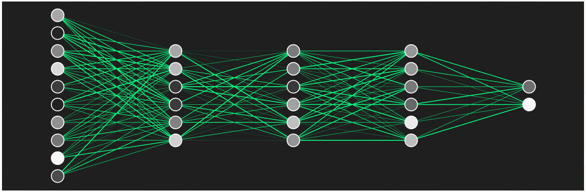
Logistic Regression and Neural Networks Pytorch

Dana Golden, Lilia Maliar



Data Science and Machine Learning - **November 30, 2024**

Presentation Outline

Continuous vs. Discrete Data

- What makes these two forms of data different?
- Why is this an important difference?
- What assumptions of models get violated with discrete data
- What models work with discrete data?

The Impact of Neural Networks

- Neural networks revolutionized the field of machine learning
- At their heart, neural networks simply are an easier way to fit a function with massive amounts of data
- Turns out to be a useful tasks for games, computer vision, NLP, chatbots, etc.

Neural Networks in Economics

- Neural networks have been slow to be adapted into economics. Why?

Neural Networks in Economics

- Neural networks have been slow to be adapted into economics. Why?
- The causality problem in neural networks is not unsolvable. Work is being done to create explainable neural networks.

Neural Networks in Economics

- Neural networks have been slow to be adapted into economics. Why?
- The causality problem in neural networks is not unsolvable. Work is being done to create explainable neural networks.
- Currently neural networks are most used for labelling data and handling unstructured data

Neural Networks in Economics

- Neural networks have been slow to be adapted into economics. Why?
- The causality problem in neural networks is not unsolvable. Work is being done to create explainable neural networks.
- Currently neural networks are most used for labelling data and handling unstructured data
- Neural networks are also incredibly common in dynamic fields such as macro and IO

Neural Networks in Economics

- Neural networks have been slow to be adapted into economics. Why?
- The causality problem in neural networks is not unsolvable. Work is being done to create explainable neural networks.
- Currently neural networks are most used for labelling data and handling unstructured data
- Neural networks are also incredibly common in dynamic fields such as macro and IO
- More applications are coming!

Logistic Vs. Linear Regression

- Why can't you use a linear regression for a discrete variable?

Logistic Vs. Linear Regression

- Why can't you use a linear regression for a discrete variable?
- Logistic regression y values are naturally bounded above by 1

Logistic Vs. Linear Regression

- Why can't you use a linear regression for a discrete variable?
- Logistic regression y values are naturally bounded above by 1
- With logistic regression, effect sizes change as output increases

Logistic Vs. Linear Regression

- Why can't you use a linear regression for a discrete variable?
- Logistic regression y values are naturally bounded above by 1
- With logistic regression, effect sizes change as output increases
- What else differentiates them?

Sigmoid Function

$$\sigma(\vec{z_i}) = \frac{1}{1 + e^{-z_j}} \tag{1}$$

$$z_j = X_i \beta \tag{2}$$

- When will this equal one half?

Softmax Function

$$\sigma(\vec{z}_i) = \frac{e^{z_i}}{\sum_{j=1}^{J} e^{z_j}} \tag{3}$$

$$z_j = X_i\beta \tag{4}$$

- Why this function? What interesting properties does it have that make it useful?

Logistic Regression steps

- Randomly initialize weights
- Take dot product and find predictions
- Take log likelihood
- Determine gradient of loglikelihood function
- Take step for weights in direction of gradient
- Repeat until convergence or hit max steps

Logistic Regression log-likelihood Equation

- Likelihood function is:

$$L(\beta|X,Y) = \prod_{i=1}^{n} P(Y_i = 1|x_i)^{y_i}(1 - P(Y_i = 1|x_i))^{1-y_i} \tag{5}$$

- Loglikelihood is:

$$l(\beta) = \sum_{i=1}^{N} y_i log(p_i) + (1 - y_i) log(1 - p_i) \tag{6}$$

- Normalize log-loss is:

$$J(\beta) = \frac{-1}{N} \sum_{i=1}^{N} y_i log(p_i) + (1 - y_i) log(1 - p_i) \tag{7}$$

Finding Gradient of Log-likelihood

- How should we go about this?

Finding Gradient of Log-likelihood

- How should we go about this?

$$\frac{\partial J}{\partial \beta_j} = -(y_i - \hat{y})x_j \tag{8}$$

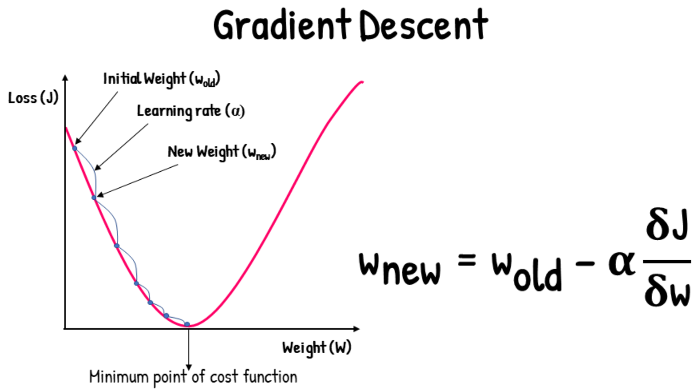
Probit vs. Logit

- Probit is a more generalized for of logit
- Probit assumes a normal standard error while logit assumes a logistic standard error
- Logit is preferable for data science because it has a closed form solution

Overview of Neural Networks

- At their basic level neural networks consist of a sequence of linear regressions followed by non-linear activation functions
- Multiple layers, special functions, and non-linearities allow logistic regressions to learn

Review of Gradient Descent



Forward propagation

- Forward propogation moves data from the input to the output in the neural network
- The most basic form of forward propogation is a linear regression
- Other types of layers can be added. e.g. Convolutional layer, recurrent layer

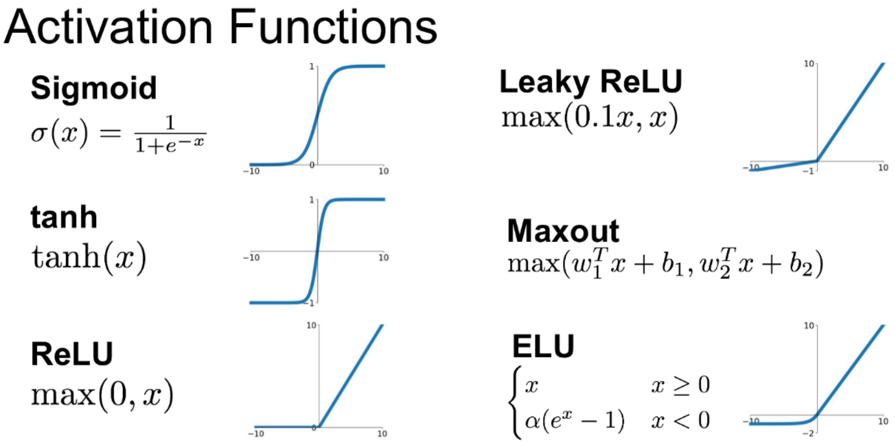
Value of multiple layers

- Each node can learn one particular feature of the dataset
- Different layers can learn different types of information
- Successive layers in the neural network learn combinations of different features in earlier layers to recognize more interesting patterns in data

Activation Functions

- Activation functions create non-linearities between the layers
- Activation functions are what allows neural networks to learn non-linear functions
- Without them, neural networks are effectively linear regressions

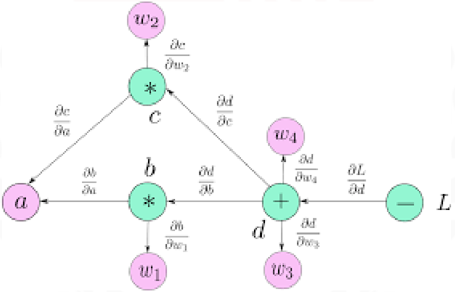
Activation Function Visual



Automatic Differentiation

- Automatic differentiation replaces manual differentiation with a network graph that automatically finds the derivative of a set of operations
- Everything is chain rule, when in doubt, chain rule

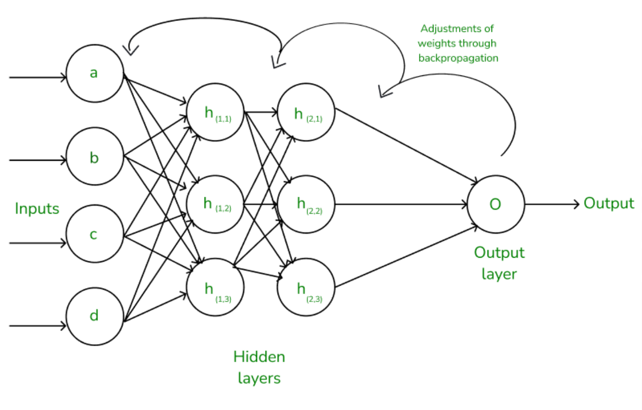
Automatic Differentiation Graph



Backpropogation

- Backpropogation finds the derivative of the cost function with respect to each of the weights
- It allows weights and biases to be adjusted based on their impact on the cost

Backpropogation Visual



Logistic Regression as a one Layer Neural Network

- Logistic regression is actually a type of neural network!
- It consists of one linear layer followed by a sigmoid activation function
- Many early neural networks utilized simple multi-stage linear regressions e.g. MLPs

Output of neural network

- The output of the neural network is based on the last linear layer, the final activation function, and the cost function

Output of neural network

- The output of the neural network is based on the last linear layer, the final activation function, and the cost function
- The number of output features of the last linear layer is the number of features of the input to the final activation functions

Output of neural network

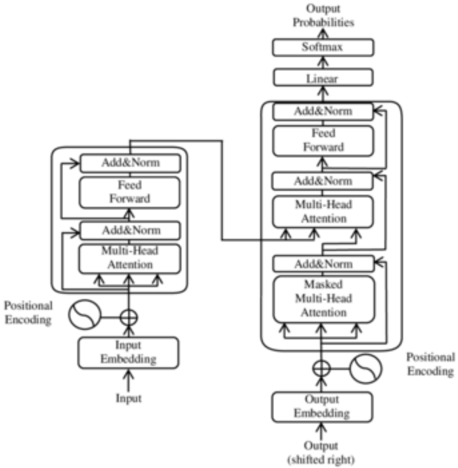
- The output of the neural network is based on the last linear layer, the final activation function, and the cost function
- The number of output features of the last linear layer is the number of features of the input to the final activation functions
- Some activation functions allow continuous variables, others like softmax are discrete

Output of neural network

- The output of the neural network is based on the last linear layer, the final activation function, and the cost function
- The number of output features of the last linear layer is the number of features of the input to the final activation functions
- Some activation functions allow continuous variables, others like softmax are discrete
- Which cost function should you use for logistic regression? Continuous variables?

Overivew of Transformer

- Transformers represent a major step forward in neural networks



Finding gradient of softmax function

$$s_i = \frac{e^{z_i}}{\sum_k e^{z_k}} \tag{9}$$

Finding gradient of softmax function

$$s_i = \frac{e^{z_i}}{\sum_k e^{z_k}} \tag{9}$$

$$s_i = \frac{e^{z_i}}{e^{z_i} + \sum_{k \neq i} e^{z_k}} \tag{10}$$

Finding gradient of softmax function

$$s_i = \frac{e^{z_i}}{\sum_k e^{z_k}} \tag{9}$$

$$s_i = \frac{e^{z_i}}{e^{z_i} + \sum_{k \neq i} e^{z_k}} \tag{10}$$

$$\frac{s_i}{\partial z_i} = \frac{(e^{z_i} + \sum_{k \neq i} e^{z_k}) e^{z_i} - e^{z_i} e^{z_i}}{(e^{z_i} + \sum_{k \neq i} e^{z_k})^2} = \tag{11}$$

Finding gradient of softmax function

$$s_i = \frac{e^{z_i}}{\sum_k e^{z_k}} \tag{9}$$

$$s_i = \frac{e^{z_i}}{e^{z_i} + \sum_{k \neq i} e^{z_k}} \tag{10}$$

$$\frac{s_i}{\partial z_i} = \frac{(e^{z_i} + \sum_{k \neq i} e^{z_k})e^{z_i} - e^{z_i} e^{z_i}}{(e^{z_i} + \sum_{k \neq i} e^{z_k})^2} = \tag{11}$$

$$\frac{e^{z_i}}{e^{z_i} + \sum_{k \neq i} e^{z_k}} * \frac{e^{z_i} + \sum_{k \neq i} e^{z_k} - e^{z_i}}{e^{z_i} + \sum_{k \neq i} e^{z_k}} = \tag{12}$$

$$s_i(1 - s_i) \tag{13}$$

Creating Or with Single Linear Threshold Neuron

$$f(x) = \begin{cases} 1 & w^T x + b \geq 0 \\ 0 & w^T x + b < 0 \end{cases}$$

Creating Or with Single Linear Threshold Neuron

$$f(x) = \begin{cases} 1 & w^T x + b \geq 0 \\ 0 & w^T x + b < 0 \end{cases}$$

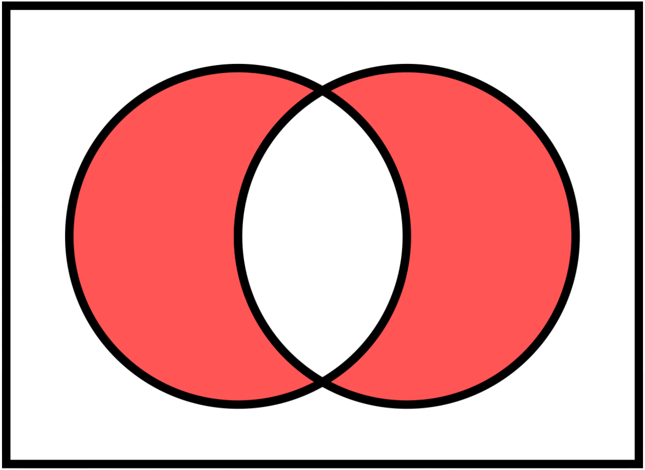
Creating And with Single Layer Threshold Neuron

$$f(x) = \begin{cases} 1 & w^T x + b \geq 0 \\ 0 & w^T x + b < 0 \end{cases}$$

Creating And with Single Layer Threshold Neuron

$$f(x) = \begin{cases} 1 & w^T x + b \geq 0 \\ 0 & w^T x + b < 0 \end{cases}$$

Proving XOR is impossible with just a single layer threshold neuron



Backpropogation Example: Single Neuron

- Network structure:
  - One input x
  - Simple linear layer: $z = w \cdot x + b$
  - No activation function
  - Mean-squared Error: $L = \frac{1}{n}(\hat{y} - y)^2$
- Backpropogation

Backpropogation Example: Single Neuron

- Network structure:
  - One input x
  - Simple linear layer: $z = w \cdot x + b$
  - No activation function
  - Mean-squared Error: $L = \frac{1}{n}(\hat{y} - y)^2$
- Backpropogation
  - Compute gradient of loss with respect to $\hat{y}$: $\frac{\partial L}{\partial \hat{y}} = \frac{2(\hat{y}-y)}{n}$

Backpropogation Example: Single Neuron

- Network structure:
  - One input x
  - Simple linear layer: $z = w \cdot x + b$
  - No activation function
  - Mean-squared Error: $L = \frac{1}{n}(\hat{y} - y)^2$
- Backpropogation
  - Compute gradient of loss with respect to $\hat{y}$: $\frac{\partial L}{\partial \hat{y}} = \frac{2(\hat{y}-y)}{n}$
  - Gradient with respect to z: $\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \frac{2(\hat{y}-y)}{n}(1)$

Backpropogation Example: Single Neuron

- Network structure:
  - One input x
  - Simple linear layer: $z = w \cdot x + b$
  - No activation function
  - Mean-squared Error: $L = \frac{1}{n}(\hat{y} - y)^2$
- Backpropogation
  - Compute gradient of loss with respect to $\hat{y}$: $\frac{\partial L}{\partial \hat{y}} = \frac{2(\hat{y}-y)}{n}$
  - Gradient with respect to z: $\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \frac{2(\hat{y}-y)}{n}(1)$
  - Compute gradient with respect to weights: $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w} = \frac{2(\hat{y}-y)}{n}(1)(x)$

Backpropogation Example: Single Neuron

- Network structure:
  - One input x
  - Simple linear layer: $z = w \cdot x + b$
  - No activation function
  - Mean-squared Error: $L = \frac{1}{n}(\hat{y} - y)^2$
- Backpropogation
  - Compute gradient of loss with respect to $\hat{y}$: $\frac{\partial L}{\partial \hat{y}} = \frac{2(\hat{y}-y)}{n}$
  - Gradient with respect to z: $\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} = \frac{2(\hat{y}-y)}{n}(1)$
  - Compute gradient with respect to weights: $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w} = \frac{2(\hat{y}-y)}{n}(1)(x)$
  - Compute gradient with respect to bias: $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b} = \frac{2(\hat{y}-y)}{n}(1)(1)$

Backpropogation Example: Two-layer Neural Network

- Network structure:
  - Two inputs x_1, x_2
  - Hidden layer with two neurons: $z_1 = w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1$, $z_2 = w_{12} \cdot x_1 + w_{22} \cdot x_2 + b_2$
  - Output layer with one neuron: $z_3 = w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3$
  - Sigmoid activation function: $\frac{1}{1+e^{-z_i}}$
  - Binary cross-entropy loss: $L = -(ylog(\hat{y}) + (1-y)(log(1-\hat{y})))$
- Backpropogation

Backpropogation Example: Two-layer Neural Network

- Network structure:
  - Two inputs x_1, x_2
  - Hidden layer with two neurons: $z_1 = w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1$, $z_2 = w_{12} \cdot x_1 + w_{22} \cdot x_2 + b_2$
  - Output layer with one neuron: $z_3 = w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3$
  - Sigmoid activation function: $\frac{1}{1+e^{-z_i}}$
  - Binary cross-entropy loss: $L = -(ylog(\hat{y}) + (1-y)(log(1-\hat{y})))$
- Backpropogation
  - Compute gradient of loss with respect to $\hat{y}$: $\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$

Backpropogation Example: Two-layer Neural Network

- Network structure:
  - Two inputs x_1, x_2
  - Hidden layer with two neurons: $z_1 = w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1$, $z_2 = w_{12} \cdot x_1 + w_{22} \cdot x_2 + b_2$
  - Output layer with one neuron: $z_3 = w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3$
  - Sigmoid activation function: $\frac{1}{1+e^{-z_i}}$
  - Binary cross-entropy loss: $L = -(ylog(\hat{y}) + (1-y)(log(1-\hat{y})))$
- Backpropogation
  - Compute gradient of loss with respect to $\hat{y}$: $\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$
  - Gradient with respect to z_3: $\frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} = \hat{y}(1-\hat{y})(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}) = -(1-\hat{y})y + (1-y)\hat{y}$

Backpropogation Example: Two-layer Neural Network

- Network structure:
  - Two inputs x_1, x_2
  - Hidden layer with two neurons: $z_1 = w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1$, $z_2 = w_{12} \cdot x_1 + w_{22} \cdot x_2 + b_2$
  - Output layer with one neuron: $z_3 = w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3$
  - Sigmoid activation function: $\frac{1}{1+e^{-z_i}}$
  - Binary cross-entropy loss: $L = -(ylog(\hat{y}) + (1-y)(log(1-\hat{y})))$
- Backpropogation
  - Compute gradient of loss with respect to $\hat{y}$: $\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$
  - Gradient with respect to z_3: $\frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} = \hat{y}(1-\hat{y})(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}) = -(1-\hat{y})y + (1-y)\hat{y}$
  - Compute gradient of z_3 with respect to weights w_{i3}: $\frac{\partial L}{\partial w_{i3}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial w_{i3}} = (-(1-\hat{y})y + (1-y)\hat{y})a_i$

Backpropogation Example: Two-layer Neural Network

- Compute gradient of z_3 with respect to bias:
  $\frac{\partial L}{\partial b_{i3}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial b_3} = -(1 - \hat{y})y + (1 - y)\hat{y}$

Backpropogation Example: Two-layer Neural Network

- Compute gradient of z_3 with respect to bias: $\frac{\partial L}{\partial b_{i3}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial b_3} = -(1-\hat{y})y + (1-y)\hat{y}$
- Compute gradient of z_3 with respect to activation a_1: $\frac{\partial L}{\partial a_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} = (-(1-\hat{y})y + (1-y)\hat{y})w_{i3}$

Backpropogation Example: Two-layer Neural Network

- Compute gradient of z_3 with respect to bias:
  $\frac{\partial L}{\partial b_{i3}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial b_3} = -(1-\hat{y})y + (1-y)\hat{y}$
- Compute gradient of z_3 with respect to activation a_1:
  $\frac{\partial L}{\partial a_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} = (-(1-\hat{y})y + (1-y)\hat{y})w_{i3}$
- Compute gradient of activation a_i with respect to z_1:
  $\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} \frac{\partial a_1}{\partial z_1} = ((-(1-\hat{y})y + (1-y)\hat{y})w_{i3})(a_1(1-a_1))$

Backpropogation Example: Two-layer Neural Network

- Compute gradient of z_3 with respect to bias:
  $\frac{\partial L}{\partial b_{i3}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial b_3} = -(1-\hat{y})y + (1-y)\hat{y}$
- Compute gradient of z_3 with respect to activation a_1:
  $\frac{\partial L}{\partial a_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} = (-(1-\hat{y})y + (1-y)\hat{y})w_{i3}$
- Compute gradient of activation a_i with respect to z_1:
  $\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} \frac{\partial a_1}{\partial z_1} = ((-(1-\hat{y})y + (1-y)\hat{y})w_{i3})(a_1(1-a_1))$
- Compute gradient of activation z_i with respect to weight w_{11}:
  $\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{11}} = ((-(1-\hat{y})y+(1-y)\hat{y})w_{i3})(a_1(1-a_1))(x_1)$

Backpropogation Example: Two-layer Neural Network

- Compute gradient of z_3 with respect to bias:
  $\frac{\partial L}{\partial b_{i3}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial b_3} = -(1-\hat{y})y + (1-y)\hat{y}$
- Compute gradient of z_3 with respect to activation a_1:
  $\frac{\partial L}{\partial a_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} = (-(1-\hat{y})y + (1-y)\hat{y})w_{i3}$
- Compute gradient of activation a_i with respect to z_1:
  $\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} \frac{\partial a_1}{\partial z_1} = ((-(1-\hat{y})y + (1-y)\hat{y})w_{i3})(a_1(1-a_1))$
- Compute gradient of activation z_i with respect to weight w_{11}:
  $\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_{11}} = ((-(1-\hat{y})y+(1-y)\hat{y})w_{i3})(a_1(1-a_1))(x_1)$
- Compute gradient of activation z_i with respect to bias b_1 : $\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_3} \frac{\partial z_3}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial b_1} = ((-(1-\hat{y})y+(1-y)\hat{y})w_{i3})(a_1(1-a_1))$

What Hugging Face is

- Hugging Face is an online repository of trained models that can be used out of the box or retrained
- Hugging Face substantially reduces the time to begin working with complex pre-trained models

Quick Tour

- HuggingFaceTour

Thank You So Much!

List of References