Understanding neural network models of language: A tutorial in R

Franklin Chang – University of Liverpool

Artificial neural networks

- Simulate computational properties of brain neurons (Rumelhart, McClelland, & the PDP Research Group, 1995)
  - Neurons 神経細胞 (firing rate = activation)
  - Connections with other neurons (strength of relationship = weights)
- Learning implicit language knowledge
  - Phonology (Elman & McClelland, 1988 TRACE)
  - Morphology (Plunkett & Juola, 1999)
  - Lexical Processing (Plaut et al., 1996)
  - Speech errors (Dell, 1986)
  - Syntax (Elman, 1990)
  - Sentence Production (Chang et al. 2006)
- Deep Learning (Hinton, 2007)
  - Youtube transcription (50% Gaussian Mixture Model)
    - Deep neural network improves performance by 20%

Mapping between meaning and words

- Language-specific mappings
  - American English: *tea* = DRINK+LEAF
  - Cantonese Chinese: *cha* = DRINK or *cha* = MEAL (*yum cha* 飲茶)
  - British English:
    - *tea* = DRINK+LEAF ("Do you drink tea?")
    - *tea* = MEAL+LATEAFTERNOON ("We often eat beans on toast for tea")
- To learn these languages, you need to link meaning and word forms
  - Meaning: DRINK, LEAF -> *tea*

Mapping represent logical functions (論理関数)

- Language mappings can be represented in terms of semantic feature inputs
  - DRINK (1=there is a drink, 2=no drink), LEAF (1=there are tea leaves, 0=no leaves)
  - tea (1 = say *tea*, 0 = don't say *tea*)

American (AND)

DRINK	LEAF	tea
1	1	1
1	0	0
0	1	0
0	0	0

Cantonese (OR)

EAT	LEAF	cha
1	1	1
1	0	1
0	1	1
0	0	0

British (XOR)

EAT	LEAF	tea
1	1	0
1	0	1
0	1	1
0	0	0

- Different logical function in each language.
  - American = AND function (論理積)
  - Cantonese = OR function (論理和)
  - British = Exclusive OR function XOR (排他的論理和)

Learning logical functions

- AND and OR functions are easier to learn than XOR.
- Regression: lm(tea ~ EAT + LEAF)
  - Predicted output in column P

AND

A	B	T	P
1	1	1	0.75
1	0	0	0.25
0	1	0	0.25
0	0	0	-0.25

OR

A	B	T	P
1	1	1	1.25
1	0	1	0.75
0	1	1	0.75
0	0	0	0.25

XOR

A	B	T	P
1	1	0	0.5
1	0	1	0.5
0	1	1	0.5
0	0	0	0.5

- AND and

OR functions are only off by 0.25, but XOR model does not learn anything.

Learning XOR functions

- If we add an interaction term, then the model can learn XOR
- Regression: lm(tea ~ EAT + LEAF + EAT:LEAF)

XOR

A	B	T	P
1	1	0	0.5
1	0	1	0.5
0	1	1	0.5
0	0	0	0.5

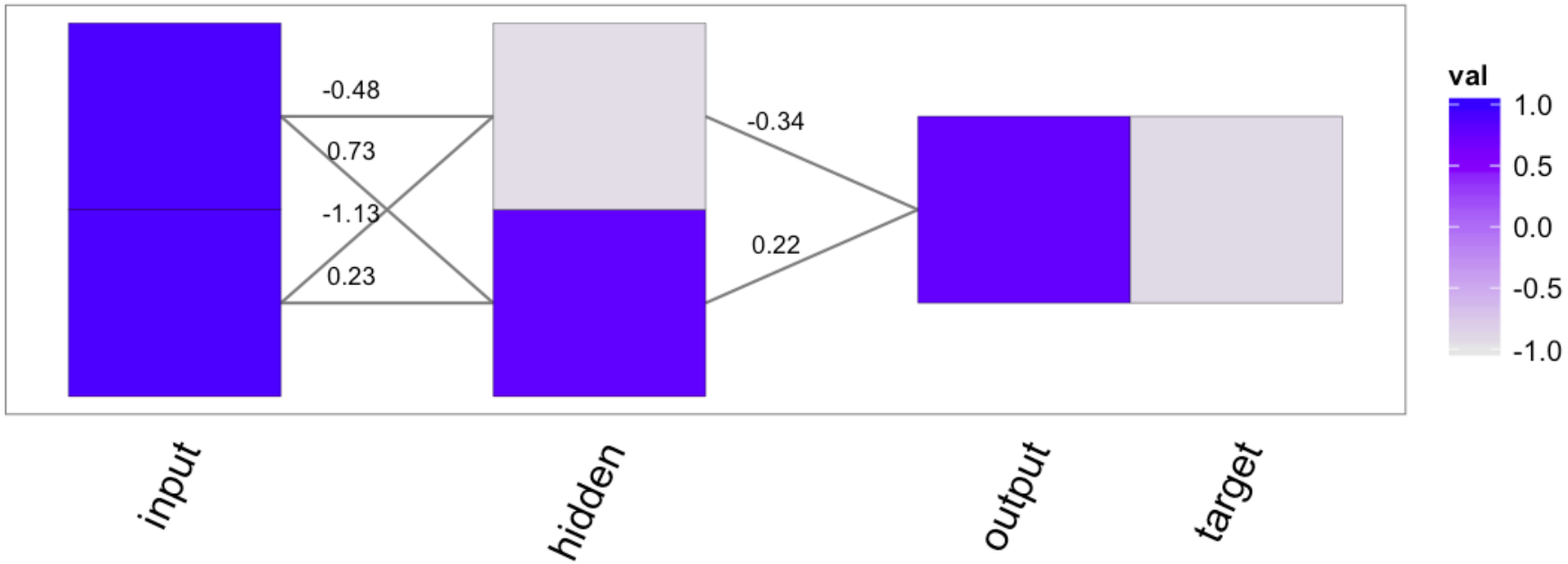
XOR (A*B)

A	B	T	P
1	1	0	0
1	0	1	1
0	1	1	1
0	0	0	0

- If we add interaction terms, then we can learn any function.
- Curse of dimensionality: If we add more features, we get too many interaction terms.
  - For c concepts, $2^c - 1$ interaction terms, e.g., 20 concepts = 1,048,575 interaction terms
- Can we learn these interaction terms?

Neural networks

- Similar to regression: Prediction
- Artificial neurons (units) encode input and output values [-1,1]
- Weights between neurons encode strength of links (betas in regression)
- Neurons are organized into layers (output layer ~ input layer)
- Beyond regression: Hidden layers can recode the input to learn mappings like XOR

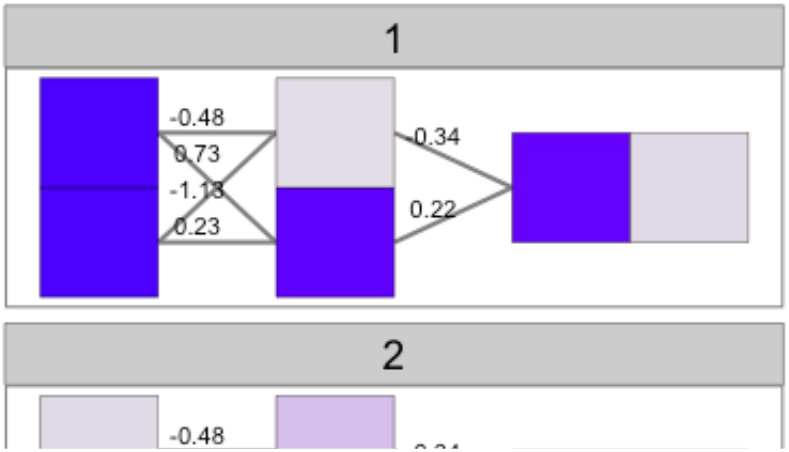


Learning Hidden Representations

- Back-propagation of error (Rumelhart, Hinton, & Williams, 1986)
  - Learns hidden representations
  - Forward pass (spreading activation)
  - Error = difference between target and output activation (residuals)
  - Backward pass (pass error back in the network to change weights)
- Vectorized/Matrix operations　（数学の線型代数　行列）
- $SSE = \sum_{i=1}^{n} (o_i - t_i)^2$
  - R, matlab, python do vectorized operations
  - SSE = sum((O - T)^2)

Matrices and networks

- Input Matrix
  - $I = \begin{bmatrix} 0.9 & 0.9 \\ 0.9 & -0.9 \\ -0.9 & 0.9 \\ -0.9 & -0.9 \end{bmatrix}$
- Target Matrix
  - $T = \begin{bmatrix} -0.9 \\ 0.9 \\ 0.9 \\ -0.9 \end{bmatrix}$
- Weight Matrix - initialized with random weights



Back-propagation Overview

- Forward Pass
  - Multiply inputs and weights（行列の積）
  - Apply activation function
- Backward Pass
  - Compute error (予測誤差)
  - Compute delta weight change matrix
    - Change weights (learning rate)
    - Add Momentum (運動量)
  - Back-propagate error to previous layer
- Repeat for each layer

Forward Pass (Input->Hidden)

- Spreading Activation
  - Add *bias* column of 1s to Input Matrix (intercept term in regression)
  - Dot product (Matrix Multiplication 行列の積) of input vector I_H with weight matrix W_H
    - 0.9 * 0.23 + 0.9 * 0.73 + 1 * 0.23 = 1.09

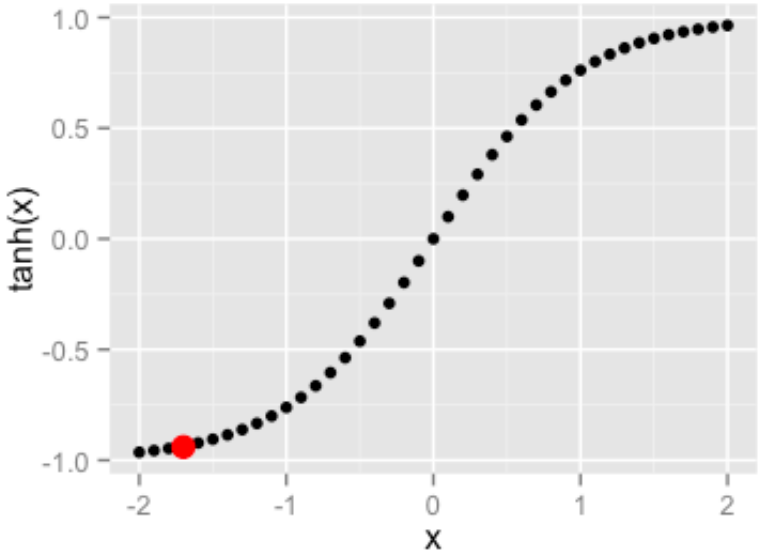
$$V_H = I_H \cdot W_H = \begin{bmatrix} 0.9 & 0.9 & 1 \\ 0.9 & -0.9 & 1 \\ -0.9 & 0.9 & 1 \\ -0.9 & -0.9 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.23 & -1.13 \\ 0.73 & -0.48 \\ 0.23 & -0.24 \end{bmatrix} = \begin{bmatrix} 1.09 & -1.7 \\ -0.22 & -0.83 \\ 0.68 & 0.34 \\ -0.63 & 1.21 \end{bmatrix}$$

- Dot product: AxB matrix · BxC matrix -> AxC matrix

Forward Pass (Input->Hidden)

- Apply activation function to netinput V_H -> output Y_H
  - Forces values to range of [-1,1]
  - Hidden layer can recode inputs

$$Y_H = \tanh(V_H) = \tanh\left(\begin{bmatrix} 1.09 & -1.7 \\ -0.22 & -0.83 \\ 0.68 & 0.34 \\ -0.63 & 1.21 \end{bmatrix}\right) = \begin{bmatrix} 0.8 & -0.94 \\ -0.22 & -0.68 \\ 0.59 & 0.33 \\ -0.56 & 0.84 \end{bmatrix}$$



Forward Pass (Hidden->Output)

- Spreading Activation: multiplying input vector I_O against the weight matrix W_O -> netinput V_O

$$V_O = I_O \cdot W_O = \begin{bmatrix} 0.8 & -0.94 & 1 \\ -0.22 & -0.68 & 1 \\ 0.59 & 0.33 & 1 \\ -0.56 & 0.84 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.22 \\ -0.34 \\ 0.54 \end{bmatrix} = \begin{bmatrix} 1.03 \\ 0.72 \\ 0.56 \\ 0.13 \end{bmatrix}$$

- Apply activation function

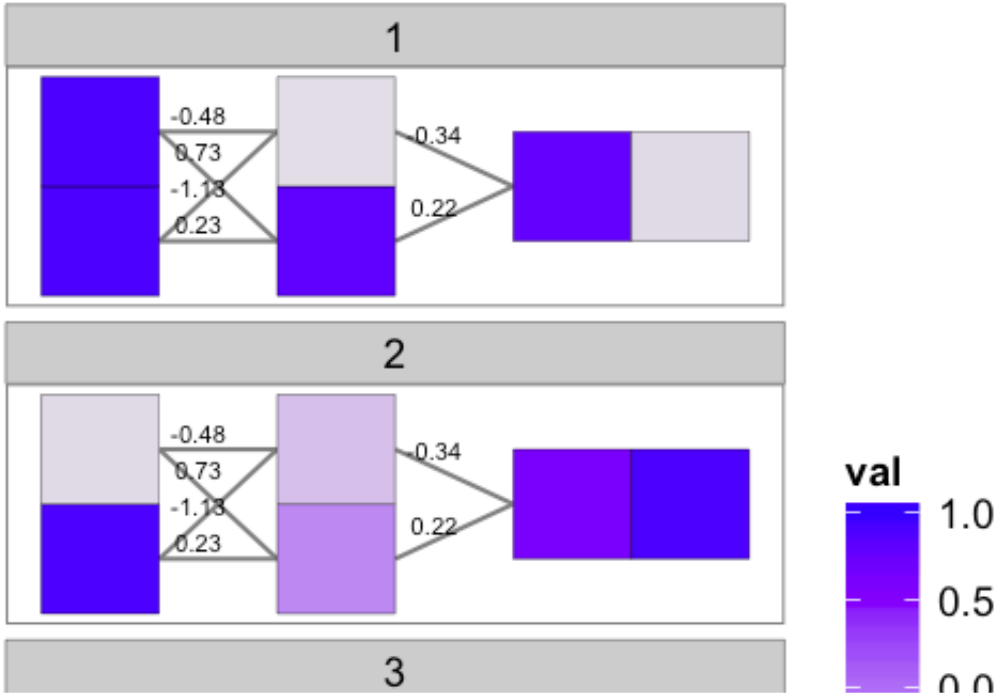
$$Y_O = \tanh(V_O) = \tanh(\begin{bmatrix} 1.03 \\ 0.72 \\ 0.56 \\ 0.13 \end{bmatrix}) = \begin{bmatrix} 0.77 \\ 0.62 \\ 0.51 \\ 0.13 \end{bmatrix}$$

Backward Pass

- Compute Error E_O (difference between output Y_O and target T)

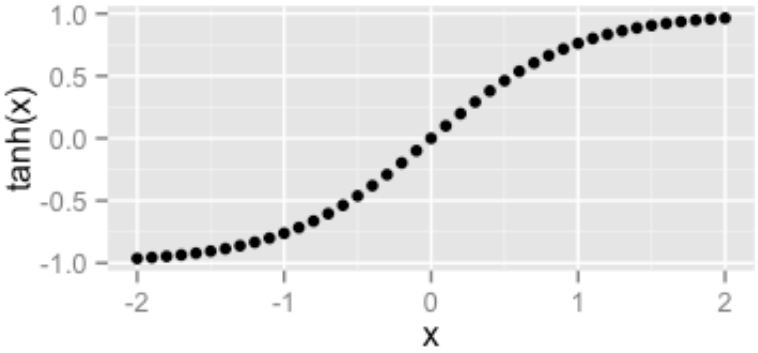
$$E_O = T - Y_O$$

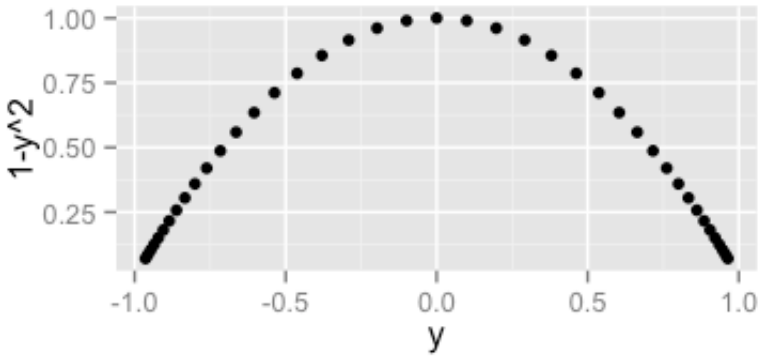
$$= \begin{bmatrix} -0.9 \\ 0.9 \\ 0.9 \\ -0.9 \end{bmatrix} - \begin{bmatrix} 0.77 \\ 0.62 \\ 0.51 \\ 0.13 \end{bmatrix} = \begin{bmatrix} -1.67 \\ 0.28 \\ 0.39 \\ -1.03 \end{bmatrix}$$



Changing weights based on error

- Derivative (導関数): how variables change in relation to change in other variables.
  - Acceleration is the derivative of speed
- How do we learn to brake when driving a new car?
  - Target would be to stop at some good distance from other cars
  - Error would be distance between your actual stopping distance and the target
  - Derivative: how changes in the car's speed changes in response to changes in force on the pedal
- How do we adjust the weights in the network?
  - tanh changes the netinput into output
  - Derivative D of tanh is $D_O = 1 - Y_O^2$





Adjusting the error by the derivative

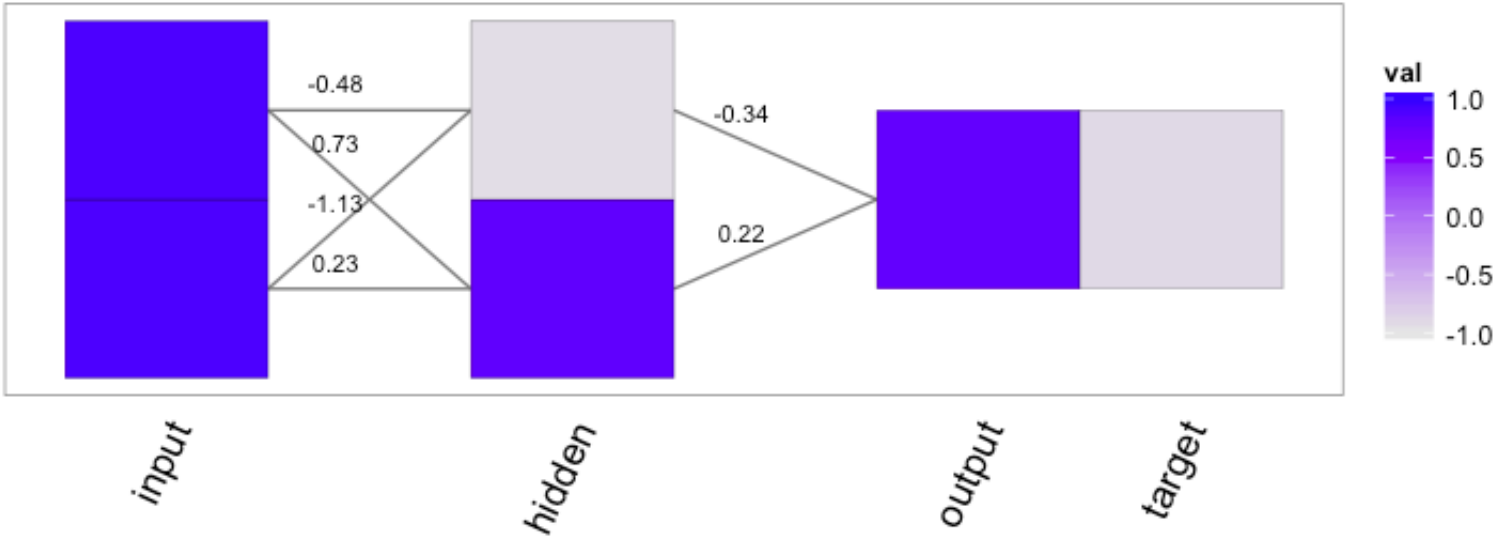
- Gradient δ_O is calculated by multiplying the error E_O by the output layer derivative D_O
  - element-wise matrix multiplication (Hadamard product $\circ$).

$$\delta_O = E_O \circ D_O = E_O \circ (1 - Y_O^2) = \begin{bmatrix} -1.67 \\ 0.28 \\ 0.39 \\ -1.03 \end{bmatrix} \circ \begin{bmatrix} 0.4 \\ 0.62 \\ 0.74 \\ 0.98 \end{bmatrix} = \begin{bmatrix} -0.67 \\ 0.18 \\ 0.29 \\ -1.01 \end{bmatrix}$$

How do we change the weights?

- delta weight matrix Δ_O is computed by calculating the dot product of the transposed input for the output layer I_O and the gradient matrix
  - If an input is activated and the output is wrong, then we blame that unit
  - Input is transposed to get the right shape delta weight matrix（行列の転置）

$$\Delta_O = I_O^T \cdot \delta_O = \begin{bmatrix} 0.8 & -0.22 & 0.59 & -0.56 \\ -0.94 & -0.68 & 0.33 & 0.84 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} -0.67 \\ 0.18 \\ 0.29 \\ -1.01 \end{bmatrix} = \begin{bmatrix} 0.16 \\ -0.24 \\ -1.21 \end{bmatrix} \quad (7)$$



Learning rate

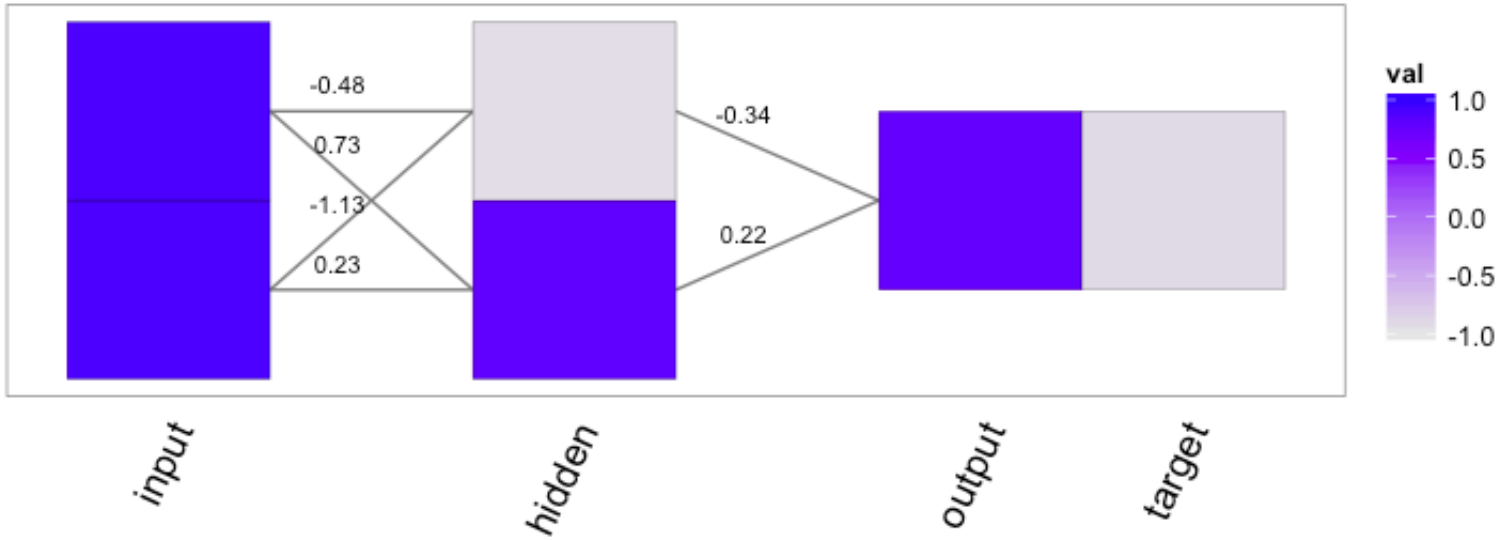
- Learning rate η allows us to adjust the speed that the model changes in response to this input
- Deltas Δ_O^t are then added to the weights W_O^{t-1} to update them to the new weights W_O^t

$$W_O^t = W_O^{t-1} + \eta\Delta_O^t = \begin{bmatrix} 0.22 \\ -0.34 \\ 0.54 \end{bmatrix} + 0.03 \begin{bmatrix} 0.16 \\ -0.24 \\ -1.21 \end{bmatrix} = \begin{bmatrix} 0.23 \\ -0.35 \\ 0.5 \end{bmatrix} \quad (8)$$

Back-propagation of Error

- Error at the output layer can be back-propagated back to the hidden layer (Rumelhart et al., 1986)
- Dot product of the gradient δ_O and the transpose output-hidden weights W_{t-1}^T

$$E_H = \delta_O \cdot W_{t-1}^T = \begin{bmatrix} -0.67 \\ 0.18 \\ 0.29 \\ -1.01 \end{bmatrix} \cdot \begin{bmatrix} 0.22 & -0.34 \end{bmatrix} = \begin{bmatrix} -0.15 & 0.23 \\ 0.04 & -0.06 \\ 0.07 & -0.1 \\ -0.23 & 0.35 \end{bmatrix}$$



Change input-hidden weights

- Same equations are used for the hidden layer

$$\delta_H = E_H \circ D_H = E_H \circ (1 - Y_H^2)$$

$$= \begin{bmatrix} -0.15 & 0.23 \\ 0.04 & -0.06 \\ 0.07 & -0.1 \\ -0.23 & 0.35 \end{bmatrix} \circ \begin{bmatrix} 0.36 & 0.12 \\ 0.95 & 0.54 \\ 0.65 & 0.89 \\ 0.69 & 0.3 \end{bmatrix} = \begin{bmatrix} -0.06 & 0.03 \\ 0.04 & -0.03 \\ 0.04 & -0.09 \\ -0.16 & 0.1 \end{bmatrix}$$

- Compute delta weight matrix

$$\Delta_H = I_H^T \cdot \delta_H$$

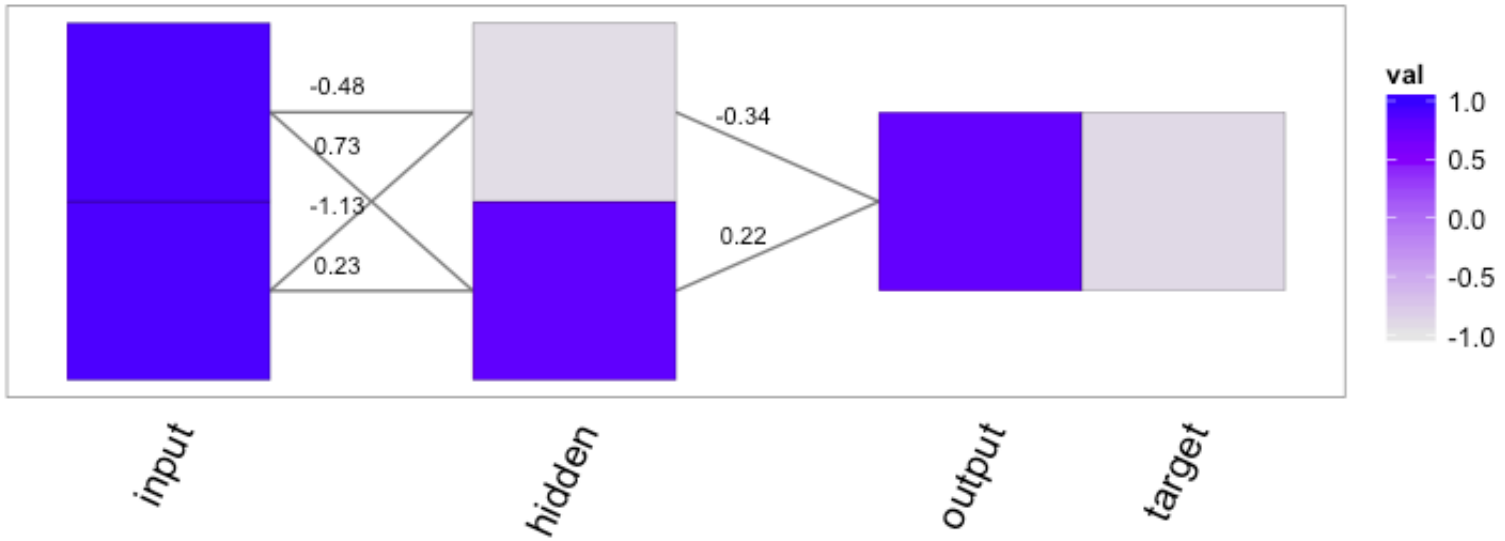
$$= \begin{bmatrix} 0.9 & 0.9 & -0.9 & -0.9 \\ 0.9 & -0.9 & 0.9 & -0.9 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} -0.06 & 0.03 \\ 0.04 & -0.03 \\ 0.04 & -0.09 \\ -0.16 & 0.1 \end{bmatrix} = \begin{bmatrix} 0.09 & -0.02 \\ 0.1 & -0.12 \\ -0.13 & 0.01 \end{bmatrix}$$

Change input-hidden weights

- Update weight using new delta weight change matrix

$$W_H^t = W_H^{t-1} + \eta \Delta_H^t$$

$$= \begin{bmatrix} 0.23 & -1.13 \\ 0.73 & -0.48 \\ 0.23 & -0.24 \end{bmatrix} + 0.03 \begin{bmatrix} 0.09 & -0.02 \\ 0.1 & -0.12 \\ -0.13 & 0.01 \end{bmatrix} = \begin{bmatrix} 0.23 & -1.14 \\ 0.73 & -0.49 \\ 0.23 & -0.24 \end{bmatrix}$$



Cost functions

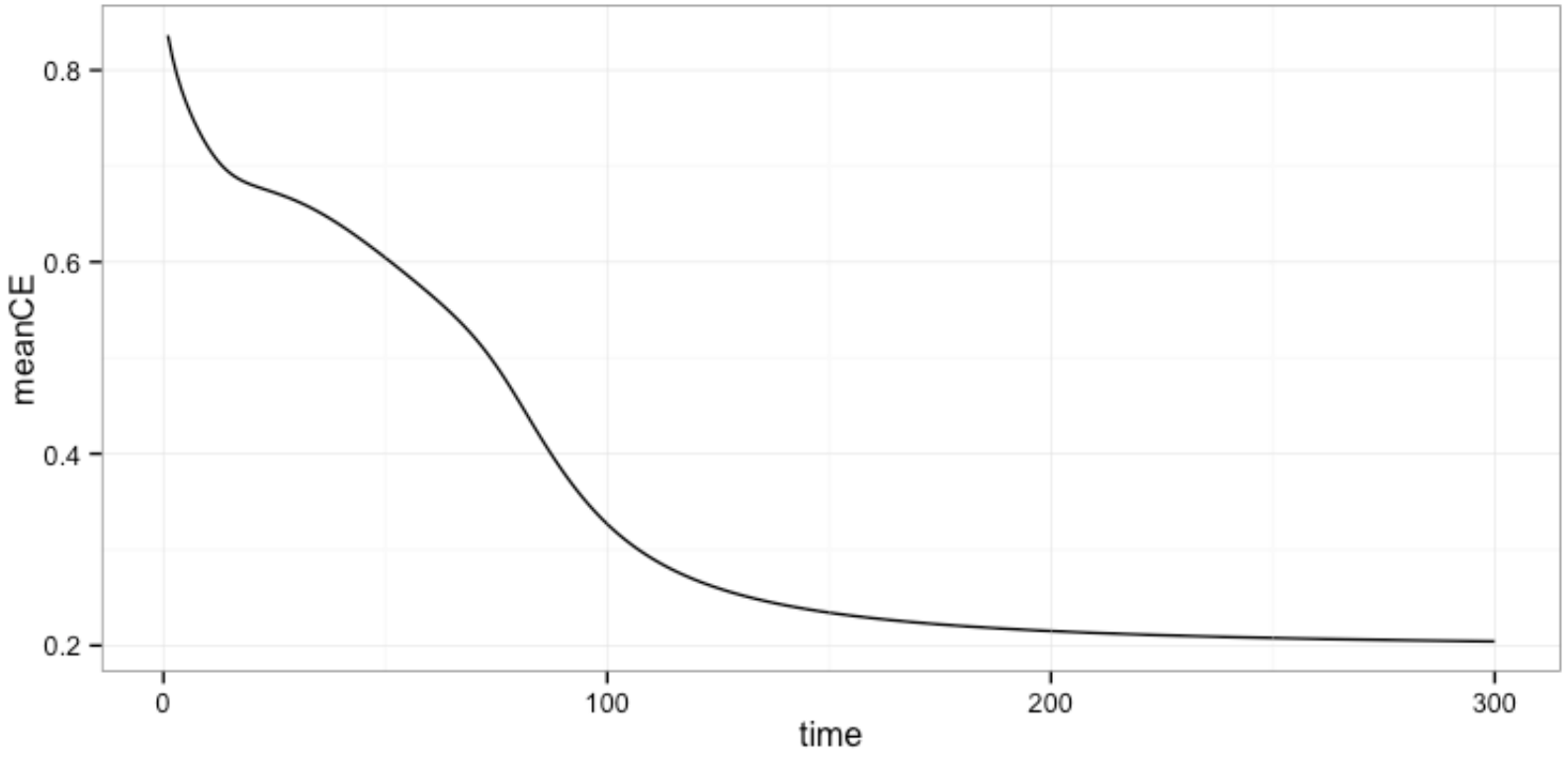
- Cost: Function that network is trying to minimize
  - Regression uses a cost function of sum of squares error $\sum(y-t)^2$, where it is trying to minimize residuals between the regression line and all of the target points in the data set
- Cross-entropy loss function L for back-propagation with tanh activation function

$$L = -T * log(Y_O) - (1-T) * log(1-Y_O)$$

$$= -\begin{bmatrix} -0.9 \\ 0.9 \\ 0.9 \\ -0.9 \end{bmatrix} * \log(\begin{bmatrix} 0.77 \\ 0.62 \\ 0.51 \\ 0.13 \end{bmatrix}) - \begin{bmatrix} 1.9 \\ 0.1 \\ 0.1 \\ 1.9 \end{bmatrix} * \log(\begin{bmatrix} 0.23 \\ 0.38 \\ 0.49 \\ 0.87 \end{bmatrix}) = \begin{bmatrix} 2.08 \\ 0.29 \\ 0.34 \\ 0.81 \end{bmatrix}$$

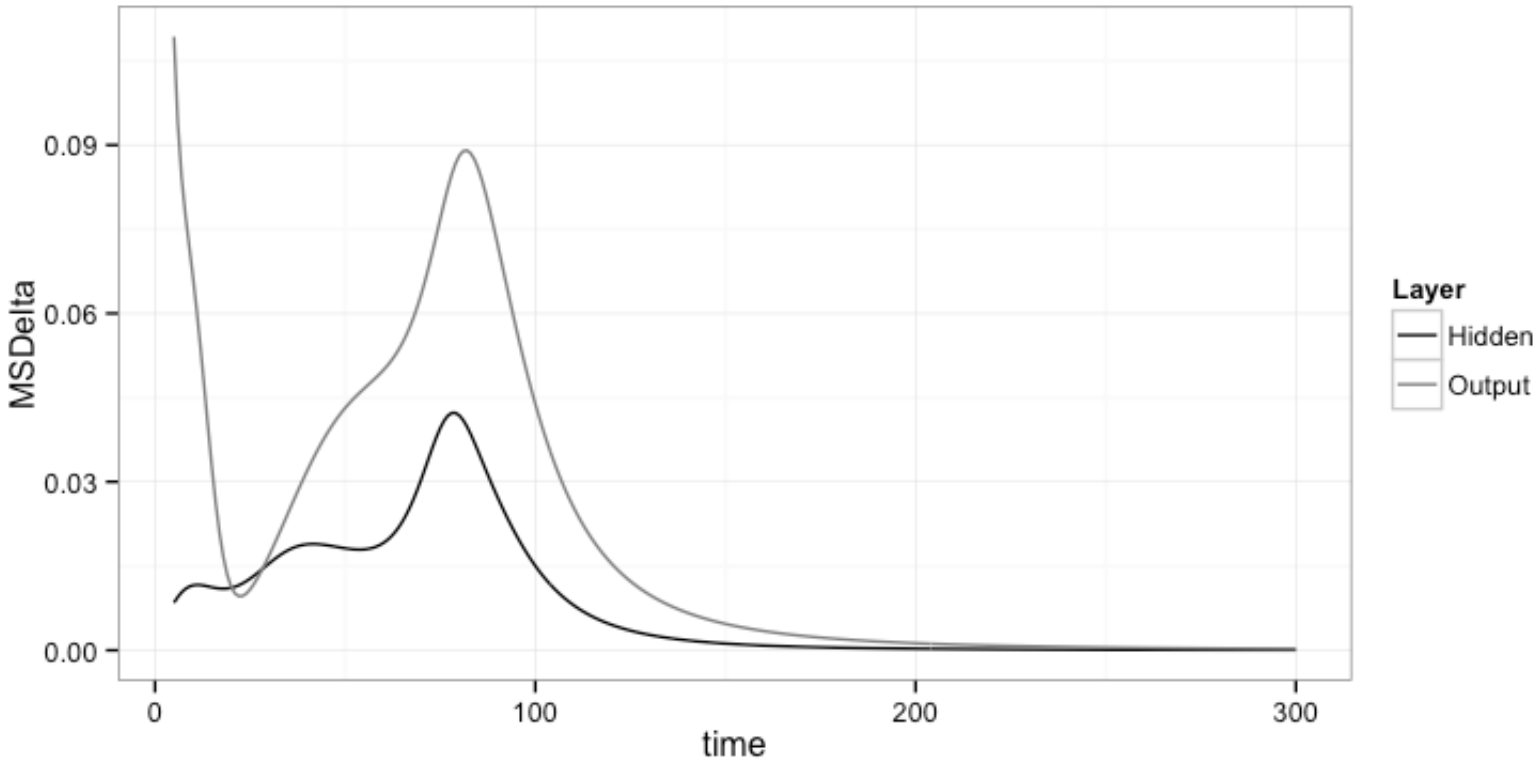
Error during training

- MeanCE: mean cross-entropy loss over all patterns
  - loss reduces as the model becomes better at predicting the correct output



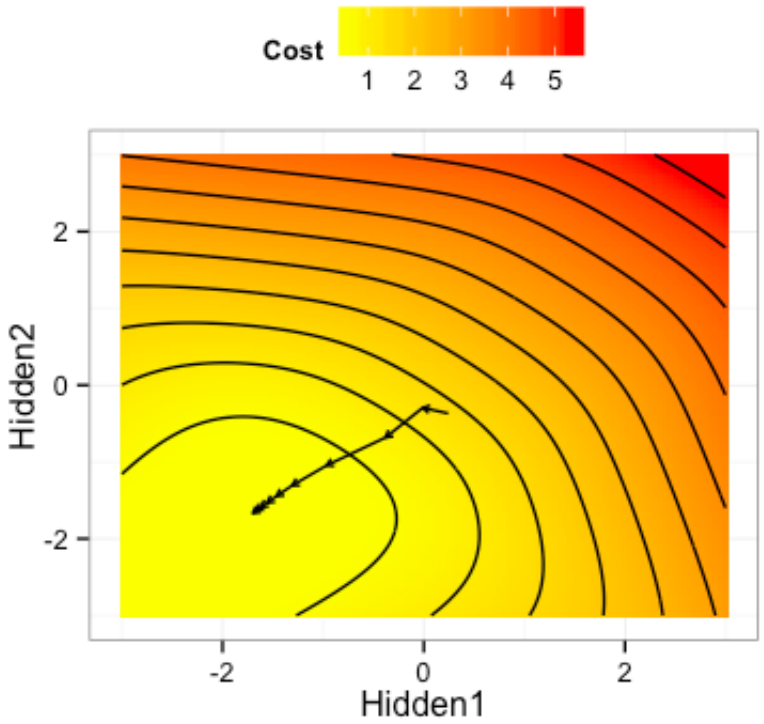
Plotting weight changes during training

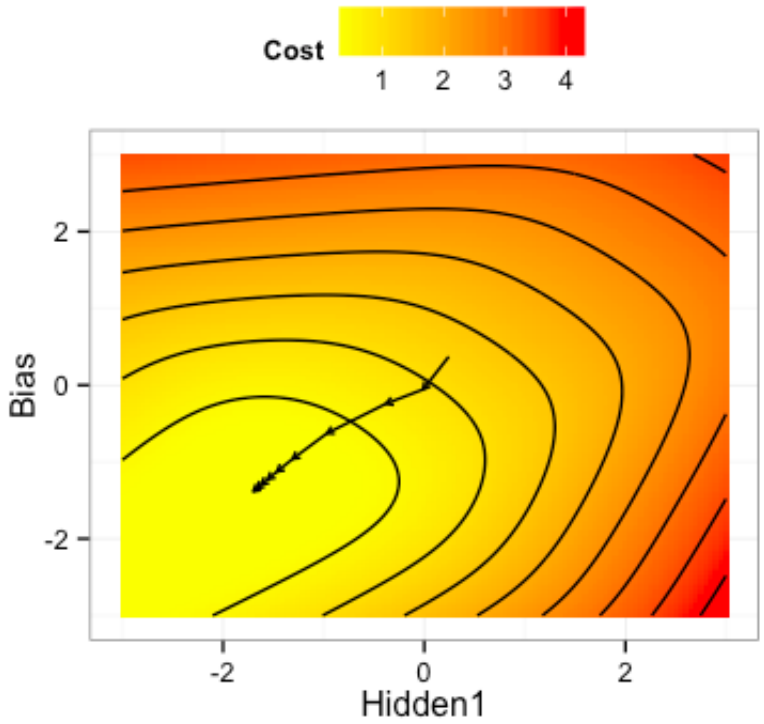
- Cost function do not provide much information about how the network is learning
- MSDelta = mean sum of squares of delta weight change matrix Δ_O



Error space

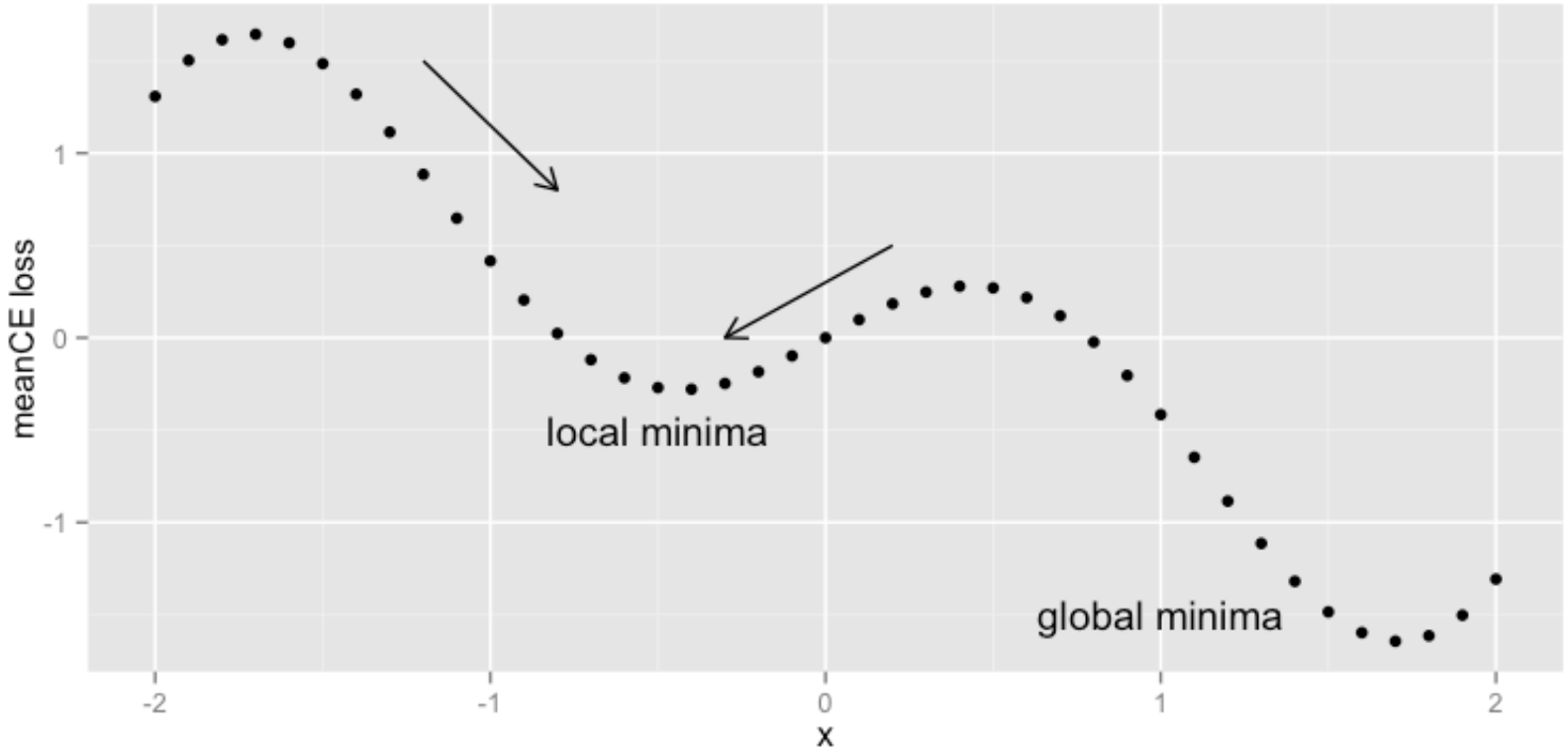
- To understand the model, it is useful to track the model's weights as it learns
  - Output layer has three weights (hidden1, hidden2, bias)
  - Simulate values [-3,3] for hidden1 and hidden2 and see how cost function changes
    - Background colour = meanCE (red = hills, yellow = valleys)
  - Model's path is shown by black lines (random initial weights)





Momentum descent

- Steepest descent: Move down in the weight space in the direction that reduces cost function
  - Sometimes traps you in local minima, rather than the global minima



Momentum descent

- move in the same direction in weight space as the last weight change
  - ball will continue traveling in the same direction due to momentum (運動量)
- Deltas from the previous timestep Δ_H^t are multiplied by the momentum term α (0.9) and added to the t+1 weights W_H^{t+1} .

$$W_O^{t+1} = W_O^t + \eta \Delta_O^t$$

$$= \begin{bmatrix} 0.2288 \\ -0.3478 \\ 0.4991 \end{bmatrix} + 0.03 \begin{bmatrix} 0.1437 \\ -0.1966 \\ -1.1419 \end{bmatrix} = \begin{bmatrix} 0.2331 \\ -0.3537 \\ 0.4649 \end{bmatrix}$$

$$W_O^{t+1} = W_O^{t+1} + \alpha \Delta_O^{t-1}$$

$$= \begin{bmatrix} 0.2331 \\ -0.3537 \\ 0.4649 \end{bmatrix} + 0.9 \begin{bmatrix} 0.0049 \\ -0.0073 \\ -0.0363 \end{bmatrix} = \begin{bmatrix} 0.2375 \\ -0.3602 \\ 0.4322 \end{bmatrix}$$

Recoding the Hidden Layer

- Mapping at end of training
  - Output Y_300 exhibits XOR function
  - Predict XOR output (P_300) from hidden H1_300 and H2_300 using regression
- Mapping at time 20
  - Output Y_20 exhibits OR function (positive when either input is on)
  - Not possible to predict XOR output (P_20) from hidden H1_20 and H2_20

recoded XOR

I1	I2	T	H1_20	H2_20	Y_20	P30	H1_300	H2_300	Y_300	P_300
0.9	0.9	-0.9	0.82	-0.96	0.27	0.16	0.85	-1	-0.83	-0.9
0.9	-0.9	0.9	-0.18	-0.69	0.12	0.23	-0.72	-0.87	0.86	0.9
-0.9	0.9	0.9	0.59	-0.04	0.04	-0.14	-0.7	-0.88	0.86	0.9
-0.9	-0.9	-0.9	-0.58	0.79	-0.24	-0.25	-1	0.9	-0.84	-0.9

Summary Back-propagation

- Forward Pass
  - Multiply inputs and weights
  - Apply activation function
- Backward Pass
  - Compute error
  - Compute delta weight change matrix
    - Change weights (learning rate)
    - Add Momentum
  - Back-propagate error to previous layer
- Repeat for each layer

Non-linear Regression

- Regression models are a type of linear model
  - Predicted outputs are a weighted sum of their inputs (e.g., y = a + bx)
  - Hidden->output part of XOR model without tanh would be linear model
- Link functions in general linear models are akin to the activation functions in neural networks
  - Binomial link function is akin to using sigmoid logistic activation function
    - Netinput to the neuron is called the logit (Bishop, 2006)
  - tanh is another type of sigmoid function that goes between [-1,1]
- Neural network models are non-linear regression models
  - Recoding the hidden layer to solve the mapping (regression cannot do this)
  - Recoding takes time and there are many solutions (local minima)
    - Children also make errors during language learning (e.g., *goed* instead of *went*)
    - Regression can't learn one solution and then recover later

R Code: Initializing the input

- backpropLib.R, xor.R

```
xor = matrix(c(0,0,   0,
               0,1,   1,
               1,0,   1,
               1,1,   0  ), nrow=4, byrow=TRUE)
xor2 = convertRange(xor,-0.9,0.9) ## tanh requires range of [-1,1]
Inputs = xor2[,1:2]
print(Inputs)
```

```
##      [,1] [,2]
## [1,] -0.9 -0.9
## [2,] -0.9  0.9
## [3,]  0.9 -0.9
## [4,]  0.9  0.9
```

```
Targets = matrix(xor2[,3])
print(t(Targets))  # transposed function t
```

```
##      [,1] [,2] [,3] [,4]
## [1,] -0.9  0.9  0.9 -0.9
```

R Code: Initializing the network

```
NumInputs = 2
NumHidden = 2
NumOutputs = 1
layerList <- list()  # network is stored in global variable layerList
makeLayer("input", NumInputs)
makeLayer("hidden", NumHidden)
makeLayer("output", NumOutputs)
makeLink("input","hidden") # create weights from input to hidden
makeLink("hidden","output") # create weights from hidden to output
class(layerList[[getLayer("output")]] ) <- c("output","layer") # set output layer as output class
```

R Code: hidden layer layerList[[2]]

```
## $name
## [1] "hidden"
##
## $num
## [1] 2
##
## $unitn
## [1] 2
##
## $inputlayer
## [1] 1
##
## $inunitn
## [1] 3
##
## $weights
## NULL
##
## $input
## NULL
##
## $netinput
## NULL
##
## $output
## NULL
##
## $targcopy
```

R Code: Forward pass function (backpropLib.R)

```
forwardPassOne.layer <- function(lay){
  # spread input activation through weights (dot/inner product %*%)
  lay$netinput <-lay$input %*% lay$weights
  # input activation at each unit passed through output function (tanh)
  lay$output <- tanhActivationFunction(lay$netinput)
  layerList[[lay$num]] <<- lay
}

errorFunc.output <- function(lay){
  if (!is.null(lay$targcopy)){
    # some layers get targets from other layers, so copy the targets in those cases
    lay$target = layerList[[lay$targcopy]]$output
  }
  # compute error
  lay$error =  lay$target-lay$output
  # zero error radius sets any error that is close enough to the target to be 0
  # this helps to reduce extreme weights and keeps values within sensitive region of activation functio
n
#  lay$error[abs(lay$error) < lay$zeroErrorRadius] = 0
  layerList[[lay$num]] <<- lay
}
```

R Code: Backward pass function (backpropLib.R)

```
backPropOne.layer <- function(lay){
  lay$deriv <- derivativeFunction(lay) # compute derivative
  lay$gradient <- lay$error * lay$deriv # gradient is error * derivative

  deltaWeights <- t(lay$input) %*% lay$gradient # compute weight change matrix
  deltaWeights <- deltaWeights * lay$lrate # modulate with lrate

  weights <- lay$weights + deltaWeights    # steepest descent
  weights <- weights +  lay$momentum * lay$prevDW  # momentum descent:

  lay$prevDW<-deltaWeights    # save delta weights with learning rate adjustment
  lay$weights <- as.matrix(weights)     # save new weights

  # error is back-propagated and saved in backerror
  W = as.matrix(weights[1:(dim(weights)[1]-1),]) # remove bias weight
  lay$backerror = lay$gradient %*% t(W)   # backprop gradient
  ind = 1
  for (j in lay$inputlayer){
    nind = ind+layerList[[j]]$unitn    # copy to all input layers
    layerList[[j]]$error <<- lay$backerror[,ind:(nind-1)]
    ind = nind
  }
}
```

R Code: Training code

```
numEpochs<-1000    # number of training cycles thru whole batch
setParamAll("lrate",0.01) # learning rate: speed of learning
setParamAll("momentum",0.9) # amount of previous weight changes that are added
setParamAll("patn",4)  # four patterns
resetNetworkWeights()  # randomize network weights
layerList[[getLayer("input")]]$output <- Inputs    # set inputs
layerList[[getLayer("output")]]$target <- Targets  # set target

# Train model 1000 times
############
history=data.frame()  # stores evaluation data during training
for (epoch in 1:numEpochs){

  forwardPass()  # spread activation forward for all patterns in training set
  backpropagateError()  # back propagate error and update weights

  slice = evaluateModelFit(epoch)  # save model parameters for figures
  slice$Hidden1 = layerList[[length(layerList)]]$weights[1]
  slice$Hidden2 = layerList[[length(layerList)]]$weights[2]
  slice$Bias = layerList[[length(layerList)]]$weights[3]
  history = rbind(history,slice)
}
```

R Code: Evaluating the model

```
# meanCE plot
costPlot = ggplot(history,aes(x=time,y=Cost))+geom_line()+ylab("Cross-entropy")
print(costPlot)

# MSDelta plot
deltaPlot = ggplot(subset(history, time > 5),aes(x=time,y=MSDelta,colour=Layer))+geom_line()
deltaPlot = deltaPlot + theme(legend.position="top",legend.direction ="horizontal")
print(deltaPlot)

# Network Diagram
modelout = addModel("output")
arr=createWeightArrows(modelout)
plotModel(modelout,xlabel=c("Input","", "Hidden","", "Output","Target"),arr=arr,col=1)

# Heatmaps
weight12fig = mapOutWeightSpace("Hidden1","Hidden2","output",history)
weight13fig = mapOutWeightSpace("Hidden1","Bias","output",history)
layout <- matrix(c(1,2,3,4),ncol = 2, nrow = 2)
print(multiplot(costPlot,deltaPlot,weight12fig,weight13fig , layout=layout))
```

Lab Setup

- Decompress neural.zip file and save neural folder on desktop
- Start Rstudio and Open File and click on xor.R
- Set working directory so that program can find files
  - Session menu, Set Working Directory, To Source File location
  - setwd("~/Desktop/neural/")
- Try to run program
  - Select all of the text in xor.R (control-A, apple-A)
  - Run the text (control-R, apple-enter)

XOR Lab Exercises

- Run the whole script several times
  - The results change depending on the random initial weights
  - Look at nnpred output in xor2.df. Did the model learn XOR?
- Search for the line that says setParamAll("momentum",0.9)
  - change momentum to 0 as in setParamAll("momentum",0)
  - Run model
  - The steps in the heat map should be smaller
- Change number of hidden units
  - NumHidden = 2 -> NumHidden = 1
  - One heat map won't work, but the other will show up
  - Look at nnpred output in xor2.df. What logical function did the model learn?
  - Try three units, can the model learn XOR?

Language sequence learning

- Many sequences in language (sentences are sequences of words, words are sequences of sounds)
- Simple recurrent network (Elman, 1990; Rohde & Plaut, 1999).
- Simple language with eight sentences
  - Four animate entities (i.e. *girl*,*boy*,*cat*,*dog*)
  - Two inanimate food entities (i.e. *apple*, *cake*).
  - verb *chase* can only go with animate agents and patients
  - verb *eats* has an animate agent and inanimate patient

Next word is training signal for model

- Previous word in a sentence to predict the next word in the sequence
  - Next word that is heard is the training signal (target)

	Previous Word	Next Word
1	.	the
2	the	boy
3	boy	eats
4	eats	the
5	the	cake
6	cake	.
44	the	girl
45	girl	chases
46	chases	the
47	the	boy
48	boy	.
49	.	.

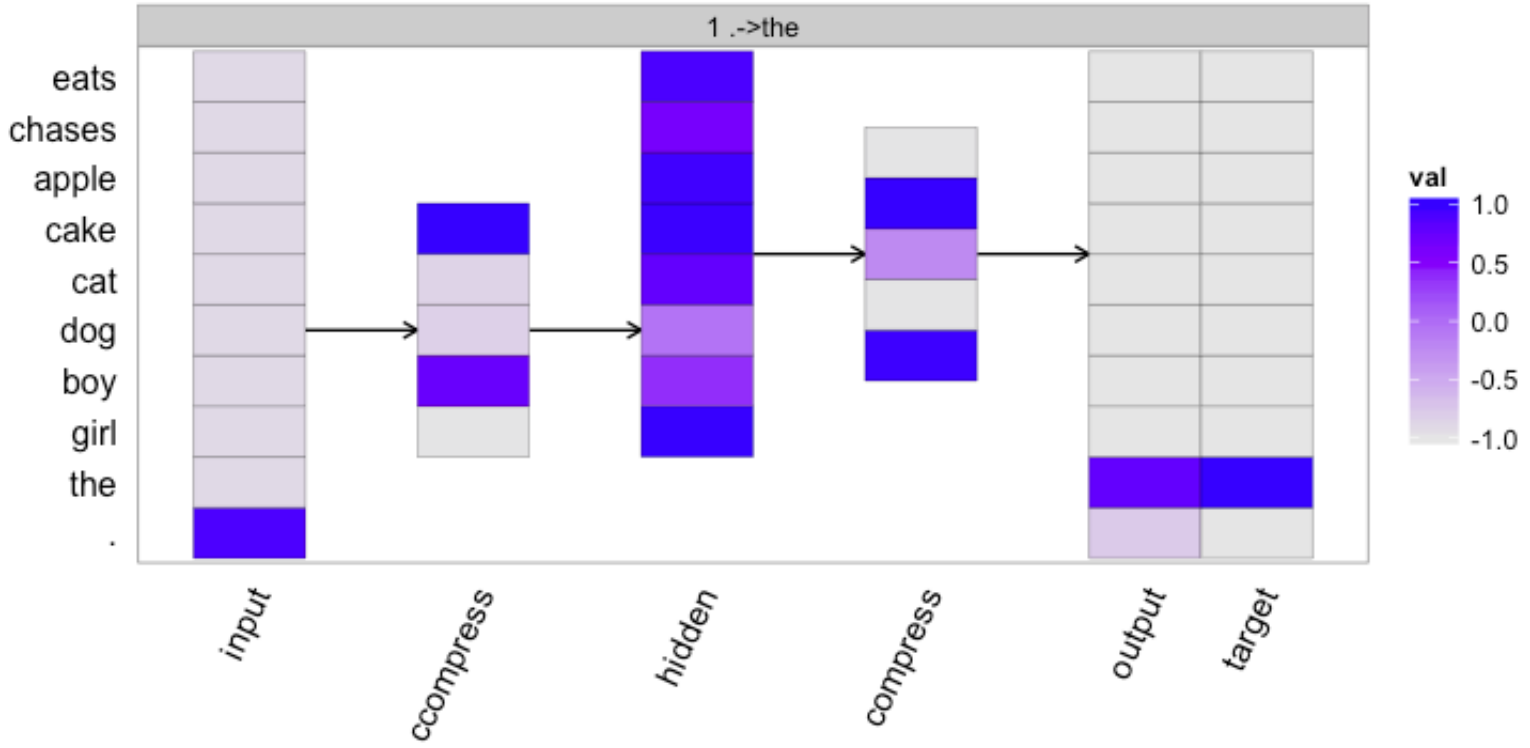
Localist Coding of Words

- Each word is represented by a single unit as 1 and the rest as 0

	Word	.	the	girl	boy	dog	cat	cake	apple	chases	eats
1	the	0	1	0	0	0	0	0	0	0	0
2	boy	0	0	0	1	0	0	0	0	0	0
3	eats	0	0	0	0	0	0	0	0	0	1
4	the	0	1	0	0	0	0	0	0	0	0
5	cake	0	0	0	0	0	0	1	0	0	0
6	.	1	0	0	0	0	0	0	0	0	0

Feed-forward Architecture

- 5 layers: input -> ccompress -> hidden -> compress -> output
- compress layers reduce the distinctions that can be represented -> syntactic categories



Soft-max activation function

- In sentence production, we want to bias towards producing one word
  - Soft-max is an activation function with this bias

$$Y_O = \frac{exp(V_O)}{\sum_r (exp(V_O))}$$

$$= \exp\left(\begin{bmatrix} 1 & 3 \\ 1 & 4 \\ 2 & 1 \end{bmatrix}\right) / \sum_r (exp(V_O))$$

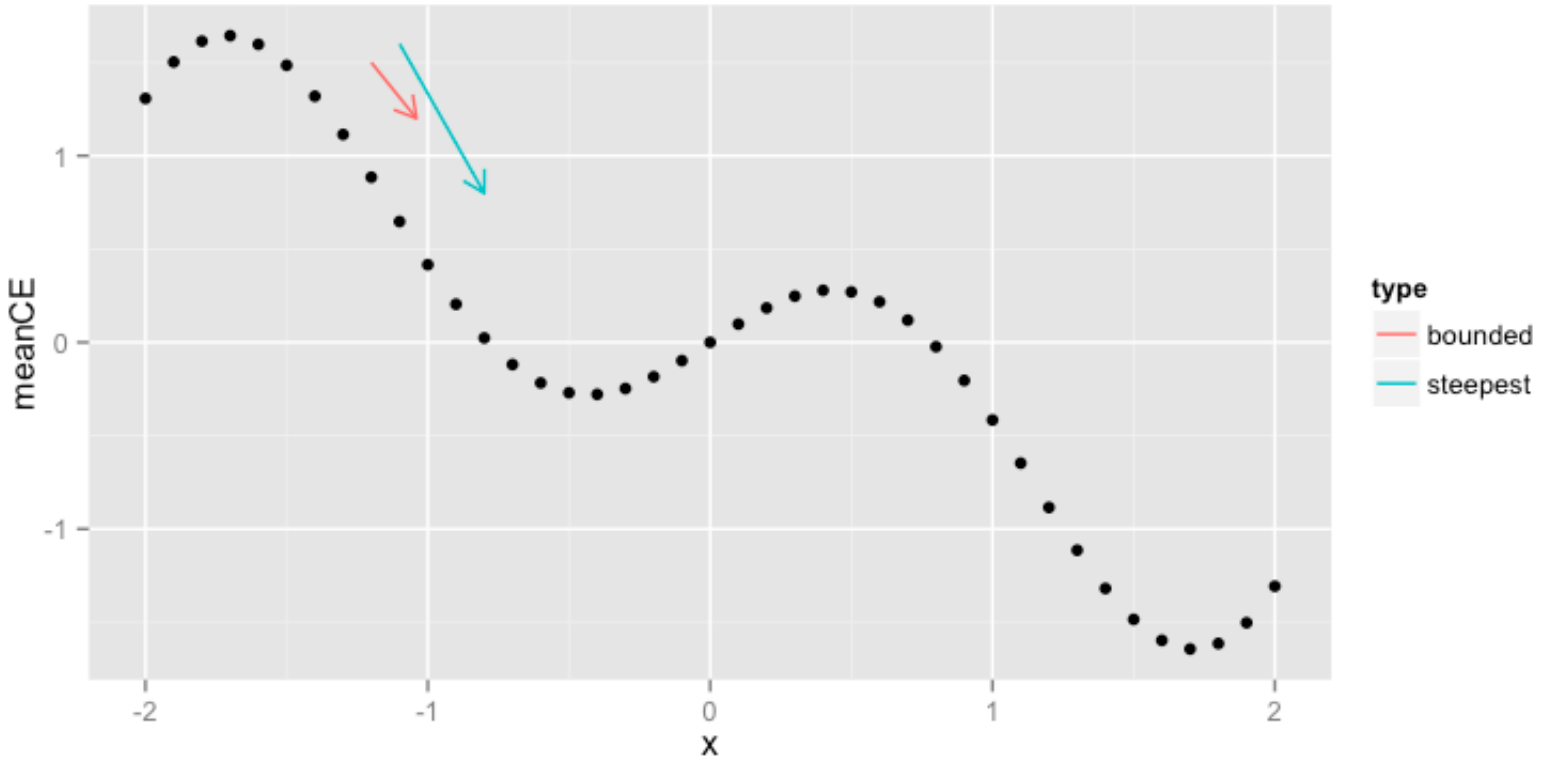
$$= \begin{bmatrix} 3 & 20 \\ 3 & 55 \\ 7 & 3 \end{bmatrix} / \begin{bmatrix} 23 \\ 58 \\ 10 \end{bmatrix}$$

$$= \begin{bmatrix} 0.13 & 0.87 \\ 0.05 & 0.95 \\ 0.7 & 0.3 \end{bmatrix} \quad (16)$$

Bounded Descent

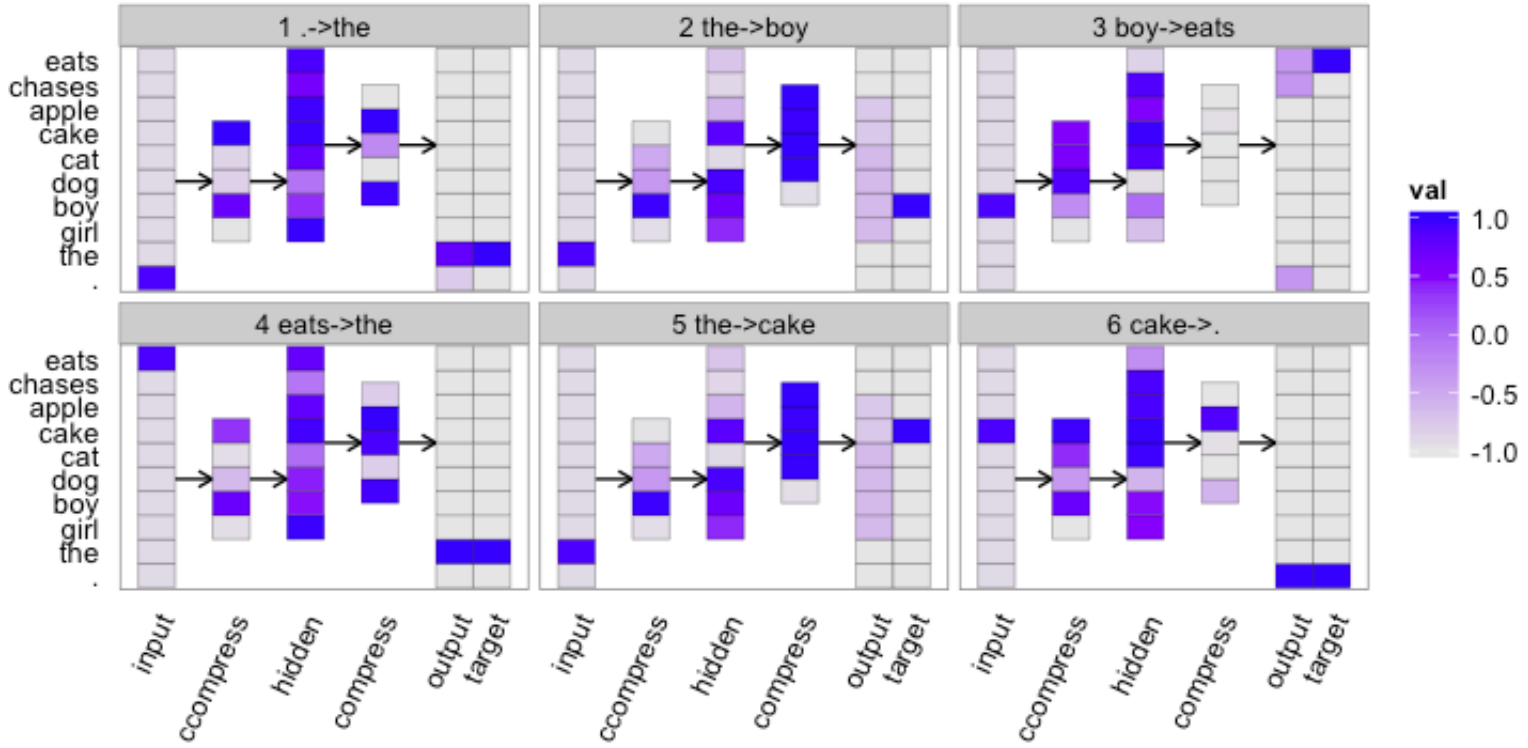
- Steepest descent takes a big step when the slope is steep
- Bounded descent restricts the length of the step if length > 1 (Rohde, 2002)

$$\Delta^B = \Delta / \sqrt{\sum \Delta^2}$$



Testing the model

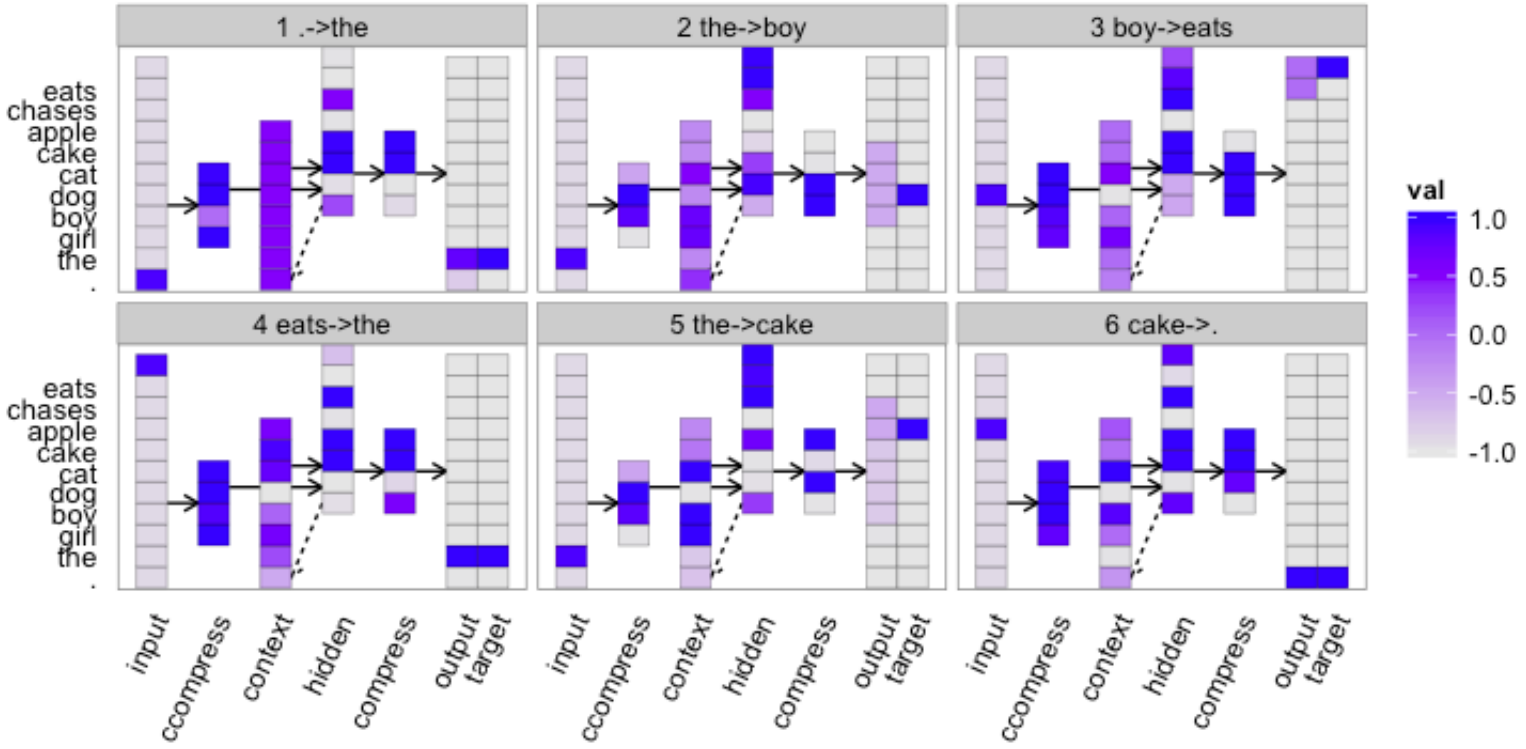
- Test sentence *the boy eats the cake*
- Model does not learn non-adjacent regularities: *eats* followed by foods



Simple recurrent network

- Context layer holds a copy of the previous hidden layer representations
- Hysteresis: How much of previous hidden activation to keep in context?

$$Y^{t}_{context} = (1-\phi)Y^{t-1}_{hidden} + \phi Y^{t-2}_{hidden}$$



Summary Simple Recurrent Network

- Model learns sequencing constraints
  - Compress layer creates syntactic categories
  - Context allows model to remember past
- Model generalizes because of positional learning
  - Novel test: *the dog eats the man*
  - Related input: *the woman chases the man*
  - Local constraint: *the -> man*
  - Harder to learn the constraint of *eat* on argument

SRN R Code: Creating the Input

```
sentences =c("the boy eats the cake .", "the boy chases the girl .", "the dog eats the apple .", "the d
og chases the cat .", "the cat chases the dog .", "the cat eats the cake .", "the girl eats the apple .
", "the girl chases the boy .")

wordseqlabels = unlist(str_split(sentences, " "))
input = data.frame(prevword=c(".",wordseqlabels), nextword = c(wordseqlabels,"."))
vocabulary = c(".","the","girl","boy","dog","cat","cake","apple", "chases","eats" )
input$prevword = factor(input$prevword,levels = vocabulary)
input$nextword = factor(input$nextword,levels = vocabulary)
vocabulary = levels(input$nextword)
InputOutput = t(mapply(convertWordVector,input$prevword,input$nextword))
periodList = InputOutput[,1]  # tells us where to reset the context
Inputs = InputOutput[,1:NumInputs]
Targets = InputOutput[,(NumInputs+1):(dim(InputOutput)[2])]
Targets[1:6,]  # the(2) boy(4) eats(10) the(2) cake(7) .(1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]    0    1    0    0    0    0    0    0    0     0
## [2,]    0    0    0    1    0    0    0    0    0     0
## [3,]    0    0    0    0    0    0    0    0    0     1
## [4,]    0    1    0    0    0    0    0    0    0     0
## [5,]    0    0    0    0    0    0    1    0    0     0
## [6,]    1    0    0    0    0    0    0    0    0     0
```

SRN R Code: Creating the Network

```
layerList <<- list()
  makeLayer("input", NumInputs)
  makeLayer("ccompress", NumCompress)
  makeContextLayer("context", NumHidden, Inputs[,1])
  makeLayer("hidden", NumHidden)
  makeLayer("compress", NumCompress)
  makeLayer("output", NumOutputs)

  makeLink("input","ccompress")
  makeLink("ccompress","hidden")
  makeLink("hidden","context")
  makeLink("context","hidden")
  makeLink("hidden","compress")
  makeLink("compress","output")
```

SRN R Code: Changing input

```
setInputs <- function(inp,tar,perlist){
  # set number of patterns in network
  setParamAll("patn",dim(tar)[1])

  layerList[[getLayer("input")]]$output <<- inp  # set input

  # we need to tell the network when the reset the context to 0.5
  # perlist is a list with a 1 if the previous word was a period, 0 otherwise.
  layerList[[getLayer("context")]]$reset <<- perlist
  # Since the context is copied from the hidden layer,
  # it also needs to be reset
  hidlay = layerList[[getLayer("hidden")]]
  hidlay$output <- matrix(0.5,hidlay$patn,hidlay$unitn)
  layerList[[getLayer("hidden")]] <<- hidlay

  layerList[[getLayer("output")]]$target <<- tar
  # this makes the final layer a softmax layer
  class(layerList[[getLayer("output")]]) <<- c("softmax","output","layer")
}
```

SRN R Code: Setting Training Parameters

```
numEpochs<-10000
## create and setup model
createSRN()
setParamAll("lrate",0.1) # learning rate: speed of learning
setParamAll("momentum",0.9) # amount of previous weight changes that are added
setParamAll("boundedDescent",TRUE)  # bounded descent algorithm is used
setParamAll("hysteresis",0.5)  # use half of previous context activation
setParamAll("zeroErrorRadius",0.1)  # error < 0.2 is set to zero
setInputs(Inputs,Targets,periodList)
resetNetworkWeights()
```

SRN R Code: Training and testing

```
history=data.frame()
for (epoch in 1:numEpochs){
  forwardPass() # spread activation forward for all patterns in training set
  backpropagateError()  # back propagate error and update weights

  if (epoch %% 1000==0){ # every 1000 epochs, test the model and plot MSDelta graph
    slice = evaluateModelFit(epoch)  # save model parameters for figures
    history = rbind(history,slice)
    # plot delta during training to see how layers are changing
    deltaPlot = ggplot(history,aes(x=time,y=MSDelta,colour=Layer))+geom_line()
    print(deltaPlot)
  }
}
```

SRN R Code: Plotting the model

```
# plot the model output for the first 6 patterns
layerList[[getLayer("compress")]]$verticalPos = 3.5
layerList[[getLayer("ccompress")]]$verticalPos = 3.5
layerList[[getLayer("hidden")]]$verticalPos = 1
input$prevword = factor(input$prevword,levels = vocabulary)
input$nextword = factor(input$nextword,levels = vocabulary)
sentseq = paste(input$prevword ,input$nextword,sep="->")
modelout = addModel("output",restrict=TRUE,labels=sentseq)
dd = subset(modelout, pattern %in% 1:12)
plotModel(dd,ylabel=vocabulary,axisfontsize=10)
```

SRN Lab

- Run the whole script several times (select all, run)
  - Does the model learn syntactic categories?
  - Sequencing constraints?
- Change the language regularities
  - Replace the sentence "the boy eats the cake ." with "the apple eats the cake ."
  - Does the model acquire this constraints?
  - Change other parts of the language and see if the model acquires them
- Add a new construction
  - Add passive structures like "the cake is eats by the boy ."
    - Verb should be *eaten*, but we leave it as *eats* for simplicity
  - You will need to add "by" and "is" to the vocabulary
  - vocabulary = c(".","the","girl","boy","dog","cat","cake","apple", "chases","eats")

Sentence Production: Using meaning to guide production

- Language is used to convey meaning
  - ACTION=CHASE AGENT=CAT PATIENT=DOG
  - the cat chased the dog 猫が犬を追いかけた
  - the dog was chased by the cat 犬を猫が追いかけた
  - SRNs can process these sentences, but it can't pick a particular sentence to convey a particular meaning
- Models of sentence production (Chang, 2002)
  - Prod-SRN model
  - Dual-path model

Japanese Training Language

- 100 Message-sentence pairs
- animals chase animals, animals eat foods
- Japanese canonical utterances (e.g., cat ga dog o chased)
- Japanese scrambled utterances (e.g., dog o cat ga chased)

Message	Sentence	Structure
ACTION=EATS AGENT=DOG PATIENT=CAKE	cake o dog ga eats .	Scrambled
ACTION=CHASES AGENT=GIRL PATIENT=CAT	cat o girl ga chases .	Scrambled
ACTION=CHASES AGENT=GIRL PATIENT=CAT	girl ga cat o chases .	Canonical
ACTION=EATS AGENT=GIRL PATIENT=APPLE	girl ga apple o eats .	Canonical
ACTION=EATS AGENT=DOG PATIENT=APPLE	dog ga apple o eats .	Canonical
ACTION=CHASES AGENT=GIRL PATIENT=BOY	girl ga boy o chases .	Canonical

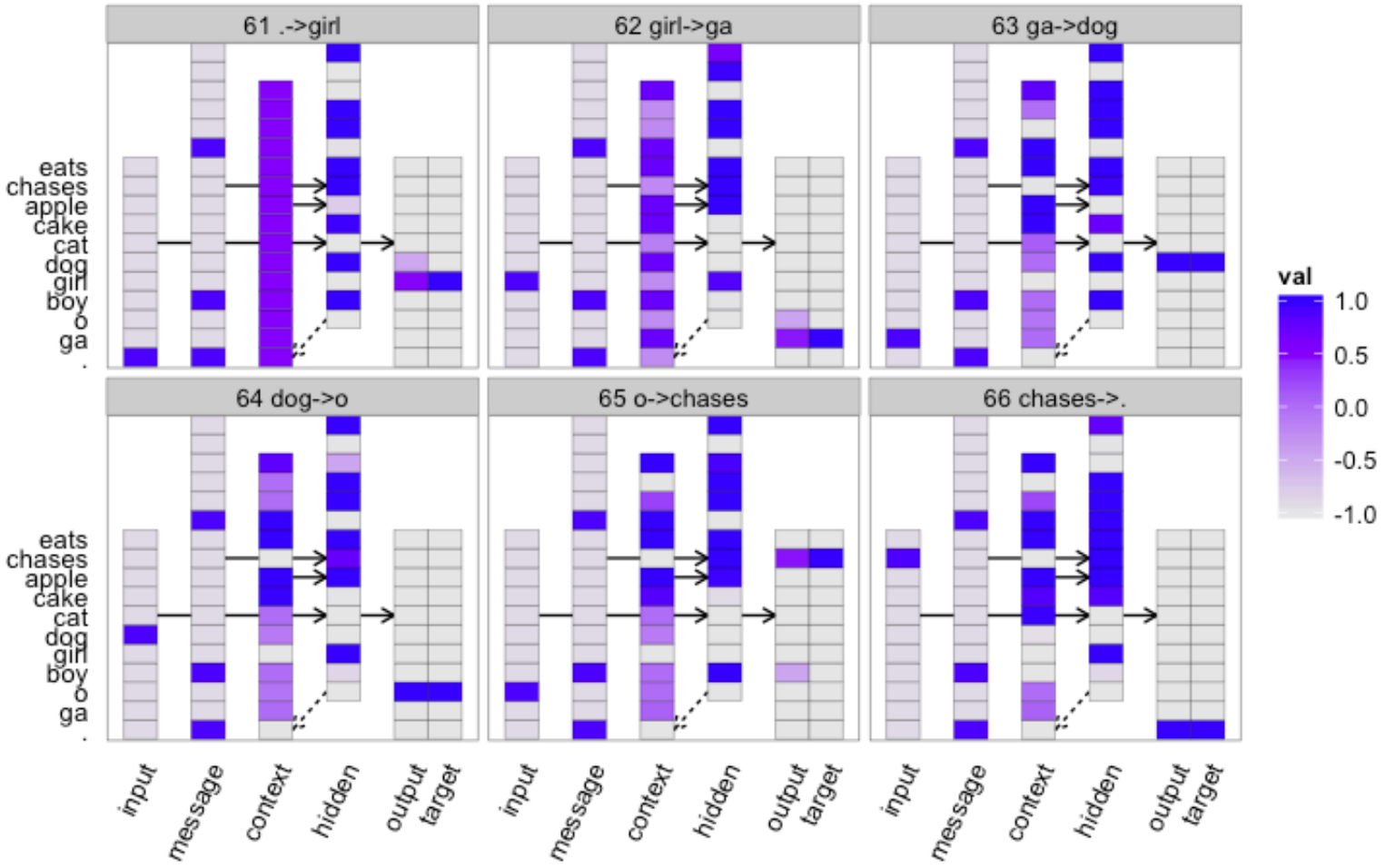
Prod-SRN

- Simplest production model is to just add a message to SRN model
- Binding-by-space message
  - Slots for ACTION (2 units), AGENT (7 units), PATIENT (7 units)
- ACTION=CHASE AGENT=CAT PATIENT=DOG
  - Unit 1 is activated for chase
  - unit 6 is activated for cat (cat is 4th noun, agent slot starts at 2)
  - unit 12 for dog (dog is 3rd noun, patient slot starts at 9).
  - In the actual message, the 0 values are changed into -1, to center the values around 0.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

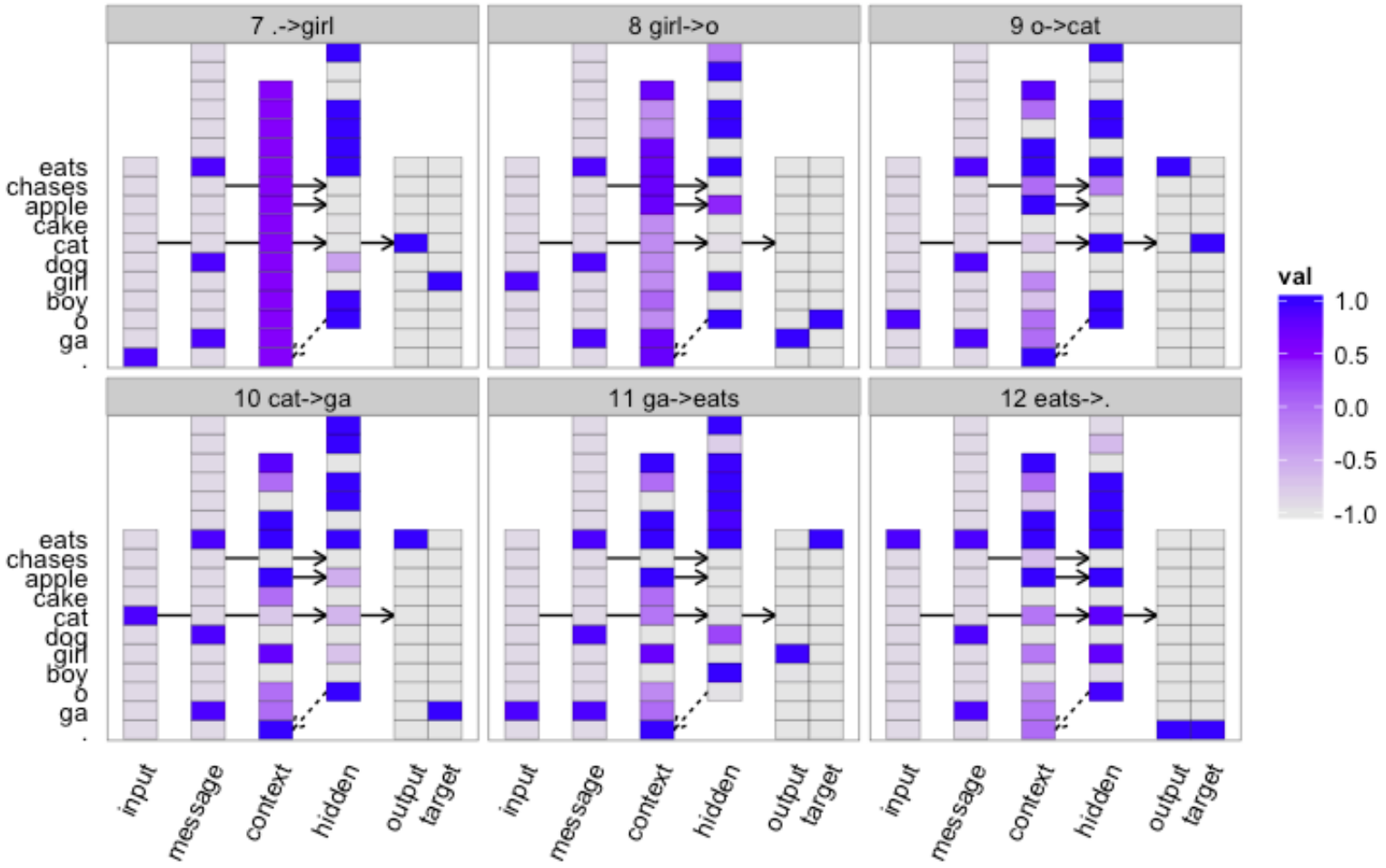
Prod-SRN Architecture

- Model is mostly correct in predicting sentences that it was trained on



Generalization

- Prod-SRN does not generalize well (unusual meaning *girl o cat ga eats*)



Summary Prod-SRN

- Lack of variable-like processing in connectionist models (Fodor & Pylyshyn, 1988; Marcus, 2003)
- Binding-by-space message: AGENT-cat, PATIENT-cat are different concepts and must be learned independently
  - Can't generalize from AGENT-cat PATIENT-dog to AGENT-dog PATIENT-cat
  - Many models use this representation (Mayberry, Crocker, & Knoeferle, 2009; Rohde, 2002; St. John & McClelland, 1990)
- Prod-SRN does not have compress units, so no syntactic categories

Dual-path model

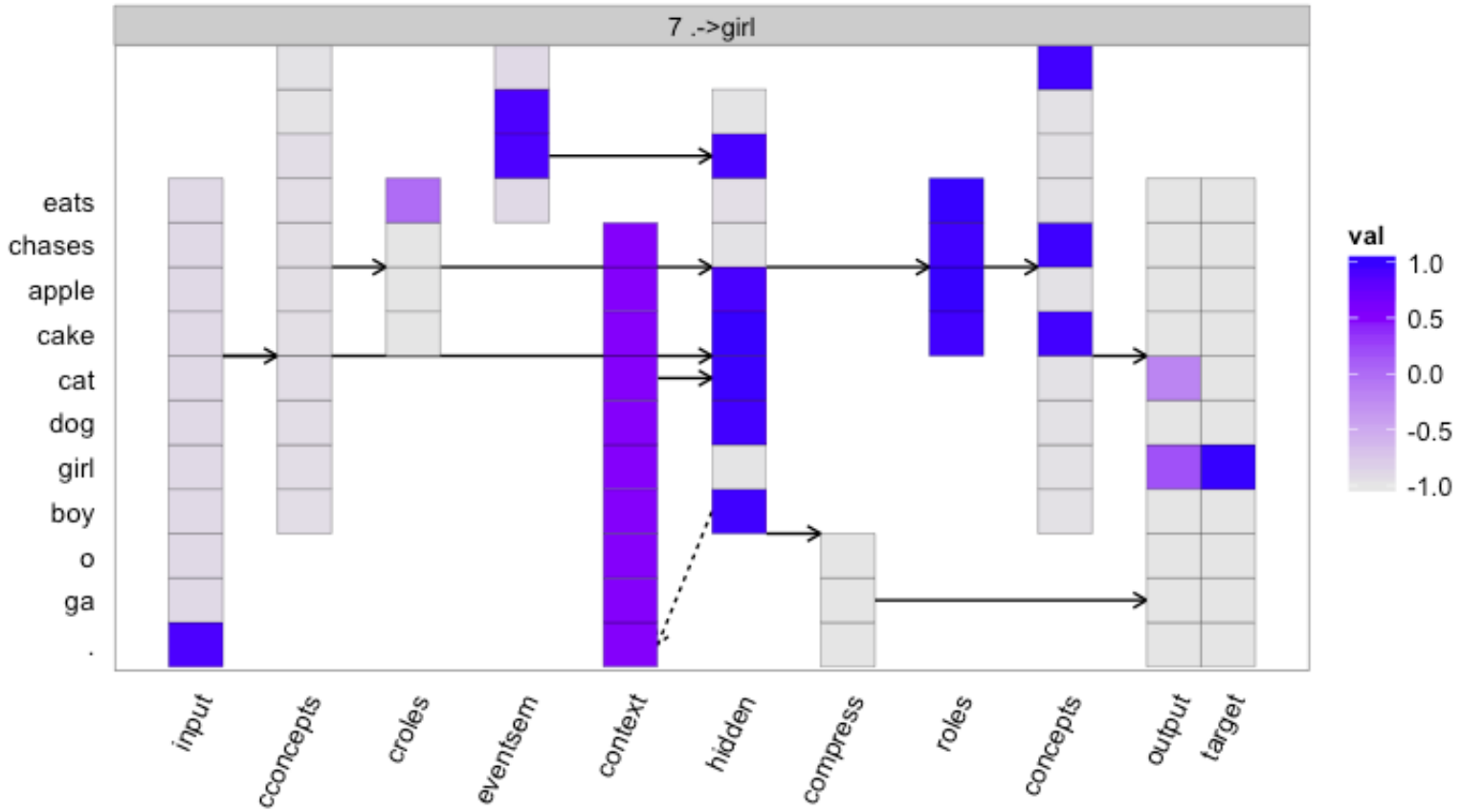
- Variables to represent binding of roles and concepts (AGENT=cat)
  - Message is instantiated in weights

$$M=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- Activation of role causes concept to be activated

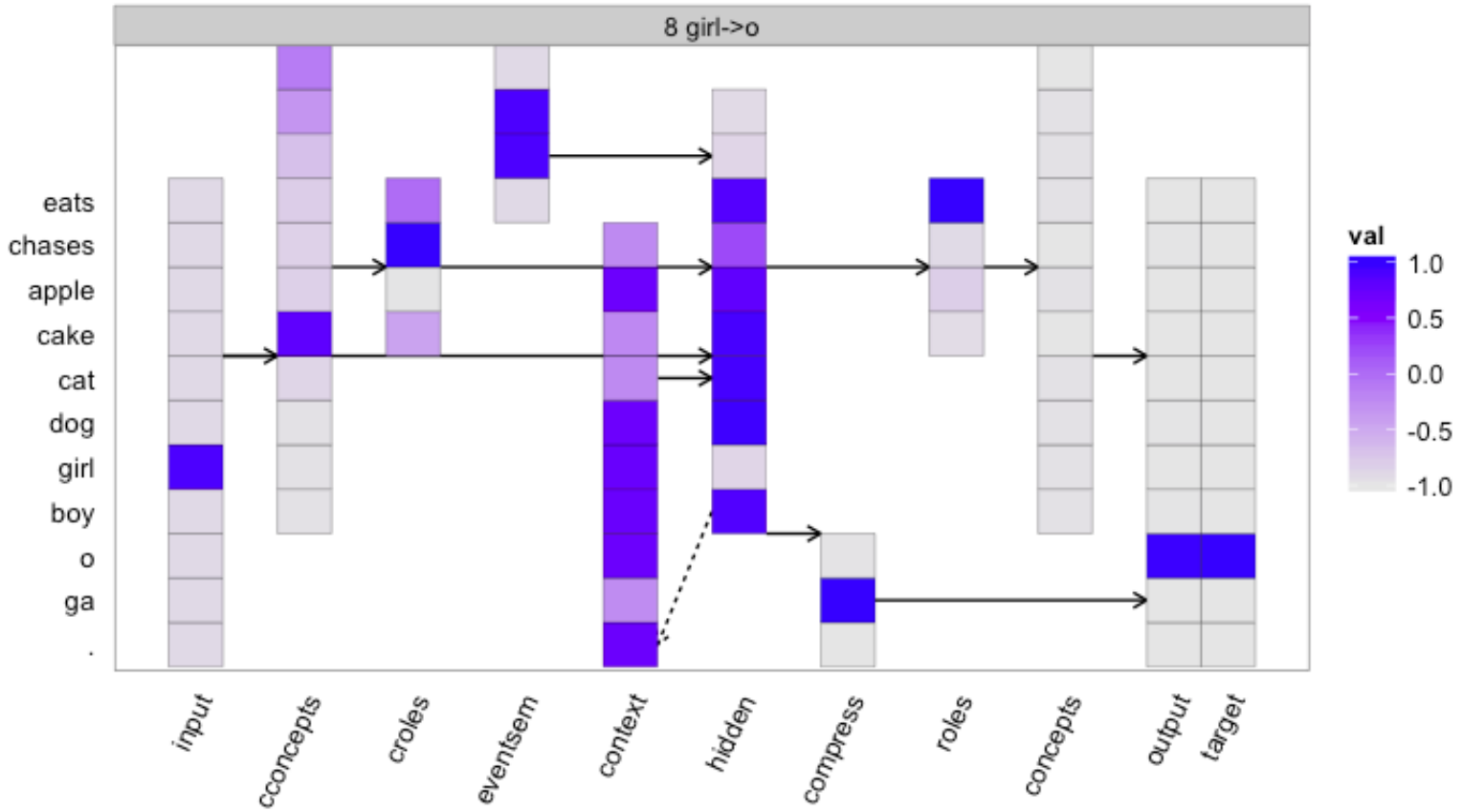
Dual-path architecture

- Message and syntactic networks are in separate pathways
- Event semantics provides information about number of roles
- *girl o cat ga eats* is scrambled and semantically unusual



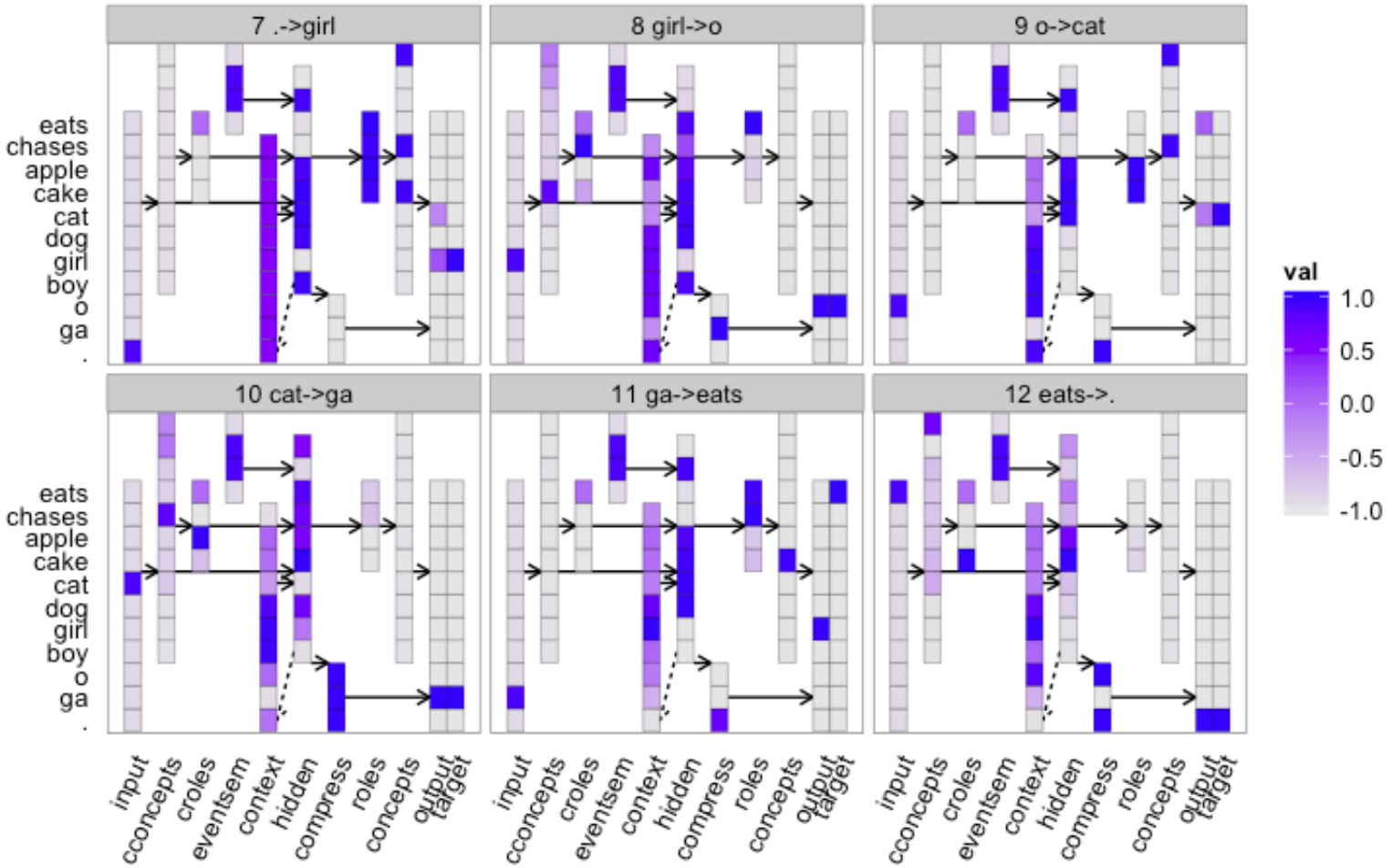
Dual-path architecture

- *girl o cat ga eats* is scrambled and semantically unusual
- Reverse cconcept->croles message tells hidden layer the role of previous word
- cconcept gets training signal from concept (learn by prediction)



Dual-path Model

- Overall can react to the input better than Prod-SRN



Summary

- Feedforward XOR model
  - Non-linear regression
  - Recodes the input
- Simple recurrent networks (Elman, 1990)
  - Previous word predicts next word
  - Non-adjacent regularities *eat* -> food
- Dual-path model
  - Meaning, Novel verb-structure regularities (Chang, 2002)
  - Aphasic double dissociations (Chang, 2002, see also Gordon & Dell, 2003).
  - Structural priming (Chang, Dell, & Bock, 2006; German: Chang, Baumann, Pappert, & Fitz, in press)
  - Heavy NP shift/Accessibility in English/Japanese (Chang, 2009)
  - Learning verb bias (Twomey et al., 2014)
  - Syntactic bootstrapping (Chang et al., 2006)

Dual-path R Code: Creating input

```
npat = 100
actors = c("boy","girl","dog","cat")
foods = c("apple","cake")
constrsent.df = data.frame(str=ifelse(runif(npat) < 0.3,"P","A"),act=ifelse(runif(npat) < 0.5,"eats","c
hases"))
constrsent.df$agent = actors[sample(1:4,npat,replace=TRUE)]
constrsent.df$patient = actors[sample(1:4,npat,replace=TRUE)]
constrsent.df$patient[constrsent.df$act == "eats"] = foods[sample(1:2,length(constrsent.df$patient[cons
trsent.df$act == "eats"]),replace=TRUE)]
constrsent.df$mess = paste("action=",constrsent.df$act," agent=",constrsent.df$agent," patient=",constr
sent.df$patient,sep="")
head(constrsent.df)
```

```
##   str    act agent patient                                  mess
## 1   P   eats   cat   apple  action=eats agent=cat patient=apple
## 2   P chases   cat     dog  action=chases agent=cat patient=dog
## 3   P chases   cat     cat  action=chases agent=cat patient=cat
## 4   P   eats   cat   apple  action=eats agent=cat patient=apple
## 5   A   eats   dog   apple  action=eats agent=dog patient=apple
## 6   P chases  girl     boy action=chases agent=girl patient=boy
```

Dual-path R Code: Creating input

```
constrsent.df$sent = paste(constrsent.df$agent,"ga",constrsent.df$patient,"o",constrsent.df$act,".")
constrsent.df$psent = paste(constrsent.df$patient,"o",constrsent.df$agent,"ga",constrsent.df$act,".")
constrsent.df$sent[constrsent.df$str == "P"] = constrsent.df$psent[constrsent.df$str == "P"]
constrsent.df$agent = NULL
constrsent.df$patient = NULL
constrsent.df$act = NULL
constrsent.df$psent = NULL
head(constrsent.df)
```

```
##   str                                  mess                   sent
## 1   P  action=eats agent=cat patient=apple  apple o cat ga eats .
## 2   P  action=chases agent=cat patient=dog  dog o cat ga chases .
## 3   P  action=chases agent=cat patient=cat  cat o cat ga chases .
## 4   P  action=eats agent=cat patient=apple  apple o cat ga eats .
## 5   A  action=eats agent=dog patient=apple  dog ga apple o eats .
## 6   P action=chases agent=girl patient=boy boy o girl ga chases .
```

Dual-path R Code: Convert message into numbers

```
convertMessageCodes <- function(mess){
  allMesList = list()
  for (m in mess){
    #  print(m)
    mesList = ""
    if (m != ""){
      pairs = str_split(m, " ")
      for (p in pairs[[1]]){
        rc=str_split_fixed(p,"=",2)
        pairList = paste(which(roles == rc[1]), which(vocabulary == rc[2]),sep=",")
        #     print(pairList)
        mesList = paste(mesList, pairList, sep=",")
      }
    }
    allMesList = append(allMesList,mesList)
  }
  unlist(allMesList)
}
convertMessageCodes("action=eats agent=dog patient=apple")
```

```
## [1] ",1,11,2,6,3,9"
```

Dual-path R Code: Create weight matrix from message

```
makeMessages <- function(m,r,c,rc){
  ml = str_split(m[[1]],",")
  ml2 = as.numeric(ml[[1]])
  wts = matrix(0,r,c)
  wts[r,] = -2
  if (length(ml2)>2){
    for (i in seq(2,length(ml2),2)){
      if (rc == TRUE){
        wts[ml2[i],ml2[i+1]] <- 4
      }else{
        wts[ml2[i+1],ml2[i]] <- 4
      }
    }
  }
  wts
}
m = convertMessageCodes("action=eats agent=boy patient=cake")
makeMessages( m ,4,11 ,TRUE)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
## [1,]    0    0    0    0    0    0    0    0    0     0     4
## [2,]    0    0    0    4    0    0    0    0    0     0     0
## [3,]    0    0    0    0    0    0    0    4    0     0     0
## [4,]   -2   -2   -2   -2   -2   -2   -2   -2   -2    -2    -2
```

Dual-path R Code: Event semantics

```
makeEventSem <- function(mes, inp,slen){
  rolelist = str_extract_all(mes,"(agent|patient)")
  mlist=NULL
  for (rl in 1:length(rolelist)){
    if (!is.na(slen[rl])){
      mat = matrix(-0.9,slen[rl],NumRoles)
      for (r in rolelist[rl][[1]]){
        for (i in 1:length(roles)){
          if (roles[i]==r){  mat[,i]=0.9 }
        }
      }
      mlist = rbind(mlist,mat)
    }
  }
  rbind(mlist , mlist[dim(mlist)[1],])
}
makeEventSem("action=eats agent=boy patient=cake",0,1)
```

```
##      [,1] [,2] [,3]
## [1,] -0.9  0.9  0.9
## [2,] -0.9  0.9  0.9
```

Dual-path R Code: Network

```
makeLayer("input", NumInputs)
makeLayer("cconcepts", NumOutputs)
makeLayer("croles", NumRoles)
makeLayer("eventsem", NumRoles)
makeContextLayer("context", NumHidden, periodList)
makeLayer("hidden", NumHidden)
makeLayer("compress", NumCompress)
makeLayer("roles", NumRoles)
makeLayer("concepts", NumOutputs)
makeLayer("output", NumOutputs)
makeLink("input","hidden")
makeLink("input","cconcepts")
makeLink("cconcepts","croles")
makeLink("croles","hidden")
makeLink("hidden","context")
makeLink("context","hidden")
makeLink("eventsem","hidden")
makeLink("hidden","roles")
makeLink("roles","concepts")
makeLink("concepts","output")
makeLink("hidden","compress")
makeLink("compress","output")

# cconcepts gets target from concepts
makeCopyTargetLink("concepts","cconcepts")
```

Dual-path R Code: Inputs

```
setInputs <- function(inp,tar,mes, slen, perlist){
  evsem = makeEventSem(mes,inp,slen)   # set the event semantics
  layerList[[getLayer("eventsem")]]$output <<-  evsem
  class(layerList[[getLayer("eventsem")]]) <<- c("input","layer")

  mlist = convertMessageCodes(mes) # put message codes into concept layer
  rc = getLayer("concepts")
  lay = layerList[[rc]]
  lay$messages <- lapply(mlist,function(m) makeMessages(m,NumRoles+1,NumConcepts,TRUE) )
  lay$sentlen = slen
  class(lay) <- c("message","layer")
  layerList[[rc]] <<- lay

  cr = getLayer("croles")   # put reverse messages in croles layer
  lay = layerList[[cr]]
  lay$messages <- lapply(mlist,function(m) makeMessages(m,NumConcepts+1,NumRoles,FALSE) )
  lay$sentlen = slen
  class(lay) <- c("message","layer")
  layerList[[cr]] <<- lay
  class(layerList[[getLayer("cconcepts")]]) <<- c("output","layer")
}
```

Dual-path R Code: Training

```
# sample from full language
    sentsample = sort(sample(1:max(sentnum),max(sentnum)/3))
    sset = which(sentnum %in% sentsample)
    setInputs(Inputs[sset,],Targets[sset,],messages[sentsample],sentlen[sentsample],periodList[sset])
    forwardPass() # spread activation forward for all patterns in training set
    backpropagateError()  # back propagate error and update weights

    if (epoch %% 1000==0){ # print delta plot every 1000 epochs
      # plot MSDelta
      deltaPlot <- ggplot( history,aes(x=time,y=MSDelta, shape=Layer, colour=Layer))+geom_line()+geom_p
oint()

      if (epoch %% 2000==0){  # test model on novel sentences every 2000 epochs
        setInputs(tInputs,tTargets,tmessages,c(6,6),tPeriodList)
        modelout = addModel("output",restrict=TRUE,labels=tsentseq)
        slice$test = computeMeanCrossEntropyLoss()
        history = rbind(history,slice)
        dd = subset(modelout, pattern %in% 1:12)
        print(plotModel(dd,ylabel=vocabulary))
      }
   }
```

Dual-path R Code: Training

```
 forwardPassOne.message <- function(lay){
  tlay = lay
  lay$netinput = NULL
  lay$output = NULL
  ind = 1
  # each message is separately forward passed
  for (m in 1:length(lay$messages)){
    nind = ind + lay$sentlen[m]-1  # sentlen specifices the length
    if (nind > dim(lay$input)[1]){
      nind = dim(lay$input)[1]
    }
    tlay$input = lay$input[ind:nind,]
    tlay$weights = lay$messages[[m]]
    lay2 <- forwardPassOne.layer(tlay)
    # combine results in lay
    lay$output = rbind(lay$output,lay2$output)
    lay$netinput = rbind(lay$netinput,lay2$netinput)
    lay$weights = tlay$weights
    ind = nind+1
  }
  layerList[[lay$num]] <<- lay
}
```

Dual-Path Lab

- Change the test set
  - tmessagesSentences =c("cake ga apple o eats .","action=eats agent=cake patient=apple")
  - setInputs(tInputs,tTargets,tmessages,c(6,6),tPeriodList)
  - dd = subset(modelout, pattern %in% 1:12)
  - print(plotModel(dd,ylabel=vocabulary))
- Chang the language
  - Translate the input into another language
  - Verb-initial?
  - Free verb position?