

10-315 Notes

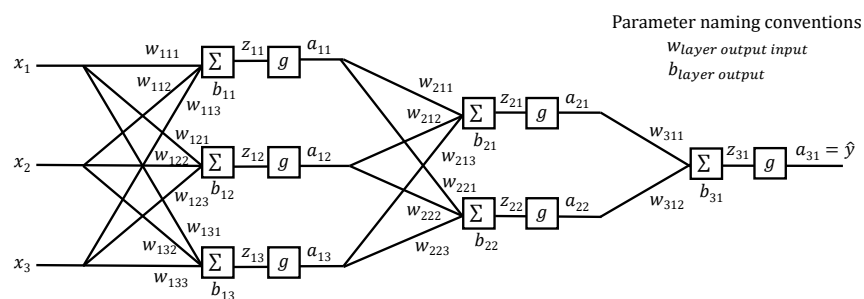
Neural Networks

Carnegie Mellon University
Machine Learning Department

Contents

1	Neural Networks	1
1.1	Networks Diagrams for Linear and Logistic Regression	2
1.2	Neuron	3
1.3	Activation functions	3
1.4	Three Neuron Neural Network	3
1.5	Adding More Neurons	5
1.5.1	Fully-connected networks	5
1.5.2	Vocab	5
1.5.3	Counting parameters	6
2	Calculus Background	8
2.1	Chain Rule	8
2.2	Multivariate Chain Rule	10
2.2.1	Adding More Functions	10
2.2.2	Vector Notation of Chain Rule	12
2.2.3	Chain Rule with Multiple Inputs	13

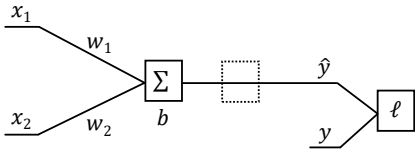
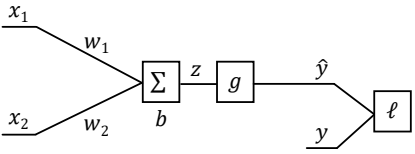
1 Neural Networks



1.1 Networks Diagrams for Linear and Logistic Regression

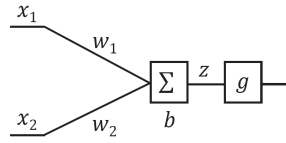
Neural networks, including the diagram above, can be intimidating. But, you actually know so much about them already!

To help you see that and make the transition to neural networks. Let's summarize both linear and logistic regression using the diagrams and notation that we'll be using for neural networks. A key difference is writing the partial derivatives of the objective function using the chain rule.

	Linear Regression	Logistic Regression
Network diagram		
Model	$\hat{y} = h_{\theta}(\mathbf{x}) = b + \sum_j w_j x_j$	$\hat{y} = h_{\theta}(\mathbf{x}) = g_{\text{logistic}}\left(b + \sum_j w_j x_j\right)$
Parameters	w_1, w_2, b	
Loss function	Squared error $\ell(y, \hat{y}) = (y - \hat{y})^2$	Cross-entropy $\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^K y_k \log \hat{y}_k$
Objective	$J(\theta) = \frac{1}{N} \sum_{i=1}^N \ell(y^{(i)}, \hat{y}^{(i)})$	
Derivatives (using chain rule)	$\begin{aligned} \frac{\partial J}{\partial w_j} &= \frac{1}{N} \sum_{i=1}^N \frac{\partial \ell}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial w_j} \\ &= \frac{1}{N} \sum_{i=1}^N (-2(y^{(i)} - \hat{y}^{(i)})) \cdot x_j^{(i)} \\ &= -\frac{2}{N} \sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} \end{aligned}$	$\begin{aligned} \frac{\partial J}{\partial w_j} &= \frac{1}{N} \sum_{i=1}^N \frac{\partial \ell}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial w_j} \\ &= -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K \frac{y_k^{(i)}}{\hat{y}_k^{(i)}} \cdot \hat{y}^{(i)} (1 - \hat{y}^{(i)}) \cdot x_j^{(i)} \\ &= -\frac{1}{N} \sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} \end{aligned}$
	$\begin{aligned} \frac{\partial J}{\partial b} &= \frac{1}{N} \sum_{i=1}^N \frac{\partial \ell}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial b} \\ &= \frac{1}{N} \sum_{i=1}^N (-2(y^{(i)} - \hat{y}^{(i)})) \cdot 1 \\ &= -\frac{2}{N} \sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)}) \end{aligned}$	$\begin{aligned} \frac{\partial J}{\partial b} &= \frac{1}{N} \sum_{i=1}^N \frac{\partial \ell}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial b} \\ &= \frac{1}{N} \sum_{i=1}^N -\frac{1}{\hat{y}_k^{(i)}} \cdot \hat{y}_k^{(i)} (1 - \hat{y}_k^{(i)}) \cdot 1 \\ &= -\frac{1}{N} \sum_{i=1}^N (1 - \hat{y}_k^{(i)}) \end{aligned}$
SGD update	$w_j \leftarrow w_j - \alpha \frac{\partial J^{(i)}}{\partial w_j}$ $b \leftarrow b - \alpha \frac{\partial J^{(i)}}{\partial b}$	

1.2 Neuron

The logistic regression function is a neuron! A neuron is a linear function followed by an **activation function**. The linear function, just like linear regression and logistic regression, is a linear combination of the input values multiplied by weight parameters and bias parameter, $z = b + \sum_j w_j x_j$.

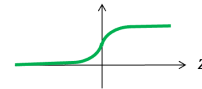


1.3 Activation functions

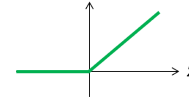
The activation function in a neuron is basically any nonlinear function. Common neuron activation functions include the logistic (sigmoid) function and the **ReLU** (rectified linear unit) function, which just takes the input value and clamps any negative values to zero.

▪ Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$ $\frac{dg}{dz} = g(z)(1 - g(z))$

▪ (Softmax)



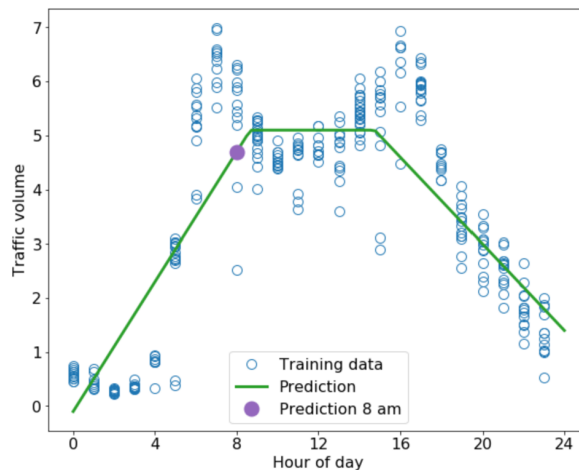
▪ ReLU: $g(z) = \max(0, z)$ $\frac{dg}{dz} = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$



1.4 Three Neuron Neural Network

Consider the traffic volume prediction dataset. Clearly, a linear model with just the single x_1 input feature will not suffice. We've learned that we could use polynomial feature engineering to create $x_2 = x_1^2$, and then use a linear model to effectively fit a quadratic function to this data. Alternatively, we can start building a simple, three-neuron neural network.

With two separate neurons, we can try to separately model the upward trend and downward trends in the data for the morning and evening, respectively. Then a third neuron can combine these to model, resulting in the green prediction function in the following figure. The input to the third neuron is just the output of the previous two neurons.



Parameters

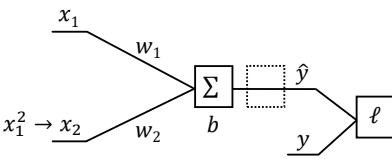
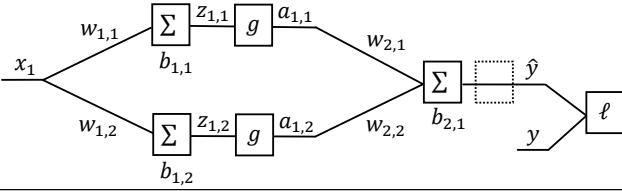
$$w_{1,1}: 0.4 \quad b_{1,1}: -5.9$$

$$w_{1,2}: -0.6 \quad b_{1,2}: 5.2$$

$$w_{2,1}: -1.0 \quad b_{2,1}: 5.1$$

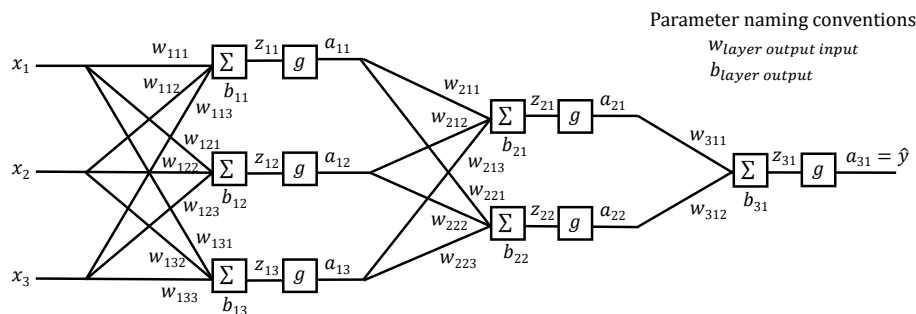
$$w_{2,2}: -1.0$$

Here's a summary table comparing the model and optimization of linear regression with a quadratic feature to our three-neuron network. Be sure to note both the similarities and the differences.

Linear Regression w/ quadratic feature	Three-neuron Network
	
$\hat{y} = h_{\theta}(\mathbf{x}) = b + \sum_j w_j x_j$	$\hat{y} = h_{\theta}(\mathbf{x}) = b_2 + \sum_j w_{2,j} a_{1,j}$ $a_{1,j} = g_{ReLU}(b_{1,j} + w_{1,j} x_1)$
w_1, w_2, b (3 params)	$w_{1,1}, b_{1,1}, b_{1,2}, w_{2,1}, w_{2,2}, b_{2,1}$ (7 params)
<p>Squared error</p> $\ell(y, \hat{y}) = (y - \hat{y})^2$	
<p>Using objective for just N=1 point</p> $J(\theta) = \ell(y^{(i)}, \hat{y}^{(i)})$	
$\frac{\partial J}{\partial w_j} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j}$ $\frac{\partial J}{\partial b} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b}$	$\frac{\partial J}{\partial w_{2,j}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{2,j}}$ $\frac{\partial J}{\partial b_{2,1}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_{2,1}}$ $\frac{\partial J}{\partial w_{1,j}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{1,j}} \frac{\partial a_{1,j}}{\partial z_{1,j}} \frac{\partial z_{1,j}}{\partial w_{1,j}}$ $\frac{\partial J}{\partial b_{1,j}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_{1,j}} \frac{\partial a_{1,j}}{\partial z_{1,j}} \frac{\partial z_{1,j}}{\partial b_{1,j}}$
$w_j \leftarrow w_j - \alpha \frac{\partial J}{\partial w_j}$ $b \leftarrow b - \alpha \frac{\partial J}{\partial b}$	$w_{1,j} \leftarrow w_{1,j} - \alpha \frac{\partial J}{\partial w_{1,j}}$ $b_{1,j} \leftarrow b_{1,j} - \alpha \frac{\partial J}{\partial b_{1,j}}$ $w_{2,j} \leftarrow w_{2,j} - \alpha \frac{\partial J}{\partial w_{2,j}}$ $b_{2,j} \leftarrow b_{2,j} - \alpha \frac{\partial J}{\partial b_{2,j}}$

1.5 Adding More Neurons

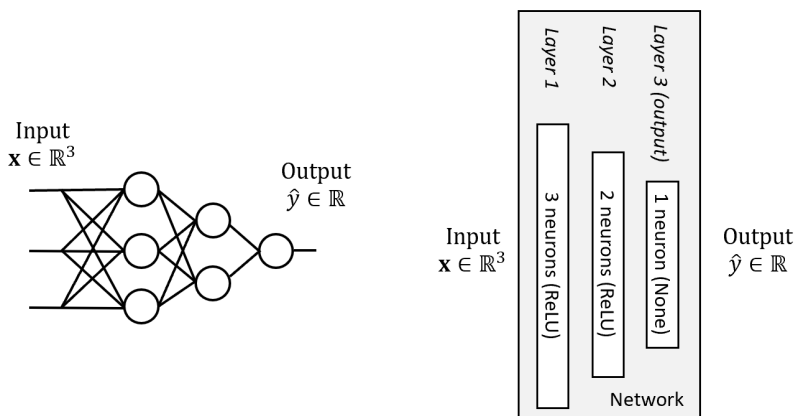
Once we know how to build a three-neuron network, we can start building more and more complex networks that can handle more inputs and outputs and model more complex functions than we could with just three neurons.



1.5.1 Fully-connected networks

This network above is an example of a **fully connected** network. There are potentially lots of ways to connect neurons to each other. A fully connected network organized the neurons into L layers, $L=3$ layers in our example. Each neuron in each layer is “fully connected” to all of the neurons in the previous layer. (For the first layer of neurons, each neuron is “fully connected” to the input data.) This means that if there are M neurons in the previous layer, each neuron would have M input values leading into its linear function.

Here are two other common conventions for displaying fully connected neural network diagrams. Once we understand all of the details in the more complex figure above, we’ll be able to switch to the much simpler figures below.



1.5.2 Vocab

We often refer to the last layer as the **output layer** and any previous layers as **hidden layers**. You’ll also hear references to the input data as the “input layer”; just be careful with that terminology so you don’t get confused between this data layer and a layer of neurons.

Fully connected layers are also referred to as **linear layers** and **dense layers**. A fully connected network is sometimes called an **multi-layer perceptron** (MLP); though this can be a bit confusing because a **perceptron** is a single-neuron network with a hard threshold activation function, while any modern fully connected networks will use more practical activation functions, rather than the historic threshold activation function.

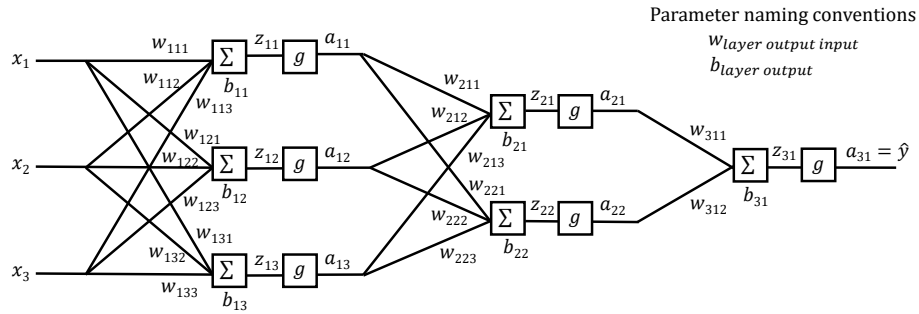
1.5.3 Counting parameters

Just like linear and logistic regression, a neuron with M inputs will have exactly one bias parameter, b , and M weight parameters, w_j , one for each of its M inputs.

For a **fully connected layer** of K neurons with a previous layer of M values, each neuron of the K neurons has M inputs and thus it will have M weight parameters plus one bias parameter. This layer then has a total of $K \cdot M$ weight parameters and K bias parameters (one for each neuron), for a total of $KM + M = K(M + 1)$ parameters.

Example: For our fully connected network with three input values, three neurons in the first layer, two neurons in the second layer, and one output neuron, here is a breakdown of the number of neurons:

$$\begin{aligned} \text{Layer 1 (3 inputs for 3 neurons)} &: 3 \cdot 3 \text{ weights} + 3 \text{ bias} = 12 \\ \text{Layer 2 (3 inputs for 2 neurons)} &: 2 \cdot 3 \text{ weights} + 2 \text{ bias} = 8 \\ \text{Layer 3 (2 inputs for 1 neuron)} &: 1 \cdot 2 \text{ weights} + 1 \text{ bias} = 3 \\ \text{Total} &: 21 \end{aligned}$$



Here is the model and optimization summary for our three-layer network.

The model and optimization math effectively stay the same, there is just more of it. Specifically, the composite functions get deeper and deeper, leading to longer and longer derivative chain rules.

There is one exception: As we start to have cycles in our network diagram, we'll need the multi-variate chain rule for calculus rather than the single-variable version that you might be familiar with from calculus class.

	Fully connected neural network with three layers of neurons
Network diagram	<p>Parameter naming conventions $w_{\text{layer output input}}$ $b_{\text{layer output}}$</p>
Model function	$\hat{y} = h_{\theta}(\mathbf{x}) = g \left(b_{3,1} + \sum_k w_{3,1,k} g \left(b_{2,k} + \sum_i w_{2,k,i} g \left(b_{1,i} + \sum_j w_{1,i,j} x_j \right) \right) \right)$ $\hat{y} = h_{\theta}(\mathbf{x}) = a_{3,1}$ $a_{3,1} = g(z_{3,1}) \quad z_{3,1} = b_{3,1} + \sum_j w_{3,1,j} a_{2,j}$ $a_{2,i} = g(z_{2,i}) \quad z_{2,i} = b_{2,i} + \sum_j w_{2,i,j} a_{1,j}$ $a_{1,i} = g(z_{1,i}) \quad z_{1,i} = b_{1,i} + \sum_j w_{1,i,j} x_j$
Parameters	$w_{1,i,j}$ (9) $b_{1,i}$ (3) $w_{2,i,j}$ (6) $b_{2,i}$ (2) $w_{3,1,j}$ (2) $b_{3,1}$ (1) (23 parameters)
Loss function	Often squared error or cross-entropy $\ell(y, \hat{y})$
Objective	Using objective for just N=1 point $J(\theta) = \ell(y^{(i)}, \hat{y}^{(i)})$
Derivatives	<p>For now, just the bias term derivatives (less crazy indexing notation)</p> $\frac{\partial J}{\partial b_{3,1}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial a_{3,1}}{\partial z_{3,1}} \frac{\partial z_{3,1}}{\partial b_{3,1}}$ $\frac{\partial J}{\partial b_{2,i}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial a_{3,1}}{\partial z_{3,1}} \frac{\partial z_{3,1}}{\partial a_{2,i}} \frac{\partial a_{2,i}}{\partial z_{2,i}} \frac{\partial z_{2,i}}{\partial b_{2,i}}$ $\frac{\partial J}{\partial b_{1,i}} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial a_{3,1}}{\partial z_{3,1}} \frac{\partial z_{3,1}}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial a_{1,i}} \frac{\partial a_{1,i}}{\partial z_{1,i}} \frac{\partial z_{1,i}}{\partial b_{1,i}}$
SGD update	$b_{\text{layer},i} \leftarrow b_{\text{layer},i} - \alpha \frac{\partial J}{\partial b_{\text{layer},i}}$ $w_{\text{layer},i,j} \leftarrow w_{\text{layer},i,j} - \alpha \frac{\partial J}{\partial w_{\text{layer},i,j}}$

2 Calculus Background

2.1 Chain Rule

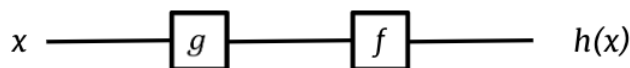
From single-variable calculus, recall the original definition of the chain rule. Give $f : \mathbb{R} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$. Consider the computation $f(g(x))$. The chain rule states that

$$\frac{d}{dx}(f(g(x))) = f'(g(x))g'(x)$$

We can additionally write this using derivative notation as

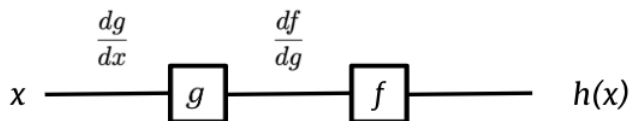
$$\frac{d}{dx}(f(g(x))) = \frac{df}{dg} \frac{dg}{dx}$$

We can note that these two representations are the same thing where $\frac{df}{dg} = f'(g(x))$ and $\frac{dg}{dx} = g'(x)$. While we have these two notations of the chain rule, we can also think about the chain rule in terms of the computation graph. For example, the computation graph for $h(x) = f(g(x))$ is notated as



As in the neural network graphs, we read this computation graph from left to right, and we can see that, given our input x , we apply our function g to compute $g(x)$. Then, we apply f on the result to get $h(x) = f(g(x))$.

Now, we can try to think about what it looks like to take the derivative notated through this computation graph. In our graph, whenever we have an input being passed into a function, we can think of the derivative of that computation occurring on that edge of the graph as well. Consider the graph annotated with derivatives below,



Here, we can note that we have the derivative of the computations on our graphs. For example, performing the operation $g(x)$ (the left-most edge on the graph), we can note that the operation has a derivative of $\frac{dg}{dx}$. A similar operation is done when we apply f onto $g(x)$, giving us the derivative $\frac{df}{dg}$.

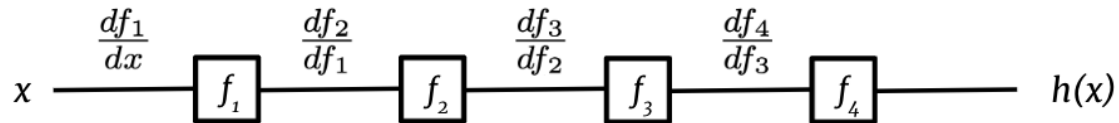
Now, we can note that if we wanted to compute the derivative of $\frac{df}{dx}$, using our understanding of the chain rule, **we are multiplying the derivatives along the edges together**, giving us that $\frac{df}{dx} = \frac{df}{dg} \frac{dg}{dx}$.

This is an important notion to understand as the process of "multiplying the derivatives along edges" will be a crucial building block when computing the derivatives for more complicated functions.

Before moving onto more complicated functions. We can see that this chain rule generalizes to any number of compositions. If we had a composition of four functions: $f_1, f_2, f_3, f_4 : \mathbb{R} \rightarrow \mathbb{R}$, then we could express the computation graph of $h(x) = f_4(f_3(f_2(f_1(x))))$ as follows



By writing the appropriate derivatives on this graph, we can see we are getting the individual terms of our chain rule.



Now, if we consider what the derivative of $\frac{df_4}{dx}$ is, when we are performing repeated applications the chain rule, we know that $\frac{df_4}{dx} = \frac{df_4}{df_3} \frac{df_3}{df_2} \frac{df_2}{df_1} \frac{df_1}{dx}$. However, if we also multiply the derivative along the edges of the graph together, we get the same result! Hopefully, with these two examples, we can get a good idea of how we can think of derivatives and the chain rule over computation graphs (as they are essentially the same thing).

Order of things to cover

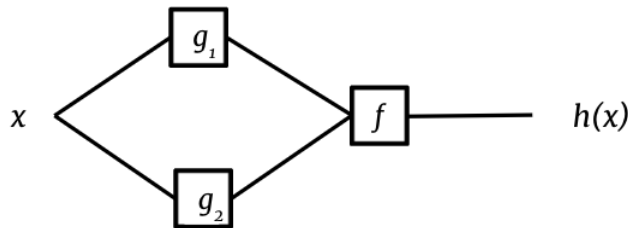
- Recap: single dimensional chain rule (do for composition of 2 functions and 3 functions to express general form, express in terms of computation graph), show both forward and backward passes
- Adding complexity: what happens if you have a function that takes in two inputs, scalar form, write graph, express chain rule for this, explain what happens with multiple outgoing edges, (explain intuition for the meaning)
- Vector form: notes that you can now re-express the inputs as a vector \mathbf{v} , and note that you get the same resulting derivatives with the multi-variate chain rule
- Multiple inputs: our output is now a vector derivative. We apply the same process for each element of the input to produce the vector derivative. Chain rule applies in the same way
- Multiple outputs: our output is now a vector, so that means we are now going to have a matrix derivative of our result, express the resulting size of that computation and how computing the chain rule will still result in an appropriately sized matrix

2.2 Multivariate Chain Rule

2.2.1 Adding More Functions

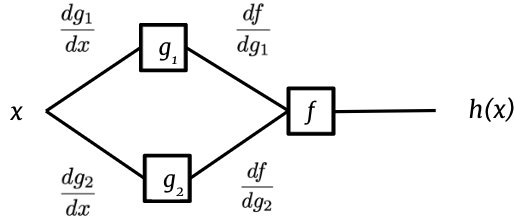
Now, armed with a better understanding of the chain rule in the form of a computation graph, we can think about what happens when we start adding more variables and start using functions of multiple variables.

Suppose that we had now three functions, $g_1, g_2 : \mathbb{R} \rightarrow \mathbb{R}$ and $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, and we define $h(x) = f(g_1(x), g_2(x))$. First, before thinking about derivatives, let's draw what the computation of $h(x)$ looks like as a computation graph.

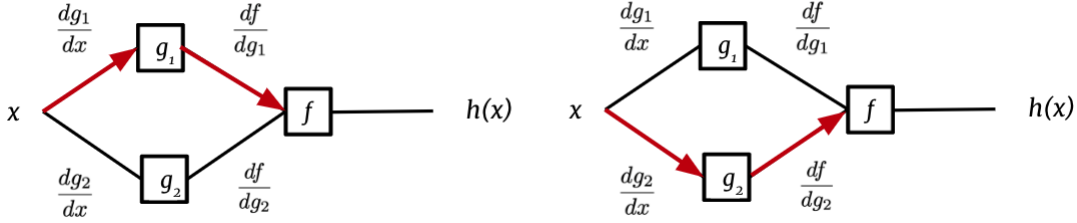


Reading from left to right as usual, we can see that first, x is passed into both g_1 and g_2 to compute $g_1(x)$ and $g_2(x)$, respectively. Note that because x is used twice in two different functions, we have two outgoing edges from x . Next, using those results, we pass them into the function f , which takes in two real-valued inputs, computing $f(g_1(x), g_2(x))$. Note that because f is a function with two real inputs, we have two edges going into the f function block.

Now considering derivatives, note that $h(x)$ is simply a function $h : \mathbb{R} \rightarrow \mathbb{R}$. As a result, we should still have a single-variable derivative. First, let's perform our usual process of annotating our computation graph with derivative values.



After writing the derivatives, we can see that the derivatives for individual edges are normal to compute. However, there is a slight difficulty. Typically, using our normal chain rule, we would multiply the derivatives along **the single** path from x to our result (in this case $h(x)$); however, we now have multiple paths from x to $h(x)$. We have a path going through g_1 and another going through g_2 , visualized below in two copies of the same computation graph.



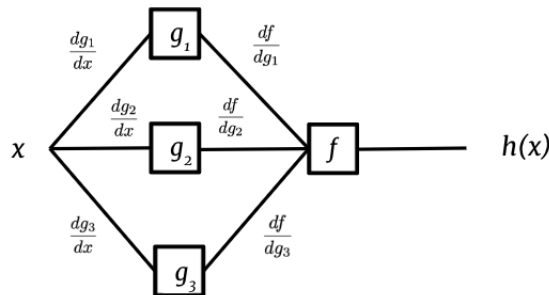
x now influences the resulting output $h(x)$ in multiple ways due to the fact that f takes in two input values and x is used to compute both of f 's inputs. If we consider the derivative of x along a single path, by multiplying the derivatives along the left path, we get that the left-most path has derivative $\frac{df}{dg_1} \frac{dg_1}{dx}$ and the right-most path has derivative $\frac{df}{dg_2} \frac{dg_2}{dx}$.

To reconcile the fact that our derivative contributes to the output in two ways, we **sum the product of derivatives along each path**. This is a generalization of the original single-variable chain rule. As a result, we can now state that

$$\frac{df}{dx} = \frac{df}{dg_1} \frac{dg_1}{dx} + \frac{df}{dg_2} \frac{dg_2}{dx}$$

In this generalized chain rule formulation, for each path, we are taking the product of the derivatives along that path. Then, we are summing the resulting products over all paths (in this case the path in the left image and the path in the right image). The intuition is that if a variable contributes to the derivative in multiple ways (i.e. multiple paths), then we need to sum these contributions to get the actual derivative.

Now, that we have a better understanding of a more general chain rule. We can take a look at this a little more. Suppose that we had $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ and $g_1, g_2, g_3 : \mathbb{R} \rightarrow \mathbb{R}$. Define $h(x) = f(g_1(x), g_2(x), g_3(x))$. Again, we can take a look at the computation graph with its derivatives annotated.



We can see that this computation graph now has 3 paths from x to the output $h(x)$. As a result, the chain rule of this expression is going to look like this formulation

$$\frac{df}{dx} = \frac{df}{dg_1} \frac{dg_1}{dx} + \frac{df}{dg_2} \frac{dg_2}{dx} + \frac{df}{dg_3} \frac{dg_3}{dx}$$

We can note that each term in the summation corresponds to the product of derivatives along a single path along the computation graph where x influences/is a function of $h(x)$.

Now, with this intuition, we can think of a formal **generalization of the multivariate chain rule**. Formally, for some $n \in \mathbb{N}$, if we have $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $g_1, g_2, \dots, g_n : \mathbb{R} \rightarrow \mathbb{R}$, the multivariate chain rule states that

$$\frac{d}{dx}(f(g_1(x), g_2(x), \dots, g_n(x))) = \sum_{i=1}^n \frac{df}{dg_i} \frac{dg_i}{dx}$$

Note that this version of the chain rule is simply a generalization of the single-variate one discussed earlier. The one discussed earlier is a special case of the general rule where $n = 1$.

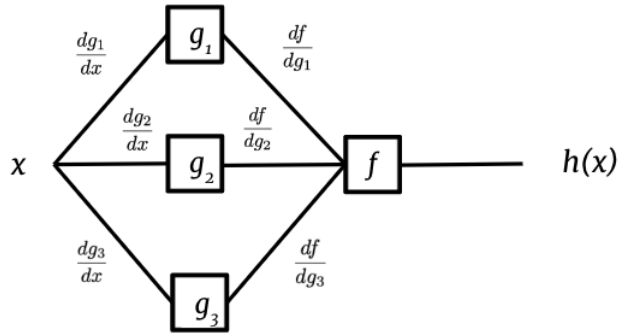
2.2.2 Vector Notation of Chain Rule

In the generalized notion of the chain rule above, for some $n \in \mathbb{N}$, we defined $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $g_1, g_2, \dots, g_n : \mathbb{R} \rightarrow \mathbb{R}$ and considered $f(g_1(x), g_2(x), \dots, g_n(x))$.

However, we can condense these definitions using our understanding of vectors. Define $\mathbf{g} : \mathbb{R} \rightarrow \mathbb{R}^n$ where

$$\mathbf{g}(x) = \begin{bmatrix} g_1(x) \\ g_2(x) \\ \vdots \\ g_n(x) \end{bmatrix}$$

Using this definition, we can note that $f(g_1(x), g_2(x), \dots, g_n(x)) = f(\mathbf{g}(x))$. Now, when considering our computation graph, we can see that we have simply condensed our computation graph with multiple paths with single-variate functions to a single path with a multi-variate function (assume that $n = 3$).



Both of these computation graphs represent the same computation (that means that they must have the same derivative), but the right-most one uses vector notation. Recall that

$$\frac{d}{dx}f(\mathbf{g}(x)) = \frac{d}{dx}(f(g_1(x), g_2(x), g_3(x))) = \frac{df}{dg_1} \frac{dg_1}{dx} + \frac{df}{dg_2} \frac{dg_2}{dx} + \frac{df}{dg_3} \frac{dg_3}{dx}$$

Suppose that we are working in denominator layout. As a result, using our knowledge of vector derivatives, we know that

$$\frac{\partial \mathbf{g}}{\partial x} = \begin{bmatrix} \frac{\partial g_1}{\partial x} & \frac{\partial g_2}{\partial x} & \frac{\partial g_3}{\partial x} \end{bmatrix} \quad \frac{\partial f}{\partial \mathbf{g}} = \begin{bmatrix} \frac{\partial f}{\partial g_1} \\ \frac{\partial f}{\partial g_2} \\ \frac{\partial f}{\partial g_3} \end{bmatrix}$$

Now, we can make a very crucial connection between the two representations of the computation graphs. Consider the right-most graph that is annotated with vector derivatives. Using our notion of the chain rule, by multiplying the derivatives of the edges together, we get that

$$\frac{d}{dx} f(\mathbf{g}(x)) = \frac{\partial \mathbf{g}}{\partial x} \frac{\partial f}{\partial \mathbf{g}}$$

However, this is simply a dot product of two vectors! As a result, we can re-write this equation as

$$\frac{d}{dx} f(\mathbf{g}(x)) = \frac{\partial \mathbf{g}}{\partial x} \frac{\partial f}{\partial \mathbf{g}} = \frac{df}{dg_1} \frac{dg_1}{dx} + \frac{df}{dg_2} \frac{dg_2}{dx} + \frac{df}{dg_3} \frac{dg_3}{dx} = \frac{d}{dx} (f(g_1(x), g_2(x), g_3(x)))$$

As a result, we have just proved that applying the chain rule on the vector representation of the equation produces the same derivative as the original representation. This shouldn't be surprising because we know that the computation graphs represent the same computation. However, **this is powerful because we can represent the chain rule in terms of vectors!**

We now write the **vector form of generalized chain rule**. Given $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and $\mathbf{g} : \mathbb{R} \rightarrow \mathbb{R}^n$, the vector form of the chain rule states that

$$\frac{\partial}{\partial x} (f(\mathbf{g}(x))) = \frac{\partial \mathbf{g}}{\partial x} \frac{\partial f}{\partial \mathbf{g}}$$

Convince yourself that this representation is equivalent to our original definition of the generalized chain rule. Additionally, now that we are working with vectors, **the order of vector/matrix multiplication matters**. Make sure to know which form (numerator or denominator) you are working in and that the shapes of the multiplications and additions are appropriate.

2.2.3 Chain Rule with Multiple Inputs

While we have covered all of the basic principles of the chain rule (multiplying the derivatives of composed functions and summing multiple contributions of a variable to a single output), we can still generalize the chain rule a little further to consider what happens when we are working with vector inputs and vector outputs (however, these are simply repeated applications of the generalized chain rule which we will show).