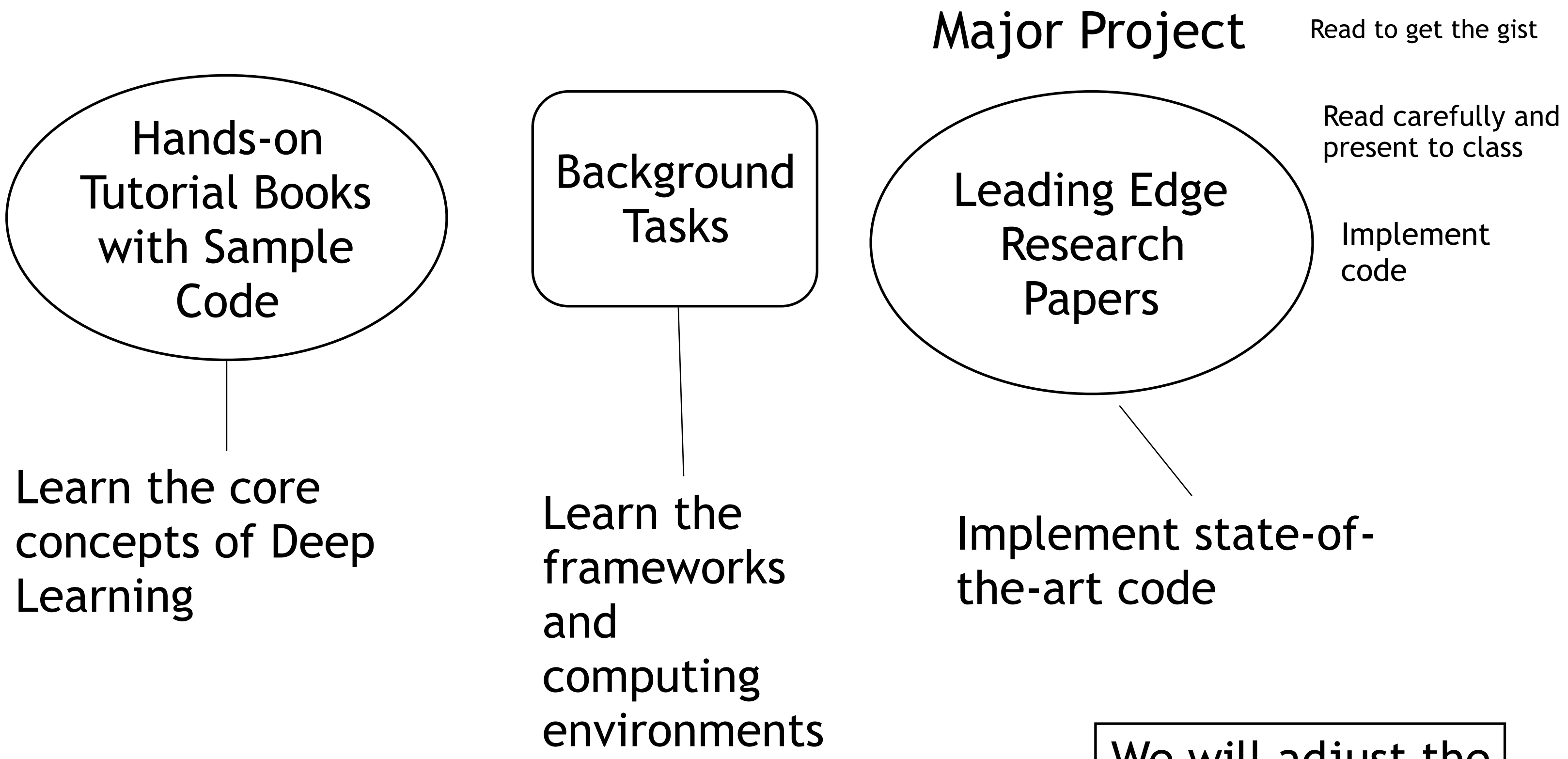
Lecture 1: Deep Neural Networks and Backpropagation Training

Reading and Research in Deep Learning

James K Baker

11-364 Deep Learning R&R


Major Project
Read to get the gist
Read carefully and present to class
Implement code
Hands-on Tutorial Books with Sample Code
Background Tasks
Leading Edge Research Papers
Learn the core concepts of Deep Learning
Learn the frameworks and computing environments
Implement state-of-the-art code

We will adjust the pace of the course as necessary

Topics of Lecture

- Information about course
- Computing with a network of simple units
- Computation of feedforward activation
- Minimizing a function: Gradient descent
- Chain rule
- Cost function
- Backpropagation algorithm
- Minibatches and stochastic gradient descent
- Iterative training

As you will see, this is a lot to cover in one lecture.

Use these slides as a reference. Refer back to it as needed. Don't try to learn it all at once.

Why so much at once?

- This is the background you need to understand the readings
- You will need this to understand the functions in the Python library and deep learning frameworks, which are sometimes used in the tutorials without being fully explained
- You will need all of this background for your major projects, so I am trying to give this background as quickly as possible
- There will be another high-density lecture, then some material that will be more spread out

Can you absorb so much?

- You are not expected to absorb it all from hearing or reading it once.
  - It will be re-enforced when you use the techniques in the tutorial books
  - It will let you fit each technique you use into the overall picture
- There will be a large number of tutorial projects, so you will see each idea many times
- You will see each idea again when a more advanced technique based on it is discussed
- This is the way to get you ready for your major projects

11-364

- This is a reading and research course
  - Based on the principle of “learn by doing” (not lectures)
  - Hands-on
    - For all the readings, you will be implementing code
  - The work will be done in teams (tentatively six teams)
    - Two orthogonal sets of teams
      - Each infrastructure team will have a background task
        - E.g. Learn a tool or framework and support the class
      - Each project team will do a major project, implementing a state-of-the art research paper
        - It will help if each project team has members from multiple infrastructure teams

Phases of Course

- Introductory, Hands-on Books
  - Individual Presentations of Chapters
- Gisting Technical Papers (to aid selection of major projects)
- Presentation of Selected Gists and Project Proposals (formal formation of project teams)
- Interim Project Presentations
- Final Project Presentations
  - Goal: Be able to participate in top competitive testbeds
  - We may even create some testbeds

- Pace of course will be adjusted as needed
  - You should each give feedback each week
  - Tell me about your problems, your successes

Start Here (There will be reading assignments in these 3 books):

- **Hands-On Introductory Books (Assigned reading in course):**

- M Nielsen: Neural Networks and Deep Learning, (free on-line book http://neuralnetworksanddeeplearning.com/)
    - This on-line book describes the actual steps in the algorithms, rather than just calling Python or framework library functions like the other introductory books. I will mainly follow this book in my presentations. There will be no student presentations from this book.

The next two books have many tutorial projects. There will be student presentations of these projects.

- N Lewis(1): Deep Learning Step by Step with Python: A Very Gentle Introduction to Deep Neural Networks for Practical Data Science (Kindle Edition ~~$9.99~~ Now $37.77)
    - Hands-on introduction of deep learning; multiple hidden layers; introduces momentum, dropout and other techniques to improve training. it uses made-up examples with its own data. Includes snippets of Python code. Using Python neural network packages. Extensive footnotes with references to the research literature.
- J Brownlee: Deep Learning with Python ($47 online purchase https://machinelearningmastery.com/deep-learning-with-python/)
    - Hands-on lessons and projects. Based on Python, Theano, TensorFlow, and Keras. Nine projects based on benchmarks such as MNIST and CIFAR-10. Covers convolutional neural networks and LSTM recurrent neural networks. The book allows development on either Windows or Linux, but tools such as Theano are aimed primarily at Linux. The MNIST and CIFAR benchmarks are like the benchmarks used in some of the research papers that will be covered in the course.
    - A brief tutorial by the author of Deep Learning with Python: An introduction to backpropagation, J Brownlee (http://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/)

In the CMU course 11-364: "Introduction to Human-Aided Deep Learning", the students will read these three books and give presentations on chapters from the books in the first 3-4 weeks of the course.

Additional References

- Goodfellow, Bengio, Courville: Deep Learning, http://www.deeplearningbook.org/ (preprint)
  - Definitive reference book; This is a reference book, not a textbook. It has much more material than can be covered in this course.

- Many on-line tutorials

Infrastructure and Tools

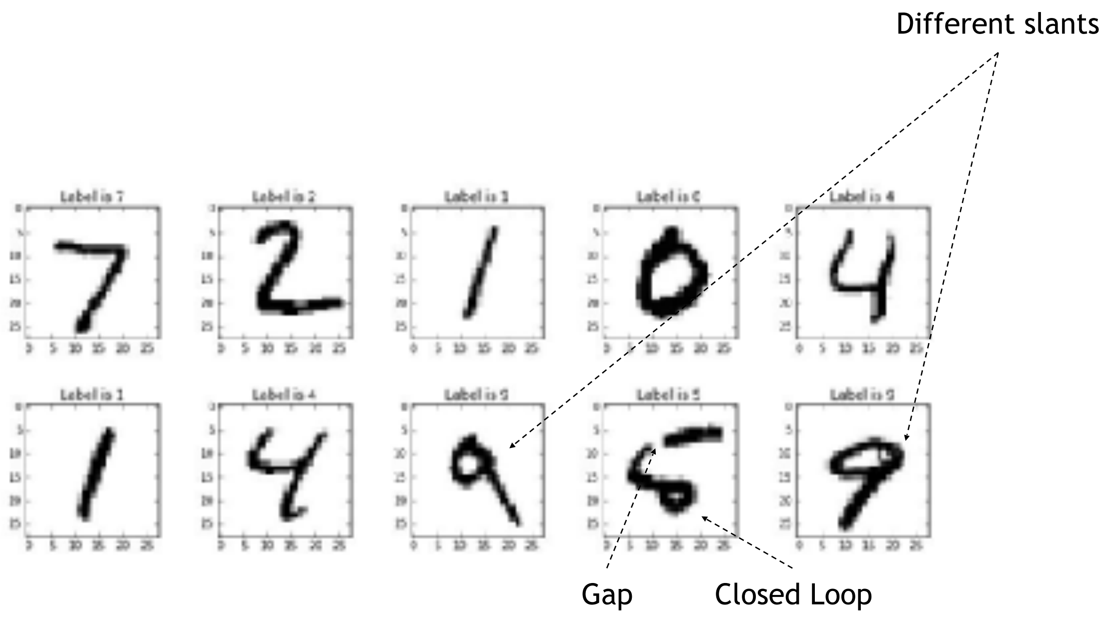
- Infrastructure Teams (not the project teams)
  - 11-364 Course Infrastructure
  - Tool Teams (tools needed for Brownlee and projects)
    - Theano
    - Tensorflow
    - Keras
    - AWS (Amazon Web Services)
    - PSC (Pittsburgh Supercomputer Center)

- During the first 3 - 4 weeks, each student will give a short presentation on a tutorial project from one of the books
- Project Teams will then be organized for the major projects

If You are Buying New Hardware to do Deep Learning

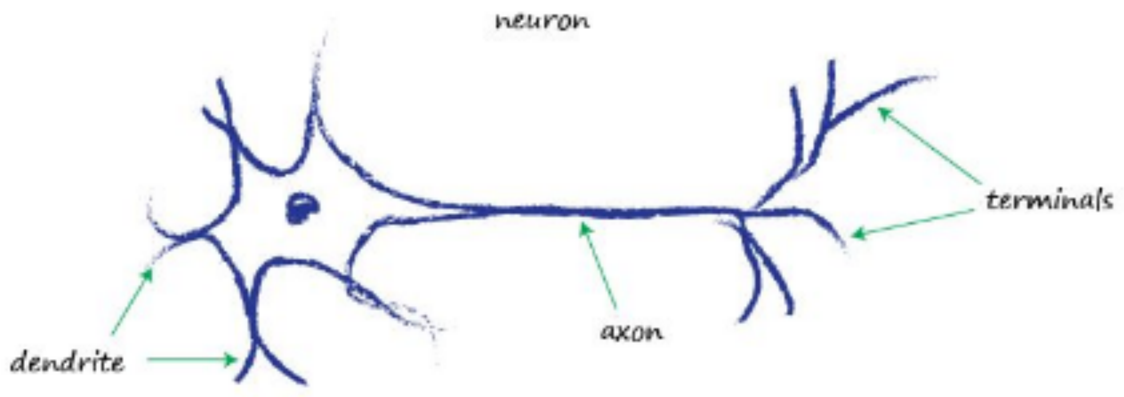
- If you already have a high-end Nvidia GPU, you can use that
- If buying new, the Nvidia GTX 1080 is recommended
  - If on a tight budget, a GTX 1050, 1060, 1070 will do
    - Make sure your computer power supply has enough power
- If buying a new computer, choose tower or laptop
  - Computer (Tower): A computer with enough slots and power for a GTX 1080
  - If buying a new laptop, the mobile versions of the 1060, 1070 and 1080 series are nearly as fast as the desktop versions
  - Get both: Alienware laptop + Graphics Amplifier + GTX 1080
    - A mobile GPU plus an external GPU
- Each project team should try to have at least one member with such a configuration
  - If not, use AWS or PCS

Introduction to Pattern Recognition: Handwritten Digits



This 5 is hard to recognize:

A Neuron



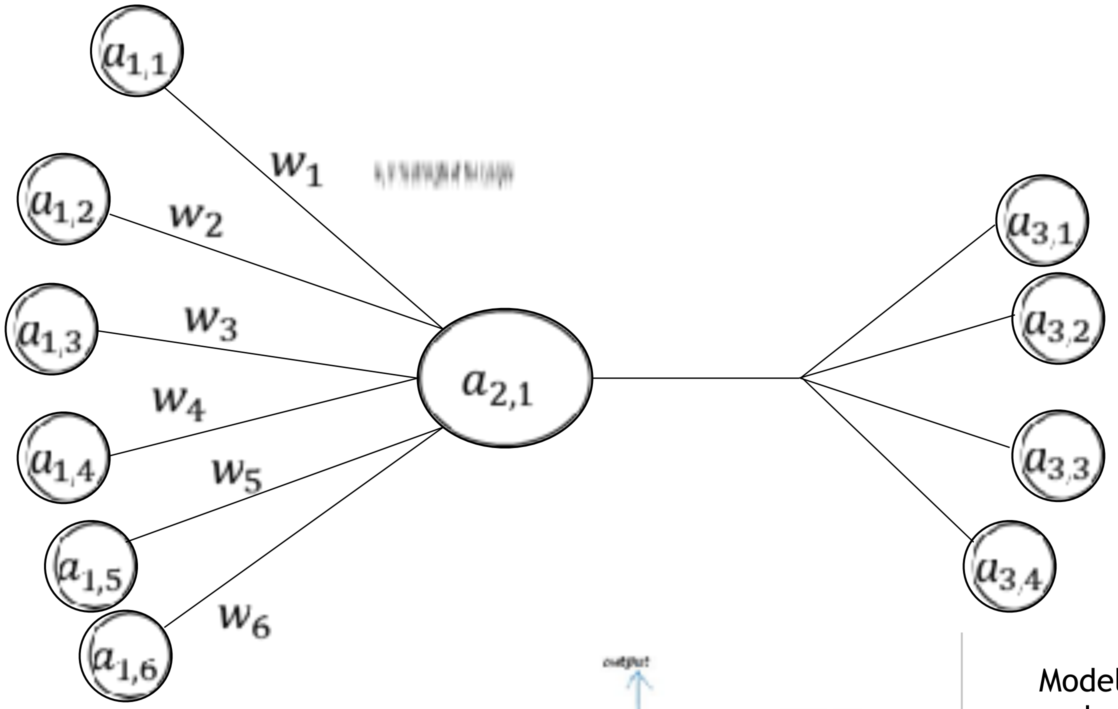
On the dendrites, there are a large number of synapses receiving input (by chemical transfer) from other neurons.

These inputs are converted into electrical potentials, which accumulate in the cell body.

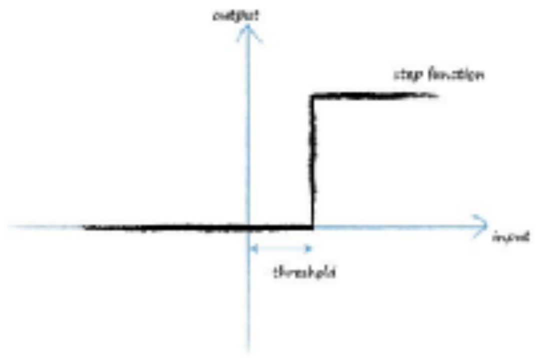
When the total electrical potential exceeds a threshold, the neuron will “fire”, sending a wave of electrical excitation along the axon.

This electrical excitation will stimulate the synapses at this neurons terminals, causing them to release chemicals that cross the synaptic gap to stimulate other neurons.

A Simplified Model of a Neuron



a_i is the activation level
= 0, if the neuron does not fire
= 1, if the neuron does fire



Model: The neuron fires if and only if the weighted sum of its stimulations exceeds a threshold.

Perceptron Model: Threshold Function

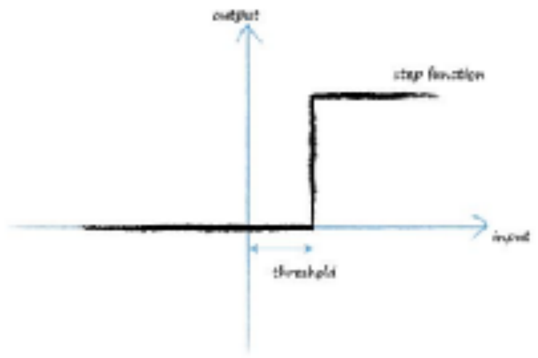
- Model: The neuron fires if and only if the weighted sum of its stimulations exceeds a threshold (called the bias)
- $z_{2,1} = \sum_{i=1\ to\ 6} a_{1,i} w_i + bias_{2,1}$
- $a_{2,1} = 0, if\ z_{2,1} < 0$
- $a_{2,1} = 1, if\ z_{2,1} \geq 0$

z is the total stimulation of the neuron.

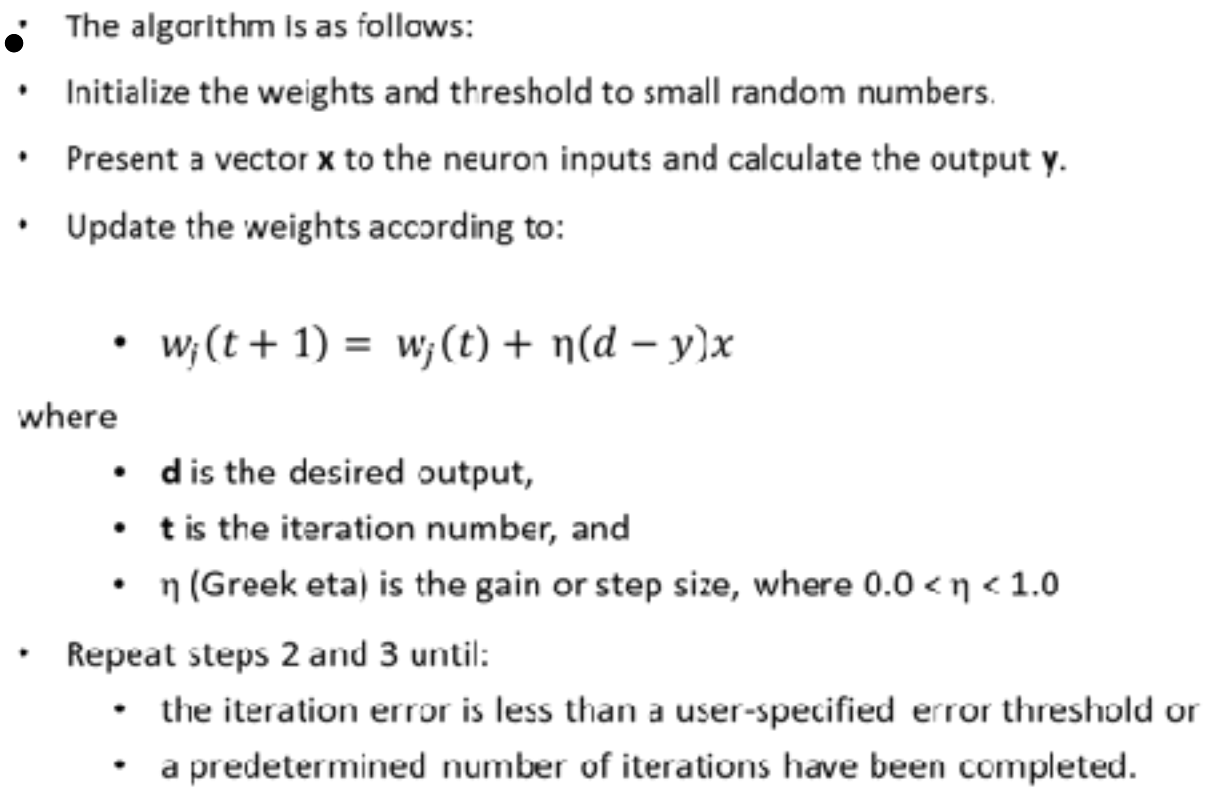
a is the output of the activation function of the neuron

The first subscript is the layer of the neuron.

The second subscript is the number of the neuron.



Perceptron (Single Neuron) Learning Algorithm



- The algorithm is as follows:
- Initialize the weights and threshold to small random numbers.
- Present a vector **x** to the neuron inputs and calculate the output **y**.
- Update the weights according to:
    - $w_j(t+1) = w_j(t) + \eta(d - y)x$

where

- **d** is the desired output,
- **t** is the iteration number, and
- η (Greek eta) is the gain or step size, where $0.0 < \eta < 1.0$

- Repeat steps 2 and 3 until:
    - the iteration error is less than a user-specified error threshold or
    - a predetermined number of iterations have been completed.

Brief History of Pattern Recognition with Artificial Neural Networks

- 1950s Single neurons (Perceptron) Rosenblatt, *Principles of Neurodynamics*
    - Adaptive learning
    - Perceptron learning algorithm (perceptron convergence theorem: if a problem is separable, a perceptron can learn it)
- 1960s Single layer of neurons

  Minsky, Papert, *Perceptrons: An Introduction to Computational Geometry*
    - Negative result: some things can never be learned with a single layer, no matter how big (e.g. millions in retina)
    - Multiple layers is a hard integer programming problem
- Gap in progress …
- 1980s Backpropagation to train hidden layer
- Another gap …
- 2006 Deep Learning
    - 2010s GPUs

A Brief Review of (a little bit of) Calculus

- Derivative: For a function f(x), the amount of change in f(x) per amount of change in x is called the derivative of f with respect to x, written as
- $f'(x)$ or $\frac{df(x)}{dx}$ or $\frac{\partial f(x)}{\partial x}$ or f' or $\frac{df}{dx}$ or $\frac{\partial f}{dx}$

- Chain Rule: If h(x) = g(f(x)) then, for any change in x, f(x) changes by $\frac{\partial f}{\partial x}$, and for each change in f, g changes by $\frac{\partial g}{\partial f}$, so the total change in h(x) = g(f(x) is
- $\frac{\partial h}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x}$

Some Formulas for Derivatives

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

$$\frac{\partial fg}{\partial x} = f\frac{\partial g}{\partial x} + g\frac{\partial g}{\partial x}$$

$$\frac{\partial \, ^{f}/_{g}}{\partial x} = \frac{g\frac{\partial f}{\partial x} - f\frac{\partial g}{\partial x}}{g^2}$$

$$\frac{\partial x^2}{\partial x} = 2x$$

$$\frac{\partial e^x}{\partial x} = e^x$$

$$\frac{\partial \ln(x)}{\partial x} = \frac{1}{x}$$

This is all the formula we will need for now. However, you may need to use the chain rule to get other forms of these.

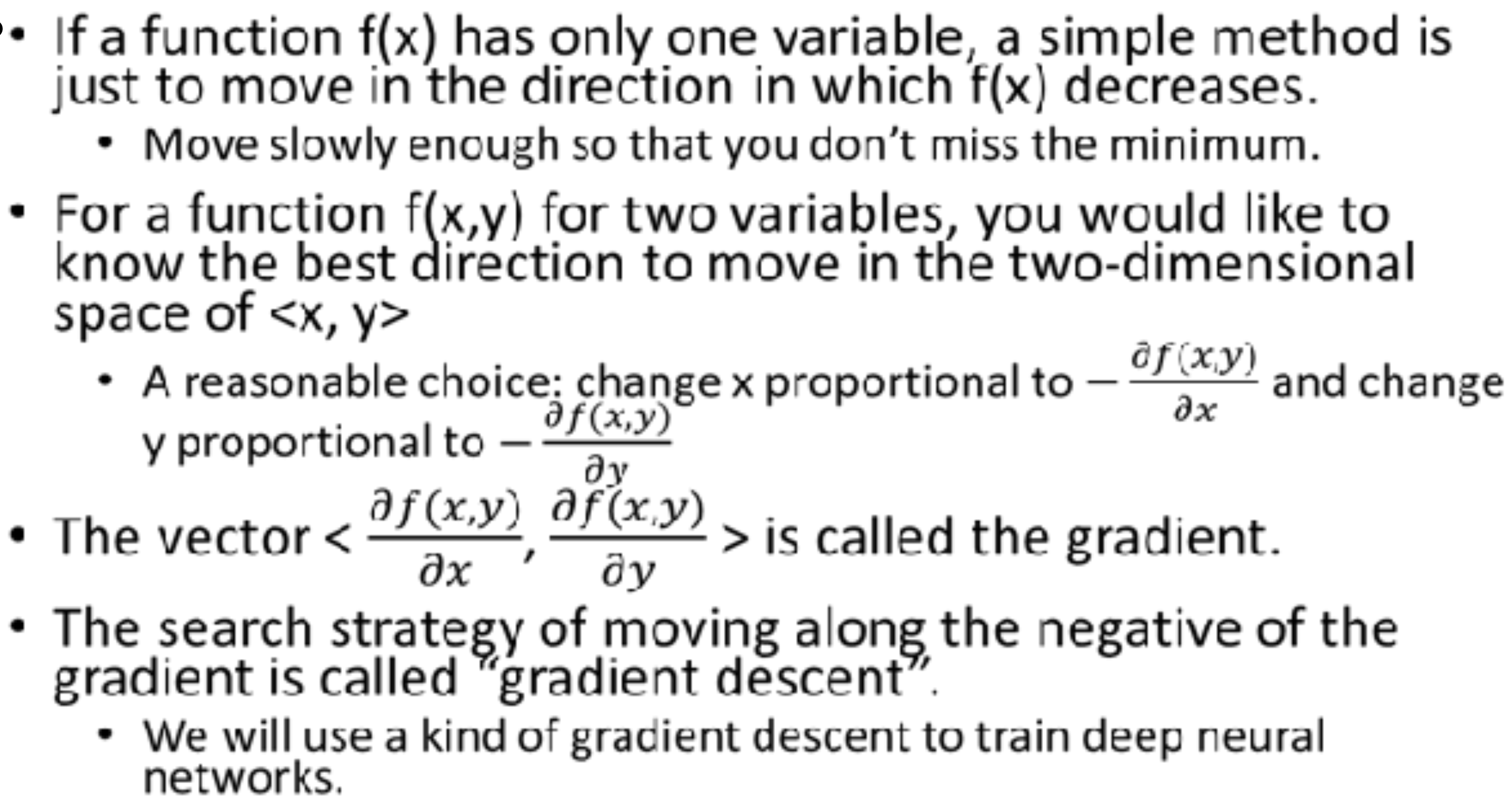
For example,

$$\frac{\partial e^{-x}}{\partial x} = \frac{\partial e^{-x}}{\partial(-x)}\frac{\partial(-x)}{\partial x} = e^{-x}(-1)$$

Exercise: What is the derivative of the sigmoid function

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

How do you find the minimum of a function of multiple variables?

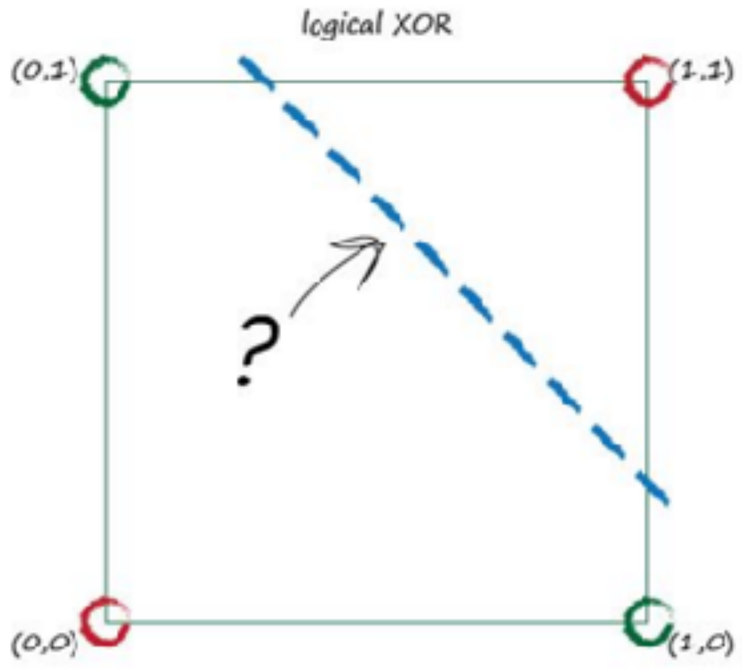


- If a function f(x) has only one variable, a simple method is just to move in the direction in which f(x) decreases.
  - Move slowly enough so that you don't miss the minimum.
- For a function f(x,y) for two variables, you would like to know the best direction to move in the two-dimensional space of <x, y>
  - A reasonable choice: change x proportional to $-\frac{\partial f(x,y)}{\partial x}$ and change y proportional to $-\frac{\partial f(x,y)}{\partial y}$
- The vector $< \frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y} >$ is called the gradient.
- The search strategy of moving along the negative of the gradient is called "gradient descent".
  - We will use a kind of gradient descent to train deep neural networks.

What caused the gap in progress? Two Problems

- (1) Some things can't be learned by one layer of perceptrons (e.g. xor)
- (2) There was no effective general algorithm for training multi-layer perceptrons

Gap in progress from
1960s to 1980s



There is no place to put a straight line to separate the green circles from the red.

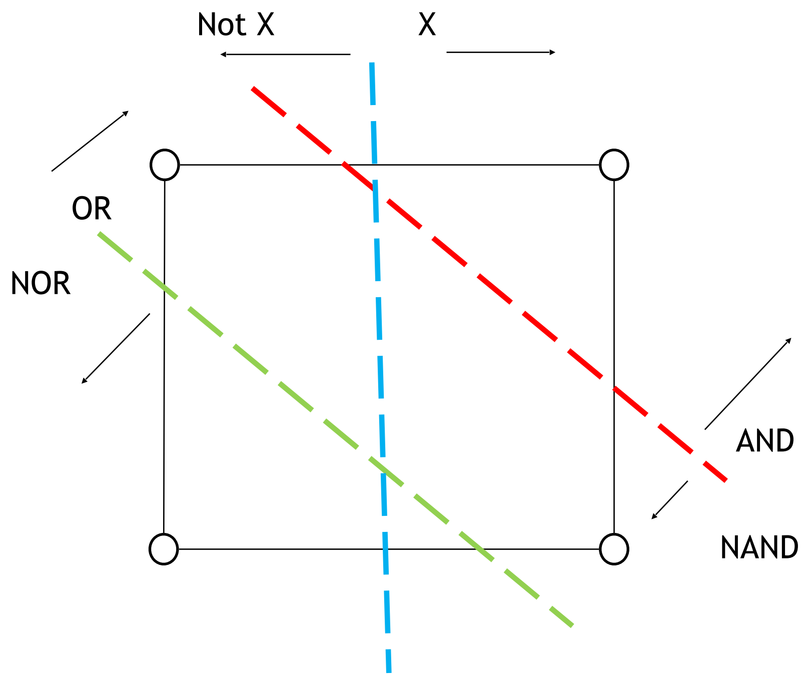
(x = y) is also impossible with a single layer!

Most Elementary Logic Functions Can be Done by a Single Unit

- AND or NAND
- OR or NOR
- Not

But a single neuron (or a whole layer) can't compute (X xor Y) or (X = Y). That requires more layers.

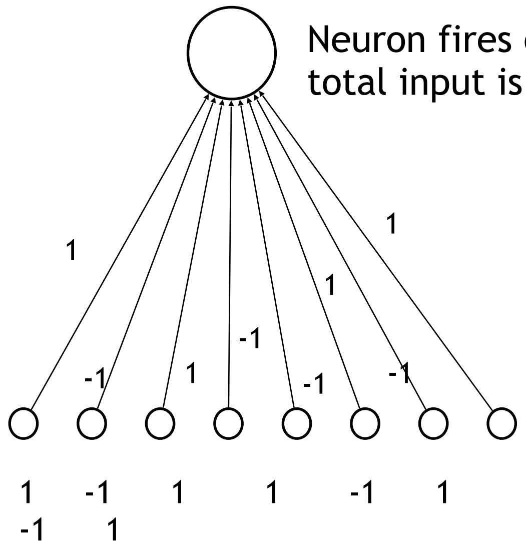
Any logic function can be built just from NAND gates (or just from NOR gates).



A Single Neuron can Match Any Single Boolean Pattern

It just requires selective positive and negative weights and a threshold that represents an n-way AND.

It rejects all other patterns.



This is equivalent to an 8-way AND of the exact pattern

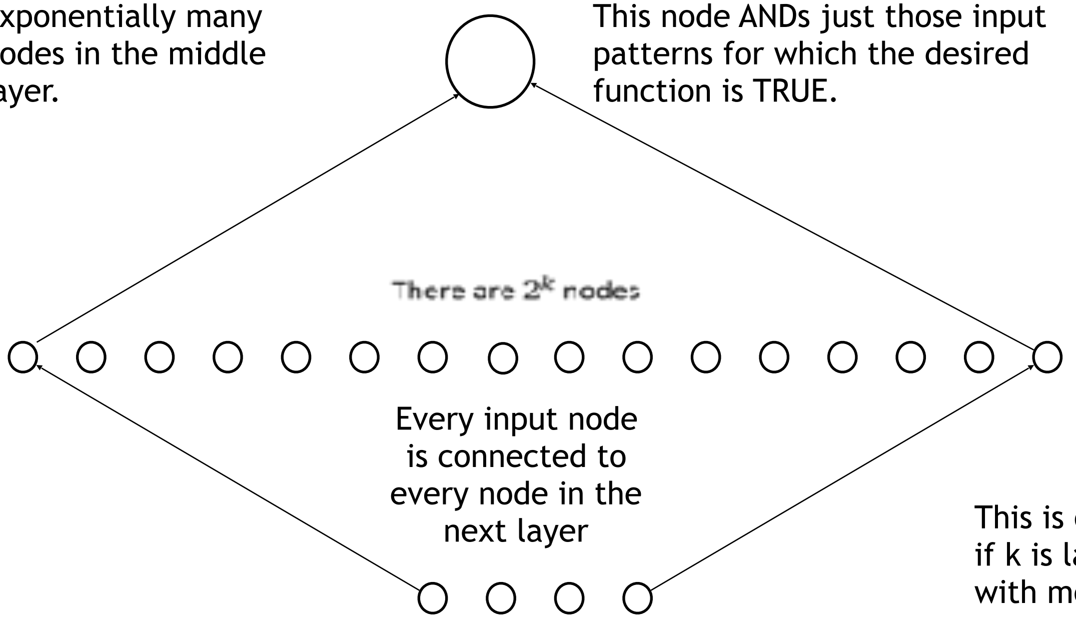
Assume the desired bit pattern is represented by 1s and -1s.

Can also be done with the normal 0, 1 representation of bits, but the arithmetic is less easy to see.

Two Layers (plus the input layer) can Compute any Boolean Function of k Variables

But it may require exponentially many nodes in the middle layer.

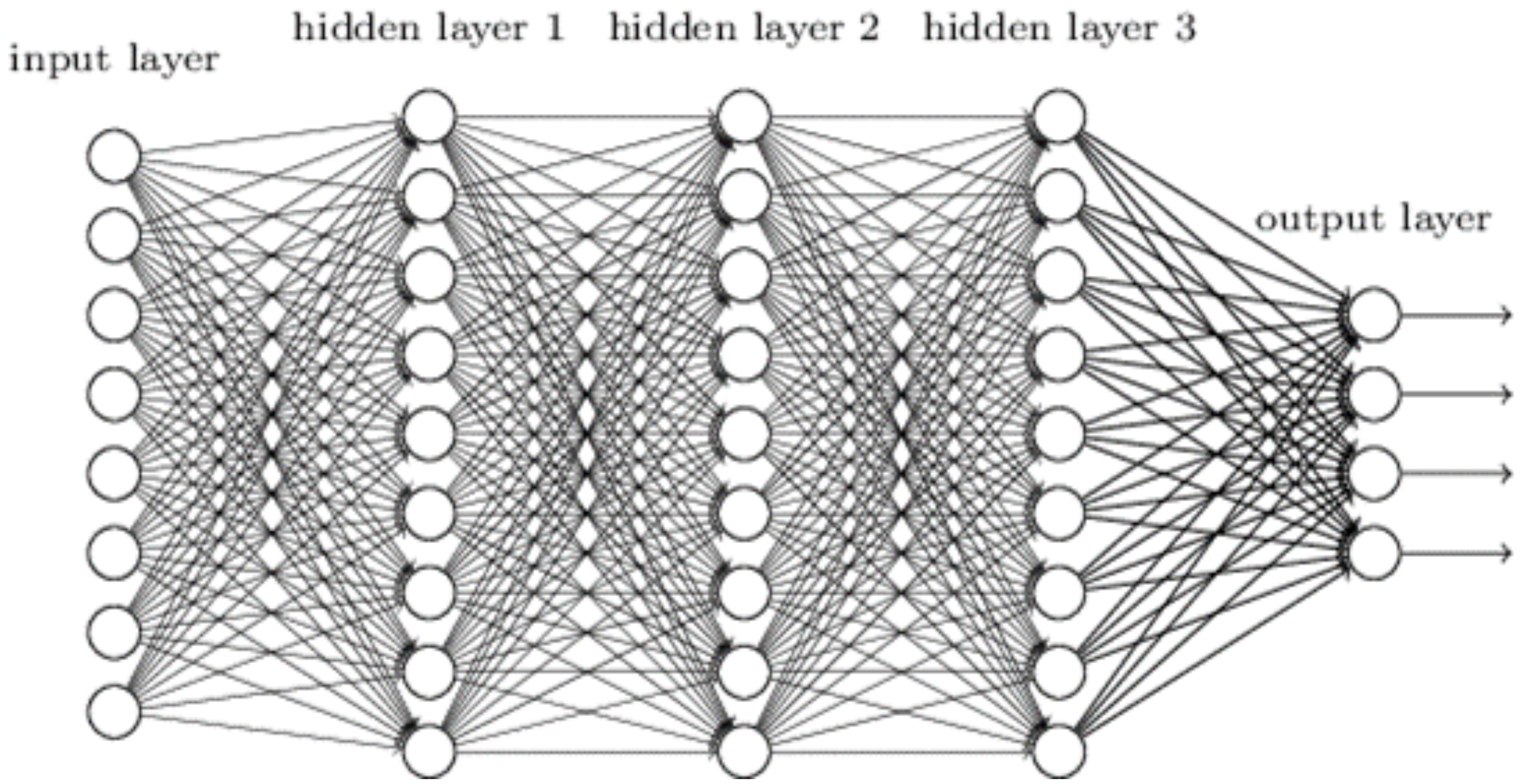
This node ANDs just those input patterns for which the desired function is TRUE.


There are 2^k nodes
Every input node is connected to every node in the next layer

Each node in this layer fires only for one specific input pattern.

This is obviously not efficient if k is large. Can we do better with more layers?

Deep neural network



Each arc has a weight
w,
which is multiplied by
its input.

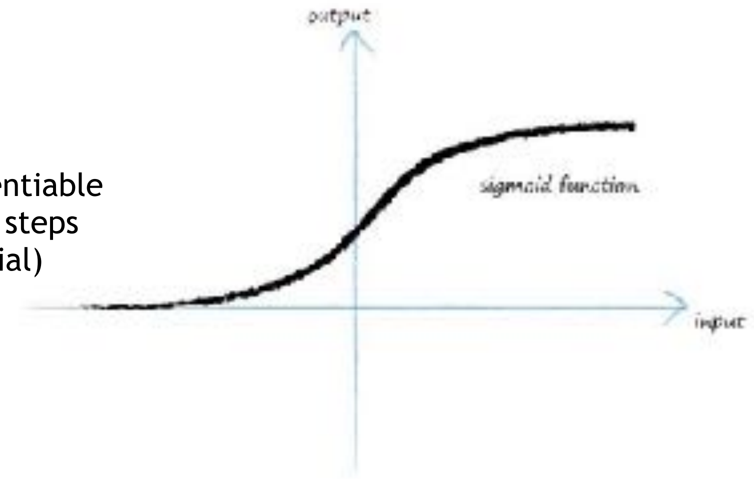
Each node generates an
output that is a function
of the sum of its inputs.

With more layers, it doesn't
require exponentially many nodes.
However, with the sign(x)
activation function it requires an
exponential search to find the
optimum weights.

The sign(x) is either 0 or 1. It is
not differentiable. We cannot
use gradient descent!

Sigmoid Activation Function

Key Insight: Use a differentiable activation function. Use steps proportional to the (partial) derivatives.

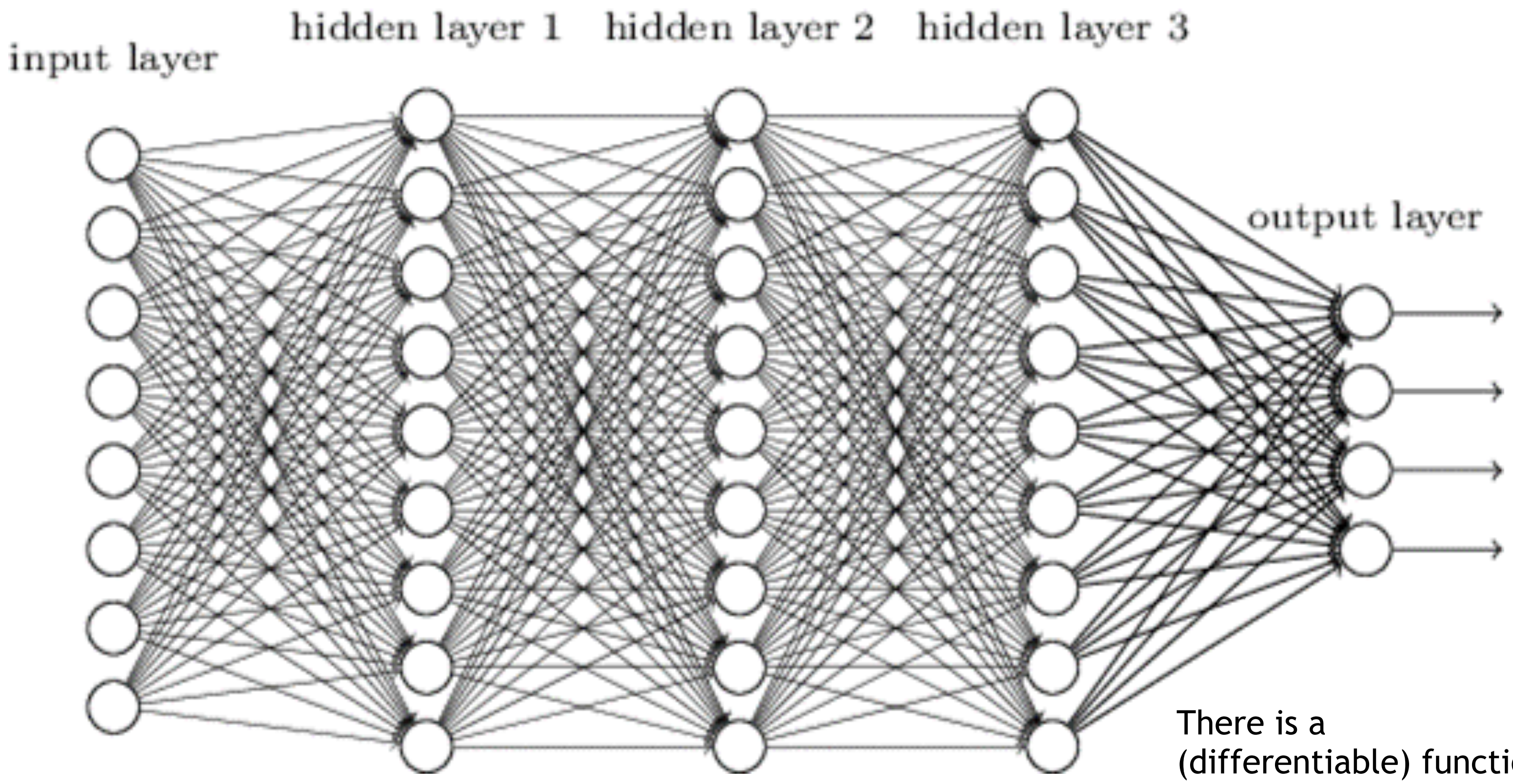


This smooth S-shaped sigmoid function is what we'll be continue to use for making our own neural network. Artificial intelligence researchers will also use other, similar looking functions, but the sigmoid is simple and actually very common too, so we're in good company.

The sigmoid function, also called the logistic function, is defined as $\sigma(x) = \frac{1}{(1+e^{-x})}$

Its derivative is $\sigma'(x) = \sigma(x)(1 - \sigma(x))$

Deep neural network



Each arc has a weight
w,
which is multiplied by
its input.

Each node generates an
output that is a
differentiable function of
the sum of its inputs.

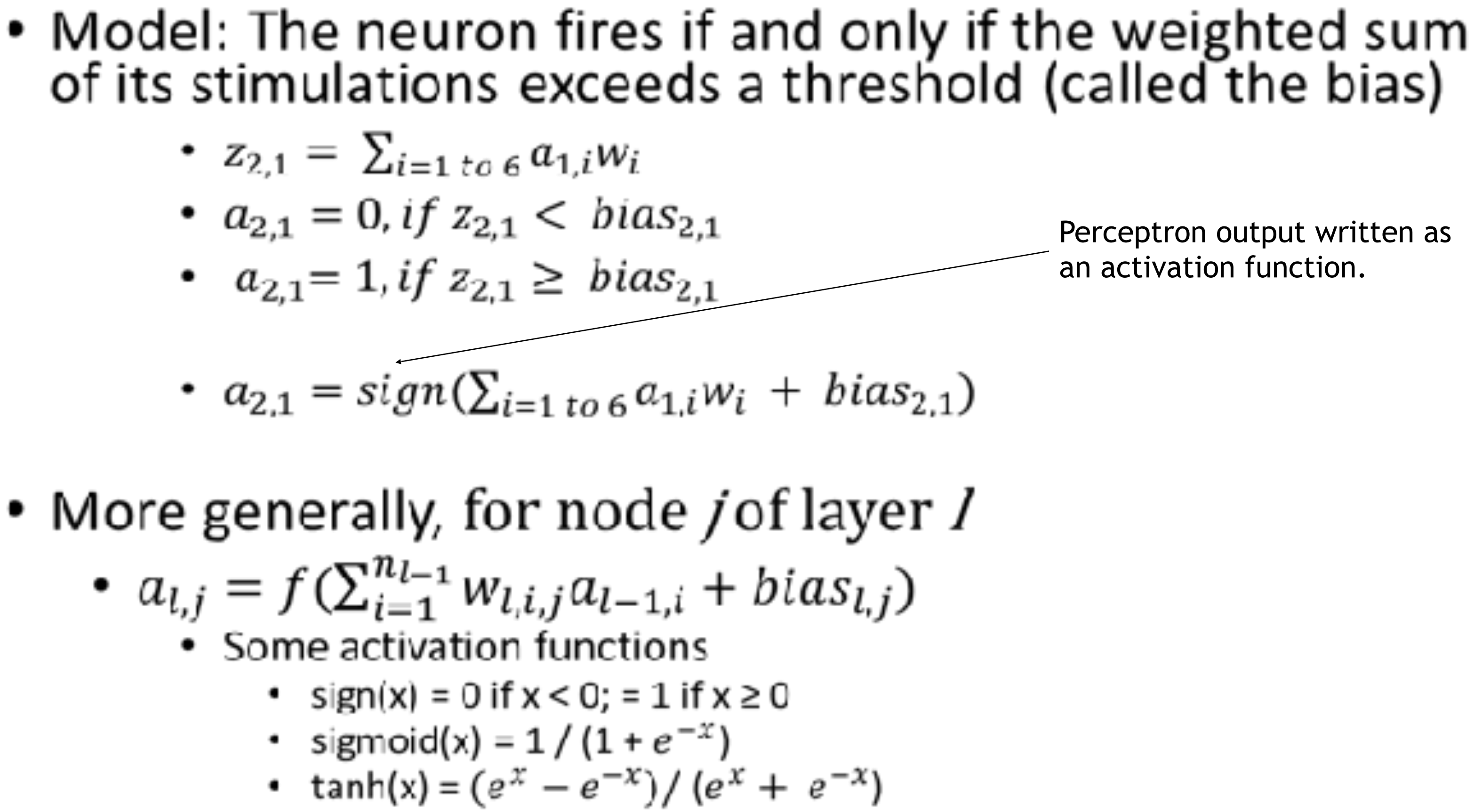
There is a
(differentiable) function
that measures the
discrepancy of the
actual output from the
desired output.

Key idea: use the
sigmoid function or
some other
differentiable
function.

$\longrightarrow$

Forward computation: The computation of the output of
each layer of nodes proceeds from left to right.

Generalizing the Perceptron Model



- Model: The neuron fires if and only if the weighted sum of its stimulations exceeds a threshold (called the bias)
  - $z_{2,1} = \sum_{i=1 \ to \ 6} a_{1,i} w_i$
  - $a_{2,1} = 0, if \ z_{2,1} < bias_{2,1}$
  - $a_{2,1} = 1, if \ z_{2,1} \geq bias_{2,1}$
  - $a_{2,1} = sign(\sum_{i=1 \ to \ 6} a_{1,i} w_i + bias_{2,1})$
- More generally, for node *j* of layer *l*
  - $a_{l,j} = f(\sum_{i=1}^{n_{l-1}} w_{l,i,j} a_{l-1,i} + bias_{l,j})$
    - Some activation functions
      - sign(x) = 0 if x < 0; = 1 if x ≥ 0
      - sigmoid(x) = 1 / (1 + e^{-x})
      - tanh(x) = ($e^x - e^{-x}$)/ ($e^x + e^{-x}$)

Perceptron output written as an activation function.

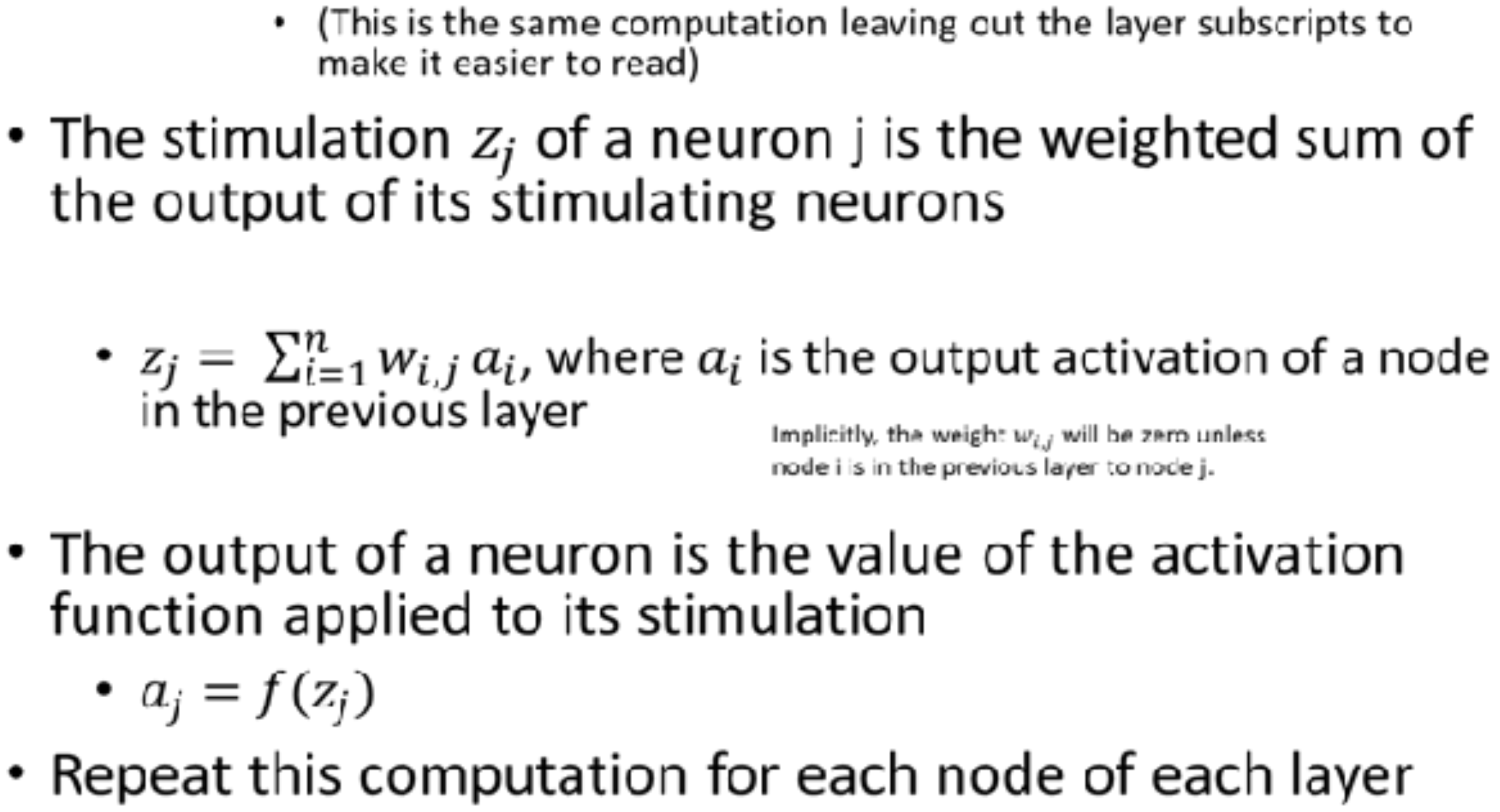
Treating Bias as the Connection Weight from a Special Constant Node

- Activation with bias
  - $a_{l,j} = f(\sum_{i=1}^{n_{l-1}} w_{l,i,j} a_{l-1,i} + bias_{l,j})$

- Include a node j = 0 whose activation is always 1
  - $a_{l-1,0} = 1$
  - Set $w_{l,j} = bias_{l,j}$
- Then
  - $a_{l,j} = f(\sum_{i=0}^{n_{l-1}} w_{l,i,j} a_{l-1,i})$

Most authors, including Michael Nielsen, do not follow this convention. Explicitly show the bias if you like that style better.

Too many subscripts?: Let the layer be implicit

- 

- (This is the same computation leaving out the layer subscripts to make it easier to read)
- The stimulation z_j of a neuron j is the weighted sum of the output of its stimulating neurons
  - $z_j = \sum_{i=1}^{n} w_{i,j}\, a_i$, where a_i is the output activation of a node in the previous layer

    Implicitly, the weight $w_{i,j}$ will be zero unless node i is in the previous layer to node j.
- The output of a neuron is the value of the activation function applied to its stimulation
  - $a_j = f(z_j)$
- Repeat this computation for each node of each layer

Node Activation (Computed Forwards)

- • $z_{l,j} = \sum_{i=0}^{n_{l-1}} w_{l-1,i,j} a_{l-1,i}$
- $a_{l,j} = \mathrm{f}(z_{l,j})$, for j = 1 to n_l
  - For example, $a_{l,j} = \sigma(z_{l,j})$
- $a_{l,o} = 1$

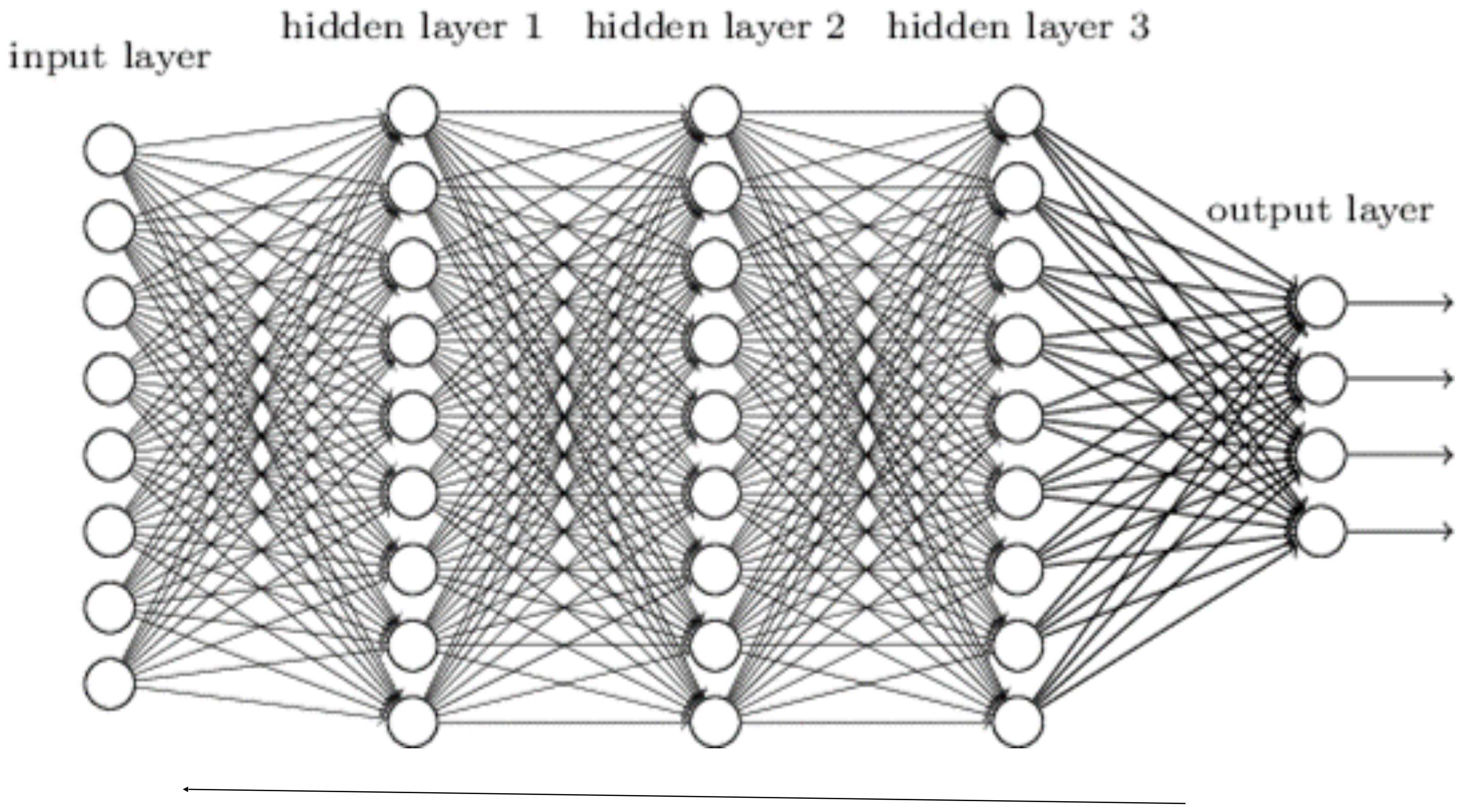
Do this computation for each node of each layer, then compare the output of the last layer with the desired output pattern.

Supervised Training

- In supervised training, we are told both the activations of the input layer and the target output for layer L
  - $a_{0,i} = input(i), for\ i = 1, \ldots, n_0\ the\ number\ of\ input\ nodes$
  - $y_k = target\ output\ for\ node\ k$
  - $o_k = a_{L,k} = actual\ output\ for\ node\ k$
  - $Error_k = (y_k - o_k), for\ k = 1, \ldots, n_L\ the\ number\ output\ nodes$

- A simple measure of the overall cost C of the errors is the mean of the squares of the individual errors
  - $C = C_{MSE} = MeanSquaredError = ({}^1/{}_{n_L}) \sum_{k=1}^{n_{output}} (Error_k)^2$
    - (Later, we will discuss a different measure of error that usually works better for classification problems)
- Next we need to use gradient descent to reduce the error
  - The only trained parameters are the connection weights

Deep neural network



Backpropagation: The computation of the derivative of the error function with respect to the weights proceeds backwards. (This is just the ordinary chain rule of elementary calculus.) Make an incremental update to each weight proportional to minus the derivative.

Rumelhart, D., Hinton, G., Williams, R., Learning representations by back propagating errors, Nature vol. 323, 9 October 1986.

The Chain Rule

- Suppose we have a function whose input is the output of another function, say h(x) = g(f(x))
- How much does a change in x affect the value of h(x)?
    - For each small change in x, f(x) changes by the derivative f'(x), that is $\Delta f = \frac{\partial f(x)}{\partial x}\Delta x$. This change in f produces a change in g, which is a also a change in h. Let z = f(x).
- $\frac{\partial h(x)}{\partial x} = \frac{\partial g(z)}{\partial z}\frac{\partial f(x)}{\partial x}$
    - This formula is called the chain rule. We will use it in computing the derivative of the error cost function going backwards through the neural network.

Change in Error for Change in an Output Node

- $C = C_{MSE} = (1/n_L) \sum_{j=1}^{n_L} (y_j - o_j)^2$
  - $o_j = a_{L,j}$
  - $\frac{\partial C}{\partial a_{L,j}} = -2(y_j - o_j)/n_L = -2(y_j - a_{L,j})/n_L$
  - By the chain rule,
  - $\frac{\partial C}{\partial z_{L,j}} = \frac{\partial C}{\partial a_{L,j}} \frac{\partial a_{L,j}}{\partial z_{L,j}}$
  - $\frac{\partial C}{\partial z_{L,j}} = -2(y_k - a_{L,j}) \frac{\partial a_{L,j}}{\partial z_{L,j}} / n_L$
  - $\frac{\partial C}{\partial z_{L,j}} = -2(y_k - a_{L,j})(\sigma(z_{L,j})(1 - \sigma(z_{L,j}))/n_L$

Proceeding Backwards, Layer by Layer

- It is convenient to represent each layer with the input $z_{l,j}$ to each node j rather than by its output $a_{l,j}$
  - In particular, let $\delta_{l,j} = \frac{\partial C}{\partial z_{l,j}}$
- Then $\frac{\partial C}{\partial w_{l,i,j}} = \frac{\partial z_{l,j}}{\partial w_{l,i,j}} \delta_{l,j} = \frac{\partial (\sum_{k=1}^{n_l} a_{l-1,k} w_{l,k,j})}{\partial w_{l,i,j}} \delta_{l,j}$
- $\frac{\partial C}{\partial w_{l,i,j}} = a_{l-1,i} \delta_{l,j}$ $\leftarrow$ The simple form of this equation is the reason for working with $\delta_{l,j}$.
- Note that we have already computed $\frac{\partial C}{\partial z_{L,j}} = \delta_{L,j}$
- We just need to compute backward from $\delta_{l,*}$ to $\delta_{l-1,*}$

Working Backward from $\delta_{l,*}$ to $\delta_{l-1,*}$

- For all j, $z_{l,j} = \sum_{k=0}^{n_{l-1}} a_{l-1,k} w_{l,k,j}$, therefore each $a_{l-1,k}$ contributes to $z_{l,j}$ for all j for which $w_{l,k,j} \neq 0$
- $\frac{\partial C}{\partial a_{l-1,i}} = \sum_{j=1}^{n_l} w_{l,i,j} \delta_{l,j}$
- $\frac{\partial C}{\partial z_{l-1,i}} = \frac{\partial C}{\partial a_{l-1,i}} \frac{\partial \sigma(z_{l-1,i})}{\partial z_{l-1,i}}$
- $\delta_{l-1,i} = \frac{\partial C}{\partial z_{l-1,i}} = \sigma(z_{l-1,i})(1 - \sigma(z_{l-1,i})) \frac{\partial C}{\partial a_{l-1,i}}$

- $\delta_{l-1,i} = a_{l-1,i}(1 - a_{l-1,i}) \sum_{j=1}^{n_l} w_{l,i,j} \delta_{l,j}$

The Backpropagation Equations

- $\delta_{L,j} = -2(y_k - \ a_{L,j})(\sigma(z_{L,j})(1 - \sigma(z_{L,j}))/n_L$

- $\delta_{l-1,i} = \ a_{l-1,i}(1 - a_{l-1,i}) \sum_{j=1}^{n_l} w_{l,i,j}\delta_{l,j}$

- $\frac{\partial C}{\partial w_{l,i,j}} = a_{l-1,i}\delta_{l,j}$

Summing Across All the Training Data

- The preceding computations were for a single input-output pair in the training data, but $C(m), a_{l-1,i,j}, (m) and\ \delta_{l,j}(m)$ all vary with the training item m (the weights $w_{l,i,j}$ are held constant for each iteration of training). Therefore, the gradient of the cost function for a training iteration is the sum of the gradient estimates for the individual training items.

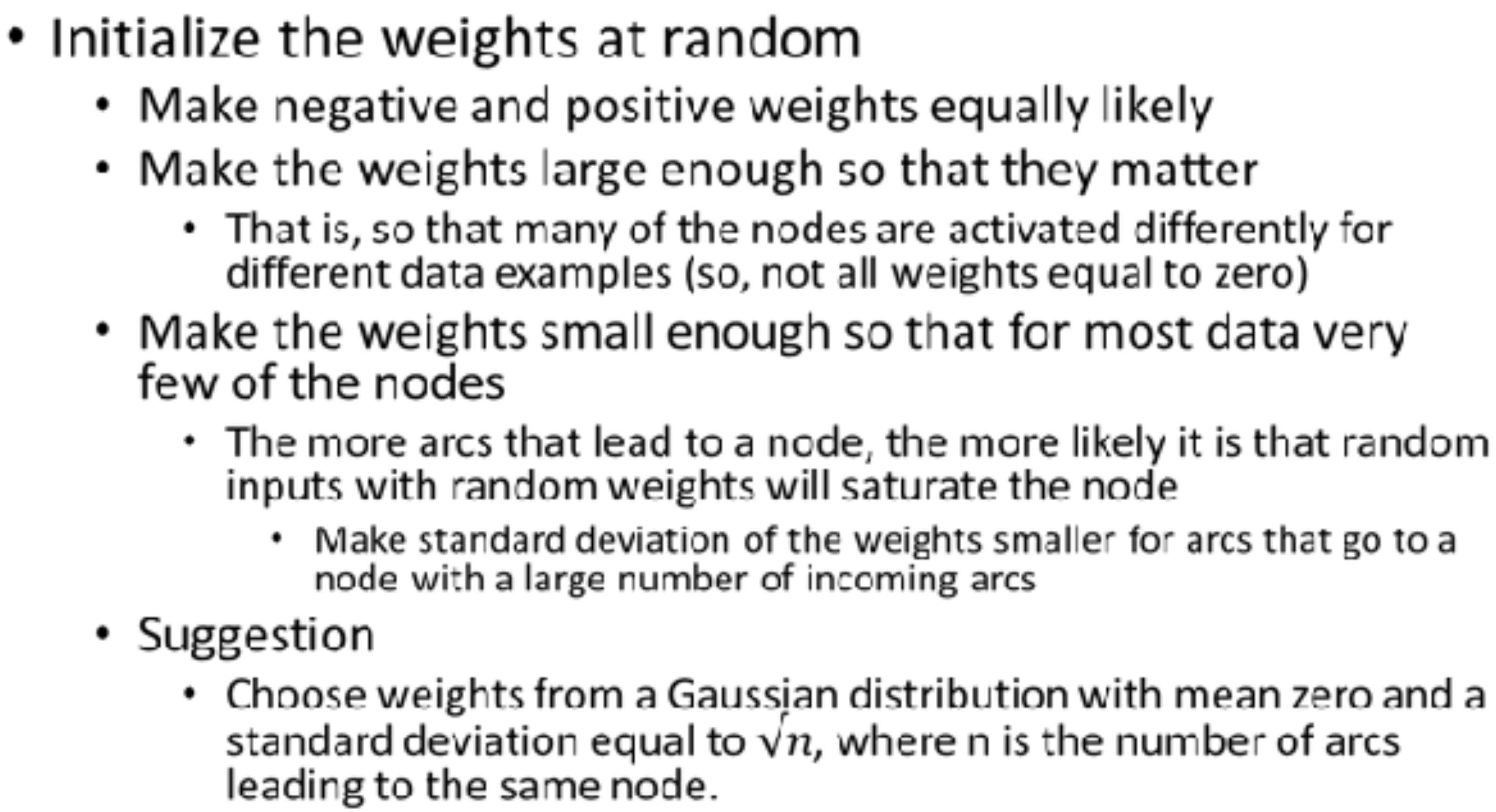
- $\frac{\partial C_{Avg}}{\partial w_{l,i,j}} = \sum_{m=1}^{M} \frac{\partial C(m)}{\partial w_{l,i,j}} / M = \sum_{m=1}^{M} a_{l-1,i}(m)\delta_{l,j}\,(m)/M$

- For each iteration, each $w_{l,i,j}$ is changed by a small amount proportional to the negative of the partial derivative of the cost with respect to the weight
  - $\Delta w_{l,i,j} = -\eta \frac{\partial C_{Avg}}{\partial w_{l,i,j}}$, where η is a parameter that controls the learning rate

Minibatches and Stochastic Gradient Descent

- Summing over all the training date gives us the gradient of the cost on the training data, so the gradient descent procedure will converge
  - However, this is only a statistical estimate of the true cost. The cost is defined as the average of the cost function across all the training data. The average cost on any one minibatch is a random variable that only approximates the cost on the total training set.
- The amount of computation per iteration of training is proportional to the amount of data that we sum together
  - We can get many more iterations of training (i.e. more updates of the estimated weights) by updating the weights after summing over a much smaller batch of training data (called a “minibatch”)
- Using a minibatch means that the gradient estimate is only a statistical estimate of the gradient on the training data, so the gradient descent only converges in a statistical sense. Therefore, this procedure is called “stochastic gradient descent”.
  - Because there are many more updates per pass through the training set, in practice it usually takes much less computation than batch updating using the whole training set.

Initialize the Weights



- Initialize the weights at random
  - Make negative and positive weights equally likely
  - Make the weights large enough so that they matter
    - That is, so that many of the nodes are activated differently for different data examples (so, not all weights equal to zero)
  - Make the weights small enough so that for most data very few of the nodes
    - The more arcs that lead to a node, the more likely it is that random inputs with random weights will saturate the node
      - Make standard deviation of the weights smaller for arcs that go to a node with a large number of incoming arcs
  - Suggestion
    - Choose weights from a Gaussian distribution with mean zero and a standard deviation equal to $\sqrt{n}$, where n is the number of arcs leading to the same node.

Training Algorithm Iteration

- 1. **Input a set (minibatch) of training examples**
2. **For each training example m**, set $a_{0,i}(m)$ and perform the following steps:
   a. **Feedforward:** For each l = 1, 2, …, L compute
   $\boldsymbol{z_{l,j}(m) = \sum_{i=0}^{n_l} w_{l-1,i,j} a_{l-1,i}(m)}$, $\boldsymbol{a_{l,j}(m) = \sigma(z_{l,j}(m))}$
   b. **Output error $\boldsymbol{\delta_{L,j}(m)}$:**
   $\boldsymbol{\delta_{L,j}(m) = -2(y_j(m) - a_{L,j}(m))(\sigma(z_{L,j}(m))(1 - \sigma(z_{L,j}(m)))/n_L}$
   c. **Backpropagate error:** For each l = L-1,L-2,…,2, 1 compute
   $$\delta_{l-1,i}(m) = a_{l-1,i}(m)(1 - a_{l-1,i}(m)) \sum_{j=1}^{n_l} w_{l,i,j}\delta_{l,j}(m)$$
3. **Gradient descent:** For each l = L-1,L-2,…,2, 1 update the weights
$$w_{l,i,j} \rightarrow w_{l,i,j} - \eta \sum_{m=1}^{M} a_{l-1,i}(m)\delta_{l,j}(m)/M$$

Sum the estimated partial derivatives across all the examples in the minibatch.

Go through all the training data separated into minibatches multiple times. Continue training iterations until a stopping criterion is met. It is recommended that you sample the data in random order.

Refinements will be added to improve learning speed and reliability.

What we have covered

- Information about course
- Computing with a network of simple units
- Feedforward computation of activation
- Minimizing a function: Gradient descent
- Chain rule
- Cost function
- Backpropagation algorithm
- Minibatches and stochastic gradient descent
- Iterative training

Part II: More about the Course

Reading and Research in Deep Learning

James K Baker

Hands-on: Learn-by-Doing

- You <u>will</u> write code
- You <u>will</u> do projects
- You <u>will</u> read research papers
- You <u>will</u> work in teams
- You <u>will</u> give presentations (team and individual)

If you think this sounds challenging, you are right. However, we will ease into it.

Easing into it:

1. At first you will just be copying tutorial code and doing small projects from the book(s).
2. At first you will only do very short individual presentations.
3. You will do many small projects from the books before tackling the research papers.
4. You will have the support of your fellow students.
5. At first you will only scan the research papers and prepare a short, structured gist.
6. Your team will get to choose one paper and concentrate on it for your major project.

By the end of the course, you should have the skills and knowledge to be a member of a team breaking records on testbed challenges.

Support Teams

Resources and Area of Responsibility	Responsible Team
Course Infrastructure, CMU Help Centers, CMU online info	Infrastructure
Brownlee Chap 2, online tutorials	Theano
Brownlee Chap 3, online tutorials	Tensorflow
Brownlee Chap 4, online tutorials	Keras
Brownlee Chap 5, Amazon docs	AWS
Pittsburgh Supercomputer seminar and documentation	PCS

I would like to have 3-4 students sign up for each team. You will be the gurus for the rest of the class. Note: If you expect to team up with another student for your major project, you should be on different support teams to increase your breadth of knowledge.

Each student will be on one support team and one major project team. The two types of teams will be interlaced.

Each support team will be the support group in your designated area, helping me and your fellow students.

Presentations from the Textbooks

Reading and Presentation Assignments	
Nielsen 1,2 Artificial Neural Nets, Backpropagation, Learning	Baker 18 Jan
Nielsen 3 Improved Learning	Baker 23 Jan
Each student will be the presenter for one tutorial	
Lewis 4 Deep Neural Nets Simplified (Python libraries)	TBD
Lewis 5 Deep Forecasting Model (Regression, Learning rate)	TBD
Lewis 6 ReLU, Num Iter, Regularization	TBD
ReLU, Momentum, Nestorov, rmsprop, adagrad, adam	Baker
Lewis 7 Binary Classifier, Holdout, Momentum	TBD
Lewis 8 Dropout,Minibatch	TBD
Lewis 9 Multiclass,softmax,rmsprop	TBD
Brownlee 6, 7, 8 Neural Net with Keras	TBD
Brownlee 9, 10 Flower Species: Grid search hyperparameters	TBD
Brownlee 11 Sonar Returns: Data preparation, tuning topology	TBD
Brownlee 12 Regression, Data preparation, tuning topology	TBD
Brownlee 13, 14, 15 Record keeping	TBD
CNN, RNN, BPTT, LSTM (Nielsen 6)	Baker
Brownlee 16 Dropout	TBD
Brownlee 17 Learning rate schedules	TBD
Brownlee 18, 19 Convolutional Neural Networks	TBD
Brownblee 20 Image augmentation	TBD
Brownlee 21 Object recognition	TBD
The Art of Gisting	Baker
Brownlee 22 Sentiment from movie reviews, word embedding	TBD
Brownlee 23 Recurrent Neural Networks (BPTT, LSTM)	TBD
Brownlee 24 Time Series Prediction wth MLP	TBD
Brownlee 25 Tine Series with LSTM	TBD
Brownlee 26 Sequence Classification Movie reviews	TBD
Brownlee 27 Stateful LSTM RNNs	TBD
Brownlee 28 Text Generation Alice in Wonderland	TBD

These three books and these presentations are just to prepare you to be able to do your major projects. You should read each of the assigned chapters and try to implement the tutorial code. If you have difficulty understanding one of the chapters or implementing the tutorial code, you may ask for help from other students. Ask for help from the guru or at least inform the guru that you had difficulty and got help from other students. Asking for and giving help will help you learn and will not hurt your grade.

If you have difficulty with the chapter for which you are a guru, ask for help from me rather than from your fellow students. It is important that I know about problems with particular chapters. Also, please inform me if a significant number of students have difficulty with your chapter.

You Set the Pace of the Course

- The first part of the course, presentation of the three tutorial books, is preliminary work to get you ready to do the Major Projects
    - I would like to get through this part as quickly as we can to leave you as much time as possible for your major projects
        - Subject to the requirement that you learn the material well enough to apply it
- Every chapter has information that you may need for your project
    - You should read every chapter
    - You should try to implement the code in every chapter
    - If you don’t have time to implement every chapter, you may skip some if you let me know that you are doing so
    - I will slow down the pace if more that a few students need more time

Write your own Code for some of the Tutorials

- Write code to implement backpropagation and the other fundamental techniques
    - The first two lectures will cover enough techniques
    - You can use this core code for many different problems
    - Do this at the beginning so you have the code available for all the projects
    - Run on the MNIST example in Nielsen
    - Ask questions if you have trouble
- The purpose is to learn better
    - You will not be graded on this
    - The tutorial books use library functions or supply sample code
        - You will learn it much better if you also write your own code and compare the results
    - You will only have time to do this on some of the tutorials
        - You should definitely do your own code on the tutorial you will be presenting
            - Ask for my help if it seems to require a technique that you haven't yet learned
    - You can use lessons that you learn in the class discussions
        - You will be graded on class participation

Student Presentations

- Each student will do two individual presentations
    - One of the tutorial sections
    - Presentation of the gist of a research paper
        - I will explain the art of gisting later in the course
- You will also do presentations as part of your teams
    - Support teams will give presentations on their area of responsibility
        - Explaining how to use the resources
        - Discussion and help session as necessary
    - Major project teams will give one or more preliminary presentations and a final presentation
        - Each team member will present of gist of at least one research paper
            - This is part of the process of selecting the project paper
        - The team will give a proposal presentation on their selected paper
            - Including proposed data collection and implementation
        - The team will give a final presentation
        - There may also be interim presentations

Length of Presentations

- The tutorial presentations should be very brief
  - Perhaps 5 minutes plus Q&A
    - We will adjust the pace as necessary
    - Take more time if other students need help
    - Take more time if you have discovered something that is not covered in the book
  - Most other students should have already read and implemented the section
    - And be ready to ask questions or comment during the Q&A
  - Your presentation should make the ideas clear to those who are having difficulty
    - Look for and report and potential gotchas
  - The quality of your presentation is more important than the quantity

Some Advice about Implementing Tutorial Code Examples

- Books and on-line tutorials have mistakes
    - To err is human; Do not assume everything is correct
    - Pay attention; Understand what you are doing
    - Report any errors you find (or anything you don't understand)
    - Report how long it takes to complete each section

It’s your Life, Take Advantage of this Opportunity

- Deep learning is “hot”
    - There is great demand for people with experience and understanding of deep learning
    - Google, Facebook, Uber, Twitter, Pinterest, IBM, Microsoft, Apple, Baidu, and many other companies are hiring as many good engineers as they can find
- However, everyone else knows this, too
    - Ten of thousands of computer scientists and engineers are aiming for jobs in deep learning
- The way to stand out is to have real experience and a greater depth of know than the other candidates
    - Being the member of a team that wins a competition or that achieves a new record result is the most convincing proof
    - This course intends to prepare you to do that, but you must do the work to prepare yourself
        - This is much more important than your grade in this course
        - Perhaps more important than all your grades

Advice for Gisting Research Papers

- First, try to get the “gist”
  - What is the main achievement?
  - What are the new ideas (often only one or two)?
  - What ideas are used by reference?
    - Which ones do you already know?
    - Which, if any, did not occur in our introductory books?
  - For selection of potential projects:
    - What tests or benchmarks?
      - How available is the training data?
      - How much computation will be required?
      - Is a scaled-down version feasible?
- I will give examples and explain the art of gisting in more depth when we get to that part of the course