Predicting the Blood-Brain Barrier Permeability of Chemical Compounds

A Combined Neural Network and Genetic Algorithm Approach

Lodewijk Brand Colorado College
lodewijk.brand@coloradocollege.edu

Abstract

Certain drugs, like those that treat African sleeping-sickness and other brain related diseases, need to be able to pass through the blood-brain barrier to treat the patient. A medicinal synthetic chemist cannot simply look at the structure of a molecule to determine whether or not it will pass through the blood-brain barrier. Instead, the chemist relies on expensive programming packages, or animal testing, to determine whether or not a drug will get into the brain. This project focuses on creating a web-based application that will reliably predict whether or not a drug will cross the blood-brain barrier. The foundation for this application is an artificial neural network. In addition to neural networks we hypothesized that genetic algorithms would help improve the speed at which artificial neural networks learn. The methods used to create the hybrid genetic-neural network behind this application employ basic theoretical concepts from neuroscience, mathematics, computer science, and biology.

I. Introduction

The Problem

Our brain contains a membrane called the blood-brain barrier. This membrane acts as a defensive mechanism that protects our brain from harmful neurotoxins. Although we have this line of defense there are some diseases (eg. African sleeping sickness, etc.) that are able to penetrate the brain. The challenge then, for a drug developer, is to create a drug that is able to penetrate the blood-brain barrier and treat the disease. The drug design process is extensive. The organic chemist must first find a chemical configuration, usually from nature, that has properties that fight the disease they are trying to treat. Then, the chemist devises an assortment of analogues, usually around a thousand, that contain the effective chemical configuration for treating a particular disease. Although, not all of these analogues will have properties that will allow them to pass through the blood-brain barrier. There are two ways around this issue. Either the chemist can "randomly" choose a compound, synthesize it, and send it in for testing, or the chemist can use an expensive programming package to predict whether or not it is able to get into the brain, and using this information, make an educated decision. The first option is time consuming, expensive and often, unethical. Developing a synthetic pathway to make the drug can take months, or longer. After the drug has been synthesized the only way to know whether or not it can pass through the blood-brain barrier is to test it on animals and analyze their brain tissue. This type of testing has serious ethical ramifications. Additionally, if the results come back negative then all the time and energy spent on making the chosen analogue is wasted. The second option, a computer programming package, is extremely expensive and often not cost-effective, especially for low-budget research institutions. Fortunately, there is a way around these expensive programming packages: neural networks.

Project Goals

This research project is designed to create an application that will predict the blood-brain barrier permeability of any chemical compound. The completed application will serve as a tool for medicinal chemists that are trying to synthesize drugs to treat brain related diseases. The code used in this project must be general and robust so that it can be applied to other problems that require neural networks and/or genetic algorithms.

What is a Neural Network?

A neural network is a data-structure that acts like the brain. We can visualize a neural network using a few simple concepts from biology. Our brain, at the most basic level, is made up of neurons and synapses. The first layer of neurons, when they get some environmental stimulus (ex. touch), can fire. If a neuron in the brain fires it sends an electrical signal to other neurons downstream via synapses. These downstream neurons in the brain will then determine how to respond to the environmental stimuli. The main focus of a neural network is to learn the relationship between a set of input and output parameters. We can visualize this process a little easier by thinking of input parameters as "questions" and output parameters as "answers." In order to program an artificial neural network three ideas must be addressed: structure, training, and testing.

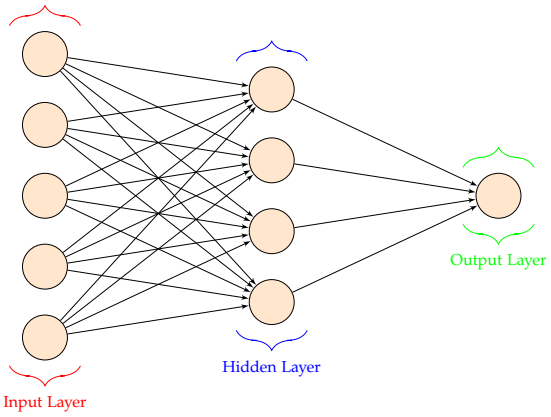


Figure 1: *Neural Network Structure*

The most common type of neural network has an input layer, a number of hidden layers (in this project only one hidden layer will be used), and an output layer **(Figure 1)**. Each layer contains a number of nodes that can hold any value between zero and one. Each node, except for those in the output layer, is connected to every other node in the following layer by an edge. These edges can hold any floating point value. An important thing to keep in mind when designing a neural network is that the structure is defined by the programmer. Once the structure of a neural network has been determined the training process can begin.

Before we can begin training a neural network we need to find an appropriate dataset. The dataset chosen needs to pair a set of input values with a set of output values. After a dataset has been obtained the dataset is shuffled and broken up into two parts: a training set, and a testing set. The testing set is usually much smaller than the training set. After a training set has been established the neural network's edges will be assigned random values. Each input, from the training set, will be fed into the input layer of the neural network and produce an output. The calculated output is then compared to the expected output, from the training set, and the error is "back-propagated" through the edges in the neural network **(Appendix A)**. This iterative process is performed a set number of times for the entire training set. The number of iterations through the training set needs to be sufficiently large in order to ensure that the neural network has a chance to learn from the data. Each iteration through the training set is defined as one âĂIJbackpropagation cycle.âĂİ Once the neural network has been sufficiently trained it's time to see how well it performs.

After the neural network has been trained the testing set is used to determine the accuracy of the trained neural network. The input/output pairs of the training set have never been "seen" by the neural network. They will have had no effect on how the neural network was trained. Because the neural network has never seen the data in the training set we can

use it to gauge how well the neural network has learned. When a neural network is trained on a dataset it gains the ability to generalize and approximate a high-dimensional, non-linear function. Testing the neural network is quite straightforward. The neural network is given an input, from the testing set, and an output is produced. This calculated output is then compared to the expected output and if they are close to one another, the neural network has performed well. On the other hand, if the calculated output and the expected output are not close to one another, the neural network has not performed well. This process is repeated for every input/output pair in the training set and the performance of the trained neural network is determined.

What is a Genetic Algorithm?

A genetic algorithm is a programming technique based on the "survival of the fittest" concept from biology. A genetic algorithm is used to optimize a set of parameters. The optimization of this set of parameters (called a chromosome) is achieved through two mechanisms: crossover and mutation. These two mechanisms are applied to a population of chromosomes in order to optimize a set of parameters. In order for crossover and mutation to work a fitness function needs to be established. Before we begin to define mutation, crossover and fitness functions we must determine how to encode a chromosome, the basic building block of a population.

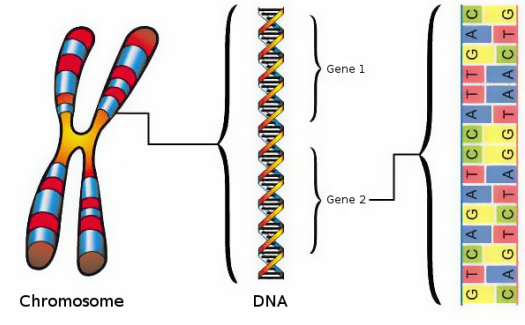


Figure 2: *Chromosome*

The goal of a genetic algorithm is to optimize a set of parameters. This set of parameters is encoded into an object called a chromosome **(Figure 2)**. The chromosome holds all the genetic information of an individual in a population. A genetic algorithm will use the parameters encoded into the chromosome to define an individual in the population. If the chromosome contains all the relevant information for an individual a "survival of the fittest" technique can be used to optimize this set of parameters. After the chromosome of an individual has been defined, the genetic algorithm optimization process can begin.

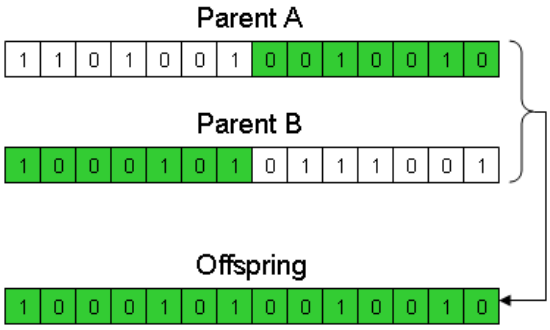


Figure 3: *Crossover*

Crossover and mutation are the two basic functions that serve as the driving forces behind a genetic algorithm. When two parent chromosomes "crossover" they create two new child chromosomes that contain attributes similar to their parents **(Figure 3)**. Crossover gives the the parent chromosomes an opportunity to pass down their genetic material. The theory behind a genetic algorithm relies on the fact that two fit chromosomes will crossover and produce fit children. When two chromosomes crossover two children are produced; these children chromosomes will then replace their parents in the next population and be available for a new round of crossover. Mutation, another genetic algorithm process, also occurs during the crossover of two chromosomes. Mutation gives any individual in the population a chance to explore the entire solution space defined by our parameters. If mutation didn't occur the genetic algorithm optimization would rely only on the parameters explicitly encoded into the first generation of chromosomes. Mutation, from the computer scientist's perspective, can be defined as a

random bit switch **(Figure 4)**. The two mechanisms, crossover and mutation, will produce the optimal solution only if our parent chromosomes are chosen wisely. One chromosome is chosen over another if it is considered to be more fit.

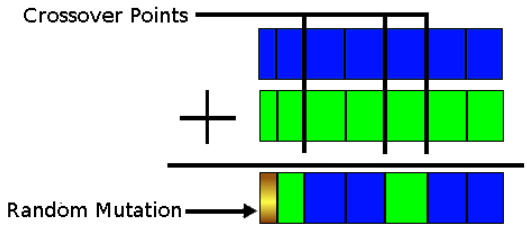


Figure 4: *Mutation*

Before a genetic algorithm can be applied to optimize a particular set of parameters a fitness metric must be established. Without a fitness function it is impossible to rank individuals in the population and crossover will have no optimizing effect. Just like in biology we want to preferentially allow the fittest individuals to reproduce. In order to illustrate how a genetic algorithm optimizes a set of parameters let a surface represents an arbitrary fitness function for bumblebees **(Figure 5)**. Let's suppose that a bumblebee's fitness is proportional to the number of flowers it pollinates. In this example I have oversimplified the bumblebee's genome to only include wing span and thorax weight. The genetic algorithm then ranks the bumblebees and preferentially allows fit individuals to crossover and mutate. As time moves forward the individuals in the population converge on the optimal chromosome configuration and the genetic algorithm has accomplished in finding a global maximum.

Genetic algorithms can be used in all sorts of optimization problems so long as a chromosome can be defined and a fitness metric established. After a chromosome and fitness function have been defined all a genetic algorithm needs is time to run. The ranking, crossover, and mutation of a population is repeated a sufficient number of times in order to optimize the set of parameters defined in the chromosome. Before we move on it is important to realize that a genetic algorithm is built upon a certain level of randomness and is not necessarily guaranteed to find the global maximum. Although a genetic algorithm isn't guaranteed to find the global maximum, it does not suffer from an issue related to the gradient: optimization to a local maximum. Finding the global maximum, instead of a local maximum, makes the genetic algorithm attractive, especially for multidimensional functions with many local maxima.

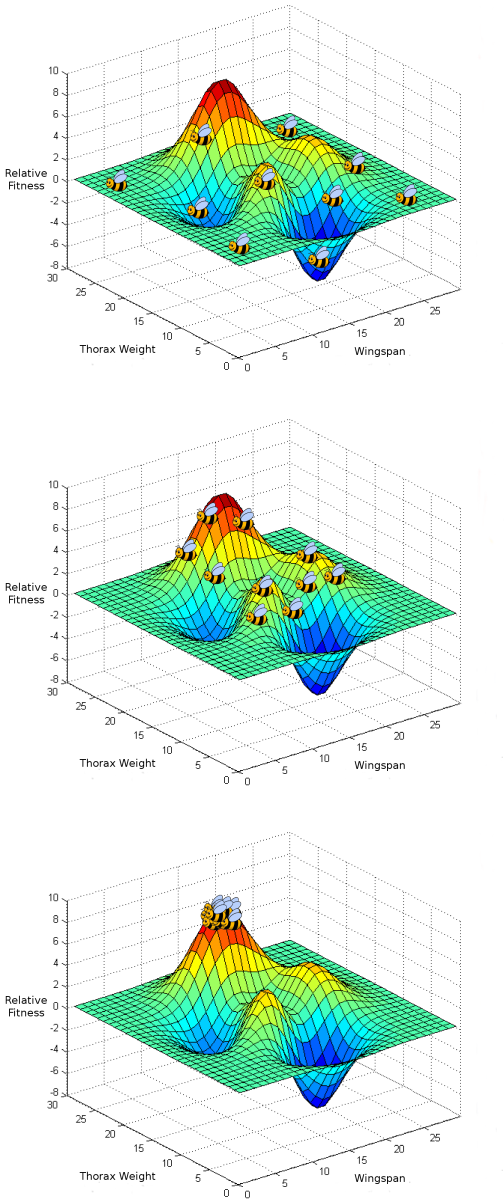


Figure 5: *Genetic Algorithm Example*

II. Methods

Neural Network Implementation

The blood-brain barrier problem this project focuses on can broken down into finding a relationship between inputs and outputs; the relationship between inputs and outputs makes it possible for us to utilize neural networks. Neural networks can be used to predict whether or not a drug penetrates the blood-brain barrier. A basic neural network was coded up in Java and was trained and tested on a set of data that paired basic chemical properties with blood-brain barrier permeability **(Figures 6 & 7)**. The constants that determine the neural network's structure, learning rate, and momentum, were arbitrarily defined and were "baked in" to the code for simplicity. In this application multiple neural networks were trained and were used to predict whether or not a drug could penetrate the blood-brain barrier. In the blind testing process each neural network was given a chance to predict whether or not a particular chemical crossed the blood brain barrier. The "majority vote," of the collection of neural networks, determined the overall prediction.

Database	
Compound	*Crosses?*
Water	Yes
Nicotine	Yes
Chloroquine	No
Amoxicillin	No
Morphine	Yes
Aspirin	Yes
...	...
...	...
...	...
...	...
...	...
...	...
...	...
...	...

Figure 6: *Compound Data [Zhao, 2012]*

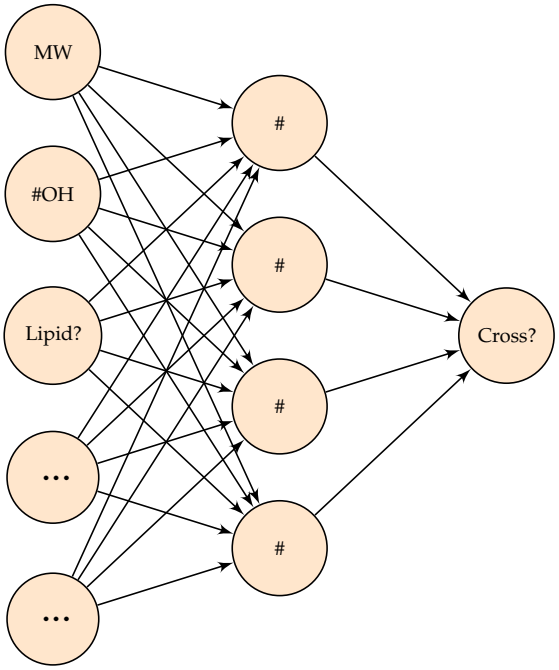


Figure 7: *Structure*

Genetic Algorithm Implementation

In the design of our neural network based predictor we hypothesized that neural networks could be improved upon if they were combined with genetic algorithms. Two researchers at the PSNA College of Engineering & Technology, Vasundara and Padmanaban, have also used a genetic algorithm to help optimize neural networks. Their algorithm encoded the edges of a neural network into a chromosome and optimized them using a genetic algorithm. After the edges were sufficiently optimized, they utilized a backpropagation algorithm to improve the accuracy of the neural network even further. Our approach is a bit different: a chromosome was defined to hold all the "baked in" parameters of a neural network and then the genetic algorithm was used to optimize them. The genetic algorithm optimized three distinct parameters:

- Number of Hidden Nodes
- Learning Rate
- Momentum

In order to optimize these three parameters a fitness function had to be defined. The fitness

of a single neural network in a population was determined by the speed at which it learned. For example, if a particular neural network learned faster (with a fixed number of backpropagation cycles) than another, it would be more likely to crossover and reproduce. The "majority vote" architecture, defined in the basic neural network, was used after the parameters were optimized by the genetic algorithm.

III. Results

The basic and hybrid genetic algorithm neural networks were implemented and were trained and tested on the blood-brain barrier dataset. The performance of each algorithm was determined via a sensitivity and specificity analysis **(Figure 8)**. This type of statistical analysis is a binary classification method and produces four values. The values are as follows:

		Condition		
		Crosses BBB	Doesn't Cross BBB	
Test	Test Outcome Positive	True Positive	False Positive	Positive Predictive Value
	Test Outcome Negative	False Negative	True Negative	Negative Predictive Value
		True Positive Rate	True Negative Rate	

$$\text{PPV} = \frac{TP}{TP+FP} \quad \text{NPV} = \frac{TN}{TN+FN}$$

$$\text{Sensitivity} = \frac{TP}{TP+FN} \quad \text{Specificity} = \frac{TN}{TN+FP}$$

Figure 8: *Specificity and Sensitivity Analysis*

A basic understanding of each of these values is imperative for understanding how each neural network algorithm performed. The positive predictive value indicates how likely it is for a compound to cross the blood-brain barrier given that compound was predicted to cross by the neural network. The negative predictive value indicates the opposite; it indicates how likely it is for a compound not to cross the blood-brain barrier given that the neural network predicted that the compound wouldn't cross. Sensitivity, also known as the true positive rate, takes false positives into account. The sensitivity value gives us the probability that the neural network will predict that a compound crosses the blood-brain barrier given an input compound (from our dataset) that crosses. Specificity, conversely, gives us the probability that the neural network will predict that a compound doesn't cross the blood brain barrier given an input compound that in reality does not cross.

The sensitivity and specificity statistical analyses were performed on both algorithms and were plotted with respect to backpropagation cycles **(Figures 9 & 10)**. At first glance the hybrid approach doesn't offer much improvement. Although, we do observe that both algorithms get better at predicting blood-brain barrier permeability as the number of backpropagation cycles increase. This makes sense because each neural network gets better at predicting blood-brain barrier permeability given a longer training time. The final, and most important, observation is related to specificity. Both algorithms give reasonably high positive predictive, negative predictive, and sensitivity values (after a sufficient number of backpropagations) but fail to produce reliably high specificity values.

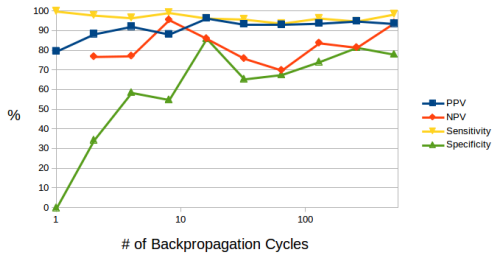


Figure 9: *Basic Neural Network*

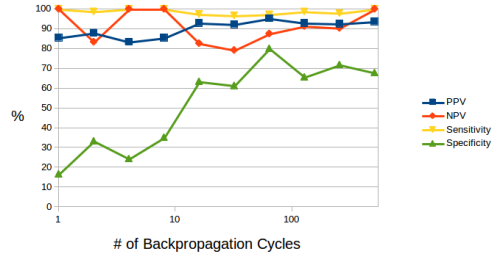


Figure 10: *Hybrid Neural Network*

IV. Discussion

The data from our hybrid approach does not indicate a significant improvement upon the basic neural network. Although, the data does indicate that a neural network works well for this dataset. The high positive predictive, negative predictive, and sensitivity values of the basic approach bode well for the performance of this application. However, the relatively low specificity value does bring up a major concern. A lower specificity value indicates a high number of false positives. This indicates that both our algorithms are prone to falsely predicting that a compound enters the brain. This is a serious problem for any drug development team using this application. If a candidate drug is incorrectly predicted to cross the blood-brain barrier a drug company may try to develop it. A high number of false positives will inevitably makes this application a less attractive tool for drug development. In order to improve the marketability of this application it must cut down on the number of false positives. Improving the specificity of our algorithm is of paramount importance, even at the expense of other values like sensitivity.

A lower sensitivity value indicates a higher number of false negatives. This is much less of a concern from the drug development perspective. Suppose that a candidate drug is incorrectly predicted not to cross the blood-brain barrier, in this case the drug development team misses out on opportunity. Missing out on opportunity is more attractive than the alternative because resources are not wasted on synthesizing the drug. Through the lens of sensitivity and specificity it is obvious that our algorithm requires some major improvements to be marketable.

V. Conclusion

This project was designed to create an application that could predict whether or not a compound entered the brain. After an analysis of our algorithms we are able to make conclusions about our approach. First, a neural network seems to be a viable strategy for finding the input-output relationships in the blood-brain barrier dataset [Zhao, 2012]. Second, with the current configuration, a genetic algorithm does not improve the speed at which a neural network learns. Finally, in order to make this a marketable tool for medicinal chemists, the low specificity value needs to be improved.

Further research should focus on improving the genetic algorithm approach and the specificity of our neural network. In order to improve this application these areas must be explored. Can we develop a new chromosome schema in order to successfully utilize genetic algorithms? We could define the fitness of a neural network as a function of specificity. In other words, we could preferentially choose a neural network with a high specificity value to crossover and reproduce. This strategy would theoretically result in a neural network that predicts with a high level of specificity.

Another approach for improving sensitivity involves modifying our dataset. The dataset our neural network trained itself on contains many more compounds that cross the blood-brain barrier than compounds that do not. This imbalance may result in an unfairly biased neural network. Further research could also focus on creating a balanced dataset that contains an equal number of compounds that cross and do not cross the blood brain barrier.

The neural network framework developed for this application is relevant not only for this project but for any other data driven application. The code for this project is general enough to work with any appropriately defined input/output dataset. Generality, in this case, makes these neural network and genetic algorithm frameworks significantly more useful for a wider range of data-driven applications.

References

[Ileana, 2004] Ileana, Ioan, Corina Rotar, and Arpad Incze (2004) The Optimization of Feed Forward Neural Networks Structure Using Genetic Algorithms *Proceedings of*

the International Conference on Theory and Applications of Mathematics and Informatics

[Pardridge, 2005] Pardridge, William M. (2005) The Blood-Brain Barrier: Bottleneck in Brain Drug Development *NeuroRx* 2(1), 3–14.

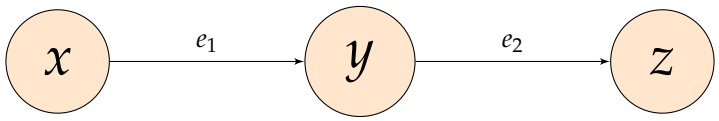
[Vasundara, 2014] Vasundara M., Padmanaban, K. (2014) Optimization of Fixture Layout and Artificial Neural Network (ANN) Weights of ANN-Finite Element Analysis Based Fixture Layout Model Using Genetic Algorithm *Journal of Engineering and Technology* 4(2), 102–109

[Zhao, 2012] Zhao, Yuan H. et al. (2012) Predicting Penetration Across the Blood-Brain Barrier from Simple Descriptors and Fragmentation Schemes *Journal of Chemical Information and Modeling* 47(1), 170–175.

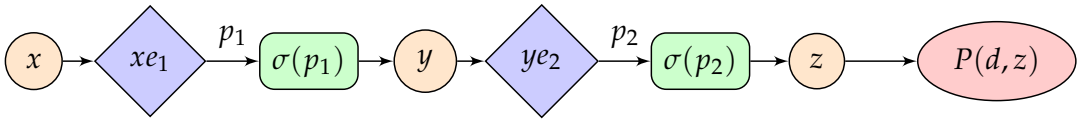
Appendix A: Backpropagation

The backpropagation algorithm is used to train the neural networks in this project. This gradient ascent approach described below varies the edges of the neural network and allows the neural network to generalize and approximate high-dimensional, non-linear functions:

The World's Simplest Neural Network:



A neural network "learns" by varying the values of edges e_1 and e_2. The extent to which e_1 and e_2 change is determined by the "backpropation" of error in the neural network.



Define performance as: $P(d,z) = -\frac{1}{2}(d-z)^2$

Where:
d = Desired Output (from our dataset)
z = Calculated Output (from our neural network)
$\sigma(x)$ = The sigmoid function = $\frac{1}{1+e^{-x}} \longrightarrow \frac{\mathrm{d}\sigma}{\mathrm{d}x} = \frac{e^{-x}}{(1+e^{-x})^2} = [1-\sigma(x)]\sigma(x)$
Goal:
Maximize $P(d,z)$ by varying e_1 and e_2
Method:
Calculate the gradient of $P(d,z)$ with respect to e_1 and e_2
Find: $\frac{\partial P}{\partial e_1}$ and $\frac{\partial P}{\partial e_2}$

Maximize: $P(d,z) = -\frac{1}{2}(d-z)^2$ (Using the Chain Rule!)

$$\frac{\partial P}{\partial e_2} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial e_2} = (d-z)\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial e_2} = (d-z)\frac{\partial z}{\partial p_2}y$$

$$\frac{\partial P}{\partial e_1} = \frac{\partial P}{\partial z}\frac{\partial z}{\partial e_1} = (d-z)\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial e_1} = (d-z)\frac{\partial z}{\partial p_2}\frac{\partial p_2}{\partial y}\frac{\partial y}{\partial e_1} = (d-z)\frac{\partial z}{\partial p_2}e_2\frac{\partial y}{\partial p_1}x$$

How do we deal with $\frac{\partial z}{\partial p_2}$ and $\frac{\partial y}{\partial p_1}$? We know: $\frac{\partial \sigma(p_n)}{\partial p_n} = [1-\sigma(p_n)]\sigma(p_n)$

$$\frac{\partial P}{\partial e_2} = (d-z)\cdot[1-z]z\cdot y$$

$$\frac{\partial P}{\partial e_1} = (d-z)\cdot[1-z]z\cdot e_2\cdot[1-y]y\cdot x$$