Started on	Sunday, 21 July 2024, 4:29 PM
State	Finished
Completed on	Sunday, 4 August 2024, 12:19 PM
Time taken	13 days 19 hours
Marks	73/73
Grade	**10** out of 10 (**100**%)

Question 1

Correct

Mark 8 out of 8

For the example binary classification of images into street light, no street light classes:

... refers to settings of the optimization procedure or the selection of model family. E.g.: The DNN architecture (number of layers, etc.), the early stopping threshold.	Hyperparameter
... values represent new samples that shall be processed by the ML model. E.g.: The vectors of size width x height x 3 representing the RGB values of the pixels in an input image.	Input
... values are fixed during inference, and are optimized during training. E.g.: The DNN weights and biases, the mean and standard deviation of batch normalization layers.	Parameter

Your answer is correct.

Correct

Marks for this submission: 8/8.

Question 2

Correct

Mark 8 out of 8

What are strict preliminaries to optimize parameters θ of a function $f : (x, \theta) \mapsto f(x; \theta)$:

True	False	
○	◉	f must be differentiable with respect to x.
◉	○	Each output f_i of f must be differentiable with respect to θ.
◉	○	A differentiable cost function that can evaluate the real-valued cost of input-output pairs of f.
○	◉	A test dataset.
◉	○	f must be differentiable with respect to θ.
○	◉	Each output f_i of f must be differentiable with respect to x.
○	◉	A training dataset.
○	◉	A differentiable cost function and a performance function that can evaluate the real-valued cost of input-output pairs of \(f \).

Correct

Marks for this submission: 8/8.

Question 3

Correct

Mark 8 out of 8

For a function F that maps parameters θ to a loss value, the i-th gradient descent step for learning rate λ is

θ^(i+1) = θ^(i) - λ ∇F (θ^(i))

θ^(i+1) θ^(i) λ ∇F

Your answer is correct.

Correct

Marks for this submission: 8/8.

Question 4

Correct

Mark 3 out of 3

What effect does adding momentum to the gradient descent step have?

Select one or more:

- [x] a. applies per-parameter summands to the step size depending on the previous steps
- [ ] b. reduces size of update steps with increasing number of steps to not overshoot
- [ ] c. applies per-parameter factors to the step size depending on the previous steps
- [x] d. decreases chances to get stuck in small minima
- [x] e. increases step length when the direction of descend does not change ("accelerates")
- [ ] f. increases memory efficiency by processing fewer training samples at once

Your answer is correct.

Correct

Marks for this submission: 3/3.

Question 5

Correct

Mark 16 out of 16

[[ERRATA: Version < 14 contained several errata (missing lambda and starting point spec, one typo in the data points; some erroneous target results); known errors are fixed now]]

Choosing the learning rate (and possibly its decay) appropriately is very important to ensure convergence of the training procedure. In the following example, our goal is to fit a parabola curve to noisy data points. Given are (1) the function $$ f(x;\theta)=x^2 + \theta, $$ (2) a starting value for \(\theta \), (3) SSE as cost function, and (4) the labeled data points \(D \) as

Data points

input x	output label
3.1	11.9
-0.3	3.1
-2.2	7.1

Now let's try out gradient descent to find a good \(\theta \). What value will \(\theta^{(3)} \) have (i.e., the parameter optimized for three subsequent gradient descent steps) for a starting value of \(\theta^{(0)}=-5 \) with

- **fixed learning rate** \(\lambda=1 \): 940.52 (does not converge, learning rate too high)
- **fixed learning rate** \(\lambda=0.1 \): 2.038 (converges)
- **decaying learning rate** \(\lambda^{(i)}=0.1^{i-1} \): 16.65 (converges very slowly, decay too large)
- **stochastic gradient descent** with \(\lambda=1 \) batch size of 1 and order of batches as in the set notation above (Note: three steps here are called one epoch, which means every sample was seen once): 8.08

Tips:

1. *Before diving into gradient descent, obtain some gut feeling about sensible values for \(\theta \). E.g., plot the points: what value do you estimate to be optimal for \(\theta \)? Or can you even identify the optimal \(\theta \) using curve analysis techniques (at zero points of the gradient)?*
2. *You might want to follow the steps of*
    1. *formulate the loss on as a function \(F(\theta) = ... \);*
    2. *determine the gradient \(F' \) with respect to \(\theta \) (possibly check out typical math tools for the differentiation, such as Wolfram Alpha);*
    3. *step-wise calculate the parameters (possibly use automation by programming the step in a notebook or similar).*

Correct

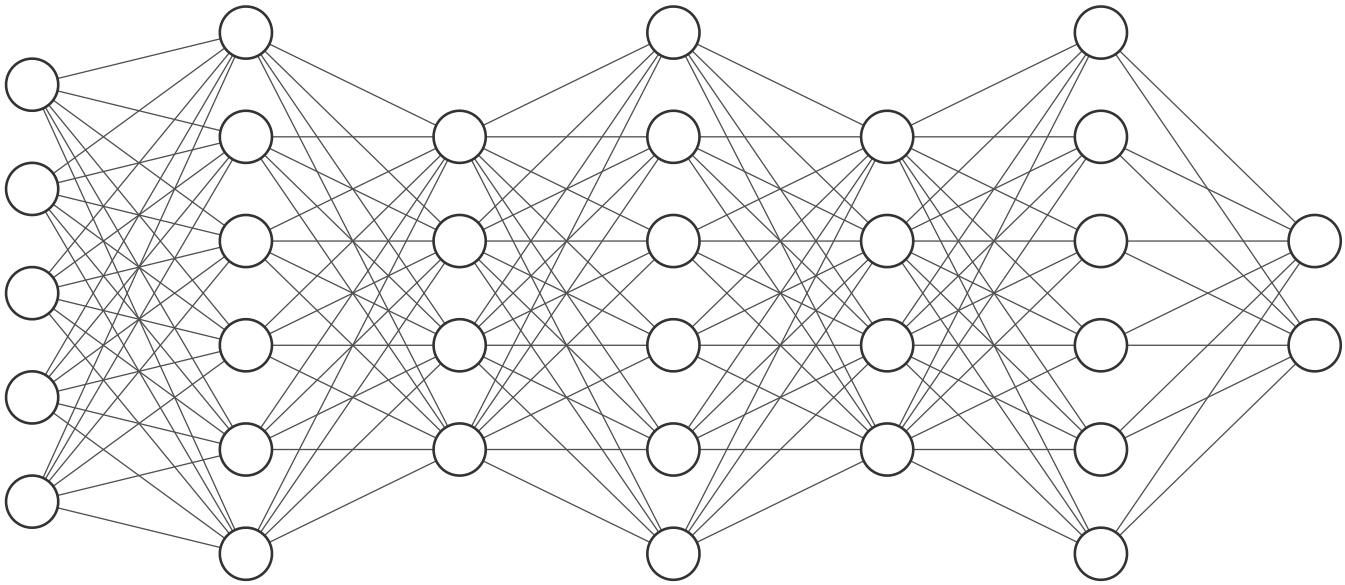
Marks for this submission: 16/16.

Question 6

Correct

Mark 4 out of 4

Let's have a look at the following small fully-connected hourglass DNN architecture (layers with 5,6,4,6,4,6,2 neurons) with ReLU activation (i.e., no additional parameters for the activation function):



Fill in the following: The DNN

- takes 5 input values,
- has 5 hidden layers,
- produces 2 output values from its output layer,
- has 166 parameters.

Correct

Marks for this submission: 4/4.

Question 7

Correct

Mark 6 out of 6

- batch normalization layers re-scale and re-center the inputs to be centered around 0 and match a given standard deviation.
- dropout layers randomly sets neuron outputs to zero during training in order to introduce more variance and thus avoid overfitting.
- dropconnect layers randomly sets weights to zero during training in order to introduce more variance and thus avoid overfitting.
- residual layer blocks take as input the output both of the previous layer, and that of the layer before that (i.e., they accept inputs that skipped a layer).
- recurrent layers accept as input both the outputs of the previous layer, and of the previous inference results of themselves.
- convolutional layers use an assignment of neurons to spatial positions to only connect a neuron to neurons from the previous layer that are spatially close.

Your answer is correct.

Correct

Marks for this submission: 6/6.

Question 8

Correct

Mark 6 out of 6

It is very important to have enough training data points for training DNNs in order to avoid easy overfitting, i.e., memorizing the data instead of learning a generalizable function. This is illustrated in the following.

Assume we want to learn a design and train a DNN-based binary classifier on some \(n \)-dimensional inputs. The DNN for now has one hidden layer, one output neuron (<0.5 output value means negative class, >0.5 positive), and uses the ReLU function as activation.

1. How many hidden neurons are needed to create the piecewise linear bump function fulfilling $$ \text{bump}(\vec{x}) = 0 \text{ if } \|x-x_{center}\|_1 > r_{bottom}, \text{ and } 1 \text{ if } \|x-x_{center}\|_1 < r_{top} ?$$
   => 4*n (write as "*<integer>*n*")
2. How many neurons are needed to memorize \(s_+=10 \) positive samples and \(s_-=15 \) negative ones in such a way that the DNN returns 0 everywhere except for a region around the input sample points, where there is a bump for each input with height of 1 (positive samples) or height of 0 (negative samples)? The radius of the region shall be smaller than the smallest distance between points.
   => 40 (integer)
3. How many neurons are needed when the DNN shall return 0.5 instead of 0 everywhere except around the training samples?
   => 100 (integer)

Correct

Marks for this submission: 6/6.

Question 9

Correct

Mark 3 out of 3

What is the product

$$ \left(\begin{matrix} 1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{matrix}\right) \cdot \left(\begin{matrix} 1\\0.5\\2\end{matrix}\right) ?$$

=> A vector with entries (8 , 18.5 , 29).

Correct

Marks for this submission: 3/3.

Question 10

Correct

Mark 6 out of 6

The process of calculating the gradient for DNN parameters is called backpropagation. It heavily uses the chain rule of differentiation: For \(f(x)=a(b(x)) \),

$$ \frac{\partial}{\partial x} f(x) = \frac{\partial a}{\partial b} (b(x)) \cdot \frac{\partial b}{\partial x} (x) . $$

For longer concatenations, such as \(g(w) = g_n(g_{n-1}(...(g_1(w)))) \), and with using helper variables \(w_i=g_i(g_(i-1)(...(g_1(w)))) \) this reads:

$$ \frac{\partial g}{\partial w} (w) = \frac{\partial g_5}{\partial w_4} (w_4) \cdot \frac{\partial g_4}{\partial w_3} (w_3) \cdot \dotsm \cdot \frac{\partial g_1}{\partial w} (w) $$

Here we also write \(\frac{\partial f}{\partial x} = \frac{\text{d} f}{\text{d} x} = \frac{\text{d}}{\text{d} x} f = \text{d/d}x f \).

This question is about (1) getting a better feeling of how gradients are calculated for DNNs, and (2) helping to read the common way of denoting gradients for multi-dimensional functions. Regarding the latter, let's have a look at this shorthand notation often used:
The variable name used in any \(\frac{\text{d}}{\text{d} x} \) notation (here: \(x \)) is actually a specifier telling for which parameter the function was derived. For a function with only a one-dimensional input, one could in principle use any name here: \(\frac{\text{d}}{\text{d} x} f = \frac{\text{d}}{\text{d} a} f = ... \). However, when specifying a function with multi-dimensional input, the positions/meanings of the parameters are usually referenced using a fixed name, e.g.: f(x,y,z) = <some function definition>. Then \(\frac{\text{d}}{\text{d} x} f \) uniquely identifies the derivative with respect to the *first* parameter of \(f \). Simultaneously, one would like to keep track of which input value variables are associated with which parameter position; so, if, e.g., we want to evaluate \(f \) or the derivative of \(f \) at some point, the points entries are usually then also named \(x,y,z \), leading to a double meaning of \(x \) in the statement \(\frac{\text{d}}{\text{d} x} f (x,y,z) \): The first occurrence of \(x \) references a parameter position, the second references a value (that is fed to this parameter position).
Going back to the chain rule for a function \(F(x) = f(g(x)) \), we could write the derivative in lengthy terms, with \(v_i \) as reference to the \(i\)-th input parameter of the function, as:

$$ (\frac{\text{d}}{\text{d} v_1} F) (x,y,z) = \left(\frac{\text{d}}{\text{d} v_1} f\right)(g(x,y,z)) \cdot \left(\frac{\text{d}}{\text{d} v_1} g\right) (x,y,z)$$

However, for the reasons explained above, the following version is preferred, using the same name for the parameter position and value, and leaving out brackets:

$$ (\frac{\text{d}}{\text{d} x} F) (x,y,z) = \frac{\text{d}}{\text{d} g} f(g(x,y,z)) \cdot \frac{\text{d}}{\text{d} x} g (x,y,z) $$

This is sometimes even further shortened to \(\frac{\text{d}}{\text{d} g} f(g) \cdot \frac{\text{d}}{\text{d} x} g (x,y,z) \), which also is the notation we work with below (where \(g \) will be the functions mapping input to layer intermediate output, and x,y,z function parameters / vector values will be both inputs and model parameters).

Let us apply this to a simple 3-hidden-layer DNN f with tanh-activations, for reasons of simplification just one neuron per layer, and SSE as cost function: Let \(y_{gt}(x) \) be the ground truth label for input \(x \), \(SSE(X;f) = \sum_{x \in X} (f(x)-y_{gt}(x))^2 \), and $$ f(x; w_1, w_2, w_3, b_1, b_2, b_3)= \tanh(b_3 + w_3\tanh(b_2 + w_2\tanh(b_1 + w_1x))) .$$ Now consider the loss $$ L(x; w_1, w_2, w_3, b_1, b_2, b_3) = \text{SSE}(\{x\}; f) $$ and set helper variables \(x_i = \text{output of layer i} \), \(s_i = \text{input to activation of layer i} \) such that $$ L(x; w_1, w_2, w_3, b_1, b_2, b_3) = \text{SSE}(\mathbf{x_3}(\mathbf{s_3}(w_3, b_3, \mathbf{x_2}(\mathbf{s_2}(w_2, b_2, \mathbf{x_1}(\mathbf{s_1}(w_1, b_1, x)))))) $$

Fill in the correct items to get the formula for the gradient with respect to single weights:

- \(\frac{\partial}{\partial w_3} L(x; w_1,w_2,w_3,b_1,b_2,b_3) = \) d/d x3 \((\) x3 – \(y_{gt}(x))^2 \) (x3) \(\cdot \) d/d\(s_3 \tanh \) (s3) \(\cdot \) d/d w3 (w3 \(\cdot x_2 +b_3 \)) (w_3)
- \(\frac{\partial}{\partial w_2} L(x; w_1,w_2,w_3,b_1,b_2,b_3) = \) d/d x3 \((\) x3 – \(y_{gt}(x))^2 \) (x3) \(\cdot \) d/d\(s_3 \tanh \)(s3) \(\cdot \) d/d x2 \((w_3x_2+b_3) \)(x2) \(\cdot \) d/ds2 tanh (s2) \(\cdot \) d/d w2 \((w_2x_1+b_2) (w_2) \)
- \(\frac{\partial}{\partial w_1} L(x; w_1,w_2,w_3,b_1,b_2,b_3) = \) d/d x3 \((\) x3 – \(y_{gt}(x))^2 \) (x3) \(\cdot \) d/d\(s_3 \tanh \)(s3) \(\cdot \) d/d x2 \((w_3x_2+b_3) \)(x2)) \(\cdot \) d/d\(s_2 \tanh \)(s2) \(\cdot \) d/d x1 \((w_2x_1+b_2) \) (x1) \(\cdot \) d/d\(s_1 \tanh \)(s1) \(\cdot \) d/d w1 \((w_1x+b_1) (w_1) \)

Note that for calculating d/d w2 L one can reuse 2 calculated factors from d/d w3 L, and for d/d w1 it is 4 factors from d/d w2 L. This allows to iteratively and very efficiently calculate the gradient with respect to each weight from output to input.

x3 | s3 | w3 | x2 | s2 | w2 | x1 | s1 | w1

Your answer is correct.

Correct

Marks for this submission: 6/6.

Question 11

Correct

Mark 5 out of 5

When storing input data, memory efficiency is key to be able to work with large batches of data points during gradient descent. In particular, each dimension of the input vector needs to be stored as a separate floating point value. Assuming each float is stored using a standard Float64 (i.e., IEEE 754 binary64 encoding), occupying each 64 bits.

What is the required memory (in kibibit Kb) of a real-valued input vector representing

- a 28x28 grayscale image, such as from the MNIST handwritten digits database: 49 Kb
- a 32x32 RGB colored image, such as from the CIFAR-10 and -100 databases: 192 Kb
- a 500x500 RGB image as input to many object detectors: 46875 Kb (nearly 50 MB!)
- a HD 16:9 RGB image: 172800 Kb (more than 150 MB!)
- a DCI 4K (4096×2160 pixels) RGBA image: 2211840 Kb (more than 2 GB!)

Correct

Marks for this submission: 5/5.

◄ 00. Quiz - Introduction to the Topic

Jump to...

02. Quiz - Stochastic and Statistics ►