ML and Optimization Lecture 21

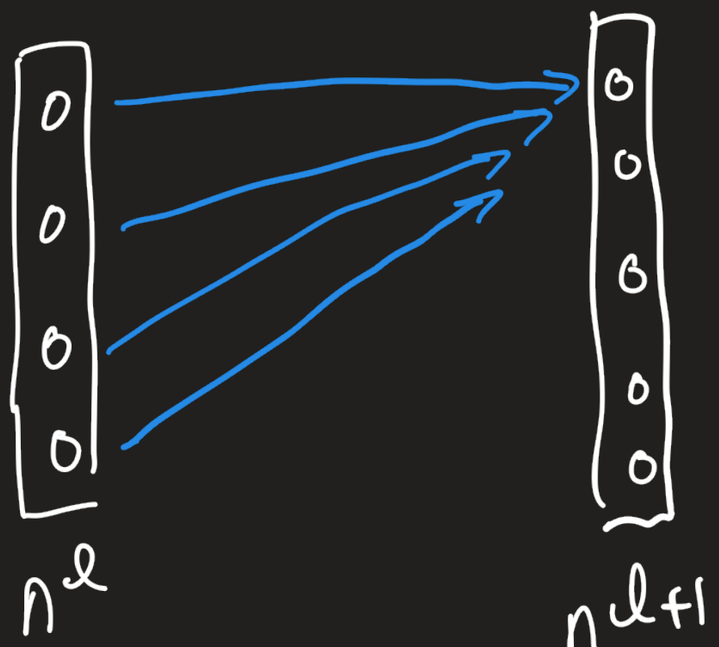
- Convolutional Neural Networks continued
- issues of deep NNs:
  - overfitting
  - vanishing & exploding gradients

remedies:

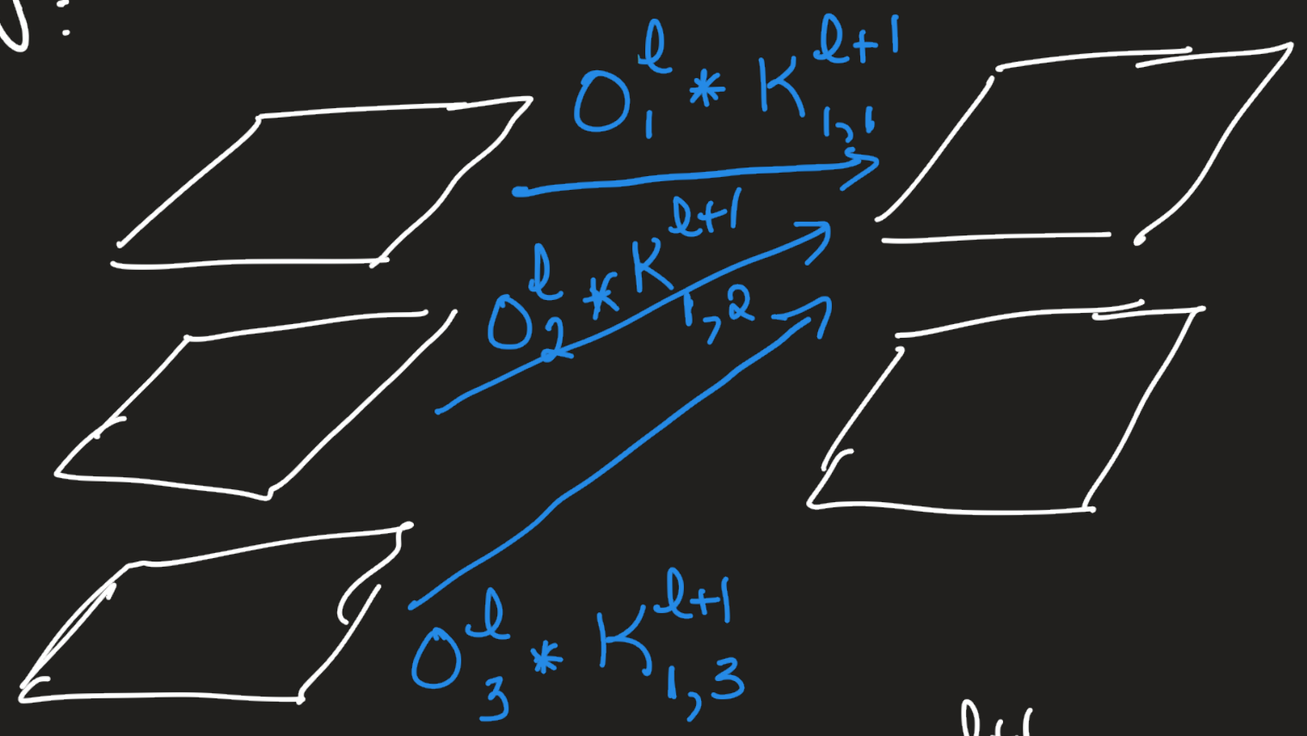
- dropout (2014)
- (2015) batch normalization

Multi-channel CNNs

MLP:



CNN:



$$A_1^{l+1} = O_1^l * K_{1,1}^{l+1} + O_2^l * K_{1,2}^{l+1} + O_3^l * K_{1,3}^{l+1} + b_1^{l+1}$$

$$O_1^{l+1} = \sigma(A_1^{l+1})$$

In general if layer l has n_l channels, then

$$A_i^{l+1} = \left(\sum_{j=1}^{n_l} O_j^l * K_{i,j}^{l+1} \right) + b_i^{l+1}$$

for $i = 1, \ldots, n_{l+1}$

$$O_i^{l+1} = \sigma(A_i^{l+1})$$

What about backprop?

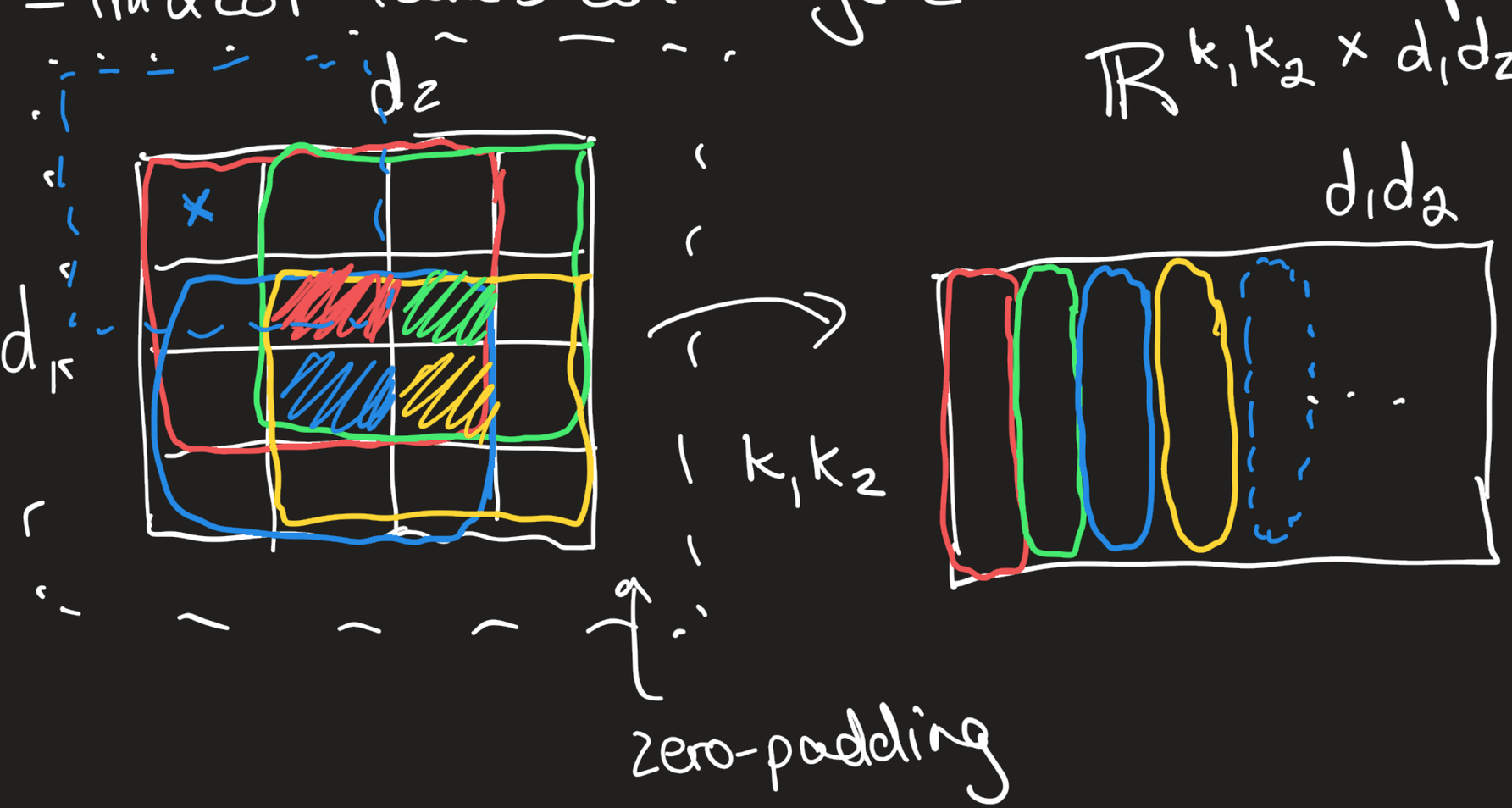
- convolutions

Convolutions in practice (GEMM)

cf. Manas Sahni "Anatomy of a High-Speed Convolution"

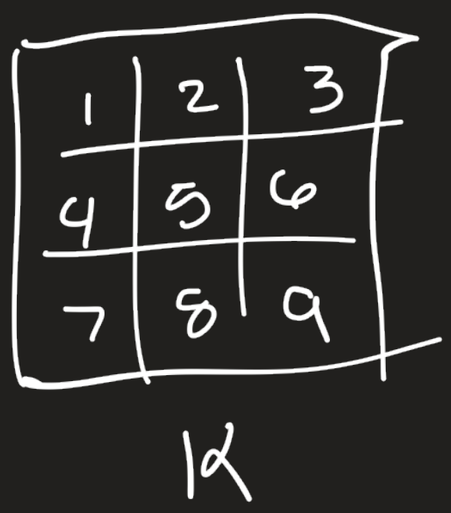
reduce convolutions to the im2col operation and GEMMs

- im2col takes an image $\in \mathbb{R}^{d_1 \times d_2}$ and maps to a matrix in $\mathbb{R}^{k_1 k_2 \times d_1 d_2}$ (corresponding to zero-padding the input and convolving by a $k_1 \times k_2$ kernel)



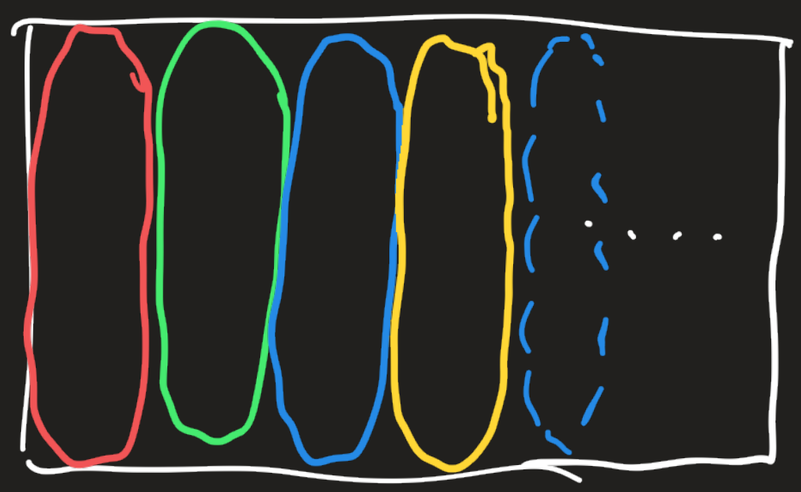
Then note that

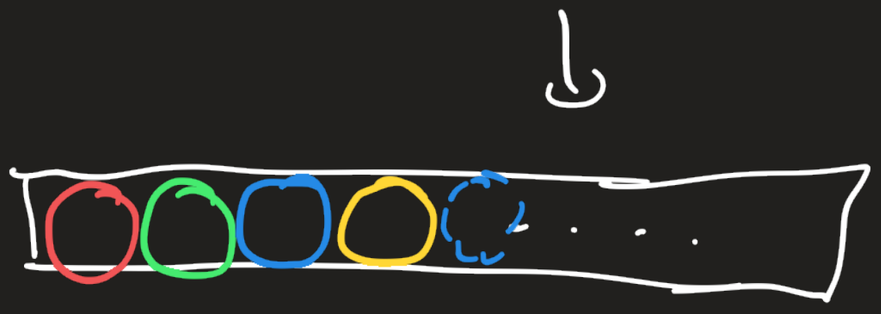
$$\mathrm{vec}(I * K) = \mathrm{vec}(K) \cdot \mathrm{im2col}(I)$$



K

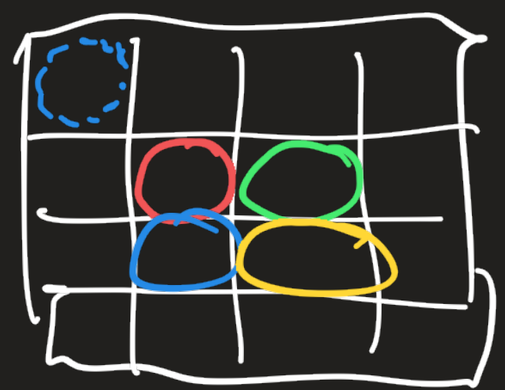
vec(K)





vec(K) · im2col(I)

unvec



Issues with deep NNs (not just CNNs):

- overfitting (too much capacity for the amount of training data)
- vanishing & exploding gradients ⇒ slow learning

- hyperparameter selection
  - optimal kernel sizes
  - # of channels per layer
  - # of layers
  - pooling types & locations
  - stride sizes
  - learning rates (function of minibatch size)
  - weight decay amount
  - more ...

Vanishing & Exploding Gradients

Phenomenon that

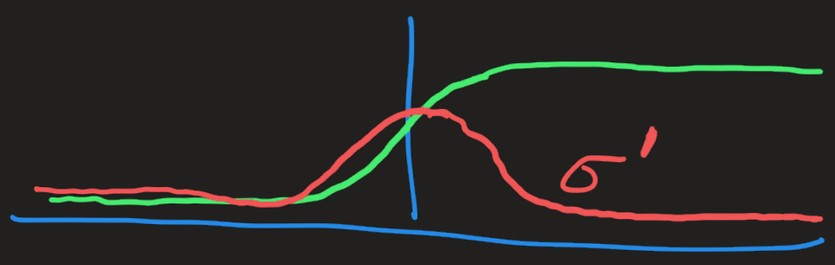
$$\| \nabla_{a^l} f \|_2 \xrightarrow{l \to 1} \begin{cases} 0 & \text{vanishing gradients} \\ \infty & \text{exploding gradients} \end{cases}$$

Why? Because of the chain rule. E.g. for MLPs

$$\nabla_{a^l} f = \operatorname{diag}\left(\sigma'(a^{l+1})\right)\left(W^{l+1}\right)^T \nabla_{a^{l+1}} f$$

Two considerations)

1) The saturation of our activation function



If preactivations are far from origin, $\sigma'(a^{l+1})$ close to zero, so

$$\| \nabla_{a^l} f \|_2 \ll \| \nabla_{a^{l+1}} f \|_2$$

2) The norm of our weight matrix W^{l+1}

- if on average $\|W^{l+1}x\|_2 \geq \|x\|_2$

then we get $\|\nabla_{O^l} f\|_2 \geq \|\nabla_{O^{l+1}} f\|_2$

- if on average $\|W^{l+1}x\|_2 \leq \|x\|_2$

then we get $\|\nabla_{O^l} f\|_2 \leq \|\nabla_{O^{l+1}} f\|_2$

Consequence: naively training deep NN architectures either fails or gives poor performance.

Remedy: Batch Normalization (2015)

Idea: the feature distributions at each layer vary wildly, so normalize them so they look like standard gaussians

In convex ML we saw the advantage of normalizing our input data so the features are on the same scale

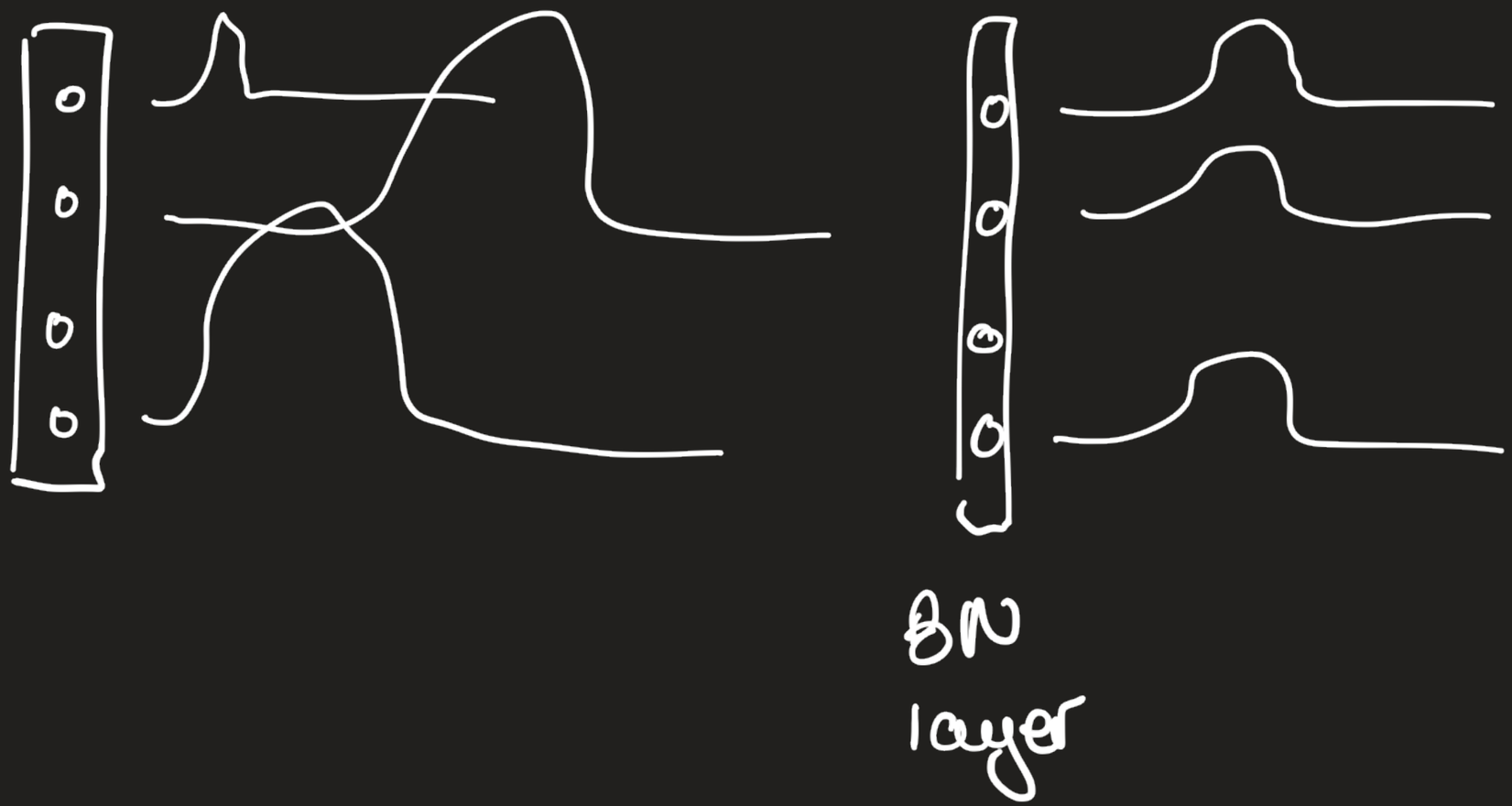
ex: training to predict credit default probability

$x = [\text{salary} \quad \text{age}]$

to get the features on the same scale we normalized them

$$x \longmapsto \left[\frac{\text{salary} - \mu_{\text{salary}}}{\sigma_{\text{salary}}} , \frac{\text{age} - \mu_{\text{age}}}{\sigma_{\text{age}}} \right]$$

Idea of BN: add normalization layer after a regular layer to rescale and center the feature distributions



BN layer

Between layer l and $l-1$ add a BN layer:

$$a^l = BN_{\gamma^l, \beta^l}(W^l o^{l-1})$$

$$o^l = \sigma(a^l)$$

μ_B = minibatch mean $= \frac{1}{m}\sum_{i=1}^{m}(o^{l-1})_i$

σ_B^2 = minibatch variance $= \frac{1}{m}\sum_{i=1}^{m}((o^{l-1})_i - \mu_B)^2$

$$BN_{\gamma^l, \beta^l}(o^{l-1}) = \gamma^l \odot \frac{o^{l-1} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} + \beta^l$$

where $\gamma^l \in \mathbb{R}^{n_{l-1}}$ is a scale vector and $\beta^l \in \mathbb{R}^{n_{l-1}}$ is a shift vector

are useful to ensure we are in the nonlinear region of the activation function

Forward & Backward pass during training is straightforward b/c μ_B and σ_B^2 are computed using minibatch means (model.train() in pytorch)

What about inference time (model.eval)

1) After fitting, run all our training data through and compute

$$\mu^l = \frac{1}{n_{train}} \sum_{i=1}^{n_{train}} (o^{l-1})_i \in \mathbb{R}^{n_{l-1}}$$

$$\sigma^l = \frac{1}{n_{train}} \sum_{i=1}^{n_{train}} ((o^{l-1})_i - \mu^l)^2 \in \mathbb{R}^{n_{l-1}}$$

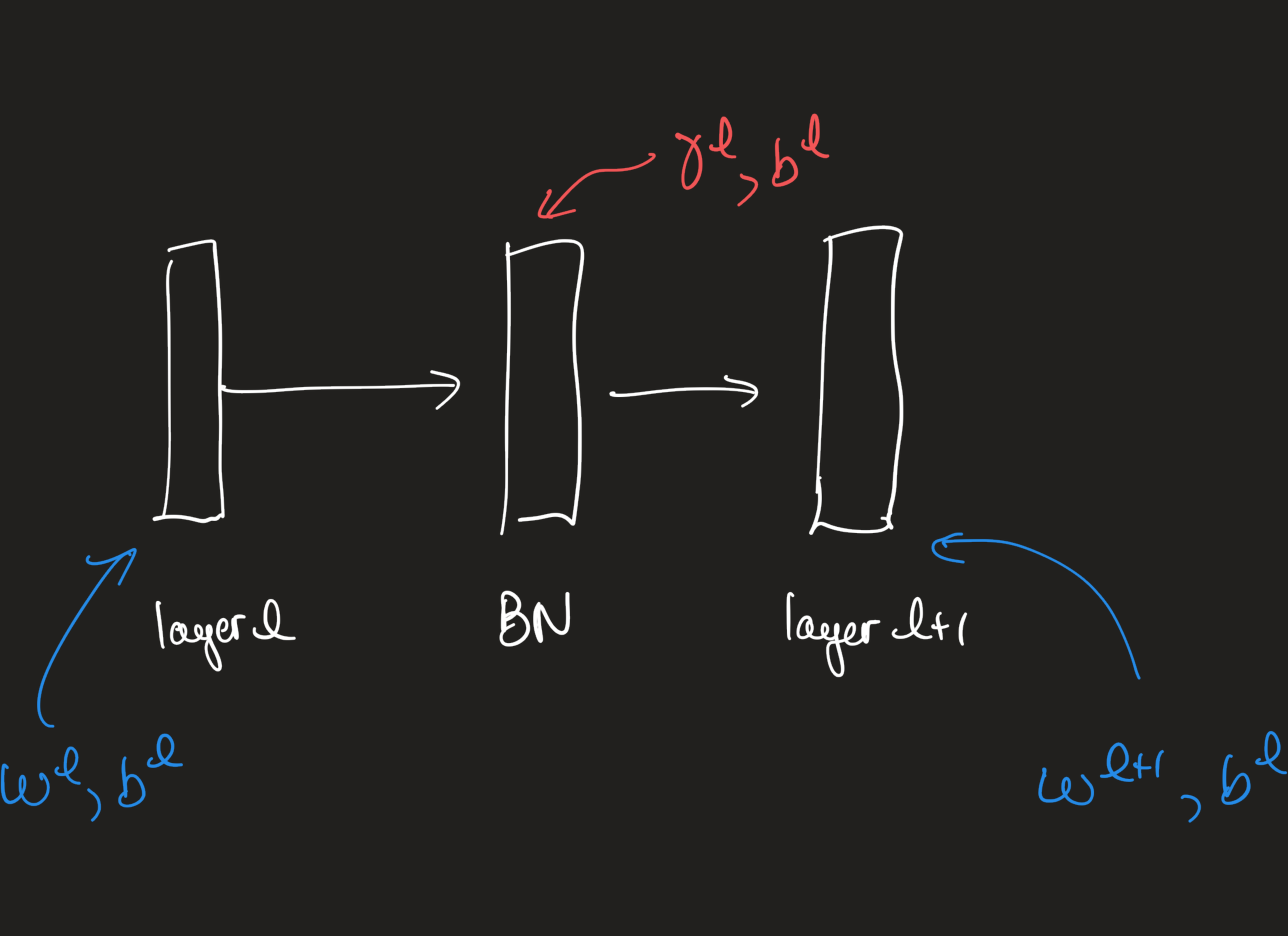
and use these

2) or maintain moving average estimates of population mean & variance

$$\mu^{[l]} \leftarrow \beta_1 \mu^{[l]} + (1-\beta_1)\mu_B^{[l]} \quad \text{for each minibatch}$$

$$\sigma^{[l]} \leftarrow \beta_2 \sigma^{[l]} + (1-\beta_2)\sigma_B^{[l]} \quad \text{for each minibatch}$$

and use $\mu^{[l]}$ and $\sigma^{[l]}$ at inference time


γ^l, b^l
layer l
BN
layer $l+1$
w^l, b^l
w^{l+1}, b^l