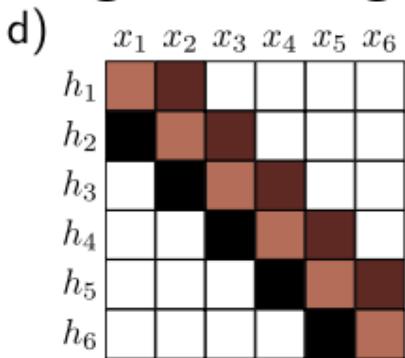
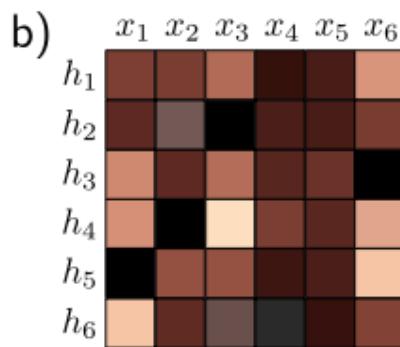
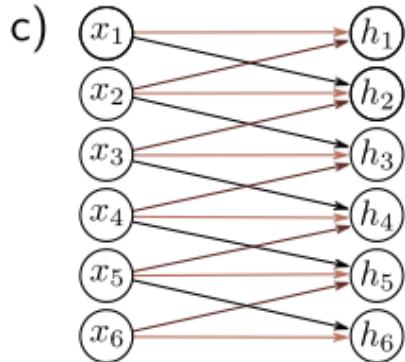
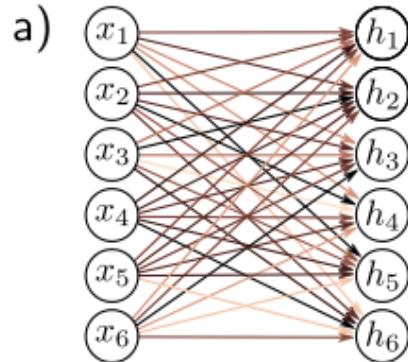


Today: convolutional neural nets
(continued)Reading: Prince through Ch 9
(so far)
Ch 10 & 11 now.

Architecture Order In Class:

MLPs \rightarrow CNNs \rightarrow Graph NN \rightarrow RNN/State-space \rightarrow Transformers

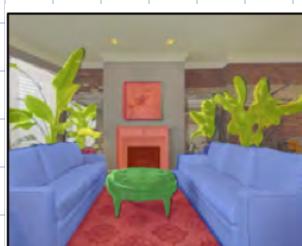
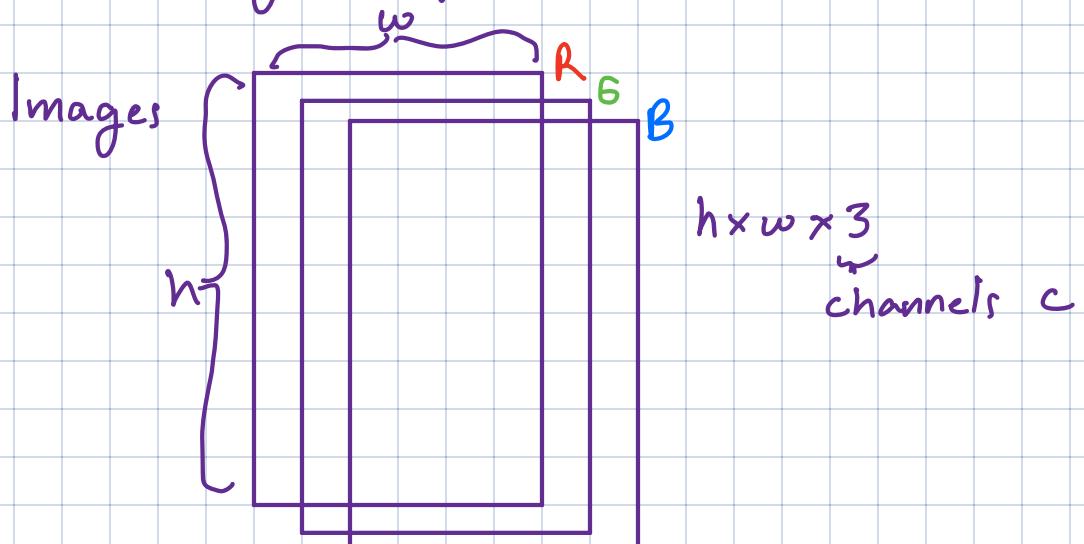
We are here

Fig 10.4
in Prince

MLP

CNN

Inspired by computer vision problems: classification,

semantic segmentation,
etc.

Step Back: 2 perspectives on a $k \times k$ conv with C_{in} input channels and C_{out} output channels

of parameters: $k^2 C_{in} C_{out}$ weights
 C_{out} biases

Perspective 1: C_{out} different $k \times k$ convs with C_{in} input channels and one output channel.

Each one has C_{in} different $k \times k$ convs with 1 input channel that are added together with a bias to give the output

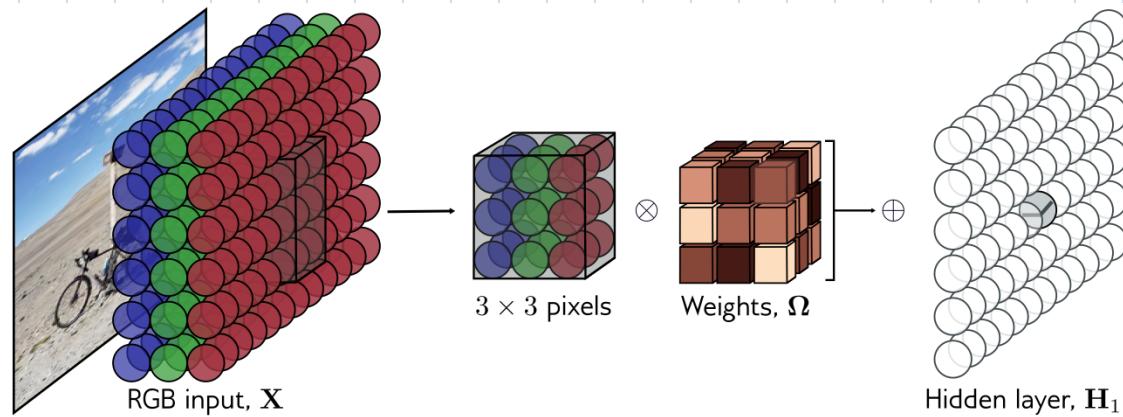


Figure 10.10
in Prince

Repeat this for each output channel.

Perspective 2: A single $k \times k$ conv, except with Weight Matrices in each box and a bias vector \vec{b} .

3x3 example:

w_{nw}	w_N	w_{ne}
w_w	w_{nc}	w_e
w_{sw}	w_s	w_{se}

C_{in}
 $C_{out} \{ W_N \}$

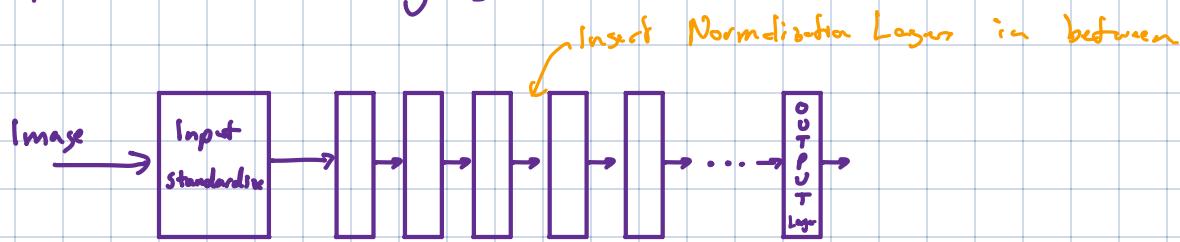
$\vec{b} \} C_{out}$

$$\vec{h}_{out} = \vec{b} + \sum_i w_i \vec{h}_{in,i}$$

Range over the $k \times k$ positions in the conv filter.

Training: Stability and Effectiveness

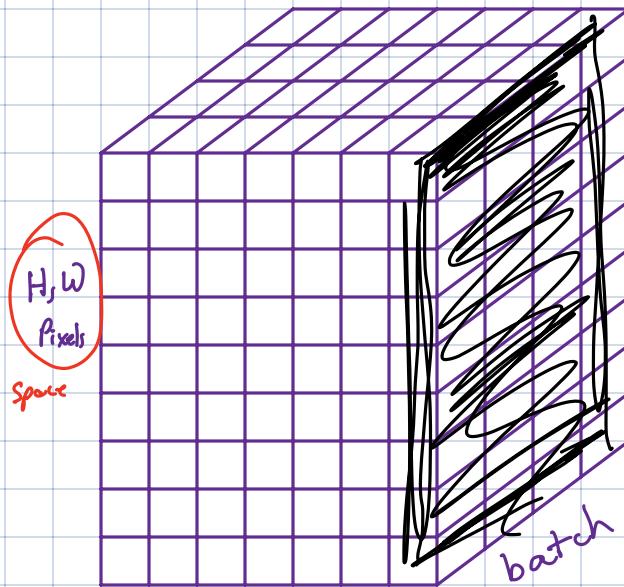
Normalization Layers



Idea: RMS Norm Layer

$$d\text{-dim} \left\{ \vec{h}_e \right\} \xrightarrow{\text{RMS Norm}} \tilde{h}_e \quad \text{has } \|\tilde{h}_e\|_{\text{RMS}} = 1$$

$$\begin{aligned} \tilde{h}_e &= \frac{\vec{h}_e}{s} \quad \text{where } s = \sqrt{d} \sqrt{\sum_i h_e[i]^2} \\ &= \frac{\vec{h}_e}{\|\vec{h}_e\|_{\text{RMS}}} + \epsilon \\ &= \frac{\vec{h}_e}{\|\vec{h}_e\|_{\text{RMS}}} + \epsilon \end{aligned}$$



Batch Norm

Average over space & batch

Let B be what is averaged over

$$m = \frac{1}{|B|} \sum_{i \in B} h_{in}$$

$$s = \sqrt{\frac{1}{|B|} \sum_{i \in B} (h_{in} - m)^2}$$

$$h_{out}[i,j] = \gamma \left(\frac{h_{in}[i,j] - m}{s + \epsilon} \right) + \delta$$

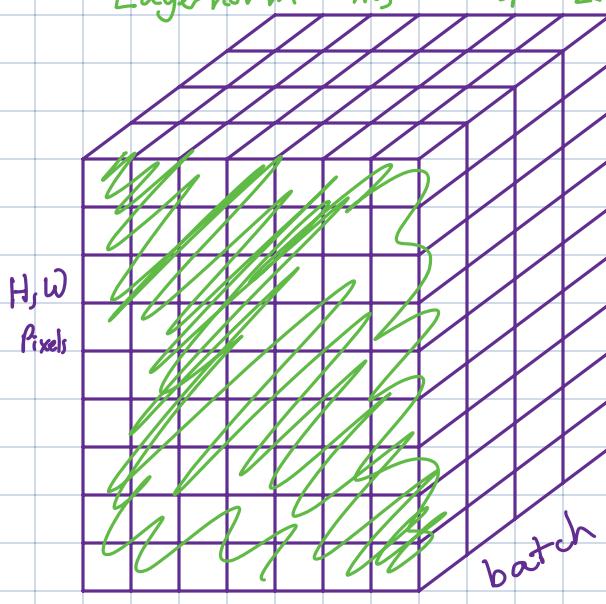
↑ learnable

Learnable center δ
scale γ

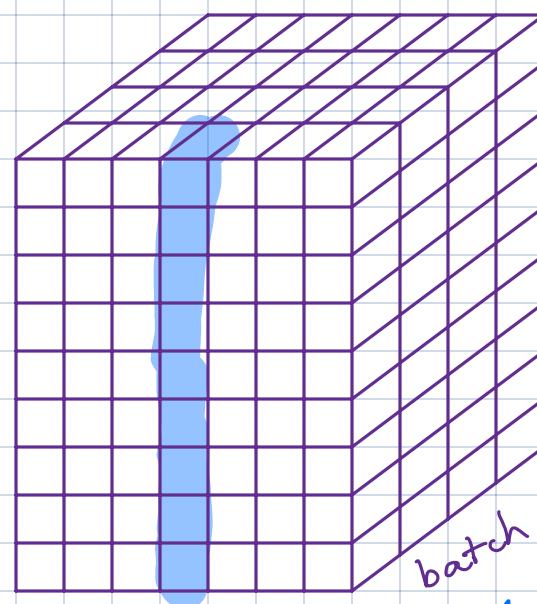
Initialize $\Rightarrow \delta = 0$
 $\gamma = 1$

Typically: Don't use weighting on δ, γ .

channels
LayerNorm: Avg over space & channels



H, W
Pixels



InstanceNorm: Avg across space only

In LayerNorm
Can have different
 γ and δ
per channel.

RMS Norm Layer don't recenter — New sample in

Question: What to do at testing/inference time?

Layer norm: No issue.

Batchnorm: ??? Don't have a batch!

Need: m, s

How to get?

- 1) Just get an average for $m \& s$ from last epoch of train
- 1a) Run an extra epoch of training with no grad updates.
- 2) Can keep a running average (exponential moving) during inference.
- 3) Just learn $m \& s$ using held-out data.

If there's a parallel to input standardization that we can apply inside our net, is there a parallel to data augmentations?

Dropout: A "data augmentation" applied inside our network during training.

Idea: Just remove/zero-out random activations inside the net

\Rightarrow Encourages internal representations to be redundant.

Typically used for MLP-style layers. (includes 1x1 convs)

$\vec{h}_e \}$ cin channels. Dropout component-wise multiplies \vec{h} with iid Bernoulli "noise"
 $\sim \vec{h}_e[i] = \begin{cases} 0 & \text{if coin is tails} \\ h_e[i] & \text{if coin is heads} \end{cases} \leftarrow \text{Prob } p \leftarrow \text{Probability of keeping} \\ 1-p \leftarrow \text{Prob of dropping}$

At training do dropout randoms. Typically p-fraction of activations survive.

What f do at test time? (inference use of model)

Try 0: Do nothing. \leftarrow Doesn't work! [Why? Messes up scale]

Solution: Scale activations by p. Replaces Multiplicative Noise by its mean.

Comment: Not what PyTorch Does.

Instead PyTorch Dropout does:

$$\tilde{h}_e[i] = \begin{cases} 0 & \text{w.p. } 1-p \\ \frac{h_e[i]}{p} & \text{w.p. } p \end{cases}$$

Residual / Skip - connections

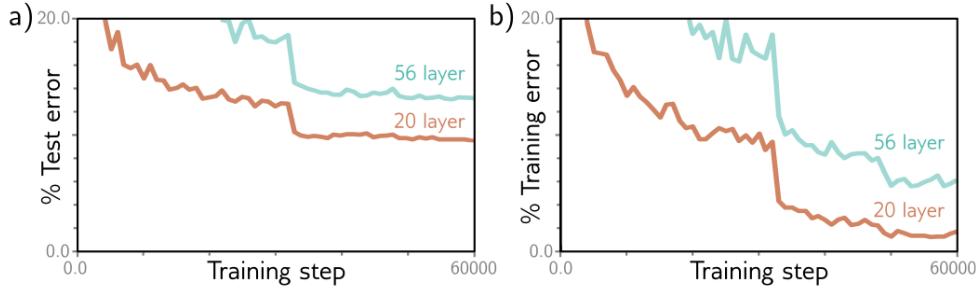
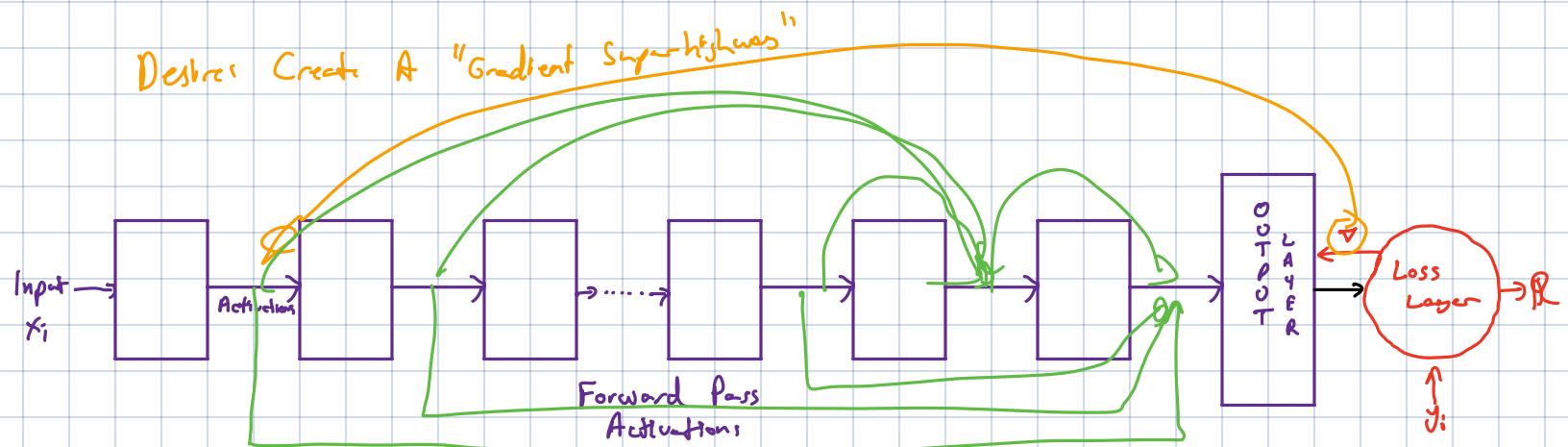
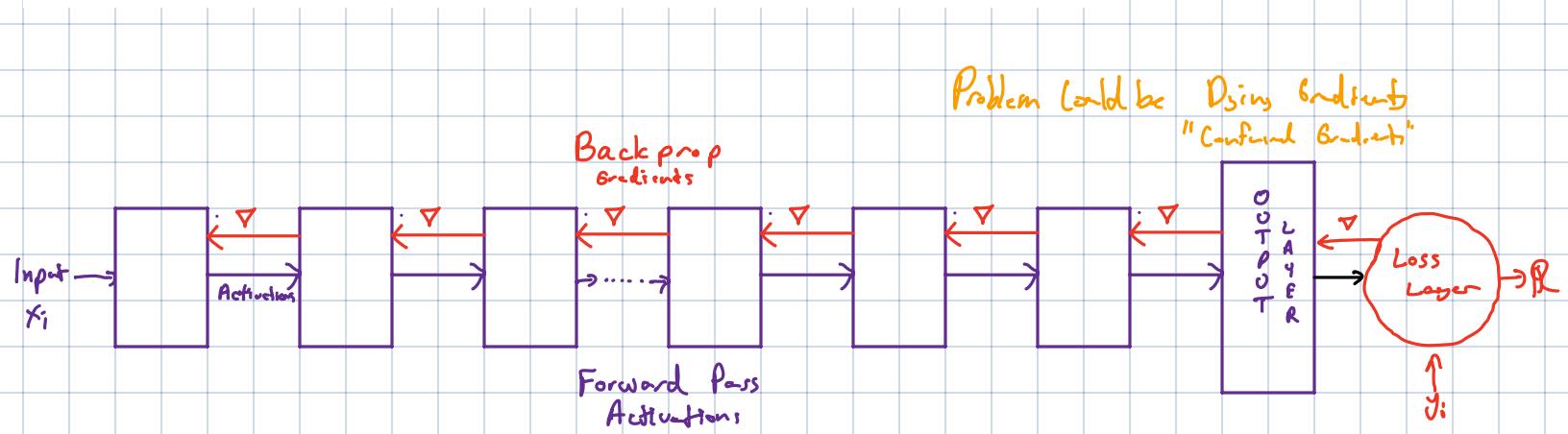


Fig 11.2 in Prince
← Feels like Underfitting.



Idea 1: Add links from outputs of all earlier layers to the input of a given layer.
Blows up activation 😢

