Machine Learning

Cesar Lema (cl4393@nyu.edu)

Spring 2020

Contents

1 Function approximation

* Lecture slides

Given data points as a set of N input-output samples i.e. $\{(\mathbf{x}^{(i)}, y^{(i)}) : i = 1, \ldots, N\}$ where $\mathbf{x} \in \mathbb{R}^D$ called a feature vector find a function $f(\mathbf{x}) : \mathbf{x} \to \hat{y}$ i.e. to predict y from $\mathbf{x}$.

This can be accomplished through regression but requires

1. Way to evaluate how good a fit a function is
2. Class of function in which to get f(x) from

Empirical Risk Minimization

A method of minimization of the average loss or cost of parameters associated with a fit $f(x)$ on training data. The loss $L(y, \hat{y}) : (y, \hat{y}) \to d \in \mathbb{R}$ is a function that maps the expected and predicted output values to a real number that measures the difference in a predicted output value $\hat{y}$ from the expected output value y i.e. how well a fit is on a data point. The empirical risk $R(\theta) : (\mathbf{x}, y, \hat{y}) \to d' \in \mathbb{R}$ given the weights θ of a fit is a function defined as

$$R(\theta) = \frac{1}{N} \sum_{i=0}^{N} L\left(y^{(i)}, f(\mathbf{x}^{(i)}, \theta)\right)$$

The empirical risk measures how well a fit and its associated parameters are on the entire training data.

Loss function

Some examples of loss functions include square error and absolute error, logistic loss function.

Some class of function for regression are linear functions, multi-dimensional functions, etc. The class of functions can introduce hyperparameters (parameters that ultimately define the parameters of the model/fit) such as the dimension of multi-dimensional fits, order of polynomial fits, etc.

Minimization

The empirical risk or cost function can then be minimized by finding the critical points of the empirical risk and the associated values of θ. *Cross validation* can be used to find the optimal values for the hyperparameters by plotting the empirical risk at the optimal θ values over the range of hyperparameters for training and testing data. The optimal hyperparamter corresponds to the minimum empirical risk on the testing data over possible values of the hyperparameter.

2 Logistic Neuron/ Perceptron

* Tony Jebara Lecture notes

Some definitions

- *Loss function:* $L(y, \hat{y})$. Outputs a scalar indicating how bad one predicted value is from the correct value. Applied to a single training example.
- *Cost function:* $J(\theta)$ or $R(\theta)$. Average loss over all training examples, essentially the cost of function parameters. How well parameters are doing on entire training examples.

Building on regression or function approximation to classification now the output is a binary value.

A neuron takes inputs then transforms them and outputs a value. A function $f(\mathbf{x})$ takes a linear combination of neuron inputs and outputs a value denoted by z. This allows each neuron to gauge how much of an input it wants to use/activate through the associated weight. An activation function $g(z)$ then transforms z and outputs a value denoted by a. The output a is the resulting output of the neuron $\hat{y}$. note output range can vary depending on activation function used.

To go from function approximation to neuron classification want to

1. Fix $f(\mathbf{x})$ to give binary output (logistic neuron)
2. Fix our definition of risk we will minimize so that we get good classification accuracy (logistic loss)
3. Make an ever better fix on $f(\mathbf{x})$ to binarize (perceptron)
4. Make an ever better risk (perceptron loss)

Linear neuron

A simple neuron that takes a linear combination of features from an input/feature vector $\mathbf{x} \in \mathbb{R}^D$ and output a real number. This is a mapping $f : \mathbb{R}^D \to \mathbb{R}$ where $\mathbf{x} \to d \in \mathbb{R}$ given by

$$f(\mathbf{x}) = \theta^T \mathbf{x}$$

where θ is a parameter vector and each element is a parameter associated with the input feature. The activation function $g(z)$ where $f(\mathbf{x}) = z$ can be considered to the identity function.

The loss is not specified to have a certain form.

Logistic neuron

Neuron with binary output using logistic activation function. The inputs are first mapped to a real value by taking a linear combination with a parameter/weights vector θ. The logistic activation function is $g : \mathbb{R} \to \{0, 1\}$ where $z \to \{0, 1\}$ given by

$$g(z) = \frac{1}{1 + e^{-z}}$$

and the neuron outputs 1 if $g(x) > 0.5$ and 0 otherwise. note this is simply logistic regression if output is any number between $\{0, 1\}$.

The loss is defined by the logistic loss function as

$$L(y, \hat{y}) = (y-1)\log(1-\hat{y}) - y\log(\hat{y})$$

where $\hat{y}$ is given by the output of the neuron.

The empirical risk or cost function does not have a closed form/analytical solution. Another way to approximate the minimum is through Gradient descent. Note we want a convex function so that we know gradient descent converges to a global minimum. The cost function using logistic loss is convex therefore will always converge to the same global minimum.

Gradient descent is heuristically implemented by repeatably running on all the parameters

$$\theta := \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$$

where α is called the learning rate.

Perceptron

Neuron with binary output taking on values $\{-1, 1\}$ using the classification activation function. The inputs are still mapped to a real value by taking a linear combination with a parameter/weights vector θ. The classification activation function is $g : \mathbb{R} \to \{-1, 1\}$ where $z \to \{-1, 1\}$ given by

$$g(z) = \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases}$$

the loss is defined by the perceptron loss function as

$$L(y, f(\mathbf{x})) = \begin{cases} -y(\theta^T\mathbf{x}) & -y(\theta^T\mathbf{x}) \leq 0 \\ 0 & -y(\theta^T\mathbf{x}) > 0 \end{cases}$$

i.e. loss is 0 if classified correctly otherwise takes on linear values.

The empirical risk or cost function is given by

$$R(\theta) = -\frac{1}{N} \sum_{i \in \text{misclassified}} y^{(i)}(\theta^T\mathbf{x}^{(i)})$$

3 Nueral Network/Multilayer Perceptron (NN/MLP)

*Coursera notes

Some definitions

- *Computation graph:* Diagram representation of computation of a function given its inputs/parameters from left to right.
- *Vectorization:* Getting rid of explicit for loops in code using linear algebra
- *Broadcasting:* "Stretching" of a dimension of a multidimensional array to perform a binary operation using numpy. note, value is not actually stretched or copied and is efficiently implemented in a vectorized operation.

Neural networks are multiple layers of parallel neurons or networks in series. Each output layer is the input to the next layer with each layer having its own weights associated with it. Activation function are used and introduce non linearity in order to learn complex mapping. Note each node/neuron splits its input space with a hyperplane. The output value for NN or MLP is computed using forward propagation and the parameters are updated using back-propagation.

Forward-propagation:

Calculation of an input into a NN by passing an input layer by layer with appropriate calculations.

Optimal NN and MLP perform calculations using a vectorized approach.

Back-propagation:

Calculations of gradients with respect to parameters at each layer of a NN for training through gradient descent. The chain rule is used to calculate the derivative of a function (in this case the cost/empirical risk) with respect to a specified variable (in this case the weights of each neuron) in terms of the auxiliary changes that the specified variable has on other inputs/variables that make up the function. This process is essentially gradient descent done layer by layer.

MLP

Notation

> For a single training sample/data point denoted with a superscript m in parenthesis the i^{th} layer of a NN calculates a linear combination of the inputs denoted by $z^{[i](m)}$, activation denoted $a^{[i](m)}$, has a weights array $W^{[i]}$ and a weight vector $b^{[i]}$.
>
> For all samples in a vectorized implementation the computations of the i^{th} layer of a NN calculate are denoted with a boldface letter. The linear combination of inputs is $Z^{[i]}$ and the activation is $Z^{[i]}$.

For a multi-layer perception, let each hidden layer use the ReLU activation function and the output layer uses the sigmoid activation function.

Forward propagation

Forward propagation for m samples in a vectorized implementation is given by iterating over each layer and calculating

1. Input layer 0:

   $$A^{[0]} = X$$

   where X is the input training data with the shape of (n_x, m)

2. Input layer 1:

   $$\begin{aligned} Z^{[L]} &= W^{[L]}A^{[L-1]} + b^{[L]} \\ A^{[L]} &= \text{ReLU}(Z^{[1]}) \end{aligned}$$

   where the ReLU activation function is applied element wise.

3. [this is repeated up until the last layer]

L. Input layer L:

   $$\begin{aligned} Z^{[1]} &= W^{[1]}A^{[0]} + b^{[1]} \\ A^{[1]} &= \text{Sigmoid}(Z^{[1]}) \end{aligned}$$

   where the Sigmoid activation function is applied element wise.

The *loss* is given by

$$\begin{aligned} L(y^{(i)}, \hat{y}^{(i)}) &= (y^{(i)} - 1)\text{log}(1 - \hat{y}^{(i)}) - y^{(i)}\text{log}(\hat{y}^{(i)}) \\ &= (y^{(i)} - 1)\text{log}(1 - a^{[L](i)}) - y^{(i)}\text{log}(a^{[L](i)}) \\ &= (y^{(i)} - 1)\text{log}(1 - g^{[L]}(z^{[L](i)})) - y^{(i)}\text{log}(g^{[L]}(z^{[L](i)})) \\ &= (y^{(i)} - 1)\text{log}(1 - g^{[L]}(W^{[L]}a^{(L-1)(i)} + b^{[L]})) - y^{(i)}\text{log}(g^{[L]}(W^{[L]}a^{[L-1](i)} + b^{[L]})) \end{aligned}$$

where the activation at output layer L is explicitly written in terms of its inputs $a^{[L-1](i)}$.

The *empirical risk* or cost is

$$\begin{aligned} R(\theta) &= \frac{1}{N}\sum_{i}^{N} L(y^{(i)}, \hat{y}^{(i)}) \\ &= \frac{1}{N}\sum_{i}^{N}(y^{(i)} - 1)\text{log}(1 - \hat{y}^{(i)}) - y^{(i)}\text{log}(\hat{y}^{(i)}) \\ &= \frac{1}{N}\sum_{i}^{N}(y^{(i)} - 1)\text{log}(1 - g^{[L]}(W^{[L]}a^{(L-1)(i)} + b^{[L]})) - y^{(i)}\text{log}(g^{[L]}(W^{[L]}a^{[L-1](i)} + b^{[L]})) \end{aligned}$$

this is calculated in a vectorized implementation by using the vector $A^{[L]}$ calculated during forward propagation and the output testing data vector. The mean is then taken of the resulting vector after calculations.

Back-propagation

The relevant partial derivatives of the empirical risk/cost function required for the weight updates in *back-propagation* are calculated for the last few layers then generalized for any layer below

1. Output layer L

   The derivative of the cost/empirical risk with respects to the neuron weights $W^{[L]}$ of the output layer is

$$\begin{aligned}
\frac{\partial R}{\partial W^{[L]}} &= \frac{\partial}{\partial W^{[L]}}\left[\frac{1}{N}\sum_{i=0}^{N} L(y^{(i)},\hat{y}^{(i)})\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial W^{[L]}}\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial a^{[L](i)}}\frac{\partial a^{[L](i)}}{\partial z^{[L](i)}}\frac{\partial z^{[L](i)}}{\partial W^{[L]}}\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[(y^{(i)}-1)\frac{-1}{1-a^{[L](i)}} - y^{(i)}\frac{1}{a^{[L](i)}}\right]\left[g'^{[L]}\right]\left[a^{[L-1](i)}\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[(y^{(i)}-1)\frac{-1}{1-g^{[L]}} - y^{(i)}\frac{1}{g^{[L]}}\right]\left[g'^{[L]}\right]\left[a^{[L-1](i)}\right] \quad \text{*NOTE } g^{[L]} \equiv g^{[L]}(z^{[L](i)}) \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[(y^{(i)}-1)\frac{-1}{1-g^{[L]}} - y^{(i)}\frac{1}{g^{[L]}}\right]\left[g^{[L]}(1-g^{[L]})\right]\left[a^{[L-1](i)}\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[(1-y^{(i)})g^{[L]} - y^{(i)}(1-g^{[L]})\right]\left[a^{[L-1](i)}\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[g^{[L]} - y^{(i)}g^{[L]} - y^{(i)} + y^{(i)}g^{[L]}\right]\left[a^{[L-1](i)}\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[g^{[L]} - y^{(i)}\right]\left[a^{[L-1](i)}\right]
\end{aligned}$$

   more concisely the derivative is given by

$$\frac{\partial R}{\partial W^{[L]}} = \frac{1}{N}\sum_{i=0}^{N}\frac{\partial L}{\partial z^{[L](i)}}\frac{\partial z^{[L](i)}}{\partial W^{[L]}} = \frac{1}{N}\sum_{i=0}^{N}\left[\left(g^{[L]}(z^{[L](i)}) - y^{(i)}\right)a^{[L-1](i)}\right]$$

   note the derivative of the logistic activation function is

$$\begin{aligned}
g'(z) &= -1(1+e^{-z})^{-2}(e^{-z})(-1) \\
&= (1+e^{-z})^{-1}\frac{e^{-z}}{1+e^{-z}} \\
&= (1+e^{-z})^{-1}\frac{e^{-z}+[+1-1]}{1+e^{-z}} \quad \text{*using trick to simplify} \\
&= (1+e^{-z})^{-1}\frac{1+e^{-z}-1}{1+e^{-z}} \\
&= (1+e^{-z})^{-1}\left[\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right] \\
&= (1+e^{-z})^{-1}\left[1-\frac{1}{1+e^{-z}}\right] \\
&= g(z)(1-g(z))
\end{aligned}$$

and the derivative of the cost/empirical risk with respects to $b^{[L]}$ is

$$\begin{aligned}
\frac{\partial R}{\partial b^{[L]}} &= \frac{\partial}{\partial b^{[L]}}\left[\frac{1}{N}\sum_{i=0}^{N} L(y^{(i)},\hat{y}^{(i)})\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial b^{[L]}}\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial a^{[L](i)}}\frac{\partial a^{[L](i)}}{\partial z^{[L](i)}}\frac{\partial z^{[L](i)}}{\partial b^{[L]}}\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[(y^{(i)}-1)\frac{-1}{1-a^{[L](i)}} - y^{(i)}\frac{1}{a^{[L](i)}}\right]\left[g^{[L]}(1-g^{[L]})\right][1] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[(1-y^{(i)})g^{[L]} - y^{(i)}(1-g^{[L]})\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[g^{[L]} - y^{(i)}g^{[L]} - y^{(i)} + y^{(i)}g^{[L]}\right] \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[g^{[L]} - y^{(i)}\right]
\end{aligned}$$

more concisely the derivative is given by

$$\frac{\partial R}{\partial b^{[L]}} = \frac{1}{N}\sum_{i=0}^{N}\frac{\partial L}{\partial z^{[L](i)}}\frac{\partial z^{[L](i)}}{\partial b^{[L]}} = \frac{1}{N}\sum_{i=0}^{N}\left[g^{[L]}(z^{[L](i)}) - y^{(i)}\right]$$

Remark: The back-propagation derivation can be written more intuitively by applying the chain rule to the empirically risk/cost function directly as

$$\begin{aligned}
\frac{\partial R}{\partial W^{[L]}} &= \frac{\partial R}{\partial A^{[L]}}\frac{\partial A^{[L]}}{\partial Z^{[L]}}\frac{\partial Z^{[L]}}{\partial W^{[L]}} \\
&= \left[\frac{1}{N}\sum_{i=0}^{N}\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial a^{[L](i)}}\right]\frac{\partial A^{[L]}}{\partial Z^{[L]}}\frac{\partial Z^{[L]}}{\partial W^{[L]}} \\
&= \frac{1}{N}\sum_{i=0}^{N}\left[\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial a^{[L](i)}}\frac{\partial a^{[L](i)}}{\partial z^{[L](i)}}\frac{\partial z^{[L](i)}}{\partial W^{[L](i)}}\right]
\end{aligned}$$

the actual calculation of the derivative used the chain rule on the loss function L instead of directly on R. The derivatives with respects to $b^{[L]}$ may be written similarly. The method of application of the chain rule for the derivation seems more intuitive but not sure if it is mathematically correct, may need corrections from matrix calculus theory.

Remark: When applying the chain rule to derive the derivatives of back-propagation, make sure to take derivatives of functions with respects to their variables instead of its functions (unless explicitly trying to do incorporate functional derivatives). For more details in possibly incorporating functional derivatives and maybe varying loss functions during training look at appendix A.1

2. Layer $L-1$

The derivative of the cost/empirical risk with respects to the neuron weights $W^{[L-1]}$ of layer

$L-1$ is

$$\begin{aligned}\frac{\partial R}{\partial W^{[L-1]}} &= \frac{\partial}{\partial W^{[L-1]}}\left[\frac{1}{N}\sum_{i=0}^{N} L(y^{(i)},\hat{y}^{(i)})\right]\\
&= \frac{1}{N}\sum_{i=0}^{N}\left[\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial W^{[L-1]}}\right]\\
&= \frac{1}{N}\sum_{i=0}^{N}\left[\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial z^{[L](i)}}\frac{\partial z^{[L](i)}}{\partial a^{[L-1](i)}}\frac{\partial a^{[L-1](i)}}{\partial z^{[L-1](i)}}\frac{\partial z^{[L-1](i)}}{\partial W^{[L-1]}}\right]\\
&= \frac{1}{N}\sum_{i=0}^{N}\left[g^{[L]}-y^{(i)}\right]\left[W^{[L]}\right]\left[g'^{[L-1]}\right]\left[a^{[L-2]}\right]\end{aligned}$$

more concisely the derivative is given by

$$\frac{\partial R}{\partial W^{[L-1]}} = \frac{1}{N}\sum_{i=0}^{N}\frac{\partial L}{\partial z^{[L-1](i)}}\frac{\partial z^{[L-1](i)}}{\partial W^{[L-1]}} = \frac{1}{N}\sum_{i=0}^{N}\left[\left(g^{[L]}(z^{[L](i)})-y^{(i)}\right)\left(W^{[L]}\right)\left(g'^{[L-1]}(z^{[L-1]})\right)\right]\left[a^{[L-2](i)}\right]$$

note the following useful functions for the derivation

$$\begin{aligned}z^{[L]} &= W^{[L]}a^{[L-1]}+b^{[L]}\\
&= W^{[L]}g^{[L-1]}(z^{[L-1]})+b^{[L]}\\
&= W^{[L]}g^{[L-1]}\left(W^{[L-1]}a^{[L-2]}+b^{[L-1]}\right)+b^{[L]}\end{aligned}$$

the ReLU activation function is

$$g(z)=\begin{cases}-y(z) & \text{misclassified}\\ 0 & \text{else}\end{cases}$$

or

$$g(z)=\min(-y(z),0)$$

its derivative is

$$g(z)=\begin{cases}-y & \text{misclassified}\\ 0 & \text{else}\end{cases}$$

or

$$g(z)=\min(-y,0)$$

The derivative of the cost/empirical risk with respects to $b^{[L-1]}$ is

$$\begin{aligned}\frac{\partial R}{\partial b^{[L-1]}} &= \frac{\partial}{\partial b^{[L-1]}}\left[\frac{1}{N}\sum_{i=0}^{N} L(y^{(i)},\hat{y}^{(i)})\right]\\
&= \frac{1}{N}\sum_{i=0}^{N}\left[\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial b^{[L-1]}}\right]\\
&= \frac{1}{N}\sum_{i=0}^{N}\left[\frac{\partial L(y^{(i)},\hat{y}^{(i)})}{\partial z^{[L](i)}}\frac{\partial z^{[L](i)}}{\partial a^{[L-1](i)}}\frac{\partial a^{[L-1](i)}}{\partial z^{[L-1](i)}}\frac{\partial z^{[L-1](i)}}{\partial b^{[L-1]}}\right]\\
&= \frac{1}{N}\sum_{i=0}^{N}\left[g^{[L]}-y^{(i)}\right]\left[W^{[L]}\right]\left[g'^{[L-1]}\right][1]\end{aligned}$$

more concisely the derivative is given by

$$\begin{aligned}
\frac{\partial R}{\partial b^{[L-1]}} &= \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial z^{[L-1](i)}} \frac{\partial z^{[L-1](i)}}{\partial b^{[L-1]}} \\
&= \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial z^{[L-1](i)}} \\
&= \frac{1}{N}\sum_{i=0}^{N} \left[\left(g^{[L]}(z^{[L](i)}) - y^{(i)}\right)\left(W^{[L]}\right)\left(g'^{[L-1]}(z^{[L-1](i)})\right)\right]
\end{aligned}$$

L-n. Layer $L-n$ where $n \in \mathbb{R} : 0 \leq n \leq L$

The chain rule is repeatedly implemented to calculated the derivatives of the weights at each layer.

The derivatives are given in terms of the derivative of loss with respects to linear combination of inputs z as

$$\frac{\partial R}{\partial W^{[L-n]}} = \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial z^{[L-n](i)}} \frac{\partial z^{[L-n](i)}}{\partial W^{[L-n]}} = \frac{1}{N}\sum_{i=0}^{N} \left[\frac{\partial L}{\partial z^{[L-n](i)}}\right]\left[a^{[L-n-1](i)}\right]$$

and

$$\frac{\partial R}{\partial b^{[L-n]}} = \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial z^{[L-n](i)}} \frac{\partial z^{[L-n](i)}}{\partial b^{[L-n]}} = \frac{1}{N}\sum_{i=0}^{N} \left[\frac{\partial L}{\partial z^{[L-n](i)}}\right][1]$$

where the derivative of loss with respects to linear combination of inputs z is

$$\frac{\partial L}{\partial z^{[L-n]}} = \frac{\partial L}{\partial z^{[L-n+1]}} \frac{\partial z^{[L-n+1]}}{\partial a^{[L-n]}} \frac{\partial a^{[L-n]}}{\partial z^{[L-n]}} = \left[\frac{\partial L}{\partial z^{[L-n+1]}}\right]\left[W^{[L-n+1]}\right]\left[g'^{[L-n]}(z^{[L-n]})\right]$$

note the layer variable $L-n$ is a dummy variable and can be replaced with a variable l for cleaner notation.

Implementation

For implementation consider the formalism below for the relevant partial derivatives needed for back-propagation of an layer l. The partial derivative of the empirical risk/cost function with respects to the weights of layer l are

$$\begin{aligned}\frac{\partial R}{\partial W^{[l]}} &= \frac{1}{N}\sum_{i=0}^{N}\frac{\partial L}{\partial z^{[l](i)}}\frac{\partial z^{[l](i)}}{\partial W^{[l]}}\\ &= \frac{1}{N}\sum_{i=0}^{N}\frac{\partial L}{\partial z^{[l](i)}}\left[a^{[l-1](i)}\right]\end{aligned}$$

and

$$\begin{aligned}\frac{\partial R}{\partial b^{[l]}} &= \frac{1}{N}\sum_{i=0}^{N}\frac{\partial L}{\partial z^{[l](i)}}\frac{\partial z^{[l](i)}}{\partial b^{[l]}}\\ &= \frac{1}{N}\sum_{i=0}^{N}\frac{\partial L}{\partial z^{[l](i)}}[1]\\ &= \frac{1}{N}\sum_{i=0}^{N}\frac{\partial L}{\partial z^{[l](i)}}\end{aligned}$$

the additional calculation of the partial derivatives with respects to the linear combination of inputs of each neuron z are

$$\begin{aligned}\frac{\partial L}{\partial z^{[l](i)}} &= \frac{\partial L}{\partial a^{[l](i)}}\frac{\partial a^{[1](i)}}{\partial z^{[l](i)}}\\ &= \left[\frac{\partial L}{\partial a^{[l](i)}}\right]\left[g'^{[l]}(z^{[l](i)})\right]\end{aligned}$$

and the partial derivatives with respects to the activation for all layers except the output layer are

$$\begin{aligned}\frac{\partial L}{\partial a^{[l](i)}} &= \frac{\partial L}{\partial z^{[l+1](i)}}\frac{\partial z^{[l+1](i)}}{\partial a^{[l](i)}}\\ &= \left[\frac{\partial L}{\partial z^{[l+1](i)}}\right]\left[W^{[l+1]}\right]\end{aligned}$$

with the partial derivative for the output layer L given by

$$\frac{\partial L}{\partial a^{[L](i)}} = \left[(y^{(i)}-1)\frac{-1}{1-a^{[L](i)}} - y^{(i)}\frac{1}{a^{[L](i)}}\right]$$

Remark: for an alternate formalism of the equations for implementing back-propagation look at appendix A.2

Remark: Variables/values needed in order to do calculations when updating weights for a layer l

1. For calculations of $\frac{\partial L}{\partial a^{[l]}}$
   (a) $\frac{\partial L}{\partial z^{[l+1]}}$ from previous layer
   (b) $W^{[l+1]}$ from previous layer
2. For calculation of $\frac{\partial L}{\partial z^{[l]}}$
   (a) $z^{[l]}$ from current layer

   (b) $g'^{[l]}(z^{[l]})$ from current layer

3. For partials with respects to weights

   (a) $a^{[l-1]}$ from the next layer

For derivation of the vectorized implementation consider the calculations below. Let $n^{[l]}$ be the number of neurons/nodes in layer l.

1. Output layer L (the layer is a single neuron/node)

$$\frac{\partial L}{\partial A^{[L]}} = \left[\frac{\partial L}{\partial a^{[L](1)}}, \frac{\partial L}{\partial a^{[L](2)}}, \dots, \frac{\partial L}{\partial a^{[L](N)}}\right]$$

this is a $(n^{[L]} \times N) = (1 \times N)$ row vector which are the dimension of $A^{[L]}$.

$$\begin{aligned}
\frac{\partial L}{\partial Z^{[L]}} &= \left[\frac{\partial L}{\partial z^{[L](1)}}, \frac{\partial L}{\partial z^{[L](2)}}, \dots, \frac{\partial L}{\partial z^{[L](N)}}\right] \\
&= \left[\frac{\partial L}{\partial a^{[L](1)}} * g'^{[L]}(z^{[L](1)}), \frac{\partial L}{\partial a^{[L](2)}} * g'^{[L]}(z^{[L](2)}), \dots, \frac{\partial L}{\partial a^{[L](N)}} * g'^{[L]}(z^{[L](N)})\right] \\
&= \left[\frac{\partial L}{\partial a^{[L](1)}}, \frac{\partial L}{\partial a^{[L](2)}}, \dots, \frac{\partial L}{\partial a^{[L](N)}}\right] * \left[g'^{[L]}(z^{[L](1)}), g'^{[L]}(z^{[L](2)}), \dots, g'^{[L]}(z^{[L](N)})\right] \\
&= \frac{\partial L}{\partial A^{[L]}} * g'^{[L]}(Z^{[L]})
\end{aligned}$$

this is a $(n^{[L]} \times N) = (1 \times N)$ row vector which are the dimension of $Z^{[L]}$.

$$\begin{aligned}
\frac{\partial R}{\partial W^{[L]}} &= \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial z^{[L](i)}} a^{[L-1](i)} \\
&= \frac{1}{N}\left[\frac{\partial L}{\partial z^{[L](1)}} a^{[L-1](1)} + \frac{\partial L}{\partial z^{[L](2)}} a^{[L-1](2)} + \dots + \frac{\partial L}{\partial z^{[L](N)}} a^{[L-1](N)}\right] \\
&= \frac{1}{N}\left[\frac{\partial L}{\partial z^{[L](1)}}, \frac{\partial L}{\partial z^{[L](2)}}, \dots, \frac{\partial L}{\partial z^{[L](N)}}\right]\left[a^{[L-1](1)}, a^{[L-1](2)}, \dots, a^{[L-1](N)}\right]^T \\
&= \frac{1}{N}\frac{\partial L}{\partial Z^{[L]}} A^{[L-1]T}
\end{aligned}$$

this is a $(n^{[L]} \times n^{[L-1]}) = (1 \times n^{[L-1]})$ row vector which are the dimensions of $W^{[L]}$.

$$\frac{\partial R}{\partial b^{[L]}} = \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial z^{[L](i)}}$$

each element in the sum is a $(n^{[L]} \times 1) = (1, 1)$ scalar and the resulting vector is a $(n^{[L]} \times 1) = (1, 1)$ scalar which is the dimensions of $b^{[L]}$.

2. Output layer $L - 1$

$$\begin{aligned}
\frac{\partial L}{\partial A^{[L-1]}} &= \left[\frac{\partial L}{\partial a^{[L](1)}}, \frac{\partial L}{\partial a^{[L](2)}}, \dots, \frac{\partial L}{\partial a^{[L](N)}}\right] \\
&= \left[\frac{\partial L}{\partial z^{[L](1)}} W^{[L]}, \frac{\partial L}{\partial z^{[L](2)}} W^{[L]}, \dots, \frac{\partial L}{\partial z^{[L](N)}} W^{[L]}\right] \\
&= W^{[L]T}\frac{\partial L}{\partial Z^{[L]}}
\end{aligned}$$

this is a $(n^{[L-1]} \times N)$ matrix which are the dimensions of $A^{[L-1]}$. The respective partial derivatives of a sample is a $(n^{[L-1]} \times 1)$ column vector.

$$\begin{aligned}
\frac{\partial L}{\partial Z^{[L-1]}} &= \left[\frac{\partial L}{\partial z^{[L-1](1)}}, \frac{\partial L}{\partial z^{[L-1](2)}}, \dots, \frac{\partial L}{\partial z^{[L-1](N)}}\right] \\
&= \left[\frac{\partial L}{\partial a^{[L-1](1)}} * g'^{[l]}(z^{[L-1](1)}), \frac{\partial L}{\partial a^{[L-1](2)}} * g'^{[l]}(z^{[L-1](2)}), \dots, \frac{\partial L}{\partial a^{[L-1](N)}} * g'^{[l]}(z^{[L-1](N)})\right] \\
&= \frac{\partial L}{\partial A^{[L-1]}} * g'^{[L-1]}(Z^{[L-1]})
\end{aligned}$$

this is a $(n^{[L-1]} \times N)$ matrix which are the dimensions of $A^{[L-1]}$. The respective partial derivatives of a sample is a $(n^{[L-1]} \times 1)$ column vector.

$$\begin{aligned}
\frac{\partial R}{\partial W^{[L-1]}} &= \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial z^{[L-1](i)}} a^{[L-2](i)T} \\
&= \frac{1}{N}\left[\frac{\partial L}{\partial z^{[L-1](1)}} a^{[L-2](1)T} + \frac{\partial L}{\partial z^{[L-1](2)}} a^{[L-2](2)T} + \dots + \frac{\partial L}{\partial z^{[L-1](N)}} a^{[L-2](N)}\right] \\
&= \frac{1}{N}\left[\frac{\partial L}{\partial z^{[L-1](1)}}, \frac{\partial L}{\partial z^{[L-1](2)}}, \dots, \frac{\partial L}{\partial z^{[L-1](N)}}\right]\left[a^{[L-2](1)}, a^{[L-2](2)}, \dots, a^{[L-2](N)}\right]^T \\
&= \frac{1}{N}\frac{\partial L}{\partial Z^{[L-1]}} A^{[L-2]T}
\end{aligned}$$

this is a $(n^{[L-1]} \times n^{[L-2]})$ matrix which are the dimensions of $W^{[L-1]}$.

$$\frac{\partial R}{\partial b^{[L-1]}} = \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial z^{[L-1](i)}}$$

each element in the sum is a $(n^{[L-1]} \times 1)$ column vector and the resulting vector is a $(n^{[L-1]} \times 1)$ column vector which are the dimensions of $b^{[L-1]}$.

L-n. Layer $L - n$ where $0 \leq n < L$ From the previous explicit calculations the vectorized implementation for the calculations of the partial derivatives for N training samples at each layer are generalized as

$$\frac{\partial L}{\partial A^{[l]}} = W^{[l+1]T}\frac{\partial L}{\partial Z^{[l+1]}}$$

the dimension are

$$(n^{[l]} \times n^{[l+1]})(n^{[l+1]} \times N) = (n^{[l]} \times N)$$

The respective partials for the output layer $\frac{\partial L}{\partial A^{[L]}}$ can be calculated through vector operations.

$$\frac{\partial L}{\partial Z^{[l]}} = \frac{\partial L}{\partial A^{[l]}} * g'^{[l]}(Z^{[l]})$$

where * denotes element-wise multiplication. The dimension are

$$(n^{[l]} \times N) * (n^{[l]} \times N) = (n^{[l]} \times N)$$

$$\frac{\partial R}{\partial W^{[l]}} = \frac{1}{N}\frac{\partial L}{\partial Z^{[l]}} A^{[l-1]T}$$

the dimension are

$$(n^{[l]} \times N) * (N \times n^{[l-1]}) = (n^{[l]} \times n^{[l-1]})$$

$$\frac{\partial R}{\partial b^{[l]}} = \frac{1}{N} \sum_{i=0}^{N} \frac{\partial L}{\partial z^{[l](i)}}$$

the dimension are

$$(n^{[l]} \times 1)$$

Remark: This probably could have been derived during calculation of the relevant partial derivatives with some knowledge of matrix calculus instead of generalized from a few calculations. As an exercise try to do so after learning some matrix calculus.

After forward propagation by iterating the NN backwards starting at the output layer the weights can be updated using this vectorized approach and by passing the required values for calculations from forward propagation and the previous iteration.

Train on batches of 32 data points or less per iteration (cite yann LecCun)

4 (1.5.4) Inference and decision

* Pattern Recognition and Machine Learning

The classification problem can be broken down into two stages

- *Inference stage:* use training data to learn a model for $p(C_k|\mathbf{x})$ where C_k is the k^{th} classification class and $\mathbf{x}$ is the data.
- *Decision stage:* use these posterior probabilities to make optimal class assignments [of new data].

Remark: Another approach to the classification problem would be to solve both problems at once by learning a function that maps the input data x to the class assignments/decisions called a *discriminant function.*

There are three distinct approaches to solving the decision/classification problem that tackle the stages differently. Given in decreasing order of complexity they are

1. **Generative models**
   First solve the *inference problem*
   (a) Find the class-conditional densities $p(\mathbf{x}|C_k)$ for each C_k individually.
   (b) Separately infer the prior class probabilities $p(C_k)$.
   (c) Then use Bayes's theorem in the form
   $$\begin{aligned} p(C_k|\mathbf{x}) &= \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_j p(\mathbf{x}|C_j)p(C_j)} \end{aligned}$$
   to find the posterior class probabilities $p(C_k|\mathbf{x})$. Note the data marginal distribution can be found using the class-conditionals and priors.
   (d) Remark:
   Equivalently, the joint distribution $p(\mathbf{x}, C_k)$ can be modeled directly and then normalized by the data marginal distribution $p(\mathbf{x})$ to obtain the the posterior class probabilities $p(C_k, \mathbf{x})$.

   Having found the posterior class distribution/probabilities, use decision theory to determine class membership of new input data $\mathbf{x}$.

   Approaches that explicitly or implicitly model the distribution of the input and outputs are known as *Generative models*, because these distributions (data marginal distribution, posterior class probabilities, etc.) can be sampled from to generate synthetic data points. It can generate data points and respective classification in the input space.

   Remarks:

   - Note that the class priors $p(C_k)$ can often be estimated simply from the fractions of the training set data points in each of the classes.
   - Ultimately trying to understand/learn the joint probability $p(\mathbf{x}, C_k)$, which can be reasonable or not depending on problem.
   - Question: How is the sampling of synthetic data actually done?
   - Personal experience after implementation:

2. **Discriminative models**
   First solve the *inference problem* by directly determining the posterior class probabilities $p(C_k|\mathbf{x})$.

   Subsequently use decision theory to determine class membership of new input data $\mathbf{x}$.

   Approaches that model the posterior probabilities directly are known as *discriminative models*.

   Remarks:

   - Includes logistic regression, SVM, etc. methods.
   - Question: How is the posterior probabilities learned directly?
   - Personal experience after implementation:

3. **Discriminant function**
   Directly learn a function f(x) called a *discriminant function* that maps the input $\mathbf{x}$ onto a class label. This can be accomplished through regression.

   In this approach probabilities play no role.

   Remarks:

   - Question:
   - Personal experience after implementation:

The generative approach (1) is the most demanding. However it offers some advantages such as

- It allows the marginal density of data $p(x)$ to be determined from. This can be useful for detecting new data points that have low probability under the model and for which the predictions may be of low accuracy. This is known as outlier detection or novelty detection.

however if only looking to find the posterior probabilities $p(C_k|\mathbf{x})$ the discriminative approach (2) is much more reasonable.

The discriminant function approach (3) is the simplest. However, it does not compute the posterior class probabilities which may be useful for reasons such as

- Minimization risk
- Reject option
- Compensating for class priors
  *when dealing with extremely rare events, with regression it is hard to avoid the trivial solution with misleading high classification accuracy as the model will have to only classify the majority of data as "normal" to achieve high accuracy.
- Combining models
  *for complex problems, instead of combining all inputs into one input space can break up the problem into finding posterior class probabilities for each input and then combine them using probability to find the posterior class probability given all inputs (example: naive Bayes model).

5 Generative Adversarial Nets

* Ian J.Goodfellow. Research journal

- *Discriminative models:*
  “Approaches that model the posterior probabilities directly are called discriminative models”- pattern recognition and machine learning
- *Generative models:*
  “Approaches that explicitly or implicitly model the distribution of inputs as well as outputs are known as generative models” - Pattern recognition and machine learning
- *Adversarial process*:
  “Supports conflicting one-sided positions held by individuals, groups or entire societies, as inputs into the conflict resolution situation.”

(3) Adversarial nets

Implementation for generative and discriminative models as multilayer perceptrons.

Let p_g represent the generator distribution over data $\mathbf{x}$, $p_z(\mathbf{z})$ be a prior of the input noise variables, and $G(\mathbf{z}; \theta_g)$ represent a mapping to data space. G is a differentiable function [that outputs a high dimensional vector] represented by a multilayer perceptron with parameters θ_g. This defines the generative model framework.

Question: what does it mean for a function to be represented by a MLP and what does that look like?

Let $D(\mathbf{x}, \theta_d)$ be another differentiable function represented by a second multilayer perceptron with parameters θ_d that outputs a single scalar. $D(\mathbf{x})$ represents the probability that $\mathbf{x}$ came from the data rather than the generator distribution p_g. This defines the discriminative model framework.

D is trained to maximize the probability of assigning the correct label to both training examples and samples from G. G is simultaneously trained to minimize the objective function

$$log(1 - D(G(\mathbf{z})))$$

where $1 - D(G(\mathbf{z}))$ is the probability that given a noise input that the output data $\mathbf{z}$ came from the generator distribution p_g rather than the data. The motivation for minimizing this distribution can be interpreted as wanting to minimize the probability that a data point came from p_g especially if it was generated by p_g implying that the discriminative model is more confident that a sampled/"synthetic" data point came from the data rather then the generative model, effectively fooling the discriminative model. Assuming a competent discrimantive model this should result in a generative model that can produce realistic data as possible and effectively has learned the data distribution.

In other words, D and G are playing the following two-player minimax game with the value [objective ?] function $V(G, D)$:

$$min$$

A Appendix

A.1 Back-Propagation with changes in Loss Function

When applying the chain rule in back-propagation, make sure to take derivatives of functions with respects to their variables instead of its functions (unless explicitly trying to do incorporate functional derivatives as explored below)

$R[L(y, \hat{y})]$ is a functional of L. Taking the variation of R gives

$$\begin{aligned}
\left[\frac{d\ i(\tau)}{d\tau}\right]\bigg|_{\tau=0} &= \left[\frac{d\ R[L(y,\hat{y}) + \tau\phi(y,\hat{y})]}{d\tau}\right]\bigg|_{\tau=0} \\
&= \left[\frac{d}{d\tau}\frac{1}{N}\sum^{N}\left(L(y,\hat{y}) + \tau\phi(y,\hat{y})\right)\right]\bigg|_{\tau=0} \\
&= \left[\frac{1}{N}\sum^{N}\phi(y,\hat{y})\right]\bigg|_{\tau=0} \\
&= \sum^{N}\left[\frac{1}{N}\right]\phi(y,\hat{y})
\end{aligned}$$

equating to the directional derivatives gives

$$\sum^{N}\frac{\delta R}{\delta L}\phi(y,\hat{y}) = \sum^{N}\left[\frac{1}{N}\right]\phi(y,\hat{y})$$

implying

$$\frac{\delta R}{\delta L} = \frac{1}{N}$$

Question: How can we use this variational derivative? (compare to variational derivative of entropy with respects to probability mass functions)
Question: Knowing functional derivative of R, can we write the below?

$$\frac{\partial R}{\partial W^{[L]}} = \frac{\delta R}{\delta L}\frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial W^{[L]}}$$

note the variational derivative of R should be zero if L does not change.

A.2 Back-Propagation implementation using alternate formulation of calculations

Alternate Implementation

For implementation purposes consider the conventions for the derivatives with the layer variable l as below

$$\begin{aligned}\frac{\partial R}{\partial W^{[l]}} &= \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial a^{[l](i)}} \frac{\partial a^{[l](i)}}{\partial z^{[l](i)}} \frac{\partial z^{[l](i)}}{\partial W^{[l]}} \\ &= \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial a^{[l](i)}} \left[g'^{[l]}(z^{[l](i)})\right] \left[a^{[l-1](i)}\right]\end{aligned}$$

and

$$\begin{aligned}\frac{\partial R}{\partial b^{[l]}} &= \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial a^{[l](i)}} \frac{\partial a^{[l](i)}}{\partial z^{[l](i)}} \frac{\partial z^{[l](i)}}{\partial b^{[l]}} \\ &= \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial a^{[l](i)}} \left[g'^{[l]}(z^{[l](i)})\right] [1] \\ &= \frac{1}{N}\sum_{i=0}^{N} \frac{\partial L}{\partial a^{[l](i)}} \left[g'^{[l]}(z^{[l](i)})\right]\end{aligned}$$

the additional calculation of the partial derivatives with respects to the activation for all layers l except the output layer are

$$\begin{aligned}\frac{\partial L}{\partial a^{[l](i)}} &= \frac{\partial L}{\partial z^{[l+1](i)}} \frac{\partial z^{[l+1](i)}}{\partial a^{[l](i)}} \\ &= \frac{\partial L}{\partial a^{[l+1](i)}} \frac{\partial a^{[l+1](i)}}{\partial z^{[l+1](i)}} \frac{\partial z^{[l+1](i)}}{\partial a^{[l](i)}} \\ &= \left[\frac{\partial L}{\partial a^{[l+1](i)}}\right] \left[g'^{[l+1]}(z^{[l+1](i)})\right] \left[W^{[l+1]}\right]\end{aligned}$$

with the respective partial derivative for the output layer L given by

$$\frac{\partial L}{\partial a^{[L](i)}} = \left[(y^{(i)} - 1)\frac{-1}{1 - a^{[L](i)}} - y^{(i)}\frac{1}{a^{[L](i)}}\right]$$

Remark: Variables/values needed in order to do calculations when updating weights for a layer l

1. For calculations of $\frac{\partial L}{\partial a^{[l]}}$
   (a) $\frac{\partial L}{\partial a^{[l+1]}}$ from previous layer
   (b) $z^{[l+1]}$ from previous layer
   (c) $g'^{[l+1]}(z^{[l+1]})$ from previous layer
   (d) $W^{[l+1]}$ from previous layer

   note most of these values are only used to recompute $\frac{\partial L}{\partial z^{[l+1]}}$ from the previous layer and not needed elsewhere.
2. For calculations of partials with respects to weights
   (a) $z^{[l]}$ from current layer

(b) $g'^{[l]}(z^{[l]})$ from current layer, note this will be computed twice/used twice in current implementation

(c) $a^{[l-1]}$ from next layer

Remark: The equations and calculations of the partial derivatives in this formalism are intuitive and more concise however its implementation is inefficient. Many quantities in this formalism are only used to compute a value from the previous layer instead of passing the computed value from the previous layer. The other formalism with its more efficient implementation will be considered over this one.