Lecture 11

Feature Normalization and Weight Initialization

STAT 479: Deep Learning, Spring 2019

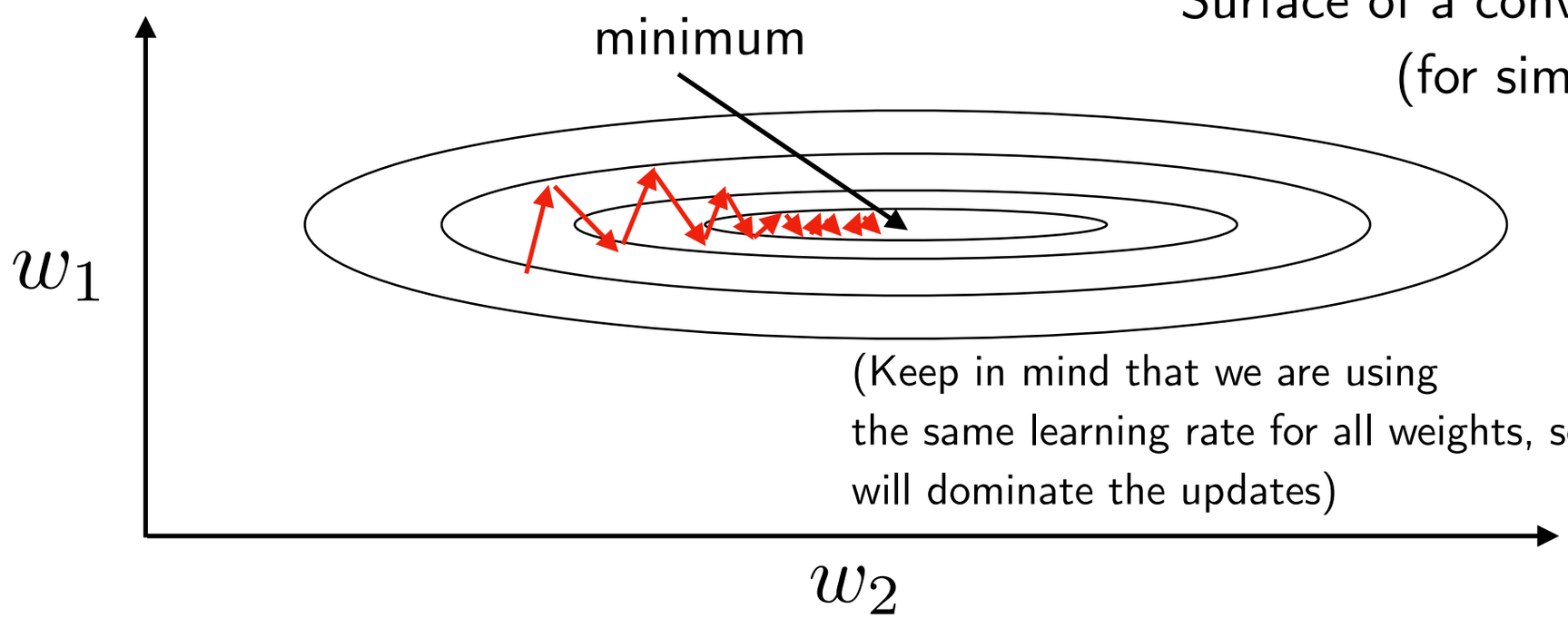
Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching/stat479-ss2019/

Overview: Additional Engineering Tricks for Neural Network Training

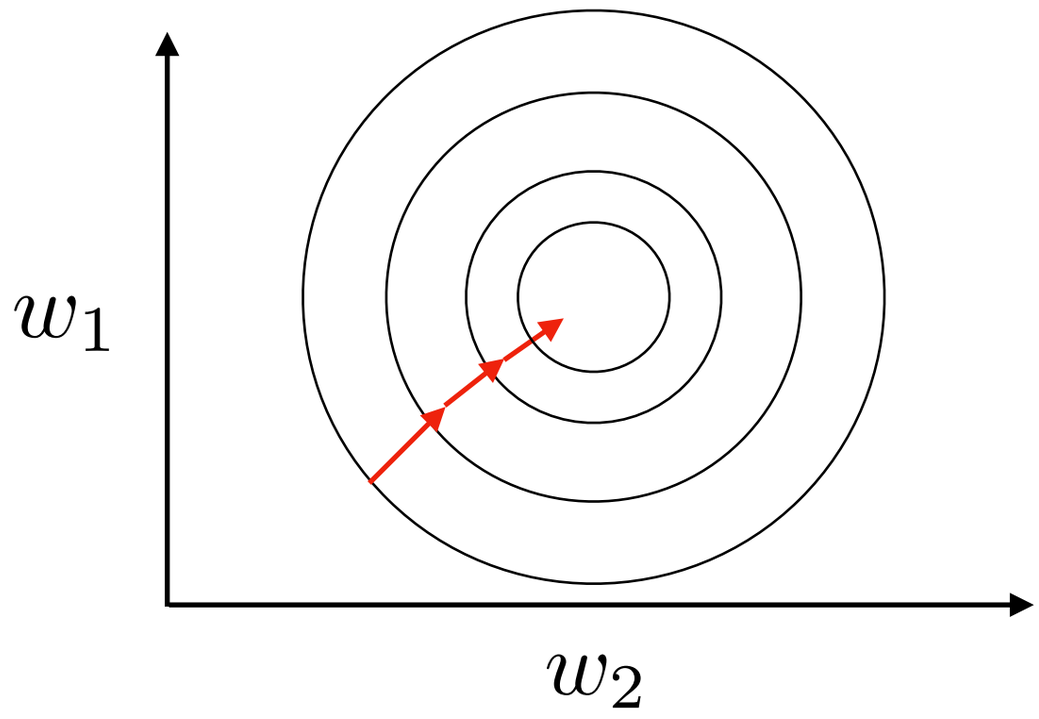
- Input Normalization (BatchNorm, InstanceNorm, GroupNorm, LayerNorm)
- Weight Initialization (Xavier, Kaiming He)
- Optimization Algorithms (RMSProp, Adagrad, ADAM)
  -- after Spring Break

Recap: Why We Normalize Inputs for Gradient Descent



Surface of a convex cost function
(for simplicity)

(Keep in mind that we are using
the same learning rate for all weights, so large parameters
will dominate the updates)



"Standardization" of input features

$$x_j'^{[i]} = \frac{x_j^{[i]} - \mu_j}{\sigma_j}$$

(scaled feature will have
zero mean, unit variance)

However, normalizing the inputs only affects the first hidden layer ... what about the other hidden layers?

Batch Normalization

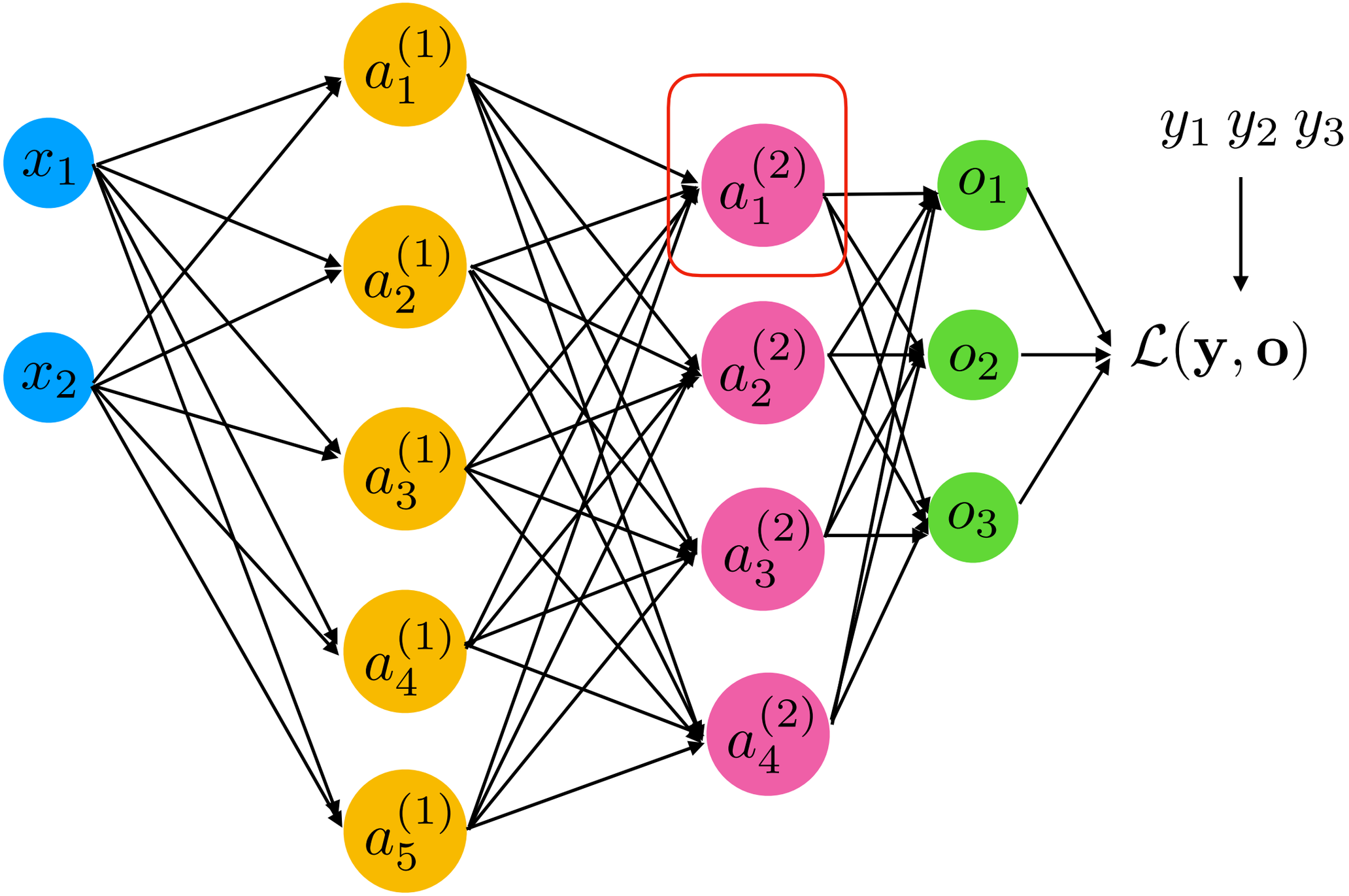
Ioffe, S., & Szegedy, C. (2015, June). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning* (pp. 448-456).

http://proceedings.mlr.press/v37/ioffe15.html

Batch Normalization

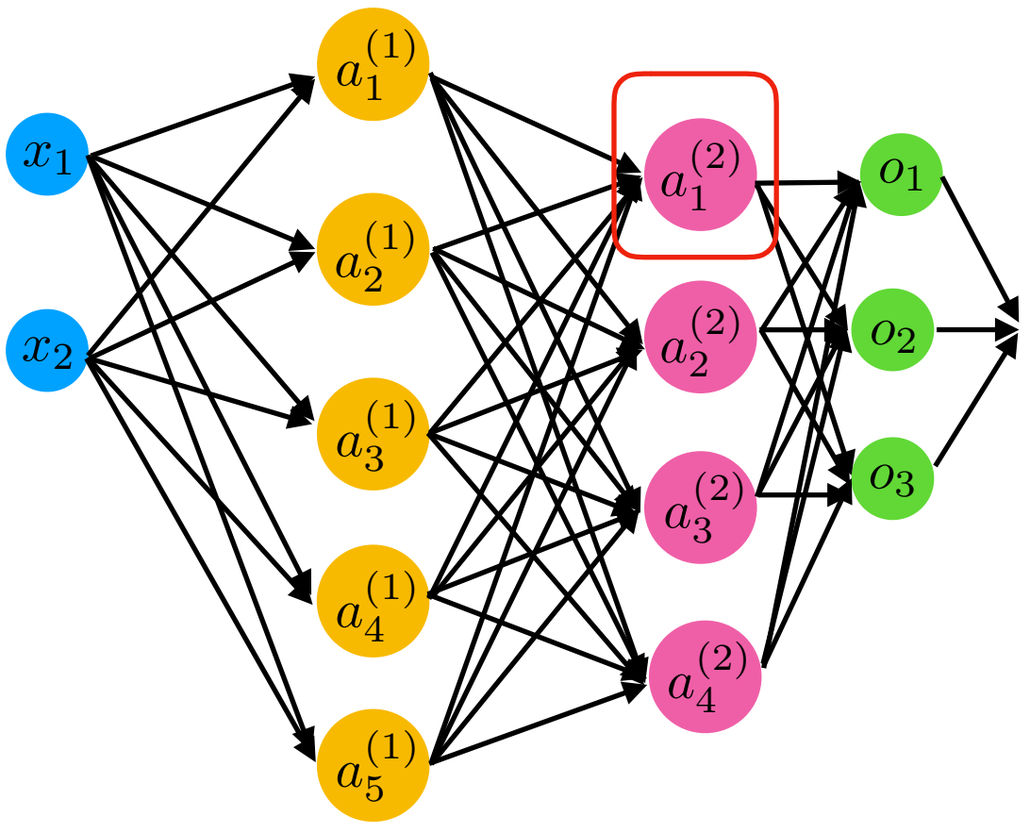
- Normalization of inputs for hidden layers
- Helps with exploding/vanishing gradient problems
- Can increase training stability and convergence rate
- Can be understood as additional normalization layers (with additional parameters)

Suppose, we have net input $z_1^{(2)}$
associated with an activation in the 2nd hidden layer



Now, consider all examples in a minibatch such that the net input of a given training example at layer 2 is written as $z_1^{(2)}[i]$

where $i \in \{1, ..., n\}$



In the next slides, let's omit the layer index, as it may be distracting...

BatchNorm Step 1: Normalize Net Inputs

$$\mu_j = \frac{1}{n}\sum_i z_j^{[i]}$$

$$\sigma_j^2 = \frac{1}{n}\sum_i (z_j^{[i]} - \mu_j)^2$$

$$z'^{[i]}_j = \frac{z_j^{[i]} - \mu_j}{\sigma_j}$$

BatchNorm Step 1: Normalize Net Inputs

$$\mu_j = \frac{1}{n}\sum_i z_j^{[i]}$$

$$\sigma_j^2 = \frac{1}{n}\sum_i (z_j^{[i]} - \mu_j)^2$$

$$z'^{[i]}_j = \frac{z_j^{[i]} - \mu_j}{\sigma_j}$$

In practice:

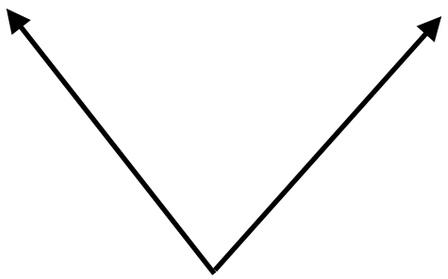
$$z'^{[i]}_j = \frac{z_j^{[i]} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

For numerical stability, where epsilon is a small number like 1E-5

BatchNorm Step 2: Pre-Activation Scaling

$$z'^{[i]}_j = \frac{z^{[i]}_j - \mu_j}{\sqrt{\sigma^2_j + \epsilon}}$$

$$a'^{[i]}_j = \gamma_j \cdot z'^{[i]}_j + \beta_j$$



These are learnable parameters

BatchNorm Step 2: Pre-Activation Scaling

$$z'^{[i]}_j = \frac{z^{[i]}_j - \mu_j}{\sigma_j}$$

$$a'^{[i]}_j = \gamma_j \cdot z'^{[i]}_j + \beta_j$$

Controls the spread or scale

Controls the mean

BatchNorm Step 2: Pre-Activation Scaling

$$z'^{[i]}_j = \frac{z^{[i]}_j - \mu_j}{\sigma_j}$$

$$a'^{[i]}_j = \gamma_j \cdot z'^{[i]}_j + \beta_j$$

Controls the spread or scale

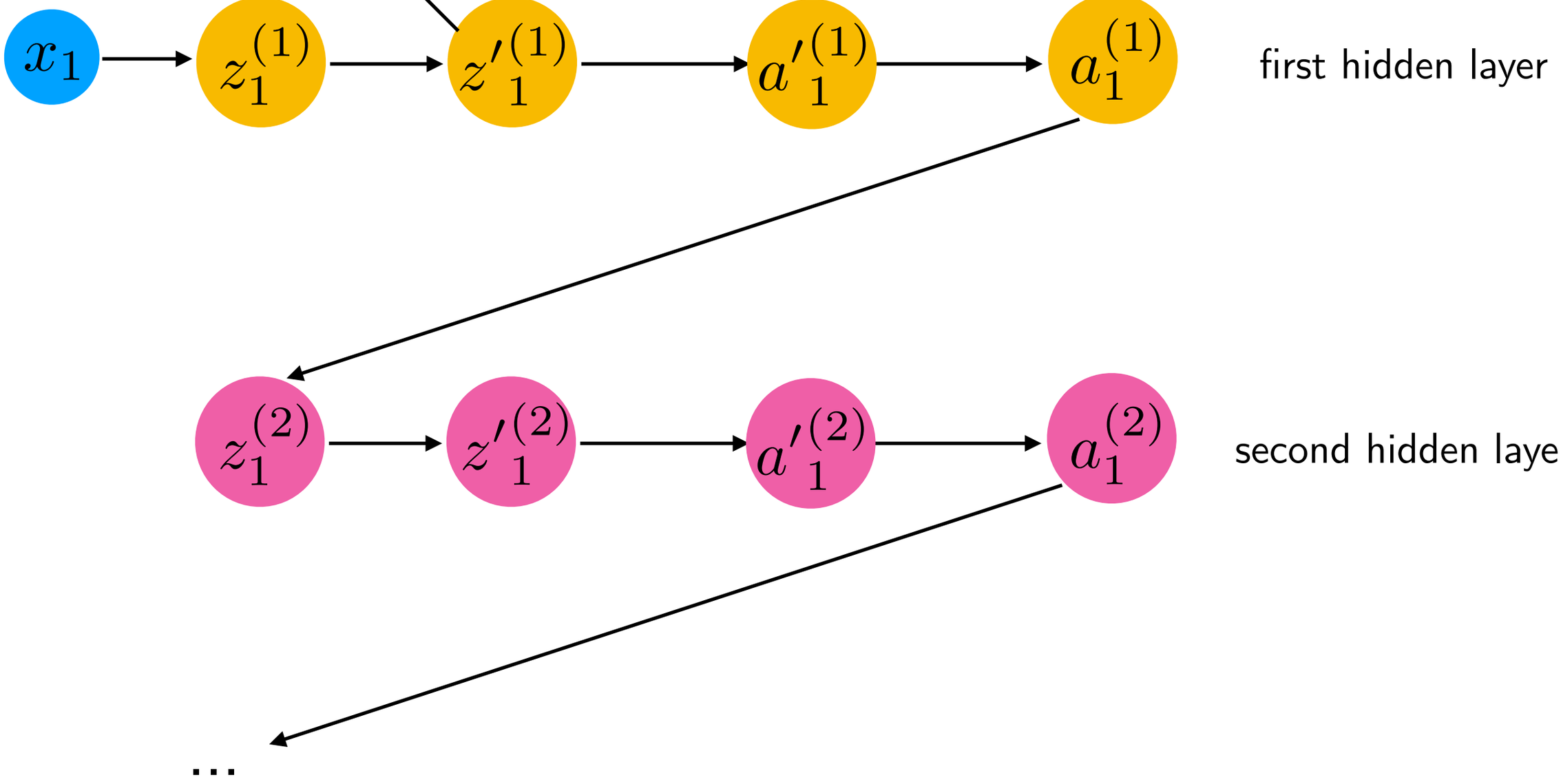
Controls the mean

Technically, a BatchNorm layer could learn to perform "standardization" with zero mean and unit variance

BatchNorm Step 1 & 2 Summarized

$$z'^{[i]}_j = \frac{z^{[i]}_j - \mu_j}{\sigma_j}$$

$$a'^{[i]}_j = \gamma_j \cdot z'^{[i]}_j + \beta_j$$



BatchNorm -- Additional Things to Consider

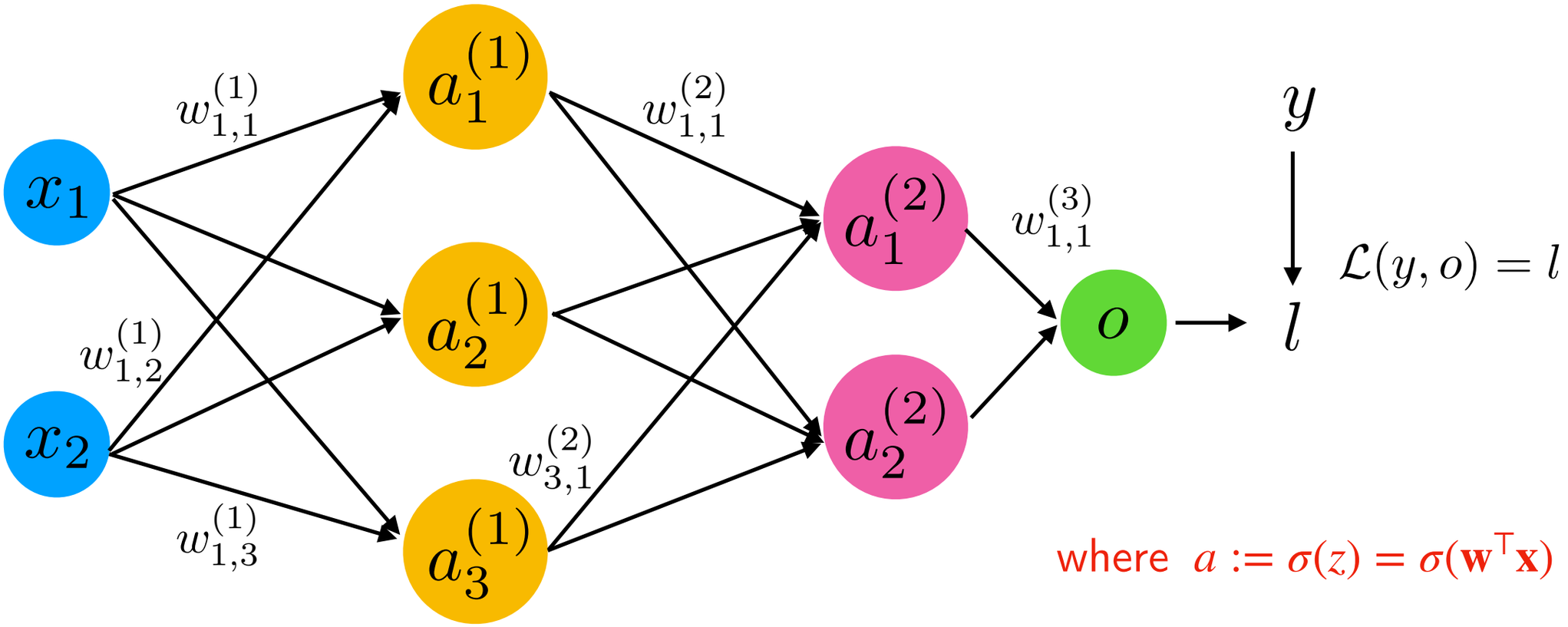
$$a'^{[i]}_j = \gamma_j \cdot z'^{[i]}_j + \beta_j$$

This parameter makes the bias units redundant

Also, note that the batchnorm parameters are vectors with the same number of elements as the bias vector

Backpropagation for BatchNorm Parameters

Reminder: Multilayer Perceptron (from lecture 9)

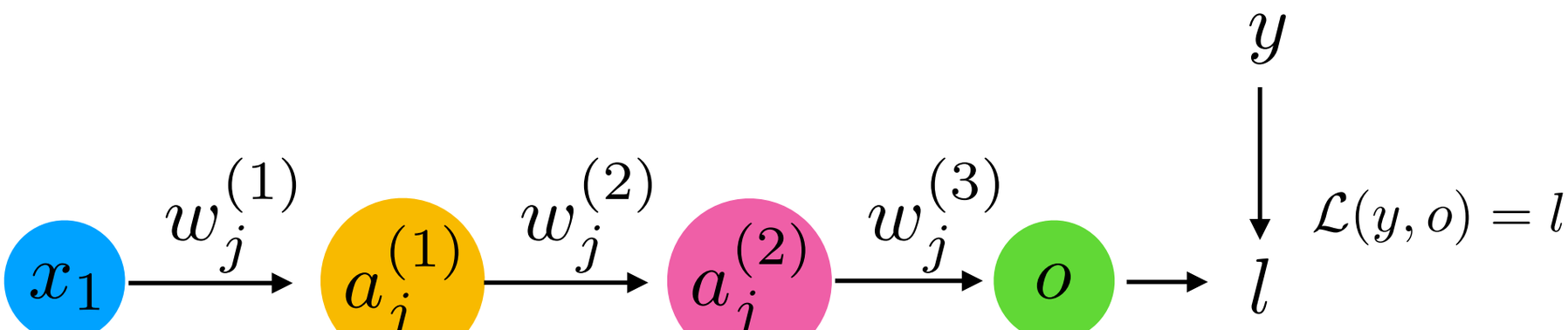


where $a := \sigma(z) = \sigma(\mathbf{w}^\top \mathbf{x})$

$$\frac{\partial l}{\partial w_{1,1}^{(1)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}} + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}$$

(Assume network for binary classification)

Let's consider a simpler case ...

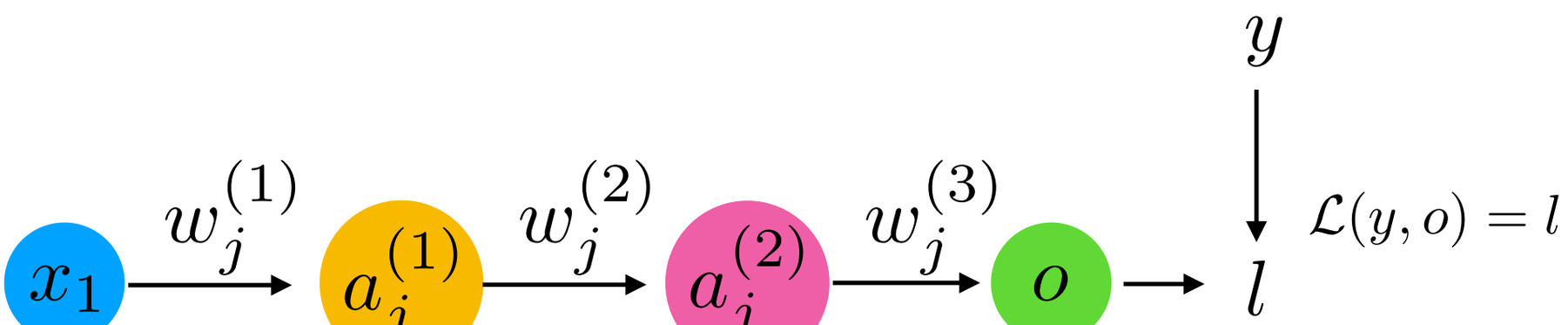


$$\frac{\partial l}{\partial w_j^{(3)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial w_j^{(3)}}$$

$$\frac{\partial l}{\partial w_j^{(2)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_j^{(2)}} \cdot \frac{\partial a_j^{(2)}}{\partial w_j^{(2)}}$$

$$\frac{\partial l}{\partial w_j^{(1)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_j^{(2)}} \cdot \frac{\partial a_j^{(2)}}{\partial a_j^{(1)}} \cdot \frac{\partial a_j^{(1)}}{\partial w_j^{(1)}}$$

Same as on previous slide, but more verbose ...

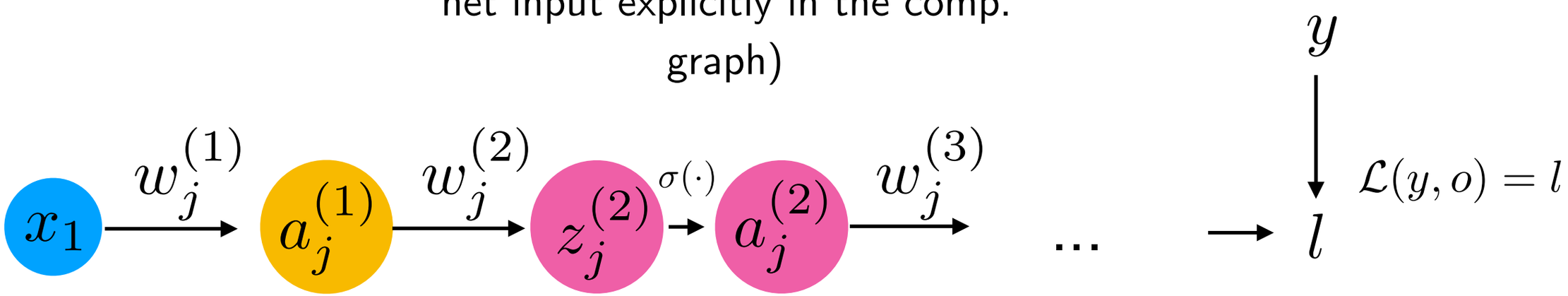


$$\frac{\partial l}{\partial w_j^{(3)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial w_j^{(3)}} \longrightarrow \frac{\partial l}{\partial w_j^{(3)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial z_j^{(3)}} \cdot \frac{\partial z_j^{(3)}}{\partial w_j^{(3)}}$$

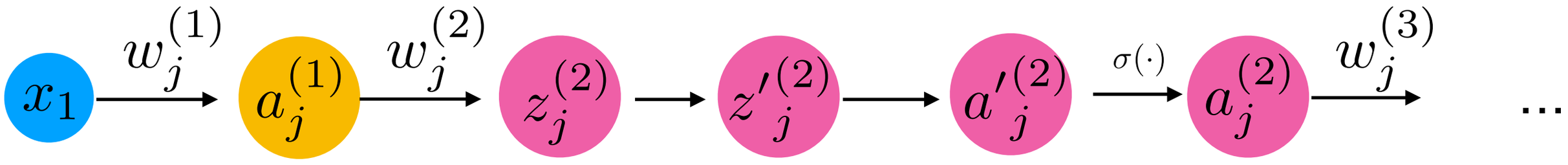
$$\frac{\partial l}{\partial w_j^{(2)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_j^{(2)}} \cdot \frac{\partial a_j^{(2)}}{\partial w_j^{(2)}} \longrightarrow \dots$$

$$\frac{\partial l}{\partial w_j^{(1)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_j^{(2)}} \cdot \frac{\partial a_j^{(2)}}{\partial a_j^{(1)}} \cdot \frac{\partial a_j^{(1)}}{\partial w_j^{(1)}} \longrightarrow \dots$$

(previously, we didn't write the net input explicitly in the comp. graph)



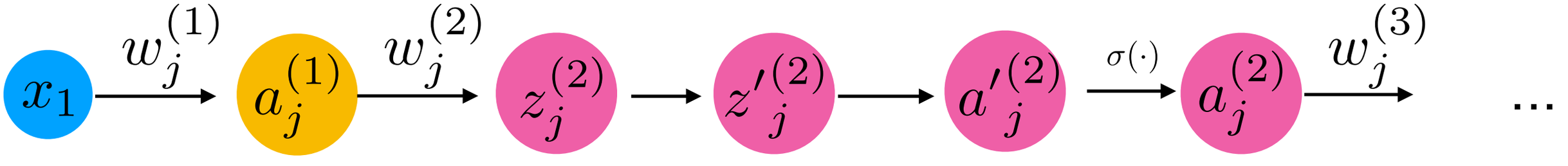
Adding a BatchNorm layer ...



$$z'^{(2)}_j = \frac{z_j^{(2)[i]} - \mu_j}{\sigma_j} \qquad a'^{(2)}_j = \gamma_j \cdot z'^{(2)}_j + \beta_j$$

Backprop for BatchNorm Parameters

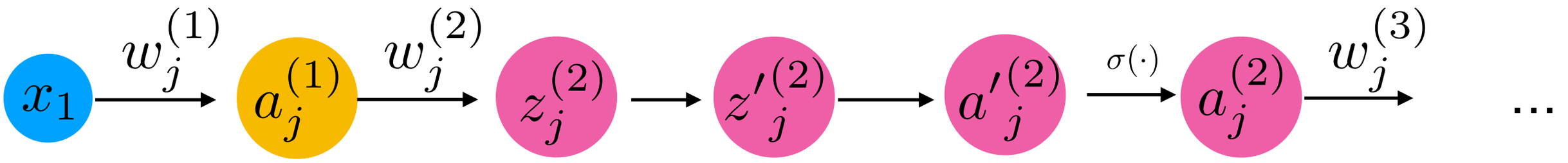
$$z'^{(2)}_j = \frac{z^{(2)[i]}_j - \mu_j}{\sigma_j} \qquad a'^{(2)}_j = \gamma_j \cdot z'^{(2)}_j + \beta_j$$



$$\frac{\partial l}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial l}{\partial a'^{(2)[i]}_j} \cdot \frac{\partial a'^{(2)[i]}_j}{\partial \beta_j} = \sum_{i=1}^{n} \frac{\partial l}{\partial a'^{(2)[i]}_j}$$

$$\frac{\partial l}{\partial \gamma_j} = \sum_{i=1}^{n} \frac{\partial l}{\partial a'^{(2)[i]}_j} \cdot \frac{\partial a'^{(2)[i]}_j}{\partial \gamma_j} = \sum_{i=1}^{n} \frac{\partial l}{\partial a'^{(2)[i]}_j} \cdot z'^{(2)[i]}_j$$

Backprop Beyond the BatchNorm Layer



Since the minibatch mean and variance act as parameters, we can/have to apply the multivariable chain rule

$$\frac{\partial l}{\partial z_j^{(2)[i]}} = \frac{\partial l}{\partial z'^{(2)[i]}_j} \cdot \frac{\partial z'^{(2)[i]}_j}{\partial z_j^{(2)[i]}} + \frac{\partial l}{\partial \mu_j} \cdot \frac{\partial \mu_j}{\partial z_j^{(2)[i]}} + \frac{\partial l}{\partial \sigma_j^2} \cdot \frac{\partial \sigma_j^2}{\partial z_j^{(2)[i]}}$$

$$= \frac{\partial l}{\partial z'^{(2)[i]}_j} \cdot \frac{1}{\sigma_j} + \frac{\partial l}{\partial \mu_j} \cdot \frac{1}{n} + \frac{\partial l}{\partial \sigma_j^2} \cdot \frac{1(z_j^{(2)} - \mu_j)}{n}$$

Backprop for BatchNorm Parameters

$$\frac{\partial l}{\partial z_j^{(2)[i]}} = \frac{\partial l}{\partial z'^{(2)[i]}_j} \cdot \frac{\partial z'^{(2)[i]}_j}{\partial z_j^{(2)[i]}} + \frac{\partial l}{\partial \mu_j} \cdot \frac{\partial \mu_j}{\partial z_j^{(2)[i]}} + \frac{\partial l}{\partial \sigma_j^2} \cdot \frac{\partial \sigma_j^2}{\partial z_j^{(2)[i]}}$$

$$= \boxed{\frac{\partial l}{\partial z'^{(2)[i]}_j}} \cdot \frac{1}{\sigma_j} + \boxed{\frac{\partial l}{\partial \mu_j}} \cdot \frac{1}{n} + \boxed{\frac{\partial l}{\partial \sigma_j^2}} \cdot \frac{1(z_j^{(2)} - \mu_j)}{n}$$

If you like engineering math, you can solve the remaining terms as an ungraded HW exercise ;)

BatchNorm in PyTorch

```
class MultilayerPerceptron(torch.nn.Module):

    def __init__(self, num_features, num_classes):
        super(MultilayerPerceptron, self).__init__()

        ### 1st hidden layer
        self.linear_1 = torch.nn.Linear(num_features, num_hidden_1)
        self.linear_1_bn = torch.nn.BatchNorm1d(num_hidden_1)

        ### 2nd hidden layer
        self.linear_2 = torch.nn.Linear(num_hidden_1, num_hidden_2)
        self.linear_2_bn = torch.nn.BatchNorm1d(num_hidden_2)

        ### Output layer
        self.linear_out = torch.nn.Linear(num_hidden_2, num_classes)
```

```
    def forward(self, x):
        out = self.linear_1(x)
        # note that batchnorm is in the classic
        # sense placed before the activation
        out = self.linear_1_bn(out)
        out = F.relu(out)

        out = self.linear_2(out)
        out = self.linear_2_bn(out)
        out = F.relu(out)

        logits = self.linear_out(out)
        probas = F.softmax(logits, dim=1)
        return logits, probas
```

BatchNorm in PyTorch

```
class MultilayerPerceptron(torch.nn.Module):

    def __init__(self, num_features, num_classes):
        super(MultilayerPerceptron, self).__init__()

        ### 1st hidden layer
        self.linear_1 = torch.nn.Linear(num_features, num_hidden_1)
        self.linear_1_bn = torch.nn.BatchNorm1d(num_hidden_1)

        ### 2nd hidden layer
        self.linear_2 = torch.nn.Linear(num_hidden_1, num_hidden_2)
        self.linear_2_bn = torch.nn.BatchNorm1d(num_hidden_2)

        ### Output layer
        self.linear_out = torch.nn.Linear(num_hidden_2, num_classes)
```

```
    def forward(self, x):
        out = self.linear_1(x)
        # note that batchnorm is in the classic
        # sense placed before the activation
        out = self.linear_1_bn(out)
        out = F.relu(out)

        out = self.linear_2(out)
        out = self.linear_2_bn(out)
        out = F.relu(out)

        logits = self.linear_out(out)
        probas = F.softmax(logits, dim=1)
        return logits, probas
```

don't forget `model.train()`
and `model.eval()`
in training and test loops

BatchNorm During Prediction ("Inference")

- Use exponentially weighted average (moving average) of mean and variance

```
running_mean = momentum * running_mean
               + (1 - momentum) * sample_mean
```

(where momentum is typically ~0.1; and same for variance)

- Alternatively, can also use global training set mean and variance

BatchNorm and Internal Covariate Shift

Ioffe, S., & Szegedy, C. (2015, June). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning* (pp. 448-456).

http://proceedings.mlr.press/v37/ioffe15.html

Internal Covariate Shift is jargon for saying that the layer input distribution changes ("feature shift" in hidden layers)

But there is actually no guarantee or evidence
that BatchNorm helps with covariate shift

In my opinion, BatchNorm just provides additional parameters that will help layers to learn a little bit more independently

How Does Batch Normalization Help Optimization?

Shibani Santurkar[*]
MIT
shibani@mit.edu

Dimitris Tsipras[*]
MIT
tsipras@mit.edu

Andrew Ilyas[*]
MIT
ailyas@mit.edu

Aleksander Mądry
MIT
madry@mit.edu

Abstract

Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

BatchNorm Enables Faster Convergence By Allowing Larger Learning Rates

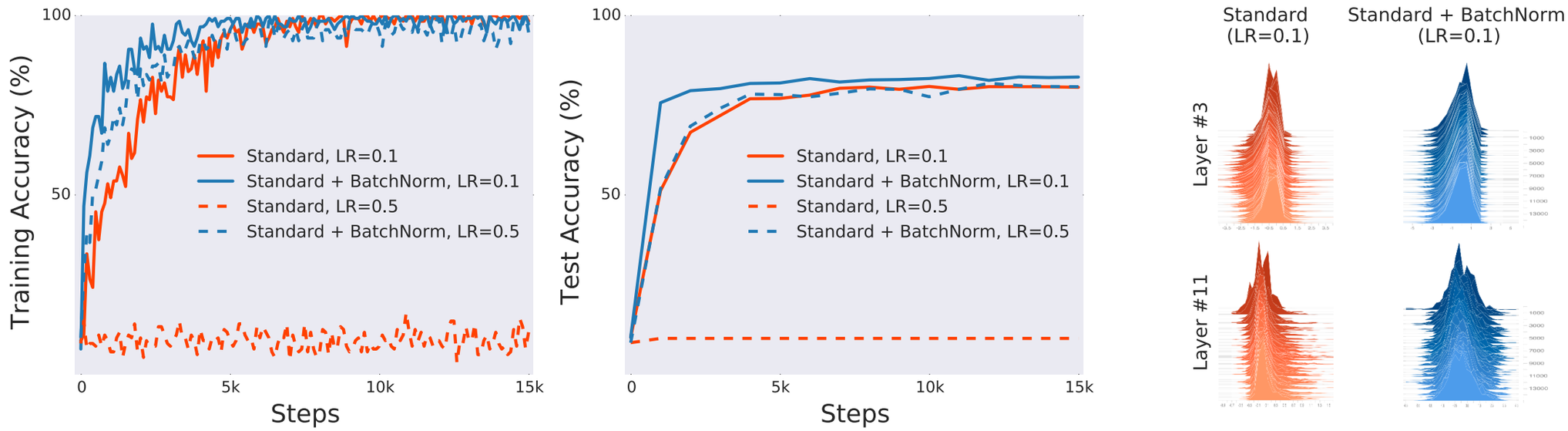


Figure 1: Comparison of (a) training (optimization) and (b) test (generalization) performance of a standard VGG network trained on CIFAR-10 with and without BatchNorm (details in Appendix A). There is a consistent gain in training speed in models with BatchNorm layers. (c) Even though the gap between the performance of the BatchNorm and non-BatchNorm networks is clear, the difference in the evolution of layer input distributions seems to be much less pronounced. (Here, we sampled activations of a given layer and visualized their distribution over training steps.)

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In *Advances in Neural Information Processing Systems* (pp. 2488-2498).

Good Performance of BatchNorm Seems Unrelated to Covariate Shift Prevention

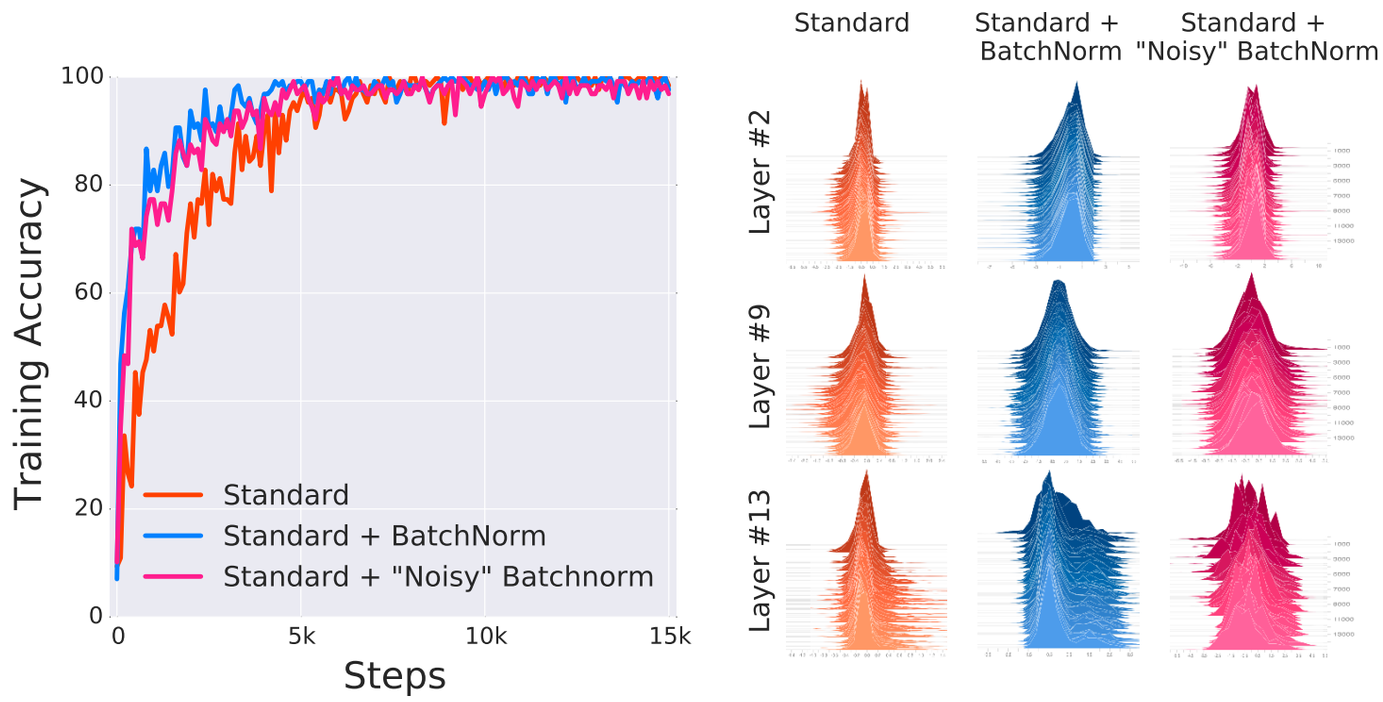


Figure 2: Connections between distributional stability and BatchNorm performance: We compare VGG networks trained without BatchNorm (Standard), with BatchNorm (Standard + BatchNorm) and with explicit "covariate shift" added to BatchNorm layers (Standard + "Noisy" BatchNorm). In the later case, we induce distributional instability by adding *time-varying*, *non-zero* mean and *non-unit* variance noise independently to each batch normalized activation. The "noisy" BatchNorm model nearly matches the performance of standard BatchNorm model, despite complete distributional instability. We sampled activations of a given layer and visualized their distributions (also cf. Figure 7).

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In *Advances in Neural Information Processing Systems* (pp. 2488-2498).

BatchNorm Variants

Pre-Activation

"Original" version
as discussed in
previous slides

compute net inputs

↓

BatchNorm

↓

apply activation function

↓

compute next-layer net inputs

Post-Activation

May make more sense,
but less common

compute net inputs

↓

apply activation function

↓

BatchNorm

↓

compute next-layer net inputs

Some Benchmarks

https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md#bn----before-or-after-relu

BN -- before or after ReLU?

Name	Accuracy	LogLoss	Comments
Before	0.474	2.35	As in paper
Before + scale&bias layer	0.478	2.33	As in paper
After	**0.499**	**2.21**	
After + scale&bias layer	0.493	2.24	

Some Benchmarks

https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md#bn----before-or-after-relu

BN and activations

Name	Accuracy	LogLoss	Comments
ReLU	0.499	2.21	
RReLU	0.500	2.20	
PReLU	**0.503**	**2.19**	
ELU	0.498	2.23	
Maxout	0.487	2.28	
Sigmoid	0.475	2.35	
TanH	0.448	2.50	
No	0.384	2.96	

Some Benchmarks

https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md#bn----before-or-after-relu

BN and dropout

ReLU non-linearity, fc6 and fc7 layer only

Name	Accuracy	LogLoss	Comments
Dropout = 0.5	0.499	2.21	
Dropout = 0.2	**0.527**	**2.09**	
Dropout = 0	0.513	2.19	

Practical Consideration

BatchNorm become more stable with larger minibatch sizes

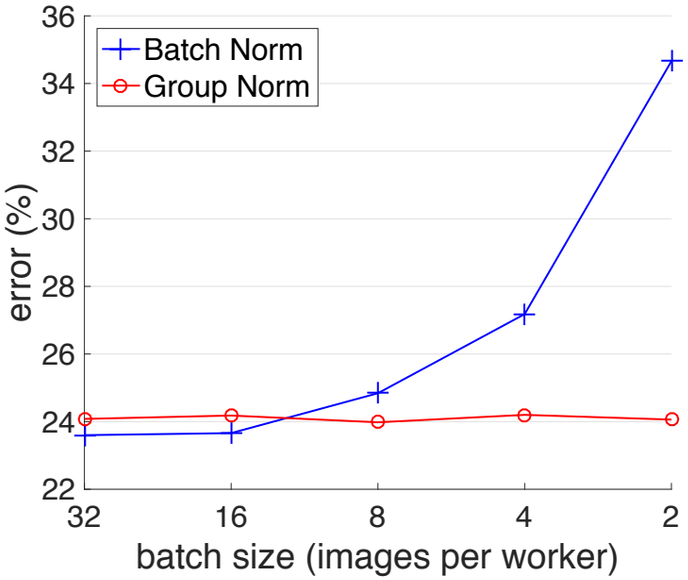


Figure 1. ImageNet classification error *vs*. batch sizes. The model is ResNet-50 trained in the ImageNet training set using 8 workers (GPUs) and evaluated in the validation set. BN's error increases rapidly when reducing the batch size. GN's computation is independent of batch sizes, and its error rate is stable despite the batch size changes. GN has substantially lower error (by 10%) than BN with a batch size of 2.

Wu, Y., & He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 3-19).

Other Normalization Methods for Hidden Activations

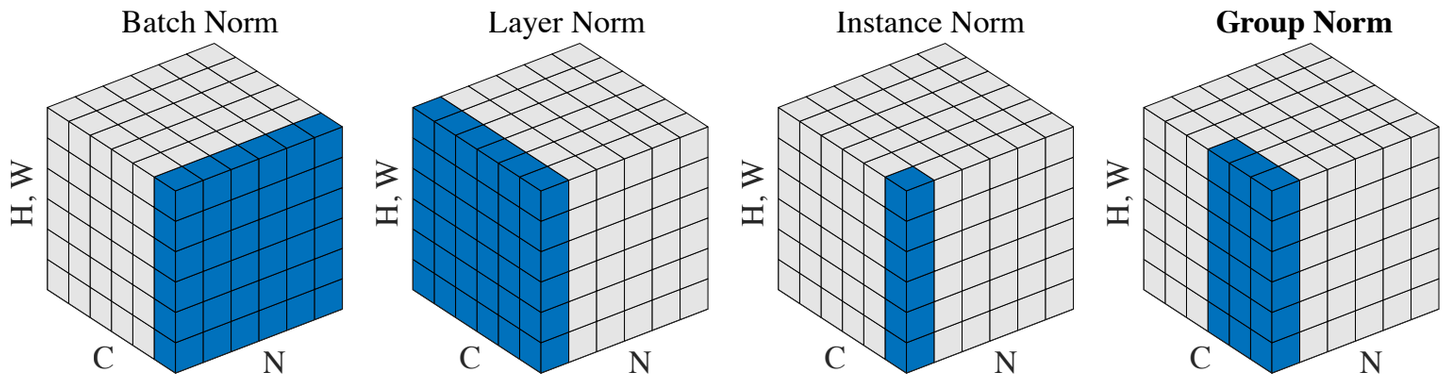


Figure 2. Normalization methods. Each subplot shows a feature map tensor. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels. Group Norm is illustrated using a group number of 2.

Wu, Y., & He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 3-19).

(will revisit after Spring Break after introducing Convolutional Neural Networks)

Reading Assignments (Optional)

Ioffe, S., & Szegedy, C. (2015, June). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning* (pp. 448-456).

http://proceedings.mlr.press/v37/ioffe15.html

Weight Initialization slides will be added during spring break and covered after spring break ...

Note that if BatchNorm is used,

initial feature weight choice is less important