# LING3804: Lecture Notes 5
# A Model of Generalization (Learning)

We saw how pre-specified neural networks can make inferences, e.g. to predict words in sequences.

This lecture will show how neural networks can be automatically specified from data (i.e. learned).

We start with individual neurons.

## Contents


5.1 Difference between memory and generalization/learning . . . . . 1
5.2 Overview of learning mechanism . . . . . 2
5.3 Training process . . . . . 2
5.4 Extra: proof of optimality of single neuron training . . . . . 5
5.5 Matrix product notation . . . . . 5
5.6 Example . . . . . 7


Language skills allow us to predict meanings from utterances.

These predictions are generalized to allow us to learn new words.

These predictions may be learned like other kinds of generalized predictions, through repetition.

- prey come out when it rains
- predators don't come out when it's dark
- when a stick is in an anthill, ants crawl up it
- words don't begin with *nd*
- subjects come before predicates

### 5.1 Difference between memory and generalization/learning

Brains learn to predict things (prey/predator behavior, word segmentation/order) through repetition.

We already covered how cued associations form by firing of pre-synaptic and post-synaptic neurons.

But that just explains memory:

- Lowly Worm came of his burrow Sep 26, 2007, during a light shower.
- Mommy said hand me a shovel.

It does not explain how **generalizations** are formed:

- Prey come out when it rains.

- words don't begin with *nd*.

## 5.2 Overview of learning mechanism

Generalizations can be learned by individual neurons through repeated refinement of predictions.

- **remember** the kind of events you didn't predict or just barely did (esp. borderline cases)

  (there is some evidence that such surprisals are stored in associative memory)
- **replay** these events over and over in your mind:
  - start with any initial synaptic weights
  - proportionately increase synaptic weights of active pre-synaptic neurons for false negatives
  - proportionately decrease synaptic weights of active pre-synaptic neurons for false positives
- seems to happen during sleep [Diekelmann & Born, 2010].

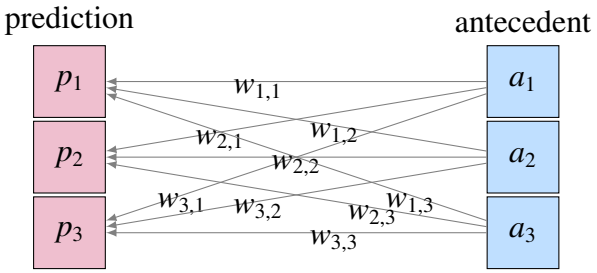
This is essentially logistic regression.

Specifically, we will assume this kind of learning is used to define certain features of mental states.

(For example, 'type' features of elementary predications, used for classifying events.)

## 5.3 Training process

Remember our model of neural activation $p_j = f(\sum_i a_i \cdot w_{i,j})$:




This model can be trained by exposing it to positively- and non-positively-labeled training examples (in particular, labeled by ones and zeros). In human language learning, positively-labeled examples might represent instances where words are found to break at a particular position or not to break at that position. Training proceeds by initializing the weights $w_{1,j}, \ldots, w_{I,j}$ (typically to random values), then iterating over the training examples and modifying the weights to improve the fit of the prediction to labeled values of the training examples. At each iteration, these weights are adjusted upward or downward based on the difference between the labeled values of the training examples and the predicted value $p_j$ of the neural model, given the values of the independent variables (features, or pre-synaptic activations) in the training examples.

Graphically (if we have multiple predicted neurons):

Get a prediction: multiply antecedent activations by current weights and apply threshold function:

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \stackrel{\text{def}}{=} f\left( \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & a_1 \\ w_{2,1} & w_{2,2} & w_{2,3} & a_2 \\ w_{3,1} & w_{3,2} & w_{3,3} & a_3 \end{bmatrix} \right)$$

Then get error: subtract prediction from remembered correct output (add with negative weight):

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} - \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Then update the weights by forming cued associations between error and antecedent activations:

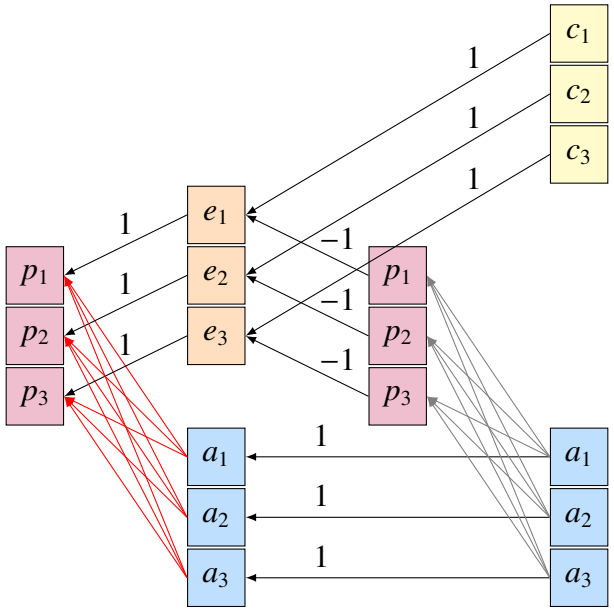
$$\begin{bmatrix} w'_{1,1} & w'_{1,2} & w'_{1,3} \\ w'_{2,1} & w'_{2,2} & w'_{2,3} \\ w'_{3,1} & w'_{3,2} & w'_{3,3} \end{bmatrix} \stackrel{\text{def}}{=} \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \\ w_{3,1} & w_{3,2} & w_{3,3} \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}$$

This repeats for several events stored as antecedent and correct output patterns in memory.

Cycles of weight adjustment for all training examples ('epochs') then repeat until adjustments stop.

This is called **convergence**.

Here's a simple implementation of this as a neural network, unrolled through time:



(Here the $p$ neurons have a softmax threshold function, explained below; all others are linear.)

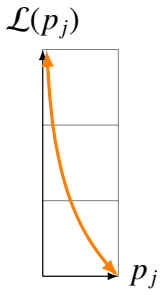
Formally: given $w_{1,1\,\ldots\,I,J} \in \mathbb{R}^{I \times J}$ and $\mathcal{D} \subseteq \mathbb{R}^{I+J}$, the updated weight $w'_{i,j}$ after each iteration is:

$$w'_{i,j} = w_{i,j} + \sum_{\langle a_1,\ldots,a_I,c_1,\ldots,c_J\rangle \in \mathcal{D}} \underbrace{\underbrace{\underbrace{\left(c_j - \underbrace{f_j\Big(\sum_{i'} w_{i',1} \cdot a_{i'}, \ldots, \sum_{i'} w_{i',J} \cdot a_{i'}\Big)}_{\text{prediction a.k.a. } p_j}\right)}_{\text{error a.k.a. } e_j} \cdot a_i}_{\text{error} \times \text{pre-synaptic activation}}}$$

It turns out these weight adjustments are guaranteed to converge to optimal (best possible) weights! Using a particular **threshold** function, it points toward the minimum of a particular **loss** function.

The loss function is the **negative log** of predicted probability of the correct output, given the input:

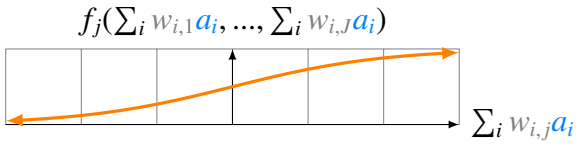
$$\mathcal{L}(p_j) = -\ln p_j$$




It has a steeper decline as the probability decreases, indicating a larger adjustment is needed.

The threshold is a **softmax** function that turns any set of activations into a probability distribution. Here's the softmax for one of $J = 2$ neurons in a distribution when the other's activation is zero:

$$p_j = f_j\Big(\sum_i w_{i,1} a_i, \ldots, \sum_i w_{i,J} a_i\Big) = \frac{e^{\sum_i w_{i,j} a_i}}{\sum_k e^{\sum_i w_{i,k} a_i}}$$




(this activation function needs ***all*** the weighted sums, because the distribution has to sum to one)
(biology may do this using ***interneurons***, which inhibit competing neurons in cortical clusters).
It's smooth, unlike step functions, so at every point we know the direction to decrease our loss.

These loss and threshold functions combine to define the above simple weight adjustment rule.

## 5.4 Extra: proof of optimality of single neuron training

Below is a proof of this optimality, assuming neuron 4 is an arbitrary one of two predicted neurons.

It uses the derivative of the loss function, which is its slope, to show which weights decrease it.

If neuron 4 indicates the recalled correct value and neuron 5 indicates an alternative value:

$$\begin{aligned}
\frac{\partial}{\partial w_{i,4}} - \ln\left(\frac{e^{w_{i,4}a_i}}{e^{w_{i,4}a_i} + e^{w_{i,5}a_i}}\right) &= \frac{\partial}{\partial w_{i,4}} - \ln e^{w_{i,4}a_i} + \ln(e^{w_{i,4}a_i} + e^{w_{i,5}a_i}) && \text{log of fraction} \\
&= \frac{\partial}{\partial w_{i,4}} - w_{i,4}a_i + \ln(e^{w_{i,4}a_i} + e^{w_{i,5}a_i}) && \text{log of exponential} \\
&= \left(\frac{\partial}{\partial w_{i,4}} - w_{i,4}a_i\right) + \left(\frac{\partial}{\partial w_{i,4}} \ln(e^{w_{i,4}a_i} + e^{w_{i,5}a_i})\right) && \text{derivative of sum} \\
&= -a_i + \left(\frac{\partial}{\partial w_{i,4}} \ln(e^{w_{i,4}a_i} + e^{w_{i,5}a_i})\right) && \text{derivative of product} \\
&= -a_i + \left(\frac{1}{e^{w_{i,4}a_i} + e^{w_{i,5}a_i}} \frac{\partial}{\partial w_{i,4}} e^{w_{i,4}a_i} + e^{w_{i,5}a_i}\right) && \text{derivative of log} \\
&= -a_i + \left(\frac{e^{w_{i,4}a_i}}{e^{w_{i,4}a_i} + e^{w_{i,5}a_i}} \frac{\partial}{\partial w_{i,4}} w_{i,4}a_i\right) && \text{derivative of exponential} \\
&= -a_i + \left(\frac{e^{w_{i,4}a_i}}{e^{w_{i,4}a_i} + e^{w_{i,5}a_i}} a_i\right) && \text{derivative of product} \\
&= -\left(1 - \frac{e^{w_{i,4}a_i}}{e^{w_{i,4}a_i} + e^{w_{i,5}a_i}}\right) a_i = -(1 - p_j)a_i && \text{distributive axiom}
\end{aligned}$$

If neuron 4 ***is not*** the recalled correct neuron and neuron 5 ***is*** the recalled correct neuron:

$$\frac{\partial}{\partial w_{i,4}} - \ln\left(\frac{e^{w_{i,5}a_i}}{e^{w_{i,4}a_i} + e^{w_{i,5}a_i}}\right) = -\left(0 - \frac{e^{w_{i,4}a_i}}{e^{w_{i,4}a_i} + e^{w_{i,5}a_i}}\right) a_i = p_j a_i$$

So, predicting $p_j$ from $a_i$, loss declines in the direction of $(1-p_j)a_i$ if $c_j$=1, and $-p_j a_i$ if $c_j$=0.

Intuitively: if the probability should be one, increase weights of positive inputs, otherwise decrease.

## 5.5 Matrix product notation

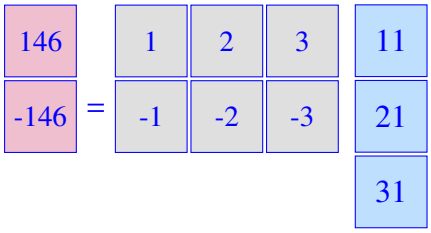
We introduce a standard notation here to talk about learning from multiple examples.

Recall that we could generalize a weighted sum (inner product), as in a neural network:

| | | | | |
|---|---|---|---|---|
| 146 | = | 1 | 2 | 3 | 11 |
| | | | | | 21 |
| | | | | | 31 |

$$146 = 11 + 42 + 93 = (1 \times 11) + (2 \times 21) + (3 \times 31)$$

to a cued association between multi-neuron states, using matrix notation:

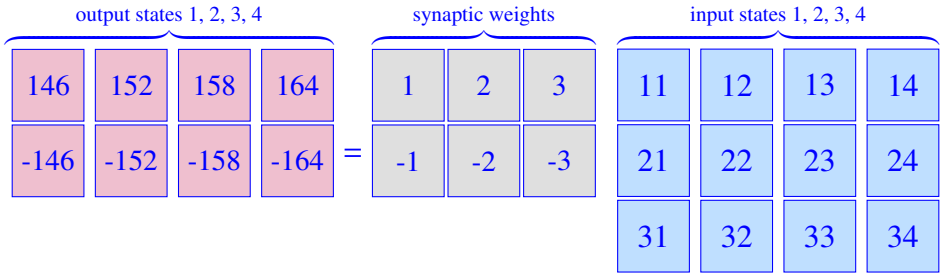



$$-146 = -11 + (-42) + (-93) = (-1 \times 11) + (-2 \times 21) + (-3 \times 31)$$

which simply stacks up the weights vertically for each output neuron

(the top output neuron uses the top weights, the bottom output neuron uses the bottom weights, etc).

Well, similarly, we can horizontally line up (multi-neuron) activation states for multiple inputs:




$$152 = 12 + 44 + 96 = (1 \times 12) + (2 \times 22) + (3 \times 32)$$
$$158 = 13 + 46 + 99 = (1 \times 13) + (2 \times 23) + (3 \times 33)$$
$$164 = 14 + 48 + 102 = (1 \times 14) + (2 \times 24) + (3 \times 34)$$

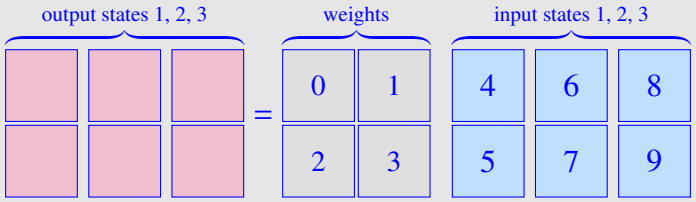
to produce a line of (multi-neuron) output activations

(the first output column uses the first input activations, the second uses the second, etc).

This is called a **matrix product**. We'll use it to learn from multiple examples.

**Practice 5.1:**

Complete the matrix product (you can just write the formula, no need to do the arithmetic):



## 5.6 Example

Here is an example training process.

Suppose we encounter six speech sound sequences, some of which lead to yummy cookies:

| | |
|---|---|
| /kʊkitɑim/ | a cookie is forthcoming |
| /ɑimgəntəkʊkərəʊst/ | no cookie is forthcoming |
| /wɑnðəməŋki/ | no cookie is forthcoming |
| /wɑntəkʊki/ | a cookie is forthcoming |
| /ɑilkʊkɪtəθmɪt/ | no cookie is forthcoming |
| /wɑnðədɔŋki/ | no cookie is forthcoming |

and we have pre-synaptic activations for speech sounds, and post-synaptic activations for 'yummy':

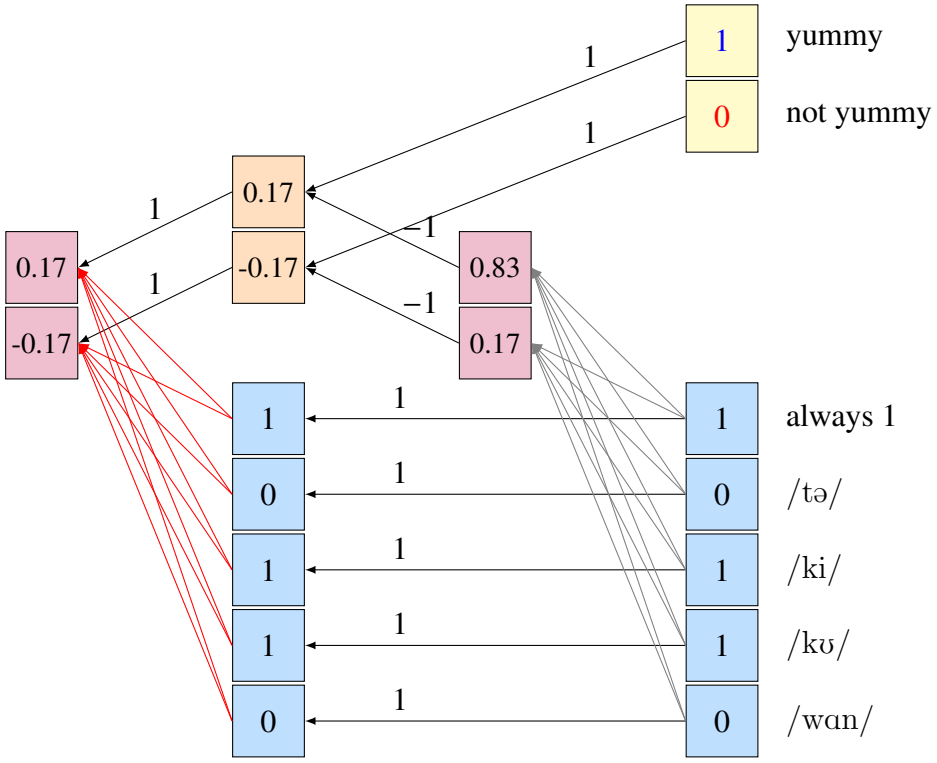
| | /kʊkitɑim/ | /ɑimgəntəkʊkərəʊst/ | /wɑnðəməŋki/ | /wɑntəkʊki/ | /ɑilkʊkɪtəθmɪt/ | /wɑnðədɔŋki/ |
|---|---|---|---|---|---|---|
| yummy | 1 | 0 | 0 | 1 | 0 | 0 |
| not yummy | 0 | 1 | 1 | 0 | 1 | 1 |
| always 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| /tə/ | 0 | 1 | 0 | 1 | 1 | 0 |
| /ki/ | 1 | 0 | 1 | 1 | 0 | 1 |
| /kʊ/ | 1 | 1 | 0 | 1 | 1 | 0 |
| /wɑn/ | 0 | 0 | 1 | 1 | 0 | 1 |

Intuitively, we should expect to find that $a_4(/\text{kʊ}/)$ and $a_3(/\text{ki}/)$ are good indicators of 'yummy'.

Start with random initial weights:

| | always 1 | /tə/ | /ki/ | /kʊ/ | /wɑn/ |
|---|---|---|---|---|---|
| yummy | 1.66 | 1.20 | -0.79 | -0.09 | -0.06 |
| not yummy | -1.66 | -1.20 | 0.79 | 0.09 | 0.06 |

Then make predictions and increase synaptic weights of active pre-synaptic neurons by the error. . .

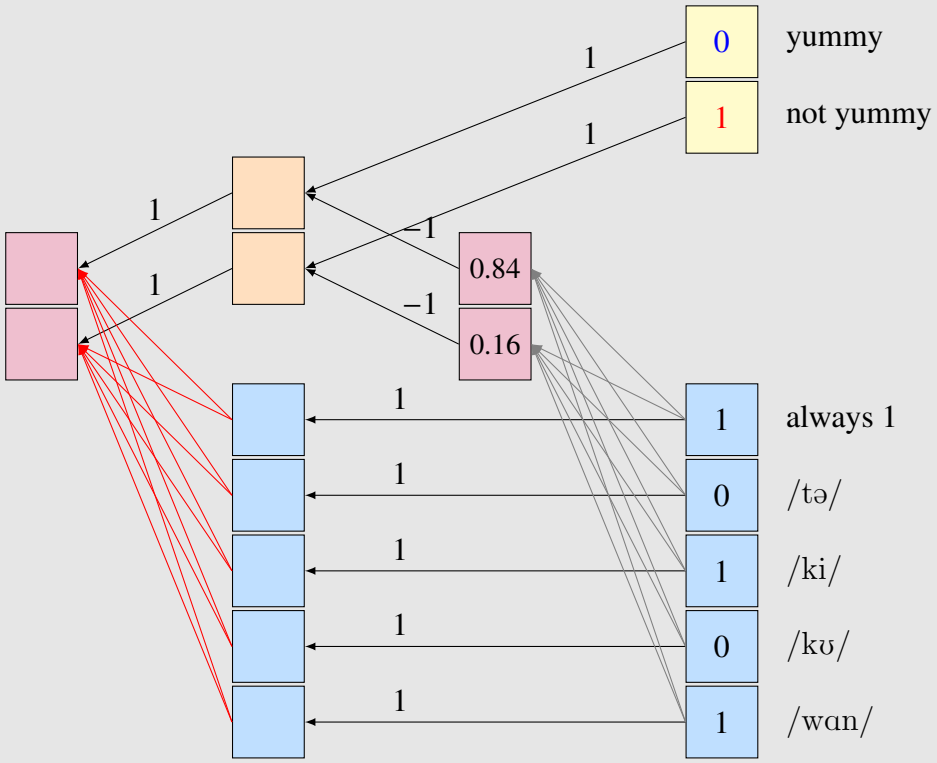



The resulting cued association (outer product) produces this adjustment, to be added to the weights:

| | always 1 | /tə/ | /ki/ | /kʊ/ | /wɑn/ |
|---|---|---|---|---|---|
| yummy | 0.17 | 0 | 0.17 | 0.17 | 0 |
| not yummy | -0.17 | 0 | -0.17 | -0.17 | 0 |

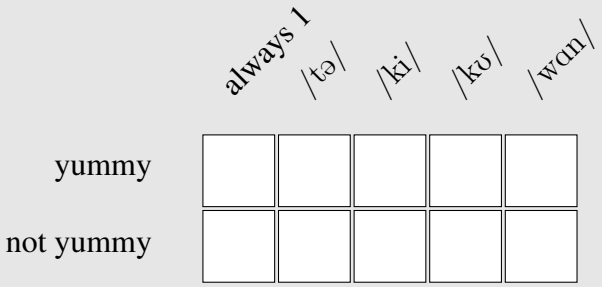
**Practice 5.2:**

Complete the weight adjustment for the following example, where all threshold functions are linear except gray connections are softmax and red connections are cued association formation:




**Practice 5.3:**

Calculate the weight adjustment for the above cued association formation (red edges):



If we compute all the adjustments in parallel (not exactly the same result as sequential) we get:

predictions for training examples

| | | | | | |
|---|---|---|---|---|---|
| .826 | .996 | .835 | .979 | .996 | .835 |
| .174 | .004 | .165 | .021 | .004 | .165 |

$= f$ (synaptic weights) (input states for training examples)

| | | | | | |
|---|---|---|---|---|---|
| 1.66 | 1.20 | -0.79 | -0.09 | -0.06 | |
| -1.66 | -1.20 | 0.79 | 0.09 | 0.06 | |

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 |

errors for training examples

| | | | | | |
|---|---|---|---|---|---|
| .173 | -.996 | -.835 | .021 | -.996 | -.835 |
| -.173 | .996 | .835 | -.021 | .996 | .835 |

$=$ (correct outputs for training examples)

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |

$-$ (predictions for training examples)

| | | | | | |
|---|---|---|---|---|---|
| .826 | .996 | .835 | .979 | .996 | .835 |
| .174 | .004 | .165 | .021 | .004 | .165 |

adjustments to synaptic weights

| | | | | | |
|---|---|---|---|---|---|
| -3.47 | -1.97 | -1.47 | -1.80 | -1.65 | |
| 3.47 | 1.97 | 1.47 | 1.80 | 1.65 | |

errors for training examples

| | | | | | |
|---|---|---|---|---|---|
| .173 | -.996 | -.835 | .021 | -.996 | -.835 |
| -.173 | .996 | .835 | -.021 | .996 | .835 |

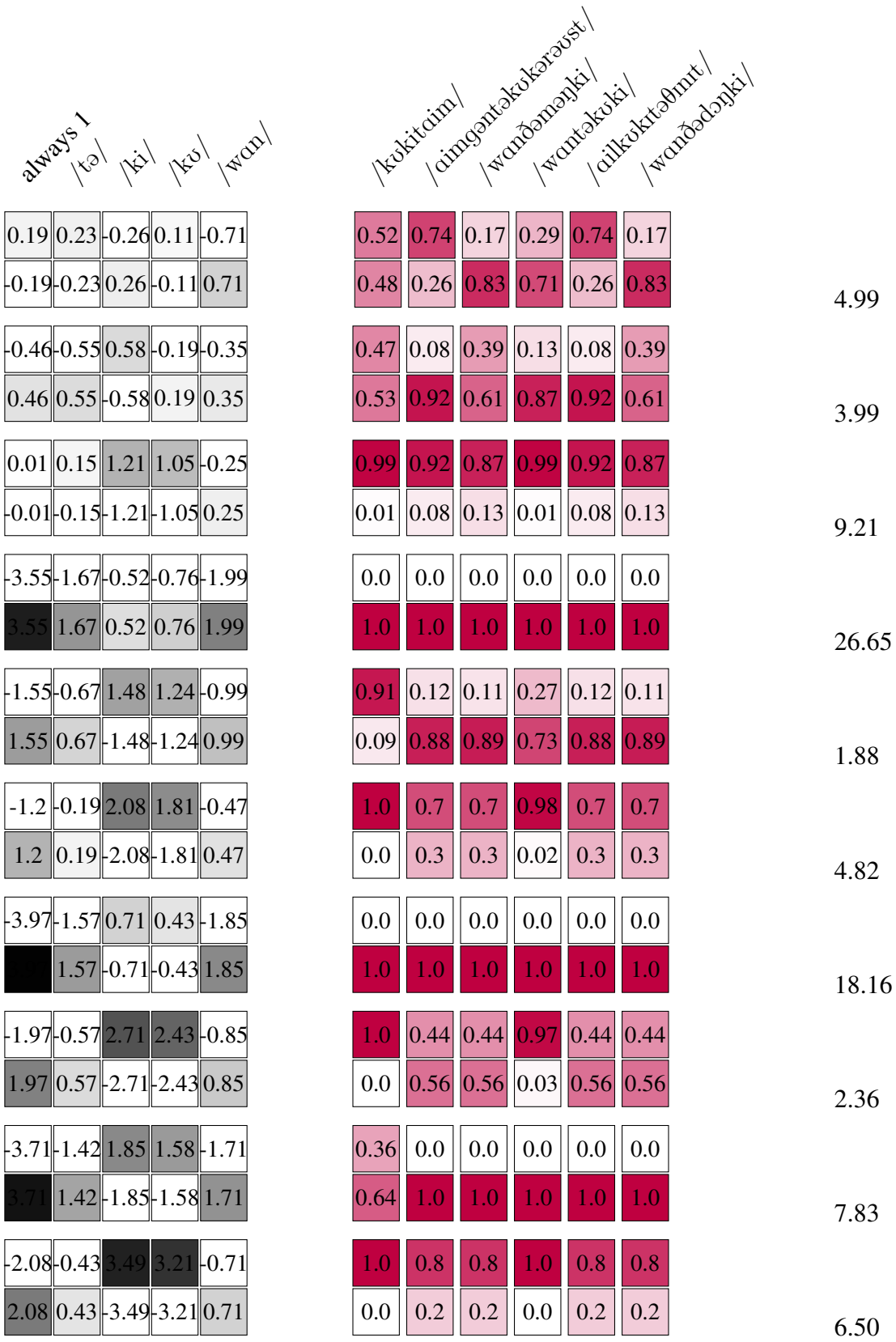
inputs for training examples (transposed)

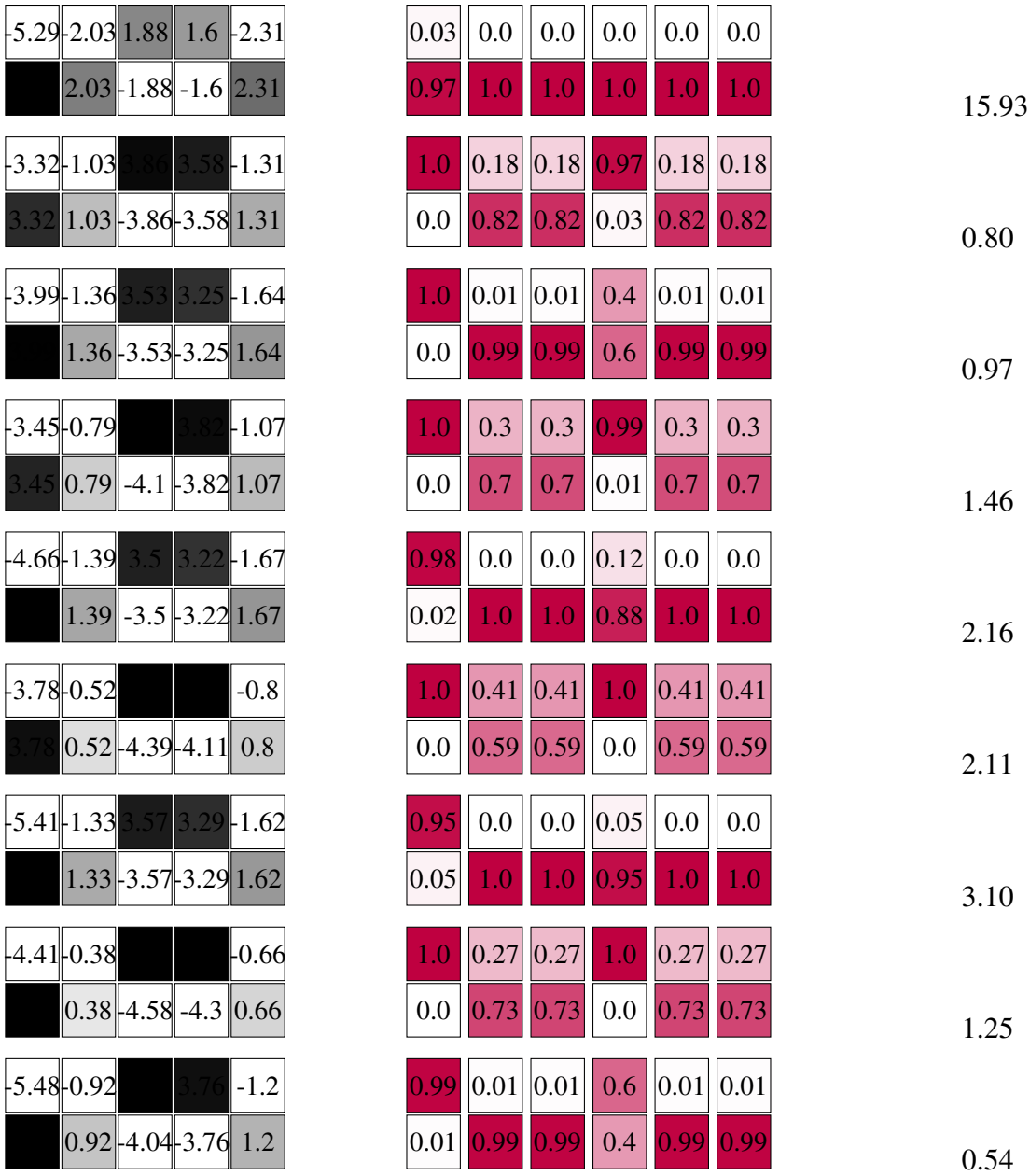
| | | | | |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |

This adjustment gives us the following weights:

| | always 1 | /tə/ | /ki/ | /kʊ/ | /wɑn/ |
|---|---|---|---|---|---|
| yummy | -1.81 | -0.77 | -2.26 | -1.89 | -1.71 |
| not yummy | 1.81 | 0.77 | 2.26 | 1.89 | 1.71 |

And the next few adjustments give us the below weights, predictions and loss values:






After a dozen or so iterations of all training examples, the predictions are generally correct, and the weights are concentrated on $a_4$(/kʊ/) and $a_3$(/ki/), matching our intuitions.

Now we can **generalize** to predict responses for stimuli not in our training set:

- /gɛtəbæibi/ : $\frac{e^{-5.8-0.92+0+0+0}}{e^{-5.8-0.92+0+0+0}+e^{5.8+0.92+0+0+0}} = .001$ – not yummy,
- /huwɑntsəkʊki/ : $\frac{e^{-5.8+0+4.04+3.76+0}}{e^{-5.8+0+4.04+3.76+0}+e^{5.8+0-4.04-3.76+0}} = 0.982$ – yummy!

This process is called **gradient descent**.

Training distributions over observations from generative sequence models is **autoregression**.

**Practice 5.4:**

What is the probability, according to the last reported model, of /kʊwɑn/ yielding a cookie (you can just write the formula, no need to do the arithmetic)?

# References

[Diekelmann & Born, 2010] Diekelmann, S. & Born, J. (2010). The memory function of sleep. *Nature Reviews Neuroscience*, 11, 114–126.