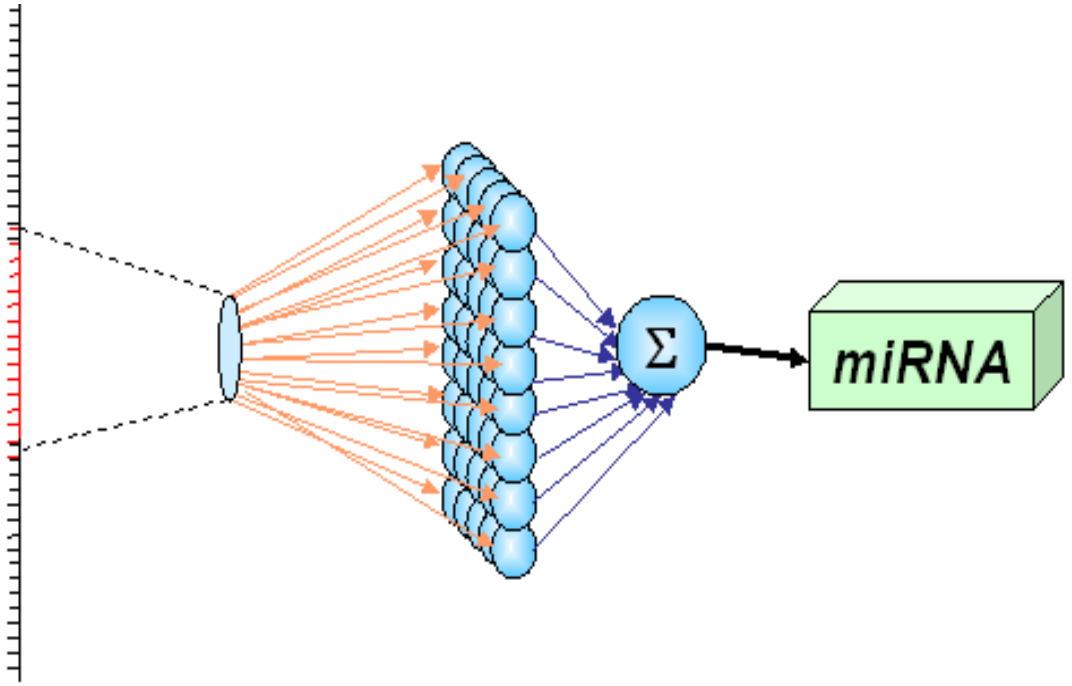
PERL/XML based Neural Networks: miRNA and DNA Scanner



by Bryce L. Meyer
b.l.meyer @ att.net
Presented to the St. Louis Unix Users Group

Outline

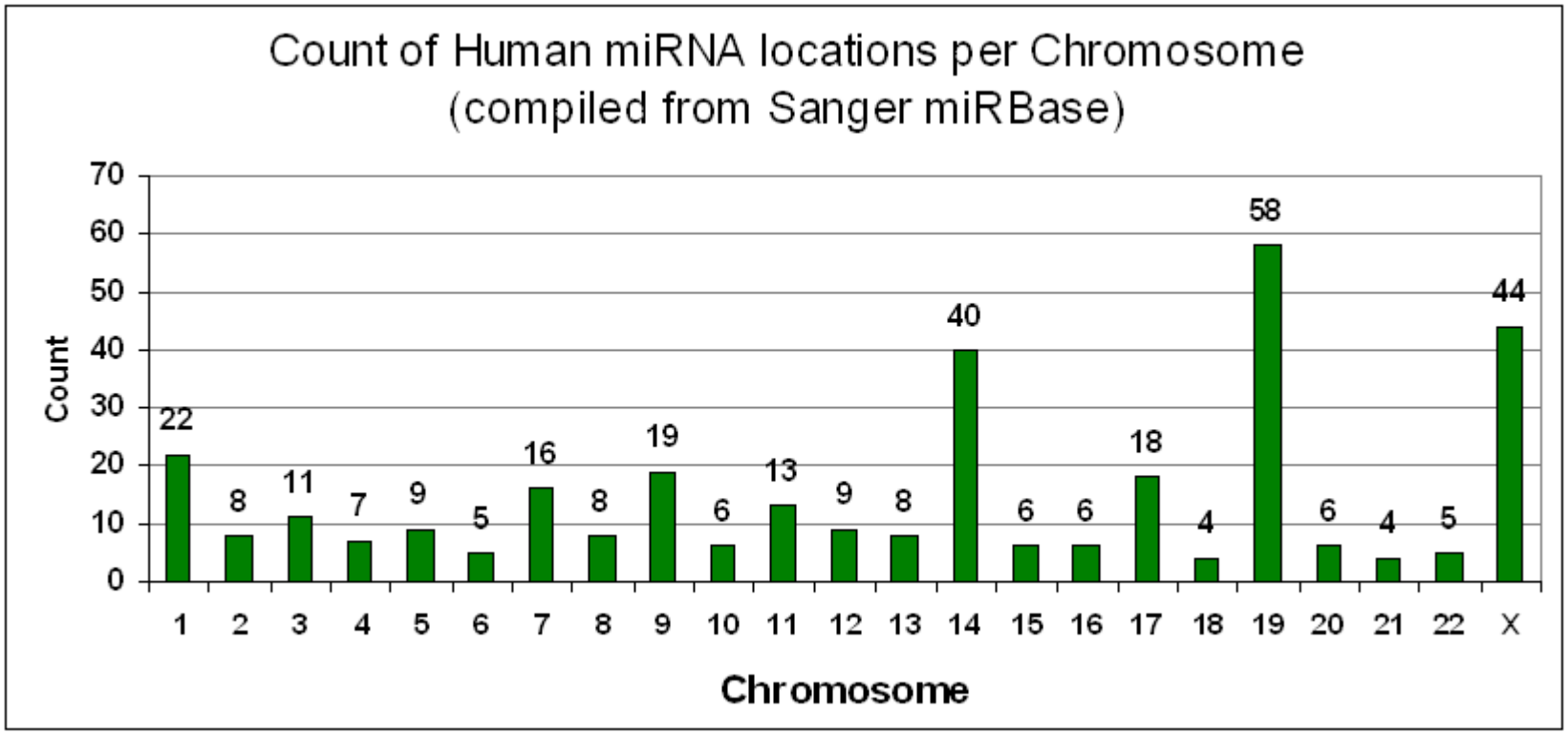
- Why am I doing this? (Problem)[Biology Stuff]
- What is a Neural Network? (Basics) [Math Stuff]
- What parts do we need for the scanner? [PERL]
  - Input Encoding (IE)
  - Forward Pass (FP)
  - Truth Data (+Making a false set)
  - Backward Pass (BP)
  - Rinse and Repeat: When is it done? (Training)
  - Storing Data
- Usage
- Future Work
- References and Recommended Reading

Problem: What are MicroRNAs?

- MicroRNAs (miRNAs, in GenBank labeled as MIR-###) are short (~20 base pair) sections of messenger RNA (mRNA).
- Can easily find a list here: **http://microrna.sanger.ac.uk/cgi-bin/sequences/browse.pl** (and can walk through to Ensembl to see chromosome context, like here : http://www.ensembl.org/Homo_sapiens/contigview?region=21&vc_start=41450061&vc_end=41462060)

Problem: Human miRNA Facts

(Important for hunting them!, as of 2006, see link for current)



Known Human miRNA Sizes in bases for 332 Known Precursors:

	Hairpin	Near-Mature
Avg.:	87.4	21.8
Max.:	137	25

Data compiled from raw data at [Sanger 2006] miRNAbase @ http://microrna.sanger.ac.uk/

Known Human miRNA Sizes in bases for 332 Known Precursors:
Where is Near-Mature precursor found?
Forward Stem (+): 189
Reverse Stem (-): 143

Data compiled from raw data at [Sanger 2006] miRNAbase @ http://microrna.sanger.ac.uk/

http://microrna.sanger.ac.uk/

Problem: Since miRNAs are Too Short We Want Hairpins!

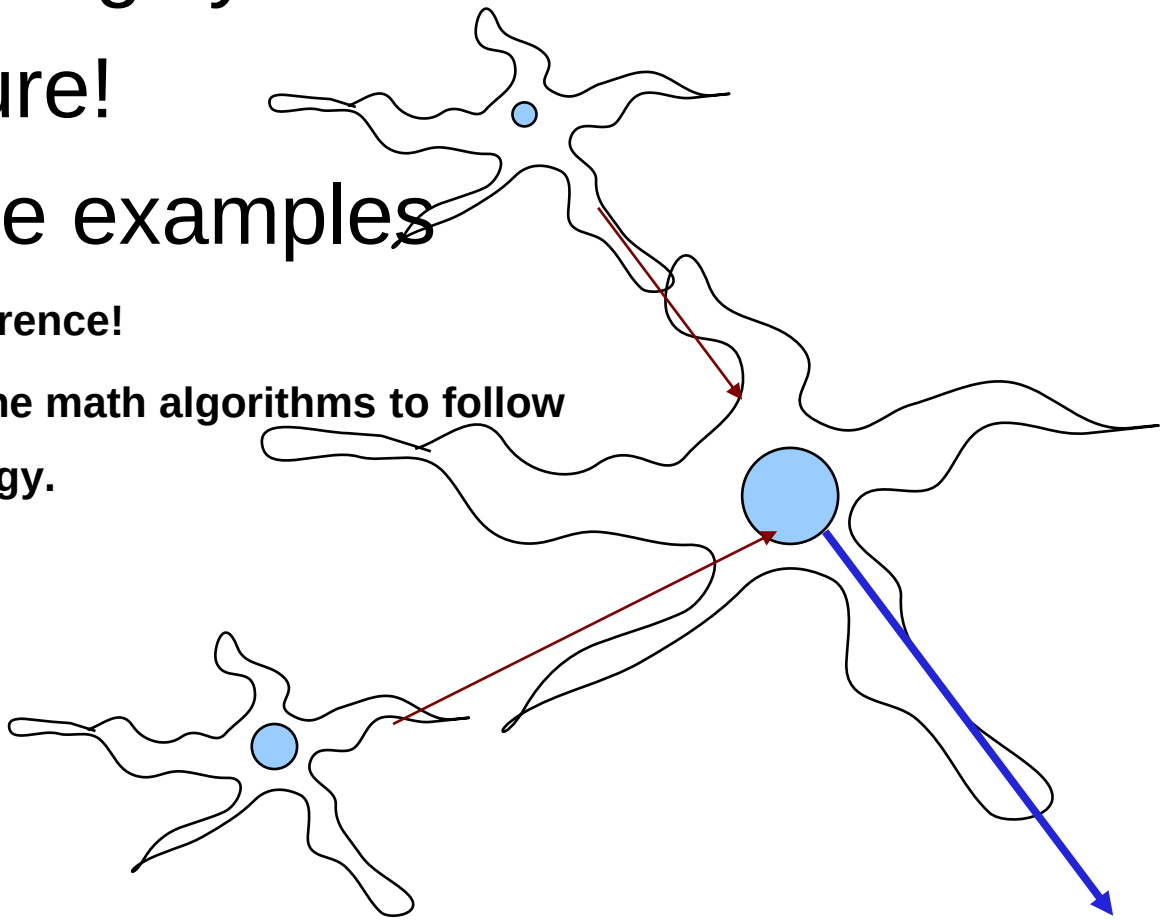
- **NEED: Make a learning program (a Neural Network) that will scan DNA for Hairpins**
- Mature miRNAs are too short for a pattern (I found out the hard way :0)
- Hairpins can be found in DNA, these hairpins are used to make miRNAs
- Hairpins MAY have a pattern, and are bigger (80+ bases)
- NOTE: Same technique can be used for ANY pattern (i.e. Non-miRNA stuff) in DNA
  - Feel like using it to find new proteins, oncogenes, etc.?

Problem Solution Plan

1. Obtain miRNA Data for Hairpins (from Sanger MIRBASE)
2. Develop an encoding method; determine sizing from miRNA data
3. Develop a data schema (XML in my case)
4. Make the Neural Network, train it, and alter until it stabilizes at 99.999% (or find out how firm is the pattern for miRNAs)
5. If I fail at #4, find a new DNA disease pattern and redo NN using core code and combine with other methods.
6. Use the stabilized NN and a custom DNA scanner to look over areas near disease causing genes
7. Send answers to key researchers in field and publish. Provide the code core as a tool set to any researcher.

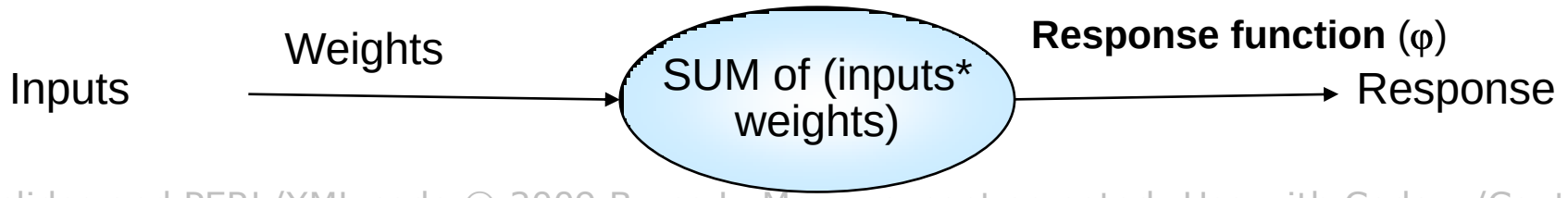
Basics: What is a Neural Network?

- Neural Network (NN) = Mathematical/ Programmable way to determine and use patterns in a learning system
- Copied from Nature!
- Your eye/brain are examples
- **[Hayken,1994] is a great NN reference!**
  - **Is used as initial basis for the math algorithms to follow**
  - **I deviate from his terminology.**



What are Neural Networks?

- Neural Networks (NNs) operate by simulating how neurons function
  - Stimuli (Inputs) enter the neuron
  - Neuron accumulates (Sums) inputs until it reaches a point to force an action potential (act positively or negatively: +1 or -1 according to response function), which may be transmitted to other neurons.
  - Feedback alters sensitivity (weighting) of each input. (learning via training)
  - An array of connected neurons forms a network.
  - The knowledge gained by the network is represented by all the weights of the network.
- NN are best at finding patterns (if they exist!).



Neural Network Theory Basics: The "Forward Pass"

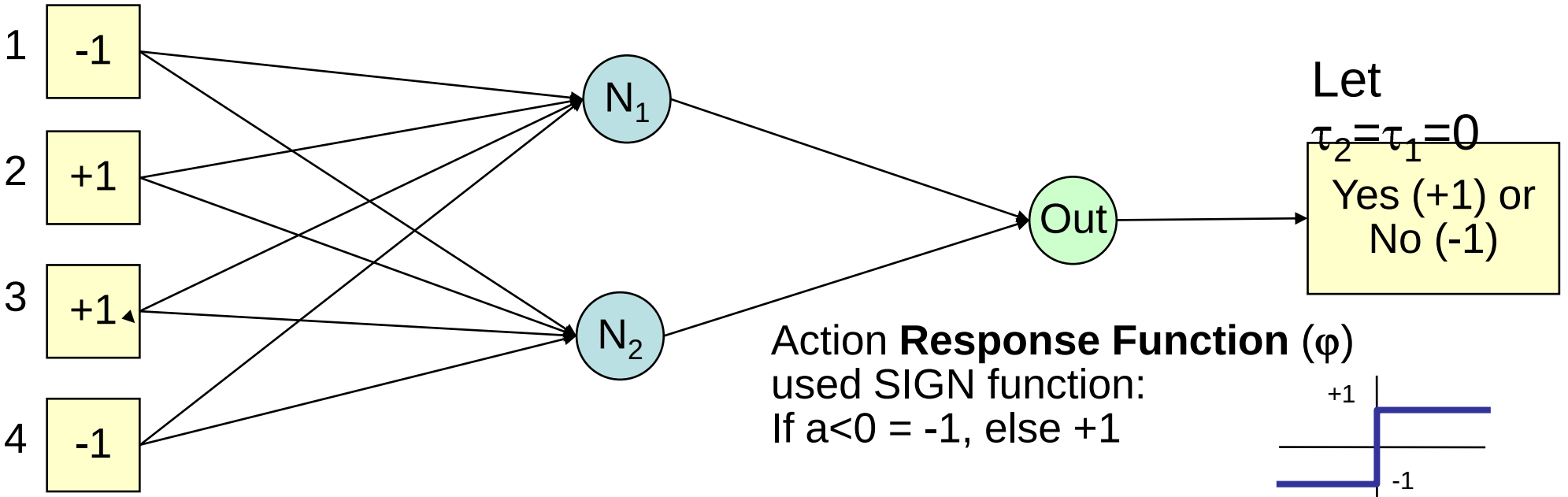
Input Array | Input Nodes | Output Node | Decision

Input Weight Matrix ([Wi]):

[I]

0.125	0.125	0.125	0.125
0.125	0.125	0.125	0.125

Output Weight Matrix ([Wo]):
[0.5 0.5]



Action **Response Function** (φ)
used SIGN function:
If a<0 = -1, else +1

BASIC EQUATION:

$\varphi([Wi]*[I]+\tau_1) = [N]$

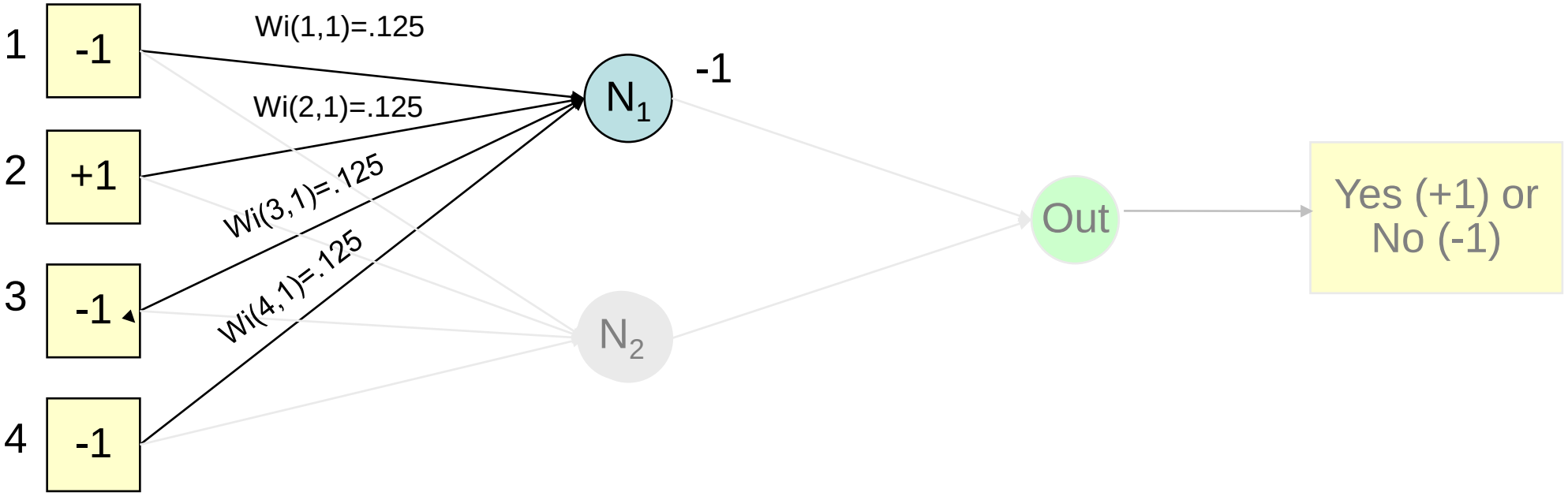
or $N(i)=SIGN[SUM(I(j)*Wi(i,j) + \tau_1)]$

[W]: Weights going into the node
Used 2 weight matrices:
[Wi] and [Wo]

Neural Net Theory Basics
(continued)

Input Weight Matrix ([Wi]):

0.125	0.125	0.125	0.125
0.125	0.125	0.125	0.125



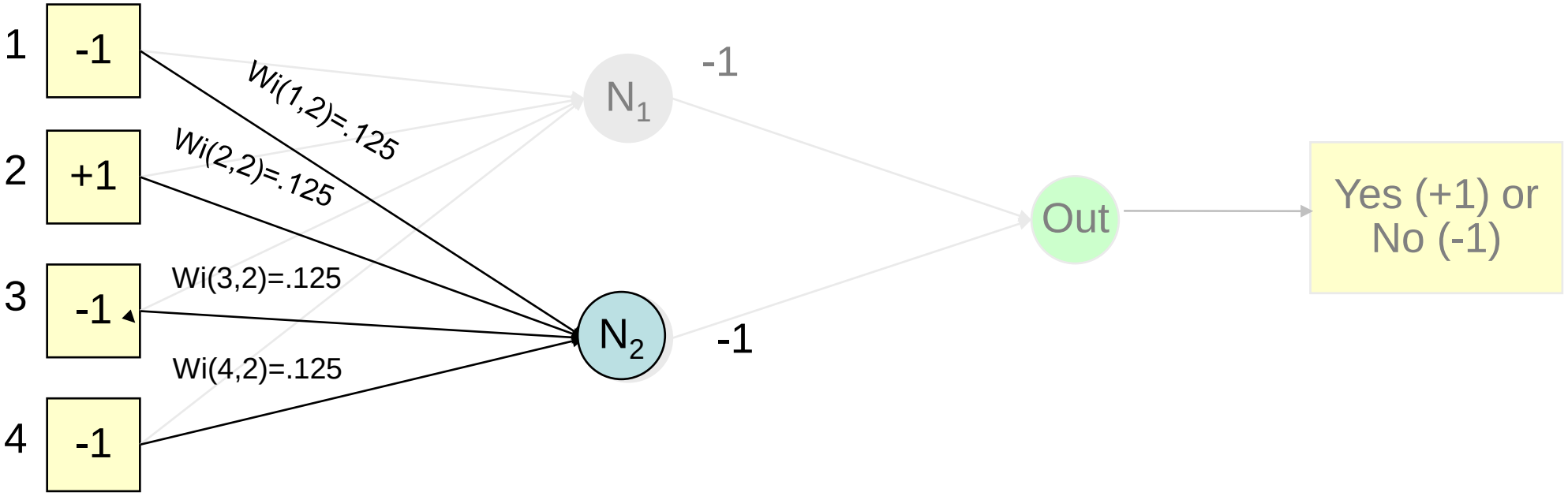
N(1)= SIGN[Wi(1,1)*I(1)+Wi(2,1)*I(2)+ Wi(3,1)*I(3)+Wi(4,1)*I(4)]
= SIGN[(0.125)*(-1)+(0.125)*(+1)+(0.125)*(-1)+(0.125)*(-1)]
= SIGN[-0.375] {note this is Y(1) -- used later}
= -1

Input Array Input Nodes

Neural Net Theory Basics
(continued #2)

Input Weight Matrix ([Wi]):

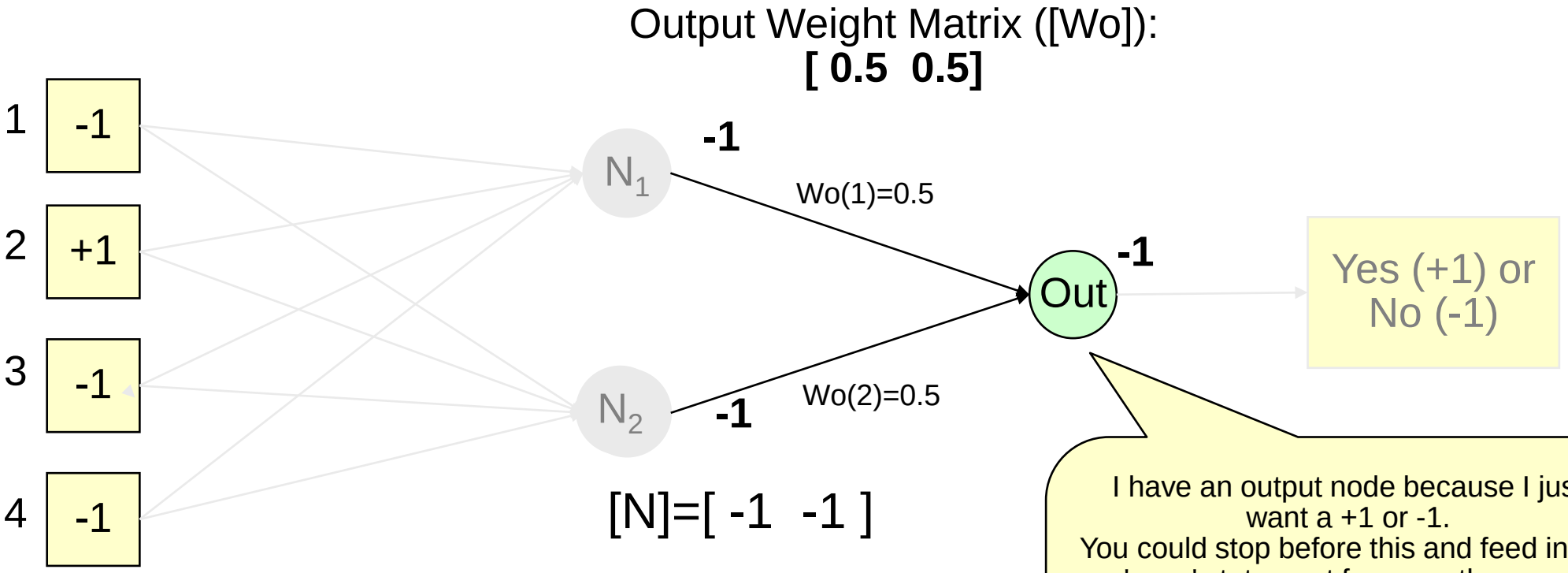
0.125	0.125	0.125	0.125



N(2)= SIGN[Wi(1,2)*I(1)+Wi(2,2)*I(2)+ Wi(3,2)*I(3)+Wi(4,2)*I(4)]
= SIGN[(0.125)*(-1)+(0.125)*(+1)+(0.125)*(-1)+(0.125)*(-1)]
= SIGN[-0.375] {note this is Y(2) -- used later}
= -1

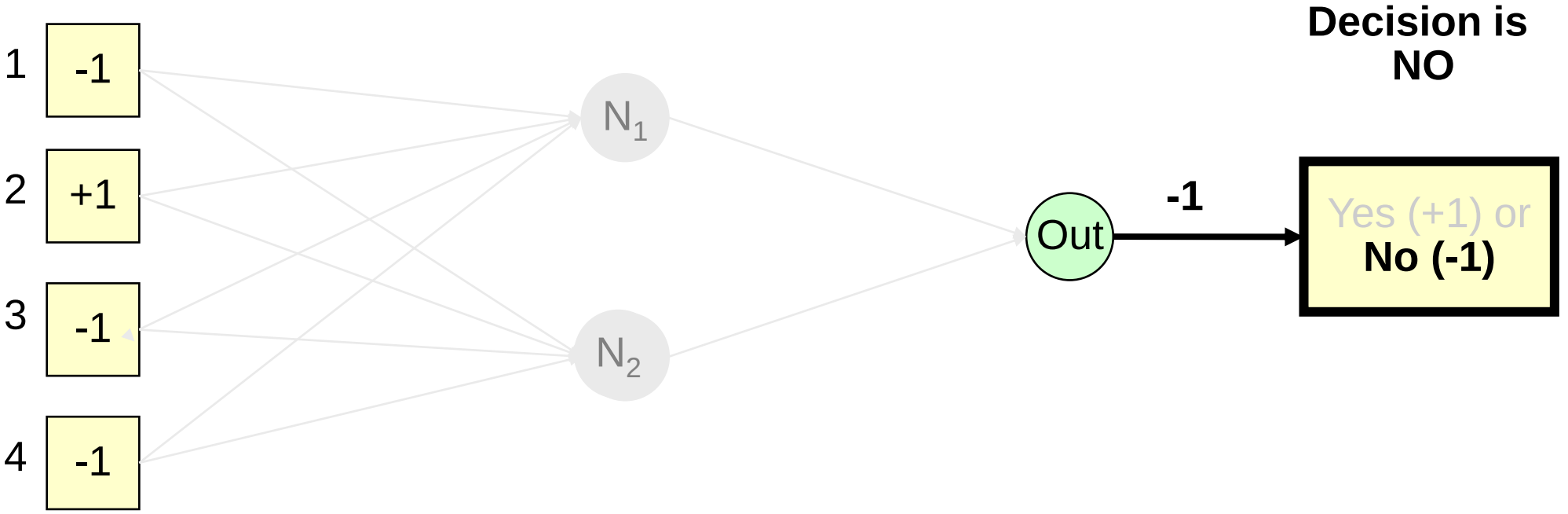
Input Array Input Nodes

Neural Net Theory Basics
(continued #3)



Out = SIGN[Wo(1)*N(1)+Wo(2)*N(2)]
= SIGN[0.5*(-1)+0.5*(-1)]
= SIGN[-1.0] {note this is O(1) -- used later}
= -1

Neural Net Theory Basics
(continued #4)



Input Array | Input Nodes | Output Node | Decision

Neural Net :
Feedback=Training= “Backward pass”*

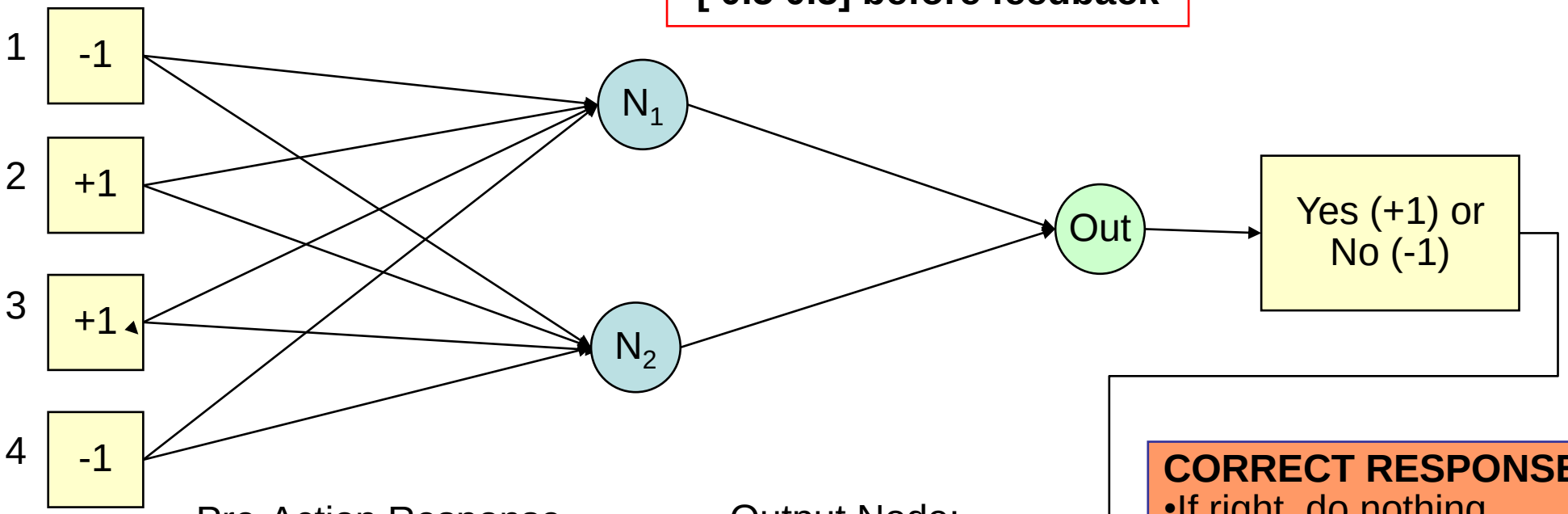
Input Weight Matrix ([Wi]):
Before feedback:

0.125	0.125	0.125	0.125
0.125	0.125	0.125	0.125

$[dWi]=\beta_2*(C-[Y])*[I]^T+\lambda_2$

Output Weight Matrix ([Wo]):
[0.5 0.5] before feedback

$[dWo]=\beta_1*(C-[O])*[Y]^T+\lambda_1$



Pre-Action Response (before doing the SIGN for N)
=[Y]
=[-0.375 -0.375]

Output Node: Pre-action response
= [O]
=[-1]

*= there are MANY training methods, and equations, I just picked one. See [Hayken,1994] Here I chose to use the pre-activated Neuron output to use in training, you may decide to use the post activated neuron output ([N] in lieu of [Y], Result in lieu of [O]).

NN Basics: The Backward Pass Example (an independent training method)

Let $\beta_2 = \beta_1 = 1$
Let $\lambda_2 = \lambda_1 = 0.01$

$[dWi] = \beta_2 * (C - [Y]) * [I]^T + \lambda_2$

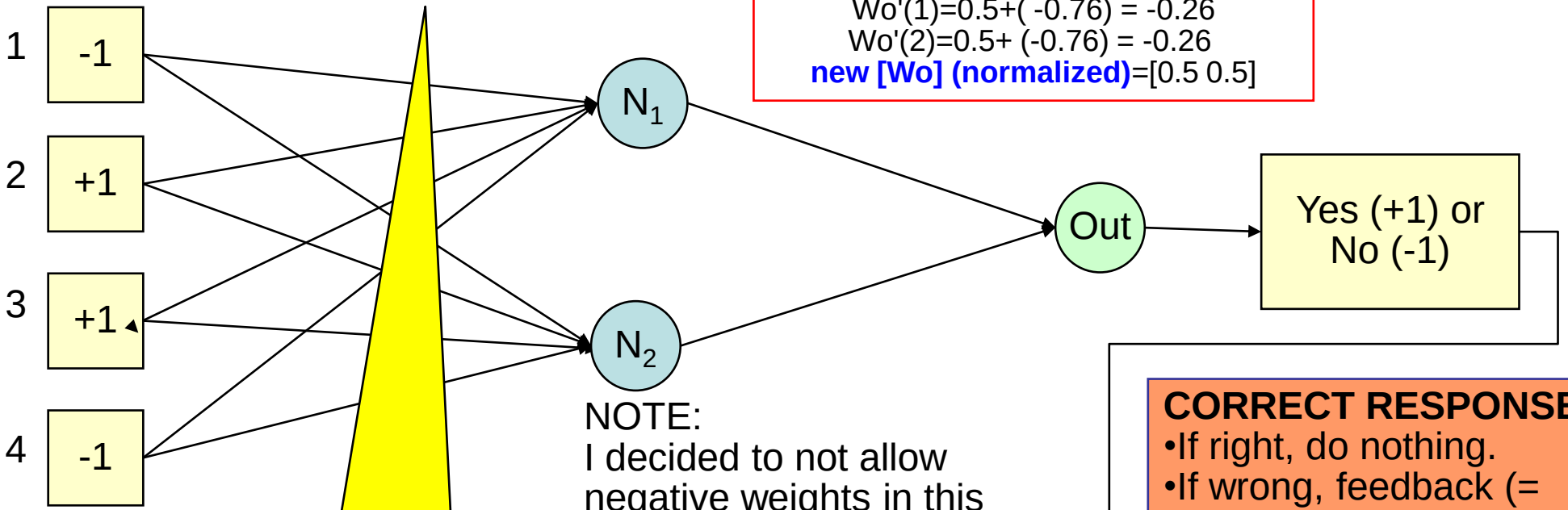
Input Weight Matrix ([Wi]):

dWi() = 1*(C- Y())*I() + 0.01
dWi(1,1)=1*(1-(-0.375)*(-1)+0.01=-1.365
Wi'()=Wi()+dWi()
Wi'(1,1)=0.125+(-1.365)= -1.24
new [Wi] (normalized):

0.11	0.14	0.14	0.11
0.11	0.14	0.14	0.11

Output Weight Matrix ([Wo])
dWo()=1*(C-O()) * Y() + 0.01
dWo(1)=1*(1-(-1))*(-0.375)+0.01= -0.76
dWo(2)=1*(1-(-1))*(-0.375)+0.01= -0.76
Wo'()=Wo(i)+dWo()
Wo'(1)=0.5+(-0.76) = -0.26
Wo'(2)=0.5+ (-0.76) = -0.26
new [Wo] (normalized)=[0.5 0.5]

$[dWo] = \beta_1 * (C - [O]) * [Y]^T + \lambda_1$

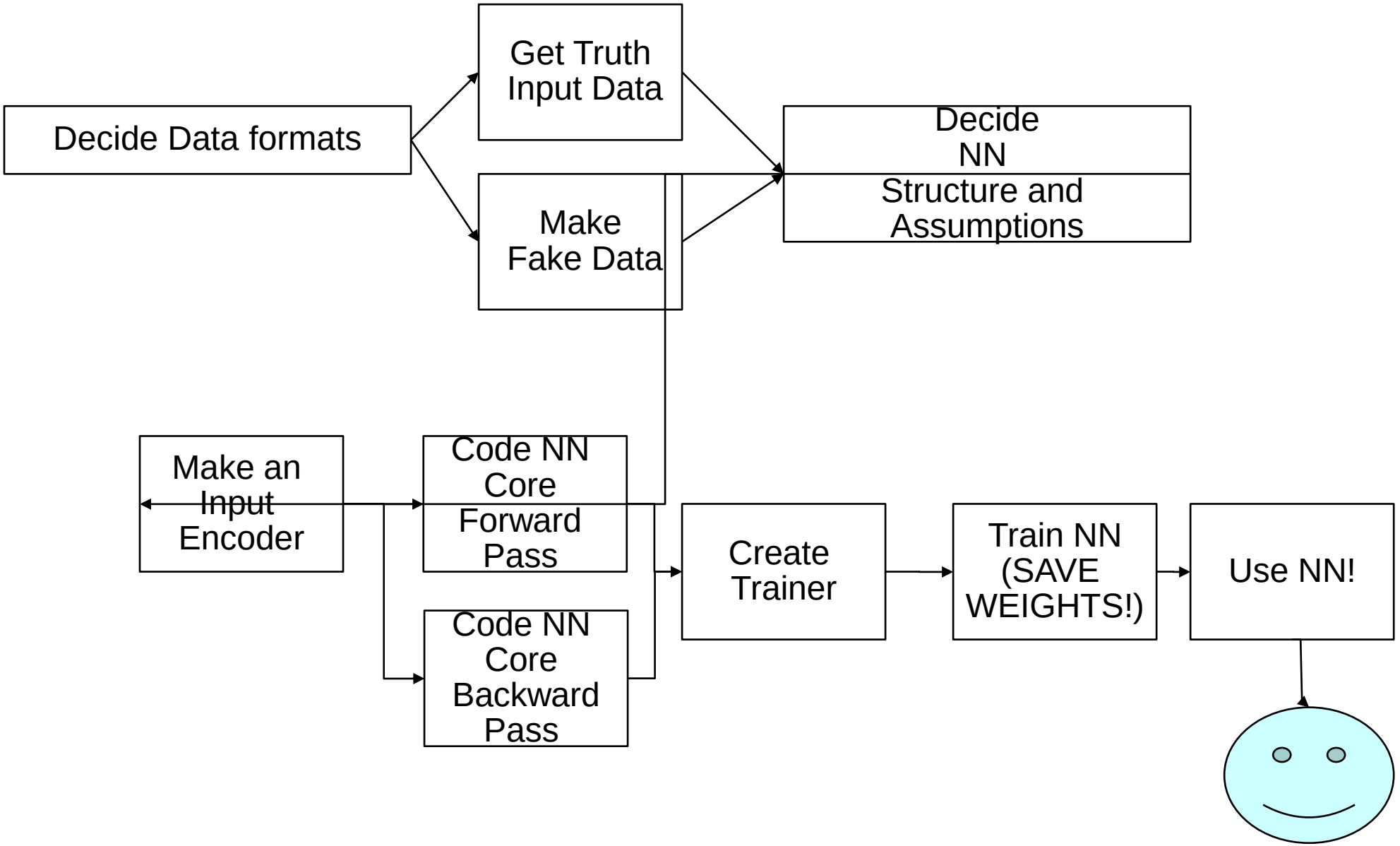


NOTE: [Wi] changed to favor the positive inputs and not favor the negative ones!

NOTE:
I decided to not allow negative weights in this example (you may decide otherwise)
So when I **normalize**:
Sum(all elements in weights matrix)=1

CORRECT RESPONSE ?
- If right, do nothing.
- If wrong, feedback (= negative feedback model).
- ASSUME NOT CORRECT HERE (Correct=+1=C)

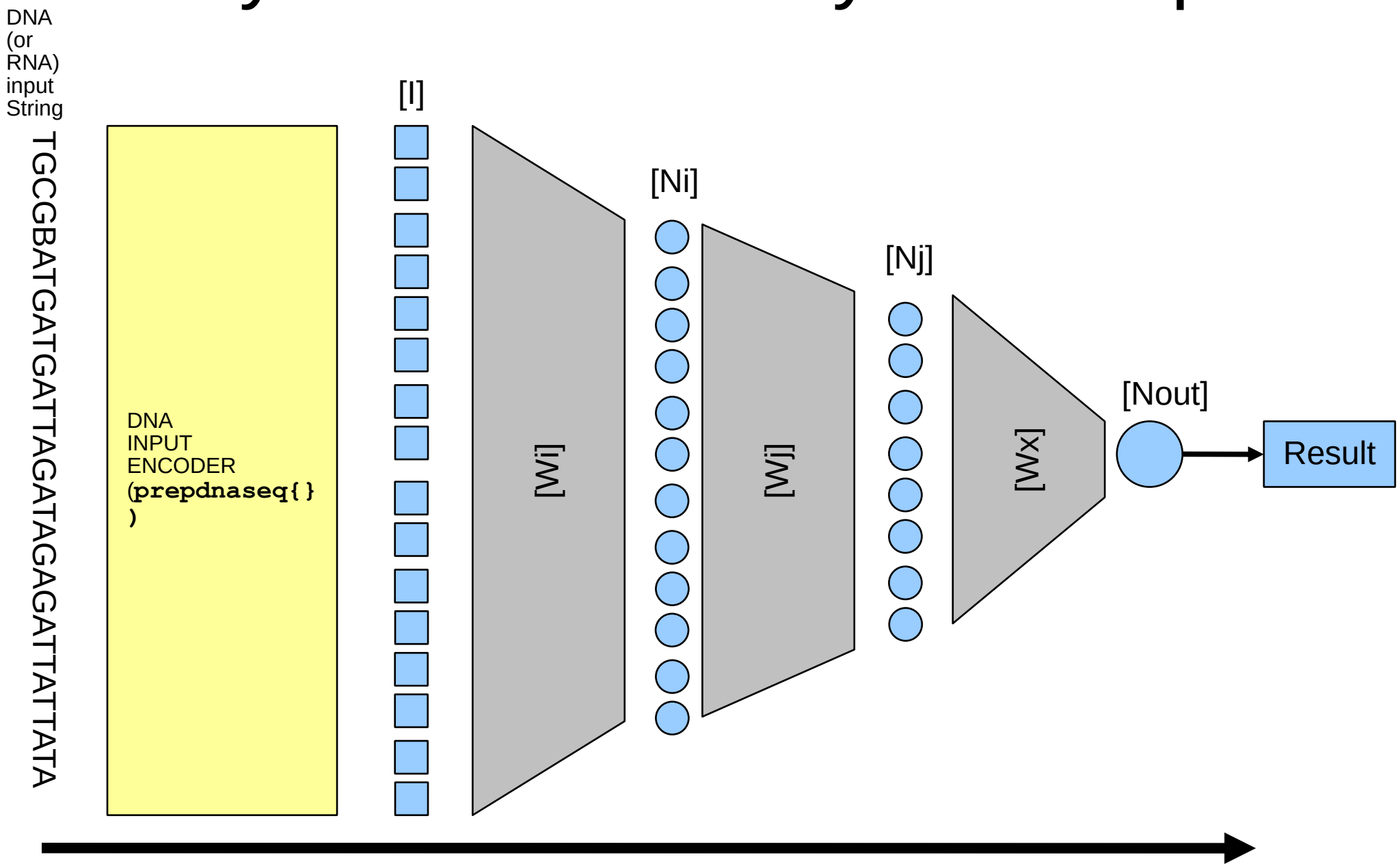
Process



Phases in Making a Neural Network Work

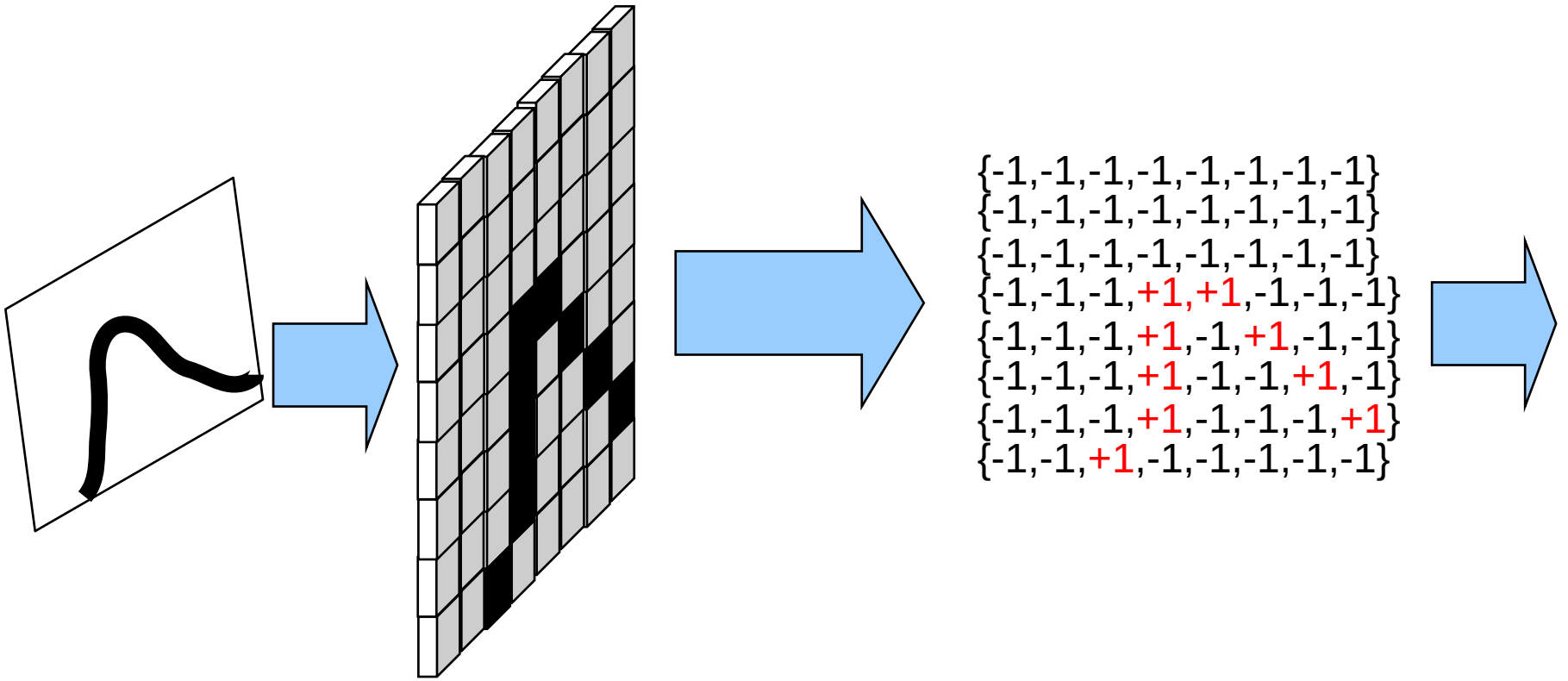
- Input Preparation (Input Encoding): Need to know how I will make my input data into +1/-1's. Know as much about your goal and input as possible.
- Data Sets:
  - Need a set of true data + Need a set of false data
- Sizing and Layers: How many neuron layers of what size?
- Training: Using the known true and false data, train the NN until it is right regularly a set % of time (~95%, 99%, 99.999%) (LONGEST PART RUNWISE!!!!!)
    - The weaker the pattern in the truth data, the more training and more/bigger layers are required
    - If the array is less than 100%, then it will have an error rate.
    - What assumptions? What training model?
- Usage: Using the trained weights matrices, scan unknown data (forward passes) and find out what it is!

My Pseudo Two layer Example



Input Encoding: Other Problems

- Images



What parts do we need for the DNA scanner?

- Since NN sees only +1/-1, but DNA is {A,T,G,C} I need an 'Input Encoder' (IE) to make data into +1/-1
- Need a nested loops to perform the 'Forward Pass' (FP)
- Need a truth table comparison to determine correctness
- Need a 'Backward Pass' (BP) to feed the results back
- Need to store statistics and weights
- Need support routines/programs (store data, retrieve data, store runs/back-up data, make training data)

IE: Encoding DNA

- DNA has only 4 nucleotides: A,T,G,C
  - A binds to T; G to C
- RNA has the same letters with U in lieu of T
- Use 0 for -1, 1 for +1, then use PERL regexs and arrays:

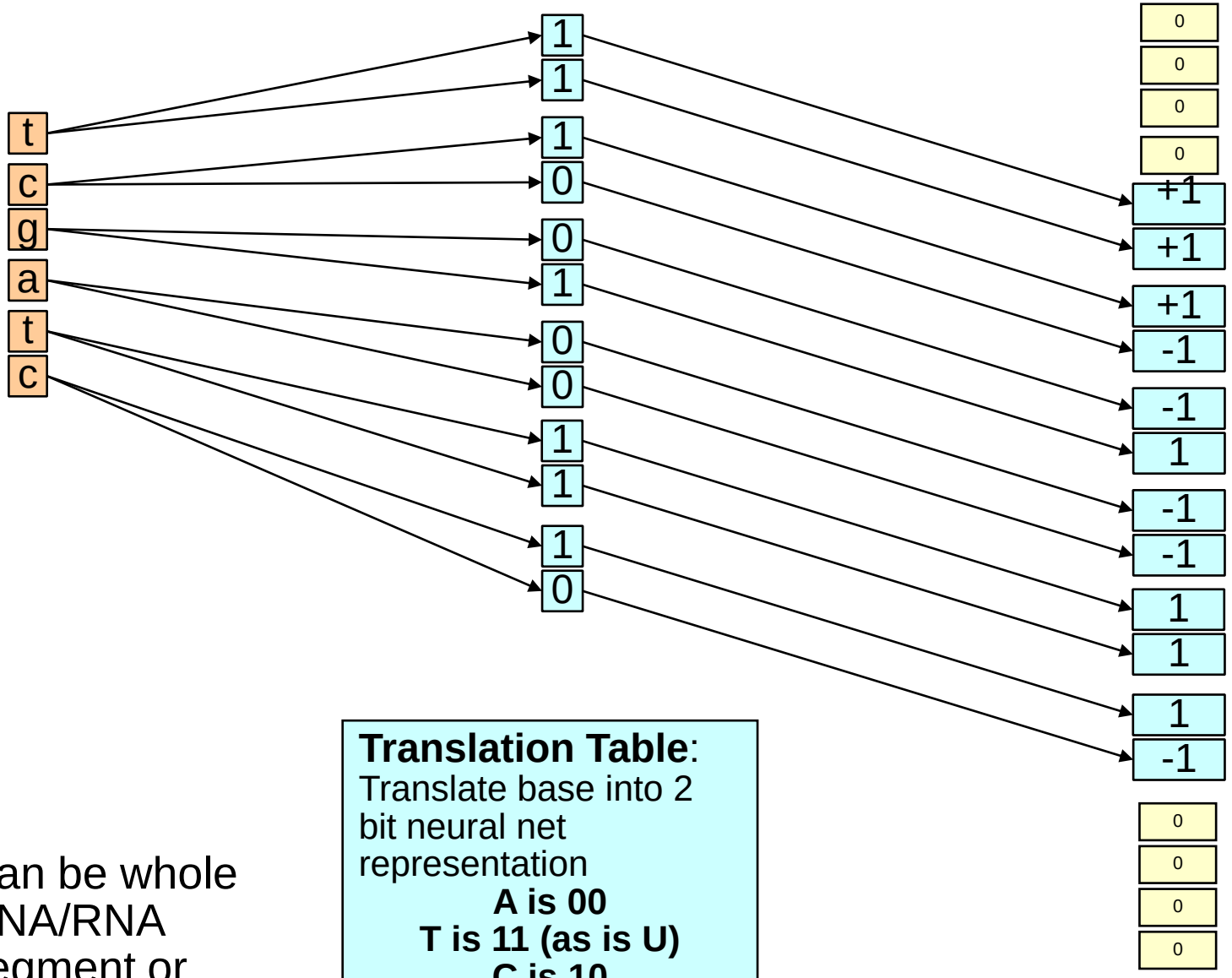
```
##########################################
### SUB BINCONVERT Binary converter from base
###########################################
sub binconvert{
  $bseq=~s/a/00/g;
  $bseq=~s/t/11/g;
  $bseq=~s/u/11/g;
  $bseq=~s/c/10/g;
  $bseq=~s/g/01/g;
  @temparray=();
  @temparray=split("",$bseq);
  @finarray=();
  foreach $titem(@temparray){
    $titem=~s/0/-1/s;
    push @finarray,$titem;
  }

}
```

Translation Table:
Translate base into 2 bit neural net representation

A is 00
T is 11 (as is U)
C is 10
G is 01
then swap '0''s for -1

NOTE: I will try to use simple code here, but there are many better ways to code this and the following PERL snippets :0)

IE: Reading in the Input Strings into the Input Array



Can be whole DNA/RNA segment or piece scanned into the NN

Translation Table:
Translate base into 2 bit neural net representation

A is 00
T is 11 (as is U)
C is 10
G is 01
then swap '0"s for -1

Pack in '0"s since our truth data VARIES in SIZE!!! 0's essentially numb inputs to the NN (like beer)

Input Array size = fixed
Make array same size as input array Center and Stuff with 0's if too small)

My DNA Scanner Example

- Uses an input array [I] fed by the DNA Inputter
- Has two primary Neuron Matrices (Arrays) [Ni] and [Nj]
  - [Wi] is the weights that multiply [I] going into [Ni]
  - [Wj] is the weights that multiply the output of [Ni] into [Nj]
- One output neuron [Nout] to get to a +1/-1 output.
  - [Wx] is the weights that feed [Nj] into [Nout]

IE: Prep DNA for Input subroutine

```
sub prepdnaseq{
#prep input array and training array for neural net useage @arraybinseq is data
ready for input aray
$lherein=length($sequ);
#print "size of input vector is $lherein\n";
$bseq=$sequ;
&binconvert;
$binseq=$bseq;
###   @finarray is from the binconvert subroutine
@arraybinseq=@finarray;
$leninputarray=$#arraybinseq+1;
#print "$bseq\n";
#print "@arraybinseq\n";
#pack in 0's for remaining length between input array size and data us
Zero fill option selected
$ststhere=$leninputarray;@nfinhere=();
if ($centerfill eq "yes" and $inputarryaysize>($ststhere+2)){
  $halfdiffarsizehere=($inputarryaysize-$ststhere)/2;
  $partherone=int($halfdiffarsizehere);
  $parttwo=$inputarryaysize-$ststhere-$partherone;
  if ($zerofill eq "yes"){
     @beginpadarr=split("","0" x $partherone);
     @endpadch=split("","0" x $parttwo);
     @nfinhere=@beginpadarr;push @nfinhere,@arraybinseq;push @nfinhere,
     @arraybinseq=@nfinhere;
  }
 .....
 $lenfinal=$#arraybinseq+1;
#print "size of outputvector\:$lenfinal\n";
###END SUB prepdnaseq
}
```

I have to center to input data in my array since it is smaller than the array Need to make sure that it is smaller too.

FP: Doing Matrix Math

- For a two layer NN (the example here) you have three weights matrices and three neuron arrays, we will look at one first:
- [I]=the input
- [Ni] the array of values for the Input Neuron Array (lets say 100 elements, or 100x1), [rawNi] is the value before we do the SIGN functions.
- [Wi] the weights that multiply against the input data and are summed in the Input Neuron Array (has to be 300 x 100 or the matrix math won't work)
- We need the values of [Ni]:
  - Mathwise: See NN basics slides
  - PERL-Wise: We use nested 'for' loops and arrays.

FP: Matrix Multiply to get the Raw Response

- @nnlayer[]=[Ni]
- @arraybinseq[]=[I]
- $wi[][]=[Wi]
- Note I can recycle this segment just by changing the input array the weights matrix and where I put the raw output (@yipre)) (or by adding a dimension to my arrays and iterating)

```
@nnlayer=0; # zeroize my layer
$sizeinputvector=$#arraybinseq;
### Tell me how much data to expect
  if ($sizeinputvector>$inputarryaysize){
      #chomp at $inputarryaysize
  }
#### THIS IS THE MATRIX MAGIC:
for($j=0;$j<$sizeonelayernn;$j++){
  for($i=0;$i<$inputarryaysize;$i++){

@nnlayer[$j]=@nnlayer[$j]+$wi[$j][$i]*@arraybinseq[$i]*$fpf;
  }
}
@yipre=@nnlayer;
```

$fpf is called a multiplicative amplifier, which can be used to strengthen the inputs to the neuron (there are such things in real neurons: vitamin B anyone?)

FP: Raw Response to Actual using SIGN

- @yipre is raw output (i.e. Not just +1 or -1, or zeros for the numbed neurons)
- I need to apply my activation function to the raw output for each neuron to get its result:
- This one is a modified SIGN to account for the 0 fills:

```
### apply activation function one
for($j=0;$j<$sizeonelayernn;$j++){
   $temphere=@nnlayer[$j]+$thetaone;
   if($temphere==0){
## Zero fill handler and sign
      $reshere=0-1;
   }
   else{$reshere=int($temphere/abs($temphere));}
   @nnlayer[$j]=$reshere;
## if @nnlayer[$j] is +1 is am activated
}
```

$thetaone is called an additive amplifier, which can be used to strengthen or weaken the inputs to the neuron (like Caffeine or Alcohol :0)

NOTE: I need to store the RAW output to use in the Backward Pass

FP: Now for the Rest of the Layers

- Example uses one layer past input layer, then a single neuron for the output layer (a pseudo 2 layer NN)
- The Matrix Multiply Step and SIGN step are repeated for each layer.
- The last layer in my example only has one neuron, making this NN a 'Boolean Classification Network' (since I classify my output to just true or false)
- If I were doing something more complex, I could have many end nodes to get an array to match against a series of results (' Non-Boolean Classifier' ex: facial recognition)

FP: Rest of the Layers in PERL

```
#### MUltiply in Wj such that [wi]*[Wj]T

@nnlayerj=0;
for($k=0;$k<$sizetwolayernn;$k++){

# add in node results from prior layer times weights matrix
  for($j=0;$j<$sizeonelayernn;$j++){
    @nnlayerj[$k]=@nnlayerj[$k]+$wj[$k][$j]*@nnlayer[$j]*$fpf2;
  }
}
@jipre=@nnlayerj;
## apply activation function nnlayer2
for($k=0;$k<$sizetwolayernn;$k++){
  $temphere=@nnlayerj[$k]+$thetatwo;
  if($temphere==0){
     $reshere=0-1;
  }
  else{$reshere=int($temphere/abs($temphere));}
  @nnlayerj[$k]=$reshere;
}
```

Middle Layer ($thetatwo is additive input coefficient (additive amplifier) for Wj)

```
#multiply by in wx
$preimpulse=0;
for($k=0;$k<$sizetwolayernn;$k++){
   $preimpulse=$preimpulse+@nnlayerj[$k]*@wx[$k]*$fpf3;
}
$preimpulse=$preimpulse+$thetathree;
#apply activation function two
#print "forward pass pre-impluse pre theta sum\:$preimpulse\n";
if ($preimpulse==0){
### if I am numb to the end the answer is FALSE
  $actionimpulse=-1;
}
else{
  $actionimpulse=int($preimpulse/abs($preimpulse));
}
# print "action impulse result is $actionimpulse\n";
#### END FORWARD PASS
```

Last Layer (goes into 1 neuron for Output) $actionimpulse is the true (+1) or false (-1) result (Output Layer)

($thetathree is input coefficient (additive amplifier) for Wx)

FP (Training): Was I right?

- For a training run, I need to see if my answer ($actionimpulse) was correct
- If it was not correct, I need to do a Backwards Pass
- If correct, save the whole thing (all the weights) first

```
### THIS IS WHAT I FED THE FORWARD PASS:
### $binsequencehere is the binary form (+1/-1/0) of the input DNA test string
($resultexpected,$binsequencehere)=split(/\:/,$sequencelinehere);
....
@arraybinseq=split(/\,/,$binsequencehere);
### $resultexpected is what this sequence should be: Either True (+1) or False (0)
$intgerresp{} makes the 0 a -1
.....

 $correctactionresp=$intgerresp{$resultexpected};

## CHECKING MY RESPONSE!

    if ($correctactionresp==$actionimpulse){
       ### Just save weights
       $actioncorrectness="Correct";
       $corrbyrun[$kk]++;
     }
     else{
       $actioncorrectness="Incorrect\-$correctactionresp $resultexpected";
       &backwardpass;
       ## again save weights after training
     }
```

The Backwards Pass (BP)

- If the response is wrong (using negative reinforcement), need to do a Backwards Pass
- The Backwards Pass uses the raw neuron outputs of the Forward Pass, in a training function with training coefficients (TCs),to change my weights matrices
  - i.e.: Change to Weight item = Multiplying TC * Raw Output *Input*(expected-actual)+ Additive TC.
- This is why you have to save the raw neuron outputs before the action response function (i.e. Before applying SIGN)
- After altering each weight by the training function, I will need to normalize the matrices, so that each item in matrix is a %age (i.e. Magnitudes add up to 1)
  - Otherwise the forward pass will be way off next run (remember I deal with -1/+1/0, nothing bigger).

BP: A Bit on Training: Truth Data

- In order to train my NN I need data I know is true, and data I know is false.
  - True data is stored with an array value of +1
  - My truth data was downloaded from Sanger miRBASE (see [Sanger 2006])
- There needs to be MANY more fake/false answers then true ones
  - I generated them by random numbers:
  - Fake strings of DNA can be any length in a range (used the same rough range as true data + 20% on each side)
  - Length = random between (below real min size and above real max size of trues)
  - Each item in string is either 0=A 1=T 2=G 3=C, then use RND(3) or similar for each base

Alter the Output Weights [Wx]

```
###NOTE SET Beta and lambdas in configuration file
#### WORK BACKWARDS--from wx[k] to Wj[k][j] to Wi[j][i]
for($k=0;$k<$sizetwolayernn;$k++){
   $tempdeltawo=$betathree*($correctactionresp-$preimpulse) *@jipre[$k]+$lambdathree;

 ### alter wx (output layer)
   @wx[$k]=@wx[$k]+$tempdeltawo;
}

### normalize wx[k]
$sumwx=0;
for($k=0;$k<$sizetwolayernn;$k++){
    $sumwx=$sumwx+abs(@wx[$k]);
}
for($k=0;$k<$sizetwolayernn;$k++){
  @wx[$k]=abs(@wx[$k]/$sumwx);
}
```

$preimpule=my raw responses

Training Function:
$betathree=Multiplying training coefficient
$lambdathree=Additive training coefficient
(how hard are we smacking the knuckles to alter behavior)
$tempdeltawo=the change to the element of [Wx]
@nnlayerj=inputs to this matrix (prior Layer)
@jipre[]=preimpulse values of layer J

Normalize and use only positive weights. Any weights matrix adds up to 1

Alter the Rest of the Layers [Wj]

- Same method, just repeated for each layer (middle layer here).

```
#####ALTER Wj[k][j]
for($k=0;$k<$sizetwolayernn;$k++){
  for($j=0;$j<$sizeonelayernn;$j++){
    $tempdeltaone=$betatwo*($correctactionresp-@jipre[$k])*@yipre[$j]+$lambdatwo;
    $wj[$k][$j]=$wj[$k][$j]+$tempdeltaone;
  }
}

### normalize Wj
$sumwzero=0;
for($k=0;$k<$sizetwolayernn;$k++){
     for($j=0;$j<$sizeonelayernn;$j++){
       $sumwzero=$sumwzero+abs($wj[$k][$j]);
     }
}
for($k=0;$k<$sizetwolayernn;$k++){
     for($j=0;$j<$sizeonelayernn;$j++){
      $wj[$k][$j]=abs($wj[$k][$j]/$sumwzero);
     }
}
```

FYI..in the very next version I just use a 3 layer matrix and iterate this instead of copying it.

Alter the Rest of the Layers [Wi]

- Same method, just repeated for each layer. (Input Layer here)

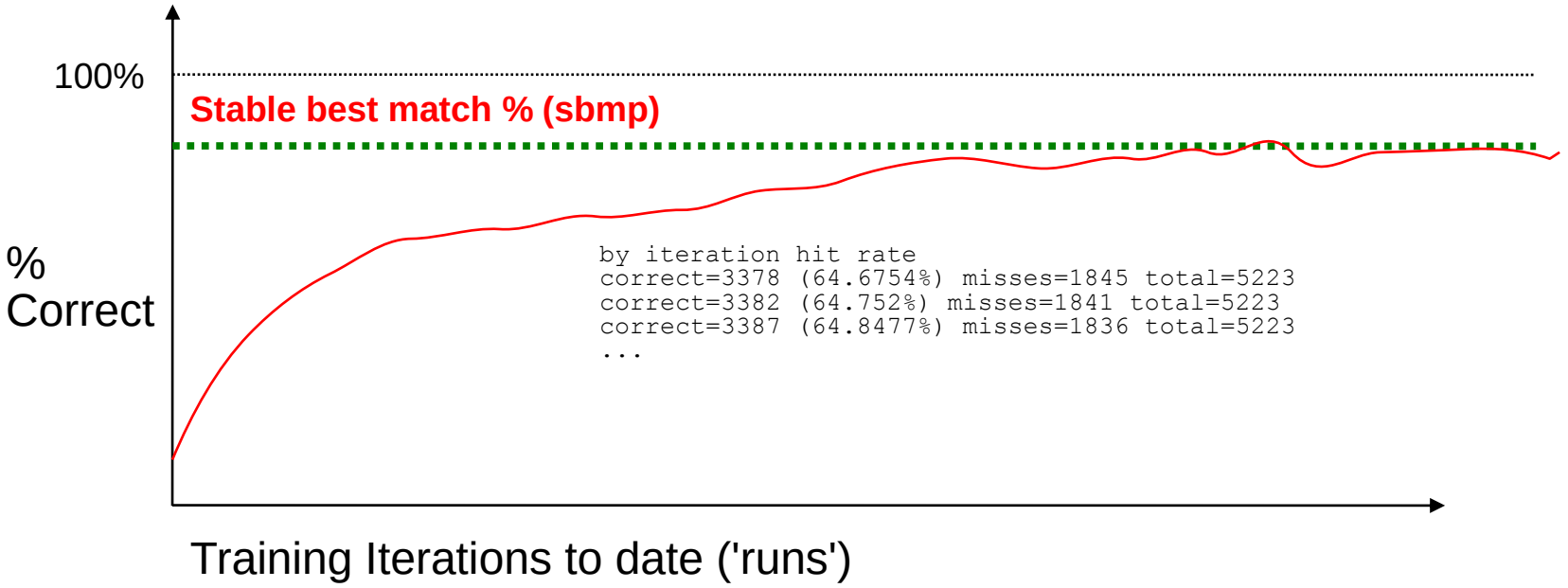
```
####alter wi
for($j=0;$j<$sizeonelayernn;$j++){
      for($i=0;$i<$inputarryaysize;$i++){
        $tempdeltaone=$betaone*($correctactionresp-@yipre[$j])*@arraybinseq[$i]+$lambdaone;
        $wi[$j][$i]=$wi[$j][$i]+$tempdeltaone;
      }
}


### normalize Wi
$sumwzero=0;
for($j=0;$j<$sizeonelayernn;$j++){
      for($i=0;$i<$inputarryaysize;$i++){
        $sumwzero=$sumwzero+abs($wi[$j][$i]);
      }
}
for($j=0;$j<$sizeonelayernn;$j++){
      for($i=0;$i<$inputarryaysize;$i++){
       $wi[$j][$i]=abs($wi[$j][$i]/$sumwzero);
      }
}
```

Rinse and Repeat

- Next, repeat forward pass-backward pass for every true and false test in your training data.
- Recommend a random shuffle of complete set each training iteration (one run through all trues and falses)
  - This avoids the danger of ordering (i.e. Go all the way +1, then all the way -1...leads to instability or you can manually mix them too).
- Tabulate statistics for success in each training iteration, I.e. Percentage of correct forward passes vs. incorrect forward passes.

Rinse and Repeat: How Do I Know What I Am Doing Is Right?

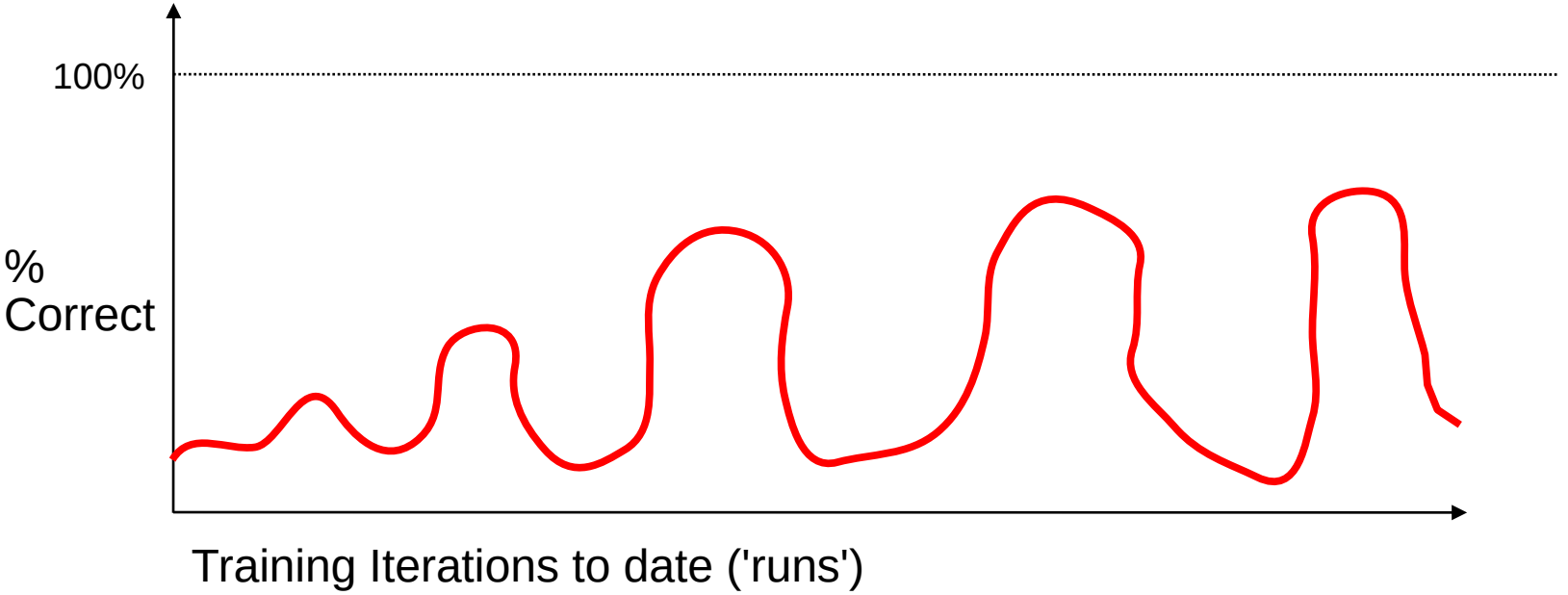
- After a large number of training iterations, the success level % should level off.



Stable best match % (sbmp) = the best my neural network can match the pattern in the data
--> will be anywhere less than 100% unless you have really easy data!
NN Error Rate (NER) = 100% - sbmp
i.e. If I use the NN against a 1000 new real items, I will be wrong at least NER * 1000 times.

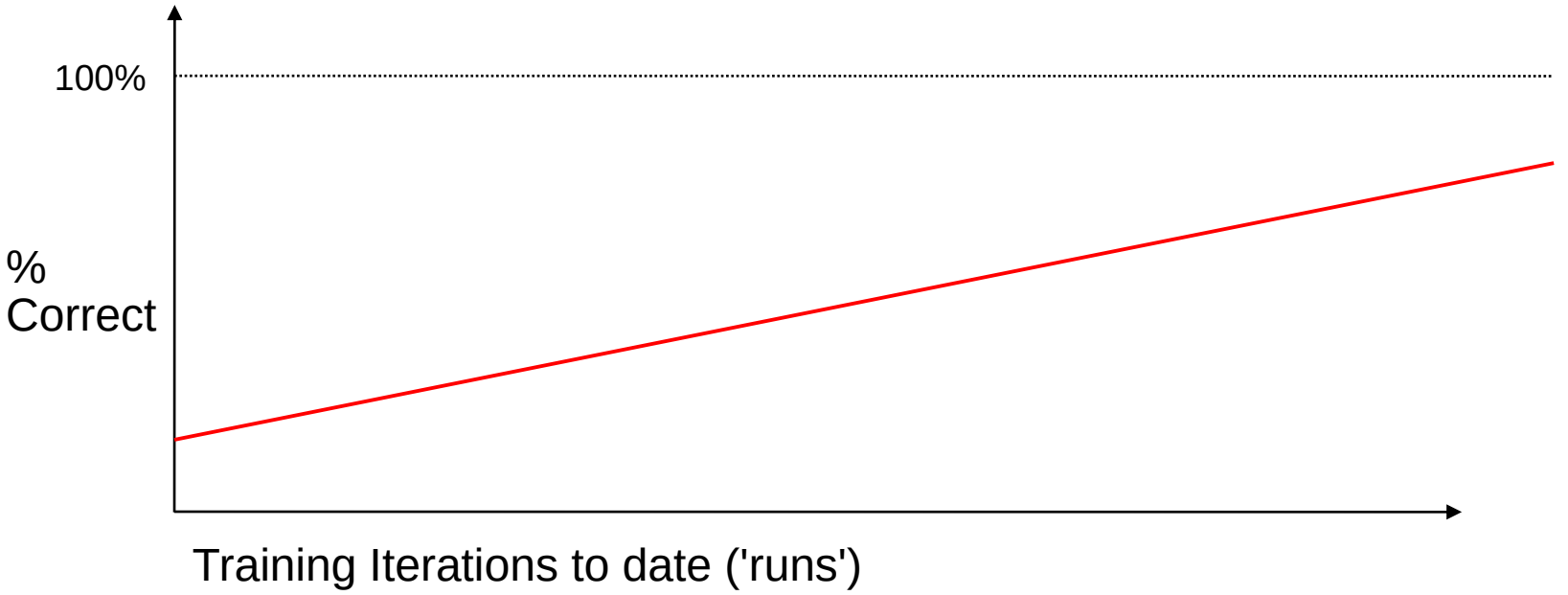
Rinse and Repeat: Instability

- If it does not stabilize:
  - Lower your training coefficients! You are hitting the NN too hard (and its knuckles are bleeding)
    - Beware using multiplicative coefficients unless you really know what the NN is doing, i.e. Start with those at 1, start with the additives very small (i.e. < 1/ (1000*{count of elements in weights matrix to be altered})



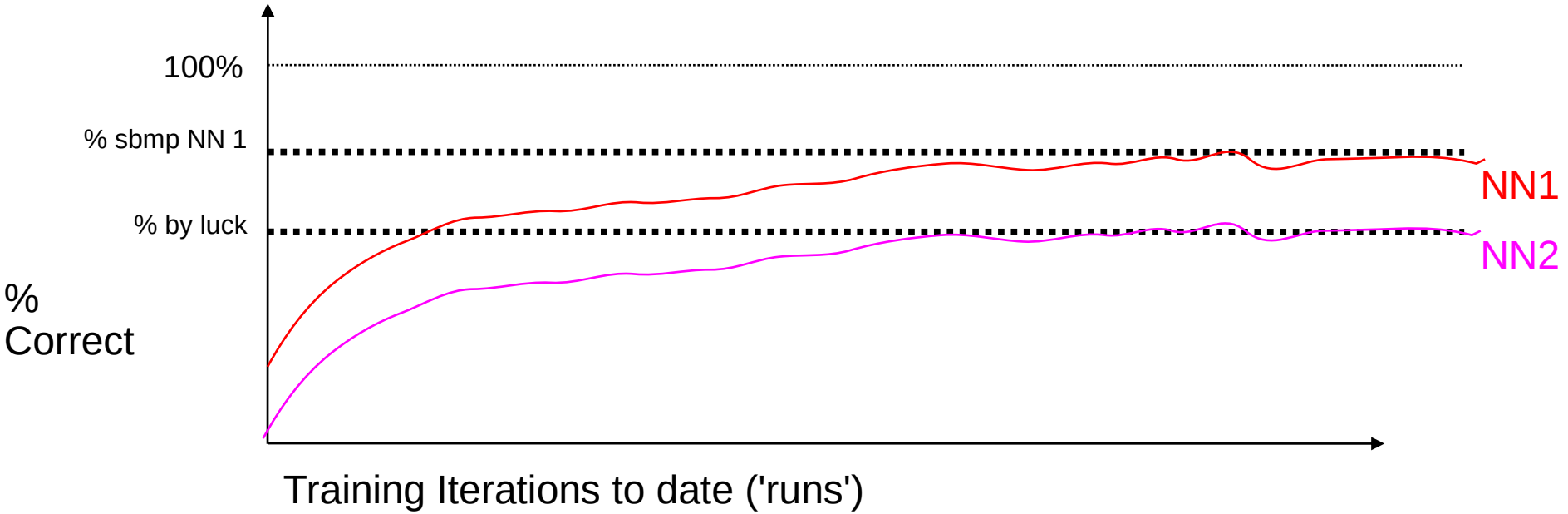
Rinse and Repeat: Converges too Slow

- If it does not converge after several thousand iterations but is still improving:
  - Run more iterations or
  - Increase training coefficients a VERY SMALL AMOUNT then continue runs



Rinse and Repeat: Converges at a Low Success Rate

- If the pattern is weak or nearly non-existent, the converged NN will still have a large error rate
  - If the NN converges at the % that matches (or is less than) the maximum of (% of trues/total and % falses/total) in the training set (i.e. % by luck), then there is no pattern this NN can find using the training data (like NN2 below)
  - If it is like NN1 below, (better than % by luck, but below the desired level), it is a weak pattern for this NN



Rinse and Repeat: Fixing a Low Success Rate

- Many strategies can help with a low rate, assuming there is a pattern in the overall data to be found:
  - Increase the training set size, i.e. Add in more trues and falses
  - Increase the number of elements in the Neuron Layers (i.e. make a bigger [Ni] or [Nj])
  - As a last resort add in another layer (i.e. add [Nk])
  - NOTE: every increases time for each run; adding another layer = exponential increase

Rinse and Repeat: Time frames

- 1 layer: 900 node single layer NN, on Sparc 10, Solaris, w/512 MB Ram, 989 Reals, 1523 fakes, 10 iterations:
  - Start Time=Mon Oct 2 15:04:10 2006
  - End time=Sat Oct 7 21:23:15 2006
- 2 layer: Size:300 x 1200 x 800, AMD Dual Core x 1 GHz, 2 GB RAM, SuSE, 989 Reals, 1623 fakes, 10 iterations
  - Start Time=Mon Jun 11 19:48:50 2007
  - End time=Thu Jun 14 01:33:32 2007

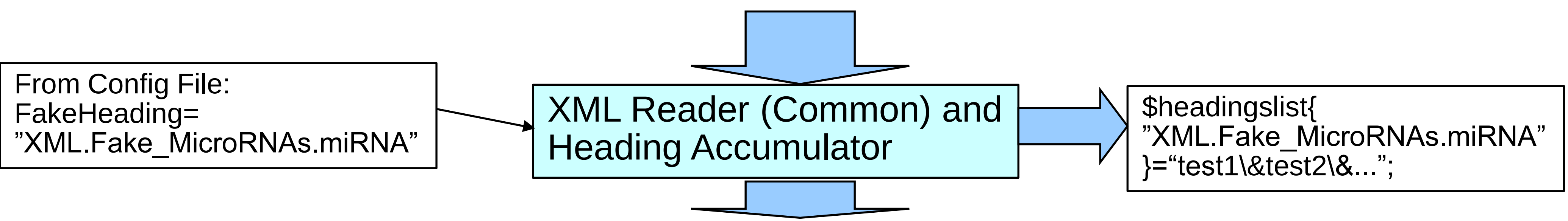
XML and Data Storage 1

- Need to store the weights data for each layer* (*most important store!)
- Need common storage method for Truth Data
- Need to store training statistics (how did I make my array)
- Need a log file
- Need to make and store configurations data (i.e. Start-up data for the NN)
- Your NN Core routines (could use a package here) need to be the same in the training and using programs
- Need a data puller to get real data for use
- Need scripts to run training and usage

XML and Data storage 2

- I use XML read into a string delimited by '.' (names by a '&')in a hash like so:

```
From 'falses.xml':
<?xml version="1.0" standalone="yes" ?>
<!--  Created Wed Feb 15 20:34:27 2006   -->
<Fake_MicroRNAs>
<miRNA name="test1">
<segment_data>aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa</segment_data>
</miRNA>
....
</Fake_MicroRNAs>
```

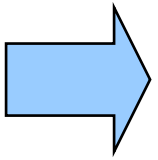



```
$xmldatitem{“XML.Fake_MicroRNAs.miRNA&test1&.segment_data"}=
”aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”
$listofheadings.=“XML.Fake_MicroRNAs.miRNA&test1&.segment_data";
```

XML and Data Storage 3

- The same method is used to save data like weights
- Iterate though the column size of each Matrix for each row and accumulate in a string (<row> tags)
- Then Iterate by row (do both in a nested loop):

```
print (ORF "<WiT>\n");
 for($j=0;$j<$sizeonelayernn;$j++){
  $valheren=$wi[$j][0];
  print (ORF "<row name\=\"$j\">");
  $linestringhere=$valheren;
  for($i=1;$i<$inputarryaysize;$i++){
   $valheren=$wi[$j][$i];
   $linestringhere.="\,$valheren";
  }
  print (ORF "$linestringhere<\/row>\n");
 }
 print (ORF "<\/WiT>\n");
```



```
<?xml version="1.0" standalone="yes" ?>
<!--   Created on Mon Jun 11 19:48:50 2007  -->
<!--  input file real realnewm.xml input fake fakesuperrand.xml --
>
<!-- beta1:0.000000001 beta2:0.000000001 lambda1:0 lambda2:0
shuffles:1 training iterations:10   -->
<NN_weights>
<Matrix_Sizes>{many more entires}</Matrix_Sizes>
<WiT>
<row name="0">3.65851129499352e-06,{many more
entires}</row>
{many more rows}
</WiT>
<WjT>
<row name="0">3.65851129499352e-06,{many more entires}</row>
{many more rows}
</WjT>
<WxT>
<array>0.00125000000000002,{many more entires}</array>
</WxT>
</NN_weights>
```

Note: for values in hashes, just iterate the headings for the hash after a split of its string into an heading_array, i.e. foreach $here(@heading_array){}

Compression

- Weights matrices are VERY large
  - 300 input x 600 node x 500 node in XML, uncompressed: 9.46 MB
- Some weights matrices are triangular matrices:
  - Can alter looping to improve speed and also store only fewer cells
- Easier Method: could use other compression tools (i.e. Gzip, tar, etc.)
  - Same 9.46MB weights file win zipped = 97KB

Using My Trained NN: A DNA Scanner to Feed Data

- Once I have a fully Trained NN (if ever :0), I can use it to scan real DNA to find candidate miRNA Hairpins that may be important
- I need to pull down real DNA sequences from EnSembl, or NCBI Blast.
- Then I need to build a subroutine to march down the DNA string in Input Array sized pieces (I need to set a 'Skip Rate'):
  - Skip Rate of 1 =Scan bases 100 to 400, then bases 101 to 401, etc.
  - Skip Rate of 10: Scan bases 100 to 400, then bases 111 to 411, then bases 121 to 421, etc.
- Then I run a Forward Pass against each piece using my saved weights data
- Then I save any thing that has a +1 result.

Using My Trained NN: Duplicating Results

- To do a quick confirm of my finds I will do the following: (to confirm unknown data)
  - Score the find against known miRNAs
    - If it already exists, then I note the location in the DNA strand stop working that find.
    - If it does not exist go to the next verification step
  - Run a hairpin-maker against it, and see if the hairpin matches characteristics for known miRNAs within a margin of error
    - If it does have a viable hairpin, NEED TO SAVE IT and ITS LOCATION...THESE ARE THE PREY I AM AFTER!
    - SEND TO RESEARCHERS AT Sanger, Wash U, UMSL, et al! Publish :0)
    - If not, store in discard pile for later examination

Error Rates Expected

- Error rates: if I scan a 120K base segment, and I have a 99.999% verified NN that uses an input array of 300 bases, and scan every set (skip rate = 1)
  - I have 120,000 – 300= 119,700 pre-NN candidates
  - False Returns at a minimum from the NN: 119,700 *(1-.99999) = 119,700 *1E-05 = 1
  - Here is the minimum errors for a NN trained to XX % for 1 Million Bases (a very likely case):

Bases in Scanned DNA	1.00E+006
Skip Rate (1=do every base)	1.00E+000
NN Trained %	Minimum False Returns
90.000	100000
95.000	50000
99.000	10000
99.900	999
99.990	100
99.999	10

Conclusion

- MiRNAs are extracted from hairpins, we can try to scan for more hairpins by training a Neural Network using known data sets
- Neural Networks emulate real neurons in living animals
  - Each neuron sums the weights * inputs for each connected input
- In PERL, nested loops can be used to perform the NN matrix functions
- XML can be used to store data, which can be pulled into, or stored from, PERL hashes
- Stable performance is a function of how well the NN can see the pattern, if any, in the truth data
- The better trained the NN, the lower the false return count.

Future Work

- Currently, my best NN s train @ ~85% using 2 layers for miRNA hairpins
  - Pattern is still weak
- Investigating bigger arrays, more layers
- Created a multi-purpose. Multi-layer (any # layers) trainer and scanner. Investigating a self expanding, self sizing NN also.
- Investigating other DNA features.
- Starting a pattern recognizer for plankton identification using same core.
- Investigating analogs of living NN to look at functions (i.e. Human eyes, Fish brains, etc.)

References and Future Reading

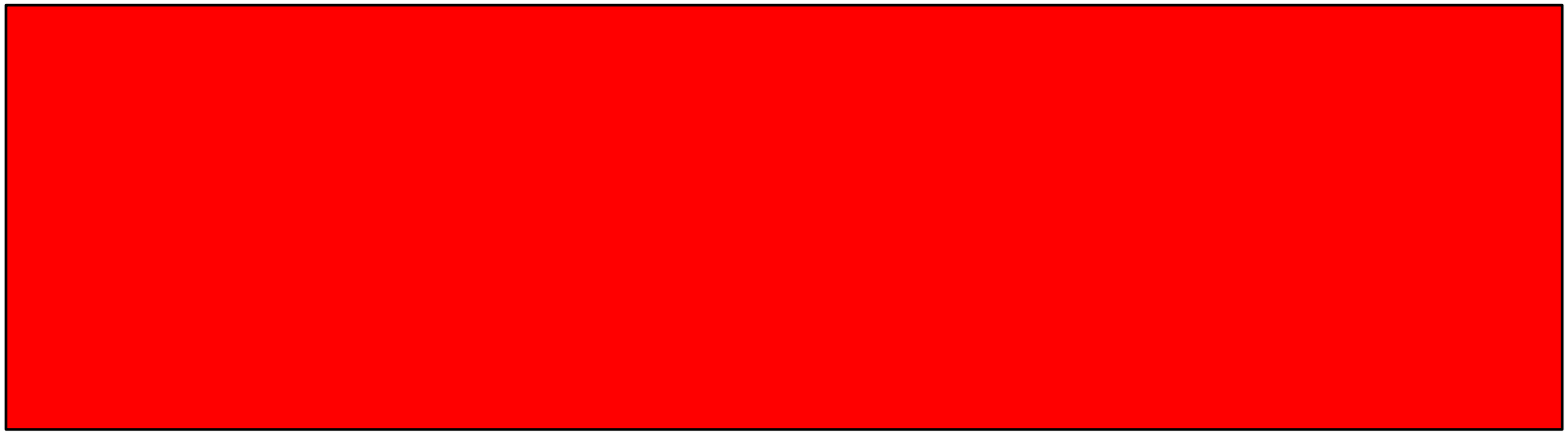
- **[Hayken,1994] Neural Networks: A Comprehensive Foundation, S. Hayken, Macmillian College Publishing,Inc., NY, 1994**
- **[B. Meyer 1996]** "Use of a Neural Network with Supervised Learning to Simulate the Feeding Response Behaviour of a Largemouth Bass " by Bryce L. Meyer, 15 April 1996. On-line at http://www.combat-fishing.com/NeuralNet/FILBFRT.htm
- **[Sanger, 2006] Sanger miRBase**:
  - Griffiths-Jones S, Grocock RJ, van Dongen S, Bateman A, Enright AJ. "miRBase: microRNA sequences, targets and gene nomenclature."NAR, 2006, 34, Database Issue, D140-D144
  - Griffiths-Jones S."The microRNA Registry." NAR, 2004, 32, Database Issue, D109-D111
  - http://microrna.sanger.ac.uk/sequences/search.shtml
- **[Ensebml, 2006] Ensembl:**
  - T. Hubbard, D. Andrews, M. Caccamo, G. Cameron, Y. Chen, M. Clamp, L. Clarke, G. Coates, T. Cox, F. Cunningham, V. Curwen, T. Cutts, T. Down, R. Durbin, X. M. Fernandez-Suarez, J. Gilbert, M. Hammond, J. Herrero, H. Hotz, K. Howe, V. Iyer, K. Jekosch, A. Kahari, A. Kasprzyk, D. Keefe, S. Keenan, F. Kokocinsci, D. London, I. Longden, G. McVicker, C. Melsopp, P. Meidl, S. Potter, G. Proctor, M. Rae, D. Rios, M. Schuster, S. Searle, J. Severin, G. Slater, D. Smedley, J. Smith, W. Spooner, A. Stabenau, J. Stalker, R. Storey, S. Trevanion, A. Ureta-Vidal, J. Vogel, S. White, C. Woodwark and E. Birney Ensembl 2005 Nucleic Acids Res. 2005 Jan 1;33 Database issue:D447-D453. doi:10.1093/nar/gki138
  - http://www.ensembl.org/Homo_sapiens/contigview?chr=11&vc_start=57163247&vc_end=57167335
- Legendre M, Lambert A, Gautheret D, 2005. "Profile-based detection of microRNA precursors in animal genomes", *Bioinformatics* 2005 21(7):841-845.
- Lu J, Getz G, Miska EA, Alvarez-Saavedra E, Lamb J, Peck D, Sweet-Cordero A, Ebert BL, Mak RH, Ferrando AA, Downing JR, Jacks T, Horvitz HR, Golub TR."MicroRNA expression profiles classify human cancers.", *Nature*, 2005 Jun 9;435(7043):834-8.
- Brown J.R., Sanseau P., 2005. "A computational view of microRNAs and their targets" Drug Discovery Today, Volume 10, Number 8, April 2005.
- Ambion,"microRNAs: Processing",online at: http://www.ambion.com/techlib/resources/miRNA/mirna_pro.html ©Copyright 2006 Ambion, Inc.

QUESTIONS?

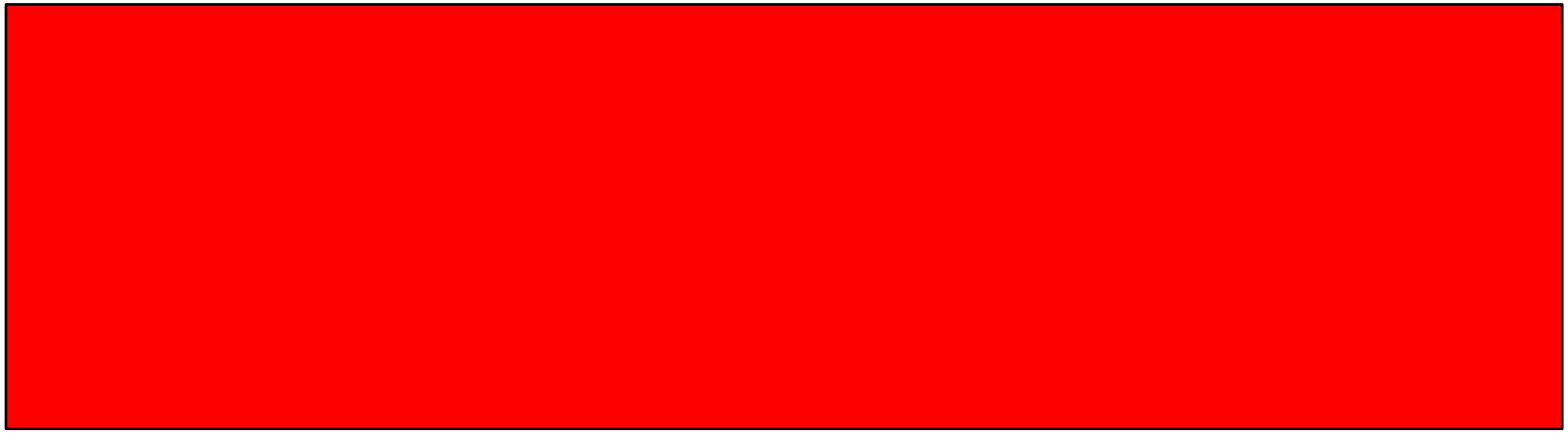
QUESTIONS? Hit my emails Or ask at next SLUUG meeting.

Thank You for listening and Good Luck on your own expeditions!

BACKUPS



BACKUPS



Before Exploring in the Unknown

- To make sure I didn't mess up:
  - Run the DNA Scanner and trained NN on areas of DNA known to contain miRNA precursors
  - Download the regions and put into my XML format using my data grabber (use NCBI Blast or Ensembl)
  - Did I find the known segments for the known miRNAs?
  - If so, then the hunt is on!
  - For other NN uses, you should use a second set of data you are certain of, to really prove your NN works.

Exploring: The Hunt, 1

- Pick a region of DNA ahead (in mRNA processing order) ahead of known disease gene locations, or begin a blind scan of the unknown sections of each chromosome.
- Use a data puller to grab a segment (say 0.5 Million Bases +/-)

Exploring: The Hunt, 2

- Set a Skip Rate for as low as your processor can do in a realistic time period
- Expect a week long run for 500K bases, skip rate of 1, on a dual core AMD, 2GB RAM

Why am I doing this?

- An outgrowth of graduate work from my two grad degrees. (Started the base N.N. core in '96 (see [B. Meyer 1996]), started using N.N.s for miRNAs in '06)
- Good excuse to use Neural Networks which provide insight to how a lot of nerve biology works.
- Takes advantage of Internet available genetic resources
- Server horsepower is now cheap.
  - Started on 80386 Windows, then to Sun Solaris, then to SuSE
- I may actually find a cause/cure for a disease.
  - You might find a cure also!

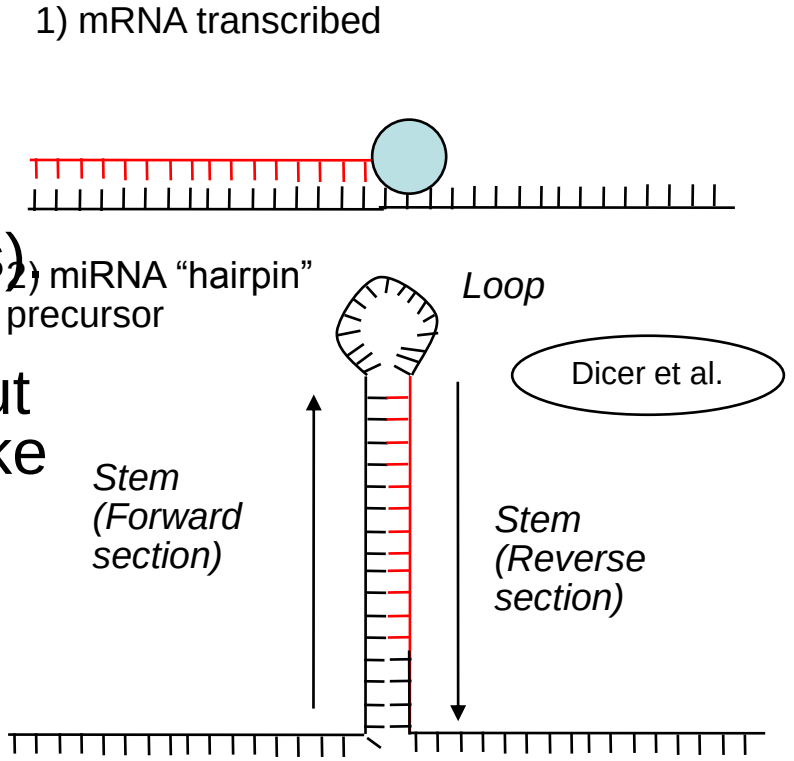
Problem: Why are miRNAs Important?

- As siRNAs (small interfering RNAs): Interacting w/ proteins, binding sites, mRNA translation.
- Associated with Cancer Causing Genes (Oncogenes): such as Leukemia and Breast Cancer.
- More uses found as time progresses

[Lu J, Getz G, Miska EA et al. 2005.]

Problem: How are miRNAs Processed?

- Many Micro-RNAs are components of an imperfect hairpin loop in mRNA
  - mRNA is transcribed from DNA
  - Sections can have areas that self compliment, forming a ‘hairpin’ loop (composed of stem and loop sections)
  - A section of the hairpin is chopped out (the precursor) and processed to make final microRNA (can be on forward, reverse, a combination, or from multiple hairpins)



For more detailed information see:
Ambion Website: http://www.ambion.com/techlib/resources/miRNA/mirna_pro.html

3) miRNA precursor processed again by enzymes for final miRNA

IE: Base Complimenting

- We may want to compliment the bases to make a mirror image of the DNA strand (or RNA Strand)
  - A pairs with T (or U), G with C
- Hashes are good for this:

```
#### This tells me if two bases are compliments
$cscore{"a"}{"t"}=1;$cscore{"t"}{"a"}=1;
$cscore{"a"}{"u"}=1;$cscore{"u"}{"a"}=1;
$cscore{"c"}{"g"}=1;$cscore{"g"}{"c"}=1;


### This tells me what the compliment of a base is.
$complbase{"c"}="g";$complbase{"g"}="c";
$complbase{"a"}="u";
$complbase{"t"}="a";$complbase{"u"}="a";
```

0.125	0.125	0.125	0.125	Wi
0.125	0.125	0.125	0.125	

-1	1	1	-1		
-1.240	1.510	1.510	-1.240	-0.375	dWi
-1.240	1.510	1.510	-1.240	-0.375	
1.24	1.51	1.51	1.24		
1.24	1.51	1.51	1.24		11

0.11	0.14	0.14	0.11	Wi'
0.11	0.14	0.14	0.11	

0.11	0.14	0.14	0.11
0.11	0.14	0.14	0.11

```
$tdwo1=($correctactionresp-$preimpulse);
$tempdeltawo=$betathree*$tdwo1*@nnlayerj[$k]+$lambdathree;
@wx[$k]=@wx[$k]+$tempdeltawo;


$tempdnjk=$correctactionresp-
@yipre[$k];
$tempdeltaone=$betatwo*$tempdnjk*@yipre[$j]+$lambdatwo
;
$wj[$k][$j]=$wj[$k][$j]+$tempdeltaone;


$tempdne=$correctactionresp-@yipre[$j];
$tempdeltaone=$betaone*$tempdne*@arraybinseq[$i]+$lambdaone;
$wi[$j][$i]=$wi[$j][$i]+$tempdeltaone;
```