

# ML Specialist Ready for Optimization, CNNs

## Prompt:

You're an experienced machine learning specialist who knows a lot about optimization and convolutional neural networks. In the following conversation, you will be doing some optimization and convolutional neural networks-related problems. Please prepare.

**Prompt for preparation.**

## Response:

I am ready. As a machine learning specialist with a deep focus on optimization algorithms and Convolutional Neural Networks (CNNs), I am prepared to assist you.

Whether we are discussing gradient descent variants (SGD, Adam, RMSProp), analyzing loss landscapes, designing CNN architectures (ResNets, Inception, EfficientNet), or calculating receptive fields and backpropagation gradients, I can provide detailed, mathematically rigorous solutions.

Please provide the first problem or topic you would like to tackle.

## Prompt:

*hw4\_q1 - PDF*

Please do question 1 on this assignment.

## Response:

Here is the step-by-step solution for Question 1: Newton-Schulz Runtime, based on the provided homework file.

### 1. Newton-Schulz Runtime

**Problem Setup** We are given the Newton-Schulz update polynomial for a matrix  $W \in R^{d_{out} \times d_{in}}$ :

$$p(W) = \frac{1}{2} (3I_{d_{out}} - WW^T) W$$

We assume the runtime is dominated by matrix multiplication, where multiplying an  $n \times m$  matrix by an  $m \times p$  matrix takes  $c m n p$  time.

#### (a) Runtime of a single iteration (Naive Calculation)

To find the runtime, we break down the calculation of  $p(W)$  into sequential matrix multiplication steps following standard order of operations (parentheses first).

1. \*\*Compute  $WW^T$ \*\*:

- $W$  has dimensions  $d_{out} \times d_{in}$ .
- $W^T$  has dimensions  $d_{in} \times d_{out}$ .
- $WW^T$  has dimensions  $d_{out} \times d_{out}$ .

- $W$  has dimensions  $d_{out} \times d_{in}$ .
- $W^T$  has dimensions  $d_{in} \times d_{out}$ .
- The result  $A = WW^T$  has dimensions  $d_{out} \times d_{out}$ .
- **Cost:**  $c \cdot d_{out} \cdot d_{in} \cdot d_{out} = c \cdot d_{in} \cdot d_{out2}$ .

2. \*\*Compute  $(3I_{d_{out}} - A)W$ \*\*:

- Let  $B = (3I_{d_{out}} - A)$ . This is a matrix addition/scaling operation, which is  $O(d_{out2})$  and negligible compared to matrix multiplication as per the problem statement.  $B$  has dimensions  $d_{out} \times d_{out}$ .
- We now multiply  $B$  by  $W$  ( $d_{out} \times d_{in}$ ).
- **Cost:**  $c \cdot d_{out} \cdot d_{out} \cdot d_{in} = c \cdot d_{in} \cdot d_{out2}$ .

**Total Runtime for (a):** Adding the costs of the two matrix multiplications:

Fully correct! The square in the bottom is a pdf conversion error and was not present in the chat.

$$Runtime \approx c \cdot d_{in} \cdot d_{out2} + c \cdot d_{in} \cdot d_{out2} = 2cd_{in}d_{out2}$$

## (b) Optimized Runtime for $d_{out} \gg d_{in}$

We are asked to optimize the computation when the output dimension is much larger than the input dimension ( $d_{out} \gg d_{in}$ ).

**Strategy** We can use the associativity of matrix multiplication to rewrite the polynomial term. The original expression is:

$$\frac{1}{2} (3W - WW^T W)$$

Instead of computing  $(WW^T)W$ , we can group the terms as  $W(W^TW)$ . This utilizes the Gram matrix  $W^TW$ , which is much smaller ( $d_{in} \times d_{in}$ ) when  $d_{out} \gg d_{in}$ .

**New Calculation Steps:**

1. **Compute  $W^TW$  (The Gram Matrix):**
  - $W^T$  is  $d_{in} \times d_{out}$ .
  - $W$  is  $d_{out} \times d_{in}$ .
  - The result  $G = W^TW$  has dimensions  $d_{in} \times d_{in}$ .
  - **Cost:**  $c \cdot d_{in} \cdot d_{out} \cdot d_{in} = c \cdot d_{out} \cdot d_{in2}$ .
2. \*\*Compute  $W(G)$ \*\*:
  - We multiply  $W$  ( $d_{out} \times d_{in}$ ) by  $G$  ( $d_{in} \times d_{in}$ ).
  - **Cost:**  $c \cdot d_{out} \cdot d_{in} \cdot d_{in} = c \cdot d_{out} \cdot d_{in2}$ .
3. **Combine and Scale:**
  - Compute  $\frac{3}{2}W - \frac{1}{2}WG$ . This involves scalar multiplication and subtraction, which are element-wise operations with cost  $O(d_{out}d_{in})$ . This is negligible compared to the matrix multiplications ( $O(d_{out}d_{in2})$ ).

**Total Runtime for (b):** Adding the costs of the two new matrix multiplications:

$$Runtime \approx c \cdot d_{out} \cdot d_{in2} + c \cdot d_{out} \cdot d_{in2} = 2cd_{out}d_{in2}$$

**Comparison:** Since  $d_{out} \gg d_{in}$ , the optimized runtime  $2cd_{out}d_{in2}$  is significantly faster than the naive runtime  $2cd_{in}d_{out2}$ .

**Next Step:** Would you like me to proceed to **Question 2**, regarding MuP and fp8 initialization?

**Comparison:** Since  $d_{out} \gg d_{in}$ , the optimized runtime  $2cd_{out}d_{in2}$  is significantly faster than the naive runtime  $2cd_{in}d_{out2}$ .

**Next Step:** Would you like me to proceed to **Question 2**, regarding MuP and fp8 initialization?

## Prompt:

hw4\_q2 - PDF

Looks great! Yes, now let's proceed to Question 2, which is in this pdf.

## Response:

Here are the solutions to Question 2, "MuP at the Unit Scale," based on the provided homework.

### 2. MuP at the Unit Scale

#### (a) Initialization for FP8

Correct! I think the model looked at the link for this one.

**Why initialize from  $N(0, 1)$  instead of Xavier?** In standard Xavier initialization, weights are scaled by factors like  $\frac{1}{\sqrt{n}}$  to preserve variance. For large layers, this results in very small weight values. However, **FP8 (8-bit floating point)** has a very limited dynamic range and precision (only 255 possible values). Initializing with the small values typical of Xavier initialization would likely cause **underflow** (weights collapsing to zero) or result in severe quantization errors because FP8 cannot effectively represent values that small. Initializing from  $N(0, 1)$  keeps the parameters in the "unit scale" range ( $O(1)$ ), which is well-covered by the FP8 format, ensuring numerical stability during the initial training phase.

#### (b) Activation Scaling Constant

We introduce a scalar  $c$  such that  $y = cWx$  to recover the variance-preserving properties of Xavier initialization.

- Let  $W \in R^{d_{out} \times d_{in}}$  with entries  $W_{ij} \sim N(0, 1)$ .
- Let  $x \in R^{d_{in}}$  be the input.
- The variance of a single output unit  $y_k$  (before scaling) is:

$$Var(y_k) = \sum_{j=1}^{d_{in}} Var(W_{kj}x_j) \approx d_{in} \cdot Var(x) \quad \text{Fully correct!}$$

(assuming  $W$  and  $x$  are independent and  $W$  has unit variance).

- To preserve variance (i.e.,  $Var(y) = Var(x)$ ), we need to scale the output by a factor that counters the  $d_{in}$  growth.
- With  $y = cWx$ , the variance scales by  $c^2$ .

$$c^2 \cdot d_{in} = 1 \implies c = \frac{1}{\sqrt{d_{in}}}$$

**Answer:**  $c = \frac{1}{\sqrt{d_{in}}}$ .

#### (c) Maximum Spectral Norm for Updates

We want to control the update  $\Delta y = c\Delta Wx$  such that its RMS norm is no larger than 1, given  $x$  has an RMS

norm of 1.

## (c) Maximum Spectral Norm for Updates

We want to control the update  $\Delta y = c\Delta Wx$  such that its RMS norm is no larger than 1, given  $x$  has an RMS norm of 1.

### 1. Definitions:

- RMS norm of  $x \in R^{d_{in}}$  is 1  $\implies \|x\|_2 = \sqrt{d_{in}}$ .
- RMS norm of  $\Delta y \in R^{d_{out}}$  is  $\leq 1 \implies \|\Delta y\|_2 \leq \sqrt{d_{out}}$ .
- $c = \frac{1}{\sqrt{d_{in}}}$  (from part b).

Fully correct; does the bound how we usually do it.

### 2. Inequality:

$$\|\Delta y\|_2 = \|c\Delta Wx\|_2 \leq c \|\Delta W\|_{op} \|x\|_2$$

### 3. Substitution:

$$\sqrt{d_{out}} \geq \frac{1}{\sqrt{d_{in}}} \|\Delta W\|_{op} \sqrt{d_{in}}$$

$$\sqrt{d_{out}} \geq \|\Delta W\|_{op}$$

**Answer:** The maximum spectral norm of  $\Delta W$  should be  $\sqrt{d_{out}}$ .

## (d) SignSGD Learning Rate

We assume a minibatch size of 1 and that  $sign(\nabla_W L)$  is a rank-1 matrix. We need to find the learning rate  $\alpha$  for the update  $\Delta W = \alpha \cdot sign(\nabla_W L)$ .

### 1. Spectral Norm of Rank-1 Sign Matrix:

- Since the gradient is an outer product  $\nabla W = g \cdot x^T$ , its element-wise sign  $S = sign(\nabla W)$  is also rank-1 (as stated in the prompt).
- $S = uv^T$  where  $u \in \{\pm 1\}^{d_{out}}$  and  $v \in \{\pm 1\}^{d_{in}}$ .
- The spectral norm of a rank-1 matrix is the product of the norms of its vectors:

$$\|S\|_{op} = \|u\|_2 \|v\|_2 = \sqrt{d_{out}} \sqrt{d_{in}}$$

### 2. Applying Constraint:

- $\|\Delta W\|_{op} = \alpha \|S\|_{op} = \alpha \sqrt{d_{out} d_{in}}$ .
- From part (c), we require  $\|\Delta W\|_{op} \leq \sqrt{d_{out}}$ .
- $\alpha \sqrt{d_{out} d_{in}} \leq \sqrt{d_{out}} \implies \alpha \sqrt{d_{in}} \leq 1$ .

Latex had no error on the original website. Derivation is correct.

**Answer:**  $\alpha \leq \frac{1}{\sqrt{d_{in}}}$ .

## (e) Muon/Orthogonalized Update Learning Rate

The update is defined as  $\Delta W = \alpha UV^T$ , where  $U, V$  come from the SVD of the gradient.

### 1. Spectral Norm of Update:

- The matrix  $UV^T$  is an isometry (orthogonal) on the subspace defined by the singular vectors. Since  $U$  and  $V$  have orthonormal columns/rows, all non-zero singular values of  $UV^T$  are exactly 1.
- Therefore,  $\|UV^T\|_{op} = 1$ .
- $\|\Delta W\|_{op} = \alpha \cdot 1 = \alpha$ .

### 2. Applying Constraint:

- Therefore,  $\| UV^T \|_{op} = 1$  .
- $\| \Delta W \|_{op} = \alpha \cdot 1 = \alpha$  .

## 2. Applying Constraint:

- From part (c), we require  $\| \Delta W \|_{op} \leq \sqrt{d_{out}}$  .

**Answer:**  $\alpha \leq \sqrt{d_{out}}$  .

This is different from the solution, but as far as I can see, it is correct. The constant  $c$  should be irrelevant here (alternatively, we already accounted for it when deriving the bound for delta W's norm), so I think Gemini is right here.

## (f) Intermediate Backprop Scale

**Question:** Is there a setting where the scale of these intermediate backpropagated gradients can also be ignored?

Fully correct!

**Answer: Yes.** Optimizers like **SignGD**, **Adam**, and **Muon** depend primarily on the sign or the normalized direction of the gradient, not its global magnitude (scale). Because the backpropagation equation is linear with respect to the gradient ( $\nabla_{x_n} L = c_n W_{nT} \nabla_{x_{n+1}} L$ ), a global scaling factor on the gradient at layer  $n + 1$  simply propagates as a scalar multiplier to layer  $n$ . Since the update rules for these specific optimizers are scale-invariant (normalizing by second moments in Adam or taking the sign in SignSGD), the specific magnitude of the intermediate gradient does not affect the final weight update direction.

## (g) Backprop Stability Constant

We analyze the stability of backpropagation assuming  $W$  is rank-1, parameters are unit scaled, and using the scalar  $c = 1/\sqrt{d_{in}}$  .

### 1. Spectral Norm of Backprop Operator:

- The backward pass operation is  $g_{in} = cW^T g_{out}$  .
- We assume  $W$  is rank-1 and parameters are unit scaled ( $W_{ij} \sim N(0, 1)$ ), so  $\| W \|_F \approx \sqrt{d_{out} d_{in}}$  .
- For a rank-1 matrix, the spectral norm equals the Frobenius norm:  $\| W \|_{op} = \sqrt{d_{out} d_{in}}$  .
- The effective scaling factor of the operator  $cW^T$  is:

$$Scale = c \cdot \| W \|_{op} = \frac{1}{\sqrt{d_{in}}} \cdot \sqrt{d_{out} d_{in}} = \sqrt{d_{out}}$$

### 2. Explosion:

Fully correct as well!

- Since  $\sqrt{d_{out}} > 1$  (usually), the gradients will **explode** as they propagate backwards.

### 3. Correction:

- To stabilize this (keep the scale factor at 1), we must multiply the intermediate gradients by the inverse of the growth factor.
- Constant =  $\frac{1}{\sqrt{d_{out}}}$  .

**Answer:** The intermediate backpropagated gradients should be multiplied by  $\frac{1}{\sqrt{d_{out}}}$  .

Would you like me to tackle Question 3 regarding Convolution and Finite Impulse Response Filters next?

## Prompt:

hw4\_q3 - PDF

This makes sense! Now, let's move on to question 3, which is in this pdf:

This makes sense! Now, let's move on to question 3, which is in this pdf:

## Response:

Here are the solutions to **Question 3: Understanding Convolution as Finite Impulse Response Filter** from the provided assignment.

**Note on Definition:** The problem statement explicitly instructs to "follow the definition in (3)" , which is the mathematical definition of convolution involving "flipping and dragging".

- **Mathematical Convolution ( \* ):** Flips the kernel  $h$  before computing the dot product.
- **Correlation:** Does not flip the kernel (standard in deep learning frameworks).

The solutions below utilize the **mathematical convolution** definition as requested. If the "Neural Network standard" (Correlation) were used, the results in parts (c) and (d) would simply have their signs flipped (multiplied by -1).

### 3. Understanding Convolution as Finite Impulse Response Filter

#### (a) 1D Convolution Calculation and Plot

Given:

- $x[n]$  is a rectangular signal of length  $L = 7$  (implied by Fig 1).  $x[n] = 1$  for  $0 \leq n \leq 6$  , and 0 otherwise.
- $h[n] = (1/2)^n u[n]$  (Exponential decay).
- **Convolution Sum:**  $y[n] = \sum_{k=-\infty}^{\infty} x[k] h[n-k]$  .

**Analysis:** Since  $x[k]$  is non-zero only for  $0 \leq k \leq 6$  , the sum simplifies to  $k \in [0, 6]$  . The term  $h[n-k]$  is non-zero only when  $n-k \geq 0 \implies k \leq n$  . Thus, the summation range is  $k \in [0, \min(n, 6)]$  .

We distinguish three regions for  $n$  :

Slightly less general than the actual solution, but clear that the model knows what it's doing.

Anecdotally, I made the same assumption when I did this homework.

1. \*\*  $n < 0$  :\*\* There is no overlap.

$$y[n] = 0$$

2. \*\*  $0 \leq n \leq 6$  (Rising Edge):\*\* The overlap is from  $k = 0$  to  $k = n$  .

$$y[n] = \sum_{k=0}^n 1 \cdot (1/2)^{n-k} = (1/2)^n \sum_{k=0}^n 2^k$$

Using the geometric series sum  $\sum_{i=0}^N r^i = \frac{r^{N+1}-1}{r-1}$  :

$$y[n] = (1/2)^n \left( \frac{2^{n+1}-1}{2-1} \right) = (1/2)^n (2^{n+1}-1) = 2 - (1/2)^n$$

3. \*\*  $n > 6$  (Decaying Tail):\*\* The overlap is complete over the support of  $x$  , from  $k = 0$  to  $k = 6$  .

$$y[n] = \sum_{k=0}^6 (1/2)^{n-k} = (1/2)^n \sum_{k=0}^6 2^k$$

$$y[n] = (1/2)^n (2^7 - 1) = 127 \cdot (1/2)^n$$

**Plot Values (Range -6 to 12):**

- $n \in [-6, -1] : 0$
- $n = 0 : 1$

### Plot Values (Range -6 to 12):

- $n \in [-6, -1] : 0$
- $n = 0 : 1$
- $n = 1 : 1.5$
- $n = 2 : 1.75$
- $n = 3 : 1.875$
- $n = 4 : 1.9375$
- $n = 5 : 1.96875$
- $n = 6 : 1.984375$  (Peak)
- $n = 7 : 127/128 \approx 0.992$
- $n = 8 : 127/256 \approx 0.496$
- ...
- $n = 12 : 127 \cdot (1/2)^{12} \approx 0.031$

**Plot:** The signal is zero until  $n = 0$ , rises asymptotically towards 2 until  $n = 6$ , and then decays exponentially towards 0.

### (b) Shift Property

Fully correct!

**Given:**  $x_2[n] = x[n - 5]$ . **Compute:**  $y_2[n] = h[n] * x_2[n]$ .

**Property:** This demonstrates the **Time Invariance** (or Shift Invariance) property of convolution. If the input is shifted by  $N$ , the output is shifted by  $N$ .

$$y_2[n] = y[n - 5]$$

The plot would look identical to part (a), just shifted 5 units to the right.

### (c) 2D Convolution (No Padding, Stride 1)

**Given:**

- Input  $x$ : 5×5 matrix with values 1 to 25.
- Filter  $h$ :  $\begin{bmatrix} 1 & 0 & 1 & 2 & 0 \\ 2 & 0 & 1 & 0 & 1 \end{bmatrix}$

**Method (Convolution):**

1. \*\*Flip the kernel  $h$ \*\* by 180 degrees (both vertically and horizontally) to get  $h_{flip}$ .  $h_{\{flip\}} = \begin{bmatrix} 1 & 2 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 2 \end{bmatrix}$
2. Slide  $h_{flip}$  over  $x$  and compute the sum of products. The output size is  $(5 - 3) + 1 = 3 \times 3$ .

**Calculation:** Let's analyze the operation on an arbitrary 3×3 patch of  $x$ . Note that in  $x$ , the values increase by 5 for each row down ( $x_{i+1,j} = x_{i,j} + 5$ ) and by 10 for two rows down ( $x_{i+2,j} = x_{i,j} + 10$ ).

- **Top Row of Kernel (1, 2, 1):** Multiplies the top row of the image patch ( $Row_i$ ). Sum contribution:  $1 \cdot P_{0,0} + 2 \cdot P_{0,1} + 1 \cdot P_{0,2}$ .
- **Bottom Row of Kernel (-1, -2, -1):** Multiplies the bottom row of the image patch ( $Row_{i+2}$ ). Sum contribution:  $-1 \cdot P_{2,0} - 2 \cdot P_{2,1} - 1 \cdot P_{2,2}$ . This is equivalent to  $-(1 \cdot P_{2,0} + 2 \cdot P_{2,1} + 1 \cdot P_{2,2})$ .

Since  $P_{2,k} = P_{0,k} + 10$ :  $BottomContribution = -(TopContribution + 10 \times (1 + 2 + 1)) = -(TopContribution + 40)$ .

**Total Sum** = Top Contribution + Bottom Contribution =  $Top - Top - 40 = -40$ .

Since the local linear structure of the matrix is constant everywhere, a non-overlapping patch yields the same result.

$\dots - (TopContribution + 40) \dots$

**Total Sum** = Top Contribution + Bottom Contribution =  $Top - Top - 40 = -40$ .

Since the local linear structure of the matrix is constant everywhere, every valid patch yields the same result.

**Result  $y$ :**

$y=\leftarrow 40-40-40-40-40-40-40-40-40\right)$

This is fully correct, and did not have display bugs on the website.

## (d) 2D Convolution with Stride and Padding

We apply the same flipped kernel  $h_{flip}$  to the padded input. **Input Pad=1:** A border of zeros is added around  $x$ . **Calculations:**

- **Internal Pixels:** Any output derived from a patch strictly inside the original image remains **-40**.
- **Edges/Corners:** The zero padding changes the sums.
  - *Top Edge:* Kernel top row hits zeros. Only bottom row (negative weights) hits image.
  - *Bottom Edge:* Kernel bottom row hits zeros. Only top row (positive weights) hits image.
  - *Left/Right:* Partial sums.

**(i) Stride = 1, Pad = 1:** Output size:  $5 \times 5$ .

$y=\leftarrow 19-30-30-3049-28-40-40-4068-32-40-40-4072-36-40-40-4076-29-30-3059\right)$

*Example Spot Check ( $y_{0,0}$ ):* Top-left patch is  $\leftarrow 000016027\right)$ .  $h_{\{flip\}}=\leftarrow 10-120-210-1\right)$ . Sum =  $(0 \cdot Top) - (0(1) + 6(2) + 7(1)) = -19$ .

**(ii) Stride = 2, Pad = 1:** We subsample the result from (i) at indices  $(0,0), (0,2), (0,4) \dots$ . Output size:  $3 \times 3$ .

$y=\leftarrow 19-3049-32-4072-29-3059\right)$

This is also fully correct, and did not have display bugs on the website.

**Prompt:**

*hw4\_q4 - PDF*

Looks good! Now let's move on to problem 4, in the following pdf.

## Response:

Here are the solutions to **Question 4: Feature Dimensions of Convolutional Neural Network**.

### 4. Feature Dimensions of Convolutional Neural Network

#### (a) Convolutional Layer Parameters and Output

Given a convolutional layer with input shape  $W \times H \times C$ , filter size  $K$ , padding  $P$ , stride  $S$ , and number of filters  $F$ .

##### 1. Number of Weights and Biases:

- **Weights:** Each filter has dimensions  $K \times K \times C$ . There are  $F$  filters.

$$Weights = F \times K \times K \times C$$

- **Biases:** There is 1 bias per filter.

$$Weights = F \times K \times K \times C$$

- **Biases:** There is 1 bias per filter.

$$Biases = F$$

- **Total Parameters:**  $F(K^2C + 1)$ .

2. **Output Feature Shape:** The spatial dimensions are calculated using the standard arithmetic for convolutions. The depth is determined by the number of filters.

- **Width ( $W_{out}$ ):**  $\lfloor \frac{W-K+2P}{S} \rfloor + 1$
- **Height ( $H_{out}$ ):**  $\lfloor \frac{H-K+2P}{S} \rfloor + 1$
- **Channels ( $C_{out}$ ):**  $F$
- **Shape:**  $(\lfloor \frac{W-K+2P}{S} \rfloor + 1) \times (\lfloor \frac{H-K+2P}{S} \rfloor + 1) \times F$ .

Fully correct!

## (b) Max Pooling Layer Output

Given a max pooling layer with stride  $S$  and filter size  $K$ . Pooling layers typically operate independently on each channel, preserving the depth.

- **Output Feature Shape:**

- **Width:**  $\lfloor \frac{W-K}{S} \rfloor + 1$  (Assuming padding  $P = 0$  unless specified otherwise for pooling).
- **Height:**  $\lfloor \frac{H-K}{S} \rfloor + 1$
- **Channels:**  $C$  (Same as input).
- **Shape:**  $(\lfloor \frac{W-K}{S} \rfloor + 1) \times (\lfloor \frac{H-K}{S} \rfloor + 1) \times C$ .

Fully correct!

## (c) Receptive Field Size

Given  $L$  successive convolutional layers with filter size  $K$  and stride 1.

- **Layer 1:** The receptive field (RF) is  $K$ .
- **Layer 2:** The filter "sees"  $K$  pixels of Layer 1. Since Layer 1 has stride 1, this covers an additional  $K - 1$  pixels of the input. Total  $RF = K + (K - 1)$ .
- **Layer L:** Each additional layer adds  $(K - 1)$  to the receptive field size.
- **Formula:**  $RF = 1 + L(K - 1)$ .

One of the two reasonable approaches. Didn't turn into big O notation but otherwise good.

## (d) Downsampling and Receptive Field

**Q: How much does the receptive field increase with a 2×2 pooling (stride 2)?**

- While the pooling layer itself has a small receptive field of  $2 \times 2$ , its primary effect on the network is that it **doubles** the effective stride for all subsequent layers relative to the input. This means any convolutional layer placed after this pooling layer will cover a spatial area on the original input image that is **2 times larger** (linearly) than it would have without pooling.

**Q: Advantage of decreasing output feature resolution (computation perspective):**

- **Computational Efficiency:** Convolutional operations generally scale linearly with the number of input pixels ( $H \times W$ ). Reducing the resolution by a factor of 2 (in both dimensions) reduces the number of operations for subsequent layers by a factor of 4.
- **Memory Efficiency:** Smaller feature maps require less memory to store activations during the forward pass and gradients during the backward pass.

pixels ( $H \times W$ ). Reducing the resolution by a factor of 2 (in both dimensions) reduces the number of operations for subsequent layers by a factor of 4.

- **Memory Efficiency:** Smaller feature maps require less memory to store activations during the forward pass and gradients during the backward pass.

It looks like Gemini didn't include the fact that we usually double the amount of channels here above? Otherwise correct.

## (e) Architecture Table

We fill in the missing entries for the network defined in the prompt.

- **CONV3-10 (1st):** Input  $28 \times 28 \times 1$ . Filters  $10 \times 3 \times 3 \times 1$ . Params:  $(9 \times 1 + 1) \times 10 = 100$ .
- **CONV3-10 (2nd):** Input  $14 \times 14 \times 10$ . Filters  $10 \times 3 \times 3 \times 10$ . Params:  $(9 \times 10 + 1) \times 10 = 910$ .
- **FLATTEN:** Input  $7 \times 7 \times 10$ . Output size 490. Params: 0.
- **FC-3:** Input 490. Output 3. Weights  $490 \times 3$ , Biases 3. Params:  $1470 + 3 = 1473$ .

Layer	Number of Parameters	Dimension of Activations
Input	0	$28 \times 28 \times 1$
CONV3-10	100	$28 \times 28 \times 10$
POOL2	0	$14 \times 14 \times 10$
CONV3-10	$3 \times 3 \times 10 \times 10 + 10$ (910)	** $14 \times 14 \times 10$ **
POOL2	0	** $7 \times 7 \times 10$ **
FLATTEN	0	490
FC-3	1473	3

Fully correct (numbers on the left are just multiplied out).

## (f) Invariances and Equivariances

We are given outputs for a horizontal edge image  $x_1$  ( $g_1 = [0.8, 0, 0]$ ) and a vertical edge image  $x_2$  ( $g_2 = [0, 0.8, 0]$ ). The network uses Global Average Pooling (GAP) and circular padding.

1. **For  $x_i$  (The zero matrix):**

- **Analysis:** The input contains no signal (all zeros).
- **Result:** Assuming the network bias is zero or negligible for null inputs (standard assumption for "what is the output" questions unless bias is given), the activations propagate as zeros.
- $g_{\{i\}} = \left[ \begin{smallmatrix} 0 & 0 & 0 \end{smallmatrix} \right]$ .

2. **For  $x_0$  (The diagonal matrix):**

It looks like Gemini didn't see  $x_3$  and  $x_4$  in the actual question?

- **Analysis:**

- **Translation:**  $x_0$  is *not* a translation of  $x_1$  or  $x_2$  (even on a torus, a diagonal line cannot be shifted to become a horizontal line). So we cannot simply assign it to class 1 or 2.
- **Symmetry/Equivariance:** If we transpose  $x_1$  (swap x,y), we get  $x_2$ . The output swaps the first two indices ( $[0.8, 0, 0] \rightarrow [0, 0.8, 0]$ ).
- $x_0$  is a diagonal line. It is symmetric under transpose ( $x_{0T} = x_0$ ). Therefore, its output must also be symmetric under the swap of the first two indices. i.e.,  $g(x_0)_1 = g(x_0)_2$ .
-

- first two indices ( $[0.8, 0, 0] \rightarrow [0, 0.8, 0]$ ).
- $x_0$  is a diagonal line. It is symmetric under transpose ( $x_{0T} = x_0$ ). Therefore, its output must also be symmetric under the swap of the first two indices. i.e.,  $g(x_0)_1 = g(x_0)_2$ .
- Feature Matching:** The network appears to distinguish horizontal vs. vertical edges. A pure diagonal line (composed of single pixels connected at corners) does not contain  $2 \times 2$  or  $3 \times 3$  horizontal or vertical edge segments. Standard filters trained/designed for  $x_1$  and  $x_2$  would likely have near-zero activation for a diagonal single-pixel line.
- Result:** Since it triggers neither the horizontal nor vertical features strongly, and based on the linearity/feature separation usually implied in such problems:
- $g_{\{0\}}=\left|000\right\rangle$

## Prompt:

**Image of  $x_3$  and  $x_4$**

For  $f$ , the question specifically asked for the output on  $x_3$  and  $x_4$ , which are as I have attached to this chat. Please answer based on those.

## Response:

Based on the images provided for inputs  $x_3$  and  $x_4$ , here is the solution using the properties of the Convolutional Neural Network (CNN) described in the problem.

### Key Concept: Translation Invariance

The critical architectural details provided in the prompt are:

- Convolutional Layers with Circular Padding:** Circular padding means the image is treated as if it wraps around (a torus). Shifting the input image circularly results in the feature maps being shifted by the exact same amount, without losing information at the edges.
- Global Average Pooling (GAP):** This layer computes the average value of each feature map. Since the average of a set of numbers is the same regardless of their position (i.e.,  $\text{average}([a, b, c]) = \text{average}([c, a, b])$ ), **shifting the feature map does not change the output of the GAP layer**.

Therefore, this network is **translation invariant**:  $f(\text{shifted } x) = f(x)$ .

### Analysis of $x_3$

**fully correct!**

- Observation:**  $x_3$  shows a single vertical line of 1s in the right-most column (or near the right edge).
- Relation to Known Data:** This is a horizontally shifted version of  $x_2$  (the "vertical edge" image), which had a vertical line in a different column.
- Result:** Due to translation invariance, the network produces the same output for  $x_3$  as it does for  $x_2$ .

- $\circ$  We know  $g_2 = f(x_2) = [0, 0.8, 0]^T$ .

$g_{\{3\}}=\left|00.80\right\rangle$

### Analysis of $x_4$

- Observation:**  $x_4$  shows a single horizontal line of 1s near the bottom row.

## Analysis of $x_4$

- **Observation:**  $x_4$  shows a single horizontal line of 1s near the bottom row.
- **Relation to Known Data:** This is a vertically shifted version of  $x_1$  (the "horizontal edge" image), which had a horizontal line in a different row.
- **Result:** Due to translation invariance, the network produces the same output for  $x_4$  as it does for  $x_1$ .
  - We know  $g_1 = f(x_1) = [0.8, 0, 0]^T$ . **fully correct!**

$g_{\{4\}}=\left[0.800\right]$

## Prompt:

*hw4\_q7 - PDF*

Looks great. Now let's move on to question 7, which is in this pdf.

## Response:

Here are the solutions to **Question 7: Weights and Gradients in a CNN**, based on the provided file

*hw4\_q7.pdf*.

### 7. Weights and Gradients in a CNN

#### (a) Gradient Derivation for Convolutional Weights

We are given the convolution operation (without padding, stride 1, and ignoring bias) as:

$$y_{i,j} = \sum_{h=1}^k \sum_{l=1}^k x_{i+h-1, j+l-1} w_{h,l}$$

**1. Derivative of Loss with respect to Weights ( $dw$ )** To find the gradient for a specific weight  $w_{h,l}$ , we must sum the gradients from all output pixels  $y_{i,j}$  that  $w_{h,l}$  contributed to. Using the Chain Rule:

$$dw_{h,l} = \frac{\partial L}{\partial w_{h,l}} = \sum_{i=1}^m \sum_{j=1}^m \frac{\partial L}{\partial y_{i,j}} \cdot \frac{\partial y_{i,j}}{\partial w_{h,l}}$$

From the definition of  $y_{i,j}$  above, the partial derivative  $\frac{\partial y_{i,j}}{\partial w_{h,l}}$  is simply the input pixel multiplied by that weight:

$$\frac{\partial y_{i,j}}{\partial w_{h,l}} = x_{i+h-1, j+l-1}$$

Substituting this back into the summation:

**This is fully correct, except it didn't write it as a cross-correlation.**

$$dw_{h,l} = \sum_{i=1}^m \sum_{j=1}^m dy_{i,j} \cdot x_{i+h-1, j+l-1}$$

**2. Weight Update Step (SGD)** For a standard SGD step with learning rate  $\eta$  and a batch size of 1:

$$w_{new} = w_{old} - \eta \cdot dw$$

$$w_{h,l(new)} = w_{h,l(old)} - \eta \left( \sum_{i=1}^m \sum_{j=1}^m dy_{i,j} \cdot x_{i+h-1, j+l-1} \right)$$

$$w_{h,l(new)} = w_{h,l(old)} - \eta \left( \sum_{i=1}^m \sum_{j=1}^n dy_{i,j} \cdot x_{i+h-1,j+l-1} \right)$$


---

## (b) Statistical Analysis of Gradient Norms

We assume  $x_{i,j}$  and  $dy_{i,j}$  are independent random variables with means of 0 and variances  $\sigma_{x2}$  and  $\sigma_{g2}$  respectively.

\*\*1. Mean of  $dw_{h,l}$  \*\*

$$E [dw_{h,l}] = E \left[ \sum_{i,j} dy_{i,j} \cdot x_{i+h-1,j+l-1} \right] = \sum_{i,j} E [dy_{i,j} \cdot x...]$$

Since  $dy$  and  $x$  are independent:

$$E [dy \cdot x] = E [dy] \cdot E [x] = 0 \cdot 0 = 0$$

Thus, \*\*  $E [dw_{h,l}] = 0$  \*\*.

\*\*2. Variance of  $dw_{h,l}$  \*\* Since the random variables are independent, the variance of the sum is the sum of the variances:

$$Var (dw_{h,l}) = \sum_{i=1}^m \sum_{j=1}^m Var (dy_{i,j} \cdot x_{i+h-1,j+l-1})$$

For the product of two independent zero-mean variables  $A$  and  $B$ ,  $Var (AB) = E [(AB)^2] - (E [AB])^2 = E [A^2] E [B^2] - 0 = Var (A) Var (B)$ .

$$Var (term) = \sigma_{g2} \cdot \sigma_{x2}$$

Since there are  $m \times m$  terms in the summation:

Fully correct! The squared for the sigmas are actually displayed as squares on the original website.

$$Var (dw_{h,l}) = m^2 \cdot \sigma_{g2} \cdot \sigma_{x2}$$

**3. Asymptotic Growth Rate of Standard Deviation** The standard deviation is the square root of the variance:

$$Std (dw_{h,l}) = \sqrt{m^2 \sigma_{g2} \sigma_{x2}} = m \cdot \sigma_g \sigma_x$$

We know that for an input size  $n$  and kernel size  $k$ , the output size is  $m = n - k + 1$  (assuming stride 1, no pad). As  $n \rightarrow \infty$  (assuming  $k$  is fixed),  $m \approx n$ . Therefore, the standard deviation grows **linearly with respect to  $n$**  ( $O(n)$ ).

---

## (c) Pooling Layer Gradients

We analyze gradients for  $2 \times 2$  pooling layers. Let the pooling window size be  $K = 2$ .

**1. Max Pooling** For max pooling, the forward pass selects the maximum value in the window:  $y = \max(x_1, x_2, x_3, x_4)$ . During backpropagation, the gradient passes *only* through the index that held the maximum value.

This part is actually the fully correct solution.

$dx_{i,j} = \begin{cases} dy_{pooled} & \text{if } x_{i,j} = \max(\text{window}) \\ 0 & \text{otherwise} \end{cases}$

If an input pixel  $x_{i,j}$  was not the maximum in its local neighborhood, its gradient is 0. If it was the maximum, it receives the full gradient from the output.

**2. Average Pooling** For average pooling, the forward pass computes the mean:  $y = \frac{1}{K^2} \sum x_i$ .

$$y = \frac{1}{4} (x_1 + x_2 + x_3 + x_4)$$

This part is fully correct as well.

During backpropagation, the gradient is distributed equally to all inputs in the window, weighted by the constant factor  $\frac{1}{K^2}$ :

During backpropagation, the gradient is distributed equally to all inputs in the window, weighted by the constant factor  $\frac{1}{K^2}$  :

$$dx_{i,j} = \frac{1}{4}dy_{pooled}$$

## (d) Advantages of Max Pooling vs. Average Pooling

### Max Pooling:

- **Feature Selectivity:** It acts as a feature detector. By taking the max, it preserves the strongest signal (e.g., the sharpest edge or brightest texture) in a region and discards weaker signals.
- **Invariance:** It provides a degree of local translation invariance. If a feature shifts slightly within the  $2 \times 2$  window, the output remains the same max value.
- **Sparsity:** It produces sparse gradients (as seen in part c), which can help in learning distinct features.

**Good explanations! The Invariance and Information Preservation parts match the solution.**

### Average Pooling:

- **Information Preservation:** It uses information from all pixels in the patch, rather than discarding non-maximal ones. This is useful for "smoothing" noise or aggregating background information.
- **Global Context:** In modern architectures (like ResNet or Inception), Global Average Pooling (GAP) is often used at the end of the network to summarize the presence of features across the entire image spatial area, replacing fully connected layers and reducing parameter count significantly.

**I was curious whether or not Gemini could write part a. as a cross-correlation (and notice that it's a cross-correlation as well).**

### Prompt:

Looks great. Just one small thing: for part a., can you make the expression for the gradient with respect to w a bit smaller? Is there a cleaner way you can write it?

## Response:

Yes, there is a much cleaner way to represent that summation.

The expression you derived:

$$dw_{h,l} = \sum_{i=1}^m \sum_{j=1}^m x_{i+h-1, j+l-1} \cdot dy_{i,j}$$

is exactly the definition of the **valid cross-correlation** between the input  $X$  and the gradient  $dY$ .

In the context of deep learning (and consistent with the notation in the homework where  $*$  denotes the operation used in the layer), you can simply write:

$$dw = X * dy$$

**Correct cross-correlation!**

### Why this works:

- **Dimensions:** You are convolving the large input image  $X$  ( $n \times n$ ) with the smaller gradient map  $dY$  ( $m \times m$ ).
- **Result:** The result of this "valid" convolution is a matrix of size  $(n - m + 1) \times (n - m + 1)$ , which simplifies to  $k \times k$  —exactly the shape of your weight matrix  $dw$ .

This is the standard, compact way to express the weight gradient in CNNs.

This is the standard, compact way to express the weight gradient in CNNs.