ENM 531: Data-driven Modeling and Probabilistic Scientific Computing
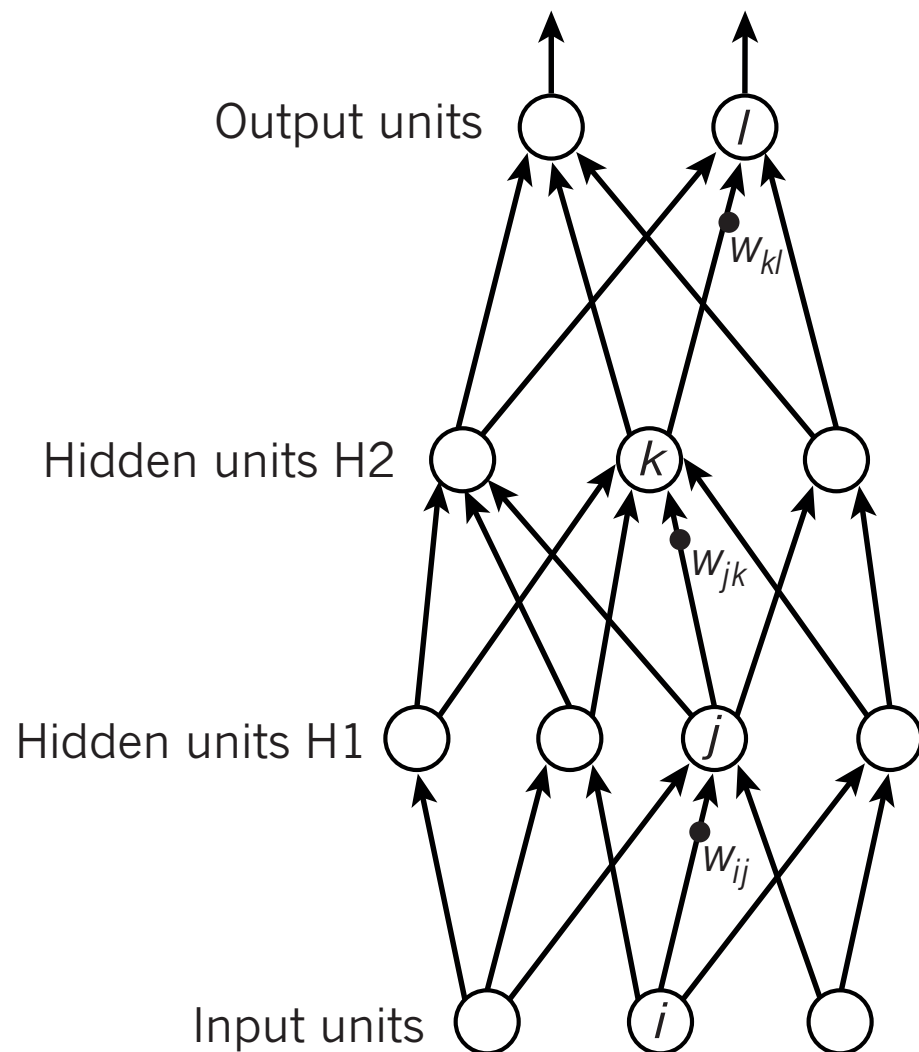
Lecture #16: Neural networks

Paris Perdikaris
March 18, 2021

Back-propagation

Forward pass

Output units

Hidden units H2

Hidden units H1

Input units

$$y_l = f(z_l)$$

$$z_l = \sum_{k \, \varepsilon \, H2} w_{kl} \, y_k$$

$$y_k = f(z_k)$$

$$z_k = \sum_{j \, \varepsilon \, H1} w_{jk} \, y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \, \varepsilon \, Input} w_{ij} \, x_i$$

Backward pass

$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial z_l}$$

$$\frac{\partial E}{\partial y_k} = \sum_{l \, \varepsilon \, out} w_{kl} \frac{\partial E}{\partial z_l}$$

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_k}$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \, \varepsilon \, H2} w_{jk} \frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j}$$
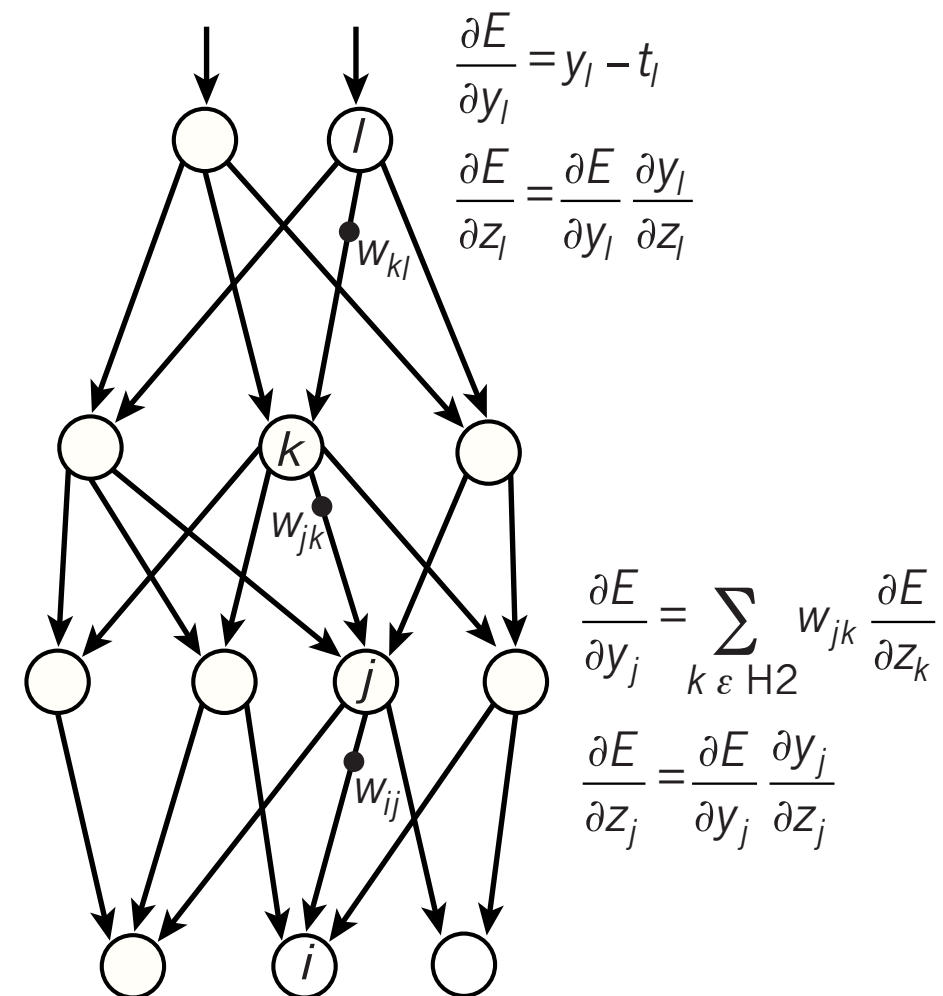
The forward pass

$$F = A_\ell W_\ell + b_\ell, \qquad F \in \mathbb{R}^{n \times p_{\ell+1}}, \qquad A_\ell \in \mathbb{R}^{n \times p_\ell}, \qquad W_\ell \in \mathbb{R}^{p_\ell \times p_{\ell+1}}, \qquad b_\ell \in \mathbb{R}^{1 \times p_{\ell+1}},$$

$$A_\ell = \tanh(H_\ell), \qquad A_\ell \in \mathbb{R}^{n \times p_\ell}, \qquad H_\ell \in \mathbb{R}^{n \times p_\ell},$$

$$H_\ell = A_{\ell-1} W_{\ell-1} + b_{\ell-1}, \qquad H_\ell \in \mathbb{R}^{n \times p_\ell}, \qquad A_{\ell-1} \in \mathbb{R}^{n \times p_{\ell-1}}, \qquad W_{\ell-1} \in \mathbb{R}^{p_{\ell-1} \times p_\ell}, \qquad b_{\ell-1} \in \mathbb{R}^{1 \times p_\ell},$$

$$A_{\ell-1} = \tanh(H_{\ell-1}), \qquad A_{\ell-1} \in \mathbb{R}^{n \times p_{\ell-1}}, \qquad H_{\ell-1} \in \mathbb{R}^{n \times p_{\ell-1}},$$

$$H_{\ell-1} = A_{\ell-2} W_{\ell-2} + b_{\ell-2}, \qquad H_{\ell-1} \in \mathbb{R}^{n \times p_{\ell-1}}, \qquad A_{\ell-2} \in \mathbb{R}^{n \times p_{\ell-2}}, \qquad W_{\ell-2} \in \mathbb{R}^{p_{\ell-2} \times p_{\ell-1}}, \qquad b_{\ell-2} \in \mathbb{R}^{1 \times p_{\ell-1}},$$

$$A_{\ell-2} = \tanh(H_{\ell-2}), \qquad A_{\ell-2} \in \mathbb{R}^{n \times p_{\ell-2}}, \qquad H_{\ell-2} \in \mathbb{R}^{n \times p_{\ell-2}},$$

$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots$$

$$H_2 = A_1 W_1 + b_1, \qquad H_2 \in \mathbb{R}^{n \times p_2}, \qquad A_1 \in \mathbb{R}^{n \times p_1}, \qquad W_1 \in \mathbb{R}^{p_1 \times p_2}, \qquad b_1 \in \mathbb{R}^{1 \times p_2},$$

$$A_1 = \tanh(H_1), \qquad A_1 \in \mathbb{R}^{n \times p_1}, \qquad H_1 \in \mathbb{R}^{n \times p_1},$$

$$H_1 = X W_0 + b_0, \qquad H_1 \in \mathbb{R}^{n \times p_1}, \qquad X \in \mathbb{R}^{n \times p_0}, \qquad W_0 \in \mathbb{R}^{p_0 \times p_1}, \qquad b_0 \in \mathbb{R}^{1 \times p_1},$$
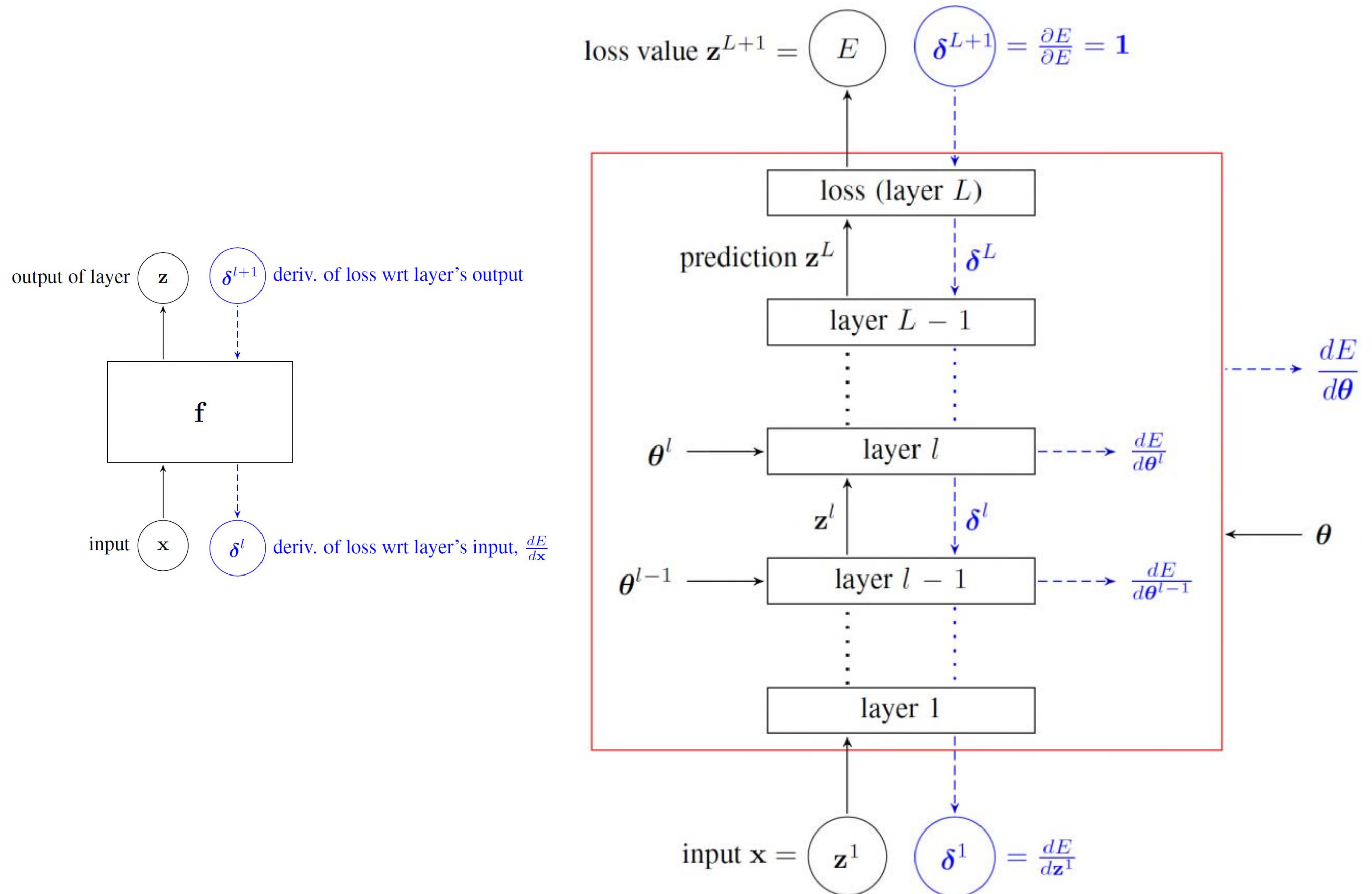
The backward pass

$$\mathcal{L} := \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{p_{\ell+1}} (F_{i,j} - Y_{i,j})^2 \longrightarrow \nabla_\theta \mathcal{L}(\theta)$$

$$G_\ell = F - Y \in \mathbb{R}^{n \times p_{\ell+1}}, \qquad \frac{\partial \mathcal{L}}{\partial W_\ell} = A_\ell^T G_\ell \in \mathbb{R}^{p_\ell \times p_{\ell+1}}, \qquad \frac{\partial \mathcal{L}}{\partial b_\ell} = \mathbb{1}^T G_\ell \in \mathbb{R}^{1 \times p_{\ell+1}},$$

$$G_{\ell-1} = (1 - A_\ell \odot A_\ell) \odot (G_\ell W_\ell^T) \in \mathbb{R}^{n \times p_\ell}, \qquad \frac{\partial \mathcal{L}}{\partial W_{\ell-1}} = A_{\ell-1}^T G_{\ell-1} \in \mathbb{R}^{p_{\ell-1} \times p_\ell}, \qquad \frac{\partial \mathcal{L}}{\partial b_{\ell-1}} = \mathbb{1}^T G_{\ell-1} \in \mathbb{R}^{1 \times p_\ell},$$

$$G_{\ell-2} = (1 - A_{\ell-1} \odot A_{\ell-1}) \odot (G_{\ell-1} W_{\ell-1}^T) \in \mathbb{R}^{n \times p_{\ell-1}}, \quad \frac{\partial \mathcal{L}}{\partial W_{\ell-2}} = A_{\ell-2}^T G_{\ell-2} \in \mathbb{R}^{p_{\ell-2} \times p_{\ell-1}}, \quad \frac{\partial \mathcal{L}}{\partial b_{\ell-2}} = \mathbb{1}^T G_{\ell-2} \in \mathbb{R}^{1 \times p_{\ell-1}},$$

$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$

$$G_1 = (1 - A_2 \odot A_2) \odot (G_2 W_2^T) \in \mathbb{R}^{n \times p_2}, \qquad \frac{\partial \mathcal{L}}{\partial W_1} = A_1^T G_1 \in \mathbb{R}^{p_1 \times p_2}, \qquad \frac{\partial \mathcal{L}}{\partial b_1} = \mathbb{1}^T G_1 \in \mathbb{R}^{1 \times p_2},$$

$$G_0 = (1 - A_1 \odot A_1) \odot (G_1 W_1^T) \in \mathbb{R}^{n \times p_1}, \qquad \frac{\partial \mathcal{L}}{\partial W_0} = X^T G_0 \in \mathbb{R}^{p_0 \times p_1}, \qquad \frac{\partial \mathcal{L}}{\partial b_0} = \mathbb{1}^T G_0 \in \mathbb{R}^{1 \times p_1}.$$
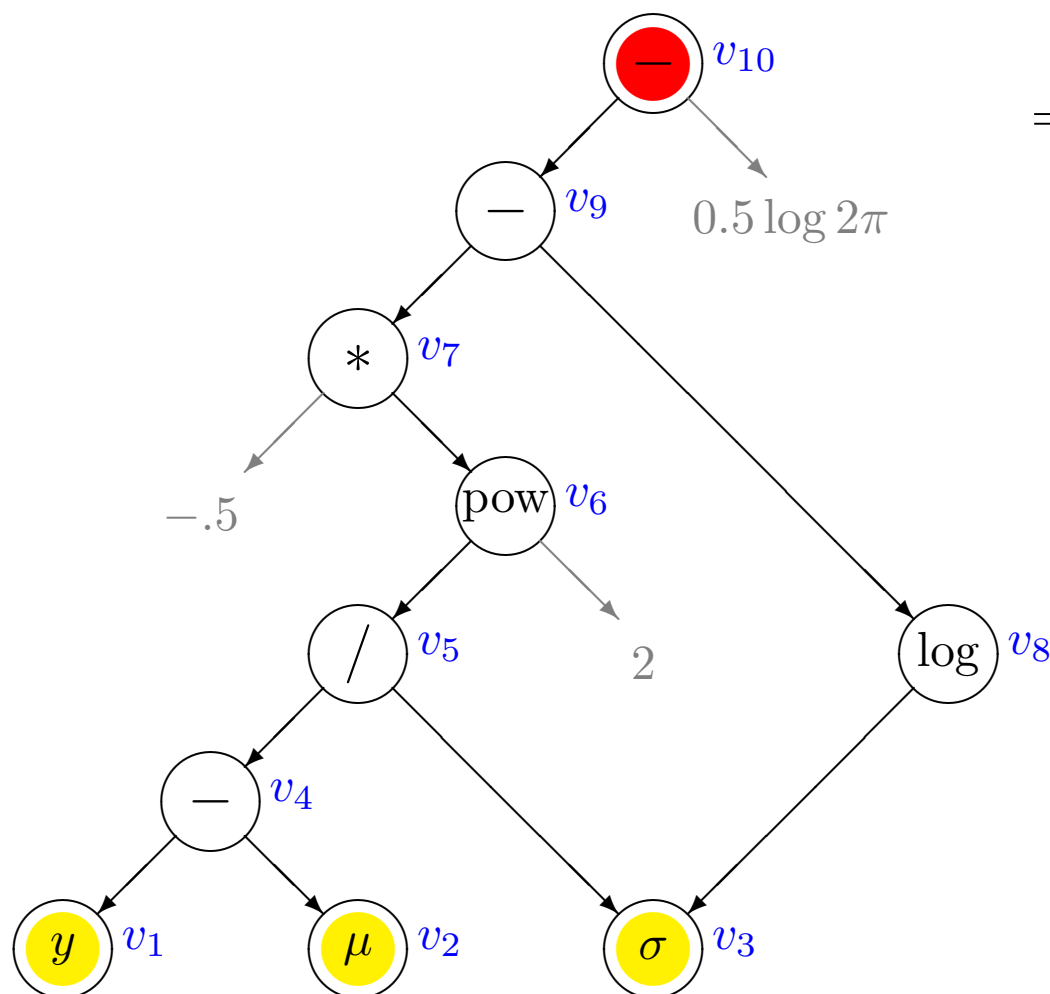
Deep learning: ReLU layer
Modular implementation

loss value $\mathbf{z}^{L+1} = \;\; E$ $\qquad$ $\boldsymbol{\delta}^{L+1} = \frac{\partial E}{\partial E} = \mathbf{1}$

loss (layer L)

prediction $\mathbf{z}^L$ $\qquad$ $\boldsymbol{\delta}^L$

layer $L - 1$

output of layer $\;\; \mathbf{z}$ $\qquad$ $\boldsymbol{\delta}^{l+1}$ deriv. of loss wrt layer's output

$\frac{dE}{d\boldsymbol{\theta}}$

$\mathbf{f}$

$\boldsymbol{\theta}^l \longrightarrow$ layer l $\qquad \dashrightarrow \frac{dE}{d\boldsymbol{\theta}^l}$

$\mathbf{z}^l$ $\qquad$ $\boldsymbol{\delta}^l$

input $\;\; \mathbf{x}$ $\qquad$ $\boldsymbol{\delta}^l$ deriv. of loss wrt layer's input, $\frac{dE}{d\mathbf{x}}$

$\boldsymbol{\theta}^{l-1} \longrightarrow$ layer $l - 1$ $\qquad \dashrightarrow \frac{dE}{d\boldsymbol{\theta}^{l-1}}$

$\boldsymbol{\theta} \longleftarrow$

layer 1

input $\mathbf{x} = \;\; \mathbf{z}^1$ $\qquad$ $\boldsymbol{\delta}^1 = \frac{dE}{d\mathbf{z}^1}$

Automatic differentiation

As an example, consider the log of the normal probability density function for a variable y with a normal distribution with mean μ and standard deviation σ,

$$f(y, \mu, \sigma) = \log\left(\mathsf{Normal}(y|\mu, \sigma)\right) = -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log\sigma - \frac{1}{2}\log(2\pi) \quad (1)$$

var	value	partials	
v_1	y		
v_2	μ		
v_3	σ		
v_4	$v_1 - v_2$	$\partial v_4/\partial v_1 = 1$	$\partial v_4/\partial v_2 = -1$
v_5	v_4/v_3	$\partial v_5/\partial v_4 = 1/v_3$	$\partial v_5/\partial v_3 = -v_4 v_3^{-2}$
v_6	$(v_5)^2$	$\partial v_6/\partial v_5 = 2v_5$	
v_7	$(-0.5)v_6$	$\partial v_7/\partial v_6 = -0.5$	
v_8	$\log v_3$	$\partial v_8/\partial v_3 = 1/v_3$	
v_9	$v_7 - v_8$	$\partial v_9/\partial v_7 = 1$	$\partial v_9/\partial v_8 = -1$
v_{10}	$v_9 - (0.5\log 2\pi)$	$\partial v_{10}/\partial v_9 = 1$	

Carpenter, B., Hoffman, M. D., Brubaker, M., Lee, D., Li, P., & Betancourt, M. (2015). The stan math library: Reverse-mode automatic differentiation in C++. arXiv preprint arXiv:1509.07164.

Overfitting
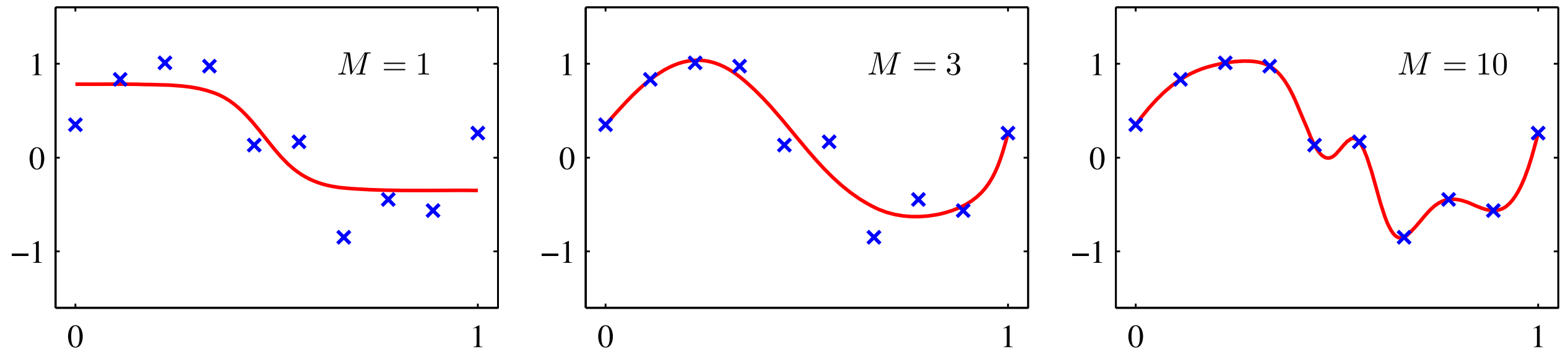
Figure 5.9   Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set. The graphs show the result of fitting networks having $M = 1, 3$ and 10 hidden units, respectively, by minimizing a sum-of-squares error function using a scaled conjugate-gradient algorithm.
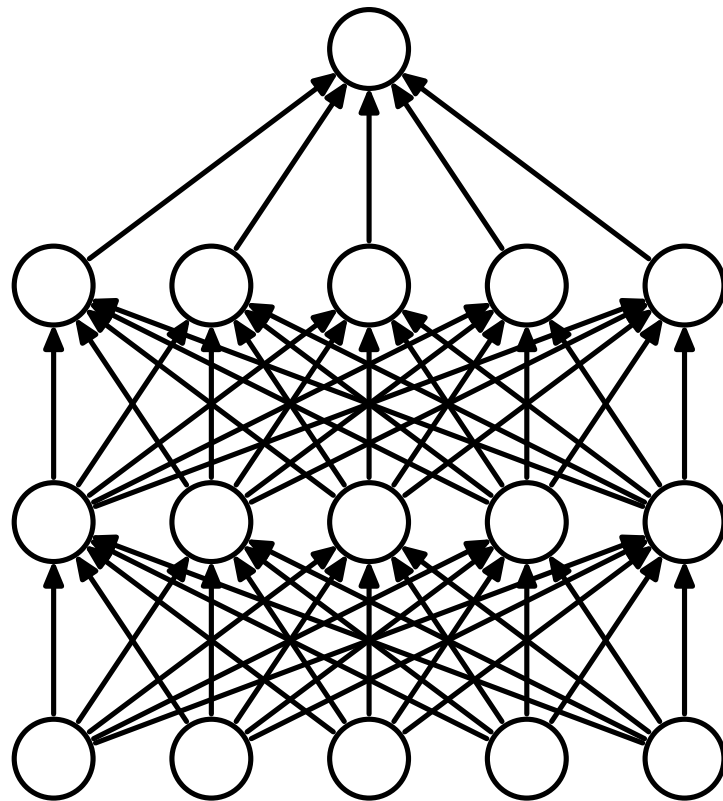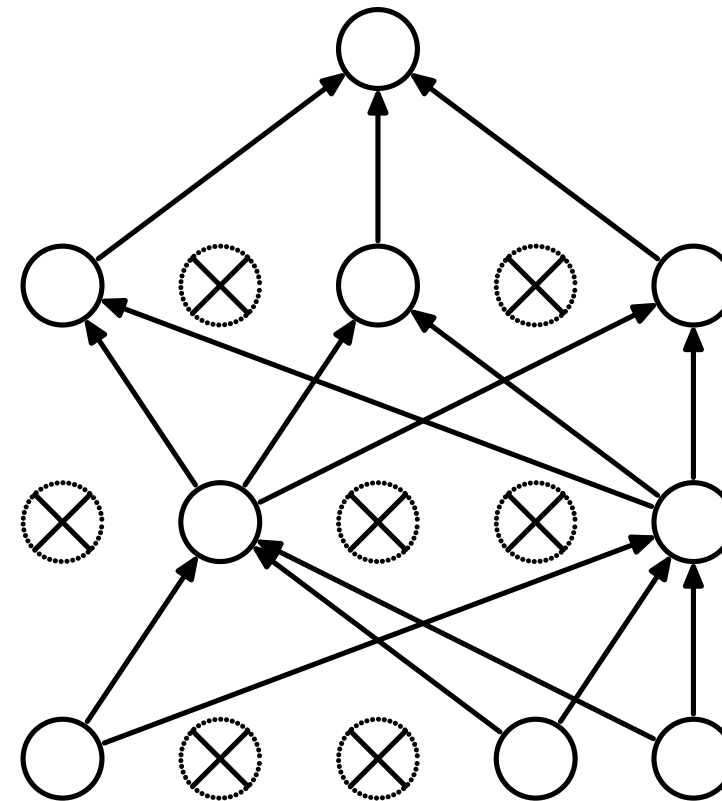
Overfitting
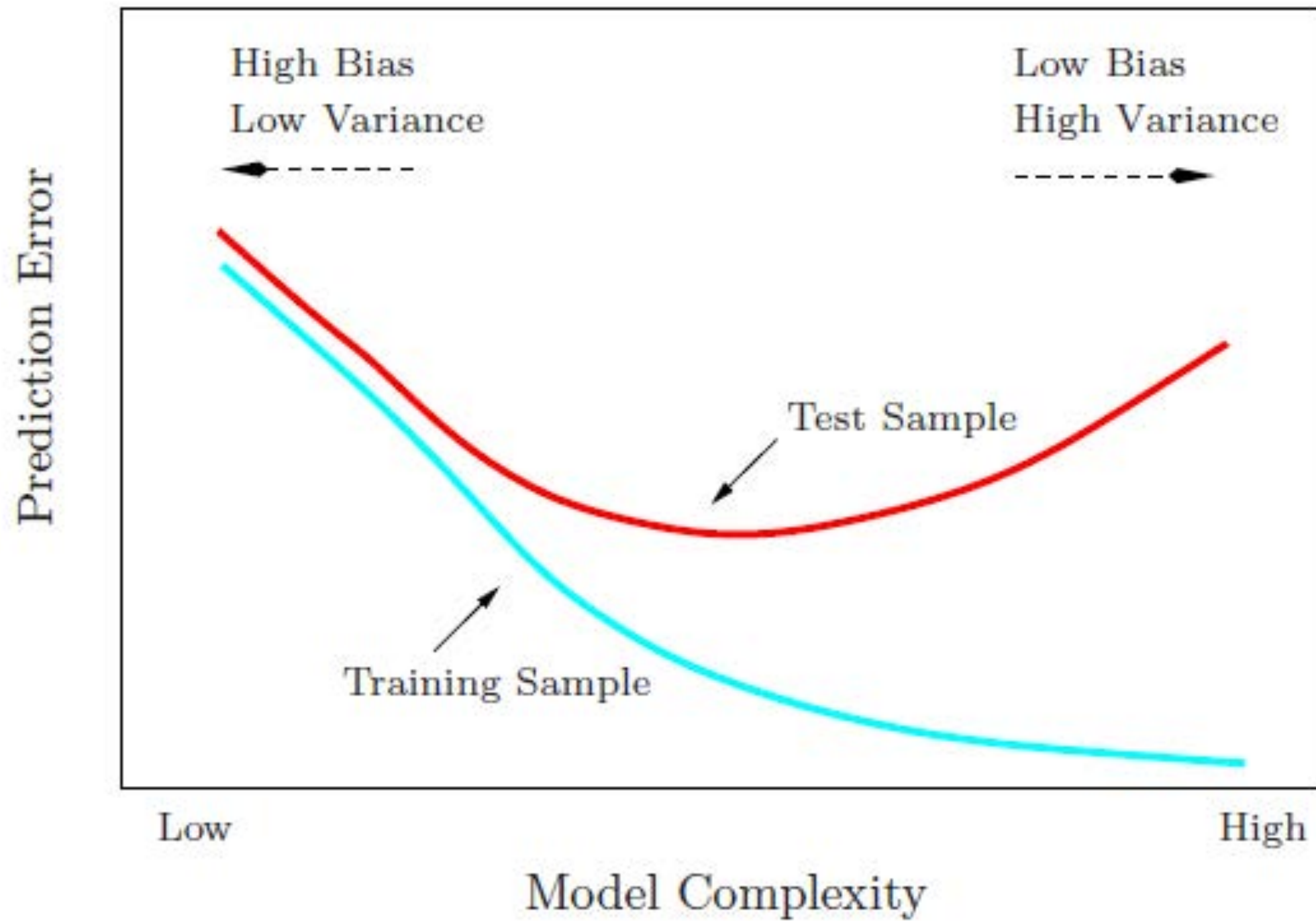
Early stopping
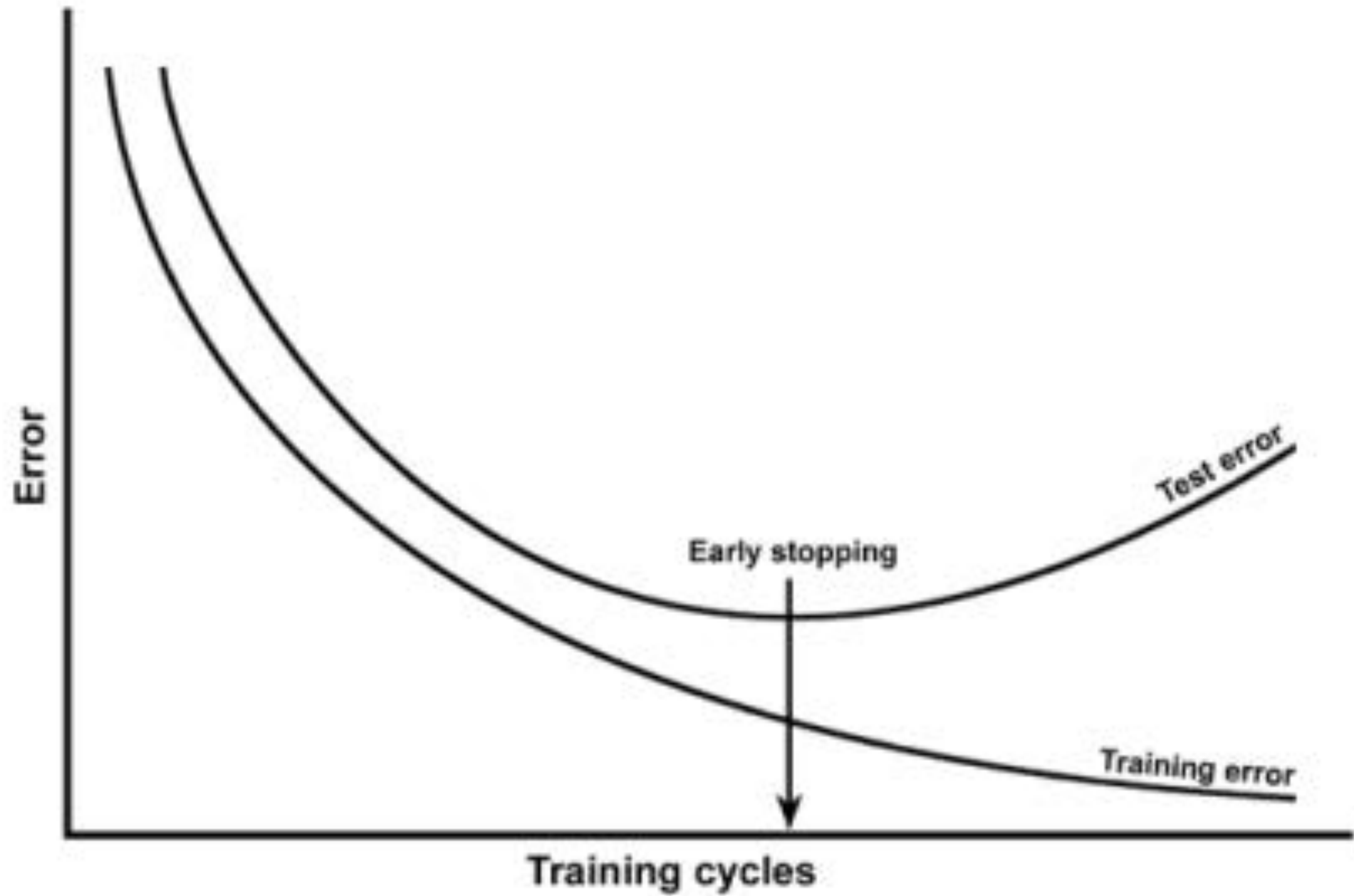
Dropout

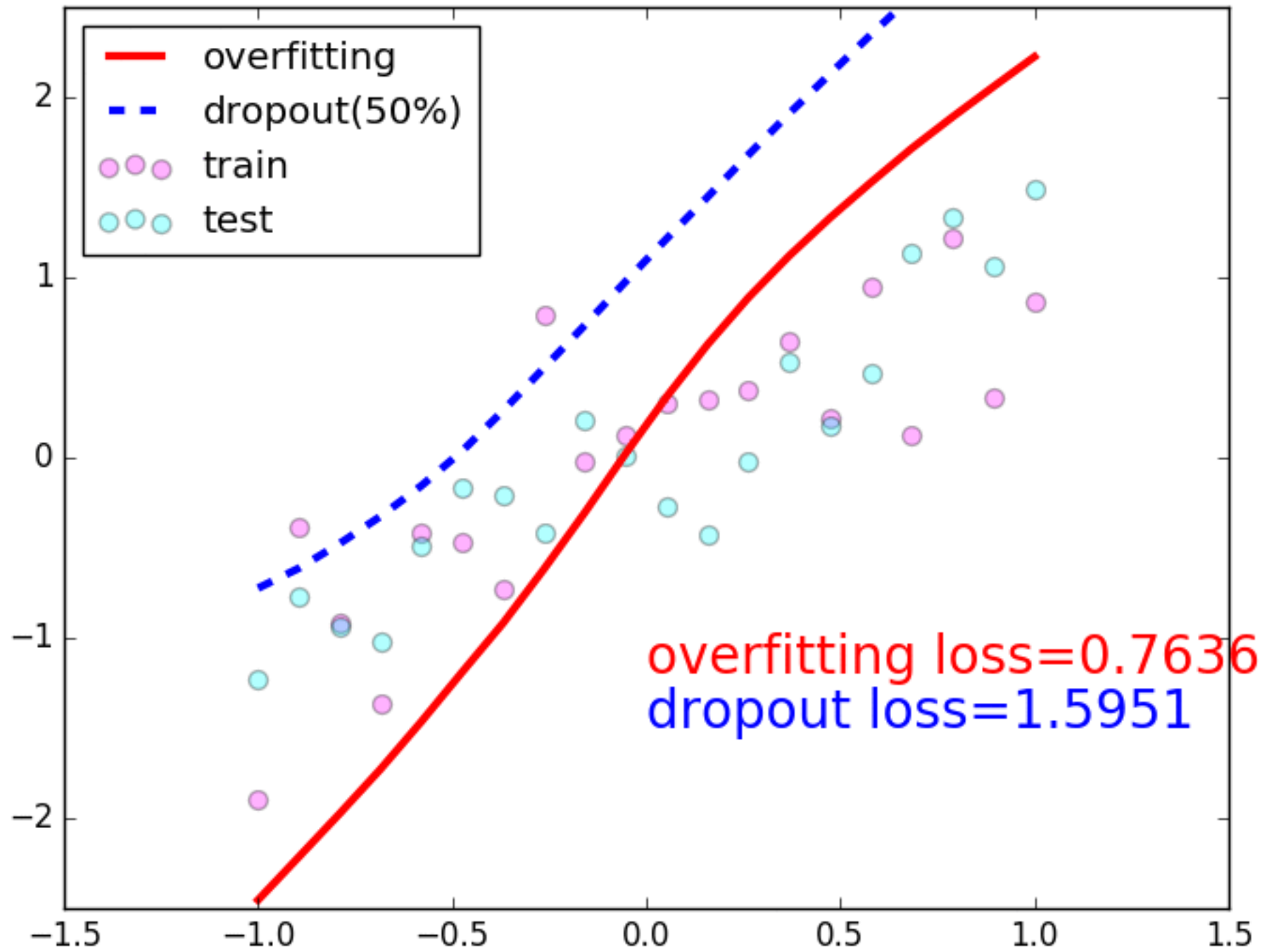(a) Standard Neural Net

(b) After applying dropout.

With probability keep_prob, outputs the input element scaled up by 1 / keep_prob, otherwise outputs 0. The scaling is so that the expected sum is unchanged.

```
for W, b in params:
        outputs = np.dot(inputs, W) + b
        inputs = np.tanh(outputs)
        if dropout_train: inputs *= np.random.binomial([np.ones_like(inputs)],(1-
   keep_prob))[0]/(1-keep_prob)
```

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

Gal, Y., & Ghahramani, Z. (2016, June). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning (pp. 1050-1059).

Dropout

- overfitting
- dropout(50%)
- train
- test

overfitting loss=0.7636
dropout loss=1.5951

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot   **Yoshua Bengio**

DIRO, Université de Montréal, Montréal, Québec, Canada

Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent relative successes and help design better algorithms in the future. We first observe the influence of the non-linear activations functions. We find that the logistic sigmoid activation is unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation. Surprisingly, we find that saturated units can move out of saturation by themselves, albeit slowly, and explaining the plateaus sometimes seen when training neural networks. We find that a new non-linearity that saturates less can often be beneficial. Finally, we study how activations and gradients vary across layers and during train- ing, with the idea that training may be more difficult when the singular values of the Jacobian associated with each layer are far from 1. Based on these considerations, we propose a new initialization scheme that brings substantially faster convergence.
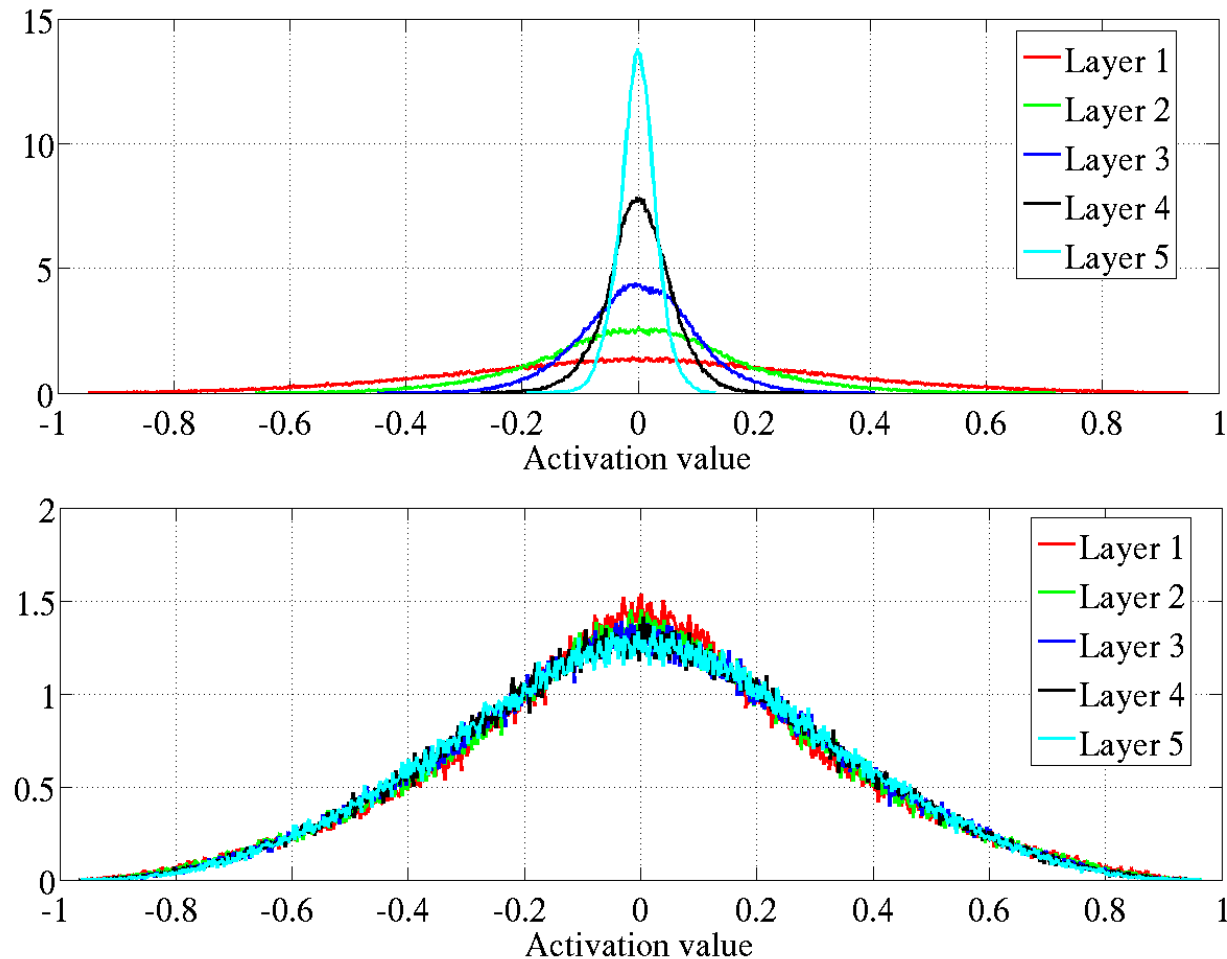
Network initialization

Figure 6: *Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.*
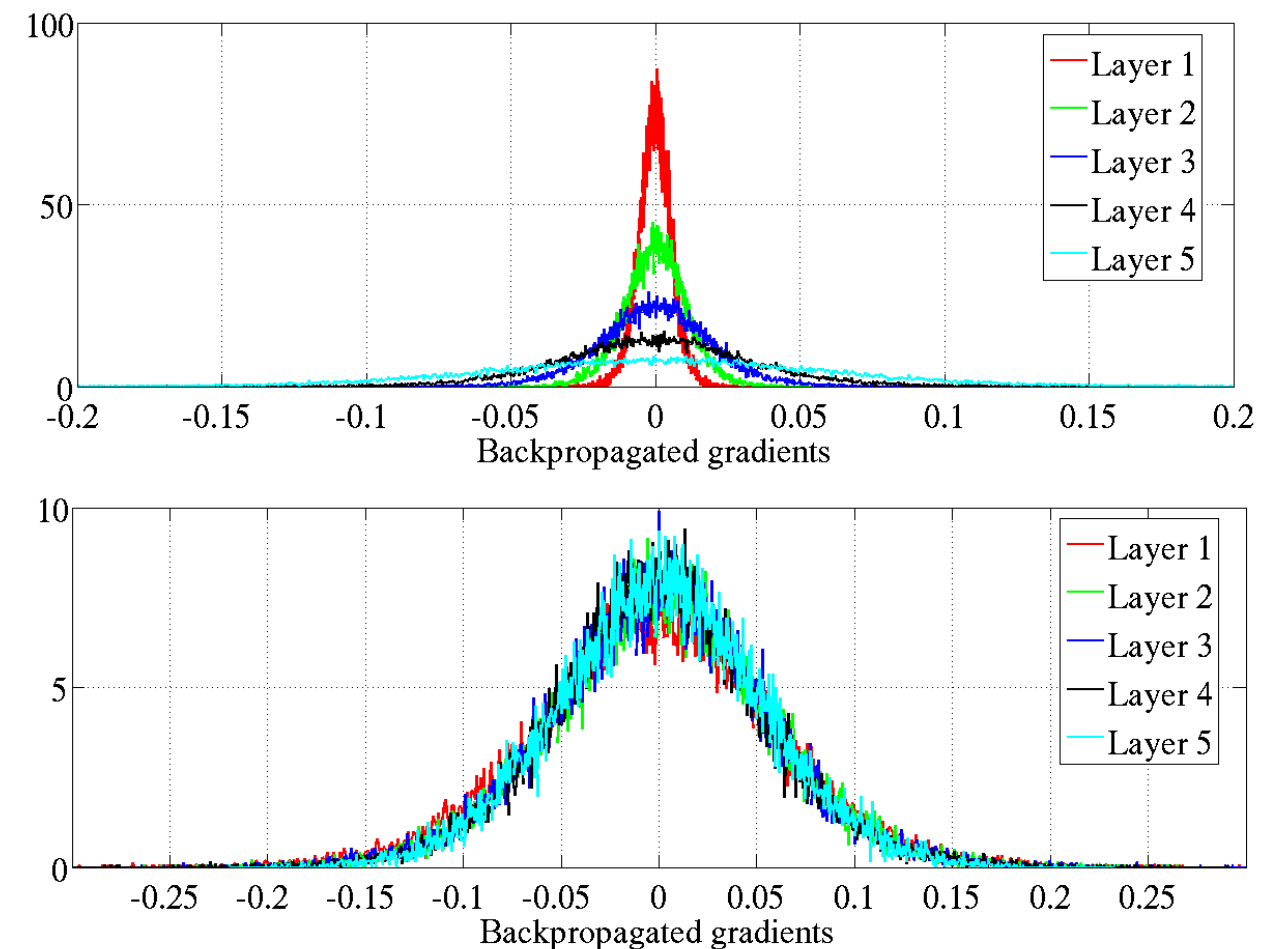
Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (pp. 249-256).

Efficient BackProp

Yann LeCun[1], Leon Bottou[1], Genevieve B. Orr[2], and Klaus-Robert Müller[3]

[1] Image Processing Research Department AT& T Labs - Research, 100 Schulz Drive, Red Bank, NJ 07701-7033, USA
[2] Willamette University, 900 State Street, Salem, OR 97301, USA
[3] GMD FIRST, Rudower Chaussee 5, 12489 Berlin, Germany
{yann,leonb}@research.att.com, gorr@willamette.edu, klaus@first.gmd.de

Abstract. The convergence of back-propagation learning is analyzed so as to explain common phenomenon observed by practitioners. Many undesirable behaviors of backprop can be avoided with tricks that are rarely exposed in serious technical publications. This paper gives some of those tricks, and offers explanations of why they work.
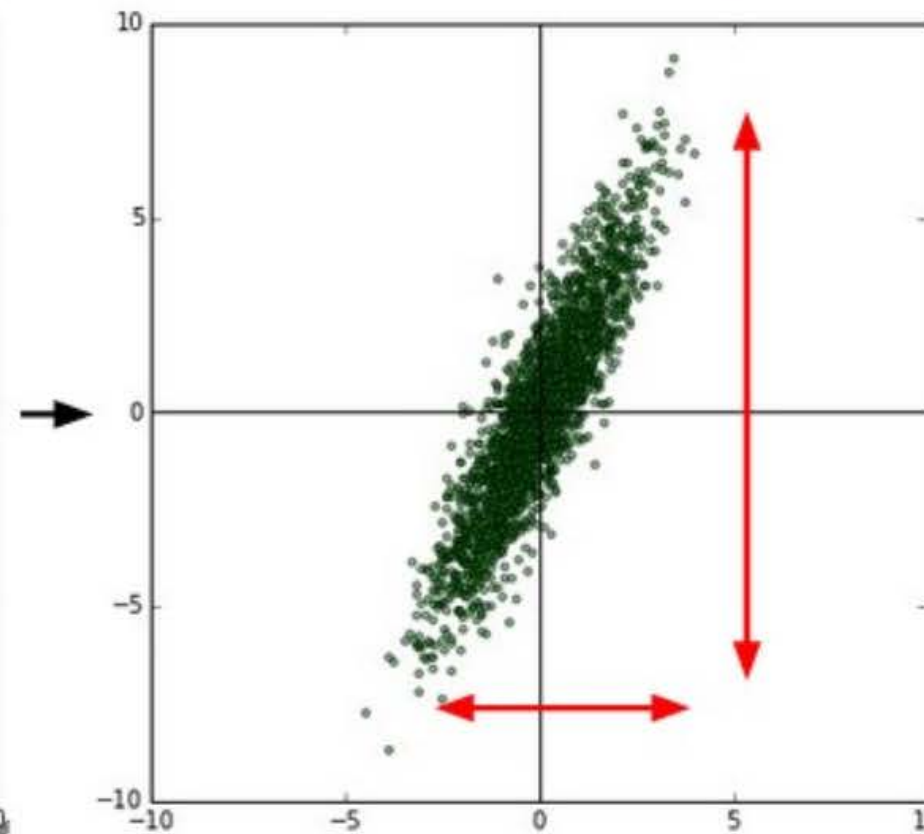
Many authors have suggested that second-order optimization methods are advantageous for neural net training. It is shown that most "classical" second-order methods are impractical for large neural networks. A few methods are proposed that do not have these limitations.
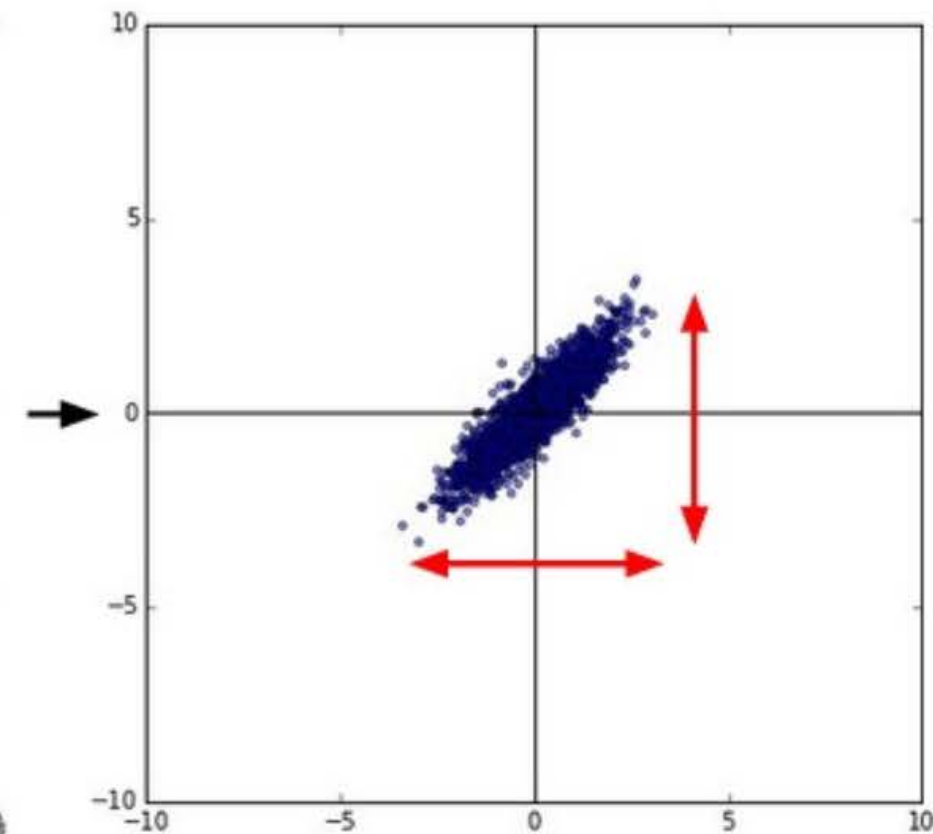
Normalizing the inputs
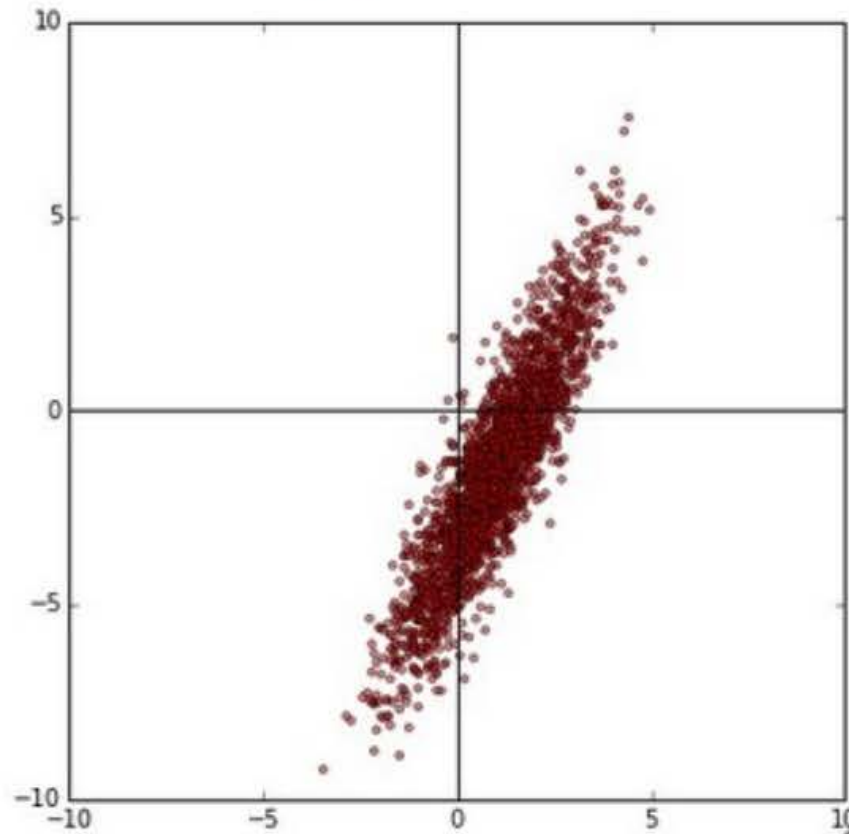
original data   zero-centered data   normalized data

Common data preprocessing pipeline. **Left**: Original toy, 2-dimensional input data. **Middle**: The data is zero-centered by subtracting the mean in each dimension. The data cloud is now centered around the origin. **Right**: Each dimension is additionally scaled by its standard deviation. The red lines indicate the extent of the data - they are of unequal length in the middle, but of equal length on the right.

https://zaffnet.github.io/batch-normalization

Batch normalization

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

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covaria shift. In International conference on machine learning (pp. 448-456).

https://zaffnet.github.io/batch-normalization

Batch normalization

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^m \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^m \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^m -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$
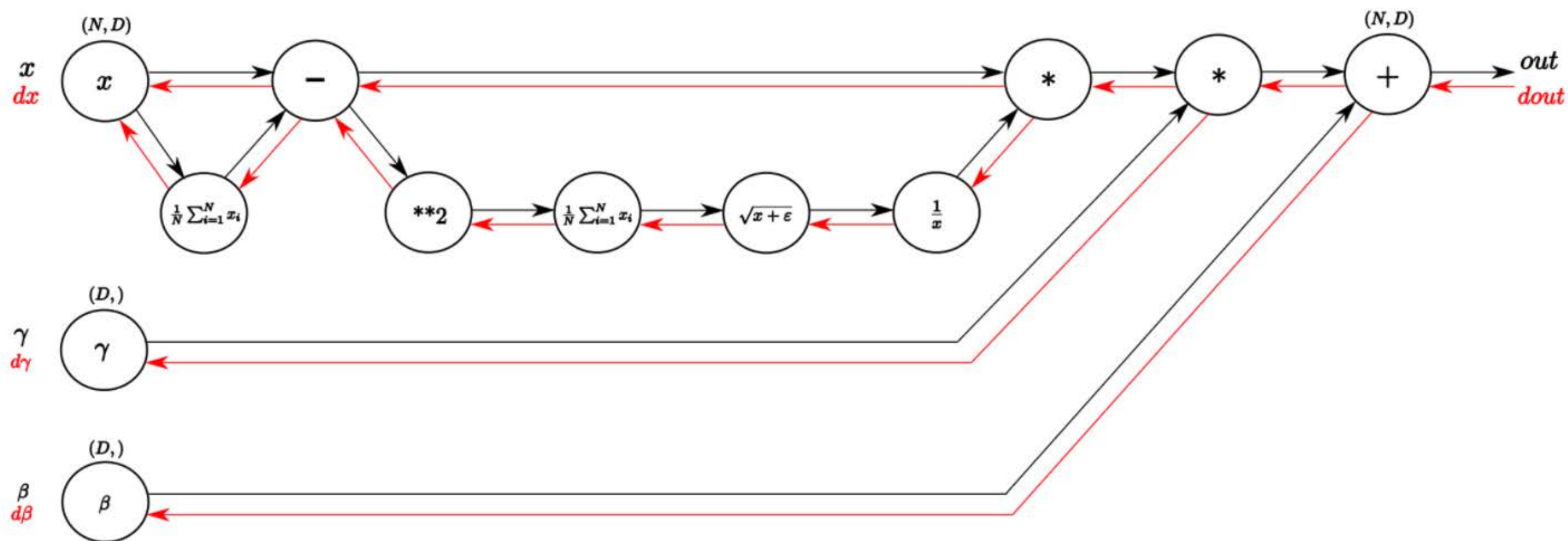
$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i}$$

Nowadays we can get the backward pass for free using automatic differentiation!

Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covaria shift. In International conference on machine learning (pp. 448-456).

Batch normalization

Just a reminder for when you're doing mini-batch SGD in deep networks…

Computational graph of the BatchNorm-Layer. From left to right, following the black arrows flows the forward pass. The inputs are a matrix X and gamma and beta as vectors. From right to left, following the red arrows flows the backward pass which distributes the gradient from above layer to gamma and beta and all the way back to the input.

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).