

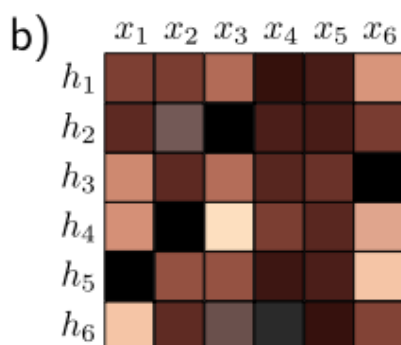
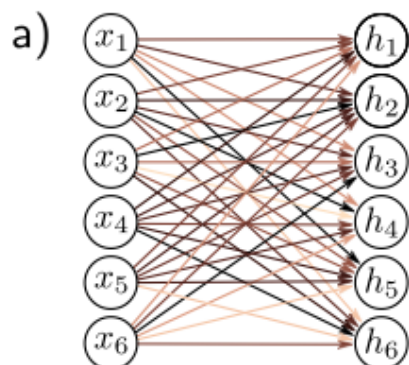
EECS 182/282A

Today: convolutional neural nets
(continued)

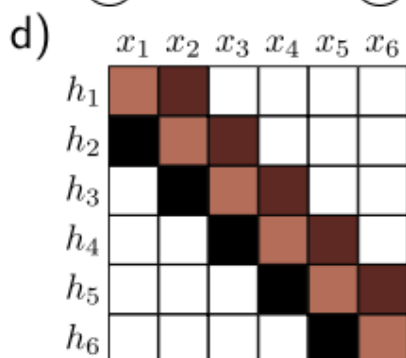
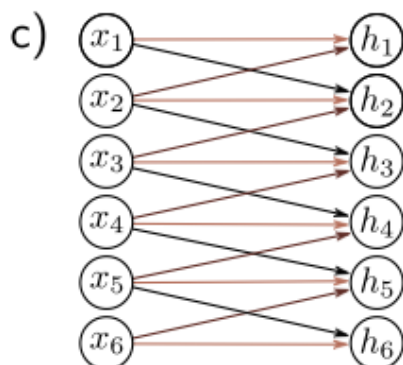
Reading: Prince through Ch 9
(so far)
Ch 10 & 11 now.

Architecture Order In Class:

MLPs \rightarrow CNNs \rightarrow Graph NN \rightarrow RNN/state-space \rightarrow Transformers
we are here



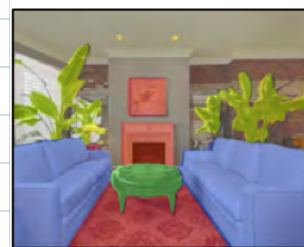
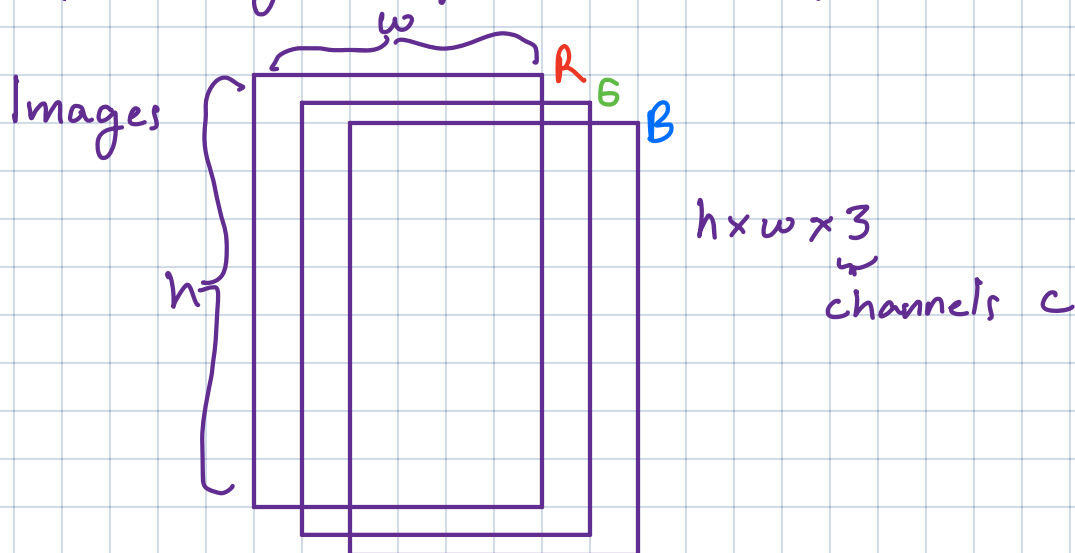
MLP



CNN

Fig 10.4
in Prince

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

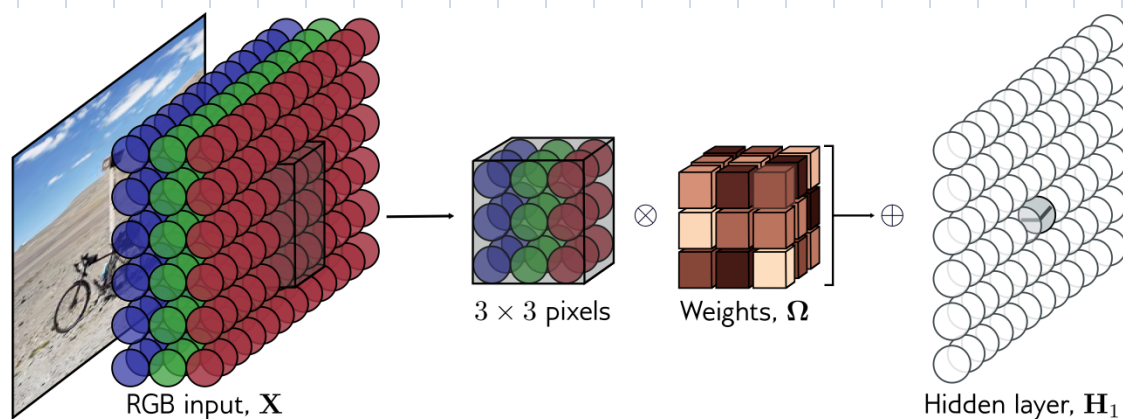


Figure 10.10
in Prince

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_{mc}	w_e
w_{sw}	w_s	w_{se}

c_{in}
 $\{ w_n \}$

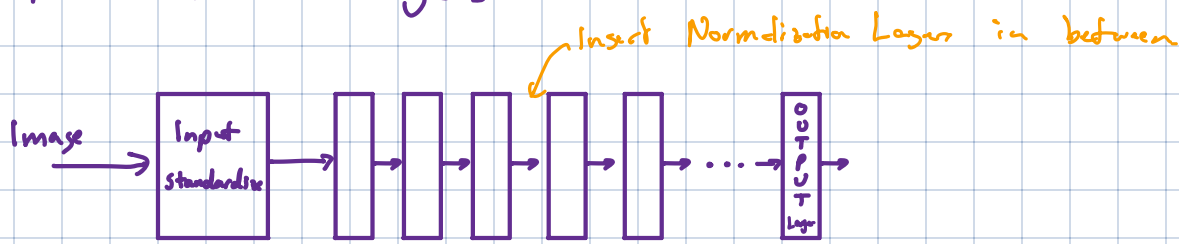
$\vec{b} \} c_{out}$

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

i \nwarrow ranging over the $k \times k$ positions in the conv filter.

Training: Stability and Effectiveness

Normalization Layers

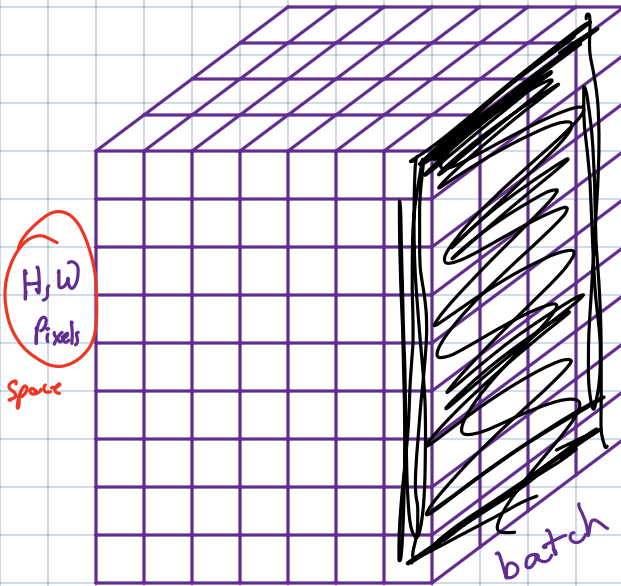


Idea: RMS Norm Layer



$$\tilde{h}_e = \frac{\vec{h}_e}{s} \quad \text{where } s = \sqrt{\frac{1}{d} \sum_i h_{e,i}^2 + \epsilon}$$

$$= \frac{\vec{h}_e}{\|\vec{h}_e\|_{\text{RMS}} + \epsilon}$$



Batch Norm

Average over space & batch

Let B be what is averaged over

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

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

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

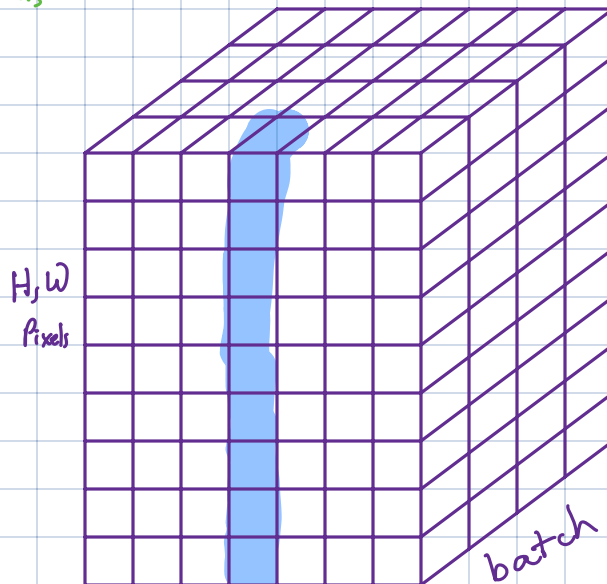
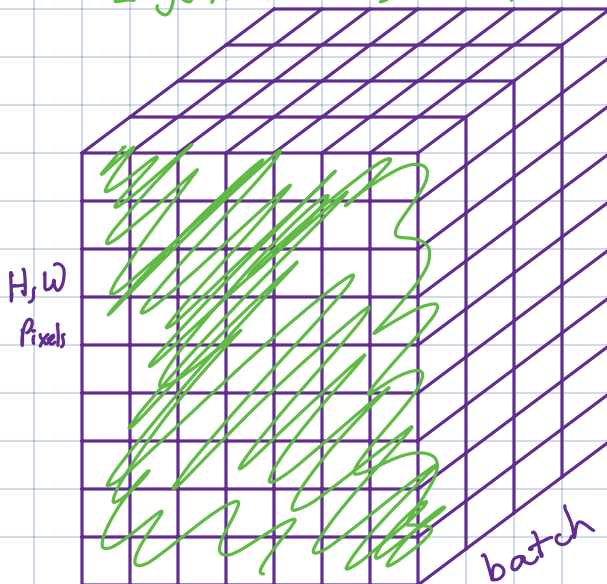
\uparrow learnable $\quad \quad \quad \uparrow$ learnable

Learnable center δ
scale γ

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

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

Layer norm: Avg over space & channels



Layer norm
Can have different γ or δ per channel.

Instance Norm: Avg across space only

RMS Norm Layers don't recenter — Norm compute m

Question: What to do at testing/inference time?

Layer norm: No issue.

Batch norm: ??? Don't have a batch!

Need: m, s

How to get?

- 1) Just get an average for m & s from last epoch of training
- 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 Representation to be redundant.

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

\vec{h}_l cin channels. Dropout component-wise multiplies \vec{h} with iid Bernoulli "noise"

$$\tilde{h}_l[i] = \begin{cases} 0 & \text{if coin is tails} \\ h_l[i] & \text{if coin is heads} \end{cases} \leftarrow \text{Prob } p \leftarrow \text{Probability of keeping} \\ 1-p \leftarrow \text{Prob of dropping}$$

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

What to 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:

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

Residual/Skip-connections

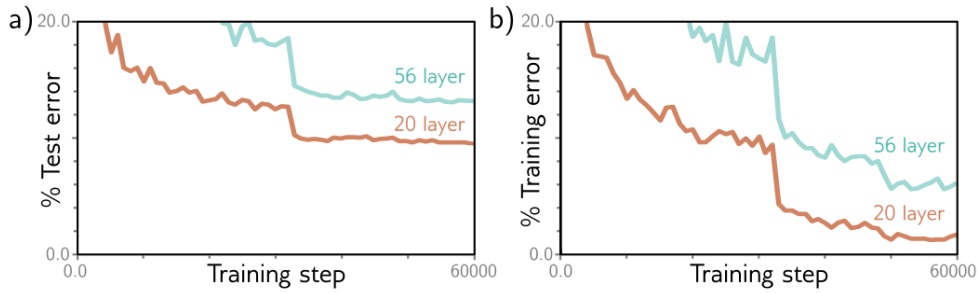
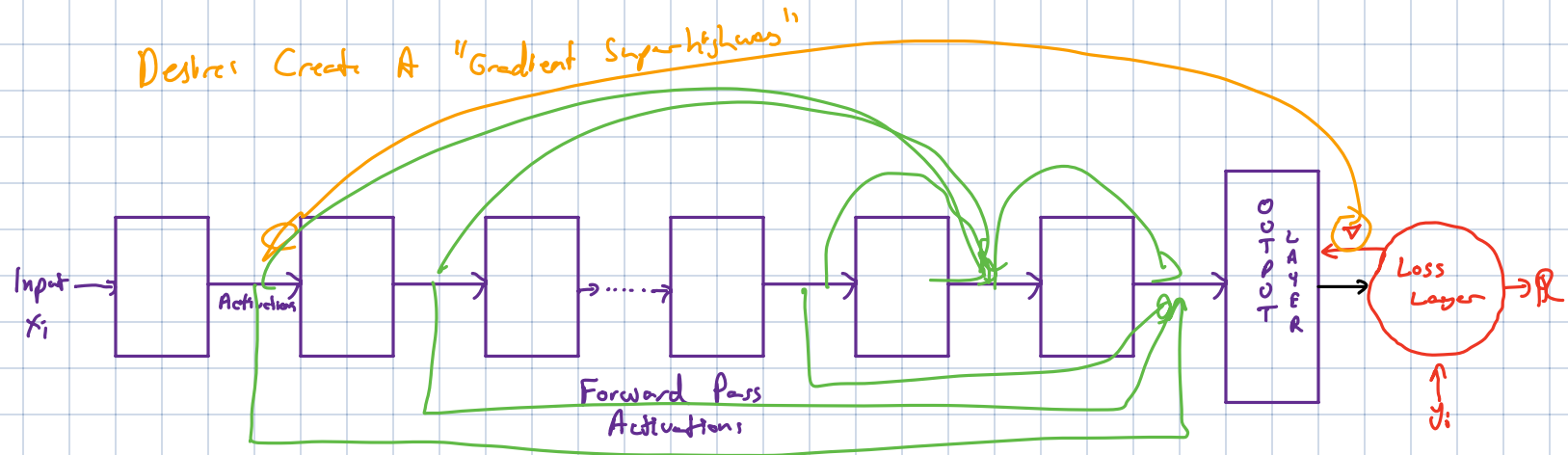
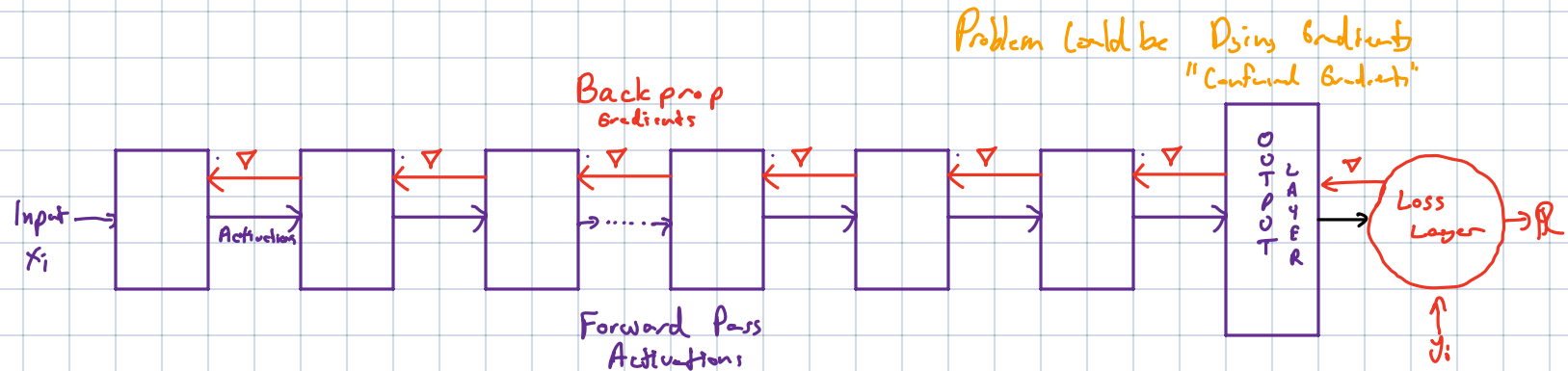


Fig 11.2 in Prince
 ← feels like Underfitting.



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

Skip Connections: Add input to the output of a layer

