CSCI-6522, Adv ML-II
Fall, 2023
Study Guide for Test #2

**(Please don't distribute this study guide.
The guide is for your study purpose only)**

------------------------ Chapter 02: Deep Neural Network -----------------------------------

01. Assume, for a 10 class-classification problem, you designed a fully connected (FC) neural network, which takes raw color image input of 1000 by 1000 pixels, with 2 hidden layers, and each hidden layer has 100 nodes. Later, you build a CNN by modifying the FC NN, replacing the 1st hidden layer with 6 different, 5 × 5 filters with stride 1 and padding 1. Then, you applied Max pooling of width 2 and stride 2. How many parameters will you have for the FC NN and CNN – show your calculations for these two networks.

Ans: Check class notes and do it yourself.

02. For deep CNN the following is given, where F, S, and P indicate filter, stride, and padding, respectively. (a) Fill in the "Parameters" column – showing your calculations. Also, indicate the layer depth in "Layer Depth" by L_i for the ith Layer.

Layer-name	Tensor Size = Memory	Operational Description	Parameters	Layer Depth
Input	227×227×3 = 154,587	-		
Conv1	55×55×96= 290,400	96 @ F:11×11, S 4, P 0		
M. Pool1	27×27×96=69,984	F: 3×3, S 2		
Norm1	27×27×96 = -	-		
Conv2	27×27×256=186,624	256 @ F : 5×5, S 1, P 2		
M. Pool2	13×13×256=43,264	F: 3×3, S 2		
Norm2	13×13×256 =-	-		
Conv3	13×13×384 =64,896	384 @ F : 3×3, S 1, P 1		
Conv4	13×13×384= 64,896	384 @ F : 3×3, S 1, P 1		

Conv5	**13×13×256 = 43,264**	**256 @ F : 3×3, S 1, P 1**		
M. Pool3	**6×6×256=9,216**	**F: 3×3, S 2**		
FC6	**4096 = 4096**	**4096 Nodes**		
FC7	**4096 = 4096**	**4096 Nodes**		
FC8	**1000 = 1000**	**1000 Nodes**		

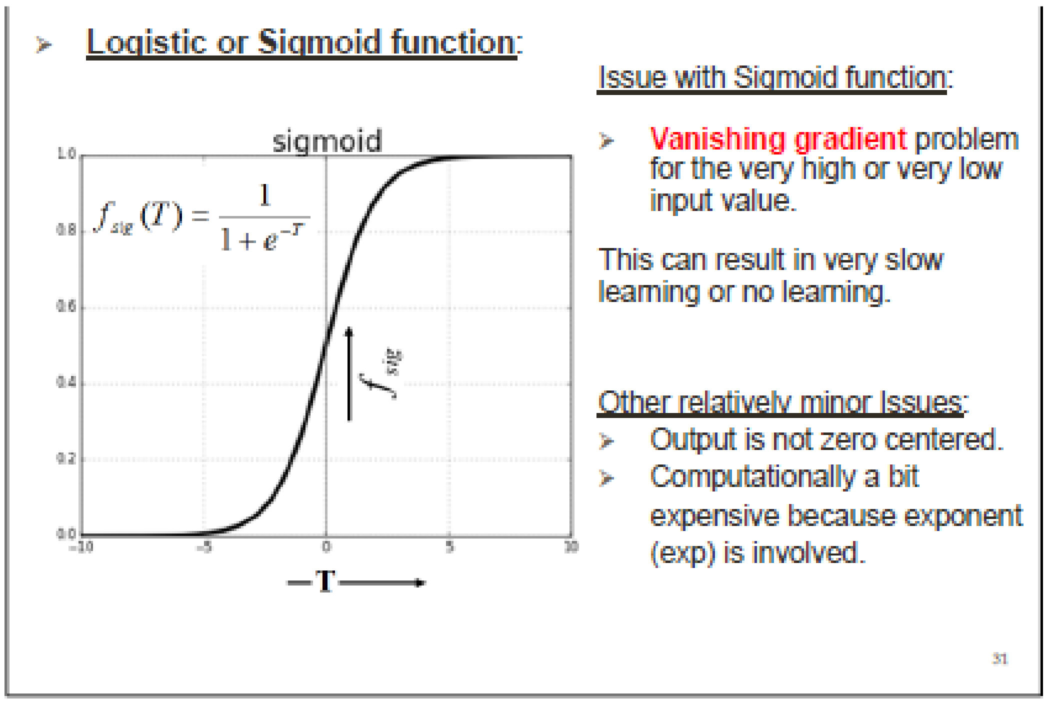
Ans:

Layer-name	Tensor Size = Memory	Operational Description	Parameters	Layer Depth
Input	227×227×3 = 154,587	-	0	
Conv1	55×55×96= 290,400	96 @ F:11×11, S 4, P 0	(11.11.3+1)96 = 34,944	L1
M. Pool1	27×27×96=69,984	F: 3×3, S 2	0	
Norm1	27×27×96 = -	-	0	
Conv2	27×27×256=186,624	256 @ F : 5×5, S 1, P 2	(5.5.96+1)256= 614,656	L2
M. Pool2	13×13×256=43,264	F: 3×3, S 2	0	
Norm2	13×13×256 =-	-	0	
Conv3	13×13×384 =64,896	384 @ F : 3×3, S 1, P 1	(3.3.256+1)384= 885,120	L3
Conv4	13×13×384= 64,896	384 @ F : 3×3, S 1, P 1	(3.3.384+1)384= 1,327,488	L4
Conv5	13×13×256 = 43,264	256 @ F : 3×3, S 1, P 1	(3.3.384+1)256=884,992	L5
M. Pool3	6×6×256=9,216	F: 3×3, S 2	0	
FC6	4096 = 4096	4096 Nodes	(6.6.256+1)4096 = 37,752,832	L6
FC7	4096 = 4096	4096 Nodes	(4096+1).4096= 16,781,312	L7

FC8	1000 = 1000	1000 Nodes	(4096+1)1000=4,097,000	L8
Total	936,323+		62,378,344	

03. In deep CNN learning, what are the issues with the sigmoid activation function – Explain and provide your conclusions.

Ans:



sigmoid
$f_{sig}(T) = \frac{1}{1+e^{-T}}$
f_{sig}
T

➢ **Logistic or Sigmoid function:**

Issue with Sigmoid function:

➢ Vanishing gradient problem for the very high or very low input value.

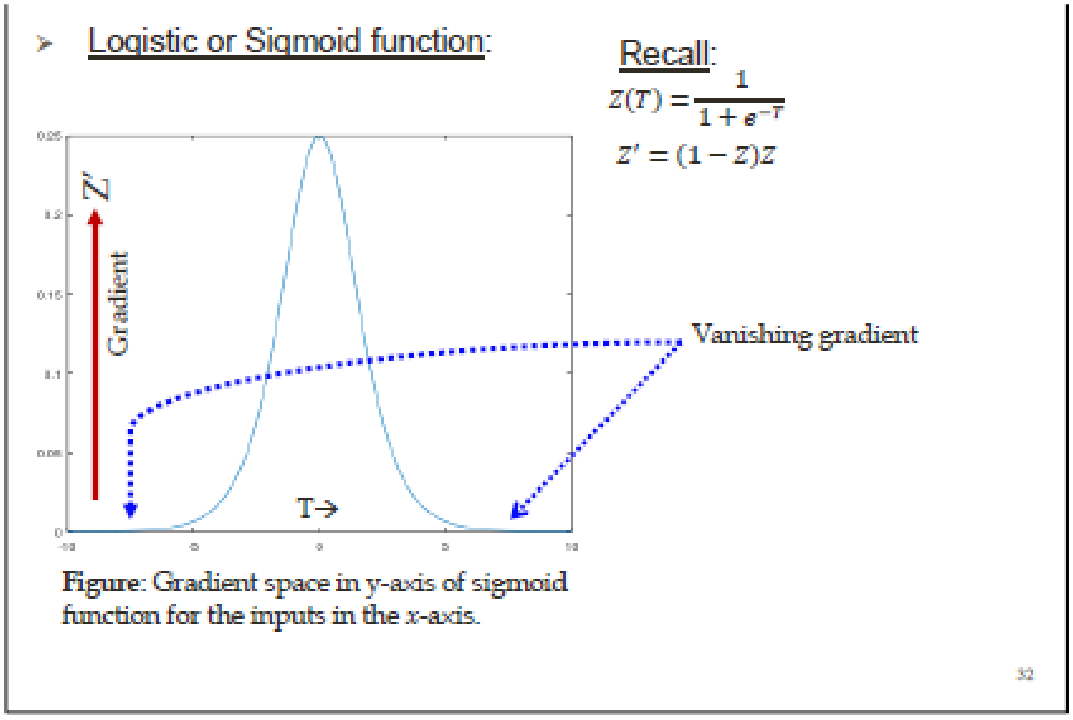
This can result in very slow learning or no learning.

Other relatively minor Issues:

➢ Output is not zero centered.

➢ Computationally a bit expensive because exponent (exp) is involved.

31



- <u>Logistic or Sigmoid function</u>:

<u>Recall</u>:

$$Z(T) = \frac{1}{1 + e^{-T}}$$

$$Z' = (1 - Z)Z$$

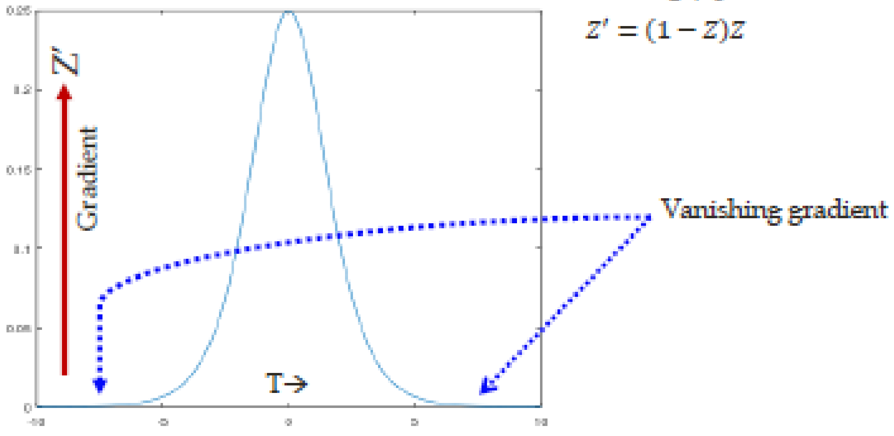
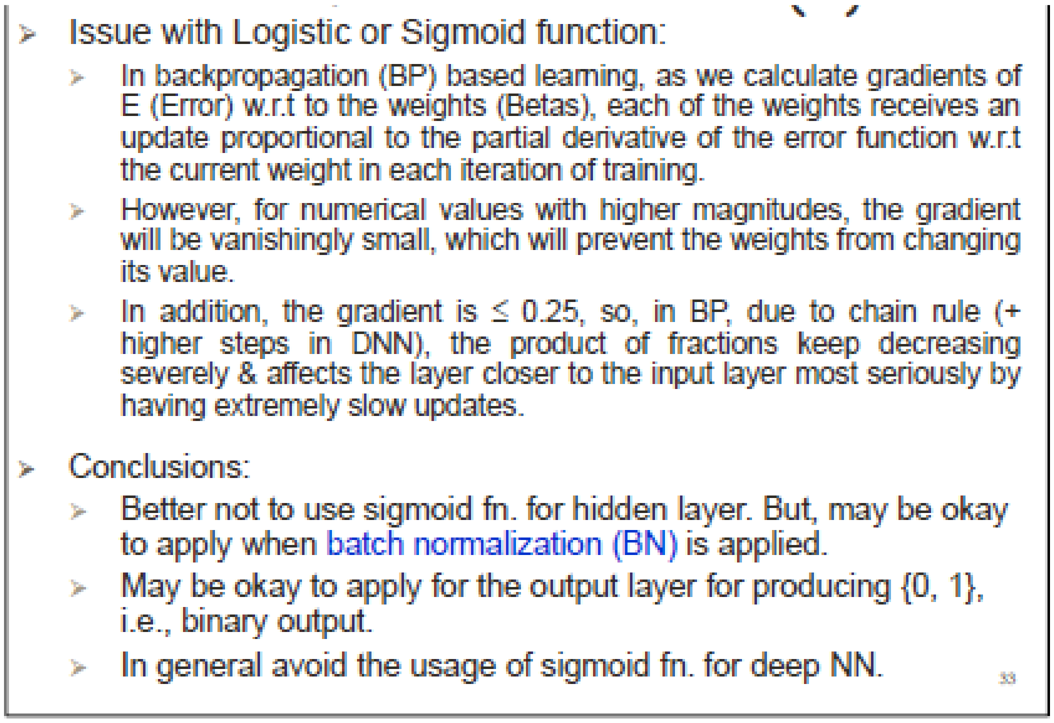


Figure: Gradient space in *y*-axis of sigmoid function for the inputs in the *x*-axis.

32

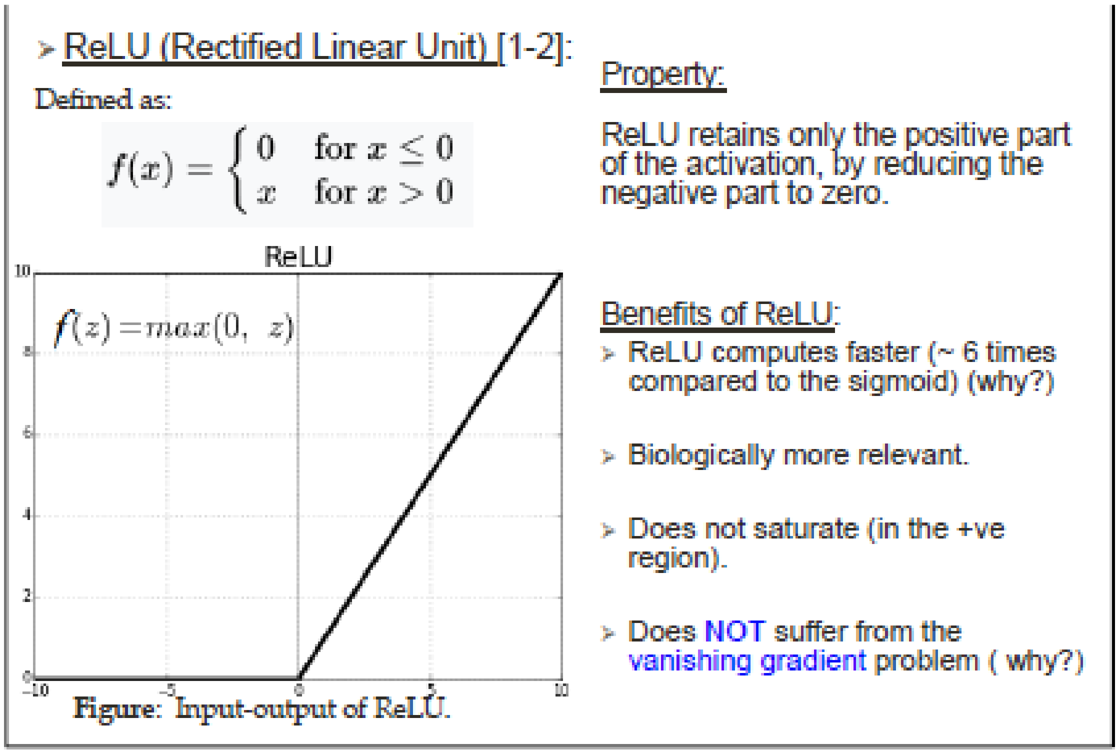


- Issue with Logistic or Sigmoid function:
  - In backpropagation (BP) based learning, as we calculate gradients of E (Error) w.r.t to the weights (Betas), each of the weights receives an update proportional to the partial derivative of the error function w.r.t the current weight in each iteration of training.
  - However, for numerical values with higher magnitudes, the gradient will be vanishingly small, which will prevent the weights from changing its value.
  - In addition, the gradient is ≤ 0.25, so, in BP, due to chain rule (+ higher steps in DNN), the product of fractions keep decreasing severely & affects the layer closer to the input layer most seriously by having extremely slow updates.
- Conclusions:
  - Better not to use sigmoid fn. for hidden layer. But, may be okay to apply when batch normalization (BN) is applied.
  - May be okay to apply for the output layer for producing {0, 1}, i.e., binary output.
  - In general avoid the usage of sigmoid fn. for deep NN.

33

04. Describe ReLU activation function

Ans:



➢ ReLU (Rectified Linear Unit) [1-2]:

Defined as:

$$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$$

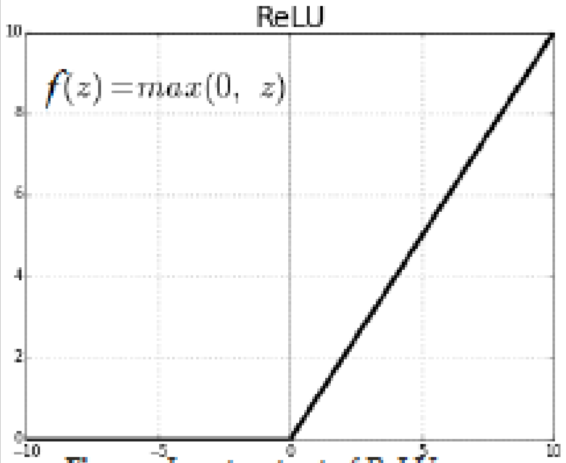


Figure: Input-output of ReLU.

Property:

ReLU retains only the positive part of the activation, by reducing the negative part to zero.

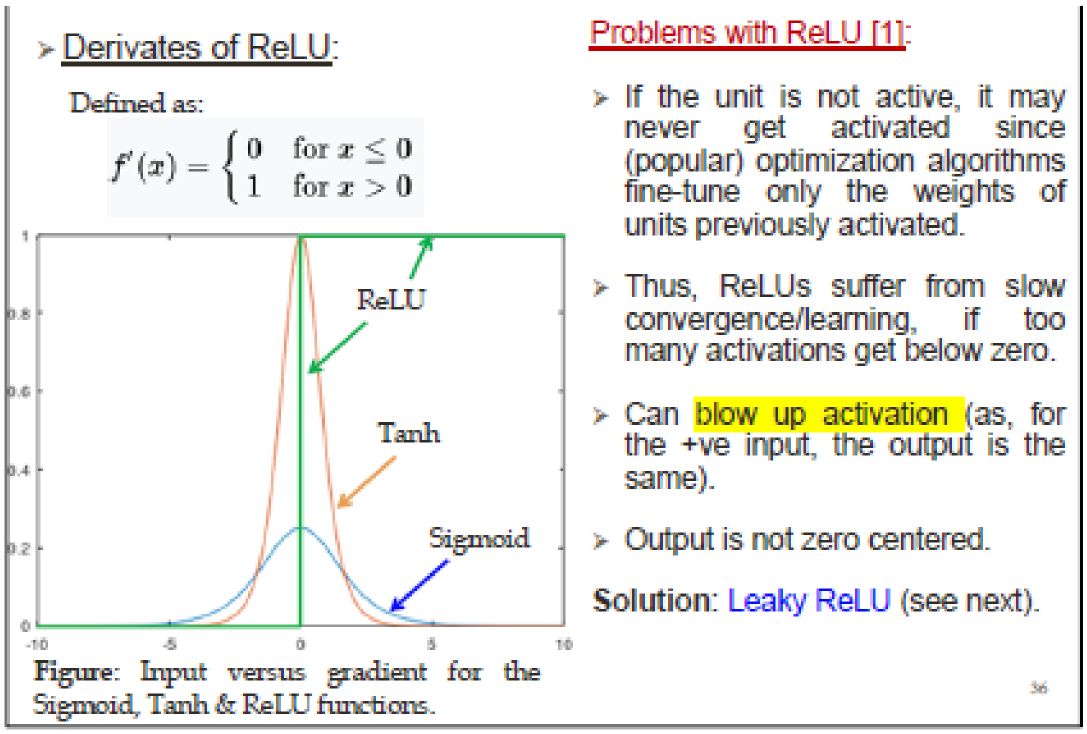
Benefits of ReLU:

➢ ReLU computes faster (~ 6 times compared to the sigmoid) (why?)

➢ Biologically more relevant.

➢ Does not saturate (in the +ve region).

➢ Does NOT suffer from the vanishing gradient problem (why?)



➢ Derivates of ReLU:

Defined as:

$$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$$

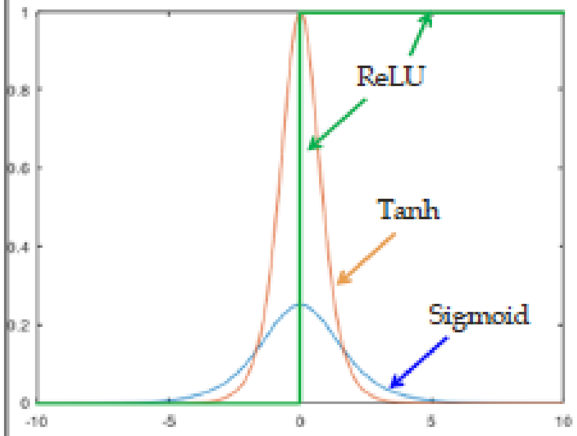


Figure: Input versus gradient for the Sigmoid, Tanh & ReLU functions.

Problems with ReLU [1]:

➢ If the unit is not active, it may never get activated since (popular) optimization algorithms fine-tune only the weights of units previously activated.

➢ Thus, ReLUs suffer from slow convergence/learning, if too many activations get below zero.

➢ Can blow up activation (as, for the +ve input, the output is the same).

➢ Output is not zero centered.

Solution: Leaky ReLU (see next).

36

05. Name 5 data augmentation methods for image data.

Ans:

Some of the popular data-augmentation methods are (mention any 5 from the list below):

- change contrast,
- add Gaussian noise,
- dropout regions,
- change hue/saturation,
- crop/pad,
- blur,
- flip horizontally/vertically,
- blend,
- alter color,
- rotate,
- change viewpoint,
- change size, etc.

06. What are the benefits of a Convolution Neural Network (CNN) over an equivalently deep Fully Connected (FC) Neural Network architecture?

Ans:

(1) CNN takes 2D input, but FC takes 1D input and loses the spatial information. Whereas CNN utilizes 2D information by extracting various useful spatial information using filters.

(2) FC, being fully connected, will have a huge number of weights to optimize in a deep learning model. For example, the input layer for a color image of 1000 by 1000 pixels, with 2 hidden layers, each with 100 nodes and 10 class-classification NN model, i.e., the configuration of L= [1000 × 1000 × 3 (R,G,B), 100, 100, 10] will have:

$$
\begin{aligned}
& [\{(1000 \times 1000 \times 3) + 1\} \times 100] + [(100+1) \times 100] + [(100+1) \times 10] \\
= \; & [3{,}000{,}001 \times 100] + [10{,}100] + [1{,}010] \\
= \; & [300{,}000{,}100] + [10{,}100] + [1{,}010] \\
= \; & 300{,}011{,}210
\end{aligned}
$$

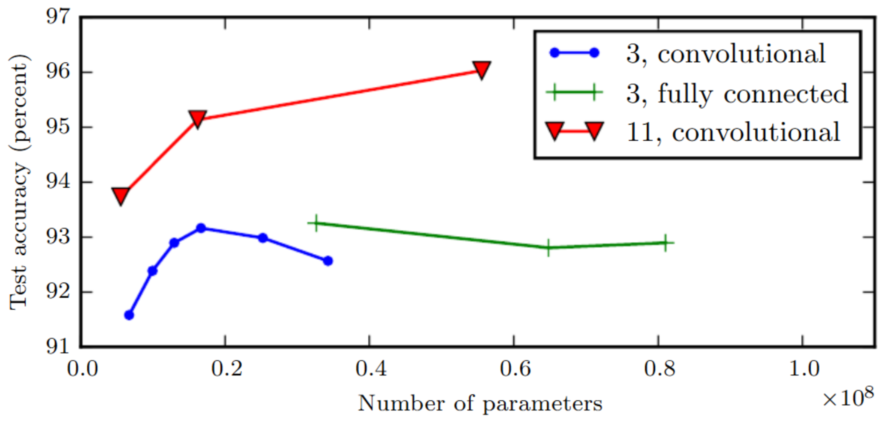


Figure 1: Effective number of parameters for various CNNs and FC NN.

(3) As shown in Figure 1, deeper models empirically tend to perform better, especially for CNN compared to FC NN. Increasing the number of parameters in layers of convolutional networks while increasing their depth is more effective in CNN than in FC NN.

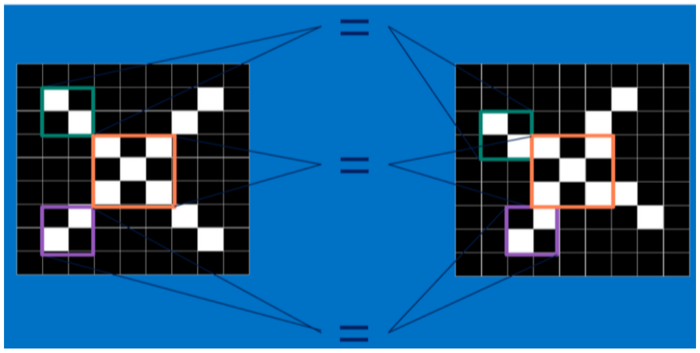


Figure 2: CNNs compare images piece (i.e., feature) by piece.

(4) As shown in Figure 2, by finding rough feature matches in roughly the same positions in two images, CNNs get a lot better at seeing similarity than whole-image matching schemes by FC NN.

07. Given the input and the filter below, compute the filter output by showing at least one calculation, where padding (p) =0, and stride (s) =1.

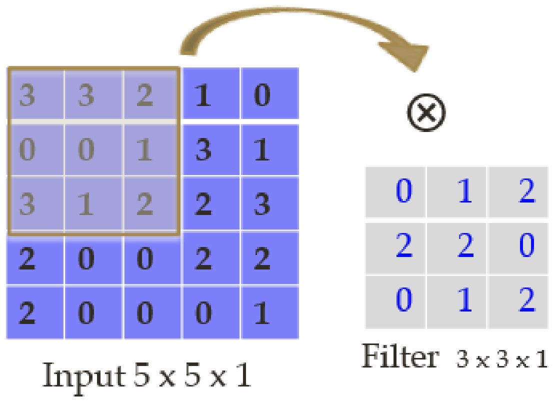
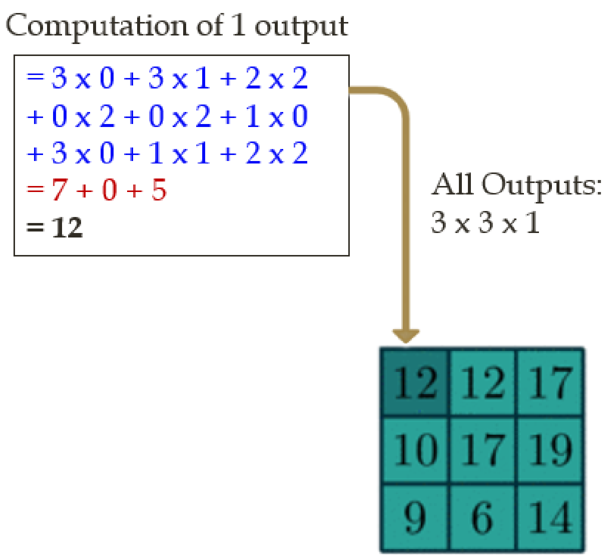


Figure 1: A filter of size 3 x 3 x 1 applying on an input of size 5 x 5 x 1.

Ans:



08. (a) For an input color image of size = 9 x 9 (x 3), and 6 filters of 7 x 7 (x3) with stride 1 and pad 3 – what would be the output size (volume)? (b) How many parameters are there in layer (a) in general?

Ans:
(a) Output for one filter, O = (I+2P-F)/S+1 = (9+2x3-7)/1+1 = 9.
Therefore, the output volume for 6 filters = (9 x 9) x 6 = 486.
(b) The filters contain almost all the parameters, including 1 bias parameter per filter. Each filter will have (7 × 7 × 3 + 1) = 148. Thus, 6 filters will have (6 × 148) = 888 parameters.

09. Max-pooling is preferable to Average-pooling – why?
Ans:
- In 2007, backpropagation was applied for the first time to a D/CNN-like architecture that used max pooling. It was shown empirically that the max-pooling operation was

vastly superior compared to the average-pooling because average-pooling has several corrections, whereas max-pooling does not need that.

- Also, to be biologically more relevant, we want to catch the maximum firing of a neuron from the receptive field rather than the average value of the firings.

10. (a) What is "Internal Covariate Shift" (ICS) – describe using a figure. (b) Why is ICS a concern in training D/CNN?

Ans:
(a) The training of a DCNN is convoluted by an internal covariate shift caused by changes to the distribution of each layer's inputs because of parameter changes in the previous layer – as shown in **Figure 1**.

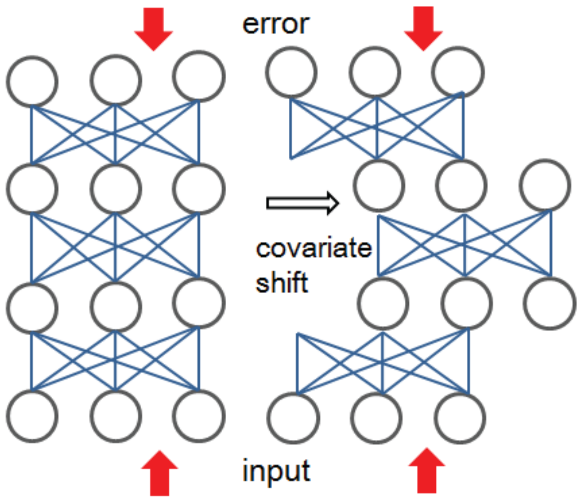


Figure 1: Internal Covariate Shift.

(b) (*i*) Due to the depth, a small perturbation in covariate shift leads to a large change in the later stage.
(*ii*) ICS slows down the training by requiring lower learning rates and careful parameter initialization and makes it very hard to train models with saturating nonlinearities.

11. (a) What is Batch Normalization (BN)? (b) Describe the BN algorithm and explain why does BN has learn-parameters: γ, β? (c) what are the advantages and disadvantages of BN?

Ans:
(a)

- Batch Normalization (BN) [1-2]: The idea is that, instead of just normalizing the inputs (for each dimensions or features) to the network, we normalize the inputs (for each dimensions or features) to layers within the network.
- It's called "batch" normalization because, during training, we normalize the activations of the previous layer for each batch, i.e., apply a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.
- BN computes the mean and variance estimates after mini-batches rather than over the entire training set (see equation below).

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

(b)

- **Algorithm** #1: BN Transform, applied to activation *x* over a mini-batch:

```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
       Parameters to be learned: $\gamma, \beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                          // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2$   // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$              // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$        // scale and shift
```

- Note: γ and β are the parameters to learn from the training (why γ, β?).
- Ans: These parameters recover the identity mapping to provide output in actual scale.
- These parameters could also be approximated as:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

(c)

- Advantages of (BN) [1-2]: The benefits of Batch Normalization are:
  - Networks train faster,
  - allows higher learning rates,
  - makes weights easier to initialize,
  - makes more activation functions viable,
  - simplifies the creation of deeper networks, and
  - provides some regularization (as it adds a little noise to the network).

- Disadvantages of BN: BN -
  - adds about 30% computational overhead, and
  - requires additional 25% of parameters per iteration.

12. (a) Discuss the weight initialization issue (assume Batch Normalization is not applied). (b) Describe the Xavier initialization approach, including the intuition behind it.

Ans:

(a)

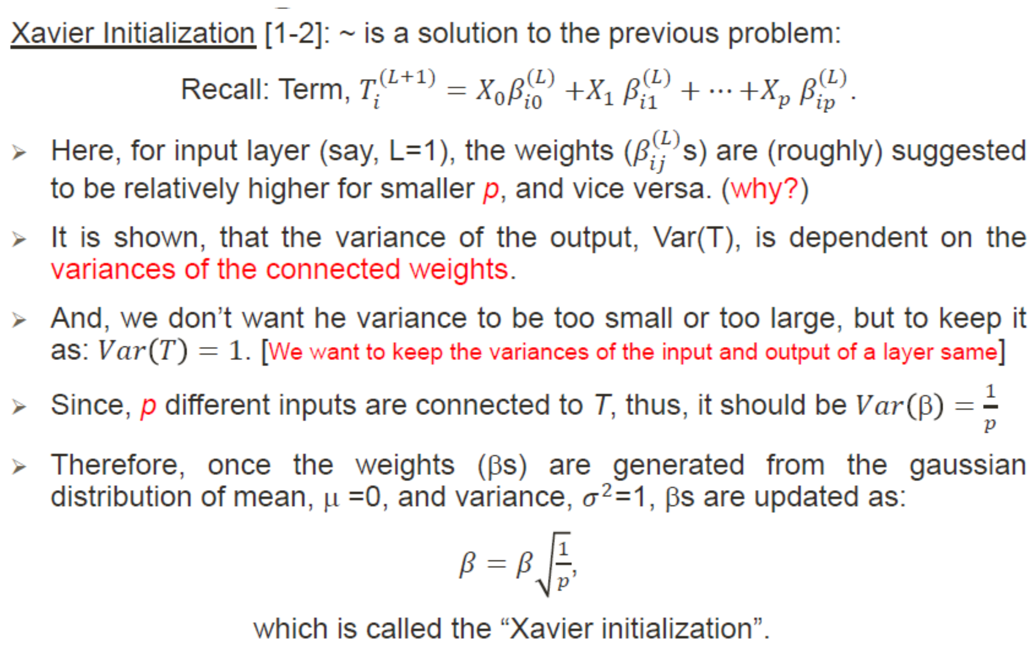
- Weight Initialization [1]: The cases are assumed without BN in place:

- **Case 1** "Zero Initialization": All weights (i.e., βs) are initialized to zero => The network will be in perfect symmetry => all node will produce the same output => the learning will be very slow.

- **Case 2** "Small value": If we initialize using mean (μ) = 0 and standard deviation (σ) = 0.01 (for example), for deep NN, the σ becomes zero or close to zero for the deeper layer for forward prop. Since the weight per layer became smaller, during BP (Back-Prop), the product get smaller further and thus, the weights do not get updated and the network does not learn.

- **Case 3** "Large value": say, $\beta \geq 1$, for sigmoid or tanh activation functions, the values become saturated, and during BP the gradient become zero. For ReLU, the values can explode.

(b)



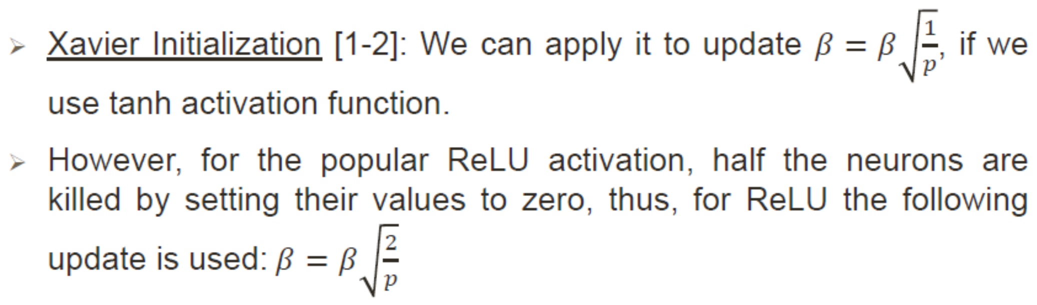
Xavier Initialization [1-2]: ~ is a solution to the previous problem:

Recall: Term, $T_i^{(L+1)} = X_0\beta_{i0}^{(L)} + X_1\,\beta_{i1}^{(L)} + \cdots + X_p\,\beta_{ip}^{(L)}$.

- Here, for input layer (say, L=1), the weights ($\beta_{ij}^{(L)}$s) are (roughly) suggested to be relatively higher for smaller *p*, and vice versa. (why?)
- It is shown, that the variance of the output, Var(T), is dependent on the variances of the connected weights.
- And, we don't want he variance to be too small or too large, but to keep it as: $Var(T) = 1$. [We want to keep the variances of the input and output of a layer same]
- Since, *p* different inputs are connected to *T*, thus, it should be $Var(\beta) = \frac{1}{p}$
- Therefore, once the weights (βs) are generated from the gaussian distribution of mean, μ =0, and variance, σ^2=1, βs are updated as:

$$\beta = \beta\sqrt{\frac{1}{p}},$$

which is called the "Xavier initialization".



- Xavier Initialization [1-2]: We can apply it to update $\beta = \beta\sqrt{\frac{1}{p}}$, if we use tanh activation function.
- However, for the popular ReLU activation, half the neurons are killed by setting their values to zero, thus, for ReLU the following update is used: $\beta = \beta\sqrt{\frac{2}{p}}$

13. (a) How did the Stochastic Gradient Descent (SGD) concept emerge? (b) Describe the SGD algorithm.

Ans:

(a)

- For a massive dataset for training in Machine Learning or BigData, computation of next β in the above equations involve, all the training instances (also called *Batch Gradient Descent*), which becomes computationally too expensive.
- Optimization algorithms that use only a single example at a time are sometimes called stochastic or sometimes online methods.
- However, the usage of a single instance or example to compute next β would be too small to be a representative or unbiased approach for the whole dataset.
- For deep learning, most algorithms fall somewhere in between, using more than one but less than all the training examples.

- These algorithms were traditionally called minibatch or minibatch stochastic methods and it is now common to simply call them stochastic methods or stochastic gradient descent (SGD).

(b)

- Algorithm 1.1 describes the SGD method:

```
Algorithm 1.1: Stochastic gradient descent (SGD)
Require: Learning rate $\alpha$; initial parameter $\beta(0)$; iteration $t = 0$.
    WHILE stopping criterion not met DO
        Sample a minibatch of $m$ examples, $\{(x,y)_1, \ldots, (x,y)_m\}$, from $N$ training points
        Compute $\beta(t+1)$ for $\forall_j$ as:
            $\beta_j(t+1) = \beta_j(t) + \frac{2\alpha}{m}\sum_{i=1}^{m}(y(i) - x^T(i)\ \beta)\ .\ x(i)_j$
        Or, $\beta_j(t+1) = \beta_j(t)\left(1 - \frac{2\alpha\lambda}{m}\right) + \frac{2\alpha}{m}\sum_{i=1}^{m}(y(i) - x^T(i)\ \beta)\ .\ x(i)_j$
    END WHILE
```

The 'or' part is the regularized version of the SGD algorithm.

14. (a) What is momentum? (b) Describe momentum in the context of the SGD algorithm. (c) Describe the preferred value(s) of the friction parameter and its impact. (d) Drawing a figure, describe the effect of momentum in SGD.

Ans:

(a)

- The method of **momentum** is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients.
- The momentum algorithm accumulates an **exponentially decaying moving average** of the past gradients and continues to move in their direction.
- Formally, the momentum algorithm introduces a variable ***v*** that plays the role of velocity—it is the direction and speed at which the parameters move through parameter space.
- The velocity is set to an exponentially decaying average of the negative gradient.

- A hyperparameter δ ∈ [0, 1), called the friction parameter, determines how quickly the contributions of previous gradients exponentially decay.

(b)

Algorithm 1.2: Stochastic gradient descent (SGD) with momentum

```
Require: Learning rate $\alpha$; initial parameter $\beta(0)$; momentum parameter $\delta$; velocity $v(-1) = 0$; iteration $t = 0$.
  WHILE stopping criterion not met DO
        Sample a minibatch of $m$ examples, $\{(x,y)_1, \dots, (x,y)_m\}$, from $N$ training points
        Compute $\beta(t+1)$ for $\forall_j$ as:
              $v(t) = v(t-1)\,\delta + \frac{2\alpha}{m}\sum_{i=1}^{m}(y(i) - x^T(i)\,\beta)\,.\,x(i)_j$
              $\beta_j(t+1) = \beta_j(t) + v(t)$
              Or, $\beta_j(t+1) = \beta_j(t)\left(1 - \frac{2\alpha\lambda}{m}\right) + v(t)$
  END WHILE
```

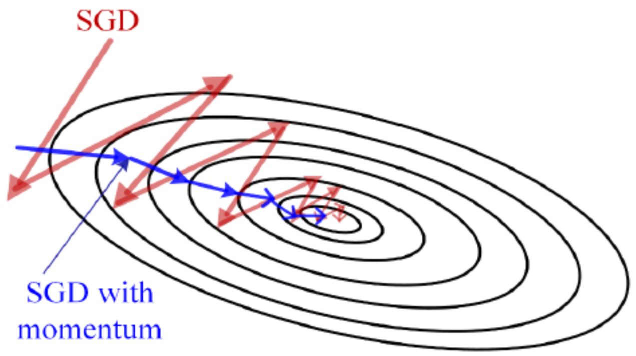
- As shown in Algorithm 1.2, the velocity *v* accumulates the gradient elements $\frac{2\alpha}{m}\sum_{i=1}^{m}(y(i) - x^T(i)\ \beta)\ .\ x(i)_j$.

(c)

- The larger the δ is relative to α, the more previous gradients affect the current direction.
- Here, the size of the step depends on how large and how aligned a sequence of gradients are. The step size is largest when many successive gradients point in exactly the same direction.
- Common values of δ used in practice include 0.5 and **0.9 to 0.99**.
- Like the learning rate, δ may also be adapted over time. Typically it begins with a small value and is later raised.
- It is less important to adapt δ over time than to shrink α over time.

(d)

- The effect of momentum is illustrated in the **Figure** below:



- **Figure 1.1:** Momentum aims primarily to solve two problems:
  - poor conditioning of the Hessian matrix and
  - variance in the stochastic gradient.
- Here, the contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix.
- The heavily zig-zak **red path** indicates the path of GD/SGD without the momentum.
- The **blue path** cutting across the contours shows the path followed by the momentum learning rule, which is a less zig-zak path that helps faster convergence.

15. (a) What are the hyperparameters? (b) Briefly describe 3 different ways to optimize hyperparameters.

Ans:
(a) Hyperparameters are the variables (e.g., learning rate, friction parameter, dropout rate, etc.) set manually with some default or pre-determined value before starting the training.

(b) We can optimize or tune the hyperparameters in the following ways:

- **Trial & Error**: In this setup, with some sample data, plot the learning curves and tune the hyperparameters to learn faster.

- **Grid Search**: ~ an exhaustive searching through a manually specified subset of the hyperparameter space (Recall the grid search for C and γ for SVM in ch5).

- **Random Search**: Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly and can outperform Grid search, especially when only a small number of hyperparameters affect the final performance of the machine learning algorithm.

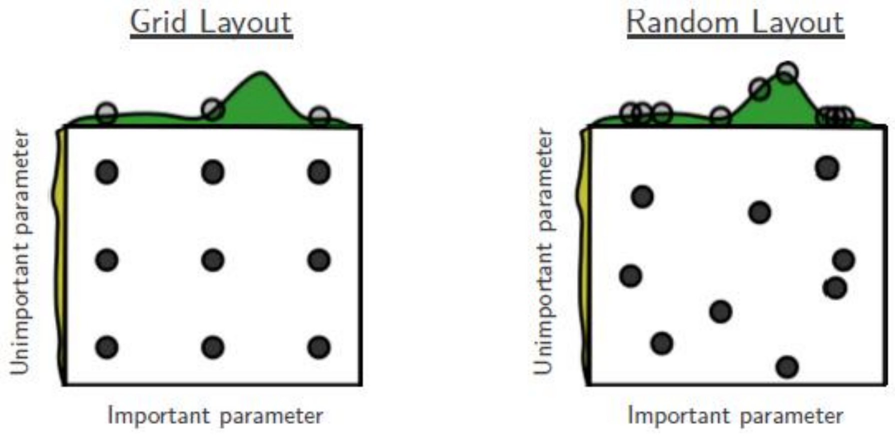


Figure: Here, grid and random search of nine trials for optimizing a function $f(x, y) = g(x)+h(y) \approx g(x)$ with low effective dimensionality have been shown.

- Above each square, $g(x)$ is shown in green, and on the left of each square $h(y)$ is shown in yellow.
- With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g.
- This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

- **Evolutionary Optimization / Genetic Algorithm / Population-Based Training**: Evolutionary optimization and Population-Based Training (PBT) learns both hyperparameter values and network weights.
  - These approaches make no assumptions regarding model architecture, loss functions or training procedures and are found to perform the best.
  - Given the same computational resource, the PBT system has been shown to achieve better accuracy and faster convergence compared to existing methods.

16. Briefly describe snapshot ensembles?

Ans:

- Ensembles of neural networks are known to be much more robust and accurate than individual networks.

- However, training multiple deep networks for model averaging is computationally expensive.
- Ensembling multiple neural networks, at no additional training cost, can be achieved by converging to several local minima while training, along its optimization path and saving the model parameters. Restarting can achieve similar or better results (at a higher cost, though).
- Snapshot Ensembling is simple yet surprisingly effective.

17. (a) What is dropout? (b) How is the dropout applied during training? (c) Why does dropout work? (d) If the dropout is applied during training, how do we handle the test case?

Ans:

(a)

- In dropout, we can effectively remove a **unit** from a network by multiplying its output value by zero.
- Dropout is a simple way to prevent DNN from overfitting & improve model accuracy.
- Dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks.
- Dropout trains the ensemble consisting of all sub-networks that can be formed by removing **non-output** units from an underlying base network.

(b)

- To train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as SGD.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all the input and hidden units in the network.
- The mask for each unit is sampled independently from all the others.
- The probability of sampling a mask value of one (causing a unit to be included or retained with probability ***p***) is a hyperparameter (called dropout hyperparameter) fixed before training begins.
- Typically, an input unit is included with probability 0.8, and a hidden unit is included with probability 0.5.
- Dropout is more common in FC, but conv. layer usually apply it by dropping an entire feature map, selected randomly.
- Dropout forces an NN to learn more **robust features** that are useful in conjunction with many different random subsets of the other neurons.

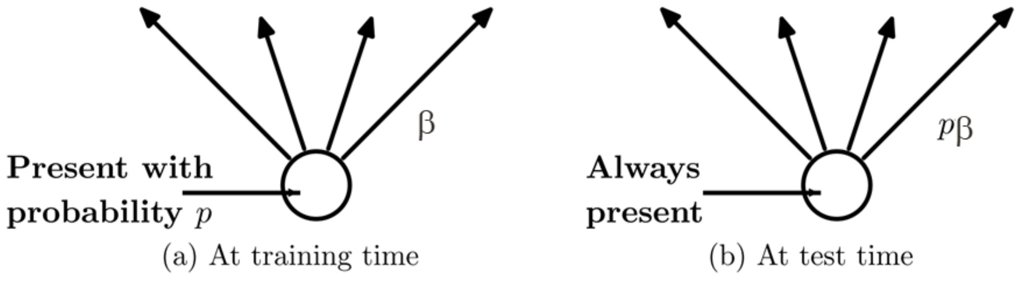
(c)

- The power of dropout arises from the fact that the masking noise is applied to the hidden units.
- This can be seen as a form of highly intelligent, adaptive destruction of the information content of the input rather than the destruction of the raw values of the input.

- For example, if the model learns a hidden unit h_i that detects a face by finding the nose, then dropping h_i corresponds to erasing the information that there is a nose in the image.
- The model must learn another h_i, either that redundantly encodes the presence of a nose, or that detects the face by another feature, such as the mouth.
- Traditional noise injection techniques that add unstructured noise at the input are not able to randomly erase the information about a nose from an image of a face unless the magnitude of the noise is so great that nearly all the information in the image is removed.
- Destroying extracted features rather than original values allows the destruction process to make use of all the knowledge about the input distribution that the model has acquired so far.

(d)

- Note, dropout is used while training, but during testing, full network is used without any dropout.
- Recall: in dropout, during training, each unit of a layer's output is retained with probability $\boldsymbol{p}$; else it is set to zero with probability $(1-\boldsymbol{p})$.
- Therefore, in **test** to get the averaging effect of the corresponding ensembles of subnetworks, units' output weights (βs) are multiplied with the nodes' corresponding dropout probability, $\boldsymbol{p}$ (see Figure below).



18. Explain "Stochastic Depth" regularization in the context of a deep network.

Ans:

Deep Networks with Stochastic Depth [1]:

- Many layers can be highly desirable at test time, however, training very deep networks comes with its own set of challenges:
  - The gradients can vanish,
  - the forward flow often diminishes, and
  - the training time can be painfully slow.

- ***Stochastic Depth*** (SD) method addresses these issues, where:
  - training uses short networks, but
  - test uses a deeper network.

- In SD, we start with very deep networks but during training, for each mini-batch, randomly drop a subset of layers and bypass them with the identity function.

- This simple approach complements the recent success of residual networks (ResNets [2-3]; ILSVRC winner in 2015: 3.57% error)

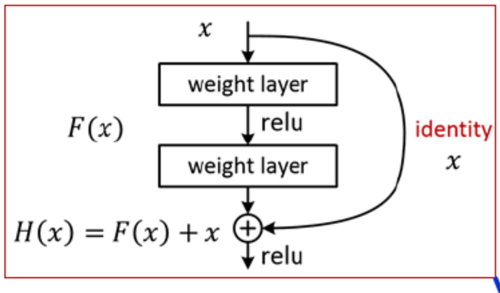


Figure 1: ResNet Block [2]. He et al. reformulated the layers of the network and forced them to learn residual functions with reference to their preceding layer inputs. This allowed for errors to be propagated directly to the preceding units and thus made these networks easier to optimize and easier to train.

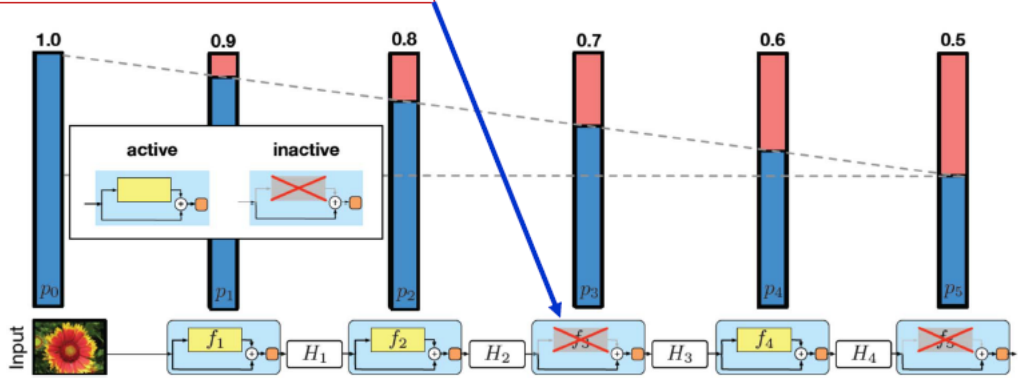


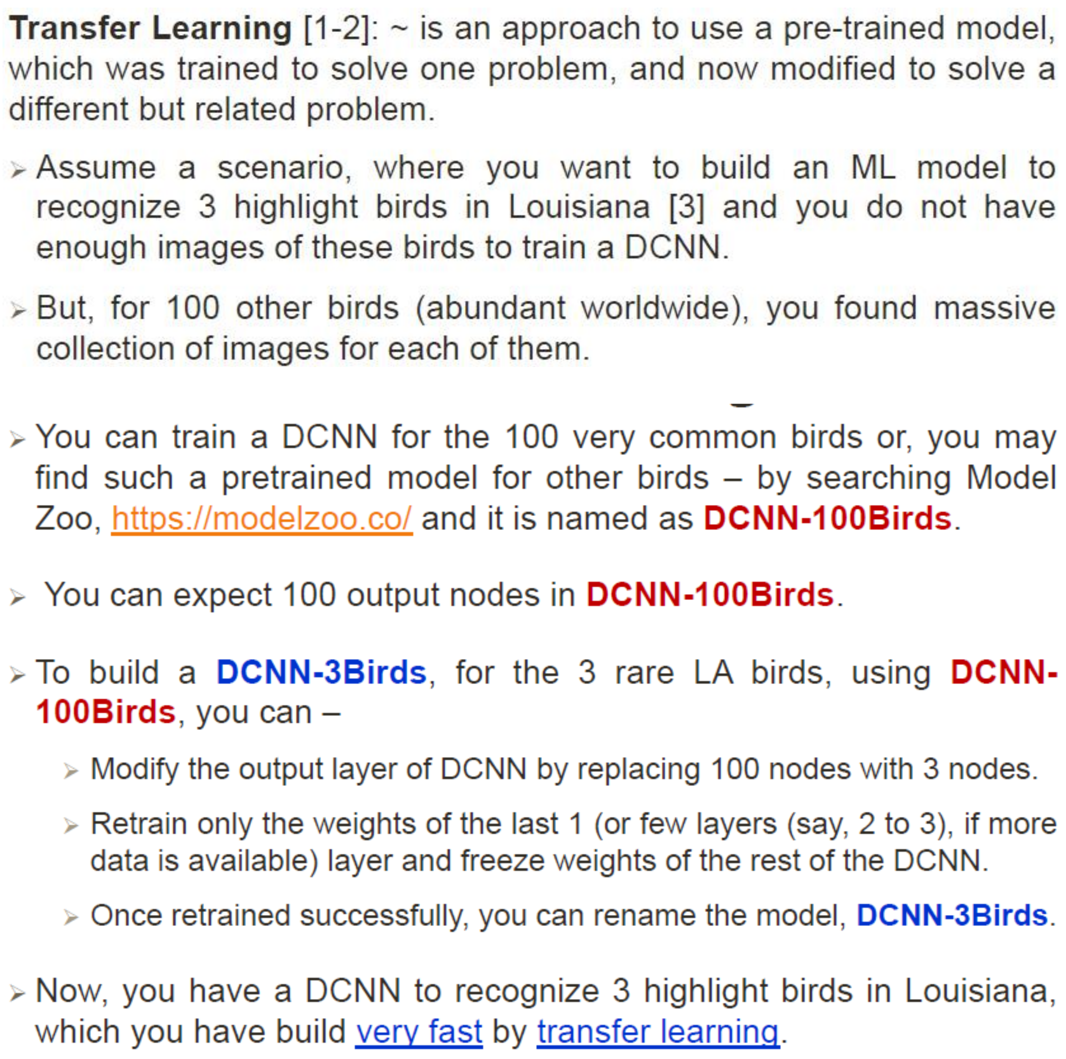
Figure 2: The linear decay of p_l illustrated on a ResNet with stochastic depth for p_0=1 and p_L = 0:5. Conceptually, we treat the input to the first ResBlock as H_0, which is always active.

81

19. (a) What is Transfer Learning (TL)? Describe TL using a scenario. (b) Why should the TL work?

Ans:

(a)


Transfer Learning [1-2]: ~ is an approach to use a pre-trained model, which was trained to solve one problem, and now modified to solve a different but related problem.

- Assume a scenario, where you want to build an ML model to recognize 3 highlight birds in Louisiana [3] and you do not have enough images of these birds to train a DCNN.
- But, for 100 other birds (abundant worldwide), you found massive collection of images for each of them.
- You can train a DCNN for the 100 very common birds or, you may find such a pretrained model for other birds – by searching Model Zoo, https://modelzoo.co/ and it is named as **DCNN-100Birds**.
- You can expect 100 output nodes in **DCNN-100Birds**.
- To build a **DCNN-3Birds**, for the 3 rare LA birds, using **DCNN-100Birds**, you can –
  - Modify the output layer of DCNN by replacing 100 nodes with 3 nodes.
  - Retrain only the weights of the last 1 (or few layers (say, 2 to 3), if more data is available) layer and freeze weights of the rest of the DCNN.
  - Once retrained successfully, you can rename the model, **DCNN-3Birds**.
- Now, you have a DCNN to recognize 3 highlight birds in Louisiana, which you have build very fast by transfer learning.

(b)

- Why should DCNN-3Birds work? Ans:
  - The first layer of deep neural network, trained on natural images, learns features like Gabor filters and color blobs, which are not specific to a particular dataset or task, but general in that they are applicable to many datasets and tasks.
  - Features eventually transit from general to specific by the last layer of the network.
  - Therefore, the unchanged weights and layers of DCNN-100Birds in DCNN-3Birds can extract the features of "**birdness**", which are helpful in general for classifying any bird.
  - Thus, the retrained weights of the last layer(s) can utilize these feature-valuables and can effectively map them to the 3 target classes in DCNN-3Birds.
- Transfer learning is often found better than start training the network, initialized with the random weights.

19. For VGG 16, the following Table is given, where F, S, and P indicate filter, stride, and padding, respectively. Fill in the "Parameters" column – showing your calculations. Also, indicate the layer depth in "Layer Depth" by L_i for the ith Layer.

Layer-name	Tensor Size = Memory	Operational Description	Parameters	Layer #
Input	224×224×**3** = 150,528	-		
Conv1-1	224×224×**64**= **3**,211,264	**64** @ F:3×3, S 1, P 1		
Conv1-2	224×224×64= **3**,211,264	64 @ F:3×3, S 1, P 1		
M. Pool1	112×112×**64**= 802,816	F: 2×2, S 2		
Conv2-1	112×112×128= **1**,605,632	128 @ F : 3×3, S 1, P 1		
Conv2-2	112×112×128= **1**,605,632	128 @ F : 3×3, S 1, P 1		
M. Pool2	56×56×**128**= 401,408	F: 2×2, S 2		
Conv3-1	56×56×256 = 802,816	256 @ F : 3×3, S 1, P 1		
Conv3-2	56×56×256 = 802,816	256 @ F : 3×3, S 1, P 1		
Conv3-3	56×56×256 = 802,816	256 @ F : 3×3, S 1, P 1		
M. Pool3	28×28×**256**= 200,704	F: 2×2, S 2		
Conv4-1	28×28×512= 401,408	512 @ F : 3×3, S 1, P 1		
Conv4-2	28×28×512= 401,408	512 @ F : 3×3, S 1, P 1		
Conv4-3	28×28×512= 401,408	512 @ F : 3×3, S 1, P 1		
M. Pool4	14×14×**512**= 100,352	F: 2×2, S 2		
Conv5-1	14×14×512= 100,352	512 @ F : 3×3, S 1, P 1		
Conv5-2	14×14×512= 100,352	512 @ F : 3×3, S 1, P 1		
Conv5-3	14×14×512= 100,352	512 @ F : 3×3, S 1, P 1		

M. Pool5	7×7×512= 25,088	F: 2×2, S 2		
FC6	4096 = 4096	4096 Nodes		
FC7	4096 = 4096	4096 Nodes		
FC8	1000 = 1000	1000 Nodes		

Ans:

Layer-name	**Tensor Size = Memory**	**Operational Description**	**Parameters**	**Layer #**
Input	224×224×**3** = 150,528	-	0	
Conv1-1	224×224×**64**= **3**,211,264	**64** @ F:3×3, S 1, P 1	(3.3.**3**+1) 64 = 1,792	L1
Conv1-2	224×224×64= **3**,211,264	64 @ F:3×3, S 1, P 1	(3.3.**64**+1) 64 = 36,928	L2
M. Pool1	112×112×**64**= 802,816	F: 2×2, S 2	0	
Conv2-1	112×112×128= **1**,605,632	128 @ F : 3×3, S 1, P 1	(3.3.**64**+1)128 = 73,856	L3
Conv2-2	112×112×128= **1**,605,632	128 @ F : 3×3, S 1, P 1	(3.3.128+1)128 = 147,584	L4
M. Pool2	56×56×**128**= 401,408	F: 2×2, S 2	0	
Conv3-1	56×56×256 = 802,816	256 @ F : 3×3, S 1, P 1	(3.3.**128**+1)256= 295,168	L5
Conv3-2	56×56×256 = 802,816	256 @ F : 3×3, S 1, P 1	(3.3.256+1)256= 590,080	L6
Conv3-3	56×56×256 = 802,816	256 @ F : 3×3, S 1, P 1	(3.3.256+1)256= 590,080	L7
M. Pool3	28×28×**256**= 200,704	F: 2×2, S 2	0	
Conv4-1	28×28×512= 401,408	512 @ F : 3×3, S 1, P 1	(3.3.**256**+1)512= **1**,180,160	L8
Conv4-2	28×28×512= 401,408	512 @ F : 3×3, S 1, P 1	(3.3.512+1)512= **2**,359,808	L9
Conv4-3	28×28×512= 401,408	512 @ F : 3×3, S 1, P 1	(3.3.512+1)512= **2**,359,808	L10
M. Pool4	14×14×**512**= 100,352	F: 2×2, S 2	0	
Conv5-1	14×14×512= 100,352	512 @ F : 3×3, S 1, P 1	(3.3.**512**+1)512= **2**,359,808	L11
Conv5-2	14×14×512= 100,352	512 @ F : 3×3, S 1, P 1	(3.3.512+1)512= **2**,359,808	L12
Conv5-3	14×14×512= 100,352	512 @ F : 3×3, S 1, P 1	(3.3.512+1)512= **2**,359,808	L13
M. Pool5	7×7×512= 25,088	F: 2×2, S 2	0	
FC6	4096 = 4096	4096 Nodes	(7.7.512+1)4096 = **102,764,544**	L14
FC7	4096 = 4096	4096 Nodes	(4096+1).4096= **16**,781,312	L15
FC8	1000 = 1000	1000 Nodes	(4096+1)1000= **4**,097,000	L16
Total	**15,237,608+**		**138,357,544**	

20. What are the benefits of smaller size filters? Explain.

Ans: The benefits are, the smaller filter will need a lower number of parameters to be optimized and the depth will make the decision function more discriminative being more nonlinear. For example, three 3×3 conv. layers have a 7×7 effective receptive field. The benefits of having three 3×3 conv. layers instead of one 7×7 are:

(1) It incorporates three non-linear rectification layers instead of one, which makes the decision function more discriminative.

(2) It decreases the number of parameters: if both the input and the output of a three-layer 3 × 3 convolution stack has C channels, the stack is parametrized by $3\,(3^2 . C^2) = 27.C^2$ weights; at the same time, a single 7×7 conv. layer would require $7^2 . C^2 = 49.C^2$ parameters, which is 81.5% more.

(a) This can be realized as imposing a regularization on the 7×7 conv. filters, forcing them to have a decomposition through the 3×3 filters (with non-linearity injected in between).

(b) Therefore, the incorporation of 1×1 conv. layers is a way to increase the nonlinearity of the decision function without affecting the receptive fields of the conv. Layers.

21. What are problems in the naïve version of the GoogLeNet of a smaller filter? How is the problem solved? Explain.

Ans:

The operating cost of the naïve inceptions modules is high. Also, the pooling layer preserves the layer depth. Therefore, the stacking will increase the layer depth in the subsequent layers. These problems are shown in Figure 1:

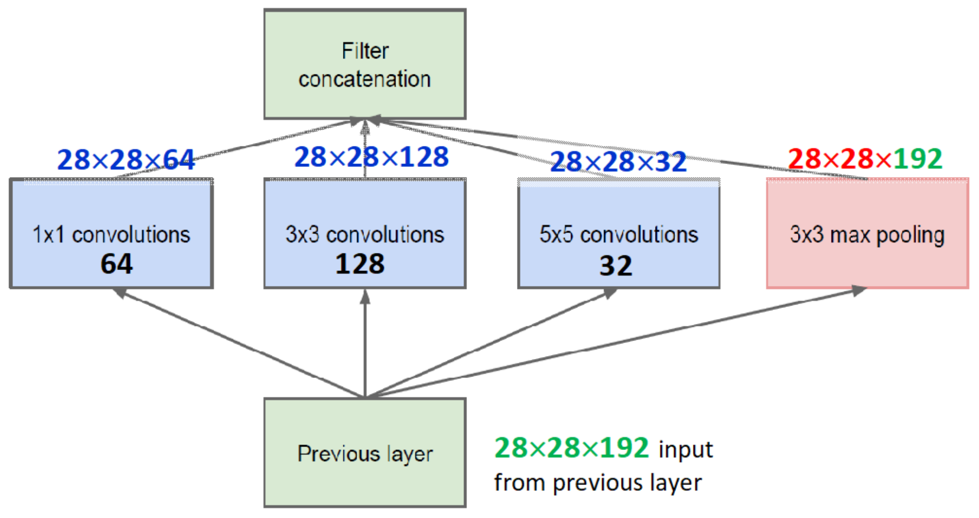


Figure1: Inception module, naïve version

The output of the filter-concatenation is: 28×28×(64+128+32+192) = 28×28×(416). We see that the layer depth increases from 192 to 416 in Figure 1.

Also, the filer operation counts for the conv. layers are:

$$[1\times1]: 28\times28\times\ 64\times1\times1\times192 = 9.6\text{M}$$
$$[3\times3]: 28\times28\times128\times\mathbf{3}\times\mathbf{3}\times192 = \mathbf{173.4}\text{M}$$
$$\underline{[5\times5]: 28\times28\times\ 32\times5\times5\times192 = \mathbf{120.4}\text{M}}$$
$$\text{Total} = \mathbf{303.4}\text{M}$$

The solution to the aforementioned problems is to apply the ‘bottleneck’ layer, which is a 1x1 conv. layer, that can reduce the feature depth and reduce the number of operations needed. This concept is applied in Figure 2:

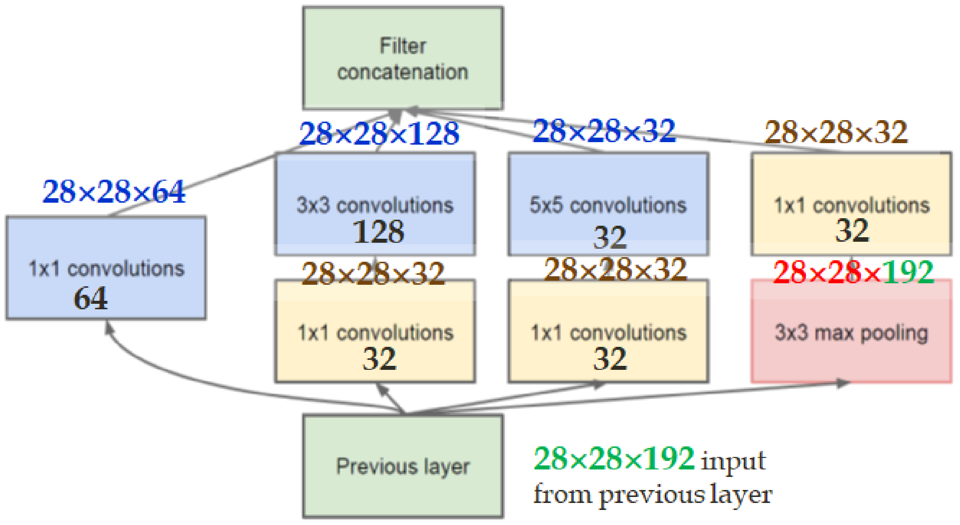


Figure 2: Inception module, with **dimension reductions**

From Figure 2, we see the output of the filter-concatenation is: 28×28×(64+128+32+32) = 28×28×(**256**), which was 28×28×(**416**), previously.

<u>The filer operation counts are:</u>

$$[1\times1]: 28\times28\times\ 64\times1\times1\times192 = 9.6\text{M}$$
$$[1\times1]: 28\times28\times\ 32\times1\times1\times192 = 4.8\text{M}$$
$$[1\times1]: 28\times28\times\ 32\times1\times1\times192 = 4.8\text{M}$$
$$[3\times3]: 28\times28\times128\times\mathbf{3}\times\mathbf{3}\times32 = \mathbf{28.9}\text{M}$$
$$\underline{[5\times5]: 28\times28\times\ 32\times5\times5\times32 = \mathbf{20.0}\text{M}}$$
$$\text{Total} = \mathbf{68.1}\text{M}$$

whereas, the total conv. operations were 303.4M.

22. (a) What is the purpose of the auxiliary output in the GoogLeNet? (b) What was the intuition behind the ResNet block?

Ans: (a) The purpose of the auxiliary output is to insert an additional gradient at the lower layers during backpropagation.

(b) Experiment on the CIFAR-10 database with 20-layer and 56-layer "plain" networks shows the deeper network has higher training as well as test errors.

The conclusion drawn from the experiment was that the deeper network is harder to optimize; however, a shallow network is a subset of a deep network. Therefore, within a 56-layer plain network, we can provide the output of the 20th layer to a higher layer to produce output ≥ the output of the 20th layer. Based on these conclusions, the residual learning block was designed as shown in Figure 1:

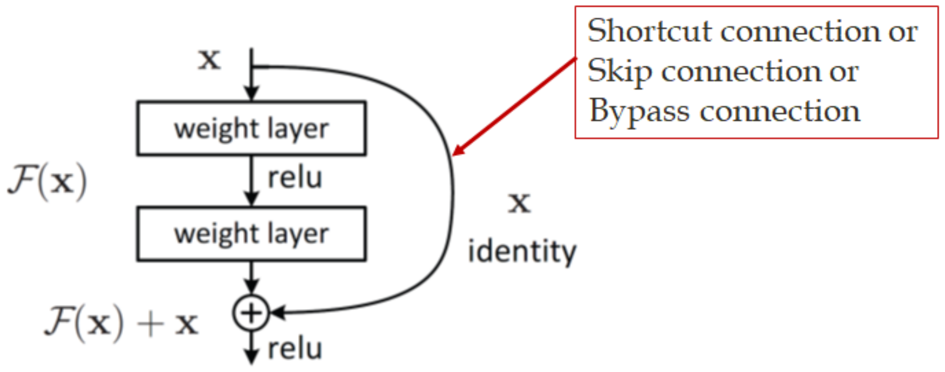

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
x
identity
Shortcut connection or Skip connection or Bypass connection

Figure 1. Residual learning: a building block.

In the original block, which is without the shortcut path in Figure 1, is supposed to learning the output function O(x), which can be decomposed as "F(x) + x" but with the shortcut path, the network needs to learn to produce a residual part, i.e., F(x). Learning F(x) is easier than O(x), because, for a degraded output, network can compare F(x) with or combine with the previous output x, optimally.

23. How many types of residual blocks are available in ResNet? Describe using examples.

Ans: There are two types of residual blocks: one is a regular residual block used when the global depth of the network is less than 50; otherwise, for depth ≥ 50, the residual block is combined with bottleneck block(s). See Figure 1 as an example.

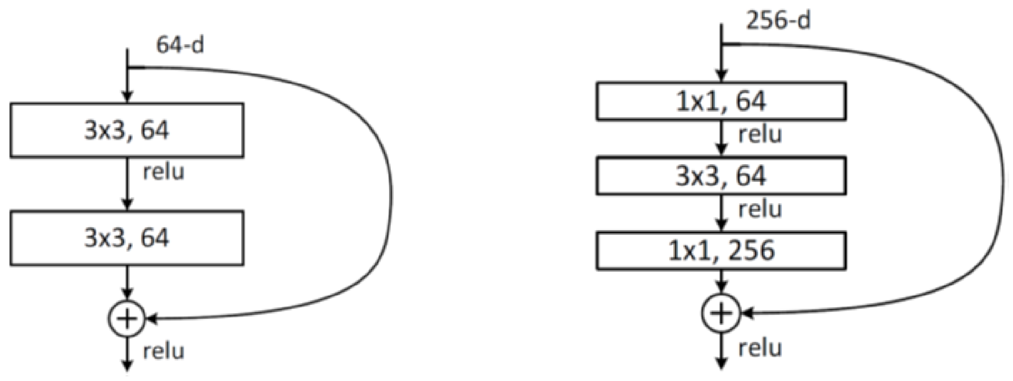


Figure: Left: a building block (on 56×56 feature maps) for ResNet-34.
Right: a "bottleneck" building block for ResNet-50/101/152.

24. Describe the "Adaptive Moments" or the "Adam" algorithm.

Ans:

- **Adaptive Moments, Adam** [1-2]: Adam can be best seen as a variant on the combination of RMSProp and momentum.

Algorithm 1.4: The Adam algorithm

Require: Learning rate α = **0.001 (suggested)**; initial parameter $\beta(0)$; small-constant, ϵ = 1e-8 (**suggested**); grad, g=0; decay rates: $\rho 1$ and $\rho 2$ in [0,1) (0.9 and 0.999 **suggested**); Initialize 1st and 2nd moment variables, $S = 0$, $r = 0$; iteration $t = 0$.

```
WHILE stopping criterion not met DO
        Sample a minibatch of $m$ examples $\{(x, y)_1, \ldots, (x, y)_m\}$, from $N$ training points
        Compute $\beta(t+1)$ as:
                $g = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial E}{\partial \beta(t)}$;
                $t = t + 1$
            1st moment update: $s = \rho_1 s + (1-\rho_1) g$
            2nd moment update: $r = \rho_2 r + (1-\rho_2)(g)^2$;        $(\,)^2$ applied element-wise.
      Correct bias in 1st moment: $\hat{s} = \frac{s}{1-\rho_1^t}$
      Correct bias in 2nd moment: $\hat{r} = \frac{r}{1-\rho_2^t}$
                $\Delta\beta(t) = -\alpha \frac{\hat{s}}{\sqrt{\hat{r}}+\epsilon}$;   Operations applied element-wise.
                $\beta(t+1) = \beta(t) + \Delta\beta(t)$
END WHILE
```

---- x ----