Master's degree ICT Internet Multimedia Engineering

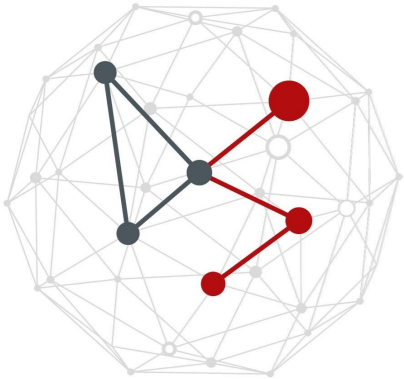
ADVANCED ARCHITECTURES: BATCH NORMALIZATION & RESIDUAL NETWORKS

Michele Rossi

michele.rossi@unipd.it

Dept. of Information Engineering

University of Padova, IT



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO MATEMATICA

800 ANNI 1222 · 2022

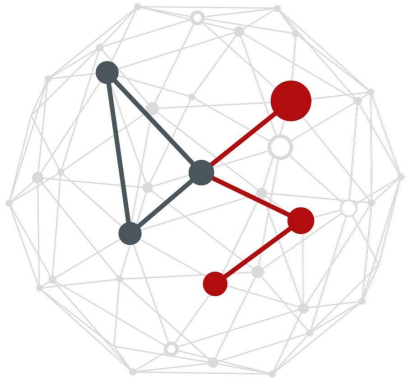
UNIVERSITÀ DEGLI STUDI DI PADOVA

Outline

- Batch normalization
  - Covariance shift – intuition
  - Covariance shift in a deep network
  - Batch normalization layer
  - Backpropagating the error gradient
- Residual networks (ResNets)
  - Evolution of Deep Networks
  - CIFAR-10 and ImageNet datasets
  - Top-x score and error metrics
  - Residual layers (skip connections) & their evolution
  - Experimental results for deep networks (1k layers)

MIME
Master's degree ICT Internet Multimedia Engineering

BATCH NORMALIZATION



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DIPARTIMENTO MATEMATICA
800 ANNI 1222 · 2022
UNIVERSITÀ DEGLI STUDI DI PADOVA

Bibliography

[Ioffe2015] Sergey Ioffe, Christian Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” Proceedings of the 32nd Int. Conf. on Machine Learning (ICML), Lille, France, July 2015. (from Google)

A sort of “re-engineering” of [Ioffe2015]:

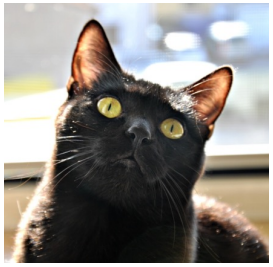
[Wu2018] Yuxin Wu, Kaiming He, “Group Normalization,”

“Deep Residual Learning for Image Recognition,” European Conference on Computer Vision (ECCV), Munich, Germany, 2018.

The input covariance shift problem

- Imagine that we have a shallow network (single layer)
- We train it on the popular cat recognition example
  - The task:
    - recognize images containing a cat (Y=1) *vs* other image contents (Y=0)
    - we perform a first training with images containing **black cats**

Y=1

Y=0




The input covariance shift problem

- The neural network after the learning phase will learn a first (nonlinear) mapping or decision function
- Now, if we test this network with images of cats having any color, we do not expect the network to do a god job

Y=?

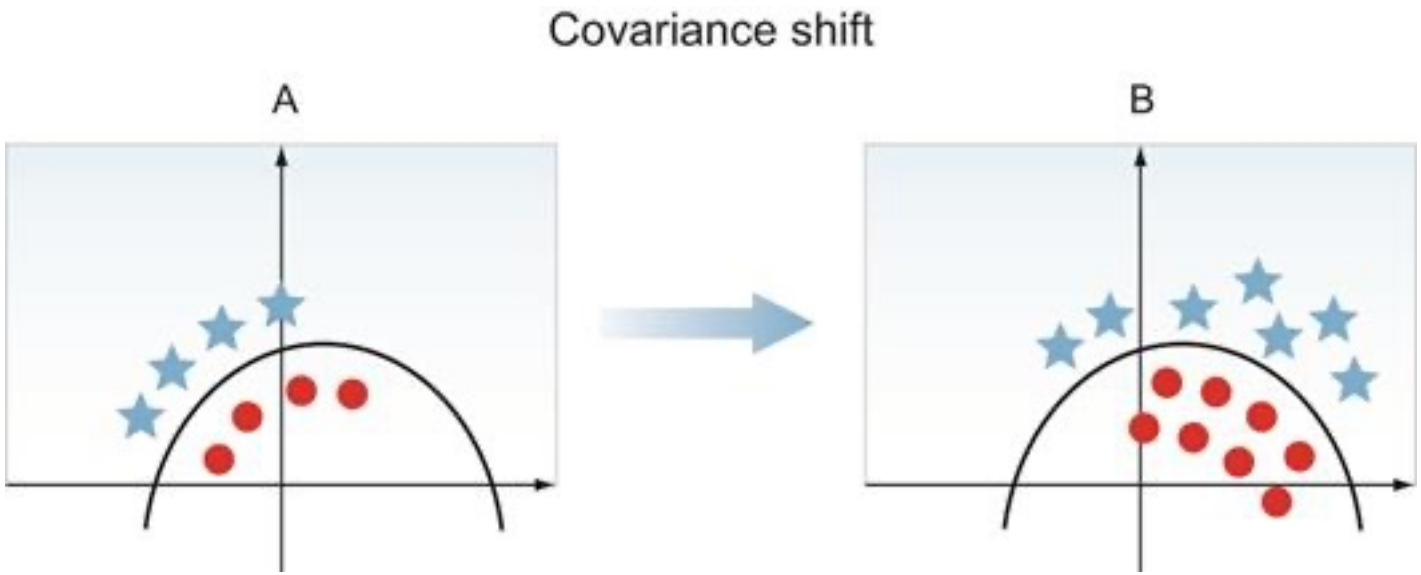



Y=0




The input covariance shift problem

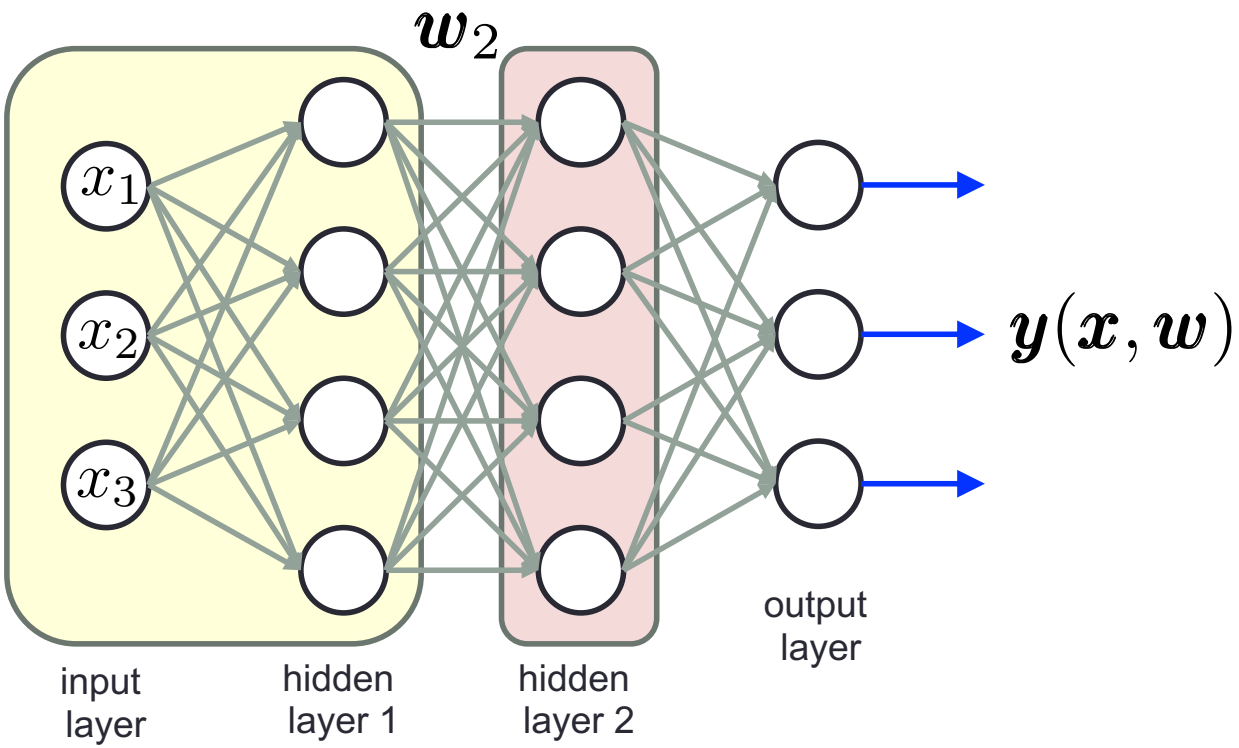
- Reason
  - The distribution of the input signal **x** changed → the previously computed weights are no longer expected to do well and we may need to retrain the learning algorithm



The learned decision boundary may or may not do well with the new data
Even though the function to be learned is unchanged

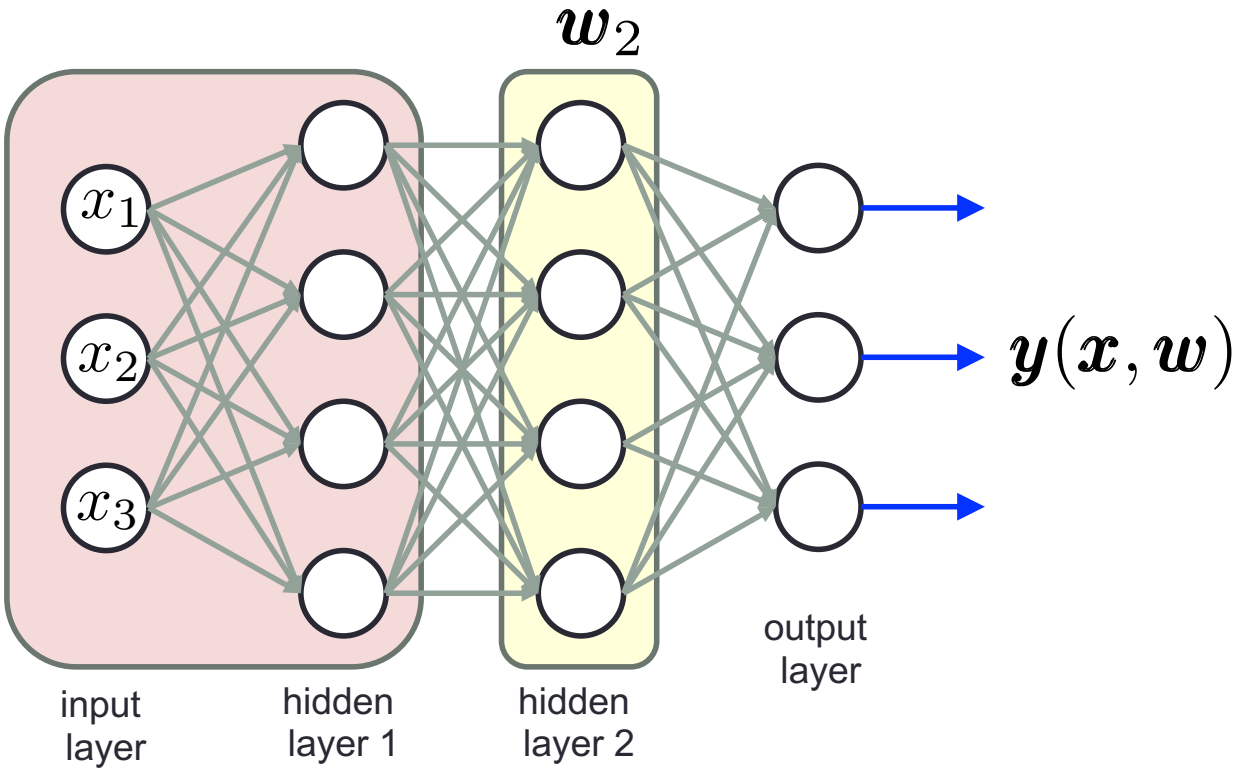
Covariance shift in a neural network

- This same problem holds true in a deep neural network
- Look at the second hidden layer (HL2) → it has learned params $\mathbf{w}_2$
- It gets values from previous layers: these are *features* and this hidden layer needs to find a way to map them into the final desired output **y**



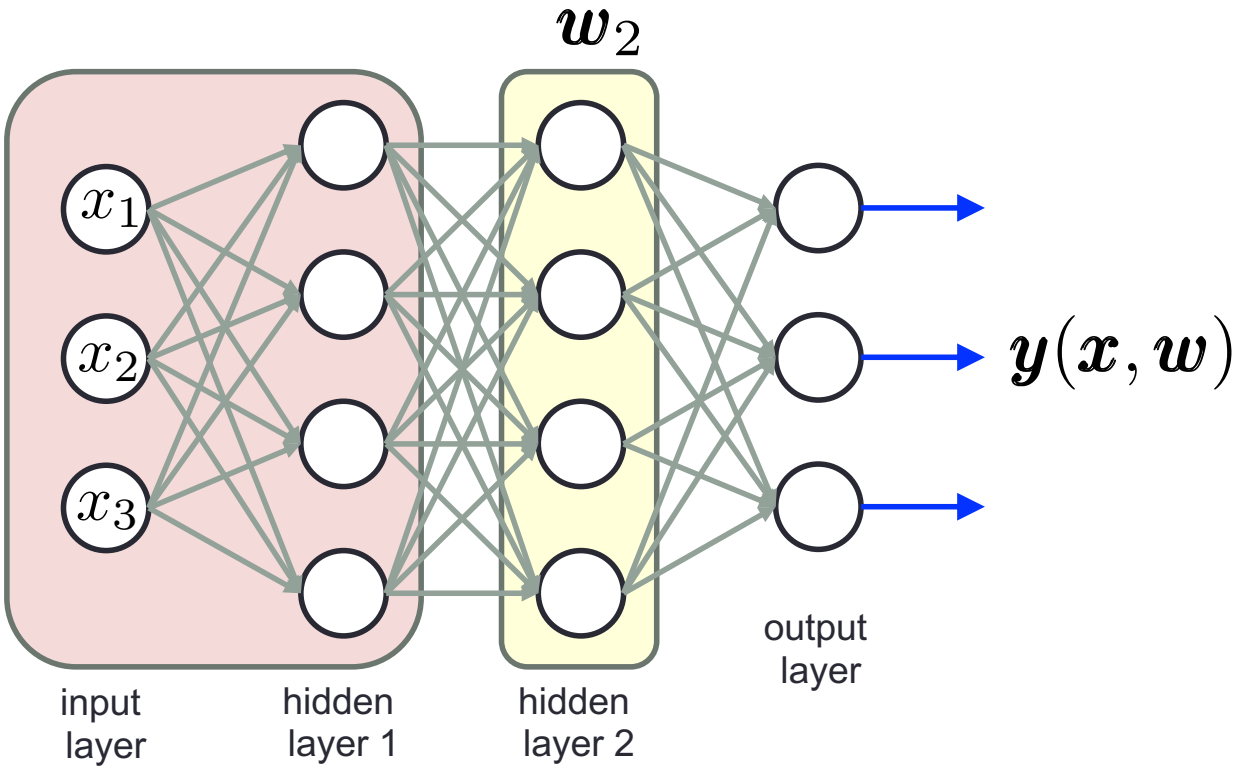
Covariance shift in a neural network

- As the previous parameters are adapted via backpropagation → their variation reflects into a variation of the range of the following features that are inputted into hidden layer no. 2 (even if the same input is reapplied)
- **This means that the distribution (range) of the input features for the hidden layer no. 2 continuously changes during learning**



Covariance shift in a neural network

- This from the perspective of hidden layer no. 2 is equivalent to changing the distribution of the input → covariance shift problem
- In a deep neural network this is referred to as internal covariance shift
  - as it is due to a change in the network weights rather than to a change in the input signal range

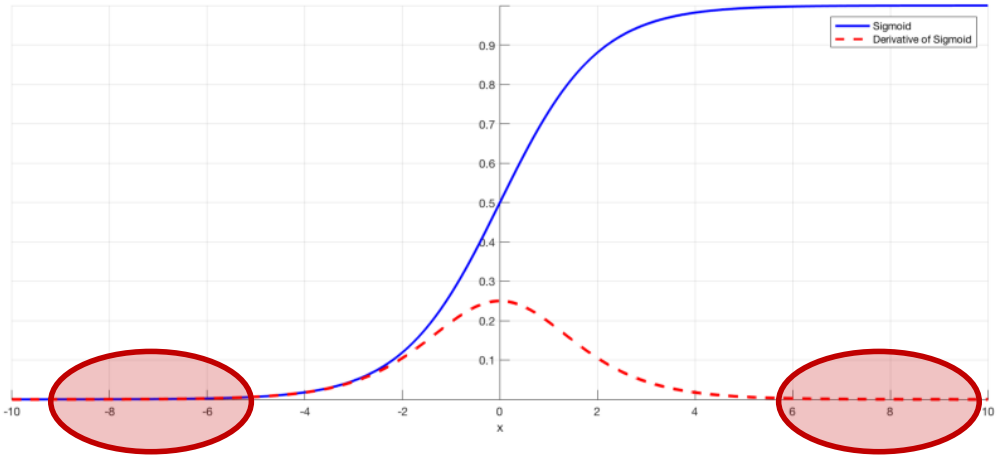


Example

- Consider a layer with sigmoidal activation functions g

$$\boldsymbol{y} = g(\boldsymbol{w} \times \boldsymbol{x} + \boldsymbol{b})$$

- **x**: input vector
- (**w**,**b**): weight matrix and bias vector to be learned
- As |**x**| increases the derivative g'(**x**) tends to zero
  - the gradient flowing down **x** vanishes → **slow training**

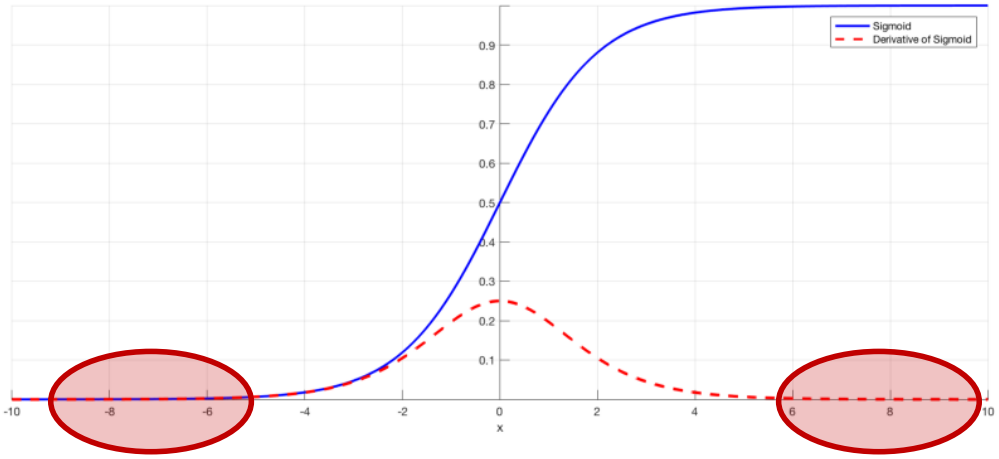


Example

- Consider now two nested sigmoidal functions

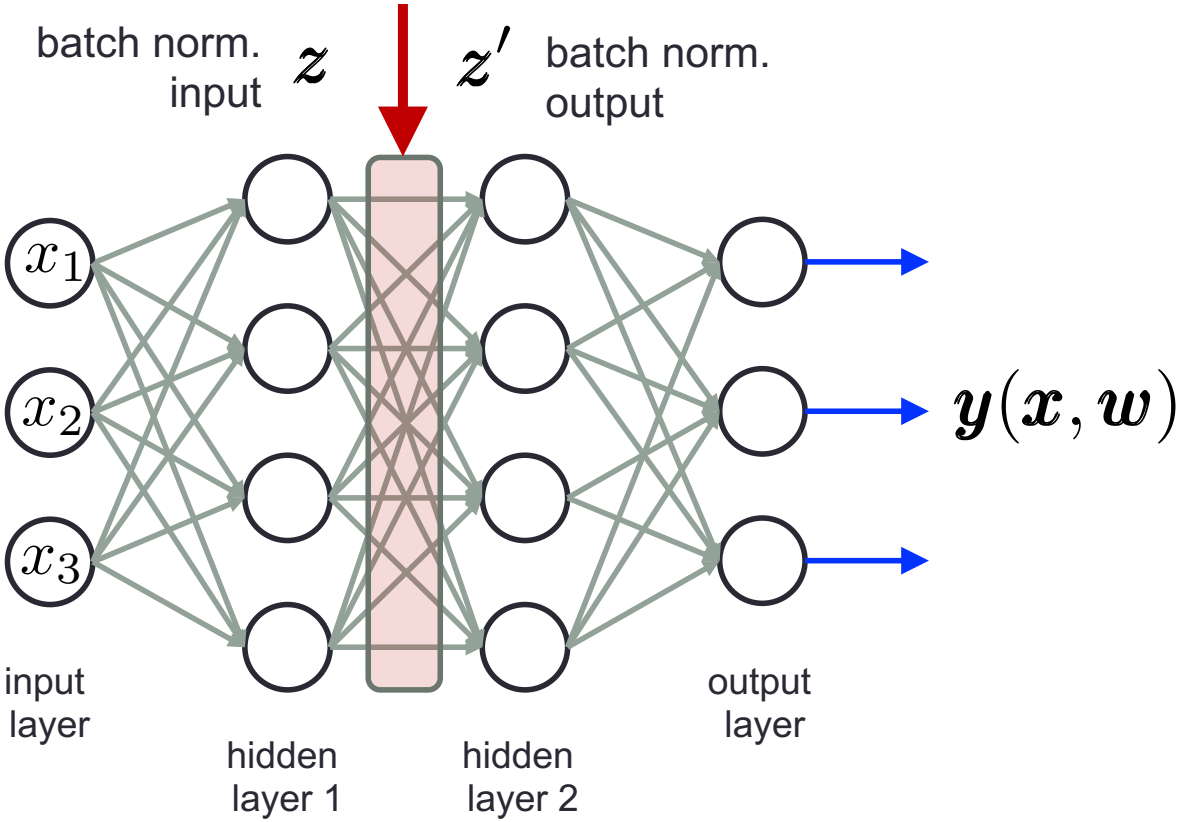
$$\boldsymbol{y} = g(\boldsymbol{w}_2 \times \boldsymbol{x} + \boldsymbol{b}_2) \quad \text{with:} \quad \boldsymbol{x} = g(\boldsymbol{w}_1 \times \boldsymbol{x}_o + \boldsymbol{b}_1)$$

If **x** itself is obtained from a sigmoid layer → its value depends on the pars $\mathbf{w}_1$ and $\mathbf{b}_1$ from the previous layer → a change of ($\mathbf{w}_1$,$\mathbf{b}_1$) during training may move many dimensions of **x** towards the saturated region by affecting learning (even though $|\mathbf{x}_o|$ may be small)



Batch normalization

- Is interposed between any two layers,
  - before the non-linear activation function for hidden layer 1
- What it does: no matter how much the output of HL1 will change, the input features for HL2 will remain invariant in terms of **mean** and **standard deviation** (reduces the variability of the input features for HL2)



Batch normalization: the effect

- With batch normalization
  - The input vector of layer n is less affected by the change in the pars in the previous layers (1,2,…,n-1)
  - The distribution of the input vector at layer n becomes "more stable"
- What we achieve
  - A decoupling between what the weights in each layer have to do
  - That is, weights in layer n no longer have to compensate for variations in the earlier weights but only for the meaningful variations in the signal
  - Allows for the layers in the network to learn a bit more independently from other layers → this speeds up the learning, especially at the deepest layers (the ones closest to the neural network output layer), which are the *most affected* by internal covariance shift

Batch normalization: regularization

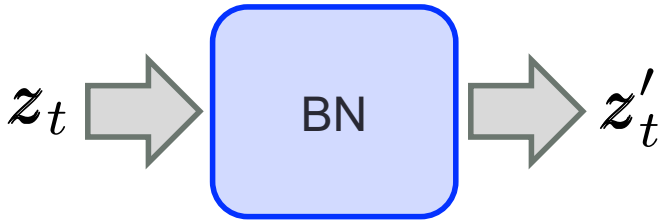
- Commonly: Stochastic Gradient Descent (SGD)
  - Learning occurs in **mini-batches** of m input samples at a time
- BN operates on each mini batch $\boldsymbol{Z}^{\{t\}} = \{\boldsymbol{z}_t, \boldsymbol{z}_{t+1}, \ldots, \boldsymbol{z}_{t+m}\}$
  - which is scaled by just the mean and variance of that minibatch
  - Scaling: results in controlled range, i.e., zero mean and unit variance, for the next layer
- Important
  - For practical reasons (implementation in conjunction with SGD)
  - Such scaling is implemented on each mini-batch, as opposed to using the mean and variance of the entire dataset
  - This means that the scaling is not carried out using the true mean and variance but using their surrogates
  - This introduces some NOISE

Batch normalization: regularization

- Normalizing on surrogate mean and variance
  - Adds some noise to each hidden layer's activations
  - Effect similar to dropout's
- Result
  - this noise has a slight regularization effect
  - as the noise added is small → this does not usually have a major regularitation effect
- Batch size
  - A larger batch size (e.g., from 64 to 512) leads to a smaller noise and to a reduced regularization effect
- Should we use it as a regularizer?
  - Not really, it is not the main intent of the technique, it is a byproduct

BN transform (1/2)

- At training time t, BN transforms input $\boldsymbol{z}_t$ into output $\boldsymbol{z}'_t$



- Input vector $\boldsymbol{z}_t = (z_t^{(1)}, z_t^{(2)}, \ldots, z_t^{(K)})$ has K activations
- The BN transform is applied to each activation $z_t^{(k)}$ independently
- Next, we refer to activation k, by omitting the apex (k)
- For this specific activation, we refer to a mini-batch

$$\mathcal{B} = \{z_{1\ldots m}\}$$

BN transform (2/2)

- Let us consider activation k
- Input: values of for such activation over a mini-batch are: $\mathcal{B} = \{z_{1\ldots m}\}$
- Output values for the mini-batch are:

$$\{z_i' = \mathrm{BN}_{\gamma,\beta}(z_i), i = 1, \ldots, m\}$$

1) $\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} z_i$

2) $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu_{\mathcal{B}})^2$

3) $\hat{z}_i \leftarrow \frac{z_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$

normalize

4) $z_i' \leftarrow \gamma\hat{z}_i + \beta \triangleq \mathrm{BN}_{\gamma,\beta}(z_i)$

scale and shift

Effect of γ and β

$$z_i' \leftarrow \gamma \hat{z}_i + \beta \triangleq \mathrm{BN}_{\gamma,\beta}(z_i) \quad \text{where,} \quad \hat{z}_i \leftarrow \frac{z_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

- Allows the learning process to automatically set the mean and variance of the ouput to any desired value
- In fact if it holds that,

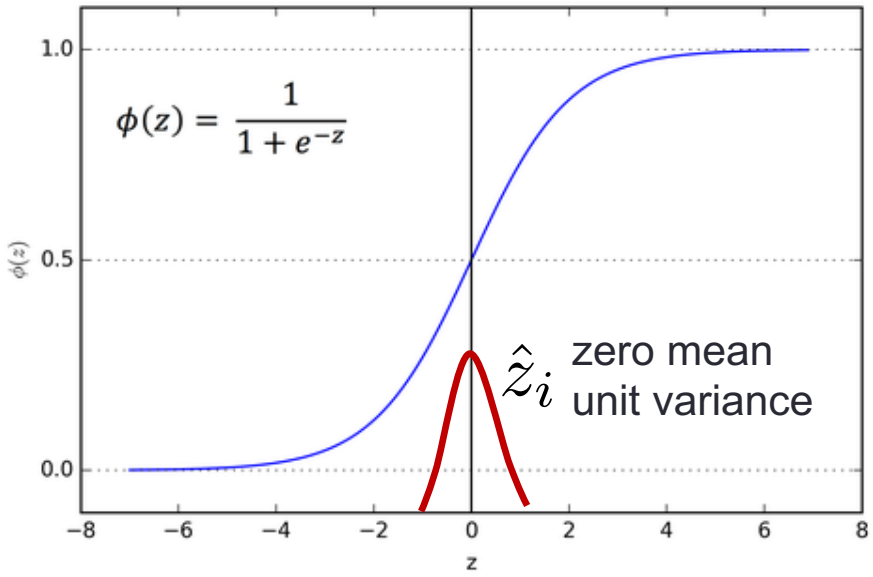
$$\gamma = \sqrt{\sigma_{\mathcal{B}}^2 + \epsilon} \quad \text{and} \quad \beta = \mu_{\mathcal{B}}$$

- Then γ and β exactly invert the BN transform by getting an identity function out of this transformation

$$z_i' = z_i$$

Effect of γ and β

- Why would we want to change the mean and variance?
- Example: if following this layer there is a sigmoid activation, then we may want the values to be distributed across a larger input space wrt $\hat{z}_i$ to take advantage of the non-linearity too (flat regions)



we let learning (backpropagation) decide the correct scaling → this freedom is provided by γ and β

As a result

- Parameters γ and β are
  - Two additional network parameters
  - That have to be learned
  - Used to scale the mean and variance of the input to each layer

- This is powerful
  - As the training algorithm (backpropagation) has now an option of shaping the input distribution to each layer to the scale that would make the layer work best, i.e., in a way that minimizes the error function during training

Training *vs* test time

- At training time – Batch-normalization
  - Computes local mean and variance
  - Over the mini-batches (m samples at a time)
  - So it is a local procedure
- At test time however
  - The batch-normalization block uses the average and standard deviation values computed over the entire dataset (two global values), by averaging over all the mini-batches

$$\mu_{\text{test}} \leftarrow E_{\mathcal{B}}\left[\mu_{\mathcal{B}}\right] = \sum_{\mathcal{B}} \mu_{\mathcal{B}} / N_{\mathcal{B}}$$

$$\sigma^2_{\text{test}} \leftarrow \frac{m}{m-1} E_{\mathcal{B}}\left[\sigma^2_{\mathcal{B}}\right]$$

BN output at test time

- At test time, BN operates as follows

$$
\begin{aligned}
z_i' &= \gamma \hat{z}_i + \beta = \\
&= \gamma \frac{z_i - \mu_{\text{test}}}{\sqrt{\sigma^2_{\text{test}} + \epsilon}} + \beta = \\
&= \gamma \frac{z_i}{\sqrt{\sigma^2_{\text{test}} + \epsilon}} + \beta - \gamma \frac{\mu_{\text{test}}}{\sqrt{\sigma^2_{\text{test}} + \epsilon}}
\end{aligned}
$$

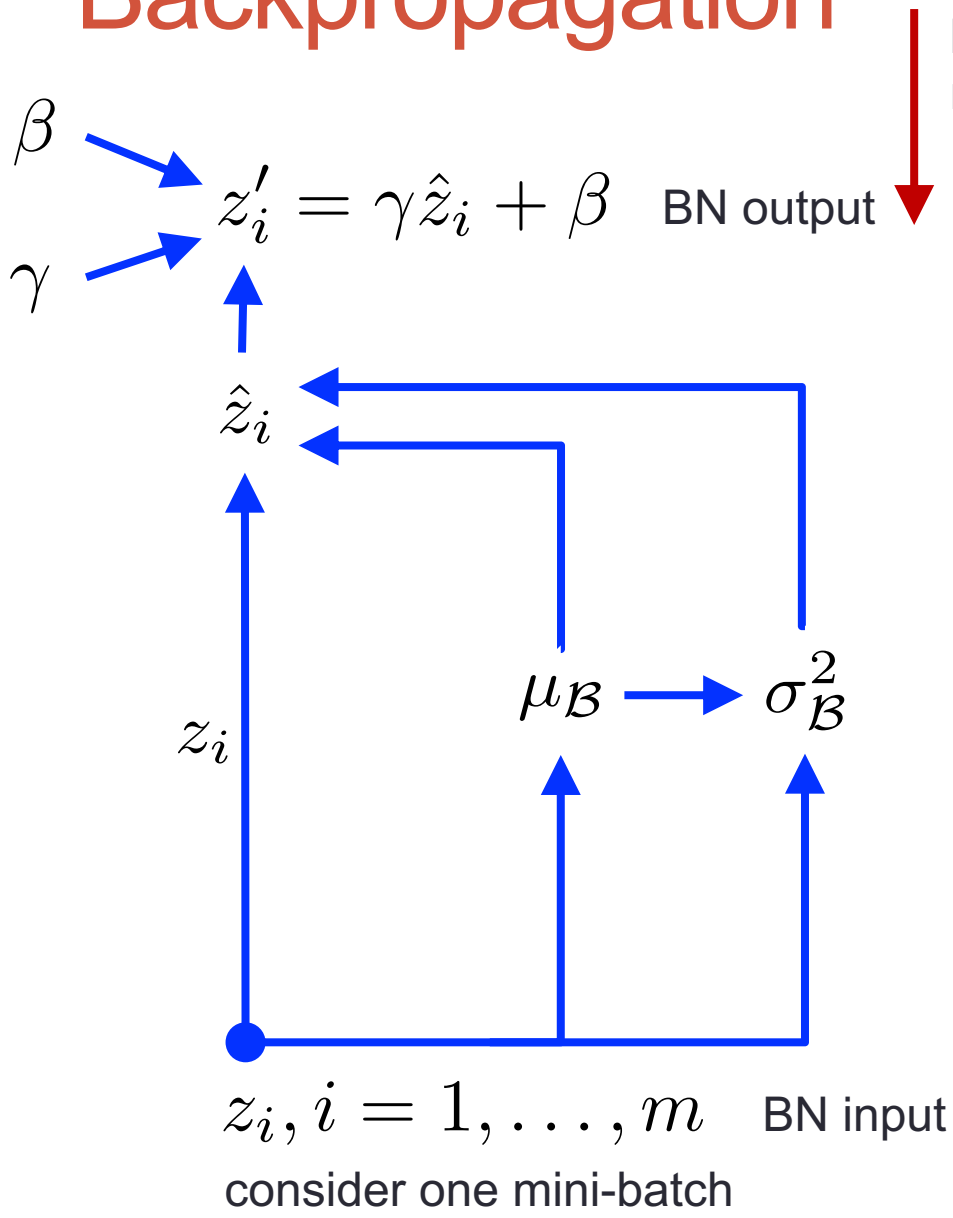
that is, using standard deviation and mean computed across all the mini-batches that were used for the training

One last point: training

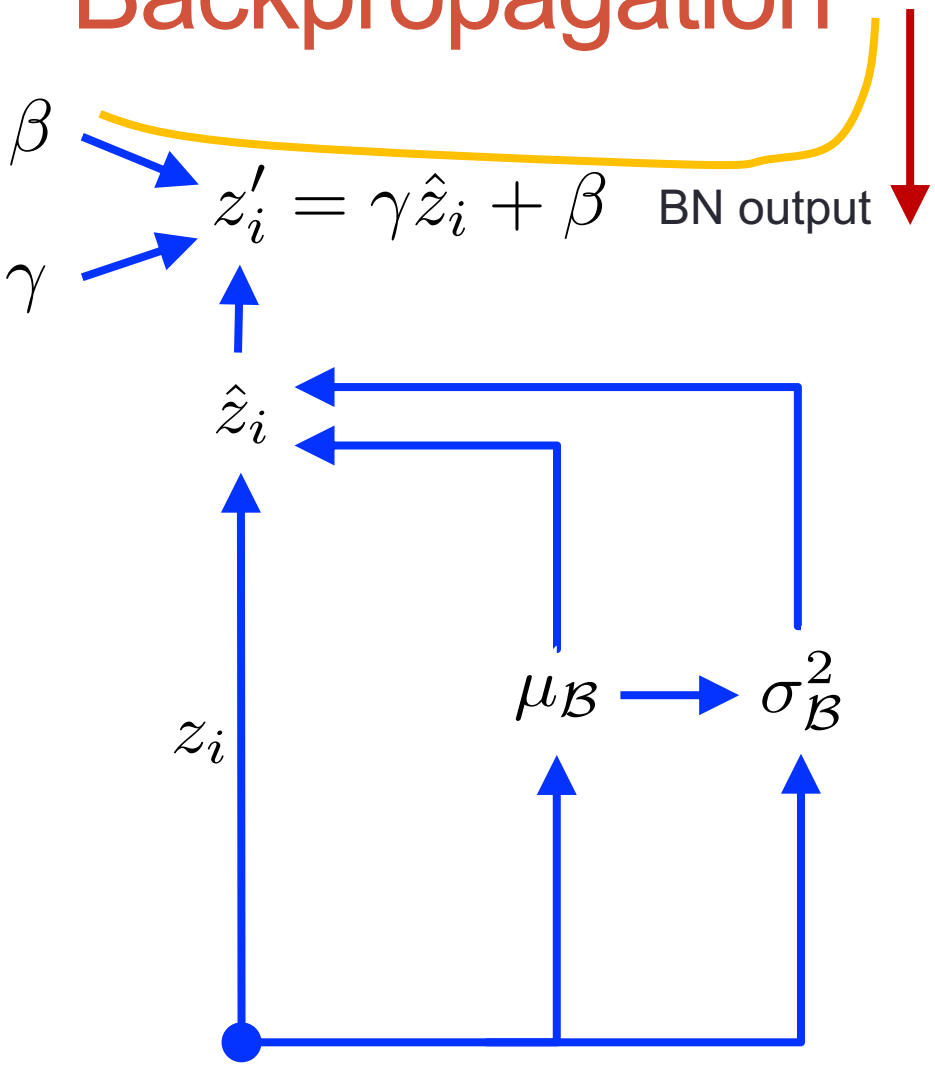
- From what we have seen so far
  - It should be clear that BN is a network layer
  - As such, its parameters γ and β have to be learned
  - As any other network parameter

- Backpropagation needs to be adapted
  - To *adjust* the BN parameters and
  - Correctly propagate the error gradient to the previous layer

Backpropagation



Backpropagation



loss L coming in from next layer must be backpropagated through z_i'

$$\frac{\partial L}{\partial z_i'}$$

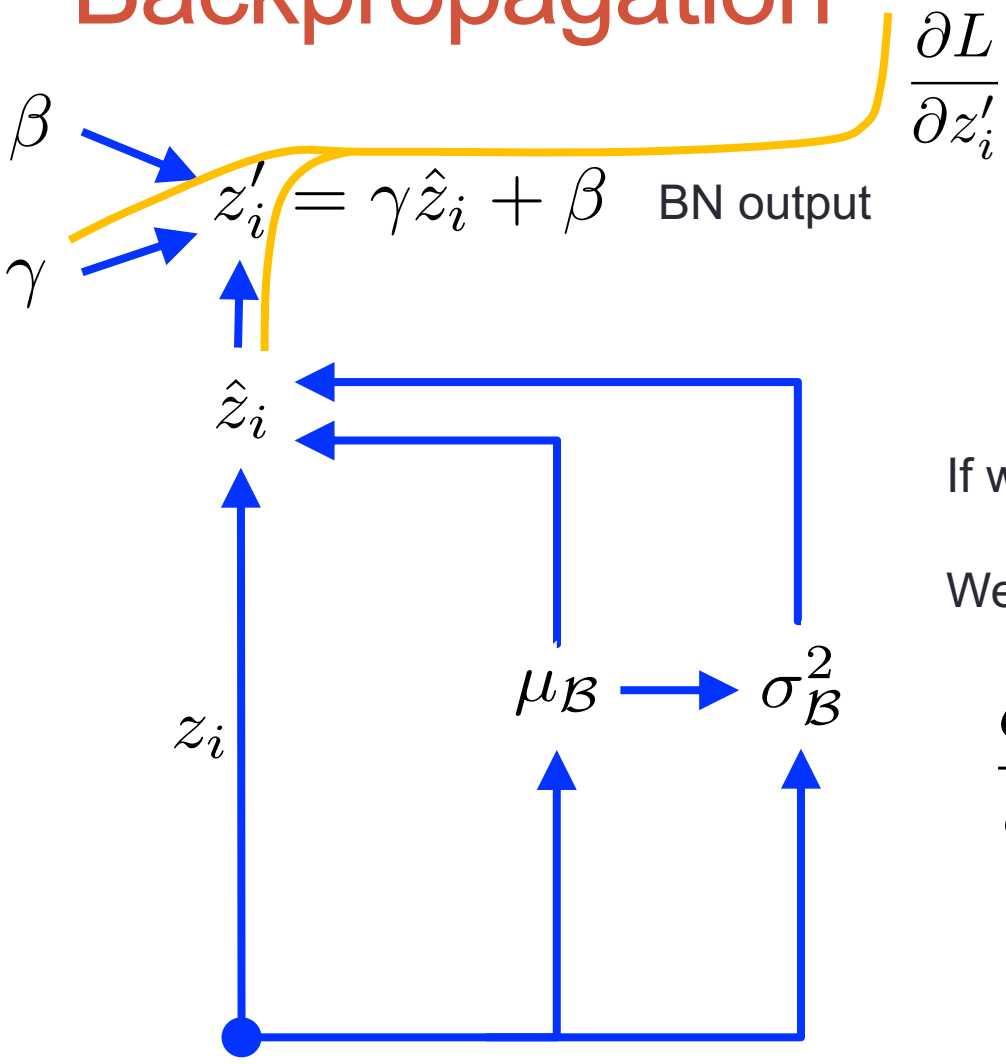
If we want to compute $\frac{\partial L}{\partial \beta}$

We need to follow the orange path

$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial z_i'} \cdot \frac{\partial z_i'}{\partial \beta} =$$

$$= \sum_{i=1}^{m} \frac{\partial L}{\partial z_i'}$$

Backpropagation



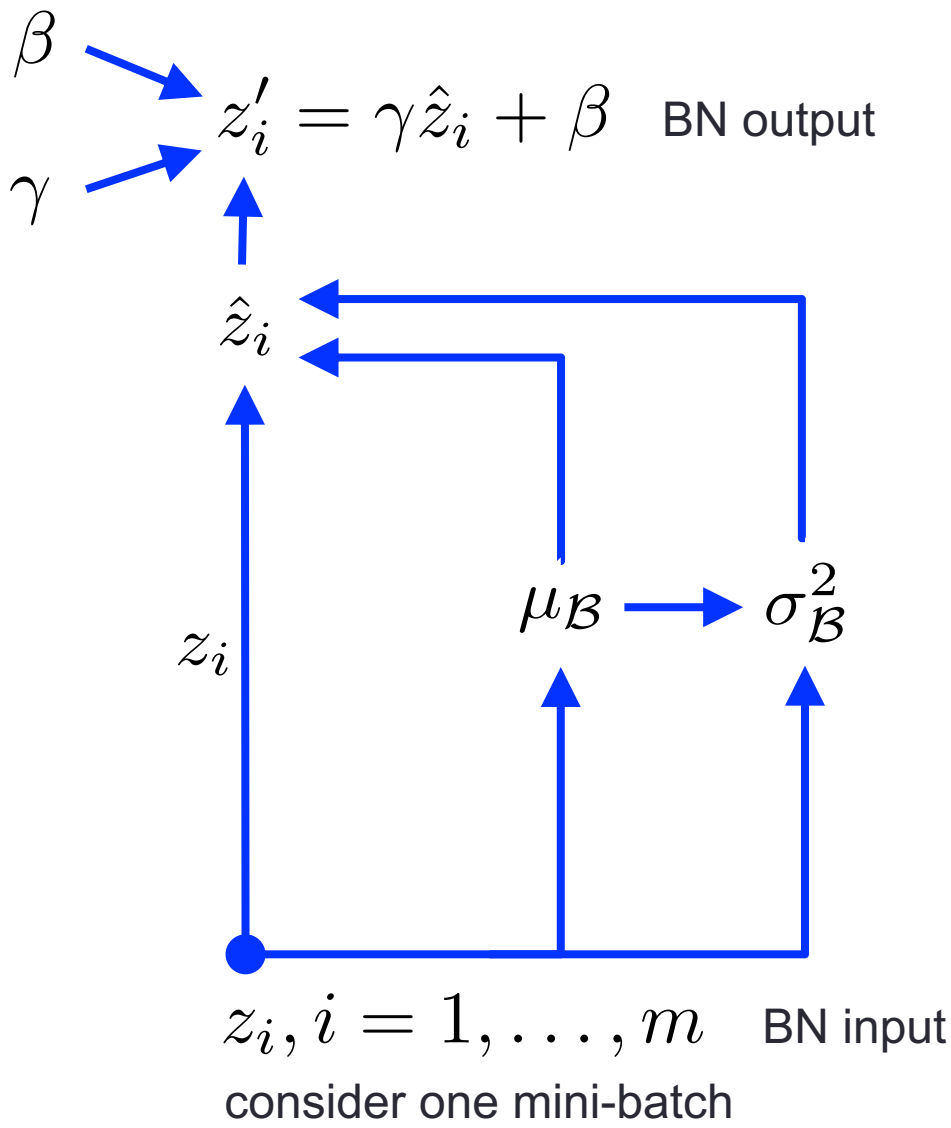
If we want to compute $\frac{\partial L}{\partial \gamma}$

We obtain:

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial z'_i} \cdot \frac{\partial z'_i}{\partial \gamma} =$$

$$= \sum_{i=1}^{m} \frac{\partial L}{\partial z'_i} \cdot \hat{z}_i$$

BN update of parameters

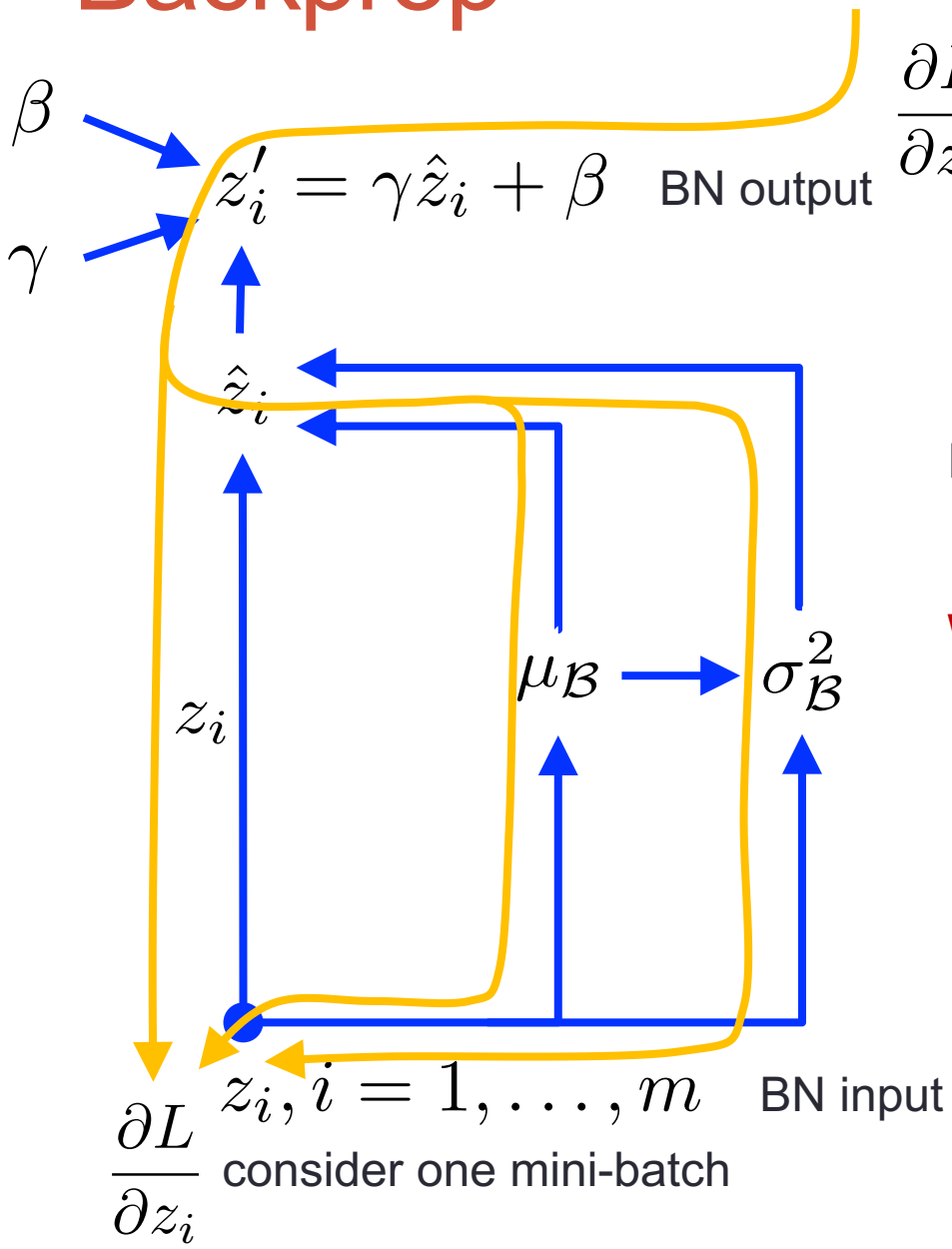


$$\gamma_{\text{new}} = \gamma_{\text{old}} - \eta \frac{\partial L}{\partial \gamma}$$

$$\beta_{\text{new}} = \beta_{\text{old}} - \eta \frac{\partial L}{\partial \beta}$$

$$\eta \rightarrow \text{ learning rate}$$

Backprop

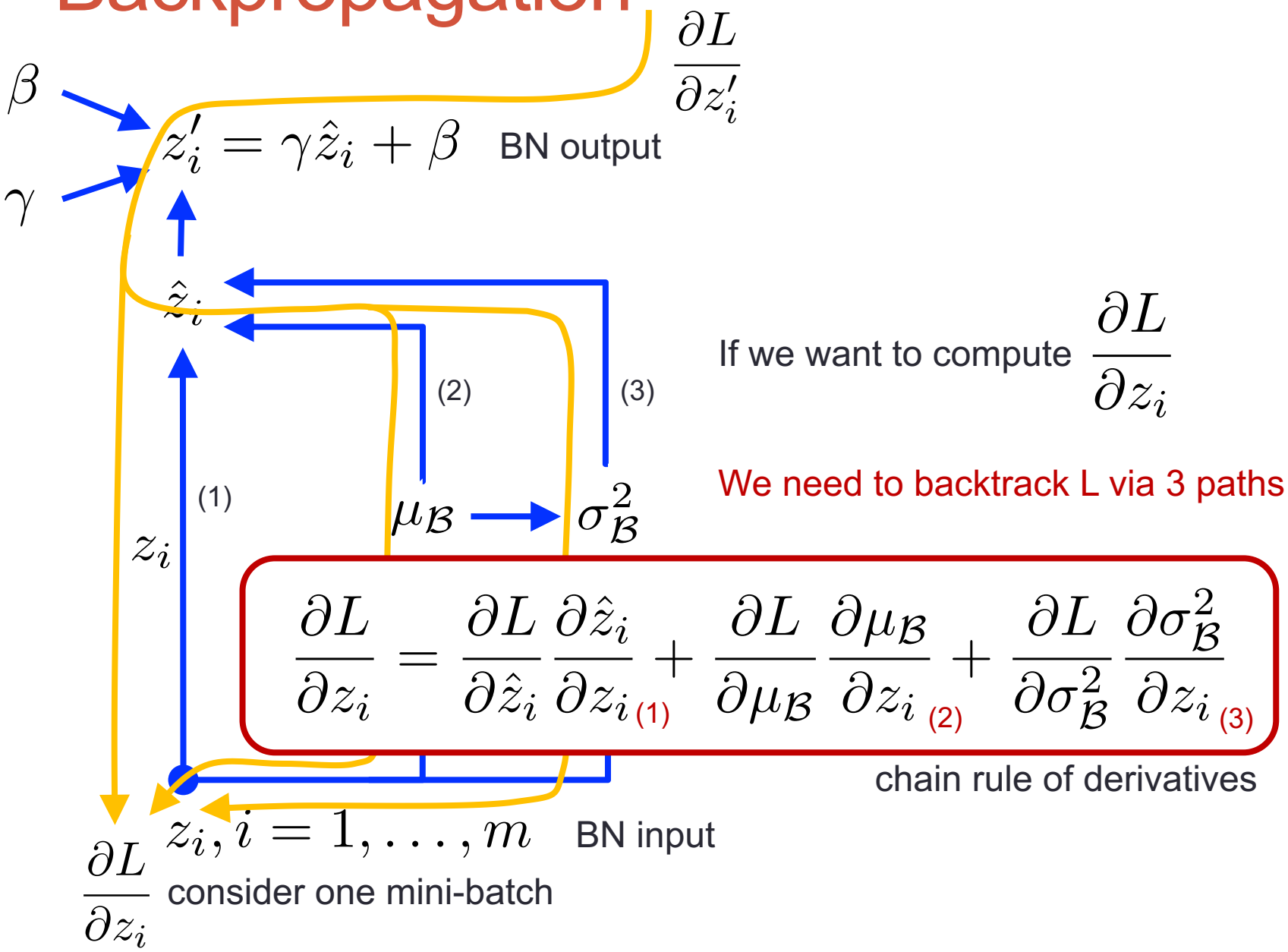


We now backpropagate the loss L from output to input section

If we want to compute $\frac{\partial L}{\partial z_i}$

We need to backtrack L via 3 paths

Backpropagation



If we want to compute $\frac{\partial L}{\partial z_i}$

We need to backtrack L via 3 paths

Deriving the remaining backprop eqs

From

$$z_i' = \gamma \hat{z}_i + \beta$$

We obtain

$$\frac{\partial L}{\partial \hat{z}_i} = \frac{\partial L}{\partial z_i'} \frac{\partial z_i'}{\partial \hat{z}_i} = \frac{\partial L}{\partial z_i'} \cdot \gamma$$

Deriving the remaining backprop eqs

Recap

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} z_i \quad \sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu_{\mathcal{B}})^2 \quad \sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu_{\mathcal{B}})^2$$

These lead to (three paths to follow downwards):

$$\frac{\partial L}{\partial z_i} = \frac{\partial L}{\partial \hat{z}_i}\frac{\partial \hat{z}_i}{\partial z_i} + \frac{\partial L}{\partial \mu_{\mathcal{B}}}\frac{\partial \mu_{\mathcal{B}}}{\partial z_i} + \frac{\partial L}{\partial \sigma^2_{\mathcal{B}}}\frac{\partial \sigma^2_{\mathcal{B}}}{\partial z_i} =$$

$$= \frac{\partial L}{\partial \hat{z}_i}\cdot\frac{1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} + \frac{\partial L}{\partial \mu_{\mathcal{B}}}\cdot\frac{1}{m} + \frac{\partial L}{\partial \sigma^2_{\mathcal{B}}}\cdot\frac{2(z_i - \mu_{\mathcal{B}})}{m}$$

Deriving the remaining backprop eqs

Using

$$\hat{z}_i \leftarrow \frac{z_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

We obtain (one path to follow downwards):

$$\frac{\partial L}{\partial \sigma^2_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{z}_i} \cdot \frac{\partial \hat{z}_i}{\partial \sigma^2_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{z}_i} \cdot (z_i - \mu_{\mathcal{B}}) \cdot \frac{\partial (\sigma^2_{\mathcal{B}} + \epsilon)^{-1/2}}{\partial \sigma^2_{\mathcal{B}}}$$

$$= \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{z}_i} \cdot (z_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma^2_{\mathcal{B}} + \epsilon)^{-3/2}$$

Deriving the remaining backprop eqs

Using

$$\hat{z}_i \leftarrow \frac{z_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \sigma^2_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu_{\mathcal{B}})^2$$

We obtain (two paths to follow downwards):

$$\frac{\partial L}{\partial \mu_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{z}_i} \cdot \frac{\partial \hat{z}_i}{\partial \mu_{\mathcal{B}}} + \frac{\partial L}{\partial \sigma^2_{\mathcal{B}}} \cdot \frac{\partial \sigma^2_{\mathcal{B}}}{\partial \mu_{\mathcal{B}}}$$

$$= \left(\sum_{i=1}^{m} \frac{\partial L}{\partial \hat{z}_i} \cdot \frac{-1}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \right) + \frac{\partial L}{\partial \sigma^2_{\mathcal{B}}} \cdot \frac{\sum_{i=1}^{m} -2(z_i - \mu_{\mathcal{B}})}{m}$$

Backpropagation

Final expression

$$\frac{\partial L}{\partial \hat{z}_i} = \frac{\partial L}{\partial z_i'} \cdot \gamma$$

$$\frac{\partial L}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial L}{\partial \hat{z}_i} \cdot (z_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2} (\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial L}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial L}{\partial \hat{z}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial L}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(z_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial L}{\partial z_i} = \frac{\partial L}{\partial \hat{z}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial L}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m} + \frac{\partial L}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(z_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial L}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial L}{\partial z_i'} \cdot \hat{z}_i$$

$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial L}{\partial z_i'}$$

1) update pars γ and β via gradient descent
2) backpropagate the gradient of the loss L

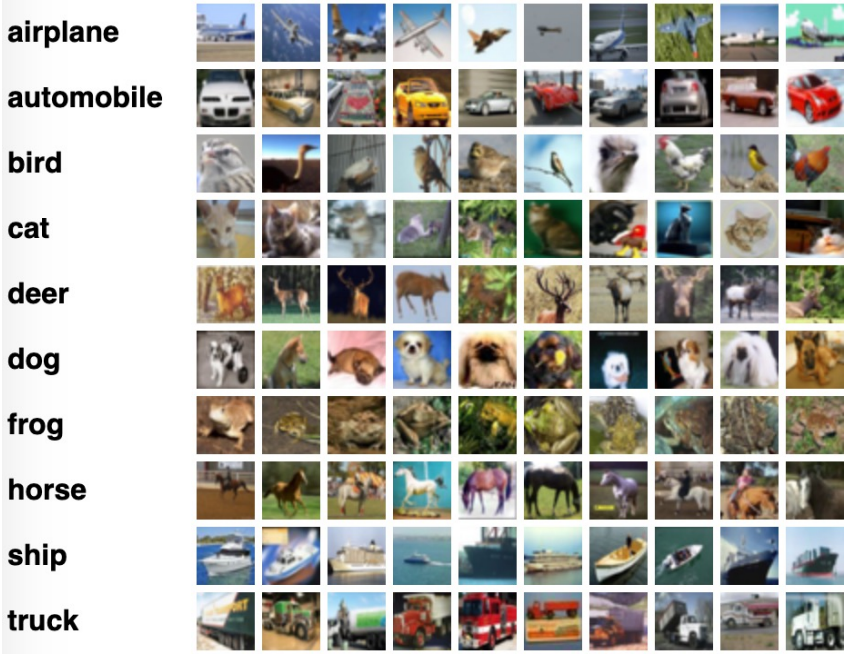
$$\frac{\partial L}{\partial z_i} \Leftarrow \frac{\partial L}{\partial z_i'}$$

Popular datasets for DL research

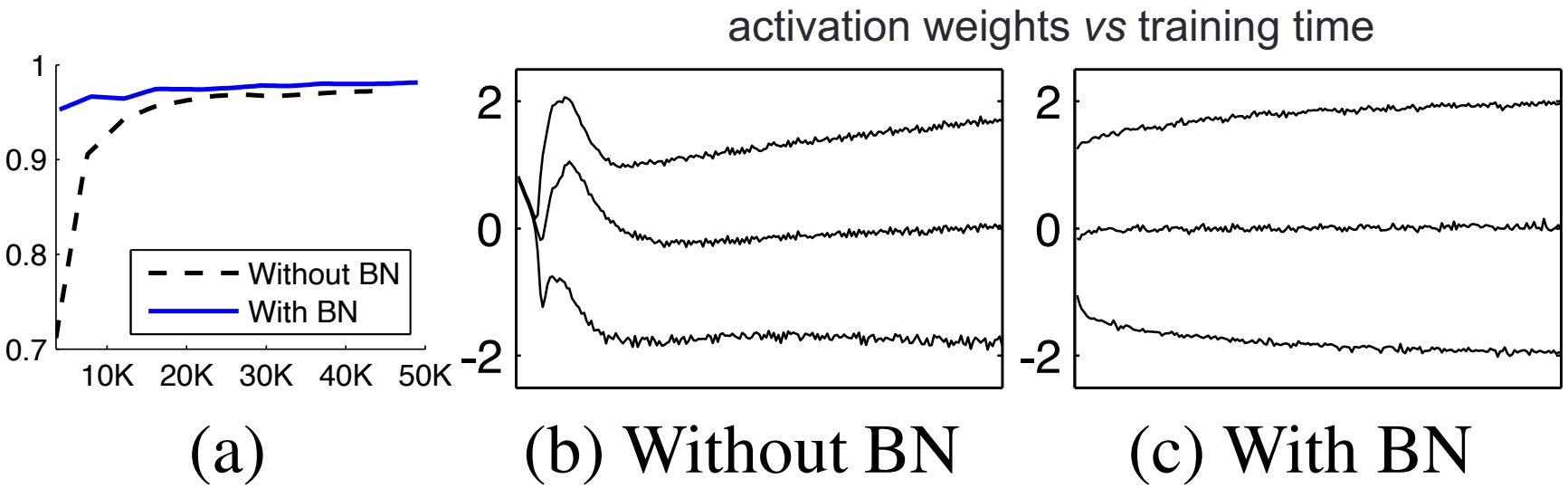
- CIFAR-10 (Canadian Institute for Advanced Research)
  - 60,000 color images of size 32x32 from 10 classes
  - Commonly used to train and test machine learning algorithms
  - It is a labaled subset of the 80 million tiny images dataset
  - Automatically collected
  - Dismissed in June 2020

IMAGENET

- ImageNet
  - Image database
  - 14,197,122 images
  - >500 images per node (noun)
  - Stanford Vision lab.



Experimental Results (1/2)

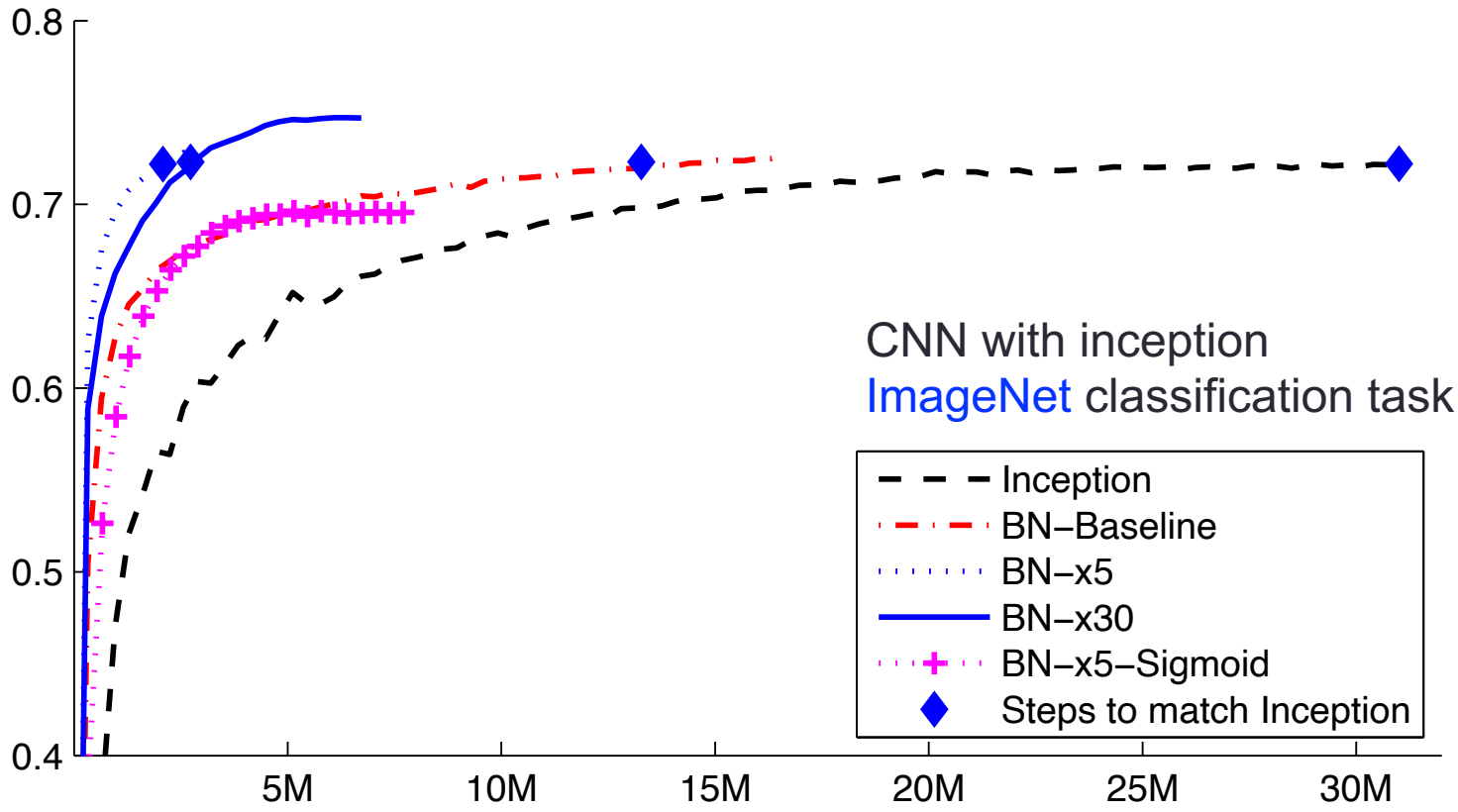


convergence *vs* time

Training performed on MNIST dataset (Le Cun et al. 1998) of hand written digits

- 28x28 binary images as input
- 3 *fully connected* layers with *sigmoid* activations
- last hidden layer followed by a fully connected layer with
  - 10 activations (one per class)
  - cross-entropy loss
- network trained for 50k steps with mini-batches of size 60

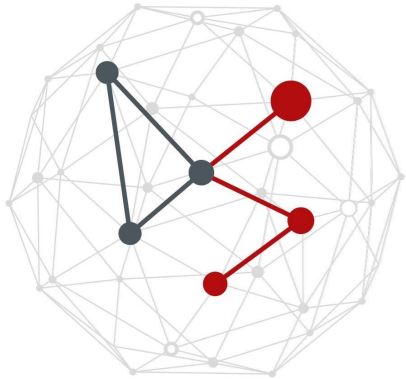
Experimental Results (2/2)



Inception: trained with initial learning rate of **0.0015**. BN-baseline: same as inception with BN before each non-linearity. BN-x5: BN with increased learning rate to 0.0075 (factor 5), no dropout (speeds up training with no overfittng), reduce L_2 weight regularizer (factor 5), accelerated learning rate decay (6 times faster)

MIME
Master's degree ICT Internet Multimedia Engineering

RESIDUAL NETWORKS



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
DIPARTIMENTO MATEMATICA
1222 · 2022 800 ANNI
UNIVERSITÀ DEGLI STUDI DI PADOVA

Bibliography

[He2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, “Deep Residual Learning for Image Recognition,” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, US, 2016. [Best Paper Award]

[He2016-b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, “Identity Mappings in Deep Residual Networks,” European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, October 2016.

Deep ResNet with 1k layers

https://github.com/KaimingHe/resnet-1k-layers

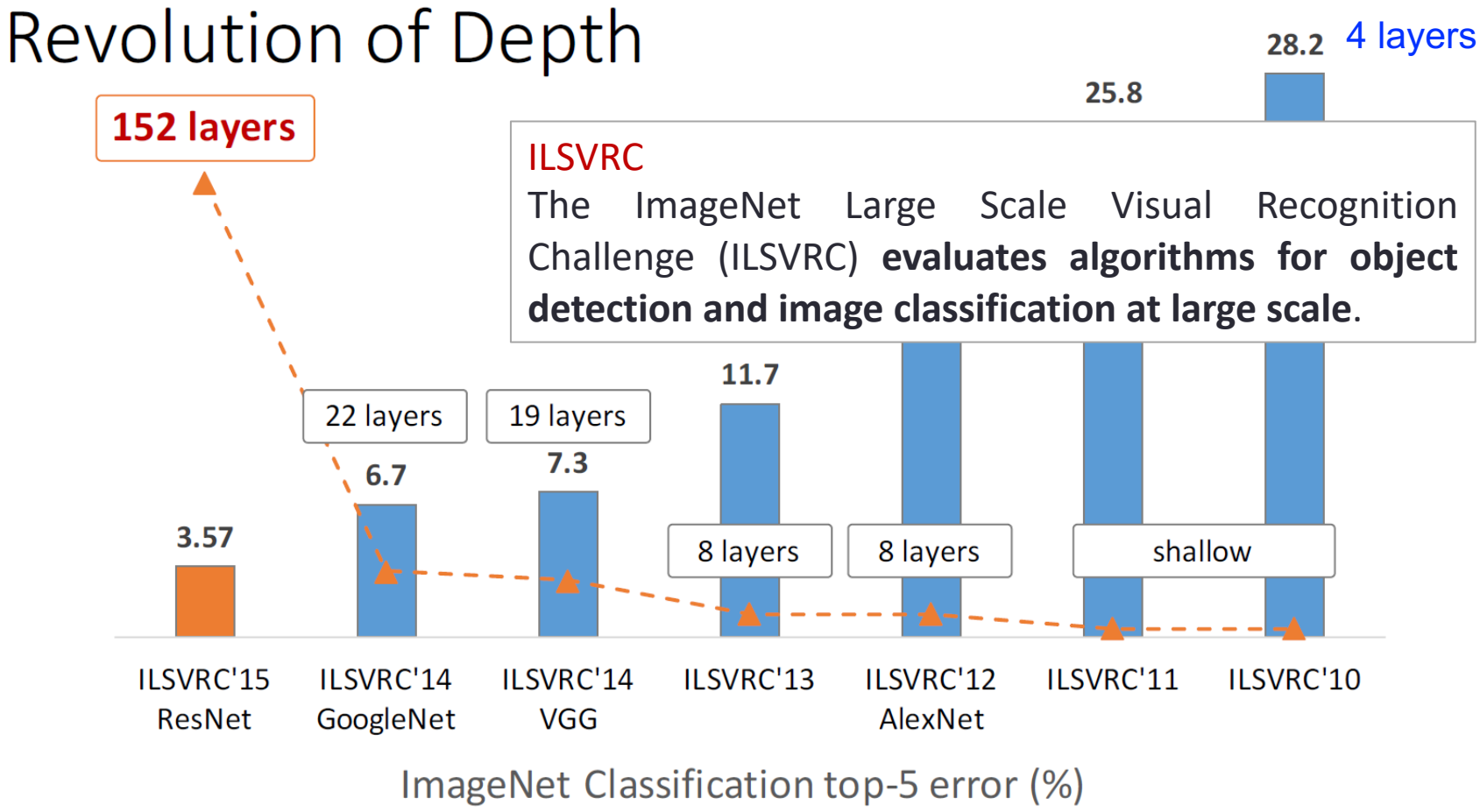
top-5 error for a classification algo?

- We would like to classify objects into K classes
- We do that using a neural network (NN), trained as a **classifier** (softmax output) with output probabilities

$$\boldsymbol{p}_{\text{out}} = [p_1\, p_2\, \cdots\, p_K]^T \qquad p_i \in [0,1]\,, \sum_i p_i = 1$$

- top-5 score
  - Input a new pattern from the test set
  - check whether the pattern's target label is one of the top 5 predictions (in the 5 highest NN output probabilities)
  - count the number of times that the predicted label is contained in the 5 most probable classes from the NN and divide it by the number of test patterns evaluated
- top-5 error
  - 1-(top-5 score)

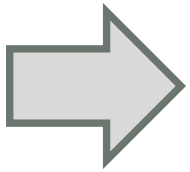
Evolution of Deep Networks



Desiderata & problem ahead

- We would like to implement deeper networks
  - Stacking as many layers as possible
  - Greater representation capability
  - Features computed by deeper layers are more abstract
    - Generally more powerful at discriminating complex structure & properties
  - This is supported by empirical evidence

- Big problem ahead
  - Backpropagation is what we use to train the network weights
  - Propagating backwards the gradient from output to input layer



vanishing gradients

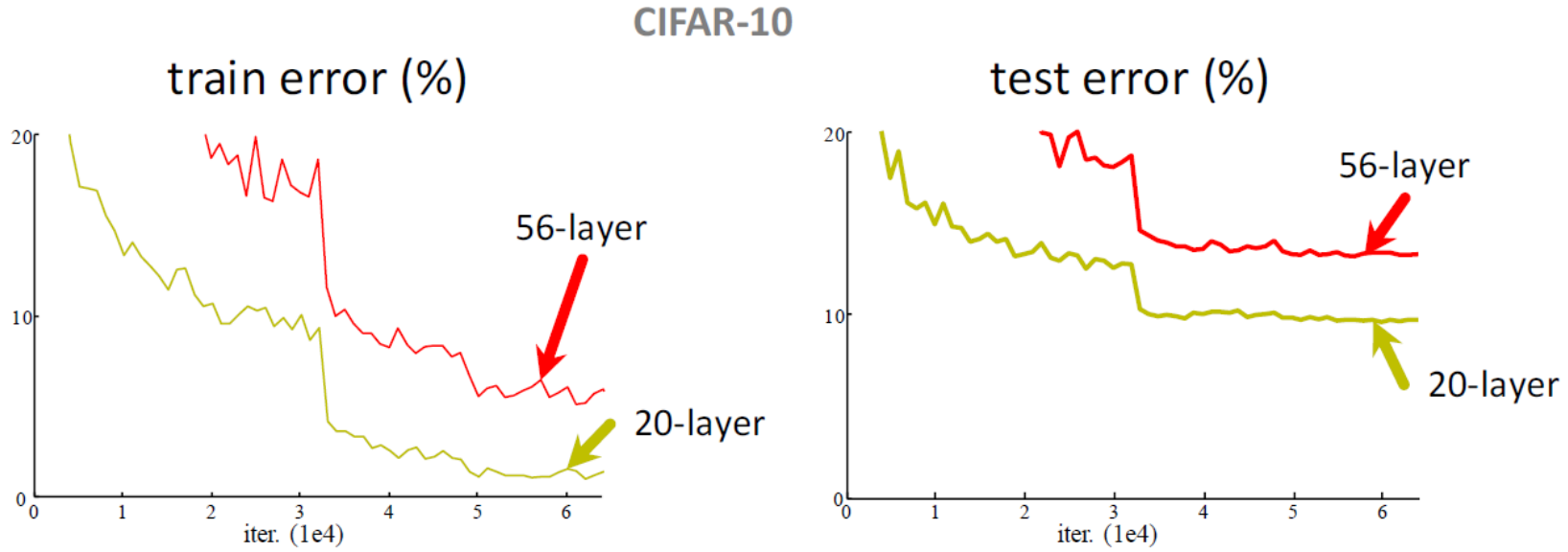
Vanishing gradients

- Can be ameliorated by

  ① Normalized initialization
  ② Batch normalization
  ③ Selection of non-linearity (to some extent)

  However, the problem is still there, and the above measures are still insufficient

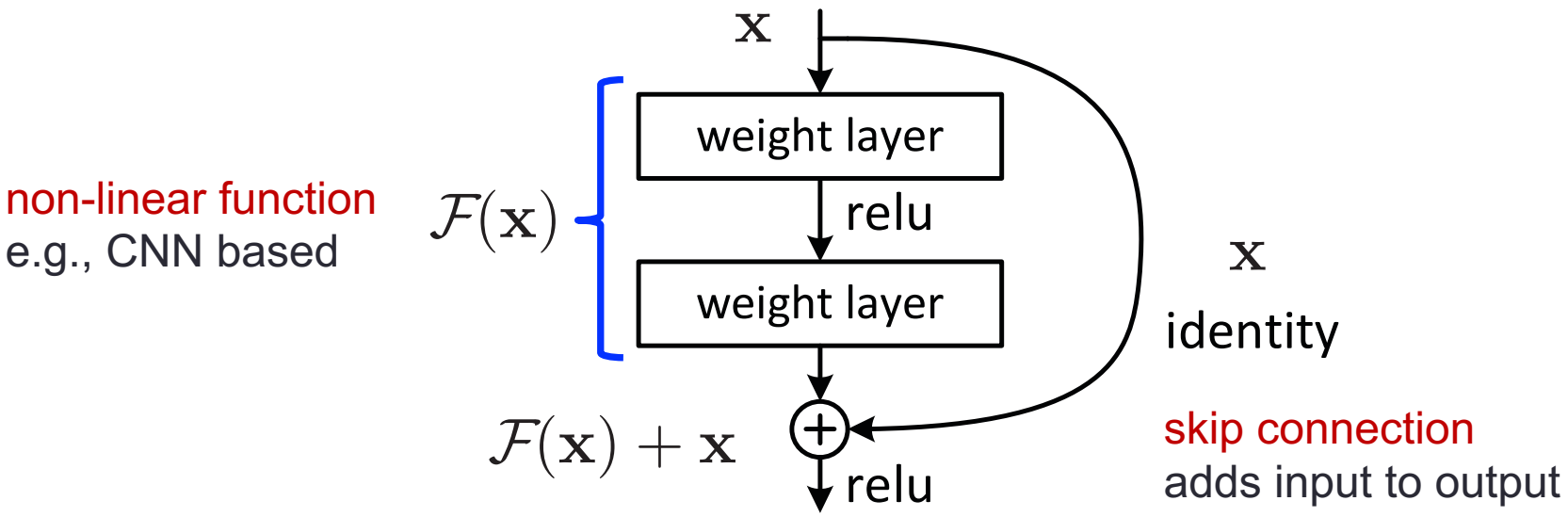
Standard deep networks on CIFAR-10



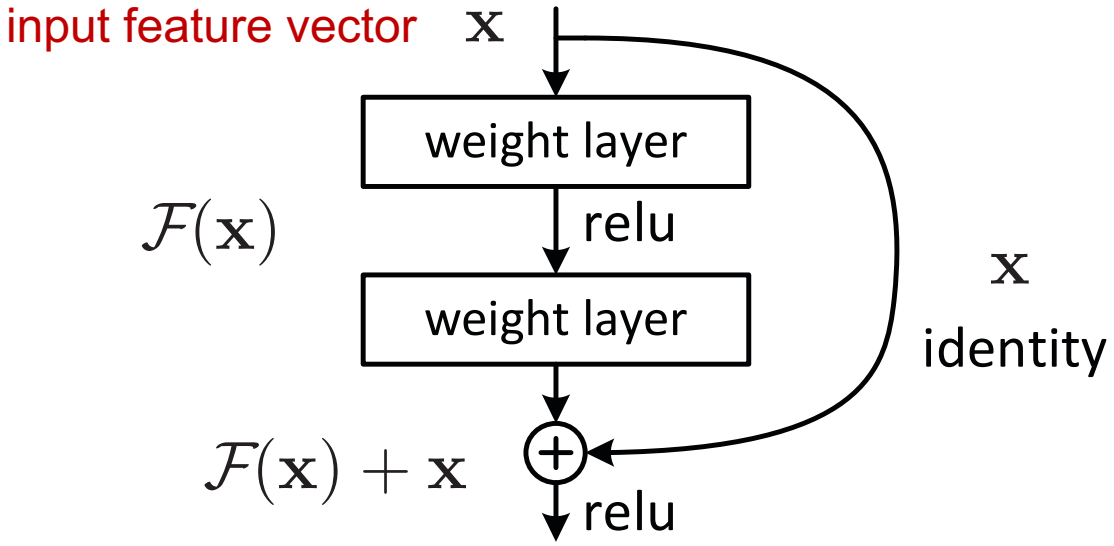
1. Rich solution space
2. Deeper, in principle, should lead to lower training errors
   1. **Original layers:** copies of “shallow” ones
   2. **Extra layers:** set as identities
   3. **Result:** should get same training error
3. However, things get worse when going deeper

Solution – residual units (1/3)

- Forcing non-linear network layers
  - to learn residuals (deviations) wrt input signal [He2016]
  - achieved by adding skip connections



Solution – residual units (2/3)



Bottom line

- $\mathbf{y} = \mathcal{F}(\mathbf{x}) + \mathbf{x}$ is the desired output feature vector
- $\mathbf{x}$ is known, so we are learning the residual

$$\mathcal{F}(\mathbf{x}) = \mathbf{y} - \mathbf{x}$$

- Usually the residual is a small quantity (good)

Residual units (3/3)

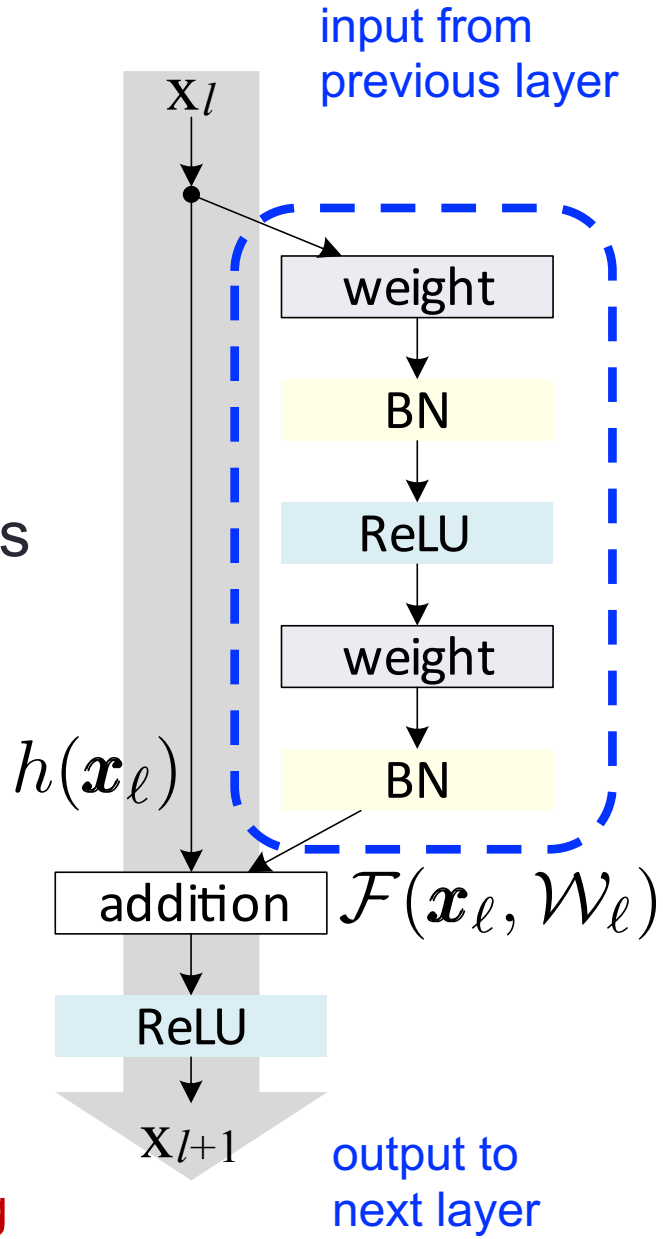
Formally

- a new way of impl. network layers
- stacking so called residual units
- each residual unit can be expressed as

$$\boldsymbol{y}_\ell = h(\boldsymbol{x}_\ell) + \mathcal{F}(\boldsymbol{x}_\ell, \mathcal{W}_\ell)$$

$$\boldsymbol{x}_{\ell+1} = f(\boldsymbol{y}_\ell)$$

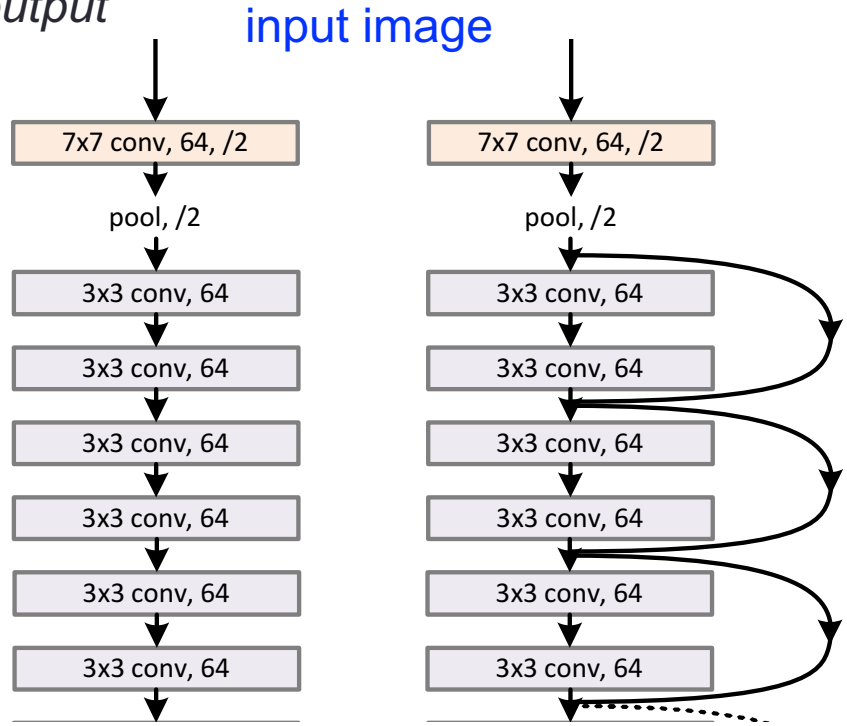
- where
  - h() is an identity mapping - key choice!
    - implemented using a skip connection
  - f() is a ReLU non-linearity
  - $\mathcal{F}(\boldsymbol{x}_\ell, \mathcal{W}_\ell)$ (non-linear) residual mapping



Design in [He2016]

- Convolutional layers
  - use CNN with (mostly) 3x3 filters
  - network ends with
    - average pooling and
    - **fully connected layer** with *softmax output*
  - 34 CONV layers in cascade

first seven layers
Left → conventional CNN
Right → CNN + residual layers



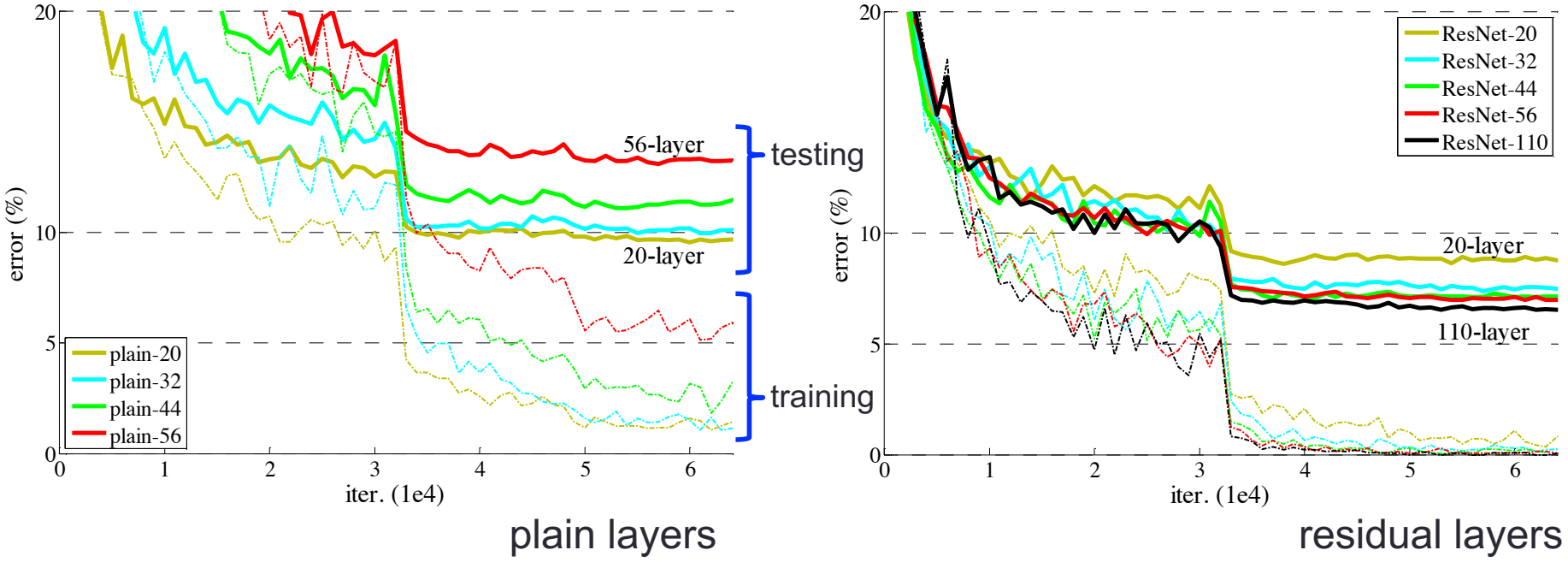
Results on ImageNet

ResNets beat all implementations from prior research

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

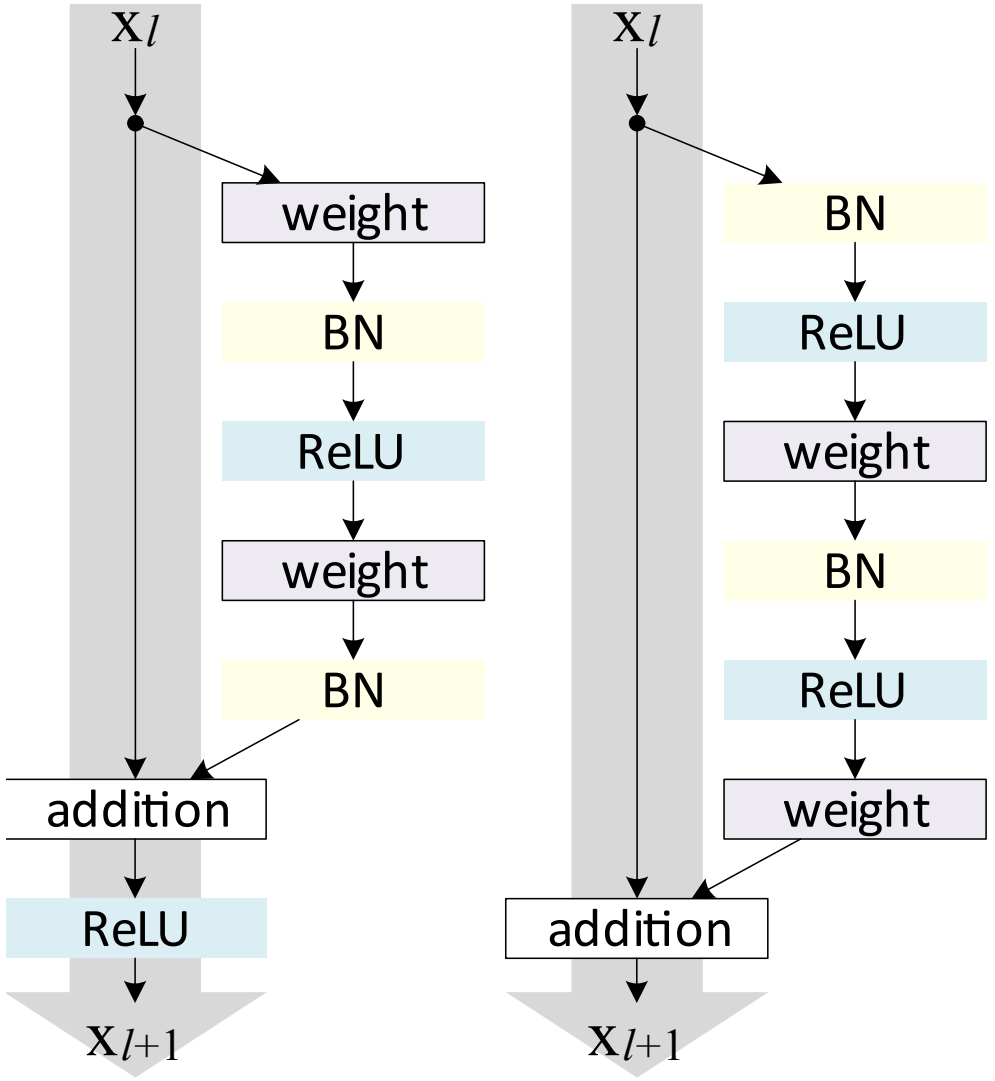
Results on CIFAR-10

ResNets can go deeper and still improve their performance



[He2016-b]

- New proposal
  - treats BN + ReLU as
    "pre-activations"
  - rather than
    "post-activations"
- NEW - both functions
  - h() and f() are identities
  - see diagram (b) on the right



(a) original (b) proposed

Residual layer in [He2016-b]

- The new residual layer → $\boldsymbol{y}_\ell = \boldsymbol{x}_\ell + \mathcal{F}(\boldsymbol{x}_\ell, \mathcal{W}_\ell)$ as before

  $\boldsymbol{x}_{\ell+1} = f(\boldsymbol{y}_\ell) = \boldsymbol{y}_\ell$ new!!!

- Stacking-up L residual layers

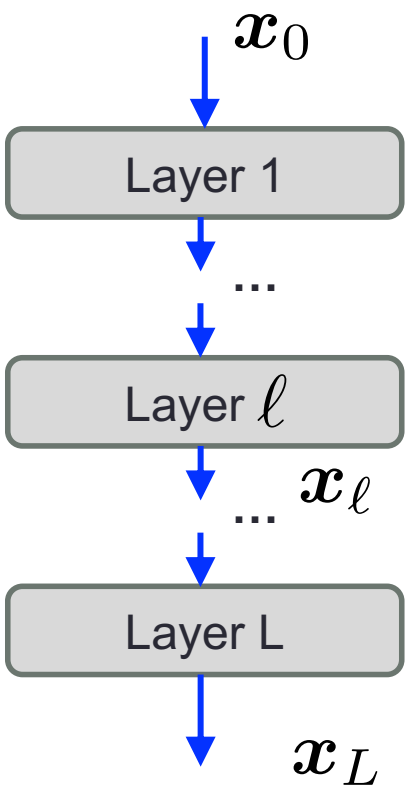
$$\boldsymbol{x}_L = \boldsymbol{x}_0 + \sum_{i=0}^{L-1} \mathcal{F}(\boldsymbol{x}_i, \mathcal{W}_i)$$

- The ouput feature of residual unit L is obtained as
  - the summation of the output of the L residual functions plus $\mathbf{x}_0$
- Information propagates *directly* from input to output layer

Backpropagating the gradient

- Starting from layer ℓ and stacking up residual layers, leads to

$$\boldsymbol{x}_L = \boldsymbol{x}_\ell + \sum_{i=\ell}^{L-1} \mathcal{F}(\boldsymbol{x}_i, \mathcal{W}_i)$$



Considering layer ℓ

output → error E

Backpropagating the gradient

- Starting from layer ℓ and stacking up residual layers, leads to

$$\boldsymbol{x}_L = \boldsymbol{x}_\ell + \sum_{i=\ell}^{L-1} \mathcal{F}(\boldsymbol{x}_i, \mathcal{W}_i)$$

- Denoting by E the loss function at the NN output, it holds

$$\frac{\partial E}{\partial \boldsymbol{x}_\ell} = \frac{\partial E}{\partial \boldsymbol{x}_L}\frac{\partial \boldsymbol{x}_L}{\partial \boldsymbol{x}_\ell} = \frac{\partial E}{\partial \boldsymbol{x}_L}\left(1 + \frac{\partial}{\partial \boldsymbol{x}_\ell}\sum_{i=\ell}^{L-1}\mathcal{F}(\boldsymbol{x}_i, \mathcal{W}_i)\right)$$

- The gradient is backpropagated directly without concerning any weight layers. The second term instead propagates through the weight layers

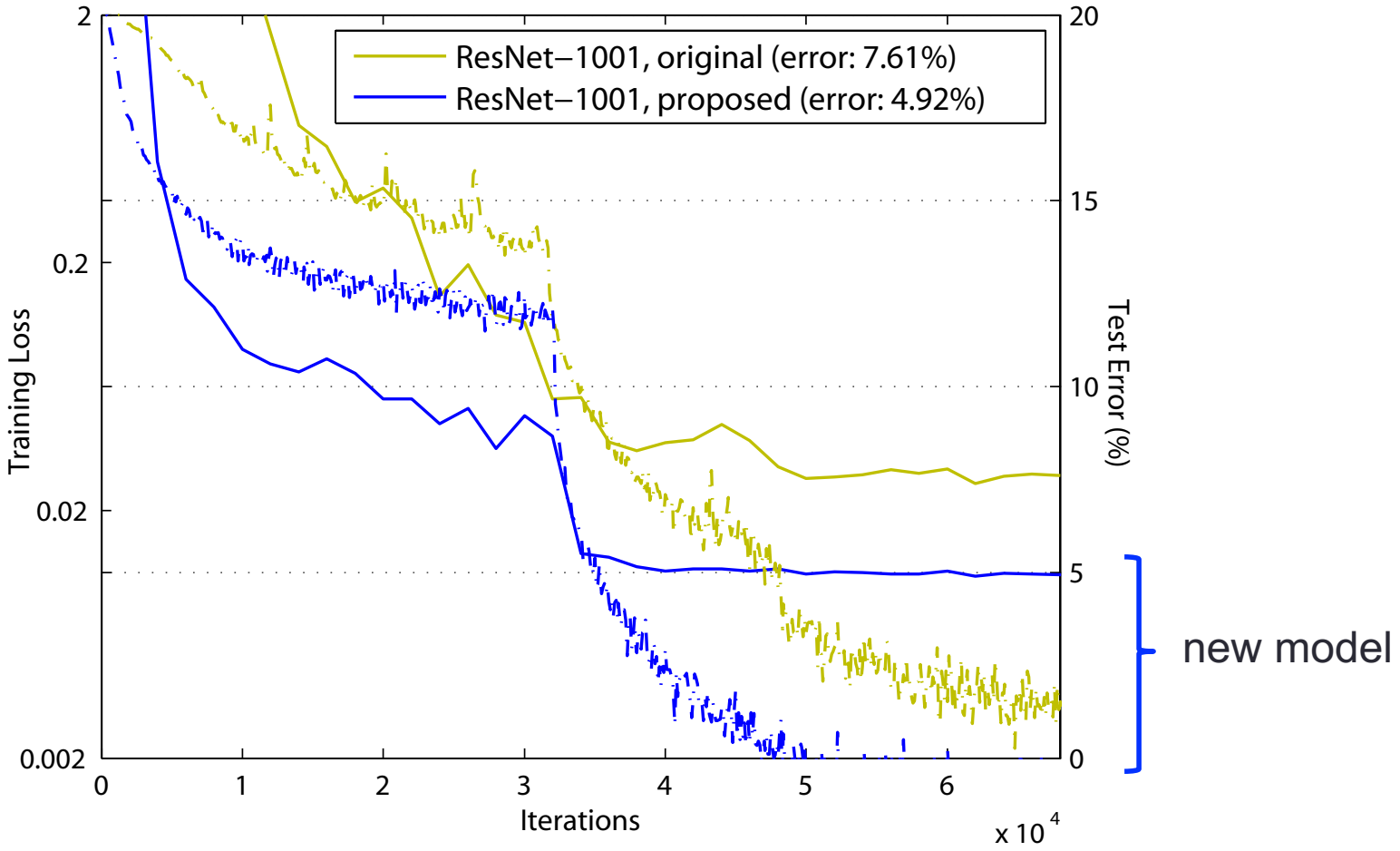
Backpropagating the gradient

$$\frac{\partial E}{\partial \boldsymbol{x}_\ell} = \frac{\partial E}{\partial \boldsymbol{x}_L}\frac{\partial \boldsymbol{x}_L}{\partial \boldsymbol{x}_\ell} = \frac{\partial E}{\partial \boldsymbol{x}_L}\left(1 + \frac{\partial}{\partial \boldsymbol{x}_\ell}\sum_{i=\ell}^{L-1}\mathcal{F}(\boldsymbol{x}_i, \mathcal{W}_i)\right)$$

From the first term

- a direct information flow is establishes in *both* forward and backward directions
- it is unlikely for a minibatch to entirely cancel out the first term, i.e., cannot be always equal to -1 for all samples in a mini-batch
- this implies that the gradient for a layer ℓ does not vanish even when the weights are arbitrarily small

Results on CIFAR-10 [He2016-b]



dashed lines: training loss, solid lines: test error

Master's degree ICT Internet Multimedia Engineering

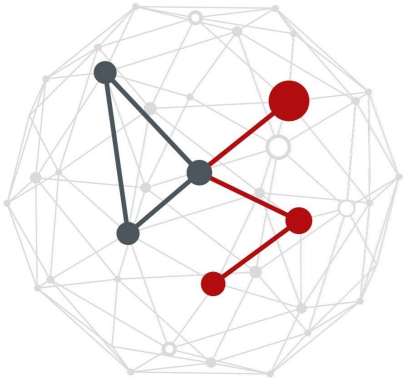
ADVANCED ARCHITECTURES: BATCH NORMALIZATION & RESIDUAL NETWORKS

Michele Rossi

michele.rossi@unipd.it

Dept. of Information Engineering

University of Padova, IT



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO MATEMATICA

800 ANNI 1222 · 2022

UNIVERSITÀ DEGLI STUDI DI PADOVA