1. In the bottom of the cells, write the name of the stochastic minimization algorithm for the function F(w) described by the pseudocode (*E.g.* inthe first cell write “Stochastic gradient descent”). Caution: one of the cells corresponds to a faulty algorithm that will not work (write “Wrong algorithm” in that cell).

**x**[t] denotes the sample drawn at time *t*, **w**[t] is the function argument at time *t*, **∇**F(x;w) denotes the stochastic gradient for the sample x and the function argument w.

|  |  |
| --- | --- |
| **w**[t+1] = **w**[t] – α[t] **∇**F(x[t];w[t]) | **g**[t+1] = μ **g**[t] + (1- μ) **∇**F(x[t]; w[t]) **⨀ ∇**F(x[t]; w[t])  **w**[t+1] = **w**[t] - α[t] **⨀ ∇**F(x[t]; w[t]) |
| g[t+1] = g[t] + **∇**F(x[t]; w[t]) **⨀ ∇**F(x[t]; w[t])  **w**[t+1] = **w**[t] - **⨀ ∇**F(x[t]; w[t]) | **v**[t+1] = μ **v**[t] – α[t]**∇**F(x[t]; w[t]+μ **v**[t])  **w**[t+1] = **w**[t] + **v**[t+1] |

1. Consider the following classifying ConvNet that maps a 64x64 RGB image into 10 numbers (corresponding to class odds). For each layer fill in the number of learnable parameters and the approximate number of multiply-and-add operations (e.g. taking a dot-product between two N-dimensional vectors takes N-1 multiply-and-add, but you can ignore ‘-1’ and just treat it as N). Ignore operations spent on non-linearities (they are not shown). For simplicity, assume that convolutions are implemented using nested loops.

|  |  |  |
| --- | --- | --- |
| Layer | Number of learnable parameters | Number of Mult-Adds |
| **InputLayer**(nOutChannels = 3, width = 64, height = 64) | 0 | 0 |
| **Conv2DLayer**(nOutChannels = 32,  filter\_size = (3,3), stride = 2, mode = ‘same’ ) |  |  |
| **Conv2DLayer**(nOutChannels = 64,  filter\_size = (3,3), stride = 2, mode = ‘same’ ) |  |  |
| **Conv2DLayer**(nOutChannels = 128, filter\_size = (3,3), stride = 2, mode = ‘same’ ) |  |  |
| **GlobalAveragePooling** | 0 | ≈128x8x8=8192 |
| **DenseLayer**(nOutUnits = 512) |  |  |
| **DenseLayer**(nOutUnits = 512) |  |  |
| **DenseLayer**(nOutUnits = 10) |  |  |

1. Consider the *sphere projection* layer that takes the input vector *x* and projects it onto the sphere with radius *R* (which is a learnable parameter): . Write the backpropagation equations (use formulas and/or pseudocode) that are needed to define this layer:

|  |
| --- |
|  |

1. Why adversarial networks are hard to train? Circle all correct answers.

* Because such training requires lots of labeled data samples
* Because the generator network cannot use convolutional layers
* Because the discriminator tends to collapse when trained for a fixed generator
* Because the discriminator and the generator have different input dimensionalities
* Because the generator tends to collapse when trained for a fixed discriminator
* Because a stochastic algorithm may not converge to a saddle point
* Because they are *adversarial*! That’s why!

1. The *reparameterization trick* can be used in the following situations (circle all correct answers):

* To implement the reshaping of the output of the convolutional layers for the input of the dense (fully-connected) layers
* To implement sampling in the decoder part of the variational autoencoder.
* To regularize autoencoder training by adding unit-variance Gaussian noise to the inputs (*denoising autoencoders*)
* To implement the *box-sampling* layer that takes as an input a 2N-dimensional vector that defines the boundaries of a bounding box in an N-dimensional space and outputs an N-dimensional vector drawn from the uniform distribution over this box.
* To promote convergence of adversarial network training by modifying the input to the discriminator

1. What is the loss minimized by the *word2vec* training? What is the optimization challenge? Give names (or one sentence summary) of the methods that are used to overcome the challenge.

|  |
| --- |
|  |

1. What are the ways to expand the effective receptive field (i.e. enable model to use longer history) during predictive sequential learning (next element prediction task)? Circle all correct answers.

* Using dilated convolutions instead of convolutions in a convolutional network
* Initializing the word embedding layer to word2vec (in the next word prediction case)
* Using LSTM units instead of regular recurrent (Elman) networks
* Decreasing the dimensionality of the hidden representation inside the recurrent neural network
* Adding noise to the transition matrices during forward propagation

1. The CTC-loss (Connectionist Temporal Classification loss) is suitable in the following application scenarios:

* Image captioning: an image TO a word sequence describing the image
* Handwritten recognition: a pen position and pressure sequence TO a letter sequence
* Part-of-Speech tagging: a sequence of words TO a sequence of part-of-speech labels
* Object detection: an image TO a set of bounding boxes containing objects of a certain class
* Video action parsing: a video TO a sequence of actions that occur in this video