneously solve for our model parameters and the best basis transformations.
- What is the forward-backward propogation?
  - forward propagation: the algorithm that pushes our inputs through the hidden layers (weight matrices, activation functions) NN to get the final outputs

Motivation

- Why do we need neural networks?
  - A problem that is intractable with raw input data may be solvable with basis-transformed data (e.g., LDA)
  - This transformation, often, is guided by domain-specific knowledge.
  - Neural networks simultaneously solve for our model parameters and the best basis transformations.
- What is the forward-backward propogation?
  - forward propagation: the algorithm that pushes our inputs through the hidden layers (weight matrices, activation functions) NN to get the final outputs
  - backward-propagation: the algorithm that finds the best-suited weight matrices that would minimize the errors in the final outputs.

Roadmap for today

Roadmap for today

- A schematic review of what NN is doing

Roadmap for today

- A schematic review of what NN is doing

Roadmap for today

- A schematic review of what NN is doing
- Using forward-backward propagation to train NNs

Roadmap for today

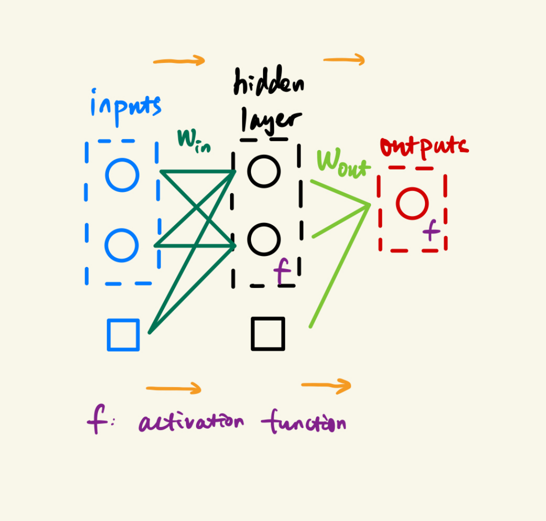
- A schematic review of what NN is doing
- Using forward-backward propagation to train NNs

Roadmap for today

- A schematic review of what NN is doing
- Using forward-backward propagation to train NNs
- Some coding exercise (`Rmd` file available on course website)

Feed-Forward Network

- The feed-forward neural network is a basic setup for a NN



More on Hidden Layers

Specifics

- Input layer is design matrix $\mathbf{X}$ which includes our usual vector of 1s

More on Hidden Layers

Specifics

- Input layer is design matrix $\mathbf{X}$ which includes our usual vector of 1s
- For binary classification, output layer is a vector of estimated probabilities $\hat{y}$

More on Hidden Layers

Specifics

- Input layer is design matrix $\mathbf{X}$ which includes our usual vector of 1s
- For binary classification, output layer is a vector of estimated probabilities $\hat{y}$
- Hidden layers are intermediate design matrices between inputs and outputs

More on Hidden Layers

Specifics

- Input layer is design matrix $\mathbf{X}$ which includes our usual vector of 1s
- For binary classification, output layer is a vector of estimated probabilities $\hat{y}$
- Hidden layers are intermediate design matrices between inputs and outputs (Deep learning is just a neural network with multiple hidden layers)

More on Hidden Layers

Specifics

- Input layer is design matrix $\mathbf{X}$ which includes our usual vector of 1s
- For binary classification, output layer is a vector of estimated probabilities $\hat{y}$
- Hidden layers are intermediate design matrices between inputs and outputs (Deep learning is just a neural network with multiple hidden layers)
- d features, then $d + 1$ input nodes

More on Hidden Layers

Specifics

- Input layer is design matrix $\mathbf{X}$ which includes our usual vector of 1s
- For binary classification, output layer is a vector of estimated probabilities $\hat{y}$
- Hidden layers are intermediate design matrices between inputs and outputs (Deep learning is just a neural network with multiple hidden layers)
- d features, then $d+1$ input nodes
- k possible classes, then $k-1$ output nodes

More on Hidden Layers

Specifics

- Input layer is design matrix $\mathbf{X}$ which includes our usual vector of 1s
- For binary classification, output layer is a vector of estimated probabilities $\hat{y}$
- Hidden layers are intermediate design matrices between inputs and outputs (Deep learning is just a neural network with multiple hidden layers)
- d features, then $d + 1$ input nodes
- k possible classes, then $k - 1$ output nodes
- h nodes in the hidden layer plus an intercept, where each of these nodes is a linear combination of the inputs, passed through an activation function (though note the intercept is always active and doesn't depend on nodes in the previous layer)

More on the Activation Function

More on the Activation Function

- Many possible choices (as seen in lecture). A common choice is the sigmoid function:

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

More on the Activation Function

- Many possible choices (as seen in lecture). A common choice is the sigmoid function:

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

- Notice that this is the inverse of the logit function:

$$\text{logit}(x) = \log\left(\frac{x}{1-x}\right)$$

, where x is probability of an event, and $\text{logit}(x)$ is its log odds.

Setting up a Feed-Forward Network

Steps

Setting up a Feed-Forward Network

Steps

1. Generate h different linear combinations of input features
2. Apply a nonlinear activation function, that for each observation, turns each hidden node 'on' or 'off'

Setting up a Feed-Forward Network

Steps

1. Generate h different linear combinations of input features
2. Apply a nonlinear activation function, that for each observation, turns each hidden node 'on' or 'off'
3. Fit logistic regression model to h transformed predictors (and intercept, sometimes called bias)

Setting up a Feed-Forward Network

Steps

1. Generate h different linear combinations of input features
2. Apply a nonlinear activation function, that for each observation, turns each hidden node 'on' or 'off'
3. Fit logistic regression model to h transformed predictors (and intercept, sometimes called bias)
4. Adjust parameters of both input and output to maximize likelihood

Setting up a Feed-Forward Network

Steps

1. Generate h different linear combinations of input features
2. Apply a nonlinear activation function, that for each observation, turns each hidden node 'on' or 'off'
3. Fit logistic regression model to h transformed predictors (and intercept, sometimes called bias)
4. Adjust parameters of both input and output to maximize likelihood
5. Repeat until the stopping criterion is met

Setting up a Feed-Forward Network

Steps

1. Generate h different linear combinations of input features
2. Apply a nonlinear activation function, that for each observation, turns each hidden node 'on' or 'off'
3. Fit logistic regression model to h transformed predictors (and intercept, sometimes called bias)
4. Adjust parameters of both input and output to maximize likelihood
5. Repeat until the stopping criterion is met

Note, when $h = 1$ there is only one linear combination of predictors. What does this become?

Setting up a Feed-Forward Network

Steps

1. Generate h different linear combinations of input features
2. Apply a nonlinear activation function, that for each observation, turns each hidden node 'on' or 'off'
3. Fit logistic regression model to h transformed predictors (and intercept, sometimes called bias)
4. Adjust parameters of both input and output to maximize likelihood
5. Repeat until the stopping criterion is met

Note, when $h = 1$ there is only one linear combination of predictors. What does this become? Without the nonlinearity in the hidden layer, the neural network reduces to a GLM.

Forward Propagation

Forward Propagation

1. Compute linear combination of features using weight matrix

$$\mathbf{z}_1 = \mathbf{X}\mathbf{W}_{\text{in}} = \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix} \mathbf{W}_{\text{in}}, \text{ where } \mathbf{W}_{\text{in}} \in \mathbb{R}^{(d+1)\times h}$$

Forward Propagation

1. Compute linear combination of features using weight matrix

$$\mathbf{z}_1 = \mathbf{X}\mathbf{W}_{\text{in}} = \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix} \mathbf{W}_{\text{in}}, \text{ where } \mathbf{W}_{\text{in}} \in \mathbb{R}^{(d+1)\times h}$$

2. Apply activation fn to get nodes in hidden layer

$$\mathbf{h} = \sigma(\mathbf{z}_1)$$

Note intercept/bias always activated, so fixed to be vector of ones

$$\mathbf{H} = \begin{bmatrix} 1 & \mathbf{h} \end{bmatrix} = \begin{bmatrix} 1 & \sigma(\mathbf{z}_1) \end{bmatrix} = \begin{bmatrix} 1 & \sigma(\mathbf{X}\mathbf{W}_{\text{in}}) \end{bmatrix}$$

Forward Propagation

1. Compute linear combination of features using weight matrix

$$\mathbf{z}_1 = \mathbf{X}\mathbf{W}_{\text{in}} = \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix} \mathbf{W}_{\text{in}}, \text{ where } \mathbf{W}_{\text{in}} \in \mathbb{R}^{(d+1)\times h}$$

2. Apply activation fn to get nodes in hidden layer

$$\mathbf{h} = \sigma(\mathbf{z}_1)$$

Note intercept/bias always activated, so fixed to be vector of ones

$$\mathbf{H} = \begin{bmatrix} 1 & \mathbf{h} \end{bmatrix} = \begin{bmatrix} 1 & \sigma(\mathbf{z}_1) \end{bmatrix} = \begin{bmatrix} 1 & \sigma(\mathbf{X}\mathbf{W}_{\text{in}}) \end{bmatrix}$$

3. For output layer, compute linear combination of hidden variables, this time using another weight matrix $\mathbf{W}_{\text{out}} \in \mathbb{R}^{(h+1)\times(k-1)}$

$$\mathbf{z}_2 = \mathbf{H}\mathbf{W}_{\text{out}} = \begin{bmatrix} 1 & \mathbf{h} \end{bmatrix} \mathbf{W}_{\text{out}}$$

Forward Propagation

1. Compute linear combination of features using weight matrix

$$\mathbf{z}_1 = \mathbf{X}\mathbf{W}_{\text{in}} = \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix} \mathbf{W}_{\text{in}}, \text{ where } \mathbf{W}_{\text{in}} \in \mathbb{R}^{(d+1)\times h}$$

2. Apply activation fn to get nodes in hidden layer

$$\mathbf{h} = \sigma(\mathbf{z}_1)$$

Note intercept/bias always activated, so fixed to be vector of ones

$$\mathbf{H} = \begin{bmatrix} 1 & \mathbf{h} \end{bmatrix} = \begin{bmatrix} 1 & \sigma(\mathbf{z}_1) \end{bmatrix} = \begin{bmatrix} 1 & \sigma(\mathbf{X}\mathbf{W}_{\text{in}}) \end{bmatrix}$$

3. For output layer, compute linear combination of hidden variables, this time using another weight matrix $\mathbf{W}_{\text{out}} \in \mathbb{R}^{(h+1)\times(k-1)}$

$$\mathbf{z}_2 = \mathbf{H}\mathbf{W}_{\text{out}} = \begin{bmatrix} 1 & \mathbf{h} \end{bmatrix} \mathbf{W}_{\text{out}}$$

4. Apply one more function to get the output

$$\hat{\mathbf{y}} = \sigma(\mathbf{z}_2)$$

Forward Propagation

1. Compute linear combination of features using weight matrix

$$\mathbf{z}_1 = \mathbf{X}\mathbf{W}_{\text{in}} = \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix} \mathbf{W}_{\text{in}}, \text{ where } \mathbf{W}_{\text{in}} \in \mathbb{R}^{(d+1)\times h}$$

2. Apply activation fn to get nodes in hidden layer

$$\mathbf{h} = \sigma(\mathbf{z}_1)$$

Note intercept/bias always activated, so fixed to be vector of ones

$$\mathbf{H} = \begin{bmatrix} 1 & \mathbf{h} \end{bmatrix} = \begin{bmatrix} 1 & \sigma(\mathbf{z}_1) \end{bmatrix} = \begin{bmatrix} 1 & \sigma(\mathbf{X}\mathbf{W}_{\text{in}}) \end{bmatrix}$$

3. For output layer, compute linear combination of hidden variables, this time using another weight matrix $\mathbf{W}_{\text{out}} \in \mathbb{R}^{(h+1)\times(k-1)}$

$$\mathbf{z}_2 = \mathbf{H}\mathbf{W}_{\text{out}} = \begin{bmatrix} 1 & \mathbf{h} \end{bmatrix} \mathbf{W}_{\text{out}}$$

4. Apply one more function to get the output

$$\hat{\mathbf{y}} = \sigma(\mathbf{z}_2)$$

5. Put it all together: $\hat{\mathbf{y}} = \sigma(\mathbf{H}\mathbf{W}_{\text{out}}) = \sigma\left(\begin{bmatrix} 1 & \sigma(\mathbf{X}\mathbf{W}_{\text{in}}) \end{bmatrix} \mathbf{W}_{\text{out}}\right)$

Backward Propagation

How do we get the weight matrices?

Backward Propagation

How do we get the weight matrices?

- Unlike $\hat{\beta}$ linear regression, there is no closed form solution to $\mathbf{W}$

Backward Propagation

How do we get the weight matrices?

- Unlike $\hat{\beta}$ linear regression, there is no closed form solution to $\mathbf{W}$
- Gradient Descent!

Backward Propagation

How do we get the weight matrices?

- Unlike $\hat{\beta}$ linear regression, there is no closed form solution to $\mathbf{W}$
- Gradient Descent!
- For binary-classifier: train the NN by minimizing cross-entropy loss (negated log-likelihood).

Backward Propagation

How do we get the weight matrices?

- Unlike $\hat{\beta}$ linear regression, there is no closed form solution to **W**
- Gradient Descent!
- For binary-classifier: train the NN by minimizing cross-entropy loss (negated log-likelihood).
- Objective Function (with sigmoid activation function):

$$\mathcal{L} = \sum_i \left(y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i) \right)$$

Backward Propagation

How do we get the weight matrices?

- Unlike $\hat{\beta}$ linear regression, there is no closed form solution to $\mathbf{W}$
- Gradient Descent!
- For binary-classifier: train the NN by minimizing cross-entropy loss (negated log-likelihood).
- Objective Function (with sigmoid activation function):

$$\mathcal{L} = \sum_i \left(y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i) \right)$$

- Optimize $-\mathcal{L} = f(\mathbf{W})$ via gradient descent by iterating the formula: where $\mathbf{W}_{t+1} = \mathbf{W}_t - \gamma \cdot \nabla f(\mathbf{W}_t)$ $\mathbf{W}$ and γ is 'learning rate'.

Backward Propagation

How do we get the gradient of the objective function?

Backward Propagation

How do we get the gradient of the objective function?

- With respect to the output weights is:

$$\frac{-\partial \mathcal{L}}{\partial \mathbf{W}_{\text{out}}} = \frac{-\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_{\text{out}}}$$

Backward Propagation

How do we get the gradient of the objective function?

- With respect to the output weights is:

$$\frac{-\partial \mathcal{L}}{\partial \mathbf{W}_{\text{out}}} = \frac{-\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_{\text{out}}} = \left[\frac{\hat{\mathbf{y}} - \mathbf{y}}{\hat{\mathbf{y}}(1 - \hat{\mathbf{y}})}\right] \left[\mathbf{H}^\top \hat{\mathbf{y}}(1 - \hat{\mathbf{y}})\right]$$

- With respect to the input weights is:

$$\frac{-\partial \mathcal{L}}{\partial \mathbf{W}_{in}} = \frac{-\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{H}} \frac{\partial \mathbf{H}}{\partial \mathbf{W}_{in}}$$

Backward Propagation

How do we get the gradient of the objective function?

- With respect to the output weights is:

$$\frac{-\partial \mathcal{L}}{\partial \mathbf{W}_{\text{out}}} = \frac{-\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{W}_{\text{out}}} = \left[\frac{\hat{\mathbf{y}} - \mathbf{y}}{\hat{\mathbf{y}}(1 - \hat{\mathbf{y}})} \right] \left[\mathbf{H}^\top \hat{\mathbf{y}}(1 - \hat{\mathbf{y}}) \right]$$

- With respect to the input weights is:

$$\frac{-\partial \mathcal{L}}{\partial \mathbf{W}_{in}} = \frac{-\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{H}} \frac{\partial \mathbf{H}}{\partial \mathbf{W}_{in}}$$

, where

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{H}} = \sigma\left(\mathbf{H}\mathbf{W}_{\text{out}}\left(1 - \sigma\left(\mathbf{H}\mathbf{W}_{\text{out}}\right)\right)\right) \mathbf{W}_{\text{out}}^\top = \hat{\mathbf{y}}(1 - \hat{\mathbf{y}})\mathbf{W}_{\text{out}}^\top$$

$$\frac{\partial \mathbf{H}}{\partial \mathbf{W}_{\text{in}}} = \mathbf{X}^\top \left[\ 1 \quad \sigma\left(\mathbf{X}\mathbf{W}_{\text{in}}\right)\left(1 - \sigma\left(\mathbf{X}\mathbf{W}_{\text{in}}\right)\right)\ \right]$$

Putting Everything Together

Training a NN using forward-backward propagation

1. Initialize weights
2. Propagate forward to get output estimates
3. Propagate error backward to update the weights toward a better solution
4. Iterate forward and back propagation until stopping criterion

- Coding exercise available on the course website (`Rmd`)