Neural Networks for Machine Learning - Geoffrey Hinton Notes

Yan JIN

December 4, 2016

Contents

1 Introduction

2 The Perceptron learning procedure

2.1 Quiz

1. If the output of a model is given by $y = f(\mathbf{x}; W)$, then which of the following choices for f are most appropriate when the task is binary classification?

- [ ] Logistic sigmoid
- [ ] Binary threshold
- [ ] Linear threshold
- [ ] Linear

Fig. 1: Exercise 02-01

2. After learning using the Perceptron algorithm, how easy is it to express the learned weight vector in terms of the input vectors and the initial weight vector? Assume the input vectors have real-valued components.

- [ ] It requires only one integer per training case.
- [ ] It requires real numbers.
- [ ] It is impossible.
- [ ] It requires one bit per training case.

Fig. 2: Exercise 02-02

3. Suppose we are given three data points:

$$\begin{array}{c} \mathbf{x} \rightarrow t \\ 1,0 \rightarrow 1 \\ 1,1 \rightarrow 1 \\ 0,1 \rightarrow 0 \end{array}$$

Furthermore, we are given the following weight vector (where the bias is set to 0):

$$\mathbf{w} = (0, -3)$$

Let $||\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}||_2$ be the distance between the weight vectors at iteration t and iteration $t-1$ of the perceptron learning algorithm. Here, for a given 2D vector $\mathbf{v}$, $||\mathbf{v}||_2 = \sqrt{v_1^2 + v_2^2}$ (this is also called the Euclidean norm). What is the maximum amount by which the weight vectors can change between successive iterations? Note that in this example we are *not* learning the bias.

- $2\sqrt{2}$
- 1
- $\sqrt{2}$
- 2

Fig. 3: Exercise 02-03

4. Suppose that we have a perceptron with weight vector $\mathbf{w}$ and we create a new set of weights $\mathbf{w}^* = c\mathbf{w}$ by scaling $\mathbf{w}$ by some positive constant c.

Assume that the bias is zero.

True or false: if the perceptron now uses $\mathbf{w}^*$ instead then it's classification decisions might change (that is, we have moved the classification boundary).

- True
- False

Fig. 4: Exercise 02-04

5. Suppose that we have a perceptron with weight vector $\mathbf{w}$ and we create a new set of weights $\mathbf{w}^* = \mathbf{w} + \mathbf{c}$ by adding some constant vector $\mathbf{c}$ to $\mathbf{w}$. Assume that the bias is zero.

True or false: if the perceptron now uses $\mathbf{w}^*$ instead then it's classification decisions might change (that is, we have moved the classification boundary).

- ○ False
- ○ True

Fig. 5: Exercise 02-05

6. Suppose we are given four training cases:

$$\begin{array}{c} \mathbf{x} \rightarrow t \\ 1,1 \rightarrow 1 \\ 1,0 \rightarrow 0 \\ 0,1 \rightarrow 0 \\ 0,0 \rightarrow 1 \end{array}$$

It is impossible for a binary threshold unit to produce the desired target outputs for all four cases. Now suppose that we add an extra input dimension so that each of the four input vectors consists of three numbers instead of two.

Which of the following ways of setting the value of the extra input will create a set of four input vectors that is linearly separable (i.e. that can be given the right target values by a binary threshold unit with appropriate weights and bias).

- ☐ Make the third value be 1 for one of the four input vectors and 0 for the other three.
- ☐ Make the third value of each input vector be the same as the target value for that input vector.
- ☐ Make the third value of each input vector be the opposite of the first value (i.e. use 1 if the first value is 0 and 0 if the first value is 1)
- ☐ Make the third value of each input vector be the same as the first value.

Fig. 6: Exercise 02-06

7. Brian wants to use a neural network to predict the price of a stock tomorrow given today's price and the price over the last 10 days. The inputs to this network are price over the last 10 days and the output is tomorrow's price. The hidden units in this network receive information from the layer below, transmit information to the layer above and do not send information within the same layer. Is this an example of a feed-forward network or a recurrent network?

- ○ Feed-forward
- ○ Recurrent

Fig. 7: Exercise 02-07

8. Brian and Andy are having an argument about the perceptron algorithm. They have a dataset that the perceptron cannot seem to classify (that is, it fails to converge to a solution). Andy reasons that if he could collect more examples, that might solve the problem by making the data set linearly separable and then the perceptron algorithm will converge. Brian claims that collecting more examples will not help. Which one of them is correct?

- Andy
- Brian

Fig. 8: Exercise 02-08

2.2 My Answers

02-01: Logistic sigmoid, Binary threshold;

02-02: 1st, 4th;

For every training case, the update to the weight matrix is determined by the output of the perceptron unit, this is 1 bit of information. However, we can also represent the model with an integer that stores whether we added / subtracted or left the weight matrix unchanged when we looked at that example (-1, 0, +1).

02-03: 3rd;

02-04: False;

02-05: True;

02-06: 1st, 2nd;

02-07: Feed-forward;

02-08: Brian;

3 The backpropagation learning procedure

3.1 Quiz

1. Which of the following neural networks are examples of a feed-forward neural network?

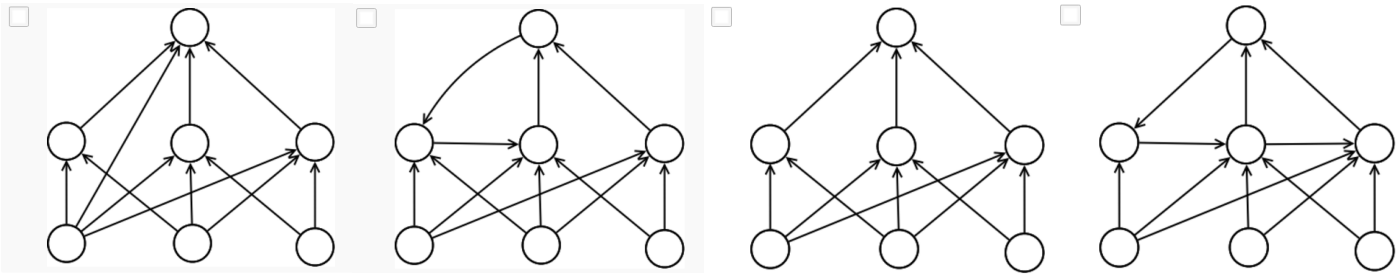


Fig. 9: Exercise 03-01

2. Consider a neural network with only one training case with input $\mathbf{x} = (x_1, x_2, \ldots, x_n)^\top$ and correct output t. There is only one output neuron, which is linear, i.e. $y = \mathbf{w}^\top \mathbf{x}$ (notice that there are no biases). The loss function is squared error. The network has no hidden units, so the inputs are directly connected to the output neuron with weights $\mathbf{w} = (w_1, w_2, \ldots, w_n)^\top$. We're in the process of training the neural network with the backpropagation algorithm. What will the algorithm add to w_i for the next iteration if we use a step size (also known as a learning rate) of ϵ?

- ☐ $\epsilon(t - \mathbf{w}^\top \mathbf{x})x_i$
- ☐ x_i
- ☐ $\epsilon(\mathbf{w}^\top \mathbf{x} - t)x_i$
- ☐ x_i if $\mathbf{w}^\top \mathbf{x} > t$

  $-x_i$ if $\mathbf{w}^\top \mathbf{x} \leq t$

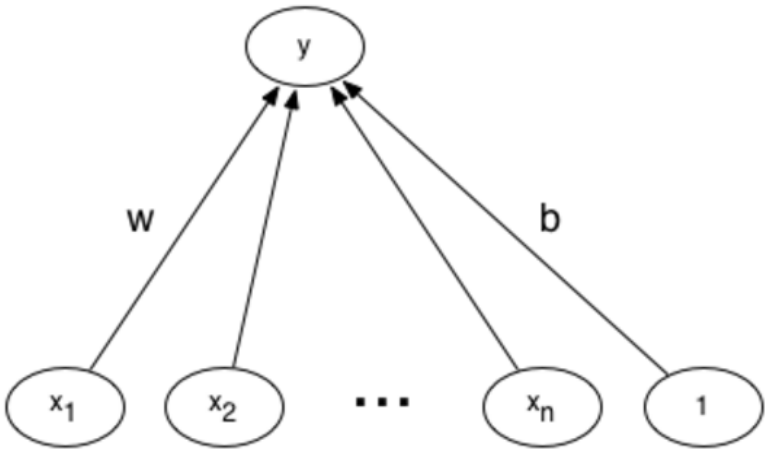
Fig. 10: Exercise 03-02

3. Suppose we have a set of examples and Brian comes in and duplicates every example, then randomly reorders the examples. We now have twice as many examples, but no more information about the problem than we had before. If we do not remove the duplicate entries, which one of the following methods will *not* be affected by this change, in terms of the computer time (time in seconds, for example) it takes to come close to convergence?

- ☐ Full-batch learning.
- ☐ Online learning, where for every iteration we randomly pick a training case.
- ☐ Mini-batch learning, where for every iteration we randomly pick 100 training cases.

Fig. 11: Exercise 03-03

4. Consider a linear output unit versus a logistic output unit for a feed-forward network with *no hidden layer* shown below. The network has a set of inputs x and an output neuron y connected to the input by weights w and bias b.



We're using the squared error cost function even though the task that we care about, in the end, is binary classification. At training time, the target output values are 1 (for one class) and 0 (for the other class). At test time we will use the classifier to make decisions in the standard way: the class of an input x according to our model **after training** is as follows:

$$\text{class of } x = \begin{cases} 1 \text{ if } w^T x + b \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

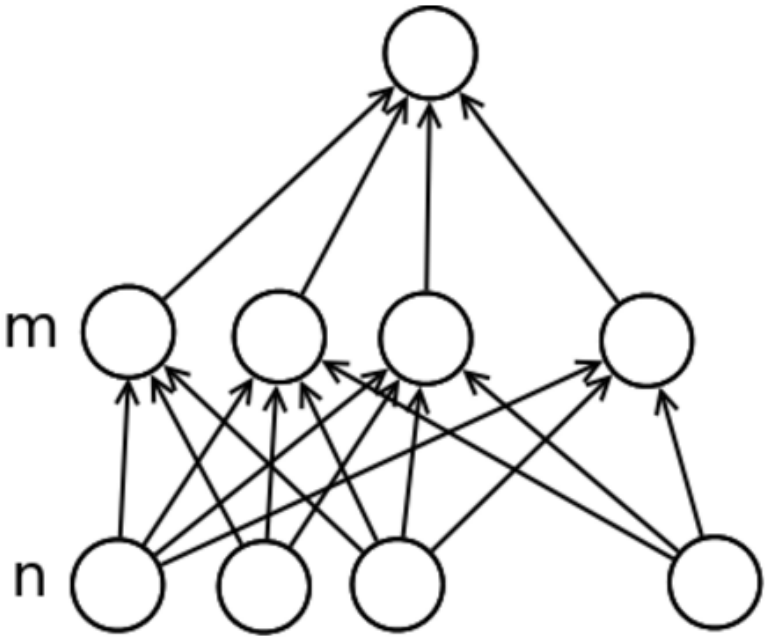
Note that we will be training the network using y, but that the decision rule shown above will be the same at *test* time, regardless of the type of output neuron we use for training.

Which of the following statements is true?

- [ ] For a logistic unit, the derivatives of the error function with respect to the weights can have unbounded magnitude, while for a linear unit they will have bounded magnitude.
- [ ] The error function (the error as a function of the weights) for both types of units will form a quadratic bowl.
- [ ] At the solution that minimizes the error, the learned weights are always the same for both types of units; they only differ in how they get to this solution.
- [ ] Unlike a logistic unit, using a linear unit will penalize us for getting the answer right too confidently.

Fig. 12: Exercise 03-04

5. Consider a neural network with one layer of **linear** hidden units (intended to be fully connected to the input units) and a logistic output unit. Suppose there are n input units and m hidden units. Which of the following statements are true? Check all that apply.



- [ ] A network with $m > n$ can learn functions that a network with $m \leq n$ cannot learn.
- [ ] Any function that can be computed by such a network can also be computed by a network without a hidden layer.
- [ ] A network with $m > n$ has more learnable parameters than a network with $m \leq n$ (for a fixed value of n).
- [ ] There is a value for m, such that there are functions that this network can learn to compute and that a network without a hidden layer cannot learn to compute.

Fig. 13: Exercise 03-05

6. Brian wants to make his feed-forward network (with no hidden units) using a **logistic** output neuron more powerful. He decides to combine the predictions of two networks by averaging them. The first network has weights w_1 and the second network has weights w_2. The predictions of this network for an example x are therefore:

$$y = \frac{1}{2}\frac{1}{1+e^{-z_1}} + \frac{1}{2}\frac{1}{1+e^{-z_2}} \text{ with } z_1 = w_1^T x \text{ and } z_2 = w_2^T x.$$

Can we get the exact same predictions as this combination of networks by using a single feed-forward network (again with no hidden units) using a **logistic** output neuron and weights $w_3 = \frac{1}{2}(w_1 + w_2)$?

- [ ] Yes
- [ ] No

Fig. 14: Exercise 03-06

3.2 My Answers

03-01: 1st, 3rd;

03-02: 1st;

03-03: 2nd;

3rd: After Brian's intervention, most mini-batches will contain duplicates and will therefore provide less information.

Full-batch learning needs to look at every example before taking a step, therefore each step will be twice as expensive. Online learning only looks at one example at a time so each step has the same computational cost as before. On expectation, online learning would make the same progress after looking at half of the dataset as it would have if Brian has not intervened.

Although this example is a bit contrived, it serves to illustrate how online learning can be advantageous when there is a lot of redundancy in the data.

03-04: 4th;

If the target is 1 and the prediction is 100, the logistic unit will squash this down to a number very close to 1 and so we will not incur a very high cost. With a linear unit, the difference between the prediction and target will be very large and we will incur a high cost as a result, despite the fact that we get the classification decision correct.

03-05: 2nd, 3rd;

2nd: **Linear** hidden units don't add modeling capacity to the network.

ATTENTION: If Change the hidden units from **linear** to **logistic**:

- [x] A network with $m > n$ has more learnable parameters than a network with $m \leq n$ (for a fixed value of n).

Correct
The bulk of the learnable parameters is in the connections from the input units to the hidden units. There are $m \cdot n$ learnable parameters there.

- [x] If $m > n$, this network can learn more functions than if m is less than n (with n being the same).

Correct
This is quite a flexible model. It can learn many functions that cannot be learned without the use of a hidden layer. The nonlinearity in the hidden layer is essential.

- [ ] Any function that can be learned by such a network can also be learned by a network without any hidden layers (with the same inputs).

Un-selected is correct

- [x] As long as $m \geq 1$, this network can learn to compute any function that can be learned by a network without any hidden layers (with the same inputs).

This should not be selected
If the weights into the hidden layer are very small, and the weights out of it are large (to compensate), then the hidden units behave like linear units, which makes lots of things possible.

Fig. 15: Exercise 03-05-ans

03-06: No;

4 Learning feature vectors for words

4.1 Learning to predict the next word

4.2 A brief diversion into cognitive science

4.3 Another diversion: The softmax output function

4.4 Neuro-probabilistic language models

Only acoustic not good, so the meaning of the words is important for speech recognition.

The standard "Trigram" method have limits;

Bengio's neural net comes to play. Similar to the family trees network, but real and bigger. Slightly worse than "Trigram", but combined with "Trigram" and improved things. Since then, this neural net improved and now better than "Trigram".

Problem with having 100,000 output words: easy to overfitting, if not many training cases;

4.5 Ways to deal with the large number of possible outputs in neuro-probabilistic language models

4.6 Quiz

1. The squared error cost function with n linear units is equivalent to:

Clarification:

Let's say that a network with n linear output units has some weights w. w is a matrix with n columns, and w_i indexes a particular column in this matrix and represents the weights from the inputs to the i^{th} output unit.

Suppose the target for a particular example is j (so that it belongs to class j in other words).

The squared error cost function for n linear units is given by:

$\frac{1}{2}\sum_{i=1}^{n}(t_i - w_i^T x)^2$

where t is a vector of zeros except for 1 in index j.

The cross-entropy cost function for an n-way softmax unit is given by:

$$-\log\left(\frac{\exp\left(w_j^T x\right)}{\sum_{i=1}^{n}\exp\left(w_i^T x\right)}\right) = -w_j^T x + \log\left(\sum_{i=1}^{n}\exp\left(w_i^T x\right)\right)$$

Finally, n logistic units would compute an output of $\sigma(w_i^T x) = \frac{1}{1+\exp(-w_i^T x)}$ *independently* for each class i. Combined with the squared error the cost would be:

$\frac{1}{2}\sum_{i=1}^{n}(t_i - \sigma(w_i^T x))^2$

Where again, t is a vector of zeros with a 1 at index j (assuming the true class of the example is j).

Using this same definition for t, the cross-entropy error for n logistic units would be the sum of the individual cross-entropy errors:

$$-\sum_{i=1}^{n} t_i \log(\sigma(w_i^T x)) + (1 - t_i) \log(1 - \sigma(w_i^T x))$$

For any set of weights w, the network with n linear output units will have some cost due to the squared error (cost function). The question is now asking whether we can define a new network with a set of weights w^* using some (possibly different) cost function such that:

a) $w^* = f(w)$ for some function f

b) For every input, the cost we get using w in the linear network with squared error is the same cost that we would get using w^* in the new network with the possibly different cost function.

- [ ] The cross-entropy cost function with an n-way softmax unit.
- [ ] The cross-entropy cost function with n logistic units.
- [ ] The squared error cost function with n logistic units.
- [ ] None of the above.

Fig. 16: Exercise 04-01

2. A 2-way softmax unit (a softmax unit with 2 elements) with the cross entropy cost function is equivalent to:

Clarification:

In a network with a logistic output, we will have a single vector of weights w. For a particular example with target t (which is 0 or 1), the cross-entropy error is given by:

$-t\log\big(\sigma(w^T x)\big) - (1-t)\log\big(1-\sigma(w^T x)\big)$ where $\sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$.

The squared error if we use a single linear unit would be:

$\frac{1}{2}\left(t - w^T x\right)^2$

Now notice that another way we might define t is by using a vector with 2 elements, [1,0] to indicate the first class, and [0,1] to indicate the second class. Using this definition, we can develop a new type of classification network using a softmax unit over these two classes instead. In this case, we would use a weight *matrix* w with two columns, where w_i is the column of the i^{th} class and connects the inputs to the i^{th} output unit.

Suppose an example belonged to class j (where j is 1 or 2 to indicate [1,0] or [0,1]). Then the cross-entropy cost for this network would be:

$$-\log\left(\frac{\exp(w_j^T x)}{\exp(w_1^T x)+\exp(w_2^T x)}\right) = -w_j^T x + \log\big(\exp(w_1^T x) + \exp(w_2^T x)\big)$$

For any set of weights w, the network with a softmax output unit over 2 classes will have some error due to the cross-entropy cost function. The question is now asking whether we can define a new network with a set of weights w^* using some (possibly different) cost function such that:

a) $w^* = f(w)$ for some function f

b) For every input, the cost we get using w in the network with a softmax output unit over 2 classes and cross-entropy error is the same cost that we would get using w^* in the new network with the possibly different cost function.

- ○ A logistic unit with the cross-entropy cost function.
- ○ A 2-way softmax unit (a softmax unit with 2 elements) with the squared error cost function.
- ○ Two linear units with the squared error cost function.
- ○ None of the above.

Fig. 17: Exercise 04-02

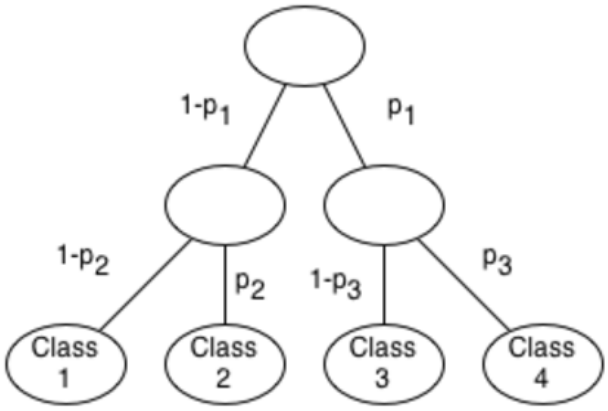
3. The output of a neuro-probabilistic language model is a large softmax unit and this creates problems if the vocabulary size is large. Andy claims that the following method solves this problem:

At every iteration of training, train the network to predict the current learned feature vector of the target word (instead of using a softmax). Since the embedding dimensionality is typically much smaller than the vocabulary size, we don't have the problem of having many output weights any more. Which of the following are correct? Check all that apply.

- [ ] In theory there's nothing wrong with Andy's idea. However, the number of learnable parameters will be so far reduced that the network no longer has sufficient learning capacity to do the task well.
- [ ] If we add in extra derivatives that change the feature vector for the target word to be more like what is predicted, it may find a trivial solution in which all words have the same feature vector.
- [ ] The serialized version of the model discussed in the slides is using the current word embedding for the output word, but it's optimizing something different than what Andy is suggesting.
- [ ] Andy is correct: this is equivalent to the serialized version of the model discussed in the lecture.

Fig. 18: Exercise 04-03

4. We are given the following tree that we will use to classify a particular example x:



In this tree, each p value indicates the probability that x will be classified as belonging to a class in the right subtree of the node at with that p was computed. For example, the probability that x belongs to Class 2 is $(1 - p_1) \times p_2$. Recall that at training time this is a very efficient representation because we only have to consider a single branch of the tree. However, at test-time we need to look over all branches in order to determine the probabilities of each outcome.

Suppose we are not interested in obtaining the exact probability of every outcome, but instead we just want to find the class with the maximum probability. A simple heuristic is to search the tree greedily by starting at the root and choosing the branch with maximum probability at each node on our way from the root to the leaves. That is, at the root of this tree we would choose to go right if $p_1 \geq 0.5$ and left otherwise.

If $p_1 = 0.45$, $p_2 = 0.6$, and $p_3 = 0.95$, then which class will the following methods report for x?

a) Evaluate the probabilities of each of the four classes (the leaf nodes) and report the class of the leaf node with the highest probability. This is the standard approach but may take quite some time.

b) The proposed alternative approach: greedily traverse the tree by choosing the branch with the highest probability and report the class of the leaf node that this finds.

- Method a) will report class 1
  Method b) will report class 4
- Method a) will report class 3
  Method b) will report class 2
- Method a) will report class 4
  Method b) will report class 2
- Method a) will report class 4
  Method b) will report class 1
- Method a) will report class 2
  Method b) will report class 2
- Method a) will report class 2
  Method b) will report class 3

Fig. 19: Exercise 04-04

5. True or false: the neural network in the lectures that was used to predict relationships in family trees had "bottleneck" layers (layers with fewer dimensions than the input). The reason these were used was to prevent the network from memorizing the training data without learning any meaningful features for generalization.

- False
- True

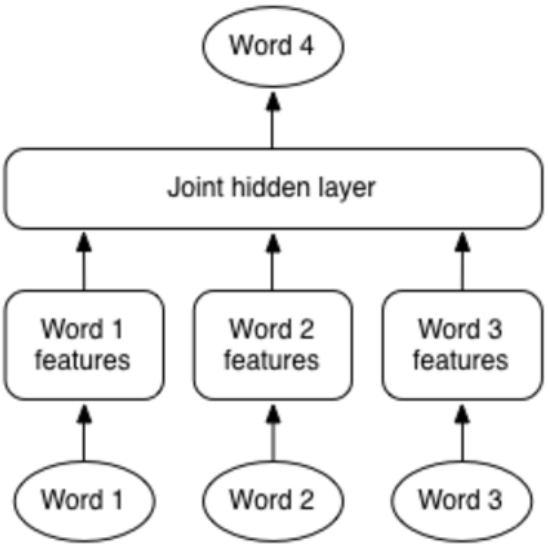
Fig. 20: Exercise 04-05

6. In the Collobert and Weston model, the problem of learning a feature vector from a sequence of words is turned into a problem of:

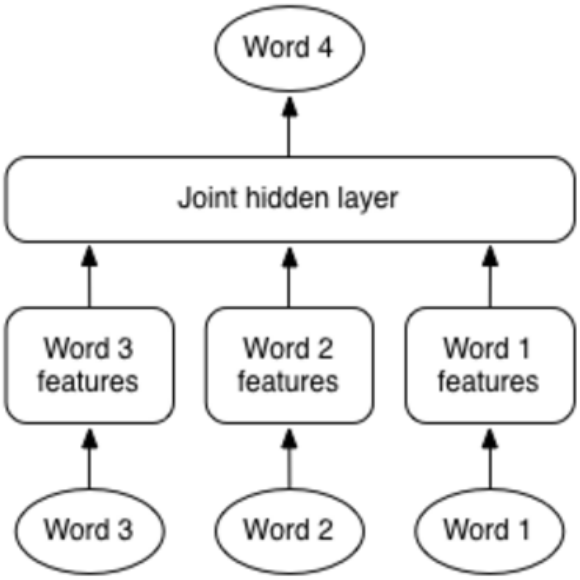
- Learning to predict the middle word in the sequence given the words that came before and the words that came after.
- Learning to predict the next word in an arbitrary length sequence.
- Learning to reconstruct the input vector.
- Learning a binary classifier.

Fig. 21: Exercise 04-06

7. Andy is in trouble! He is in the middle of training a feed-forward neural network that is supposed to predict the fourth word in a sequence given the previous three words (shown below).



Brian was supposed to get the dataset ready, but accidentally swapped the first and third word in every sequence! Now the neural network sees sequences of the form *word 3, word 2, word 1, word 4*; so what is actually being trained is this network:



Andy has been training this network for a long time and doesn't want to start over, but he is worried that the network won't do very well because of Brian's mistake. Should Andy be worried?

- Andy should not be worried. Even though the network knows that each word in the sequence is in a different place, it does not care about the ordering of the input words as long as input sequences are always shown to it in a consistent way.
- Andy should be worried. The network will not do a very good job because of Brian's mistake. We know this because changing the order will make things more difficult for a human, so for artificial neural networks it will be even more difficult.
- Andy should be worried. Even if they show the test examples in a consistent way (by swapping the order of the words in the same way that the network was trained) the performance will suffer. However, they can correct the ordering and re-train the network for a small number of iterations to fix things.
- Andy should not be worried. His feed-forward network is smart enough to automatically recognize that the input sequences were given in the incorrect order and will change the sequences back to the correct order as it's learning.

Fig. 22: Exercise 04-07

4.7 My Answers

04-01: None of the above;

04-02: 1st;

A logistic unit with the cross-entropy cost function.

Correct

With weights w_1 and w_2 we can write the softmax probability for an input x as:

$$P(\text{x belongs to class 1}) = \frac{\exp(w_1^T x)}{\exp(w_1^T x)+\exp(w_2^T x)}.$$

Dividing the top and bottom by $\exp(w_1^T x)$:

$$P(\text{x belongs to class 1}) = \frac{1}{1+\exp(w_2^T x - w_1^T x)} = \frac{1}{1+\exp(-(w_1-w_2)^T x)}.$$

So this is equivalent to a logistic unit with weights $w_1 - w_2$. Since the cross-entropy is the negative log-probability, it will be the same for both representations.

Fig. 23: Exercise 04-02-ans

04-03: 2nd, 3rd;

04-04: Method a) will report class 4; Method b) will report class 2;

04-05: True;

04-06: Learning a binary classifier;

Learning to predict the middle word in the sequence given the words that came before and the words that came after.

This should not be selected

It wasn't about predicting the middle word: it was just about saying whether a suggested middle word looked good or looked random.

Fig. 24: Exercise 04-06-ans1

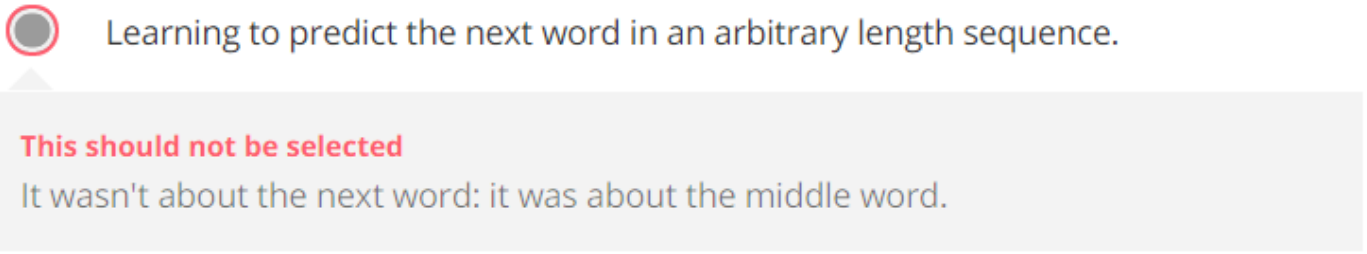


Fig. 25: Exercise 04-06-ans2

04-07: 1st;

Additional quiz:

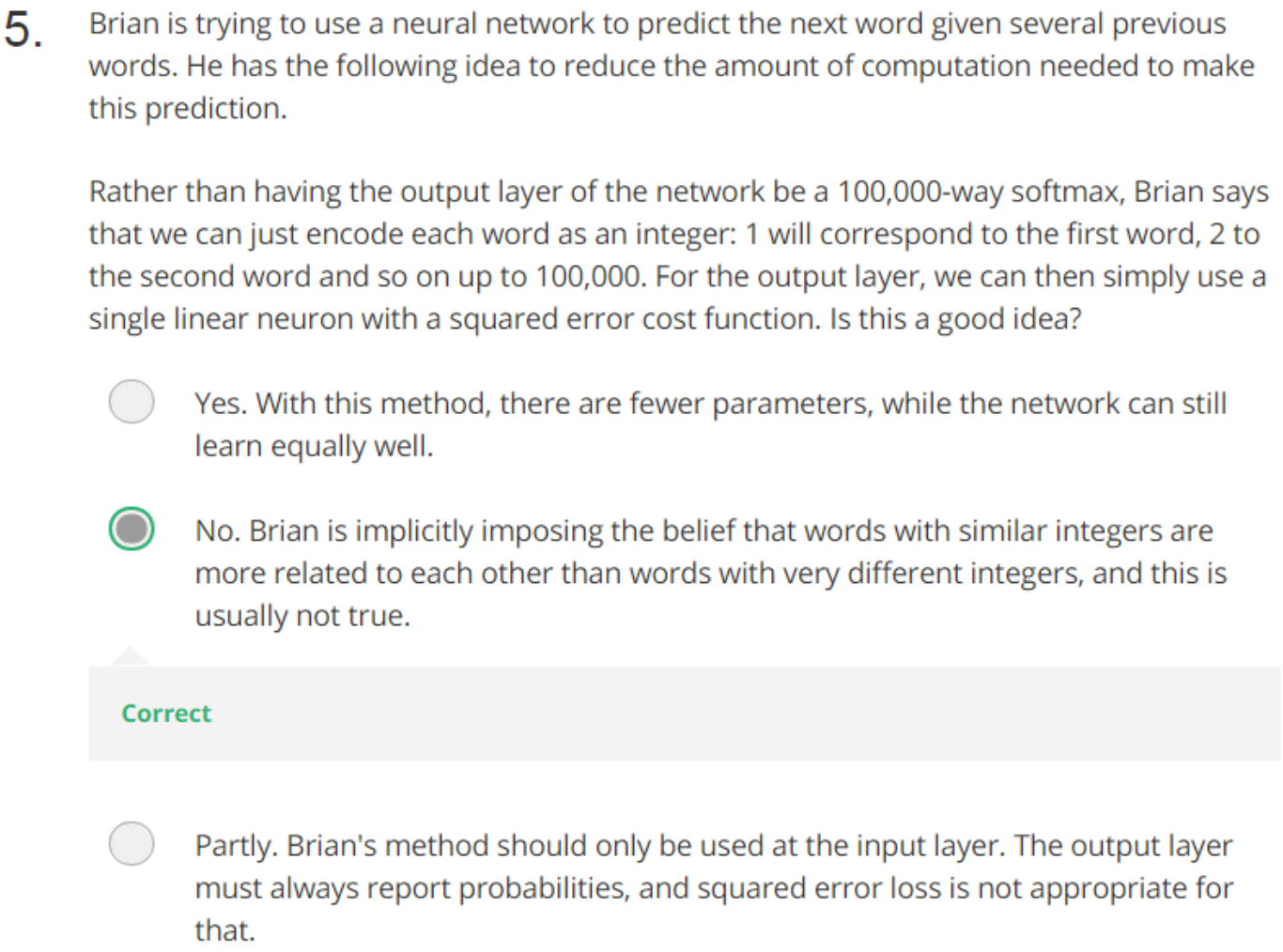


Fig. 26: Exercise 04-plus1

7. Suppose that we have a vocabulary of 3 words, "a", "b", and "c", and we want to predict the next word in a sentence given the previous two words. Also suppose that we don't want to use feature vectors for words: we simply use the local encoding, i.e. a 3-component vector with one entry being 1 and all other two entries being 0.

In the language models that we have seen so far, each of the context words has its own dedicated section of the network, so we would encode this problem with two 3-dimensional inputs. That makes for a total of 6 dimensions; clearly, the more context words we want to include, the more input units our network must have. Here's a method that uses fewer input units:

We could instead encode the **counts** of each word in the context. So a context of "a a" would be encoded as input vector [2 0 0] instead of [1 0 0 1 0 0], and "b c" would be encoded as input vector [0 1 1] instead of [0 1 0 0 0 1]. Now we only need an input vector of the size of our vocabulary (3 in our case), as opposed to the size of our vocabulary times the length of the context (which makes for a total of 6 in our case). Are there any significant problems with this idea?

- ○ Yes: even though the input has a smaller dimensionality, each entry of the input now requires more bits to encode, because it's no longer just 1 or 0. Therefore, there would be no significant advantage.
- ◉ Yes: the network loses the knowledge of the location at which a context word occurs, and that is valuable knowledge.

Correct

- ○ Yes: the neural networks shown in the course so far cannot deal with integer inputs (as opposed to binary inputs).
- ○ Yes: although we could encode the context in this way, we would then need a smaller bottleneck layer than we did before, thereby lowering the learning capacity of the model.

Fig. 27: Exercise 04-plus2

5 Object recognition with neural nets

5.1 Quiz

1. For which of the following tasks can we expect that the problem of "dimension hopping" will occur (given that the data is input correctly)? Check all that apply.

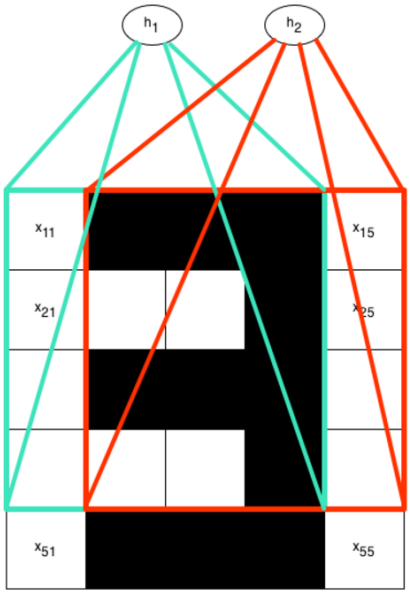
- [ ] Estimating the market price of a house given the number of rooms in the house, the number of schools and the average income of the surrounding neighbourhood, and the average sale price of the surrounding houses.
- [ ] Determining whether a wave has high frequency or low frequency. The input is a set of time values along with their corresponding vertical displacements.
- [ ] Determining whether a given image shows a bike or a car. The bike or car might appear anywhere in the image. The input is the whole set of pixels for the image.
- [ ] Estimating the risk that a patient will develop heart disease given their age, weight, blood pressure, and cholesterol level.

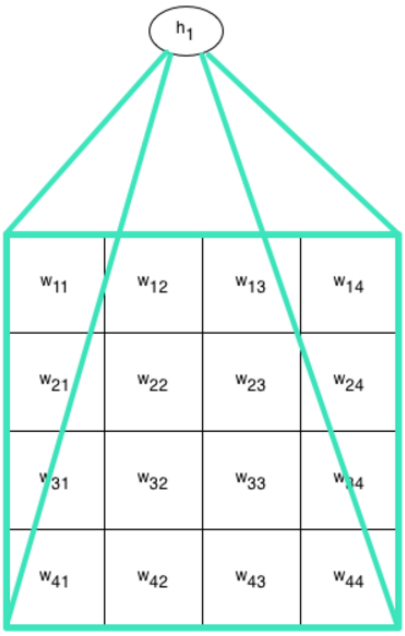
Fig. 28: Exercise 05-01

2. We have a convolutional neural network for images of 5 by 5 pixels. In this network, each hidden unit is connected to a different 4 x 4 region of the input image:

- The first hidden unit, h_1, is connected to the upper left 4x4 portion of the input image (as shown below).
- The second hidden unit, h_2, is connected to the upper right 4x4 portion of the input image (as shown below).
- The third hidden unit, h_3, is connected to the lower left 4x4 portion of the input image (not shown in the diagram).
- The fourth hidden unit, h_4, is connected to the lower right 4x4 portion of the input image (not shown in the diagram).

Because it's a convolutional network, the weights (connection strengths) are the same for all hidden units: the only difference between the hidden units is that each of them connects to a different part of the input image. In the second diagram, we show the array of weights, which are the same for each of the four hidden units.





For h_1, weight w_{11} is connected to the top-left pixel, i.e. x_{11}, while for hidden unit h_2, weight w_{11} connects to the pixel that is one to the right of the top left pixel, i.e. x_{12}.

Imagine that for some training case, we have an input image where each of the black pixels in the top diagram has value 1, and each of the white ones has value 0. Notice that the image shows a "3" in pixels.

The network has no biases. The weights of the network are given as follows:

$$\begin{array}{llll} w_{11}=1 & w_{12}=1 & w_{13}=1 & w_{14}=0 \\ w_{21}=0 & w_{22}=0 & w_{23}=1 & w_{24}=0 \\ w_{31}=1 & w_{32}=1 & w_{33}=1 & w_{34}=0 \\ w_{41}=0 & w_{42}=0 & w_{43}=1 & w_{44}=0 \end{array}$$

The hidden units are *linear*.

For the training case with that "3" input image, what is the output y_1, y_2, y_3, y_4 of each of the four hidden units?

- $y_1 = 2, y_2 = 4, y_3 = 4, y_4 = 8$
- $y_1 = 4, y_2 = 4, y_3 = 4, y_4 = 4$
- $y_1 = 4, y_2 = 2, y_3 = 8, y_4 = 4$
- $y_1 = 4, y_2 = 8, y_3 = 2, y_4 = 4$

3. Recall that pooling is the process of combining the outputs of several hidden units to create a single hidden unit. This introduces some invariance to

local transformations in the input image.

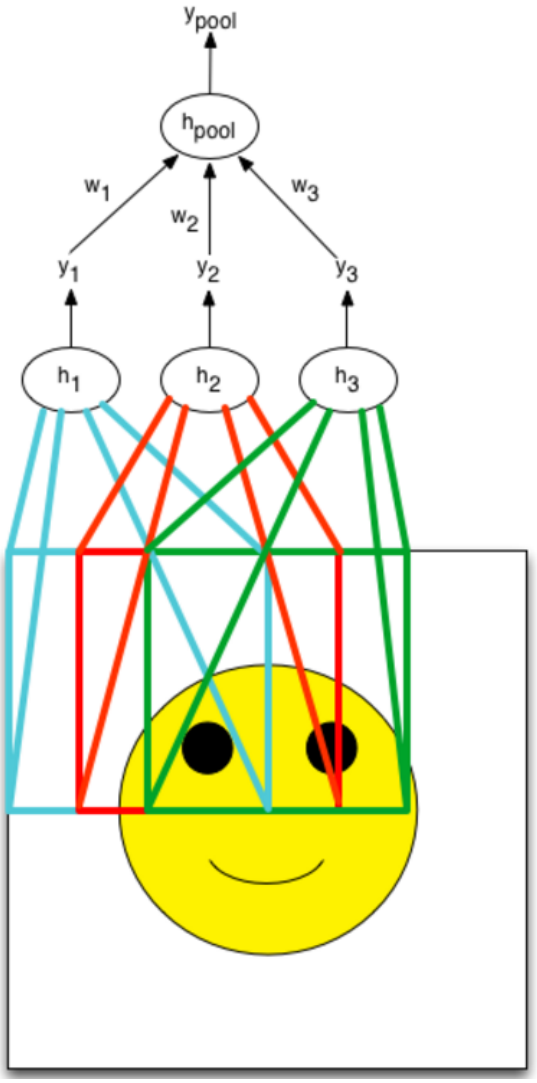
In this example we are pooling hidden units h_1, h_2, and h_3. Let's denote the output of hidden unit h_i as y_i. The hidden unit

that we're creating by pooling is called h_{pool}, with output y_{pool}. **Average pooling** is defined as $y_{\text{pool}} = \frac{y_1+y_2+y_3}{3}$.

This form of pooling can be equivalently represented by making h_{pool} a regular hidden unit that takes the output of the other three hidden

units, multiplies them by some weights w_1, w_2 and w_3, adds up the resulting numbers and then outputs some function of this sum. This is

illustrated in the figure below.



For this to be equivalent to **average pooling**, what type of neuron should h_{pool} be, and what should the weights be?

- $w_1 = 1, w_2 = 1, w_3 = 1$, and h_{pool} is a logistic neuron.
- $w_1 = 1, w_2 = 1, w_3 = 1$, and h_{pool} is a linear neuron.
- $w_1 = \frac{1}{3}, w_2 = \frac{1}{3}, w_3 = \frac{1}{3}$, and h_{pool} is a logistic neuron.
- $w_1 = \frac{1}{3}, w_2 = \frac{1}{3}, w_3 = \frac{1}{3}$, and h_{pool} is a linear neuron.

Fig. 30: Exercise 05-03

4. Suppose that we have a vocabulary of 3 words, "a", "b", and "c", and we want to predict the next word in a sentence given the previous two words. For

this network, we don't want to use feature vectors for words: we simply use the local encoding, i.e. a 3-component vector with one entry being 1

and the other two entries being 0.

In the language models that we have seen so far, each of the context words has its own dedicated section of the network, so we would encode this

problem with two 3-dimensional inputs. That makes for a total of 6 dimensions. For example, if the two preceding words (the "context" words) are "c"

and "b", then the input would be $(0, 0, 1, 0, 1, 0)$. Clearly, the more context words we want to include, the more input units our network must

have. More inputs means more parameters, and thus increases the risk of overfitting. Here is a proposal to reduce the number of parameters in the

model:

Consider a single neuron that is connected to this input, and call the weights that connect the input to this neuron w_1, w_2, w_3, w_4, w_5, and

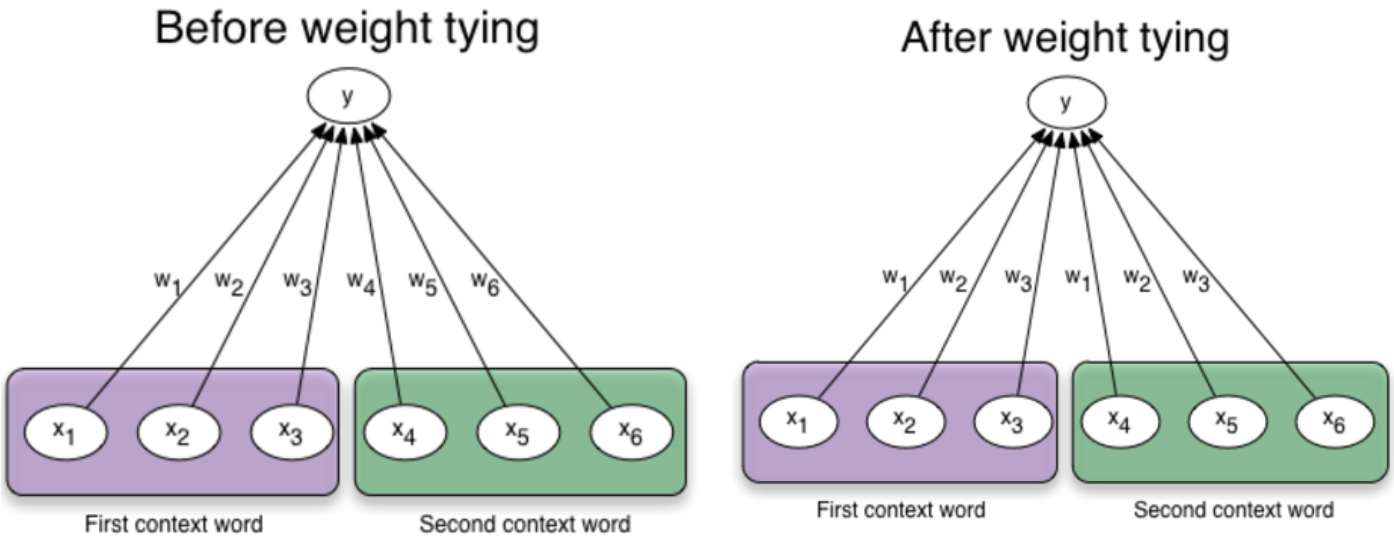
w_6. w_1 connects the neuron to the first input unit, w_2 connects it to the second input unit, etc. Notice how for every neuron, we need

as many weights as there are input dimensions (6 in our case), which will be the number of words times the length of the context. A way to reduce the number of

parameters is to *tie* certain weights together, so that they share a parameter. One possibility is to tie the weights coming from input units

that correspond to the same word but at different context positions. In our example that would mean that $w_1 = w_4$, $w_2 = w_5$, and $w_3 = w_6$

(see the "after" diagram below).



Are there any significant problems with this approach?

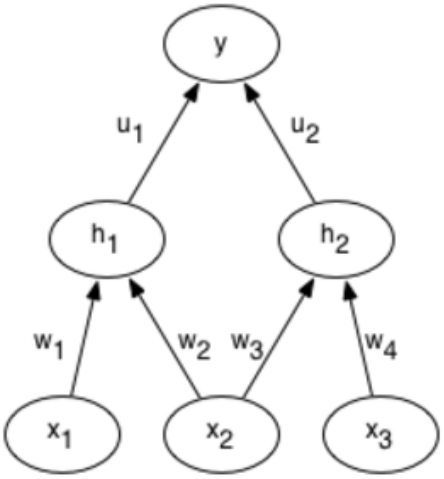
- Yes: weight tying only makes sense when we are working with images.
- Yes: the network loses the knowledge of the location at which a context word occurs, and that is valuable knowledge.
- No: the new model after weight tying is an example of a convolutional neural network, and these are more powerful than a non-convolutional network because they are invariant to small transformations in the data.
- No: this method is an appropriate solution in that it will reduce the number of parameters and therefore always improve generalization.

Fig. 31: Exercise 05-04

5. Let's look at what weight tying does to gradients, computed using the backpropagation algorithm. For this question, our network has three input units,

x_1, x_2, x_3, two *logistic* hidden units h_1, h_2, four input to hidden weights w_1, w_2, w_3, w_4, and two hidden to output weights

u_1, u_2. The output neuron y is a *linear neuron*, and we are using the *squared error* cost function.



Consider a single training case with target output t. The sequence of computations required to compute the error (this is called forward

propagation) are as follows:

$$
\begin{aligned}
z_1 &= w_1 x_1 + w_2 x_2 \\
z_2 &= w_3 x_2 + w_4 x_3 \\
h_1 &= \sigma(z_1) \\
h_2 &= \sigma(z_2) \\
y &= u_1 h_1 + u_2 h_2 \\
E &= \frac{1}{2}(t - y)^2
\end{aligned}
$$

E is the error. Reminder: $\sigma(x) = \frac{1}{1+\exp(-x)}$

z_1 and z_2 are called the *total inputs* to hidden units h_1 and h_2 respectively.

Suppose we now decide to tie the weights so that $w_1 = w_2 = w_{\text{tied}}$. What is the derivative of the error E with respect to w_{tied}?

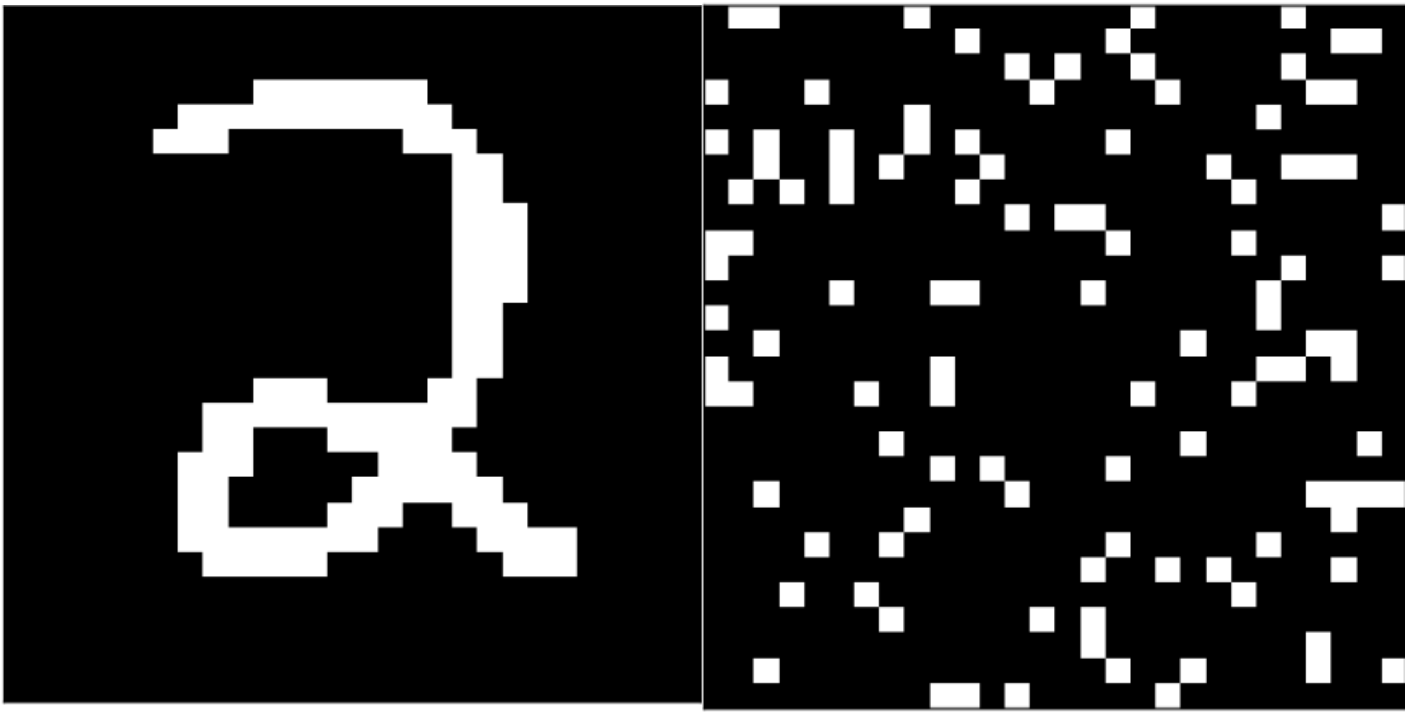
Hint: forget about weight tying for a moment and simply compute the derivatives $\frac{\partial E}{\partial w_1}$ and $\frac{\partial E}{\partial w_2}$. Now set $\frac{\partial E}{\partial w_{\text{tied}}} = \frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$.

- $\frac{\partial E}{\partial w_{\text{tied}}} = -2(t-y)(u_1 h_1(1-h_1)x_2)$
- $\frac{\partial E}{\partial w_{\text{tied}}} = -(t-y)u_1 h_1(1-h_1)(x_1+x_2)$
- $\frac{\partial E}{\partial w_{\text{tied}}} = -2(t-y)(u_1 h_1(1-h_1)x_1)$
- $\frac{\partial E}{\partial w_{\text{tied}}} = -(t-y)(u_1 h_1(1-h_1) + u_2 h_2(1-h_2))x_2$

Fig. 32: Exercise 05-05

6. Oh no! Brian's done it again! Claire had a dataset of 28 x 28 pixel handwritten digits nicely prepared to train a neural network, but Brian has gone and accidentally scrambled the images by re-ordering the pixels in some totally meaningless way, and now they can't get the original dataset back! Below is an image of a handwritten '2' before and after being scrambled by Brian.

   BeforeAfter



Luckily, all of the images (in both the training set and the test set) were changed in the same way. For example, if pixels number 1 and number 3 switched places in one image, then they switched places in every other image as well. Because of that, Claire thinks that perhaps she can train a network after all.

Whether Claire is right or not depends largely on the type of neural network that she has in mind. Which of the following neural networks will be at a **disadvantage** because of Brian's mistake? Check all that apply.

- [ ] A feed-forward neural network with one hidden layer of linear units (and no convolution).
- [ ] A feed-forward neural network with no hidden layer and logistic units (and no convolution).
- [ ] A convolutional neural network where the size of each weight filter is 10 x 10.
- [ ] A convolutional neural network where the size of each weight filter is 8 x 8.

Fig. 33: Exercise 05-06

5.2 My Answers

05-01: 2nd, 3rd;

Dimension hopping occurs when one can take the information contained in the dimensions of some input, and move this between dimensions while not changing the target. The canonical example is taking an image of a handwritten digit and translating it within the image. The dimensions that contain "ink" are now different (they have been moved to other dimensions), however the label we assign to the digit has not changed. Note that this is not something that happens consistently across the dataset, that is we may have a dataset containing two handwritten digits where one is a translated version of the other, however this still does not change the corresponding label of the digits.

05-02: 4th;

05-03: 4th;

05-04: 2nd;

05-05: 2nd;

05-06: 3rd, 4th;

6 Optimization: How to make the learning go faster

6.1 Quiz

1. Suppose w is the weight on some connection in a neural network. The network is trained using gradient descent until the learning *converges*. We plot the change of w as training progresses. Which of the following scenarios shows that convergence has occurred?
Notice that we're plotting the change in w, as opposed to w itself.

   Note that in the plots below, each *iteration* refers to a single *step* of steepest descent on a *single minibatch*.

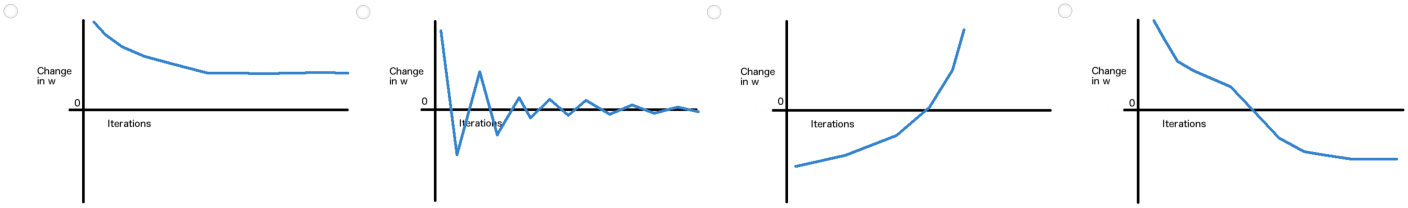


Fig. 34: Exercise 06-01

2. Suppose you are using mini-batch gradient descent for training some neural net on a large dataset. You have to decide on the learning rate, weight initializations, preprocess the inputs etc. You try some values for these and find that the value of the objective function on the training set decreases smoothly but very slowly. What could be causing this? Check all that apply.

- [ ] The learning rate may be too small.
- [ ] The inputs might have a very large scale (hint: think of what this would do to the logistic hidden units).
- [ ] The minibatch size is too small.
- [ ] The weights might have been initialized to very large values (hint: think of what this would do to the logistic hidden units).

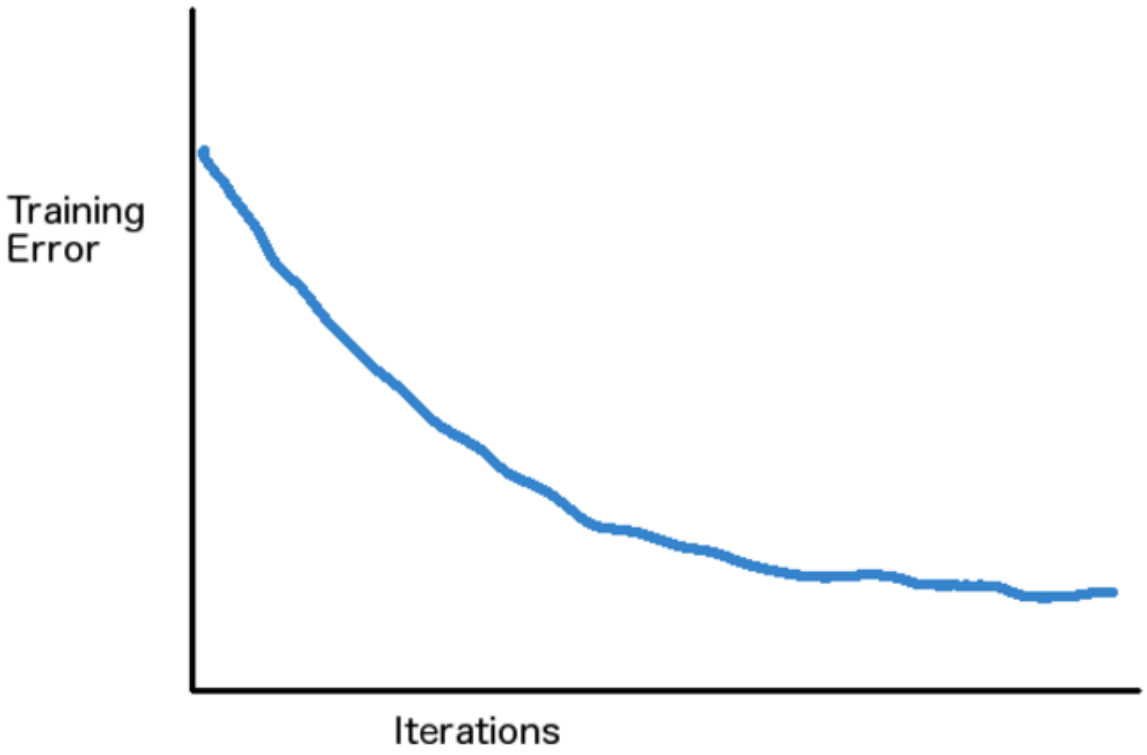
Fig. 35: Exercise 06-02

3. Full-batch gradient descent can be used to minimize an objective function if the dataset is not too large. Which statement regarding full-batch gradient descent is **false**?

- ( ) For every **fixed** learning rate, the objective function will monotonically decrease.
- ( ) There is always a learning rate schedule such that the objective function monotonically decreases.
- ( ) For some setting of the learning rate, it is possible that the objective function increases in some iteration.
- ( ) Adaptive learning rate methods perform well for full-batch (or large mini-batch) gradient descent.

Fig. 36: Exercise 06-03

4. Claire is training a neural net using mini-batch gradient descent. She chose a particular learning rate and found that the training error decreased as more iterations of training were performed as shown here in blue



She was not sure if this was the best she could do. So she tried a **smaller** learning rate. Which of the following error curves (shown in red) might she observe now? Select the two most likely plots.

Note that in the plots below, each *iteration* refers to a single *step* of steepest descent on a *single minibatch*.

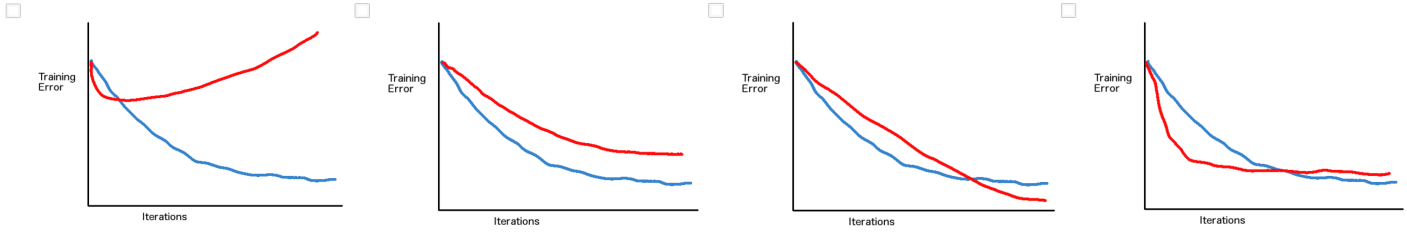


Fig. 37: Exercise 06-04

5. In the lectures, we discussed two kinds of gradient descent algorithms: mini-batch and full-batch. For which of the following problems is mini-batch gradient descent likely to be **a lot better** than full-batch gradient descent?

- [ ] Sentiment Analysis: Decide whether a given movie review says that the movie is 'good' or 'bad'. The input consists of the word count in the review, for each of 50,000 words. The training set consists of 100 movie reviews written by experts for a newspaper.
- [ ] Predict if an experiment at the Large Hadron Collider is going to yield positive results. The input consists of 25 experiment parameters (energy level, types of particles, etc). The training set consists of the 200 experiments that have already been completed (some of those yielded positive results; some yielded only negative results).
- [ ] Sentiment Analysis: Decide whether a given movie review says that the movie is 'good' or 'bad'. The input consists of the word count in the review, for each of 50,000 words. The training set consists of 1,000,000 movie reviews found on the internet.
- [ ] Language modeling: Predict the next word using the previous 3 words. The vocabulary consists of 100,000 words. The dataset consists of all Wikipedia articles.

Fig. 38: Exercise 06-05

6.2 My Answers

06-01: 2nd;

If the optimization has converged, w must converge to (or at most oscillate around) a point. So the change in w must converge to (or oscillate around) zero. 06-02: 1st, 2nd, 4th;

A small learning rate leads to small changes in the parameters, and to slow convergence.

Large values of inputs may saturate the hidden units. Their derivatives would become small (be on a "plateau") and learning would get slowed down.

Large values of weights may saturate the hidden units. Their derivatives would become small (be on a "plateau") and learning would get slowed down. 06-03: 1st;

If the learning rate is high, the weights get updated too much in the direction of steepest descent and "climb up the ravine". 06-04: 2nd, 3rd;

06-05: 3rd, 4th;

Additional Question:

Four datasets are shown below. Each dataset has two input values (plotted below) and a target value (not shown). Each point in the plots denotes one training case. Assume that we are solving a classification problem. Which of the following datasets would most likely be easiest to train using neural nets ?

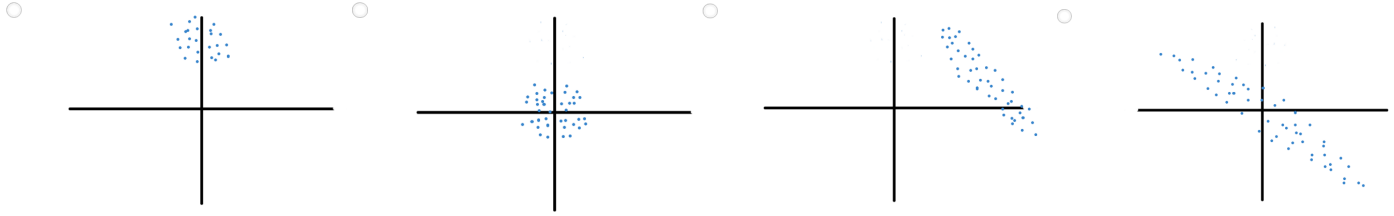


Fig. 39: Exercise 06-plus1

Answer: 2nd;

This dataset has mean zero and diagonal covariance.

7 Recurrent neural networks

7.1 Quiz

1. How many bits of information can be modeled by the hidden state (at some specific time) of a Hidden Markov Model with 16 hidden units?

- 4
- >16
- 16
- 2

Fig. 40: Exercise 07-01

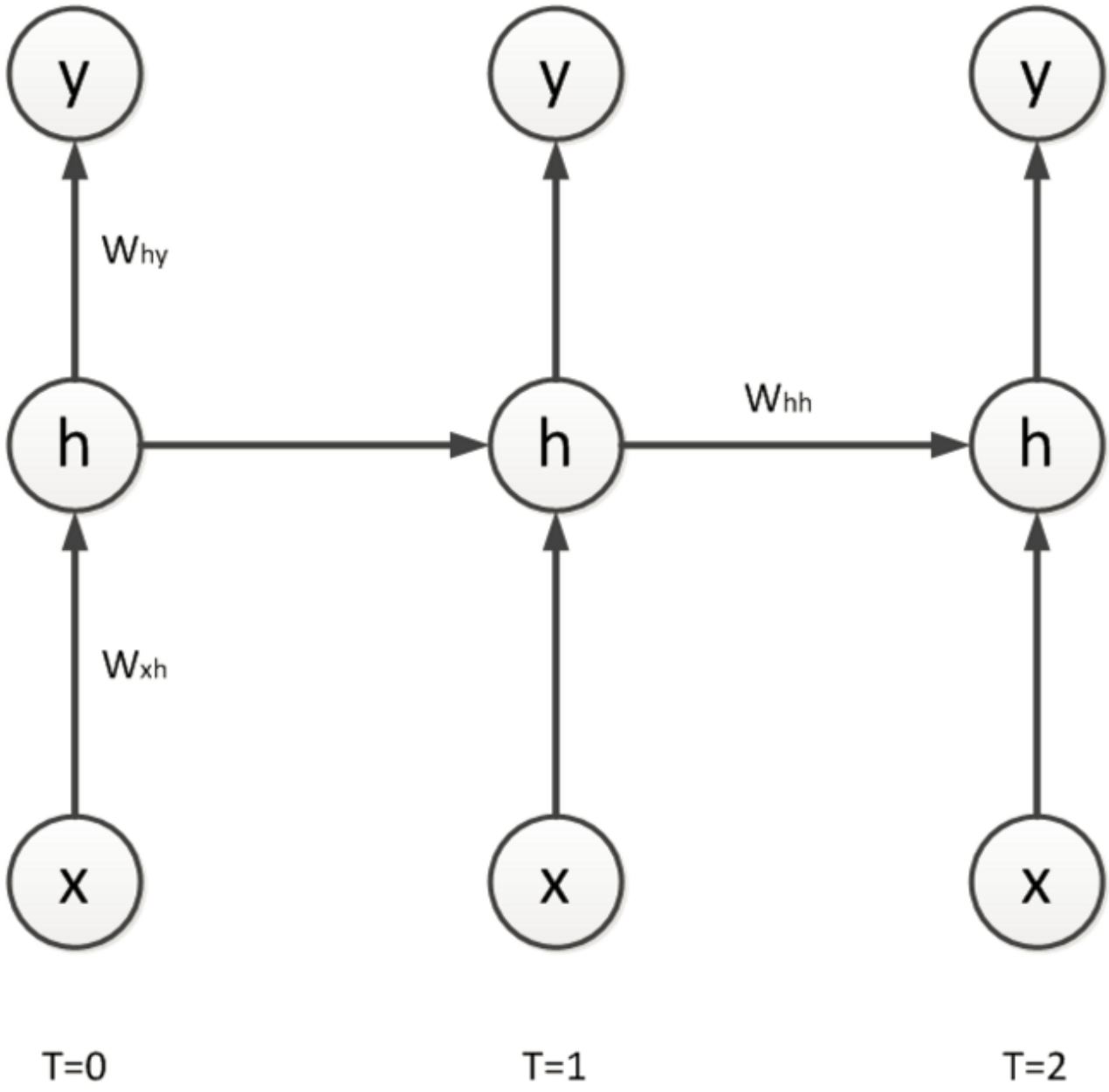
2. This question is about speech recognition. To accurately recognize what phoneme is being spoken at a particular time, one needs to know the sound data from 100ms before that time to 100ms after that time, i.e. a total of 200ms of sound data. Which of the following setups have access to enough sound data to recognize what phoneme was being spoken 100ms into the past?

- [ ] A feed forward Neural Network with 30ms of input
- [ ] A feed forward Neural Network with 200ms of input
- [ ] A Recurrent Neural Network (RNN) with 30ms of input
- [ ] A Recurrent Neural Network (RNN) with 200ms of input

Fig. 41: Exercise 07-02

3. The figure below shows a Recurrent Neural Network (RNN) with one input unit x, one logistic hidden unit h, and one linear output unit

y. The RNN is unrolled in time for T=0,1, and 2.



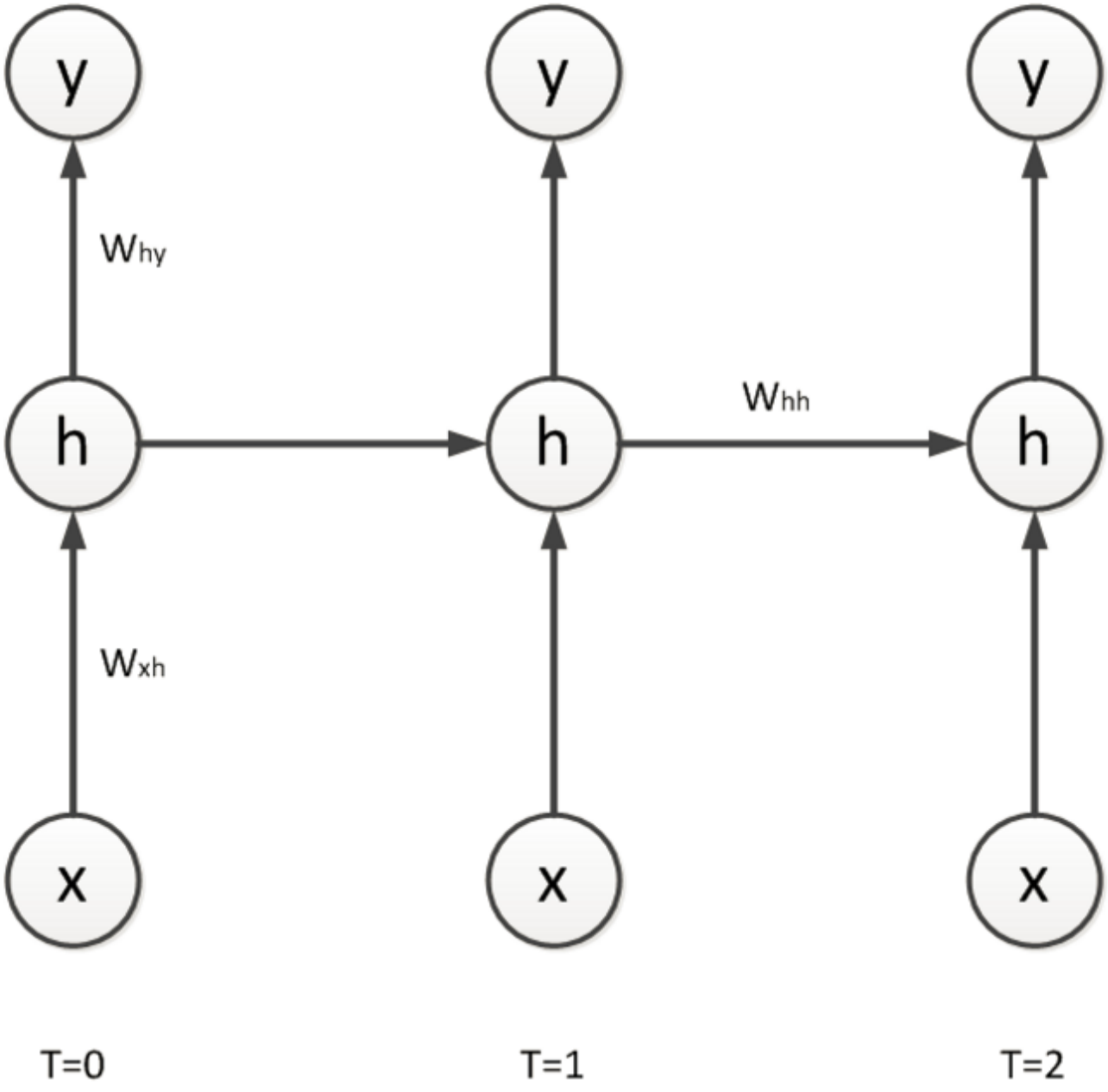
The network parameters are: $W_{xh} = 0.5$, $W_{hh} = -1.0$, $W_{hy} = -0.7$, $h_{bias} = -1.0$, and $y_{bias} = 0.0$. Remember, $\sigma(k) = \frac{1}{1+\exp(-k)}$.

If the input x takes the values $9, 4, -2$ at time steps $0, 1, 2$ respectively, what is the value of the hidden state h at $T = 2$? Give your answer with at least two digits after the decimal point.

Fig. 42: Exercise 07-03

4. The figure below shows a Recurrent Neural Network (RNN) with one input unit x, one logistic hidden unit h, and one linear output unit

y. The RNN is unrolled in time for T=0,1, and 2.



The network parameters are: $W_{xh} = -0.1$, $W_{hh} = 0.5$, $W_{hy} = 0.25$, $h_{\text{bias}} = 0.4$, and $y_{\text{bias}} = 0.0$.

If the input x takes the values $18, 9, -8$ at time steps $0, 1, 2$ respectively, the hidden unit values will be $0.2, 0.4, 0.8$ and the output unit values will be $0.05, 0.1, 0.2$ (you can check these values as an exercise).

A variable z is defined as the total input to the hidden unit before the logistic nonlinearity.

If we are using the squared loss, with targets t_0, t_1, t_2, then the sequence of calculations required to compute the total error E is as follows:

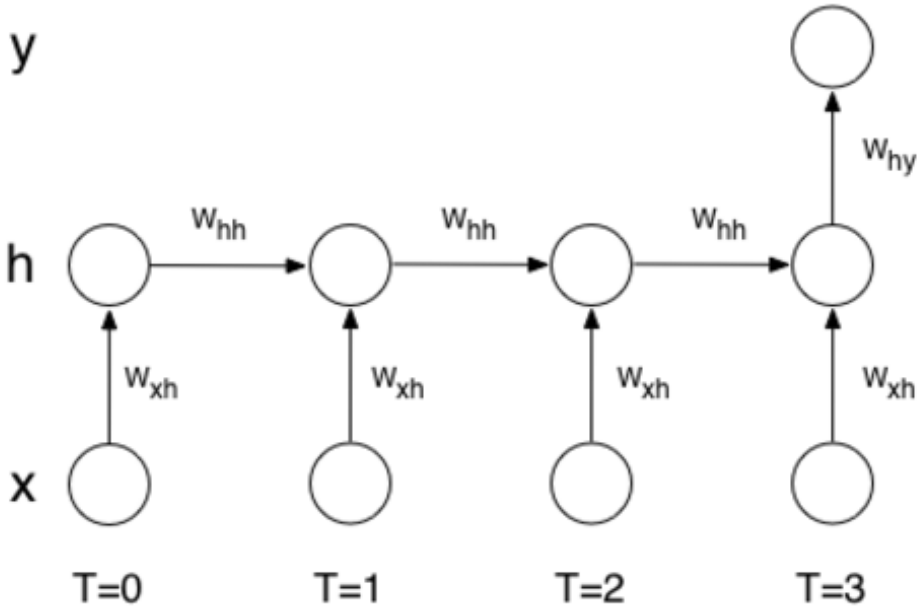
$$
\begin{array}{lll}
z_0 = W_{xh}x_0 + h_{\text{bias}} & z_1 = W_{xh}x_1 + W_{hh}h_0 + h_{\text{bias}} & z_2 = W_{xh}x_2 + W_{hh}h_1 + h_{\text{bias}} \\
h_0 = \sigma(z_0) & h_1 = \sigma(z_1) & h_2 = \sigma(z_2) \\
y_0 = W_{hy}h_0 + y_{\text{bias}} & y_1 = W_{hy}h_1 + y_{\text{bias}} & y_2 = W_{hy}h_2 + y_{\text{bias}} \\
E_0 = \frac{1}{2}(t_0 - y_0)^2 & E_1 = \frac{1}{2}(t_1 - y_1)^2 & E_2 = \frac{1}{2}(t_2 - y_2)^2 \\
E = E_0 + E_1 + E_2 & &
\end{array}
$$

If the target output values are $t_0 = 0.1, t_1 = -0.1, t_2 = -0.2$ and the squared error loss is used, what is the value of the error derivative just before the hidden unit nonlinearity at $T = 2$ (i.e. $\frac{\partial E}{\partial z_2}$)? Write your answer up to at least the fourth decimal place.

Fig. 43: Exercise 07-04

5. Consider a Recurrent Neural Network with one input unit, one logistic hidden unit, and one linear output unit. This RNN is for modeling sequences of

length 4 only, and the output unit exists only at the last time step, i.e. T=3. This diagram shows the RNN unrolled in time:



Suppose that the model has learned the following parameter values:

- $w_{xh} = 1$
- $w_{hh} = 2$
- $w_{hy} = 1$
- All biases are 0

For one specific training case, the input is 1 at T=0 and 0 at T=1, T=2, and T=3. The target output (at T=3) is 0.5, and we're using the squared error

loss function.

We're interested in the gradient for w_{xh}, i.e. $\frac{\partial E}{\partial w_{xh}}$. Because it's only at T=0 that the input is not zero, and it's only at T=3 that there's an output, the

error needs to be backpropagated from T=3 to T=0, and that's the kind of situations where RNN's often get either vanishing gradients or exploding

gradients. Which of those two occurs in this situation?

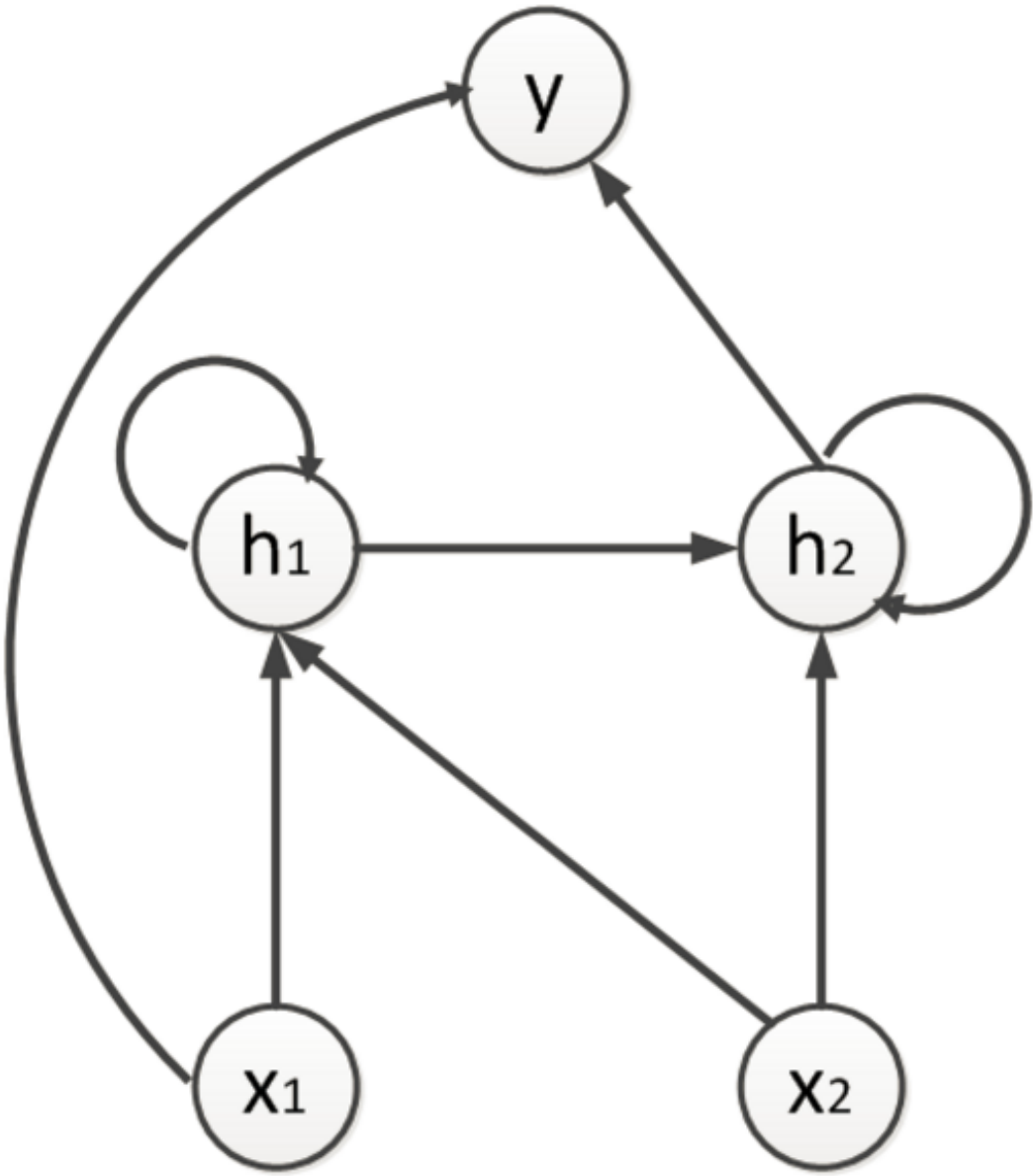
You can either do the calculations, and find the answer that way, or you can find the answer with more thinking and less math, by thinking about the

slope $\frac{\partial y}{\partial z}$ of the logistic function, and what role that plays in the backpropagation process.

- Exploding gradient
- Vanishing gradient

Fig. 44: Exercise 07-05

6. Consider the following Recurrent Neural Network (RNN):



As you can see, the RNN has two input units, two hidden units, and one output unit.

For this question, every arrow denotes the effect of a variable at time t on a variable at time $t+1$.

Which feed forward Neural Network is equivalent to this network unrolled in time?

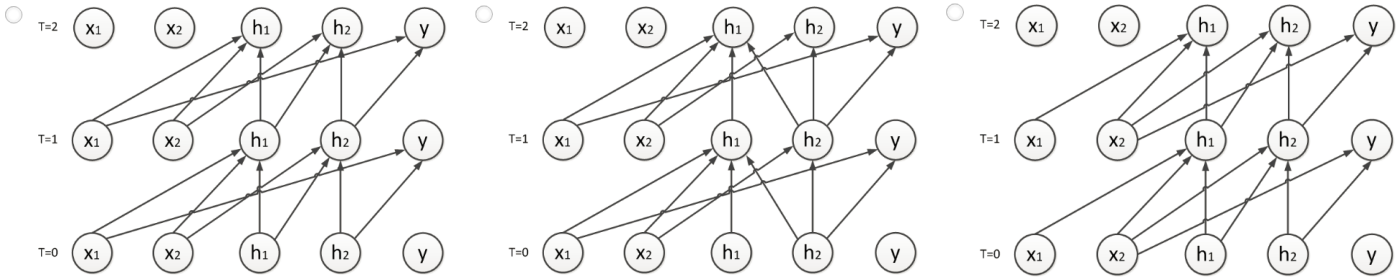


Fig. 45: Exercise 07-06

7.2 My Answers

07-01: 1st;

07-02: 2nd, 3rd, 4th;

07-03: 0.0487678078178;

```
import math
x0 = 9;x1 = 4;x2 = -2
wxh = 0.5;whh = -1.0;why = -0.7
hb = -1.0;yb = 0

z0 = x0*wxh+hb
h0 = 1.0/(math.exp(-z0)+1.0)
z1 = h0*whh+x1*wxh+hb
h1 = 1.0/(math.exp(-z1)+1.0)
z2 = h0*whh+x2*wxh+hb
h2 = 1.0/(math.exp(-z2)+1.0)
y0 = h0*why+yb
y1 = h1*why+yb
y2 = h2*why+yb
print(h2)
```

Fig. 46: Answer 07-03

07-04:

We need to calculate $\frac{\partial E}{\partial z_1}$. We can do this by backpropagation:

$$\frac{\partial E}{\partial z_1} = \frac{\partial E_1}{\partial z_1} + \frac{\partial E_2}{\partial z_1}$$
$$\frac{\partial E_1}{z_1} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial h_1}\frac{\partial h_1}{\partial z_1}$$
$$\frac{\partial E_2}{z_1} = \frac{\partial E_2}{\partial y_2}\frac{\partial y_2}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial h_1}\frac{\partial h_1}{\partial z_1}$$

The first equality holds because only the error at time steps 1 and 2 depend on z_1.

Fig. 47: Answer 07-04

07-05: Vanishing gradient;

07-06: 1st;

8 More recurrent neural networks

8.1 Quiz

1. Imagine that we have a fully trained RNN that uses multiplicative connections as explained in the lecture. It's been trained well, i.e. we found the model parameters with which the network performs well. Now we want to convert this well-trained model into an equivalent model with a different architecture. Which of the following statements are correct?

- [ ] We can use the model where the input character chooses the whole hidden-to-hidden weight matrix. It can be more difficult to train, but it's sufficiently flexible.
- [ ] We can use the additive input model proposed in the lecture (see the page about "An obvious recurrent neural net" in the video about multiplicative connections), with a modification inspired by what we saw in the older language models: instead of connecting the input character directly to the hidden state at the next time step, we use a different vector representation for each of the 86 input characters, and we connect that vector representation directly to the hidden state at the next time step. If the vector has as many elements as there are factors in the original model, this will be flexible enough that an equivalent model can always be built.
- [ ] We can use the additive input model proposed in the lecture (see the page about "An obvious recurrent neural net" in the video about multiplicative connections). It can be more difficult to train, but it's sufficiently flexible.
- [ ] We can use the model where the input character chooses the whole hidden-to-hidden weight matrix, but only if there aren't too many factors in the multiplicative model. If there are too many factors, then the multiplicative model will have more parameters than this alternative model, which means that there will not always be an equivalent model of this alternative form.

Fig. 48: Exercise 08-01

2. The multiplicative factors described in the lecture are an alternative to simply letting the input character choose the hidden-to-hidden weight matrix. Let's carefully compare these two methods of connecting the current hidden state and the input character to the next hidden state.

Suppose that all model parameters (weights, biases, factor connections if there are factors) are between -1 and 1, and that the hidden units are logistic, i.e. their output values are between 0 and 1. Normally, not all neural network model parameters are between -1 and 1 (although they typically end up being between -100 and 100), but for this question we simplify things and say that they are between -1 and 1.

For the simple model, this restriction on the parameter size and hidden unit output means that the largest possible contribution that hidden unit #56 at time t can make to the input (i.e. before the logistic) of hidden unit #201 at time $t + 1$ is 1, no matter what the input character is. This happens when the hidden-to-hidden weight matrix chosen by the input unit has a value of 1 for the connection from #56 to #201, and hidden unit #56 at time t is maximally activated, i.e. its state (after the logistic) is 1. Those two get multiplied together, for a total contribution of 1.

Let's say that our factor model has 1000 factors and 1500 hidden units. What is the largest possible contribution that hidden unit #56 at time t can possibly make to the input (i.e. before the logistic) of hidden unit #201 at time $t + 1$, in this factor model, subject to the same restriction on parameter size and hidden unit output?

Enter answer here

Fig. 49: Exercise 08-02

3. The multiplicative factors described in the lecture are an alternative to simply letting the input character choose the hidden-to-hidden weight matrix. In the

lecture, it was explained that that simple model would have 86 x 1500 x 1500 = 193 500 000 parameters, to specify how the hidden units and the input

character at time t influence the hidden units at time $t + 1$. How many parameters does the model with the factors have for that same purpose,

i.e. for specifying how the hidden units and the input character at time t influence the hidden units at time $t + 1$? Let's say that there are

1500 hidden units, 86 different input characters, and 1000 factors.

Enter answer here

Fig. 50: Exercise 08-03

4. In the lecture, you saw some examples of text that Ilya Sutskever's model generated, after being trained on Wikipedia articles. If we ask the model to generate a couple of sentences of text, it quickly becomes clear that what it's saying is not something that was actually written in Wikipedia. Wikipedia articles typically make much more sense than what this model generates. Why doesn't the model generate significant portions of Wikipedia articles?

- That would have been overfitting, which was carefully avoided. The model has to generalize well.
- Basic calculations about the size of the hidden state vector show that the model can never learn to reliably generate any fixed string of text that's more than 38 characters long (38 is the square root of 1500, the number of hidden units).
- It should learn to generate whole Wikipedia articles, eventually, with more training. However, that might require more compute power than is available nowadays (the model had to be trained for a month already, on a very fast computer).

Fig. 51: Exercise 08-04

5. Echo State Networks need to have many hidden units. The reason for that was explained in the lecture. This means that they also have many hidden-to-hidden connections. Does that fact place ESN's at risk of overfitting?

- No
- Yes

Fig. 52: Exercise 08-05

6. Recurrent Neural Networks are often plagued by vanishing or exploding gradients, as a result of backpropagating through many time steps. The longer the input sequence, i.e. the more time steps there are, the greater this danger becomes. Do Echo State Networks suffer the same problem?

- No
- Yes

Fig. 53: Exercise 08-06

7. In Echo State Networks, does it matter whether the hidden units are linear or logistic (or some other nonlinearity)?

- No. The hidden units are not learned, so the usual Neural Network concerns don't apply here.
- Yes. With linear hidden units, the output would become a linear function of the inputs, and we typically want to learn nonlinear functions of the input. Therefore, linear hidden units are a bad choice.

Fig. 54: Exercise 08-07

8.2 My Answers

08-01: 1st;

The multiplicative factors effectively create a different hidden-to-hidden matrix for each input character.

08-02: 1000;

A total of 1 per factor. There are 1000 factors, so the answer is 1000.

08-03: 4629000;

Each factor has connections to all of the 86 input characters, to the hidden units at time t, and to the hidden units at time t+1, for a total of 86+1500+1500=3086 parameters per factor. The total is 3086*1500=4629000.

08-04: 1st;

08-05: No;

The hidden-to-hidden connections are not learned, and without learning, there is no overfitting.

08-06: No;

ESN't don't backpropagate through time, so those gradients are irrelevant.

08-07: Yes;

Additional Questions:

5.

Recall that Neural Networks for language modeling often learn word representation vectors, to avoid the need to have a weight from every possible input word to every hidden unit. Can we use the same learning method in an Echo State Network?

- No
- Yes

Fig. 55: Exercise 08-plus1

Answer: No;

That would require backpropagating through time, which is exactly what ESN's don't do.

6. Visualizing what a Neural Network has learned is often difficult. Here are two ideas for how to generate text from Ilya Sutskever's language model. Which one will best show what the model has learned?

- Give it the beginning of a sentence, e.g. "Once upon a time, ", and ask it which character it thinks will come next. That's a probability distribution over the 86 possible characters. As the next character in our sentence-under-construction, we use the character that the model considers most likely. Append that to the partial sentence, and repeat the procedure to get the next character. Repeat this until we have several pages of text.

- Give it the beginning of a sentence, e.g. "Once upon a time, ", and ask it which character it thinks will come next. That's a probability distribution over the 86 possible characters. Pick a character according to that distribution. Append that to the partial sentence, and repeat the procedure to get the next character. Repeat this until we have several pages of text.

Fig. 56: Exercise 08-plus2

Answer: 2nd;

If we go for the most probable character, we'd probably see the model repeating the same sentence over and over again. A probability distribution is better visualized by samples from it than by showing only what has the highest probability.

Reference