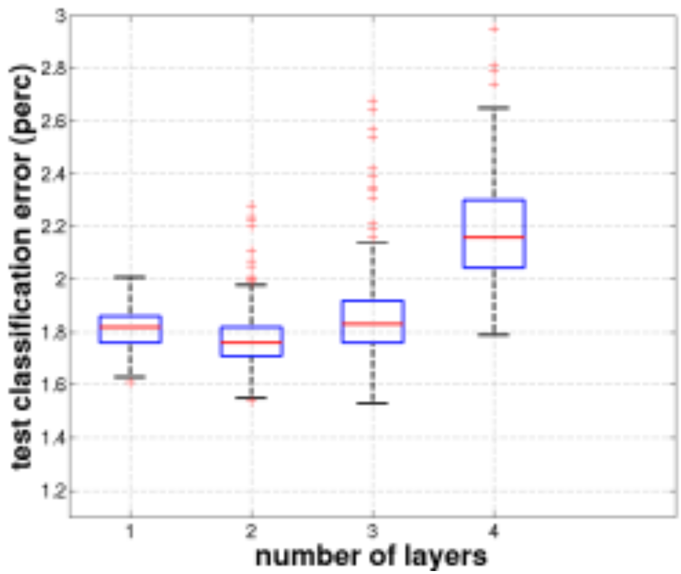
CMSC 510 – L20
Regularization Methods for Machine Learning

Part 20a: BatchNorm

Instructor:
Dr. Tom Arodz

Training deep nets

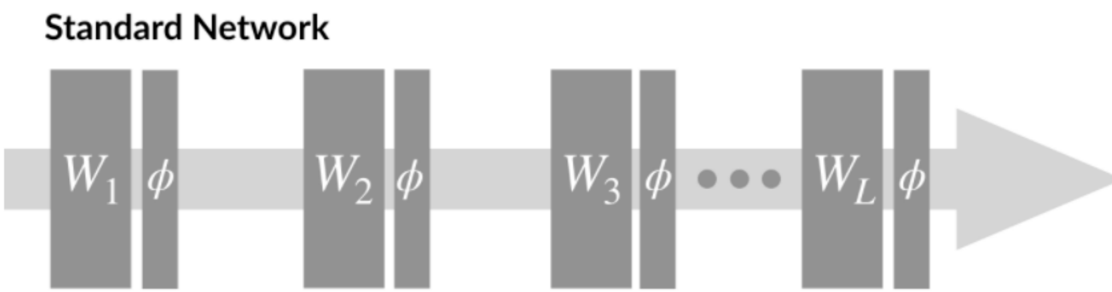
- Historically, deep networks (with many layers) were difficult to train



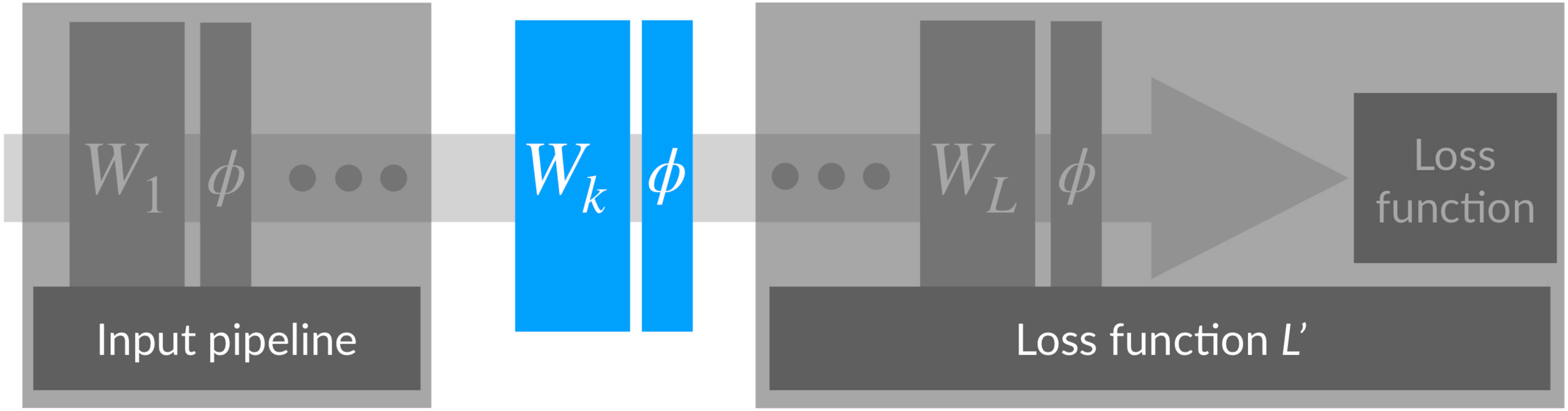
- A lot effort in recent years went into training deep nets easier
  - ReLU instead of sigmoid
  - Unsupervised pre-training
  - Normalization techniques

Batch normalization

- A standard deep net has layers, where each layer is
  - a linear transformation Wx, where x is input from previous layer
  - a nonlinear activation function acting on Wx, e.g. ReLU(Wx)

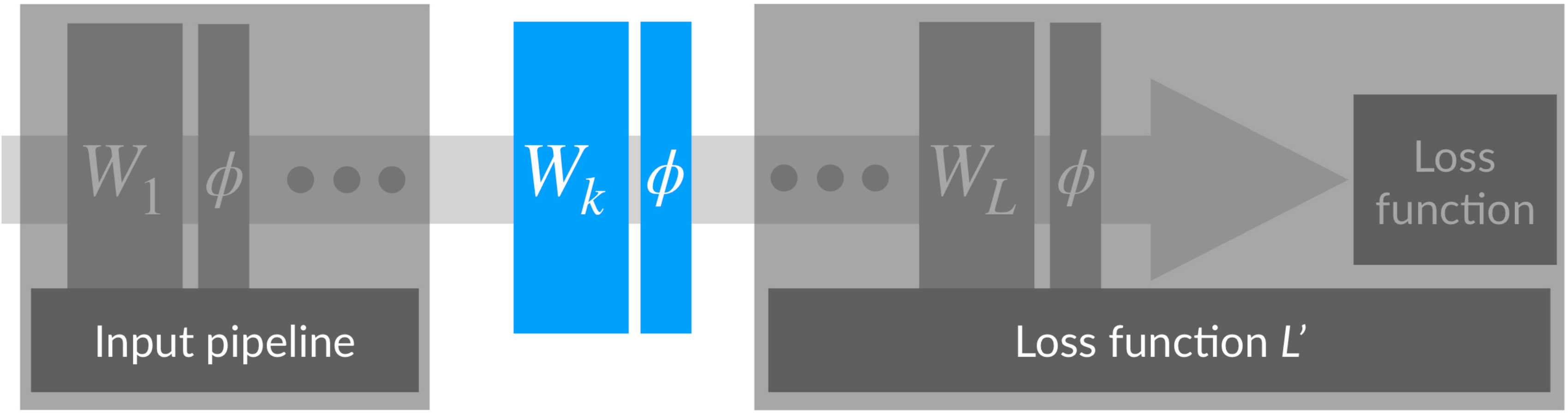


- We can look at the training from the perspective of a single layer – as if the other layers were constant



Batch normalization

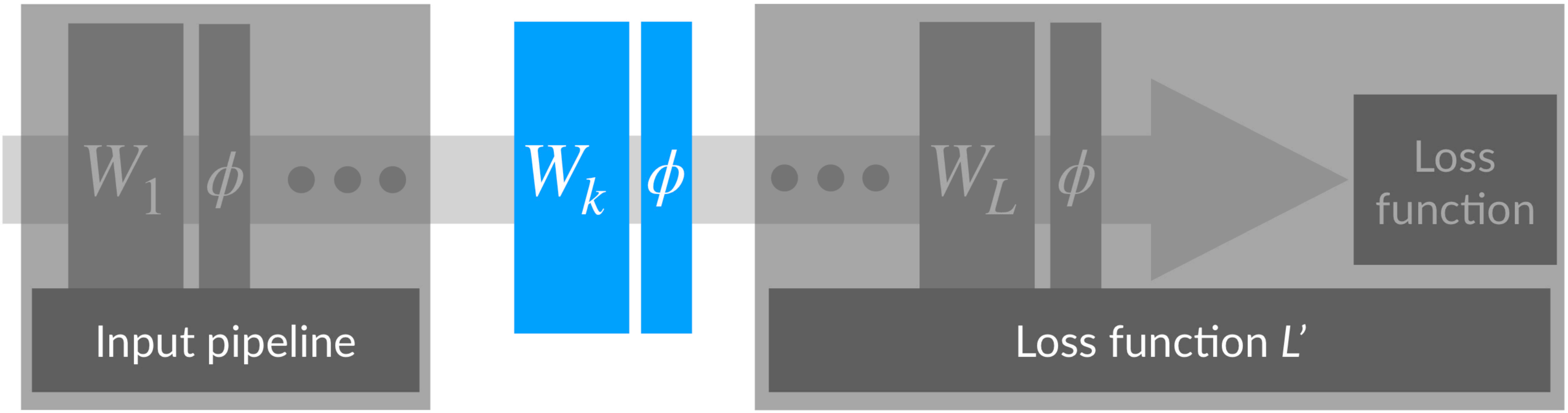
- We can look at the training from the perspective of a single layer – as if the other layers were constant



- Layer k will learn how to transform its input (output of layers 1,…,k-1) into something that minimizes the "new loss" (layers k+1,…,L + original loss).

Batch normalization

- We can look at the training from the perspective of a single layer – as if the other layers were constant

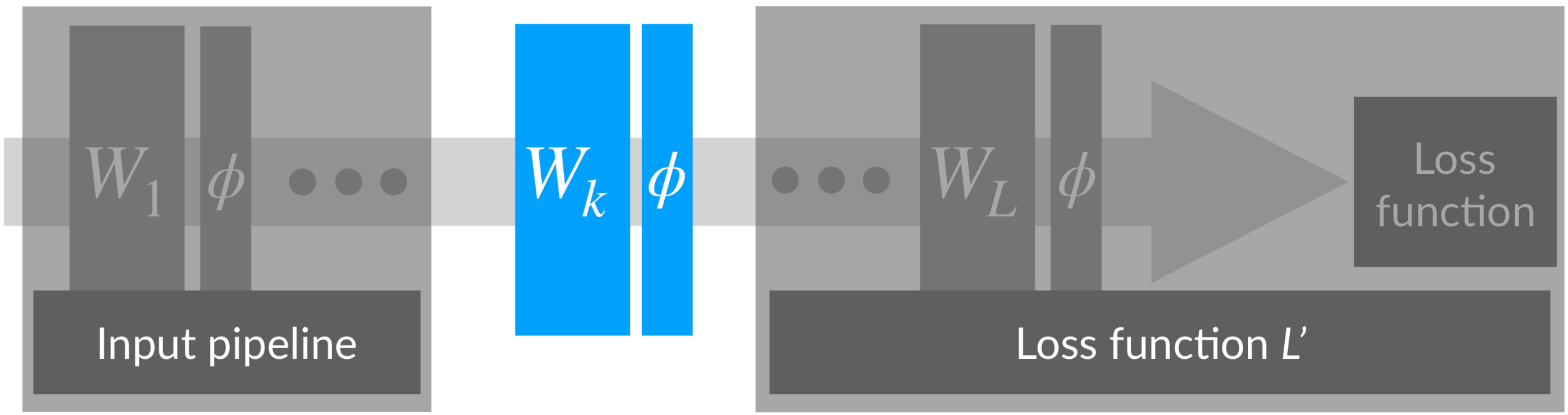


- If layers 1,…,k-1 are not constant (are trained), the distribution of input to layer k changes all the time
- If layers k+1,…,L are not constant (are trained), the "new loss" also changes all the time

- If we fix the first problem, inductively we also fix the second problem

Batch normalization

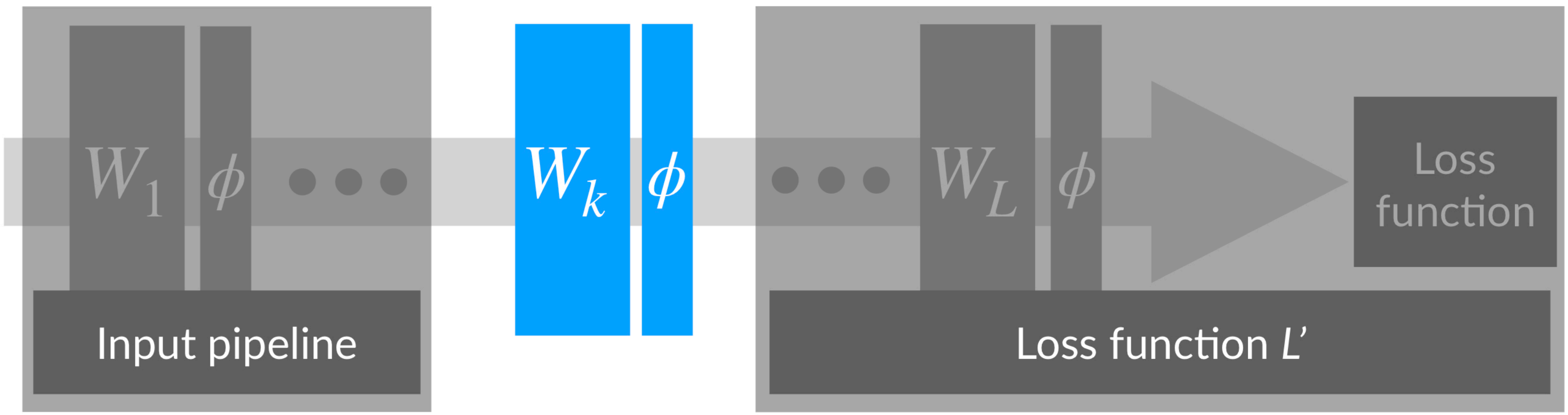
- We can look at the training from the perspective of a single layer – as if the other layers were constant



- If layers 1,…,k-1 are not constant (are trained), the distribution of input to layer k changes all the time
- We can't fix it, really!
  - We do want to early layers to learn, i.e., change what they're producing
- But we can fix some general property of what it produces

Batch normalization

- We can look at the training from the perspective of a single layer – as if the other layers were constant



- We can fix some general property of what it produces

- Normalization:
  - Make some statistic (e.g. mean) of the outputs of a layer constant, even if the actual output vectors change during training

Batch normalization

- Normalization:
  - Make some statistic (e.g. mean, or std.dev) of the outputs of a layer constant, even if the actual output vectors change during training
- Output of a layer with 6 neurons, on a batch of 4 samples
  - a 4x6 matrix WX

$$\begin{bmatrix} 1 & -3 & 4 & -2 & 5 & 4 \\ 2 & -6 & 9 & -1 & 8 & 2 \\ 2 & -6 & 9 & -1 & 9 & 7 \\ -1 & 3 & -4 & 2 & -5 & -4 \end{bmatrix}$$

- Possible options:
  - Normalize rows (separately each sample, across the neurons)
    - To add up to 1
    - Or to have 0 mean
  - Normalize columns (separately each neuron, across the batch):
    - To add up to 1
    - To have 0 mean

Batch normalization

- Normalization:
  - Make some statistic (e.g. mean, or std.dev) of the outputs of a layer constant, even if the actual output vectors change during training

- BatchNorm

  - Normalize columns (separately each neuron, across the batch):
    - Squares add up to 1, i.e. Std.Dev=1
    - Mean = 0

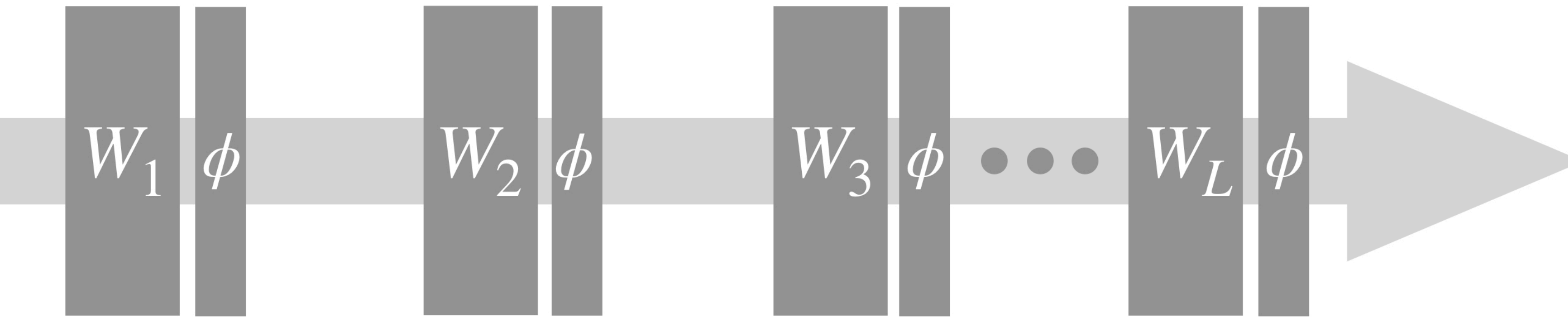
$$BN(y_j)^{(b)} = \gamma \cdot \left(\frac{y_j^{(b)} - \mu(y_j)}{\sigma(y_j)} \right) + \beta,$$

    - Add scale and offset parameters (so Mean is beta, Std.Dev. is gamma)

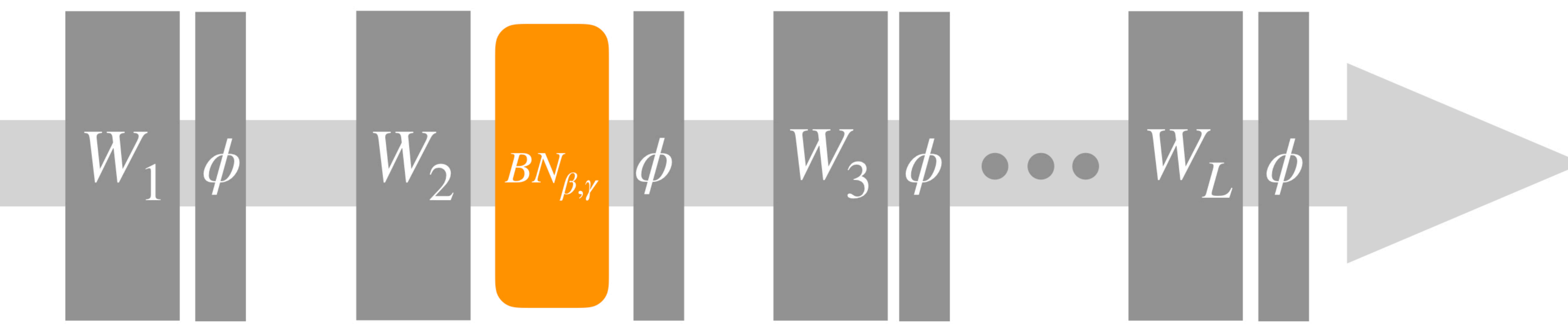
BatchNorm

$$BN(y_j)^{(b)} = \gamma \cdot \left(\frac{y_j^{(b)} - \mu(y_j)}{\sigma(y_j)} \right) + \beta$$

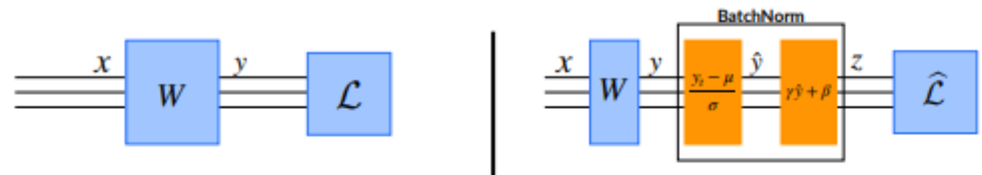
Standard Network



Adding a BatchNorm layer (between weights and activation function)



Batch normalization



(a) Vanilla Network

(b) Vanilla Network + BatchNorm Layer

- These are normal tensorflow / pytorch computations in the computational graph
  - i.e. chain rule applies to the operation of calculating mean/std.dev

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

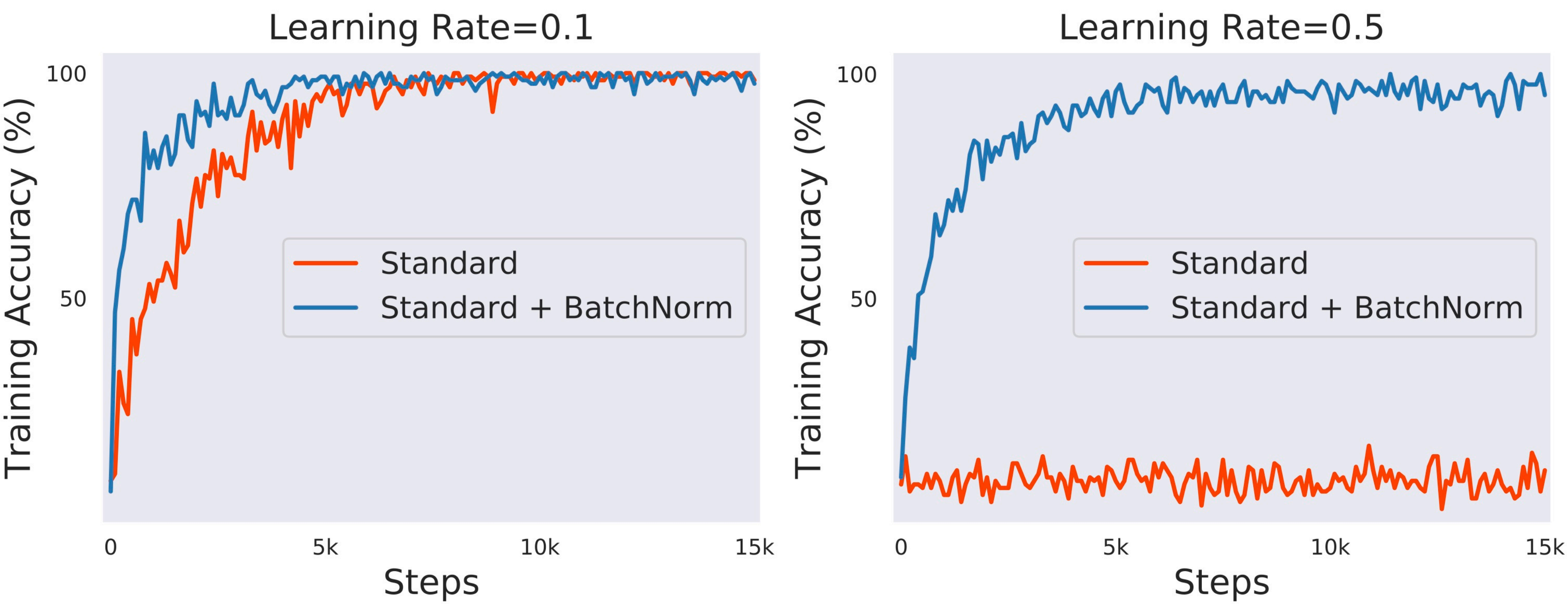
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

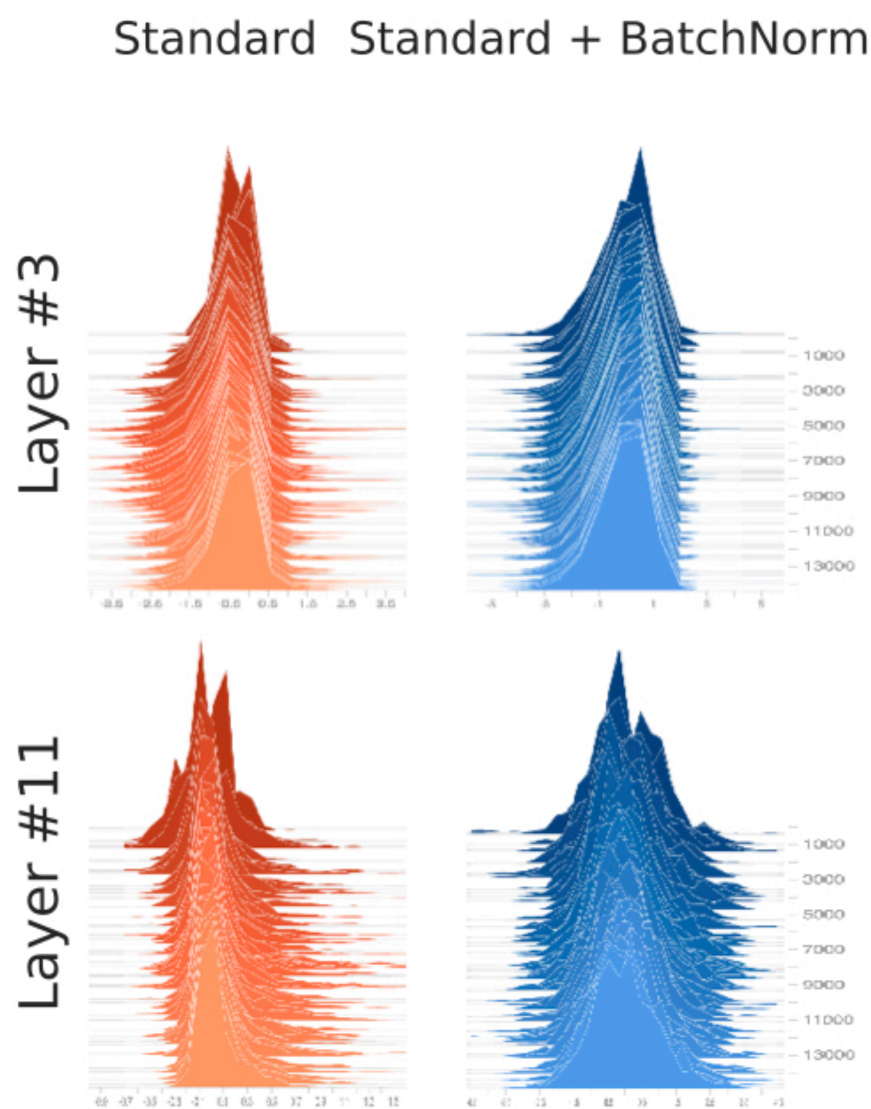
Batch normalization

- Does BatchNorm help?



Batch normalization

- How does BatchNorm help?



- BatchNorm is supposed to help with "internal covariate shift"
  - Normalize columns (separately each neuron, across the batch):
    - Squares add up to 1, i.e. Std.Dev=1
    - Mean = 0
- But even without BatchNorm we don't see much instability in the distributions

Batch normalization

- A three-layer neural network with ReLU activation is:
  - $Y = \mathrm{ReLU}(\ W_3\, \mathrm{ReLU}(\ W_2\, \mathrm{ReLU}(\ W_1 X\ )\ )\ )$
- Expanding ReLU(wx)=max(0,wx), we see either wx, or 0
  - $\mathrm{ReLU}(w_2 \mathrm{ReLu}(w_1 x))$ can be $w_2 * w_1 * x$
- Jointly over three layers, we see terms like $z = w_{3ij} * w_{2kl} * w_{1mn} * x$
- The derivative of z over w_{1mn} is $w_{3ij} * w_{2kl} * x$
- The derivative can quickly get large even if individual w's are not that much larger than 1
- Or can get very small if individual w's are close to zero

Batch normalization

- A neural network with ReLU activation is essentially:
  - $Y = ReLU(\ W_3\ ReLU(\ W_2\ ReLU(\ W_1 X\ )\ )\ )$

- We see terms like $w_{3ij} * w_{2kl} * w_{1mn} * x$

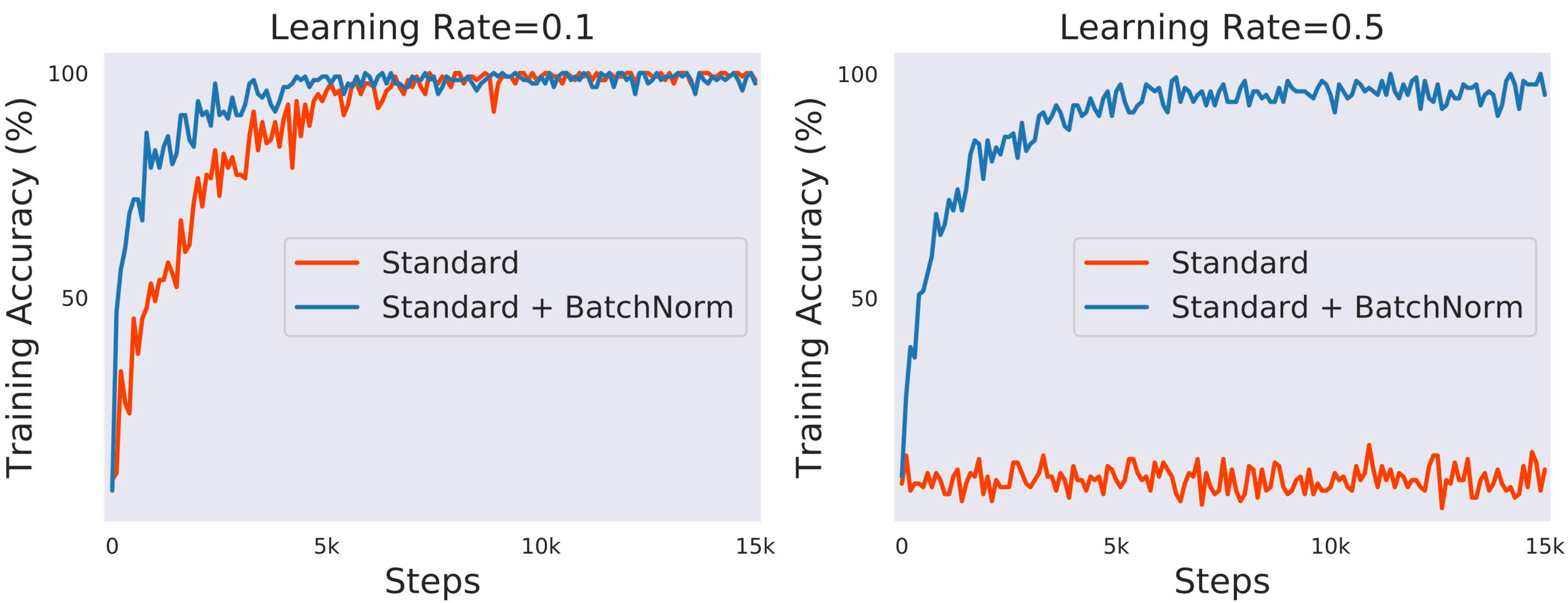
- The derivative over w_{2kl} is $w_{3ij} * w_{1mn} * x$

- The derivative can quickly get large even if individual w's are not that much larger than 1
- Or can get very small if individual w's are close to zero

- Vanishing/exploding gradient!

Batch normalization

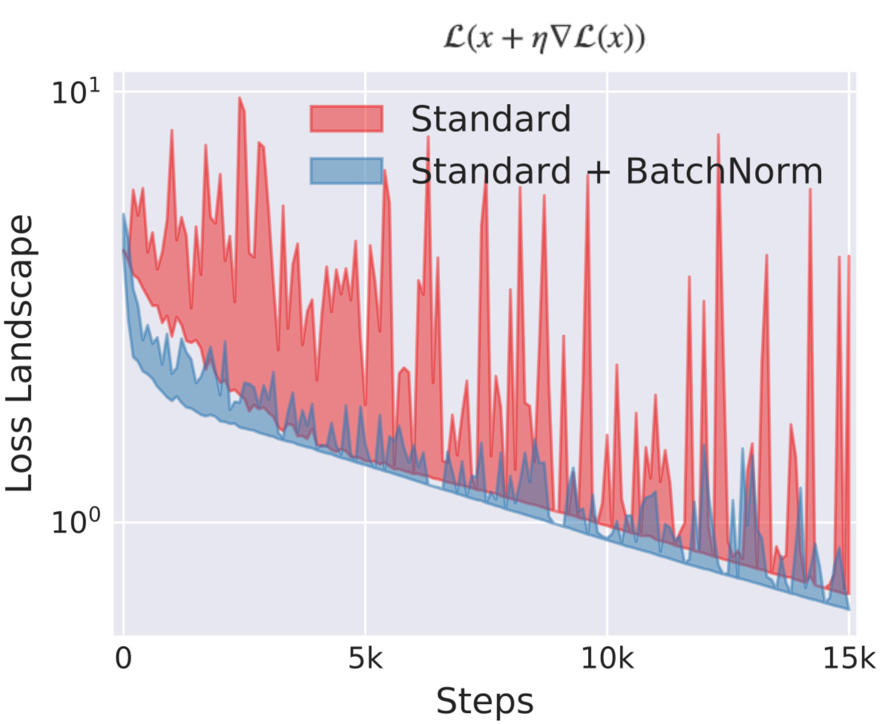
- Vanishing gradients slow learning
- Exploding gradients prevent learning

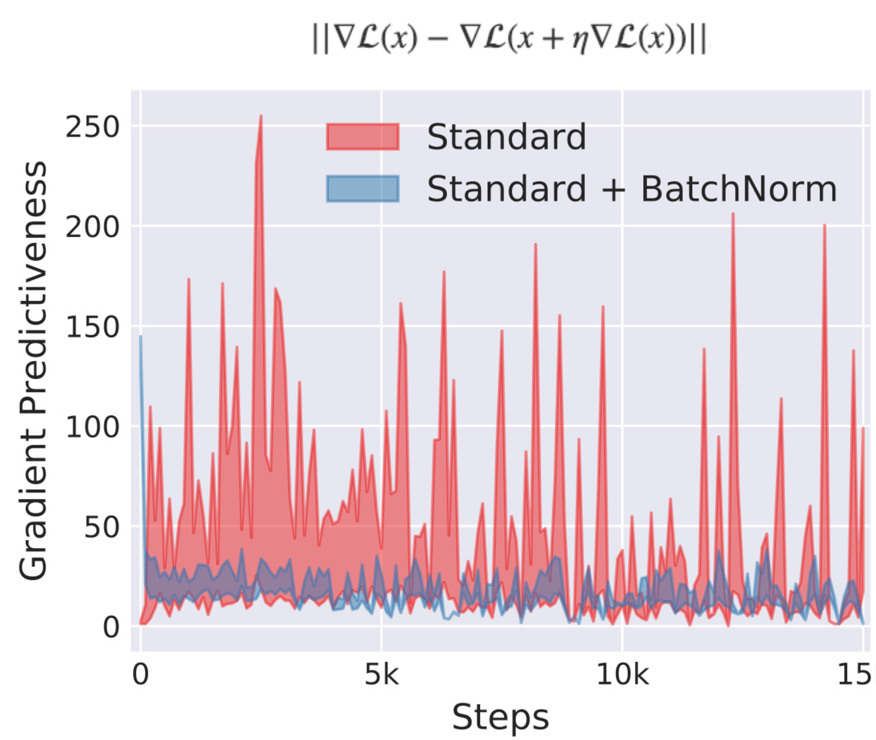


Batch normalization

- The derivative over w_{2kl} is w_{3ij}* w_{1mn} *x

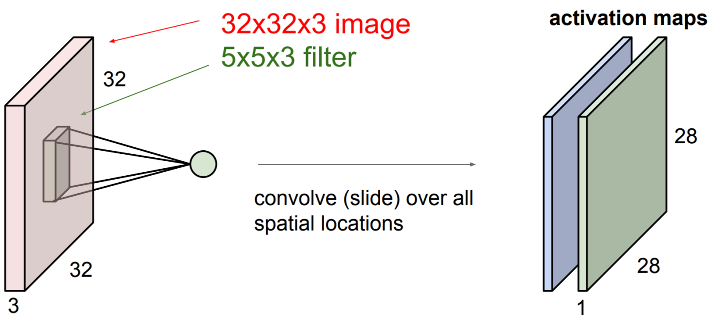
  ~ w_{3ij}*activation(prev.layer)
- Solution: make activations "just right"
  - E.g. make them roughly follow a Gaussian with 0-mean, unit norm
- Help keep loss stable:





Normalizations

- Possible normalization options:
  - Layer norm: Normalize rows (separately each sample, across the neurons)
  - Batch norm: Normalize columns (separately each neuron, across the batch):
- In ConvNets we also have a third axis (channels / "colors")
  - Like Layer norm, but within each channel separately



Multiple filters, each with its own set of weights