CS 231N Midterm Review

Midterm Logistics

- Multiple Choice
- True/False
- Short Answer Questions
- More emphasis on topics covered earlier in the course than those discussed more recently

Focus is more on high-level understanding of concepts

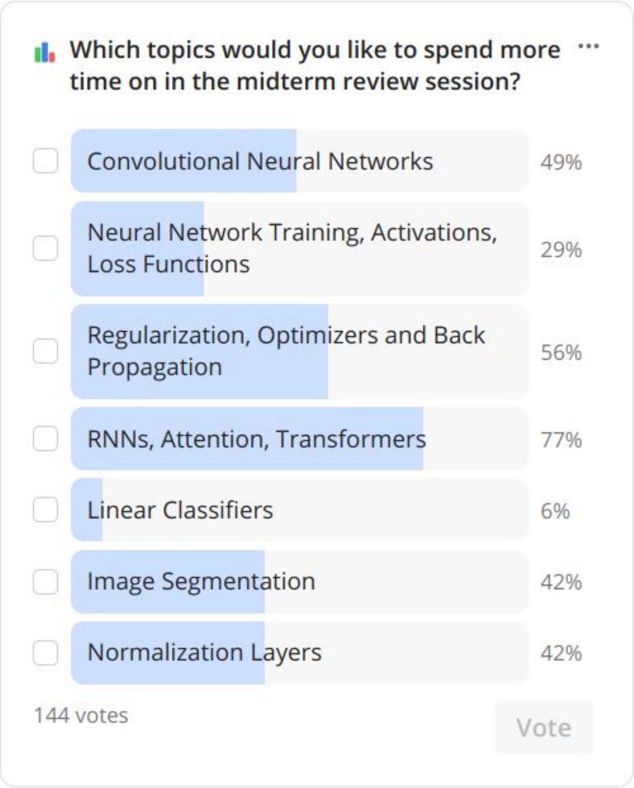
How many layers in a ResNet?

What problem does the ResNet solve and how?

More Logistics…

- The midterm exam will take place at **12:00 - 1:20pm PT on Tuesday, May 16 in person** at NVIDIA Auditorium, 420-040, and Hewlett 200.
- If your last name begins with a letter between **A** and **G** (inclusive), you will take the exam at NVIDIA Auditorium.
- If your last name begins with a letter between **H** and **M** (inclusive), you will take the exam at 420-040.
- If your last name begins with a letter between **N** and **Z** (inclusive), you will take the exam at Hewlett 200.
- Closed-book, no internet. One double-sided cheat sheet (written or typed)
- The exam may cover material from Assignments 1 and 2 and all lectures up to and including Lecture 12 (Visualizing and Understanding).
- You will have 80 minutes to complete the exam. The exam will start immediately at 12:00pm. **If you arrive late, you will not be given additional time.**

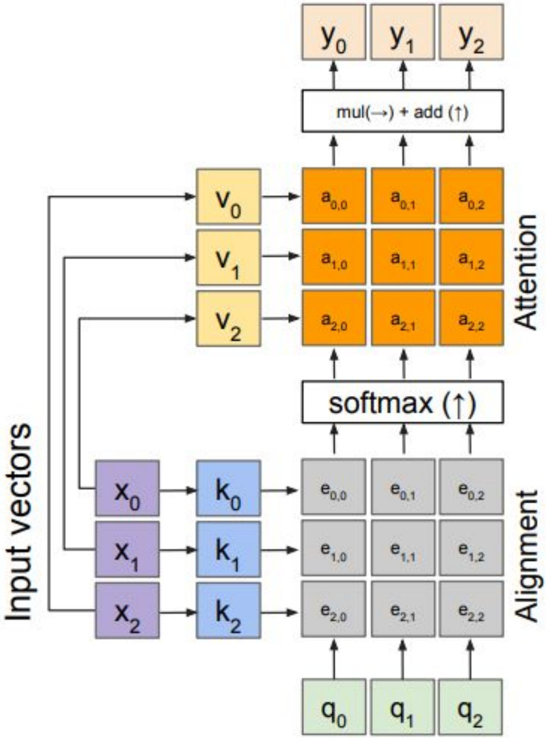
Midterm Review



Plan

1. Transformers & Attention
2. RNNs
3. Back Propagation
4. Optimizers
5. CNNs
6. Normalization Layers
7. Regularization Techniques

General Attention

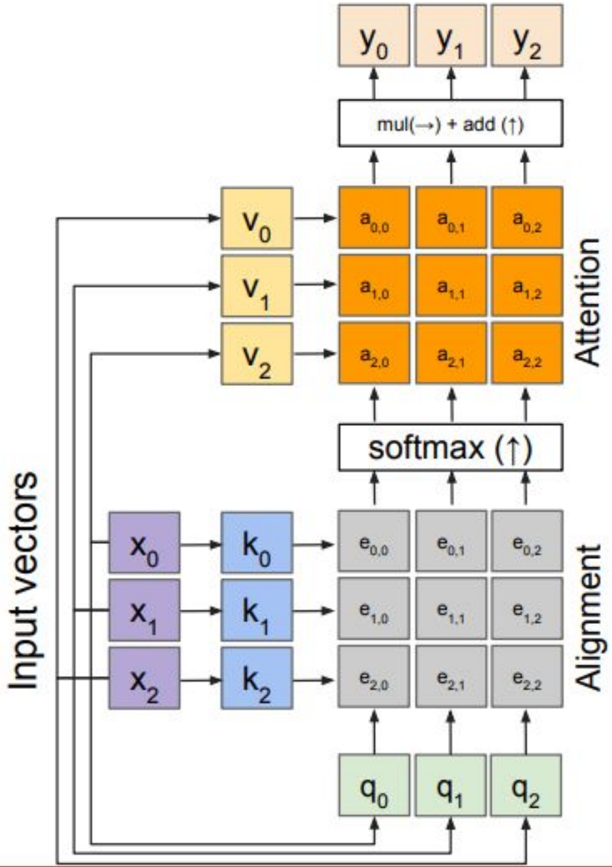


Outputs:
context vectors: $\mathbf{y}$ (shape: D_v)

Operations:
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: $\mathbf{x}$ (shape: N x D)
Queries: $\mathbf{q}$ (shape: M x D_k)

Self-Attention

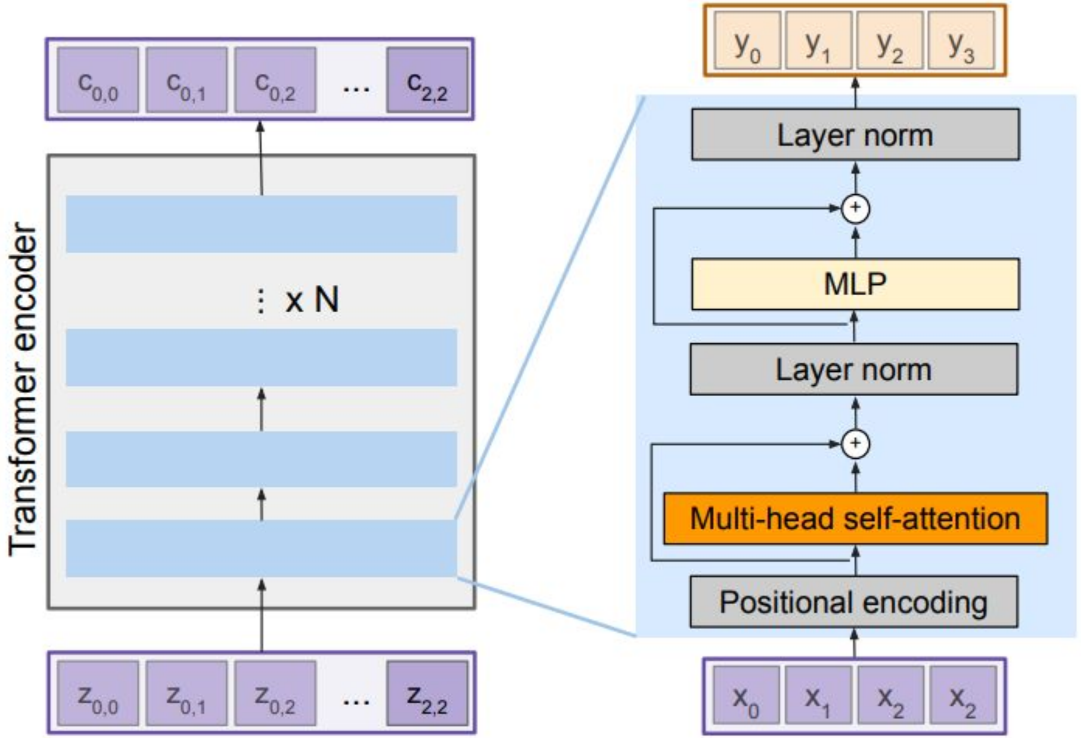


Outputs:
context vectors: **y** (shape: D_v)

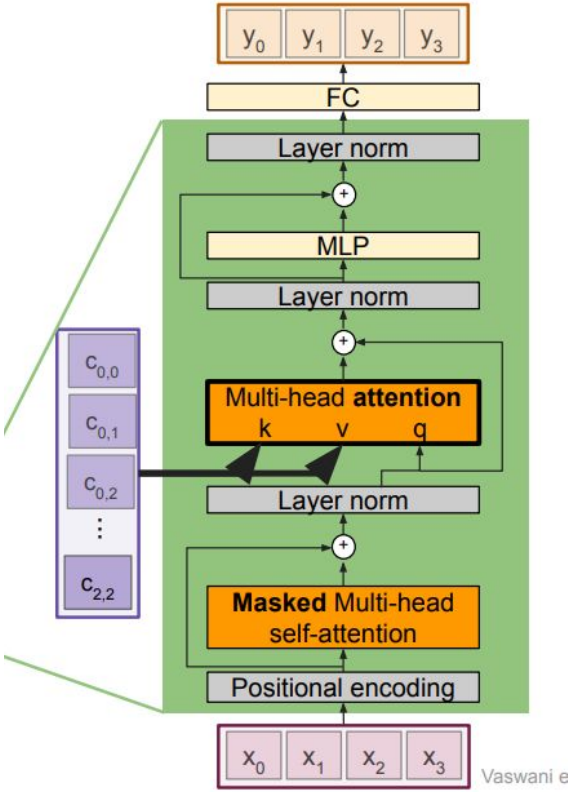
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)

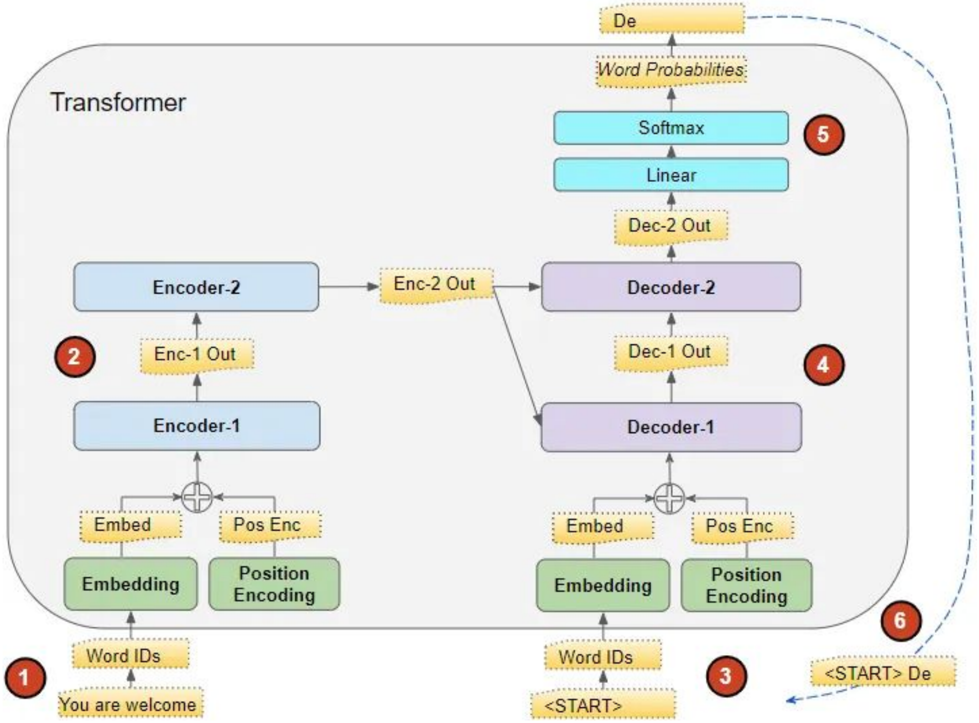
Transformer Encoder



Transformer Decoder

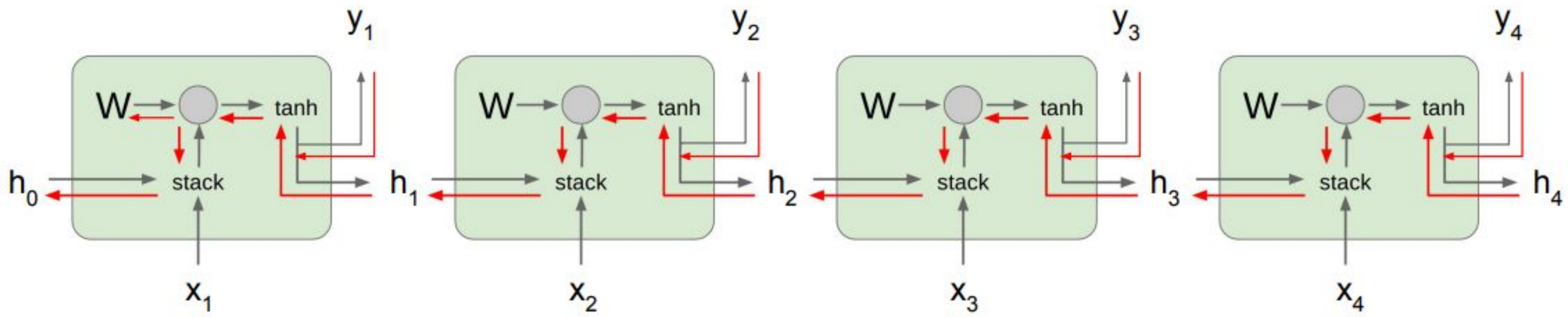


2 Layer Transformer Example

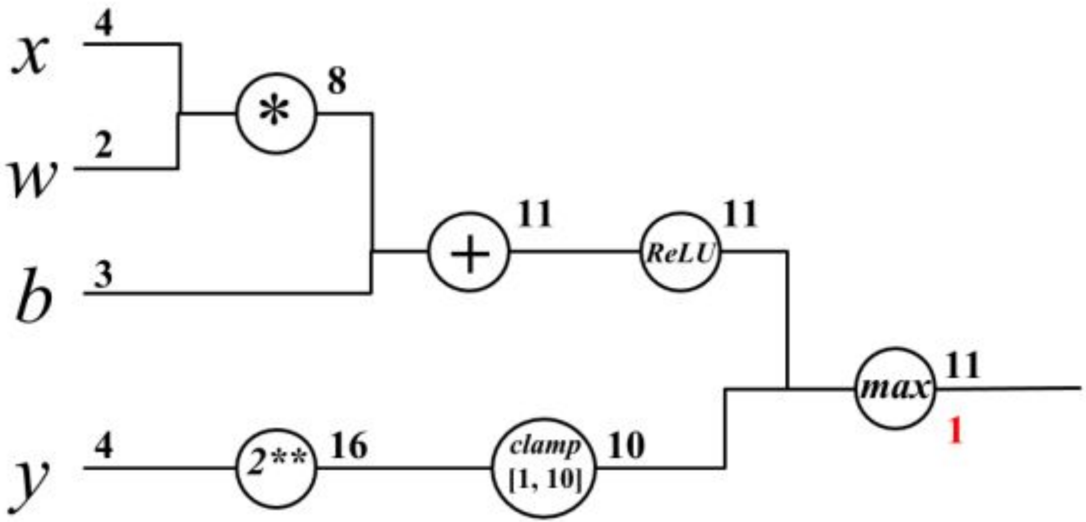


Credit: [Medium]() article by Ketan Doshi

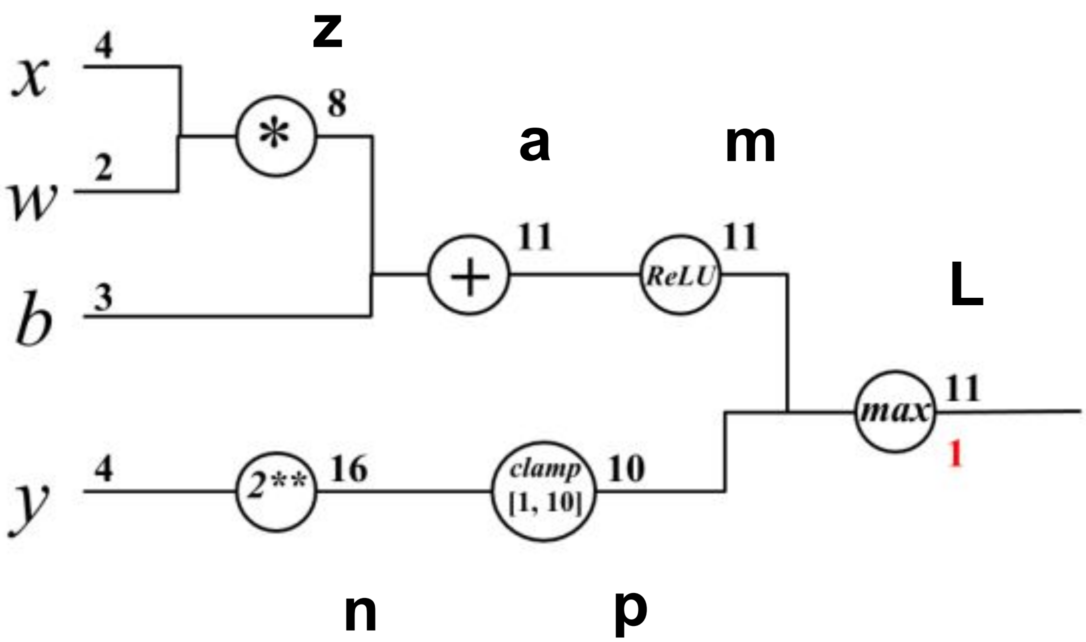
RNNs



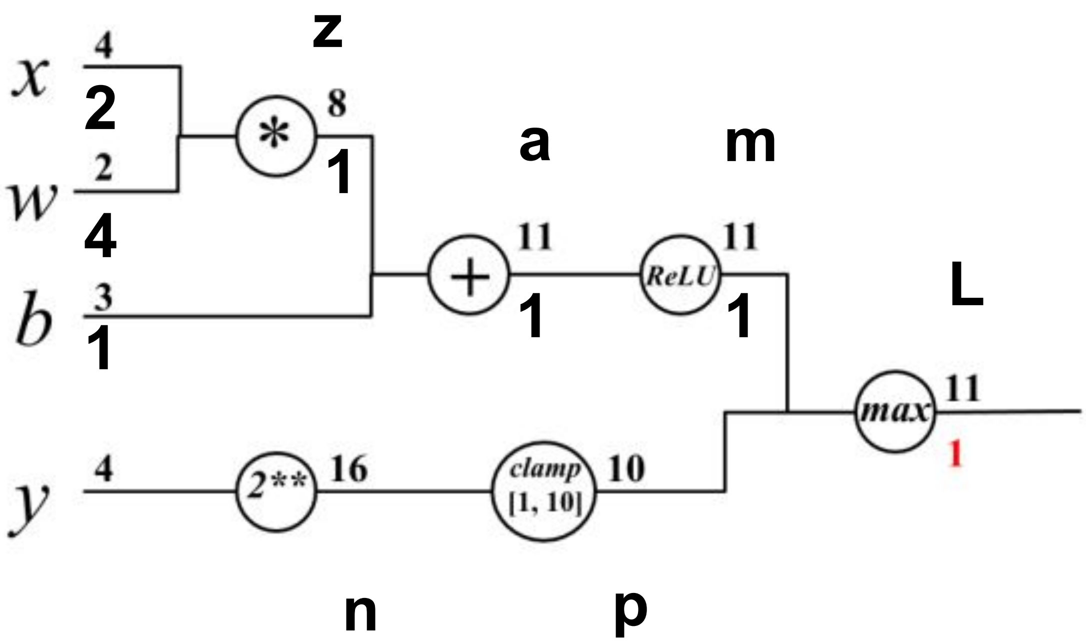
Backpropagation



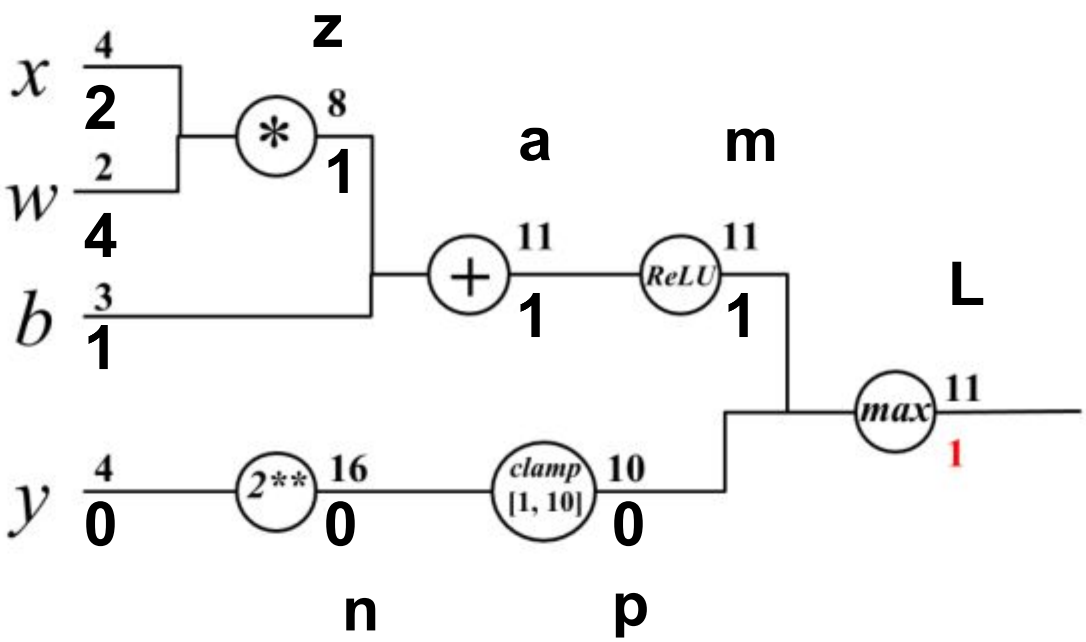
Backpropagation



Backpropagation



Backpropagation



Optimizers

Optimizer	Per-Parameter Learning Rate	Momentum
SGD	No	No
SGD + Momentum	No	Yes
AdaGrad	Yes	No
RMSProp	Yes	No
Adam	Yes	Yes

SGD

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

AdaGrad - Learning Rate Scaling (No momentum)

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp - Slowing down scaling

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Adam - Momentum + RMSProp

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

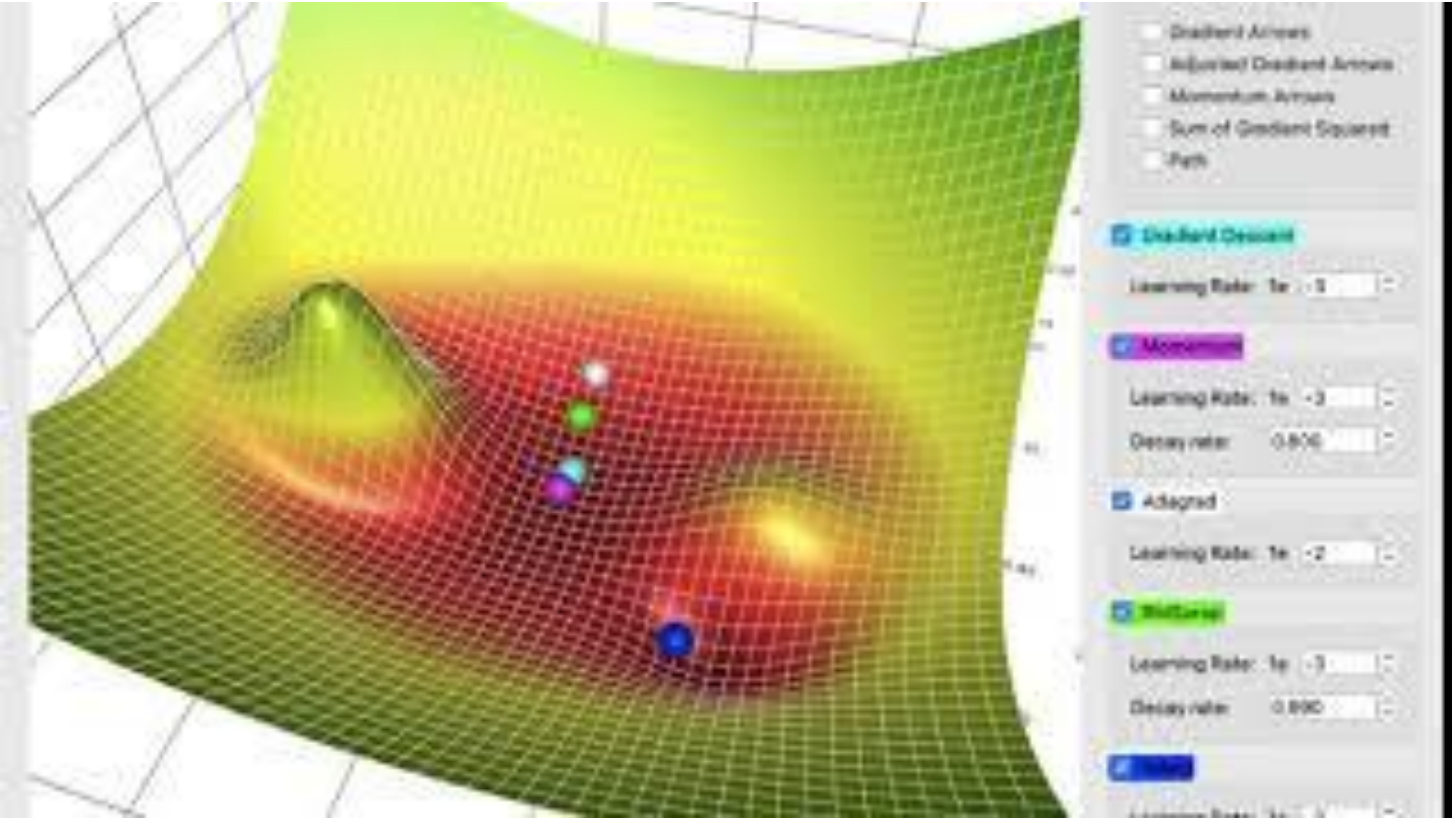
Momentum

Bias correction

AdaGrad / RMSProp

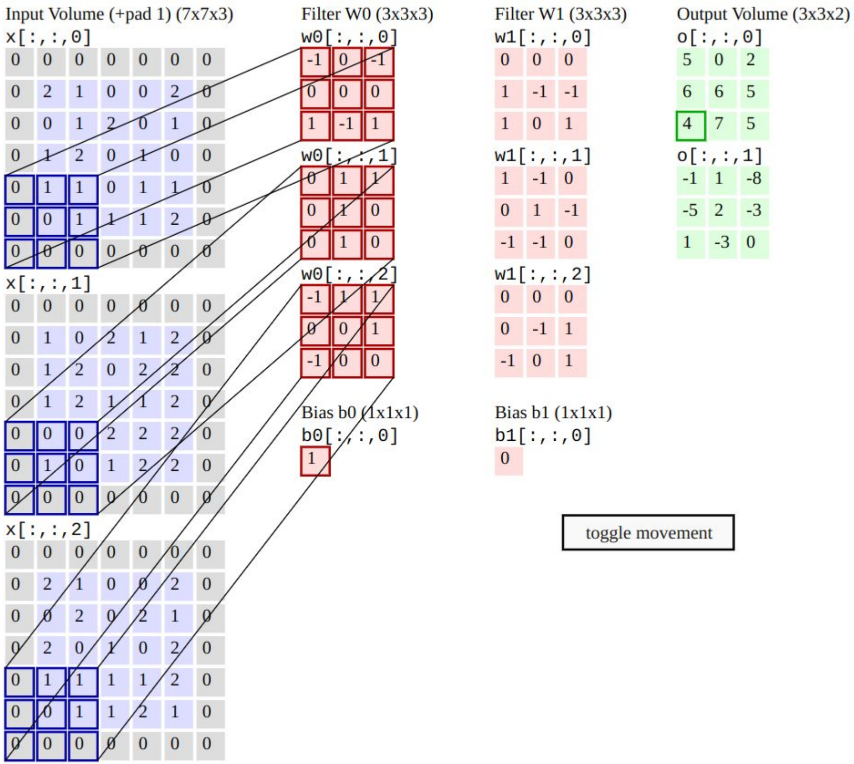
Bias correction for the fact that first and second moment estimates start at zero

Visualization



Video: Lily Jiang

CNNs

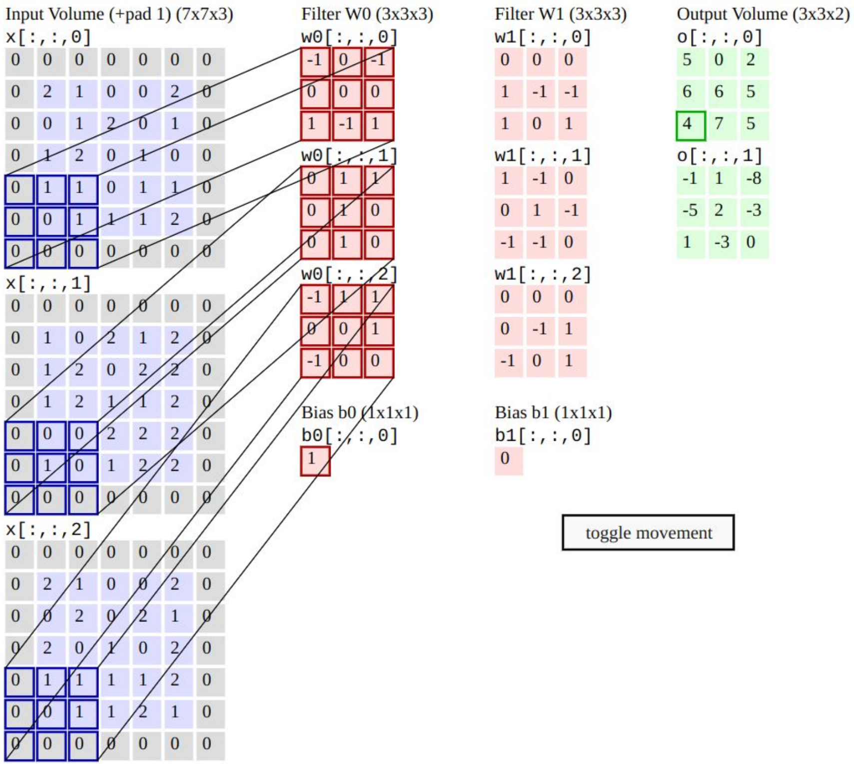


Each filter has the same number of channels as the input image.

Each filter outputs just a one channel feature image.

Therefore, the total number of channels in the output vector is the same as the number of filters.

CNNs

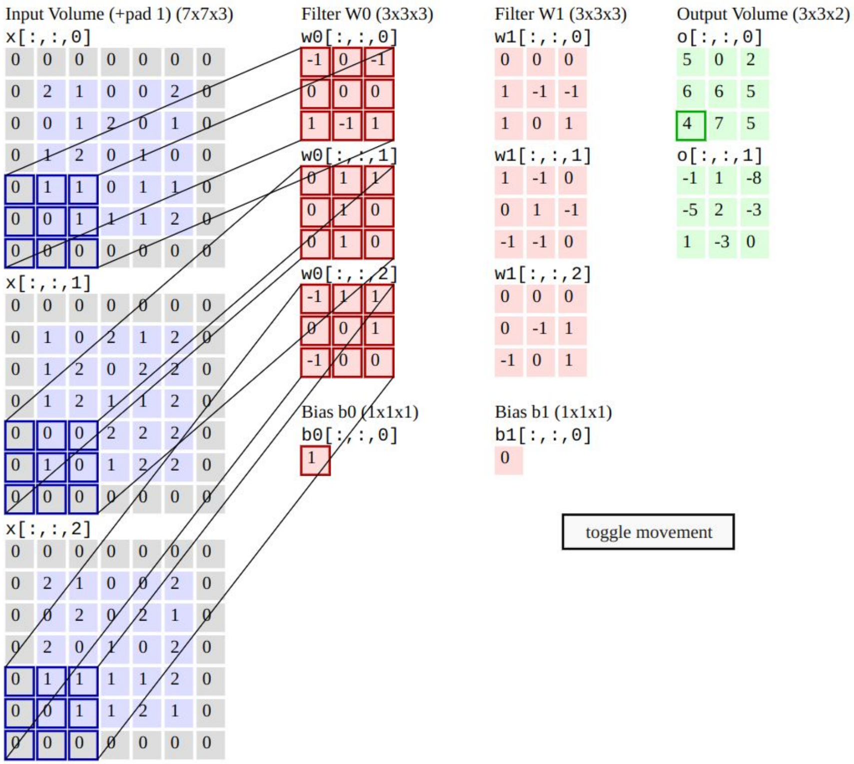


The learnable parameters are the weights and biases.

Each filter has one bias, that is applied to all channels.

For an input image with C channels, N filters, each of size (FxF), the layer has N*(C*(F*F) + 1) learnable parameters

CNNs



Input Shape: (C,H,W)
User specifies: N filters, each of shape (FxF). Padding P, and Stride S

Output Shape:
(N, H’, W’)

$W' = (W - F + 2P)/S + 1$
$H' = (H - F + 2P)/S + 1$

Note: Image on the left has Stride=2

BatchNorm vs LayerNorm

BatchNorm: Normalize across all data-points in the batch

LayerNorm: Normalize across the features of each data-point

Input shape: (N, D)

BatchNorm: Normalizes across N

LayerNorm: Normalizes across D

BatchNorm vs LayerNorm

BatchNorm: Normalize across all data-points in the batch

LayerNorm: Normalize across the features of each data-point

Input shape: (N, C, H, W)

BatchNorm: **Normalizes across N*H*W**
(calculates mean and var for each channel, across all images in the batch)

LayerNorm: **Normalizes across C*H*W** (calculates mean and var for each image, across all pixels in all channels)

BatchNorm vs LayerNorm

Input shape: (N, C, H, W)

BatchNorm: **Normalizes across N*H*W**
(calculates mean and var for each channel, across all images in the batch)

LayerNorm: **Normalizes across C*H*W** (calculates mean and var for each image, across all channels)

What is the size of their learnable parameters?

BatchNorm vs LayerNorm

Input shape: (N, C, H, W)

BatchNorm: **Normalizes across N*H*W** (reshape (N*H*W, C))
(calculates mean and var for each channel, across all images in the batch)

LayerNorm: **Normalizes across C*H*W** (reshape (N, C*H*W))
(calculates mean and var for each image, across all channels)

What is the size of their learnable parameters?

BatchNorm: **C**
LayerNorm: **(C*H*W)**

BatchNorm vs LayerNorm

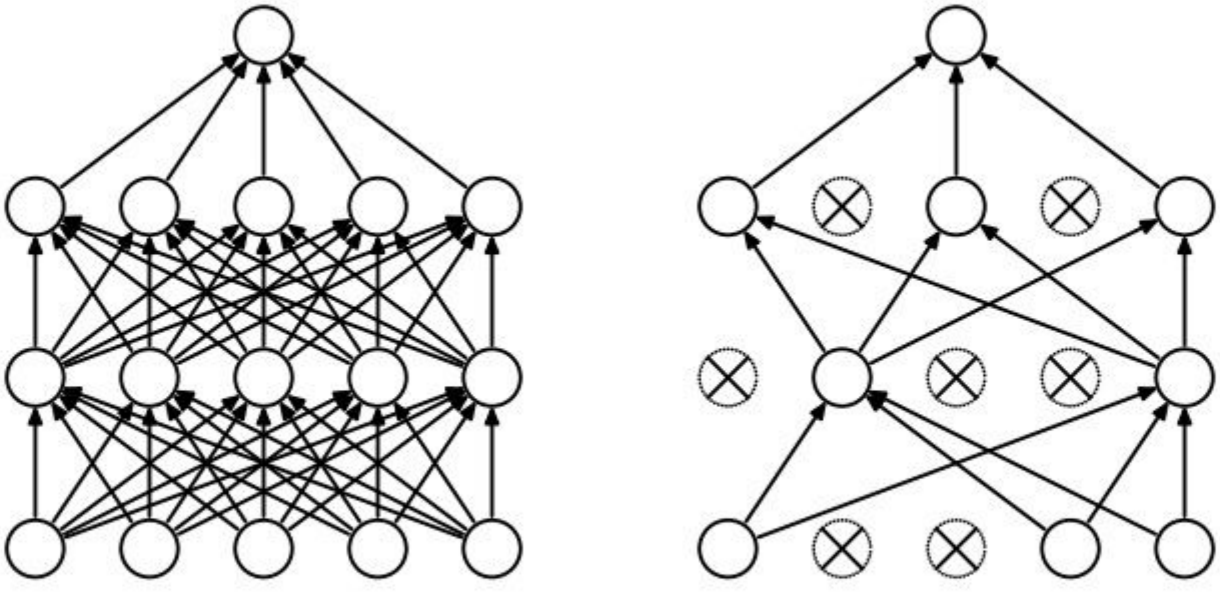
One important difference:

BatchNorm calculates the mean and var across the batch, and stores a running average which is used during test

LayerNorm performs the same during test and train

Regularization / Training a Neural Network

- L1 and L2 Regularization penalize the size of weights
- Dropout adds redundancies to learned parameters



(a) Standard Neural Net

(b) After applying dropout.