Neural Networks

Mingfei Sun

Foundations of Machine Learning
The University of Manchester

A quick recap

In the previous lecture, we've covered:

- Basics of vector and matrix calculus
- Logistic regression model: a linear classifier
- Stochastic gradient descent methods

Outline

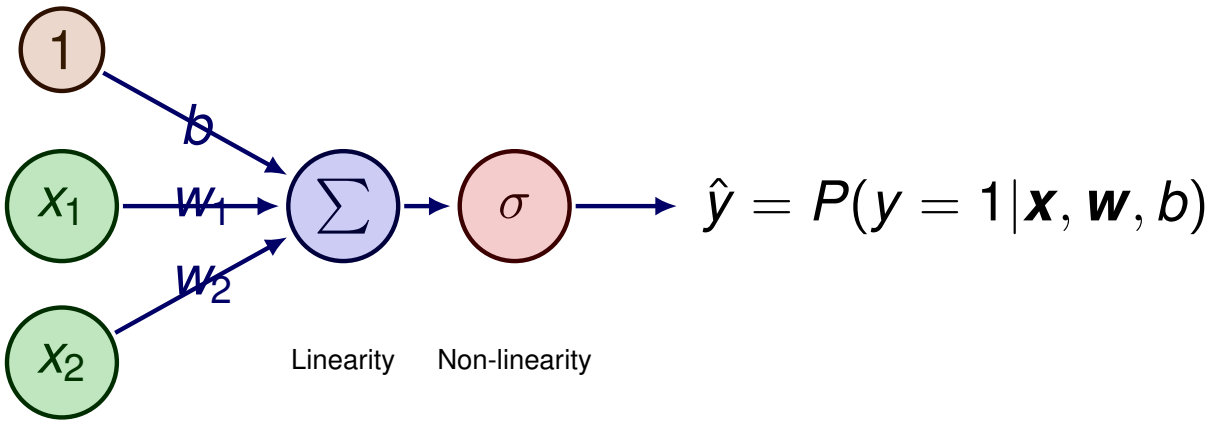
Intended Learning Outcomes

By the end of this lecture, you'll be able to

- Identify the key components of feedforward neural networks
- Explain how the feedforward neural networks can be effectively optimized
- Apply variants of SGD methods to neural parameter update
- Implement convolutional neural networks and SGD/RMSProp/Adam in practice

Outline

Logistic Regression (LR)



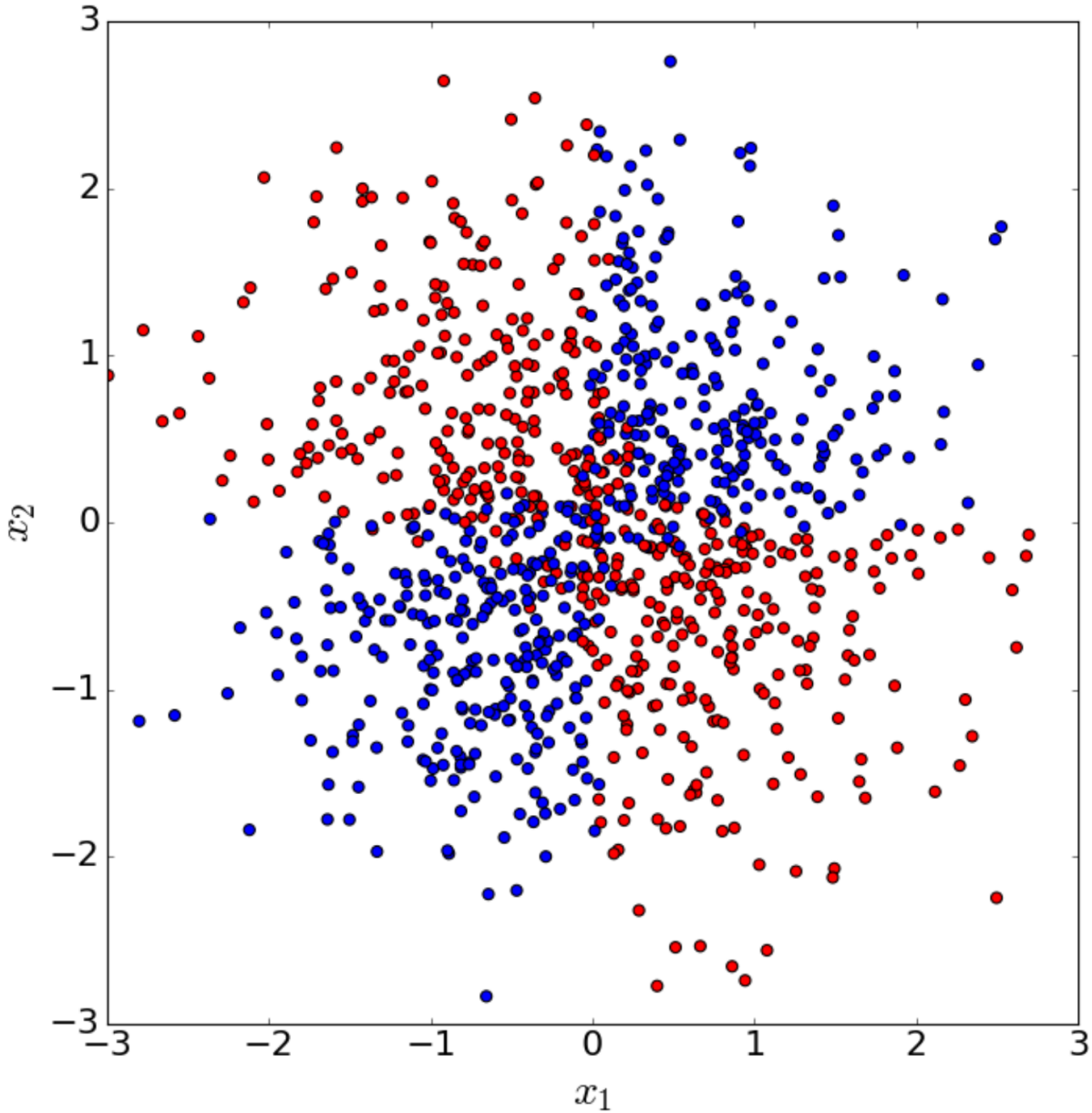
inputs

- A unit in a neural network computes a linear function of its input and is then composed with a non-linear *activation* function
- For logistic regression, the non-linear activation is *sigmoid*

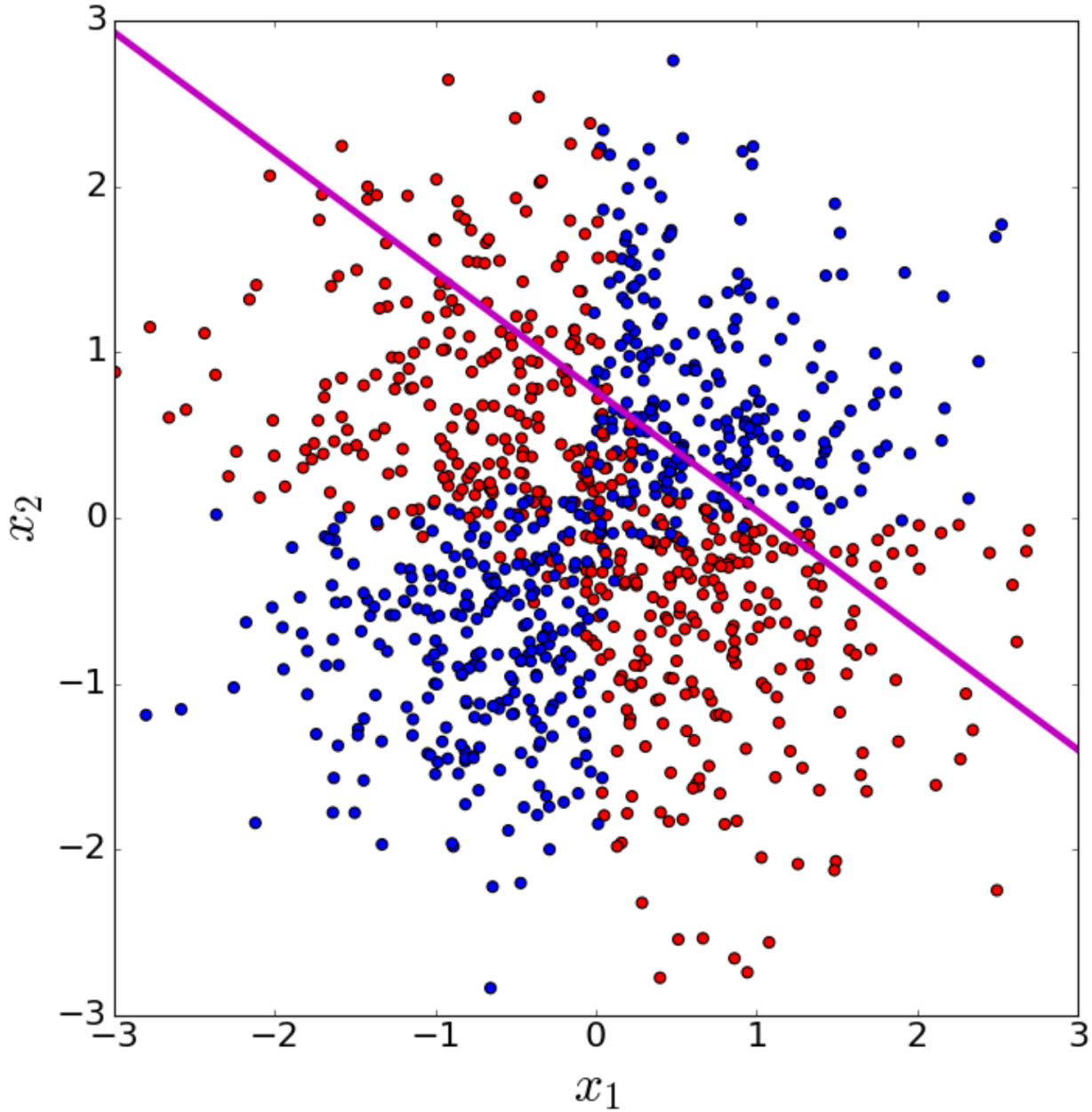
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- The separating surface is linear

A toy example



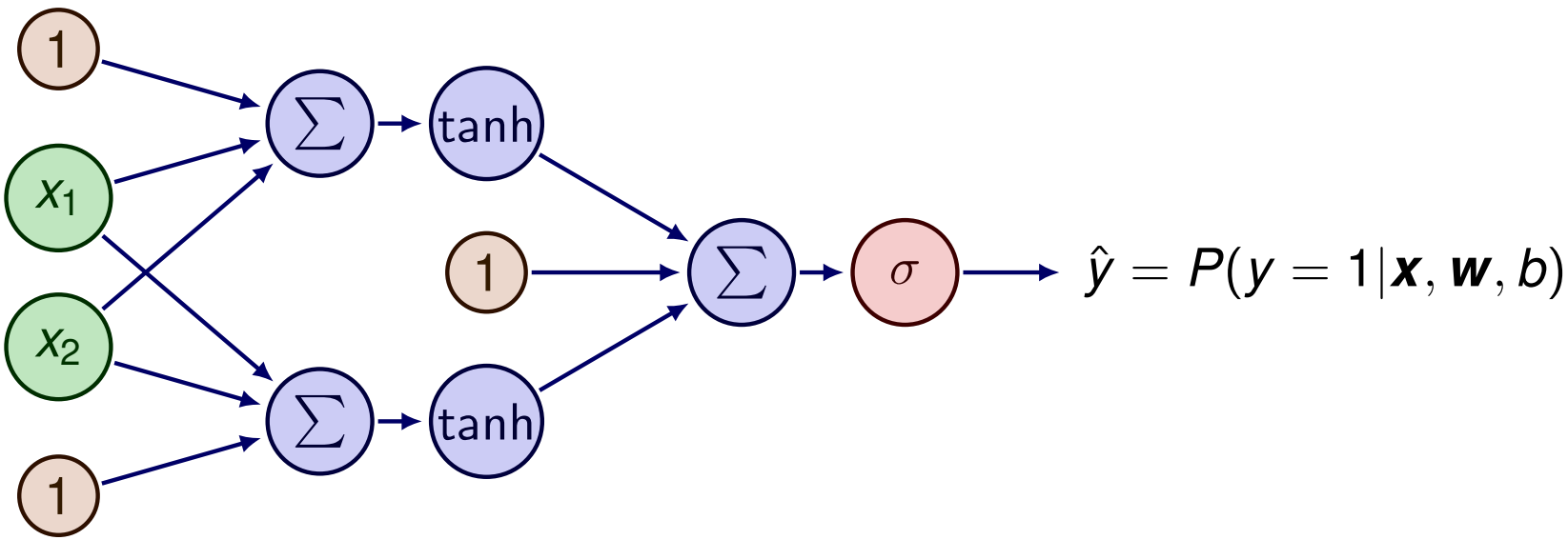
Logistic regression fails badly



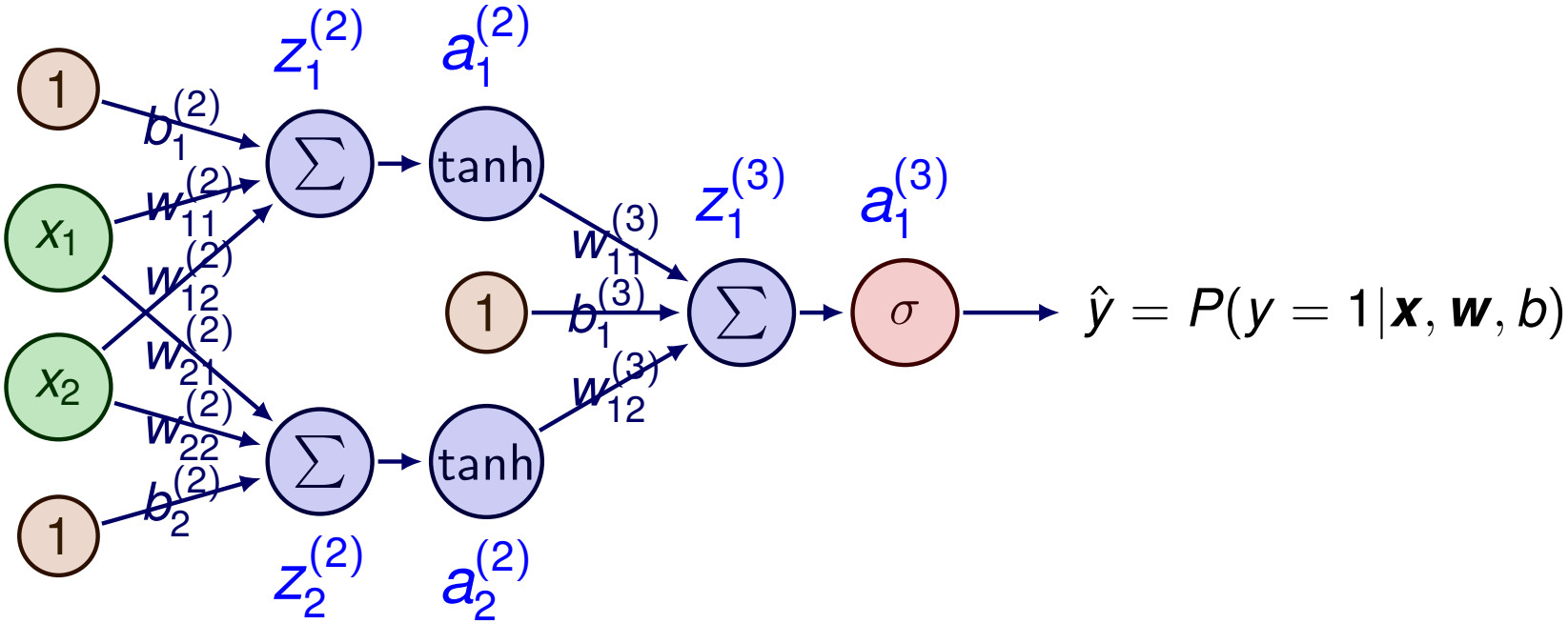
Multi-Layer Perceptron (MLP)

Extend LR model to have one more layer (3 layers in total):

- *Input layer*
- *Hidden layer*
- *Output layer*



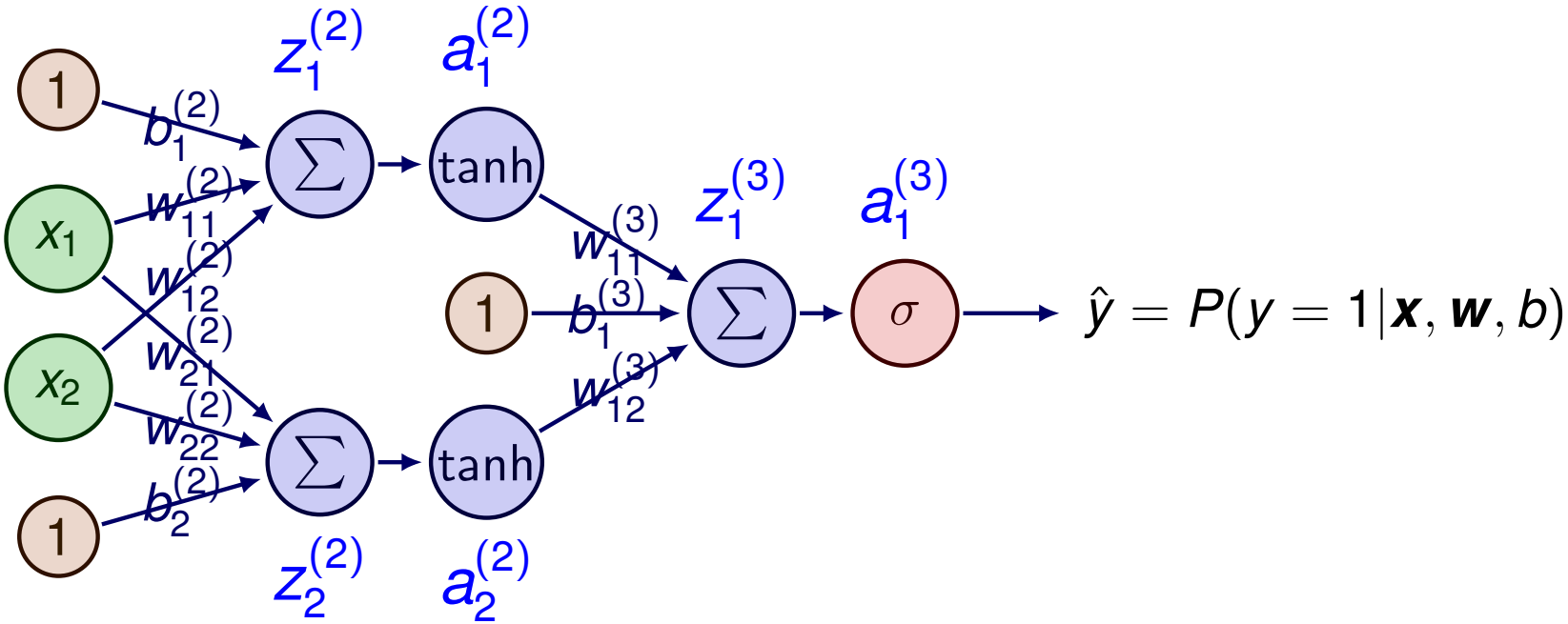
Solve using MLP



Notations:

- $w_{ij}^{(k)}$ ⟵ Superscript: k-th layer
  ⟵ Subscript: link from unit j to unit i
- $z_i^{(k)}$ ⟵ Superscript: k-th layer
  ⟵ Subscript: unit i
- $a_i^{(k)}$ ⟵ Superscript: k-th layer
  ⟵ Subscript: unit i

Solve using MLP



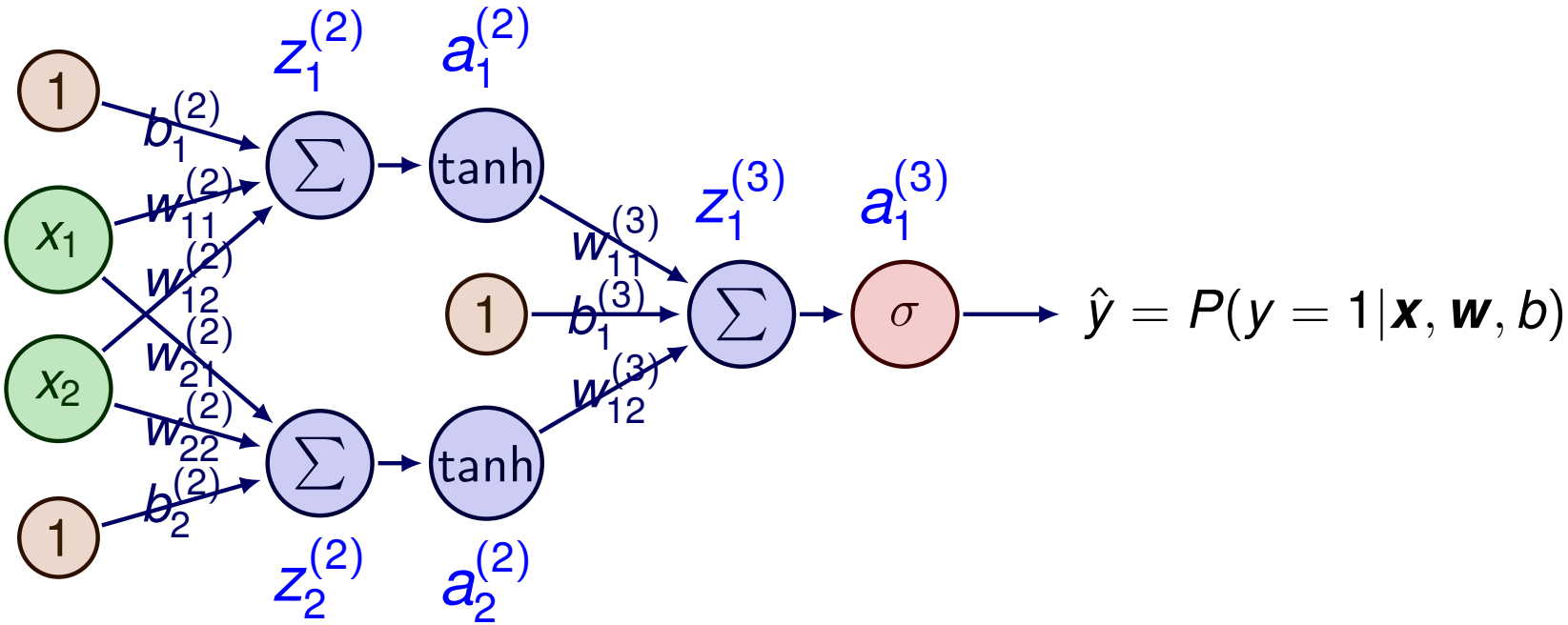
Vector and matrix notation:

$$\boldsymbol{x}^{(1)} = [x_1, x_2]^\top \qquad \boldsymbol{b}^{(2)} = [b_1^{(2)}, b_2^{(2)}]^\top$$

$$\boldsymbol{z}^{(2)} = [z_1^{(2)}, z_2^{(2)}]^\top \qquad \boldsymbol{a}^{(2)} = [a_1^{(2)}, a_1^{(2)}]^\top$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{11}^{(2)}, w_{12}^{(2)} \\ w_{21}^{(2)}, w_{22}^{(2)} \end{bmatrix} \qquad \boldsymbol{w}^{(3)} = [w_{11}^{(3)}, w_{12}^{(3)}]^\top$$

Solve using MLP



Vector and matrix notation:

$$\boldsymbol{x}^{(1)} = [x_1, x_2]^\top \qquad \boldsymbol{b}^{(2)} = [b_1^{(2)}, b_2^{(2)}]^\top$$

$$\boldsymbol{z}^{(2)} = [z_1^{(2)}, z_2^{(2)}]^\top \qquad \boldsymbol{a}^{(2)} = [a_1^{(2)}, a_1^{(2)}]^\top$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{11}^{(2)}, w_{12}^{(2)} \\ w_{21}^{(2)}, w_{22}^{(2)} \end{bmatrix} \qquad \boldsymbol{w}^{(3)} = [w_{11}^{(3)}, w_{12}^{(3)}]^\top$$

Let us use the notation:

$$\boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{x}^{(1)} + \boldsymbol{b}^{(2)}$$

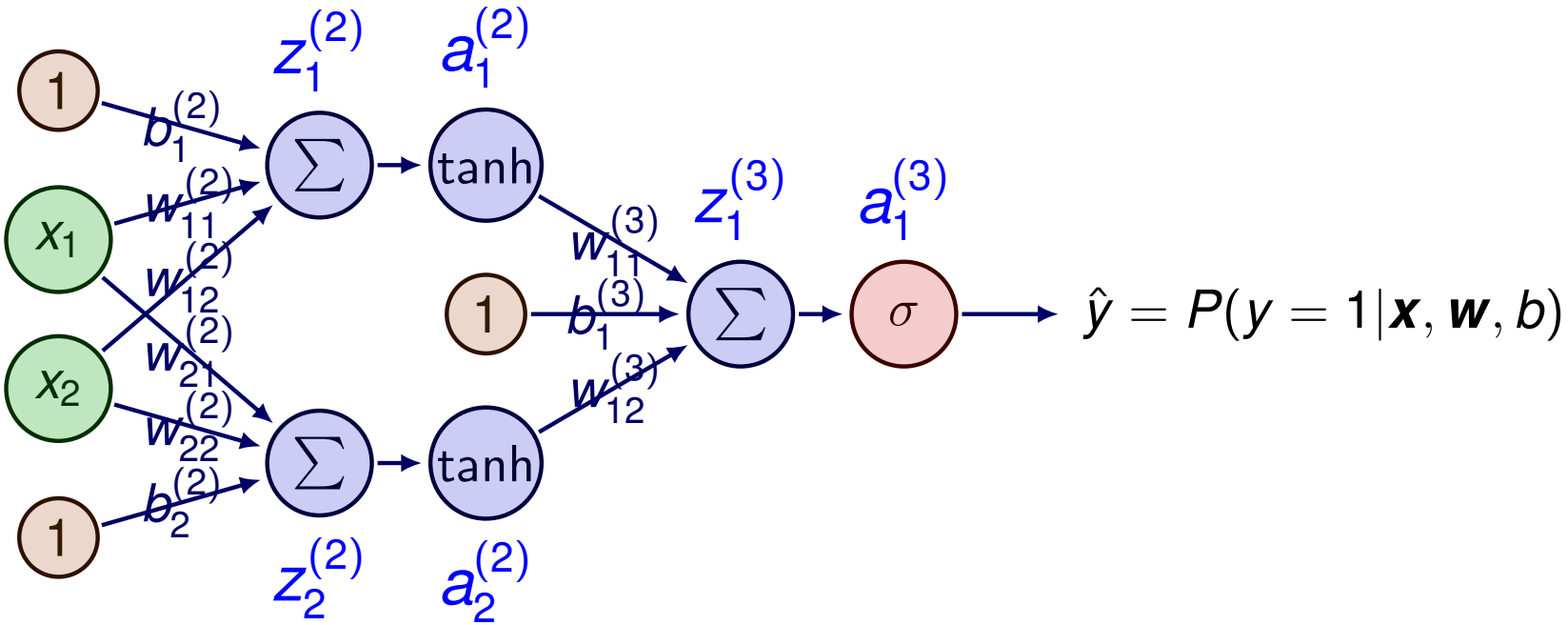
$$\boldsymbol{a}^{(2)} = \tanh(\boldsymbol{z}^{(2)})$$

$$z_1^{(3)} = \boldsymbol{w}^{(3)\top}\boldsymbol{a}^{(2)} + b_1^{(3)}$$

$$\hat{y} = a_1^{(3)} = \sigma(z_1^{(3)})$$

Compute gradients on a toy example (1)

To minimize the error function, we need to calculate the derivatives.



To minimize an error function of the form:

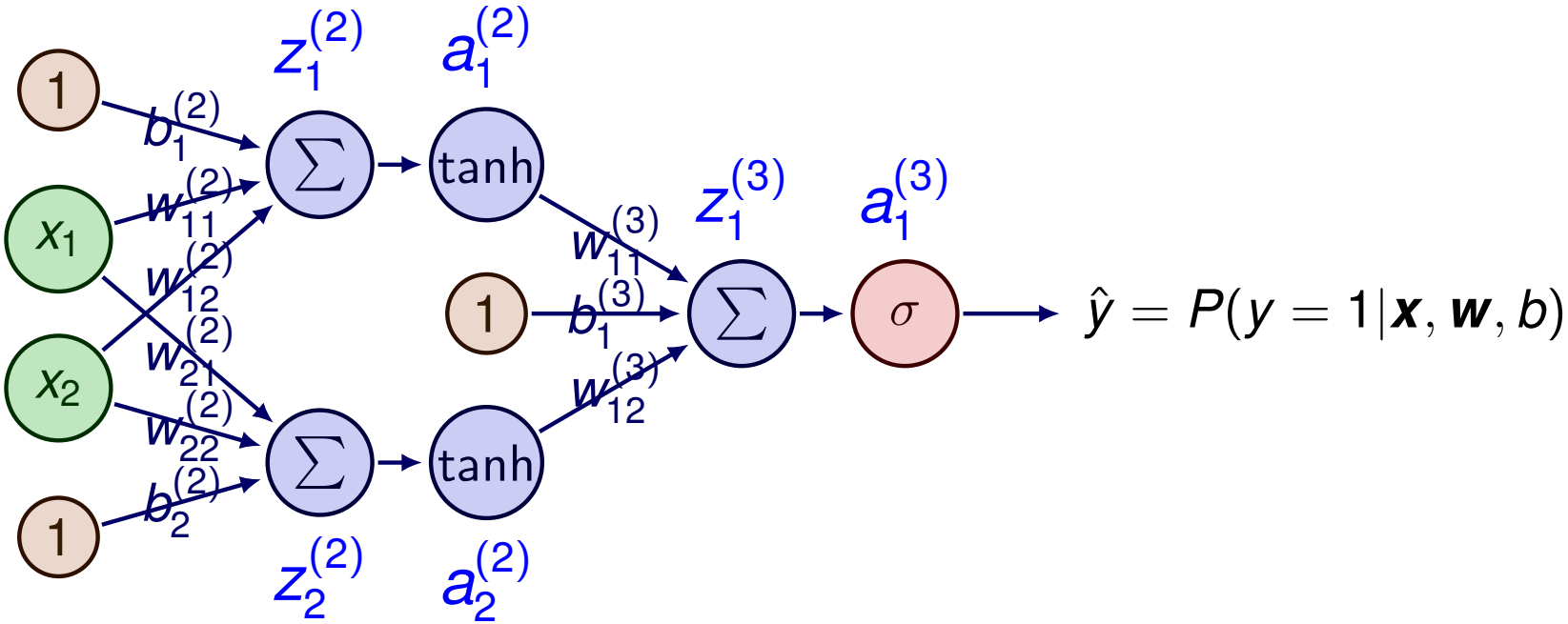
$$E(\underbrace{\boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{W}^{(3)}, b_1^{(3)}}_{\text{Differentiable parameters}}; \mathcal{D}) = \sum_{n=1}^{N} E_n(\boldsymbol{x}_n, y_n; \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{W}^{(3)}, b_1^{(3)}),$$

Need derivatives: $\frac{\partial E}{\partial w_{11}^{(2)}}, \frac{\partial E}{\partial w_{12}^{(2)}}, \frac{\partial E}{\partial w_{21}^{(2)}}, \frac{\partial E}{\partial w_{22}^{(2)}}, \frac{\partial E}{\partial w_{11}^{(3)}}, \frac{\partial E}{\partial w_{12}^{(3)}}, \frac{\partial E}{\partial b_1^{(2)}}, \frac{\partial E}{\partial b_2^{(2)}}, \frac{\partial E}{\partial b_1^{(3)}}$.

Compute gradients on a toy example (2)

For classification problems, such as this, we can use the cross-entropy error function (for a single data point),

$$E_n(\boldsymbol{x}_n, y_n; \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}, \boldsymbol{w}^{(3)}, b_1^{(3)}) = -\left(y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)\right)$$



According to the chain rule, it suffices to compute the following:

$$\frac{\partial E_n}{\partial a_1^{(3)}} \qquad \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \qquad \frac{\partial z_1^{(3)}}{\partial \boldsymbol{a}^{(2)}} \qquad \frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(2)}}$$

Compute gradients on a toy example (3)

Let us compute the following:

1. $\frac{\partial E_n}{\partial a_1^{(3)}} = -\frac{y_n}{a_1^{(3)}} + \frac{1-y_n}{1-a_1^{(3)}} = \frac{a_1^{(3)} - y_n}{a_1^{(3)}(1-a_1^{(3)})}$

2. $\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = a_1^{(3)}(1 - a_1^{(3)}) \quad \longleftarrow \boxed{\sigma'(x) = \sigma(x)\,(1 - \sigma(x))}$

3. $\frac{\partial z_1^{(3)}}{\partial \boldsymbol{a}^{(2)}} = [w_{11}^{(3)}, w_{12}^{(3)}] \quad \longleftarrow (\textit{scalar-by-vector})$

4. $\frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(2)}} = \underbrace{\begin{bmatrix} 1 - \tanh^2(z_1^{(2)}) & 0 \\ 0 & 1 - \tanh^2(z_2^{(2)}) \end{bmatrix}}_{\boxed{\tanh(x)' = 1 - \tanh^2(x)}} \longleftarrow (\textit{vector-by-vector})$

Compute gradients on a toy example (3)

Let us compute the following:

1. $\frac{\partial E_n}{\partial a_1^{(3)}} = -\frac{y_n}{a_1^{(3)}} + \frac{1-y_n}{1-a_1^{(3)}} = \frac{a_1^{(3)} - y_n}{a_1^{(3)}(1-a_1^{(3)})}$

2. $\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = a_1^{(3)}(1 - a_1^{(3)})$

3. $\frac{\partial z_1^{(3)}}{\partial \boldsymbol{a}^{(2)}} = [w_{11}^{(3)}, w_{12}^{(3)}]$ $\longleftarrow$ (*scalar-by-vector*)

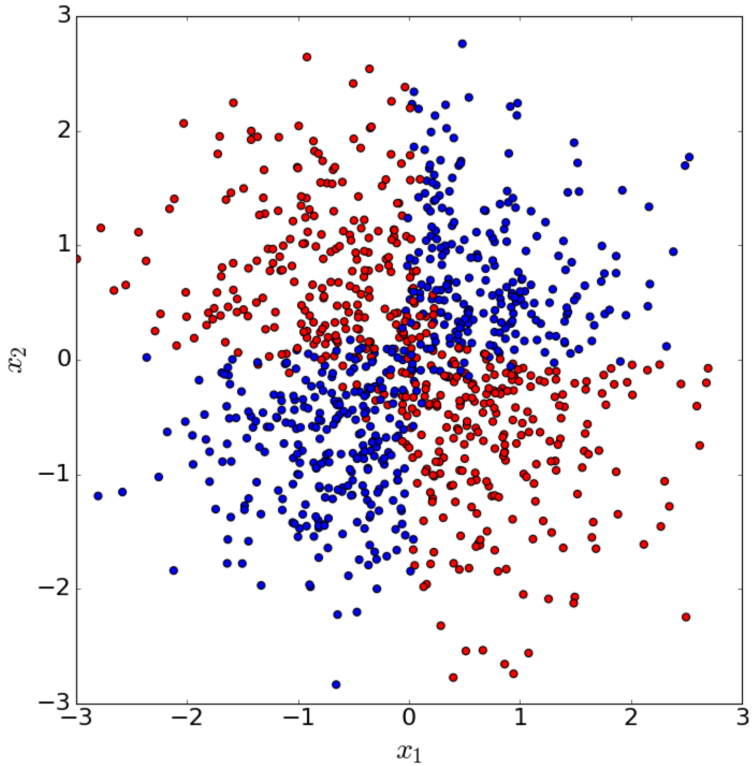
4. $\frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(2)}} = \begin{bmatrix} 1 - \tanh^2(z_1^{(2)}) & 0 \\ 0 & 1 - \tanh^2(z_2^{(2)}) \end{bmatrix}$ $\longleftarrow$ (*vector-by-vector*)

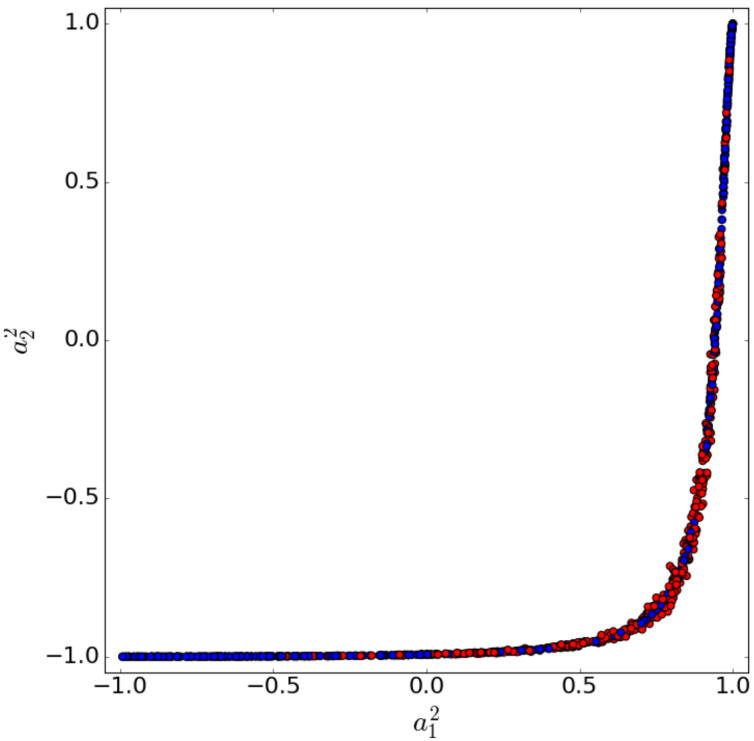
Then we can calculate (take two examples)

$$\frac{\partial E_n}{\partial w_{11}^{(3)}} = \left(\frac{\partial E_n}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \right) \frac{\partial z_1^{(3)}}{\partial w_{11}^{(3)}} = (a_1^{(3)} - y_n) a_1^{(2)}$$

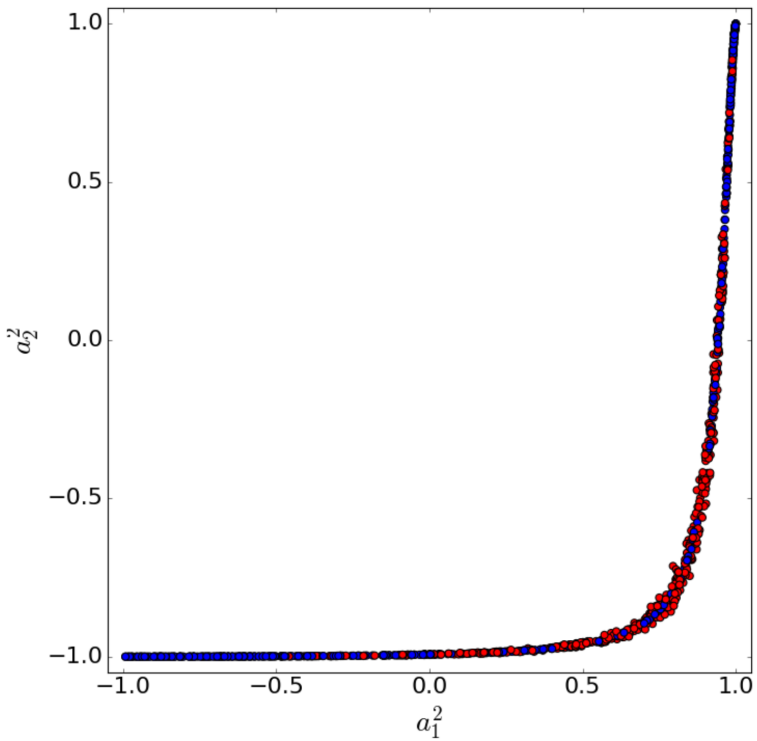
$$\frac{\partial E_n}{\partial w_{12}^{(2)}} = \left(\frac{\partial E_n}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \boldsymbol{a}^{(2)}} \frac{\partial \boldsymbol{a}^{(2)}}{\partial \boldsymbol{z}^{(2)}} \right) \frac{\partial \boldsymbol{z}^{(2)}}{\partial w_{12}^{(2)}} = (a_1^{(3)} - y_n) w_{11}^{(3)} \left(1 - \tanh^2(z_1^{(2)}) \right) x_{n2}$$

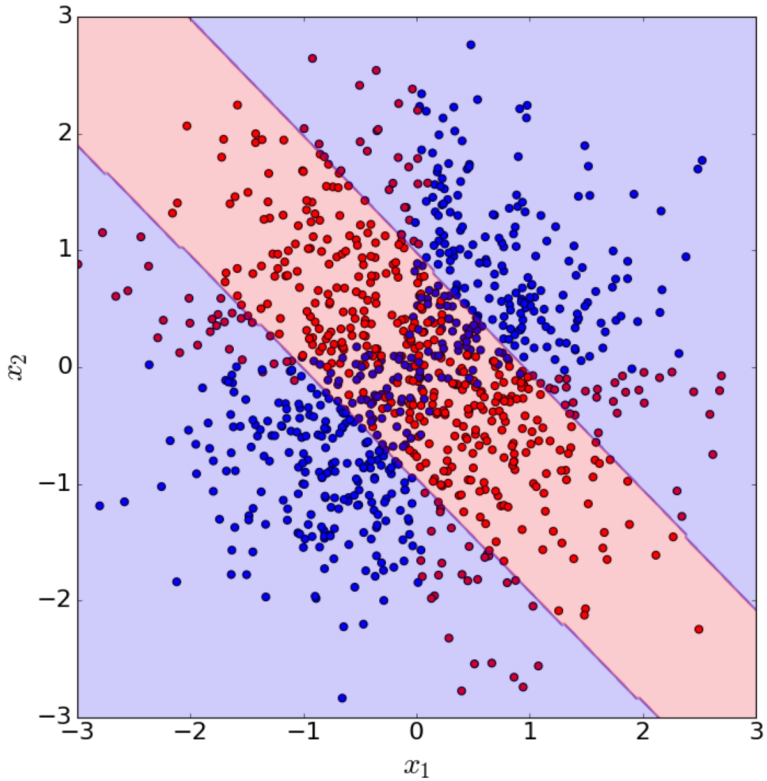
Scatterplot comparison (x_1, x_2) vs $(a_1^{(2)}, a_2^{(2)})$





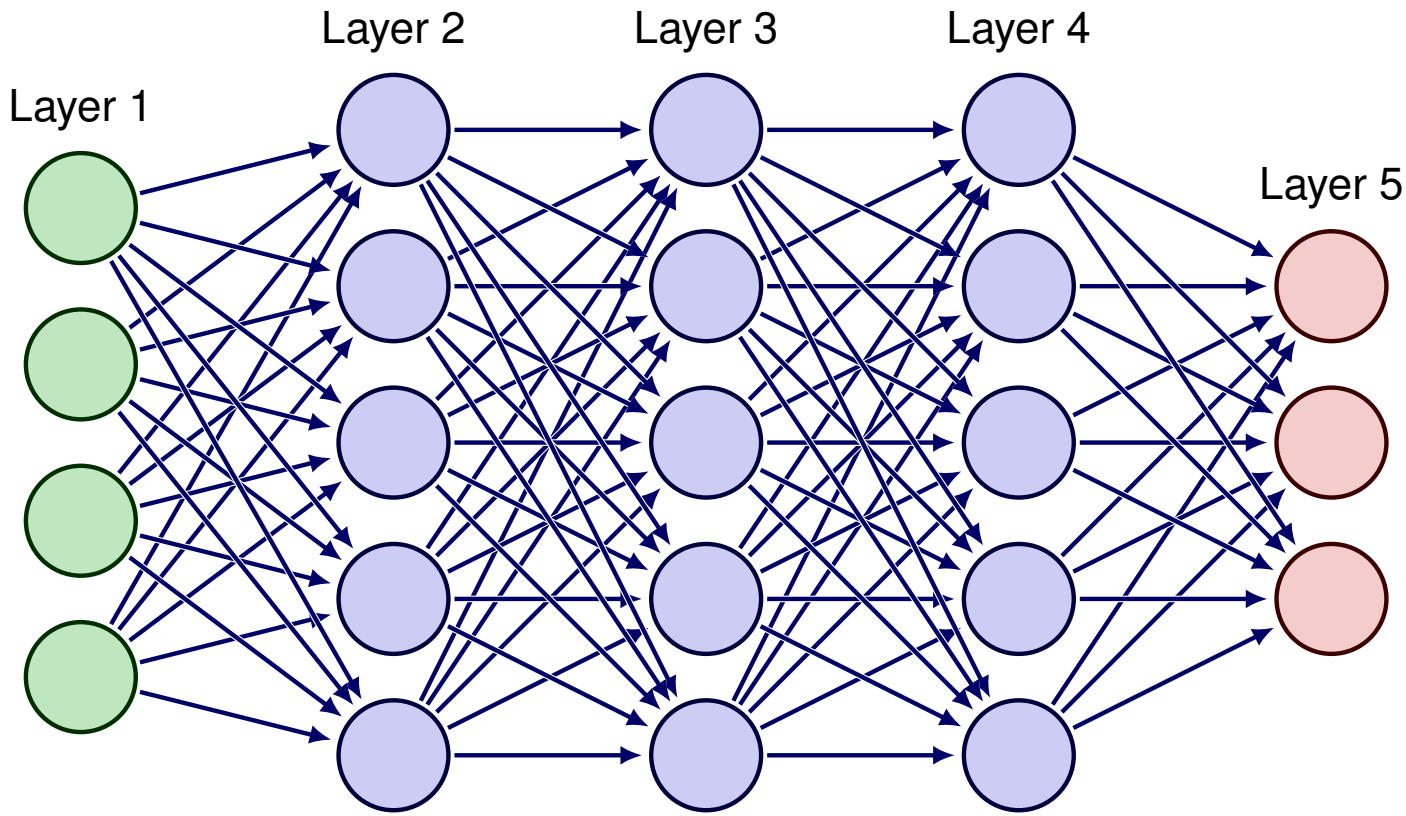
Decision boundary of MLP





Feedforward neural networks (1)

What is a feedforward neural network?

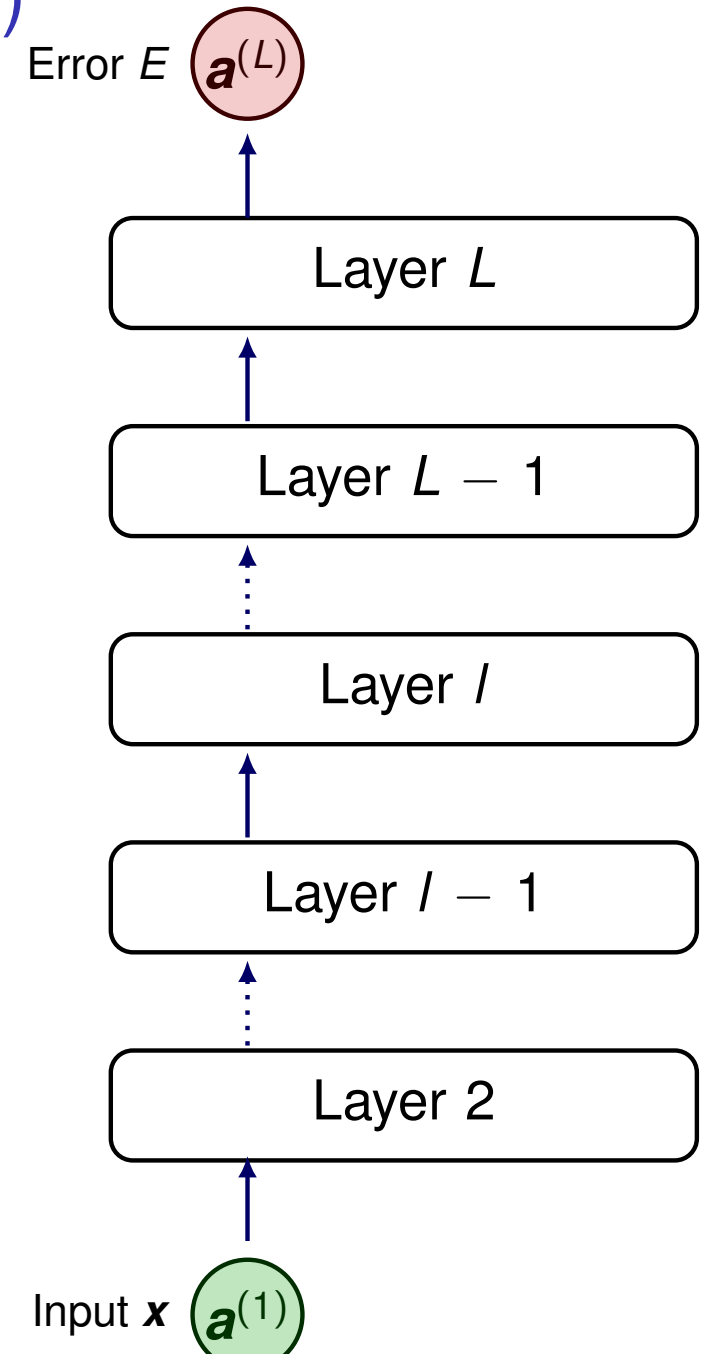


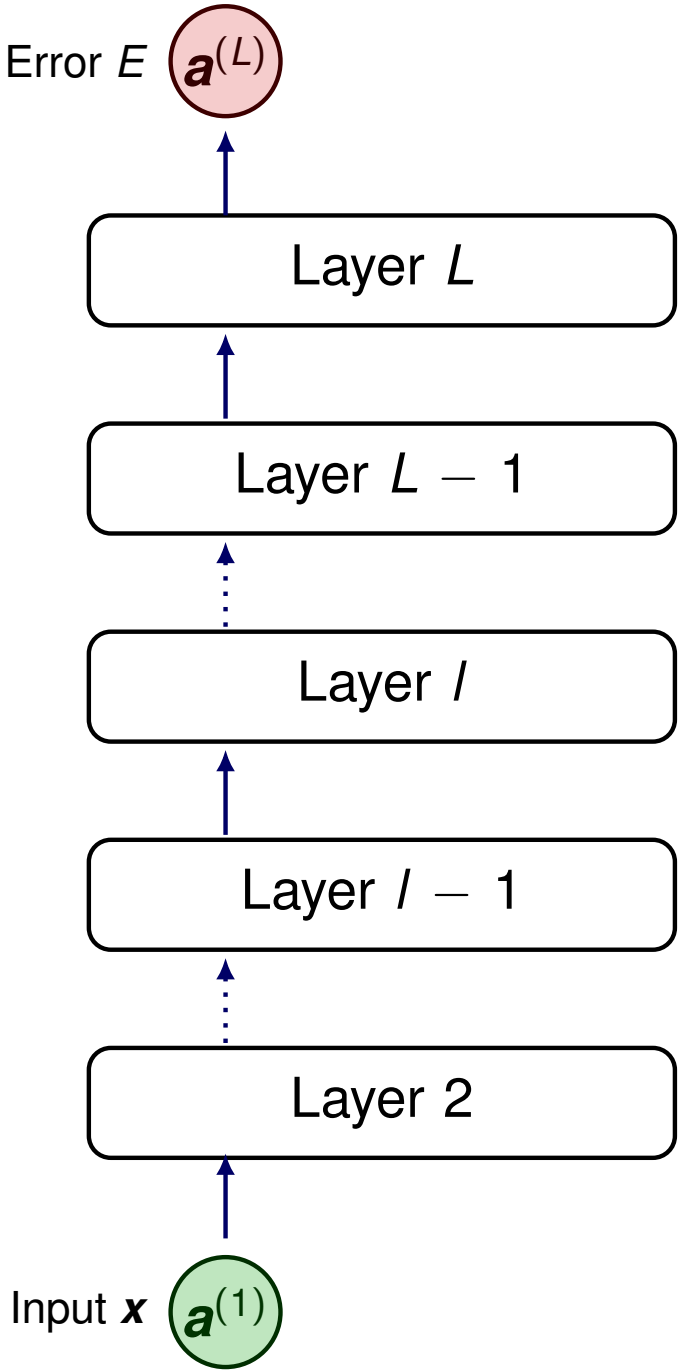
Feedforward neural networks (2)

We can easily extend the MLP architecture to any finite number L of layers, in which layer l = 1, . . . , L computes the following function:

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}, \quad \boldsymbol{a}^{(l)} = h_l(\boldsymbol{z}^{(l)})$$

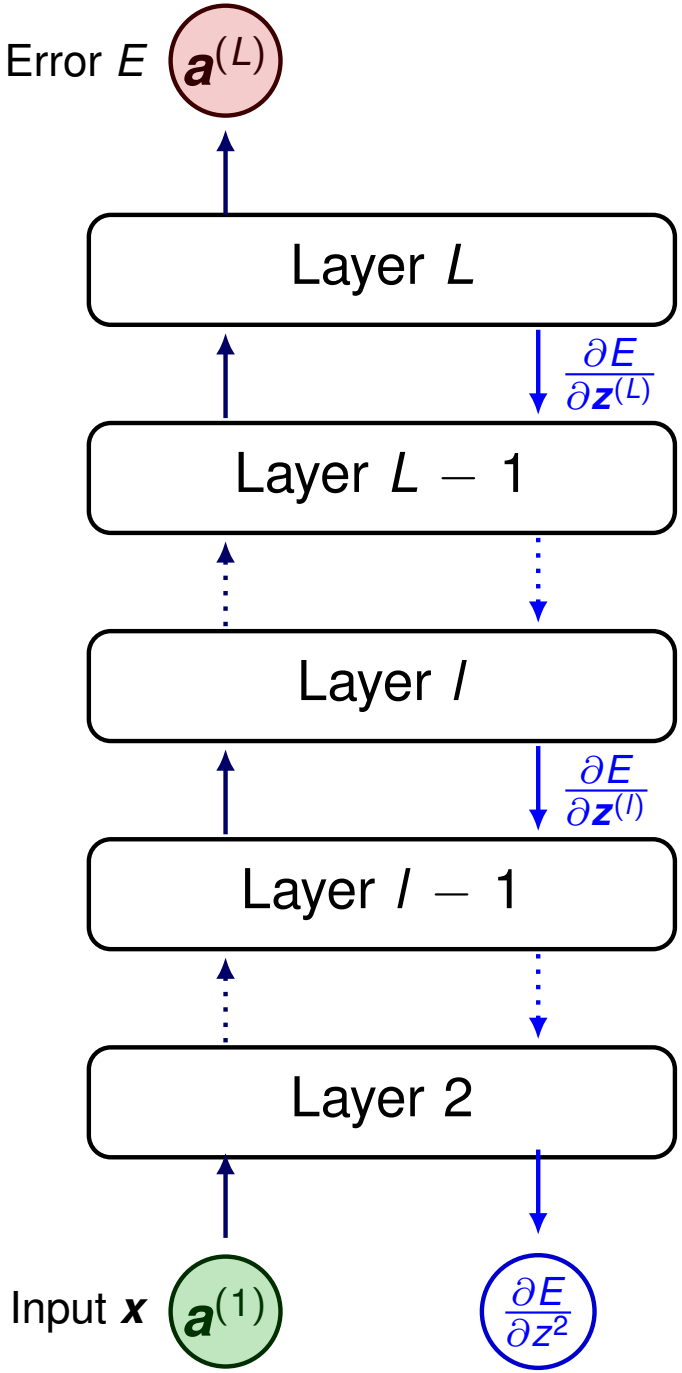
where h_l is the non-linear activation in layer l.





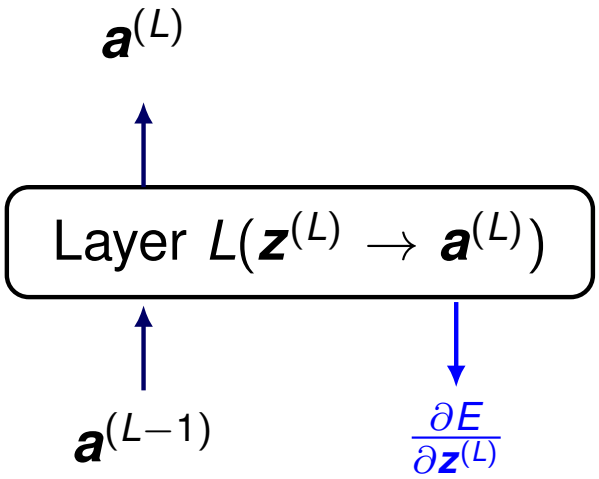
Forward Equations

1. $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
2. $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$ for $l = 2, ..., L$
3. $\boldsymbol{a}^{(l)} = h_l(\boldsymbol{z}^{(l)})$ for $l = 2, ..., L$
4. $E(\boldsymbol{a}^{(L)}, \boldsymbol{y})$



Backward pass to compute gradients

Back propagation: output layer



Transformation:

$$\boldsymbol{z}^{(L)} = \boldsymbol{W}^{(L)}\boldsymbol{a}^{(L-1)} + \boldsymbol{b}^{(L)}, \quad \boldsymbol{a}^{(L)} = h_L(\boldsymbol{z}^{(L)}).$$

Given error $E(\boldsymbol{a}^{(L)}, \boldsymbol{y})$, we want

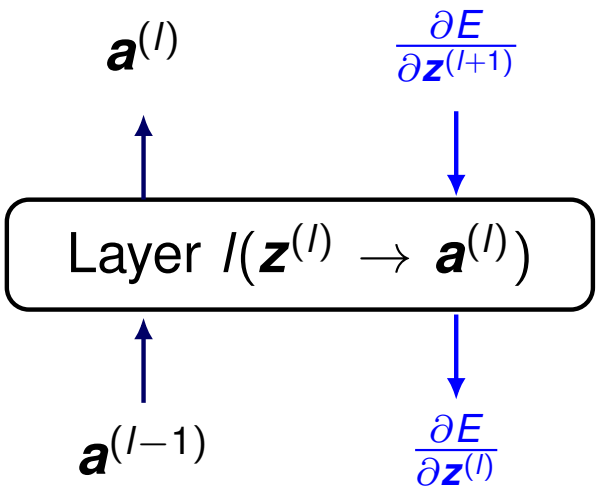
$$\frac{\partial E}{\partial \boldsymbol{z}^{(L)}} = \frac{\partial E}{\partial \boldsymbol{a}^{(L)}} \cdot \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}}.$$

If there are n_L units in Layer L, $\frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}}$ is the $n_L \times n_L$ Jacobian matrix:

$$\frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} = \begin{bmatrix} \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} & \frac{\partial a_1^{(L)}}{\partial z_2^{(L)}} & \cdots & \frac{\partial a_1^{(L)}}{\partial z_{n_L}^{(L)}} \\ \frac{\partial a_2^{(L)}}{\partial z_1^{(L)}} & \frac{\partial a_2^{(L)}}{\partial z_2^{(L)}} & \cdots & \frac{\partial a_2^{(L)}}{\partial z_{n_L}^{(L)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial a_{n_L}^{(L)}}{\partial z_1^{(L)}} & \frac{\partial a_{n_L}^{(L)}}{\partial z_2^{(L)}} & \cdots & \frac{\partial a_{n_L}^{(L)}}{\partial z_{n_L}^{(L)}} \end{bmatrix}$$

If h_L is applied element-wise, then this matrix is diagonal.

Back propagation: hidden layer



Transformation in layer $l+1$:

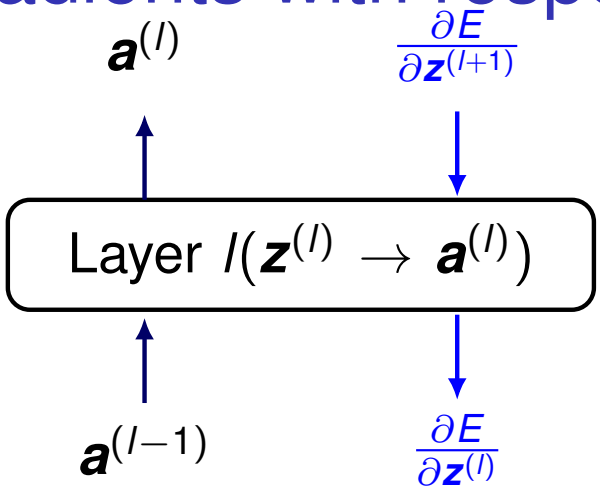
$$\boldsymbol{z}^{(l+1)} = \boldsymbol{W}^{(l+1)}\boldsymbol{a}^{(l)} + \boldsymbol{b}^{(l+1)}$$

where $w_{i,j}^{l+1}$ weight on connection from j-th unit in layer l to i-th unit in layer $l+1$

Compute the gradient with respect to $\boldsymbol{z}^{(l)}$:

$$\begin{aligned}
\frac{\partial E}{\partial \boldsymbol{z}^{(l)}} &= \frac{\partial E}{\partial \boldsymbol{z}^{(l+1)}} \cdot \frac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{z}^{(l)}} \\
&= \frac{\partial E}{\partial \boldsymbol{z}^{(l+1)}} \cdot \frac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} \cdot \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}} \\
&= \frac{\partial E}{\partial \boldsymbol{z}^{(l+1)}} \cdot \boldsymbol{W}^{(l+1)} \cdot \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}}
\end{aligned}$$

Gradients with respect to parameters



Transformation in layer l:

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}, \quad \boldsymbol{a}^{(l)} = h_l(\boldsymbol{z}^{(l)}).$$

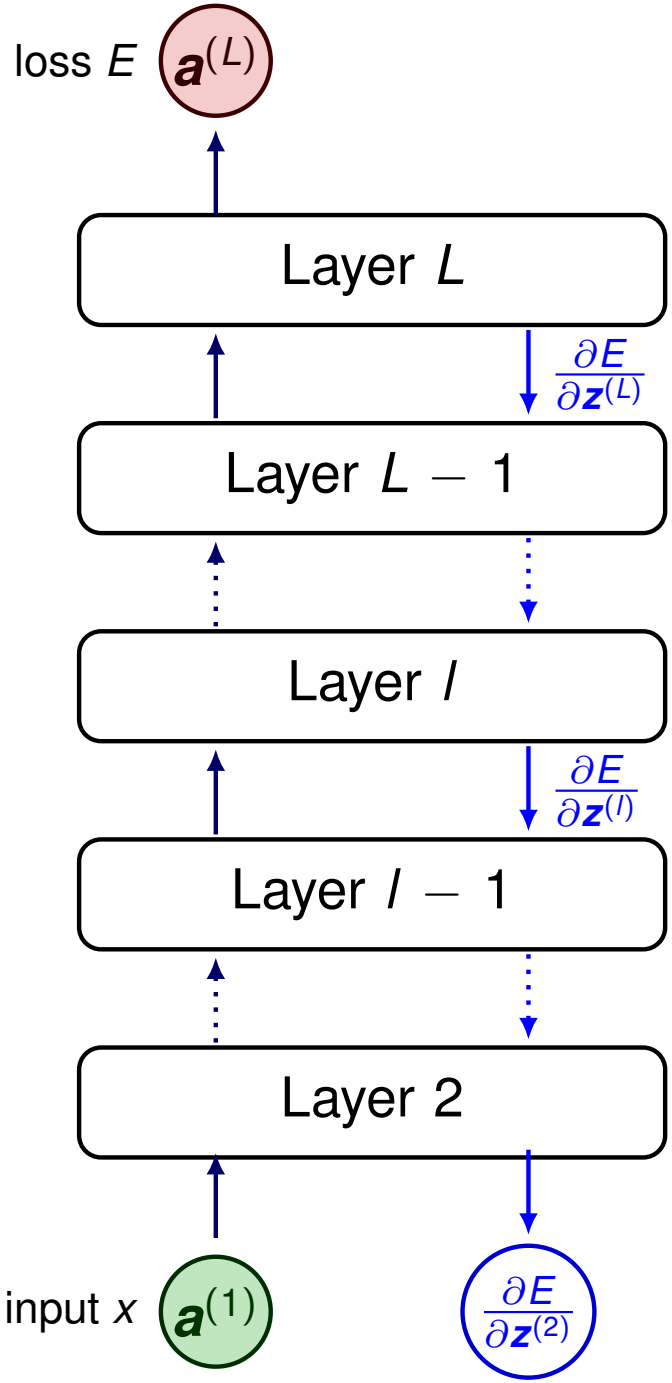
$\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are parameters to be optimized.

Compute the gradients with respect to $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial \boldsymbol{z}^{(l)}}\left(\frac{\partial \boldsymbol{z}^{(l)}}{\partial z_i^{(l)}} \cdot \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}\right) = \frac{\partial E}{\partial \boldsymbol{z}^{(l)}} \cdot \begin{bmatrix} 0 \\ \vdots \\ a_j^{(l-1)} \text{(i-th row)} \\ \vdots \\ 0 \end{bmatrix}, \quad \frac{\partial E}{\partial b_i^{(l)}} = \frac{\partial E}{\partial z_i^{(l)}}.$$

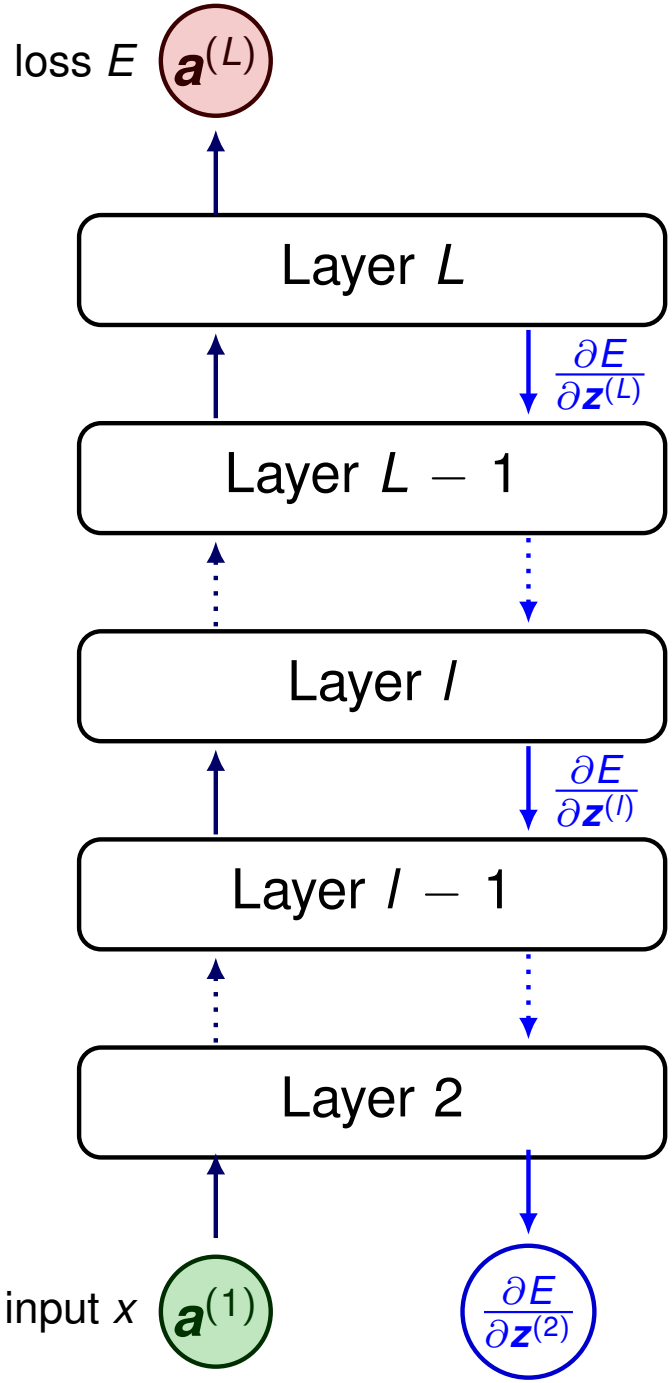
More succinctly, we may write

$$\frac{\partial E}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{a}^{(l-1)}\left(\frac{\partial E}{\partial \boldsymbol{z}^{(l)}}\right) \qquad \frac{\partial E}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial E}{\partial \boldsymbol{z}^{(l)}}$$



Forward equations

1. $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
2. $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$ for $l = 2, ..., L$
3. $\boldsymbol{a}^{(l)} = h_l(\boldsymbol{z}^{(l)})$ for $l = 2, ..., L$
4. $E(\boldsymbol{a}^{(L)}, \boldsymbol{y})$



Forward equations

1. $\boldsymbol{a}^{(1)} = \boldsymbol{x}$
2. $\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$ for $l = 2, ..., L$
3. $\boldsymbol{a}^{(l)} = h_l(\boldsymbol{z}^{(l)})$ for $l = 2, ..., L$
4. $E(\boldsymbol{a}^{(L)}, \boldsymbol{y})$

Back-propagation equations

1. Compute $\frac{\partial E}{\partial \boldsymbol{z}^{(L)}} = \frac{\partial E}{\partial \boldsymbol{a}^{(L)}} \cdot \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}}$
2. Compute $\frac{\partial E}{\partial \boldsymbol{z}^{(l)}} = \frac{\partial E}{\partial \boldsymbol{z}^{(l+1)}} \cdot \boldsymbol{W}^{(l+1)} \cdot \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}}$
3. Weight gradient: $\frac{\partial E}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{a}^{(l-1)} \left(\frac{\partial E}{\partial \boldsymbol{z}^{(l)}}\right)$
4. Bias gradient: $\frac{\partial E}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial E}{\partial \boldsymbol{z}^{(l)}}$

Computational questions

What is the running time to compute the gradients?

- As many matrix multiplications as there are fully connected layers
- Performed twice during forward and backward pass

What is the space requirement?

- Need to store vectors $\boldsymbol{a}^{(l)}$, $\boldsymbol{z}^{(l)}$, and $\frac{\partial E}{\partial \boldsymbol{z}^{(l)}}$ for each layer

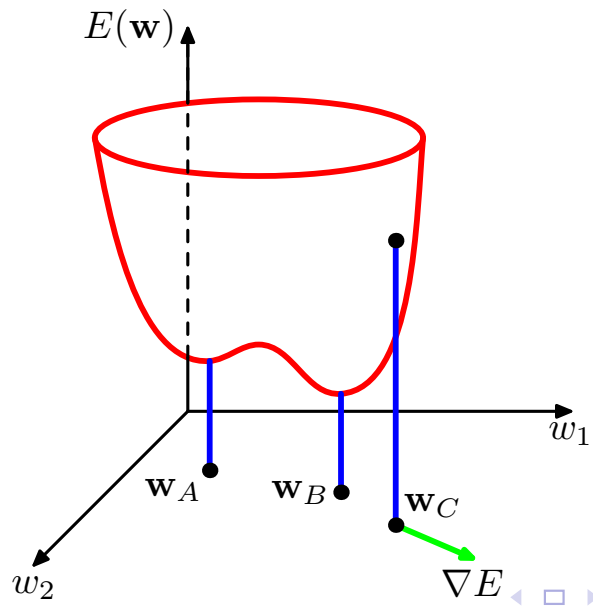
Can we process multiple examples together?

- Yes, if we minibatch, we perform tensor operations
- Make sure that all parameters fit in GPU memory

Outline

Training Deep Neural Networks

- Stochastic gradient descent (SGD): a mini-batch of samples
- A complete pass through the whole training set is known as a training *epoch*
- Points at which the gradient vanishes are called *stationary points*
- A minimum that corresponds to the smallest value of the error function across the whole of w-space is said to be a *global minimum*.
- Any other minima corresponding to higher values of the error function are said to be *local minima*.



Parameter Initialization

- Have a significant effect on convergence and the generalization performance
- *Symmetry breaking*: the same initialized values will lead to the same update of parameters.
- Consider either a uniform distribution in $[-\epsilon, \epsilon]$ or a zero-mean Gaussian $\mathcal{N}(0, \epsilon^2)$.
- The choice of ϵ is important. Consider a network layer:

$$z_i^{(l)} = \sum_{j=1}^{M} w_{ij} a_j^{(l-1)}, \qquad a_i^{(l-1)} = \text{ReLU}(z_i^{(l-1)}).$$

With $\mathcal{N}(0, \epsilon^2)$ initialization and fixed variance λ^2 of $z_j^{(l-1)}$:

$$\mathbb{E}[z_i^{(l)}] = 0 \qquad \text{Var}[z_j^{(l)}] = \frac{M}{2}\epsilon^2\lambda^2$$

- *He initialization* (He et al., 2015): set $\frac{M}{2}\epsilon^2 = 1 \rightarrow \epsilon = \sqrt{\frac{2}{M}}$.

Normalization (1)

To removes the need to deal with extremely large or small values

- *Data normalization*: for **continuous inputs**, to re-scale and re-center the input values:

$$\mu_i = \frac{1}{N}\sum_{n=1}^{N} x_{ni}, \quad \sigma_i^2 = \frac{1}{N}\sum_{n=1}^{N}(x_{ni} - \mu_i)^2 \quad \rightarrow \tilde{x}_{ni} = \frac{x_{ni} - \mu_i}{\sigma_i}.$$

- *Batch normalization*: for **each nonlinear unit** $a_i = h(z_i)$, to normalize the pre-activations within a mini-batch of size K:

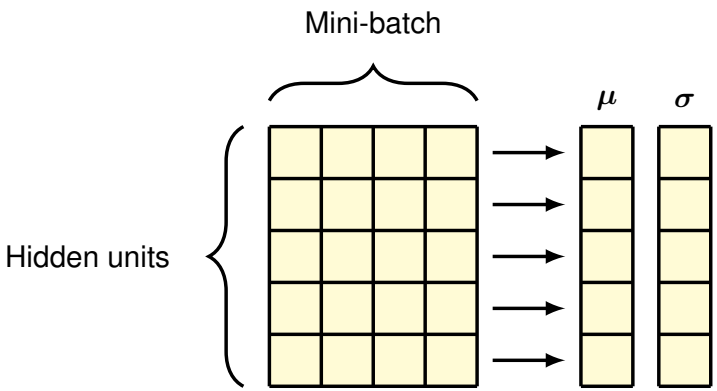
$$\mu_i = \frac{1}{K}\sum_{n=1}^{K} z_{ni}, \quad \sigma_i^2 = \frac{1}{N}\sum_{n=1}^{N}(z_{ni} - \mu_i)^2, \quad \rightarrow \tilde{z}_{ni} = \gamma_i \frac{z_{ni} - \mu_i}{\sigma_i + \delta} + \beta_i$$

- *Layer normalization*: to normalize across the **hidden-unit values for each data point** separately

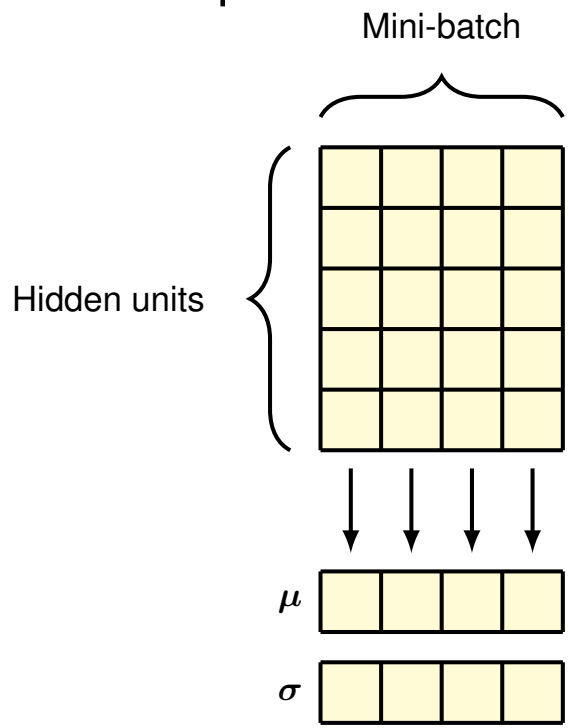
$$\mu_n = \frac{1}{M}\sum_{i=1}^{M} z_{ni}, \quad \sigma_n^2 = \frac{1}{M}\sum_{i=1}^{M}(z_{ni} - \mu_i)^2, \quad \rightarrow \tilde{z}_{ni} = \gamma_n \frac{z_{ni} - \mu_n}{\sigma_n + \delta} + \beta_n$$

Normalization (2)

Batch normalization: the mean and variance are computed across the mini-batch separately for each hidden unit.



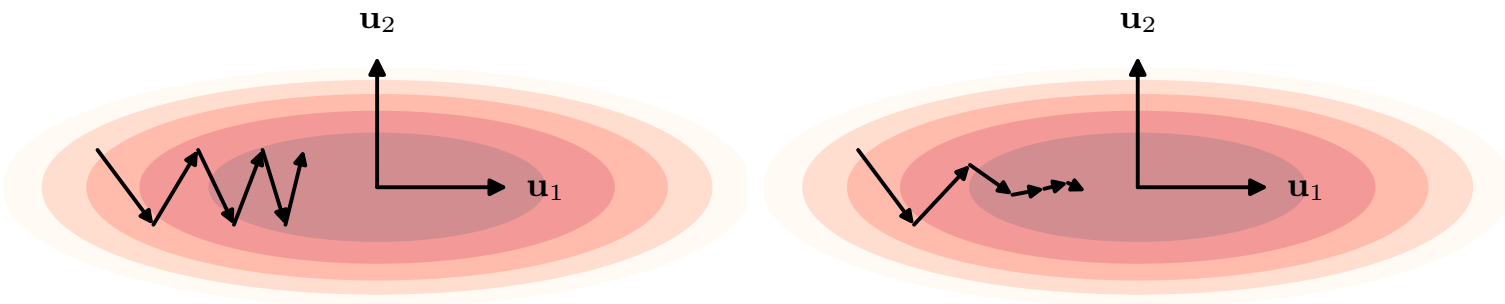
Layer normalization: the mean and variance are computed across the hidden units separately for each data point.



Gradient momentum

Stochastic gradient descent (one data point at a time):

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} + \Delta \boldsymbol{w}^{(\tau-1)}, \quad \Delta \boldsymbol{w}^{(\tau-1)} = -\eta \nabla E_n(\boldsymbol{w}^{\tau-1})$$



- Can lead to oscillations for fixed step-sizes
- To add inertia to the motion through weight space and smooth out the oscillations

$$\Delta \boldsymbol{w}^{(\tau-1)} = -\eta \nabla E_n(\boldsymbol{w}^{\tau-1}) + \mu \Delta \boldsymbol{w}^{(\tau-2)}$$

where μ is called the momentum parameter.

Adaptive gradients

Optimal learning rate depends on the local curvature of error surface

- *AdaGrad* (Adaptive gradient): to reduce each learning rate by using the accumulated sum of squared gradients

$$r_i^{(\tau)} = r_i^{(\tau-1)} + \left(\frac{\partial E(\boldsymbol{w})}{\partial w_i}\right)^2, w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{r_i^{(\tau)}} + \epsilon} \frac{\partial E(\boldsymbol{w})}{\partial w_i}$$

- *RMSProp* (Root mean square propagation): moving average

$$r_i^{(\tau)} = \beta r_i^{(\tau-1)} + (1-\beta)\left(\frac{\partial E(\boldsymbol{w})}{\partial w_i}\right)^2, w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{r_i^{(\tau)}} + \epsilon} \frac{\partial E(\boldsymbol{w})}{\partial w_i}$$

- *Adam* (Adaptive moments): moving average for both

$$s_i^{(\tau)} = \beta_1 s_i^{(\tau-1)} + (1-\beta_1)\frac{\partial E(\boldsymbol{w})}{\partial w_i}, r_i^{(\tau)} = \beta_2 r_i^{(\tau-1)} + (1-\beta_2)\left(\frac{\partial E(\boldsymbol{w})}{\partial w_i}\right)^2$$

$$\underbrace{\hat{s}_i^{(\tau)} = \frac{s_i^{(\tau)}}{1-\beta_1^\tau}, \hat{r}_i^{(\tau)} = \frac{r_i^{(\tau)}}{1-\beta_2^\tau}}_{\text{Bias correction}}, w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{\hat{r}_i^{(\tau)}} + \epsilon} \hat{s}_i^{(\tau)}$$

Regularization: weight decay

- Regularization in linear models:

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \underbrace{\frac{\lambda}{2}\boldsymbol{w}^\top \boldsymbol{w}}_{\text{Regularization term}}$$

- *Weight decay* in machine learning: encourages weight values to decay towards zero, unless supported by the data.

$$\nabla\tilde{E}(\boldsymbol{w}) = \nabla E(\boldsymbol{w}) + \lambda \boldsymbol{w}$$

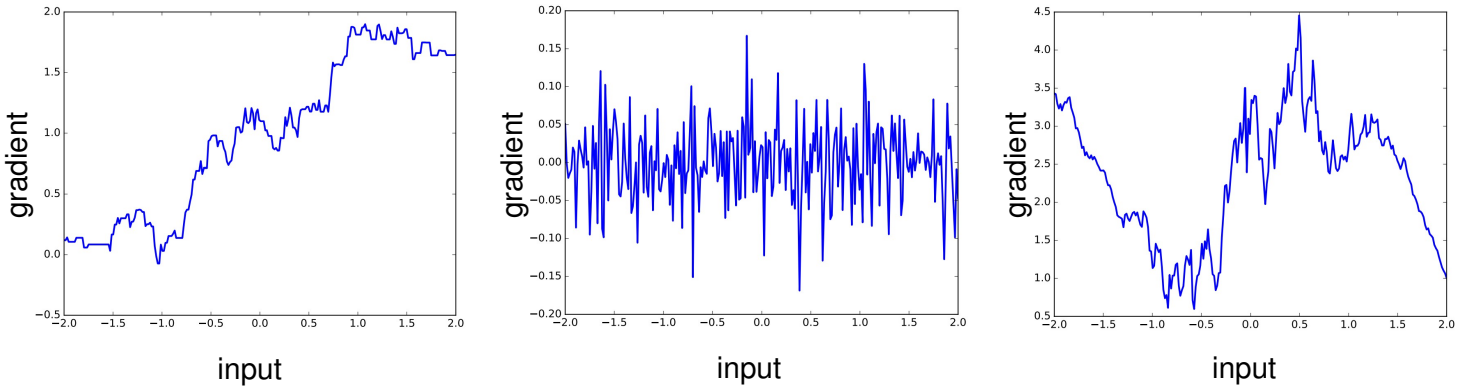
- *AdamW*: Adam with weight decay (to decouple the weight decay from the L2 regularization)

$$w_i^{(\tau)} = w_i^{(\tau-1)} - \eta \left(\frac{\hat{s}_i^{(\tau)}}{\sqrt{\hat{r}_i^{(\tau)}} + \delta} + \lambda w_i^{(\tau-1)} \right)$$

Regularization: residual connections

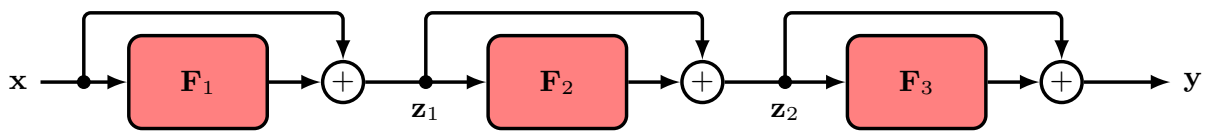
Training networks with a large number of layers is difficult

- *Shattered gradients*



Jacobian for networks with a single input and a single output: (a) two layers, (b) 25 layers, and (c) 51 layers with residual connections

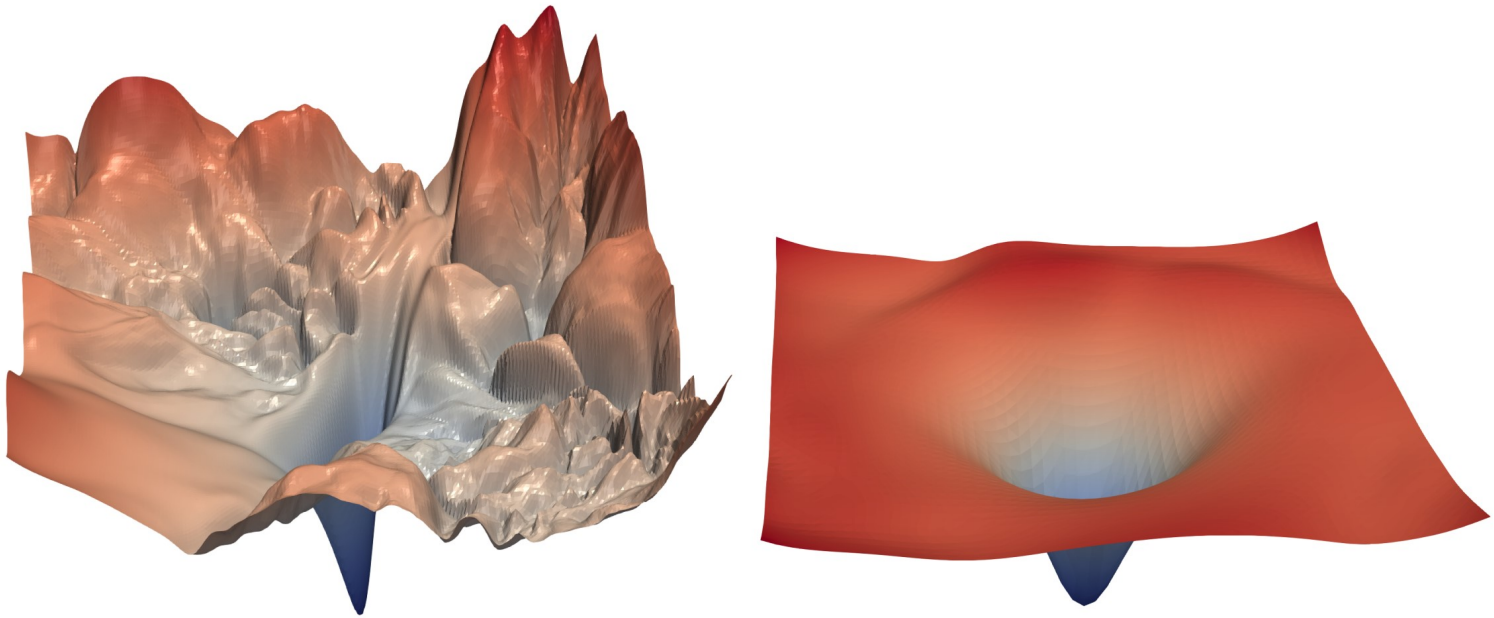
- *Residual connections*



$$\boldsymbol{z}_1 = \boldsymbol{F}_1(\boldsymbol{x}) + \boldsymbol{x}, \boldsymbol{z}_2 = \boldsymbol{F}_2(\boldsymbol{z}_1) + \boldsymbol{z}_1, \boldsymbol{y} = \boldsymbol{F}_2(\boldsymbol{z}_2) + \boldsymbol{z}_2,$$

Regularization: residual connections (cont.)

Error landscape with/without residual connections:



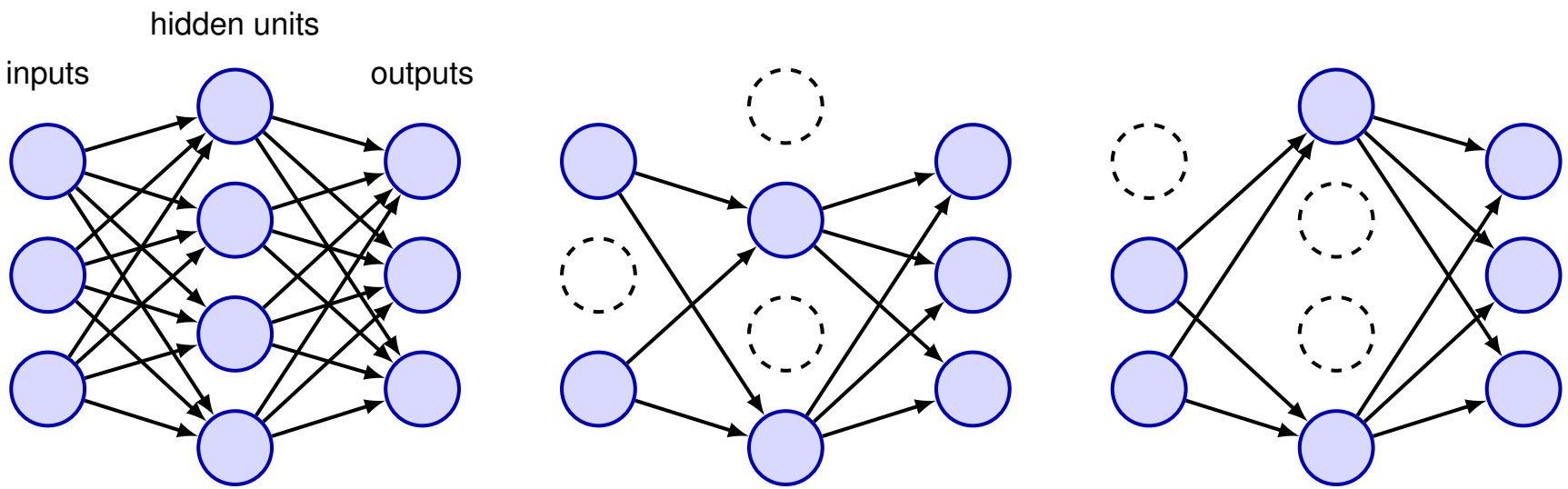
(a) (b)

- (a) A visualization of the error surface for a network with 56 layers.
- (b) The same network with the inclusion of residual connections.

Regularization: dropout

Dropout is one of the most effective forms of regularization and is widely used in applications to overcome overfitting

- To delete nodes from the network, including their connections, at random during training
- Applied to both hidden nodes and input nodes, but not outputs



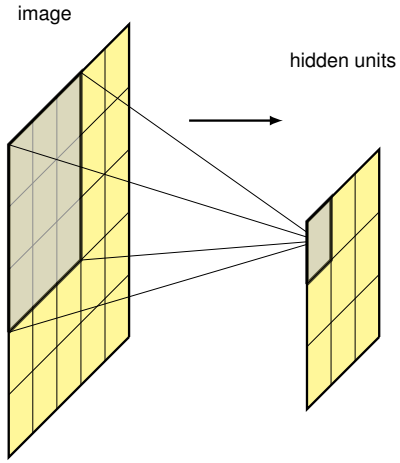
Outline

Structured input

- So far, the observed data are unstructured: the elements of the data vectors $\boldsymbol{x} = (x_1, ..., x_D)$ are treated as if we do not know anything in advance about how the individual elements might relate to each other
- Many applications of machine learning, however, involve structured data in which there are additional relationships between input variables
- *Two-dimensional input*: the pixels of an image have a spatial relationship and are arranged in a two-dimensional grid
  - **Convolutional neural networks**
- *Sequential input*: the words in natural language form a sequence
  - **Recurrent neural networks and transformers**

Convolution

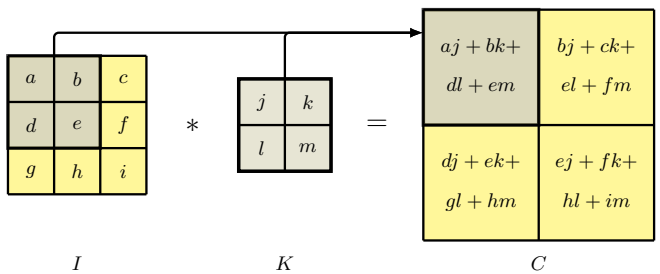
Receptive field: a small rectangular region, or patch, to capture the notion of locality



The output of this unit is given by

$$z = \text{ReLU}(\underbrace{\boldsymbol{w}^\top \boldsymbol{x} + w_0}_{\text{Convolution operation for 2D input}})$$

Convolution: $C(j,k) = \sum_l \sum_m I(j+l, k+m) K(l,m)$.



Example of a 3×3 image convolved with a 2×2 filter to give a resulting 2×2 feature map

Padding and stride

For a filter of dimensionality $M \times M$ applied to image of $J \times K$ pixels

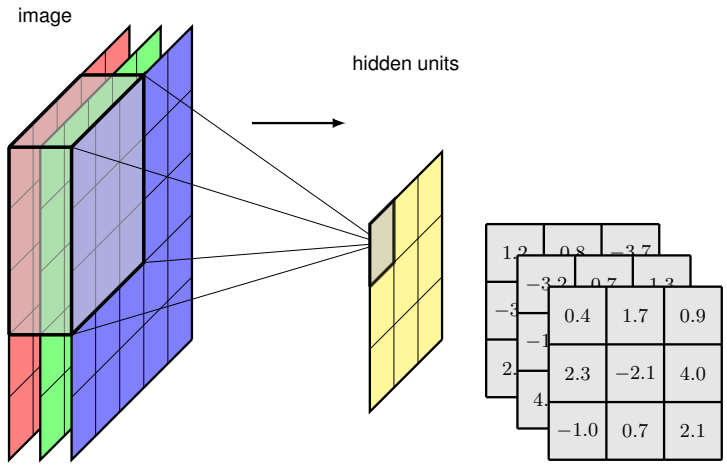
- **Padding**: To have the the feature map of the same dimension as the original image, padding the original image with additional P pixels around the outside
- **Stride**: To move the filter over the image in S strides at a time
- If use the same stride horizontally and vertically, the shape of the feature map will be

$$\lfloor \frac{J + 2P - M}{S} - 1 \rfloor \times \lfloor \frac{K + 2P - M}{S} - 1 \rfloor$$

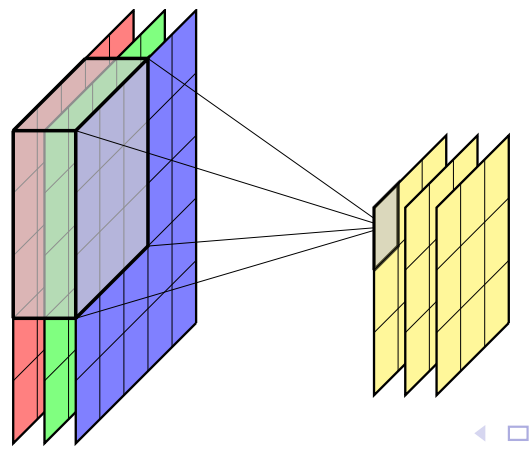
0	0	0	0	0	0
0	X_{11}	X_{12}	X_{13}	X_{14}	0
0	X_{21}	X_{22}	X_{23}	X_{24}	0
0	X_{31}	X_{32}	X_{33}	X_{34}	0
0	X_{41}	X_{42}	X_{43}	X_{44}	0
0	0	0	0	0	0

Multi-dimensional convolution

For an image with C channels described by a *tensor* of dimensionality $J \times K \times C$, we introduce a filter of dimensionality $M \times M \times C$

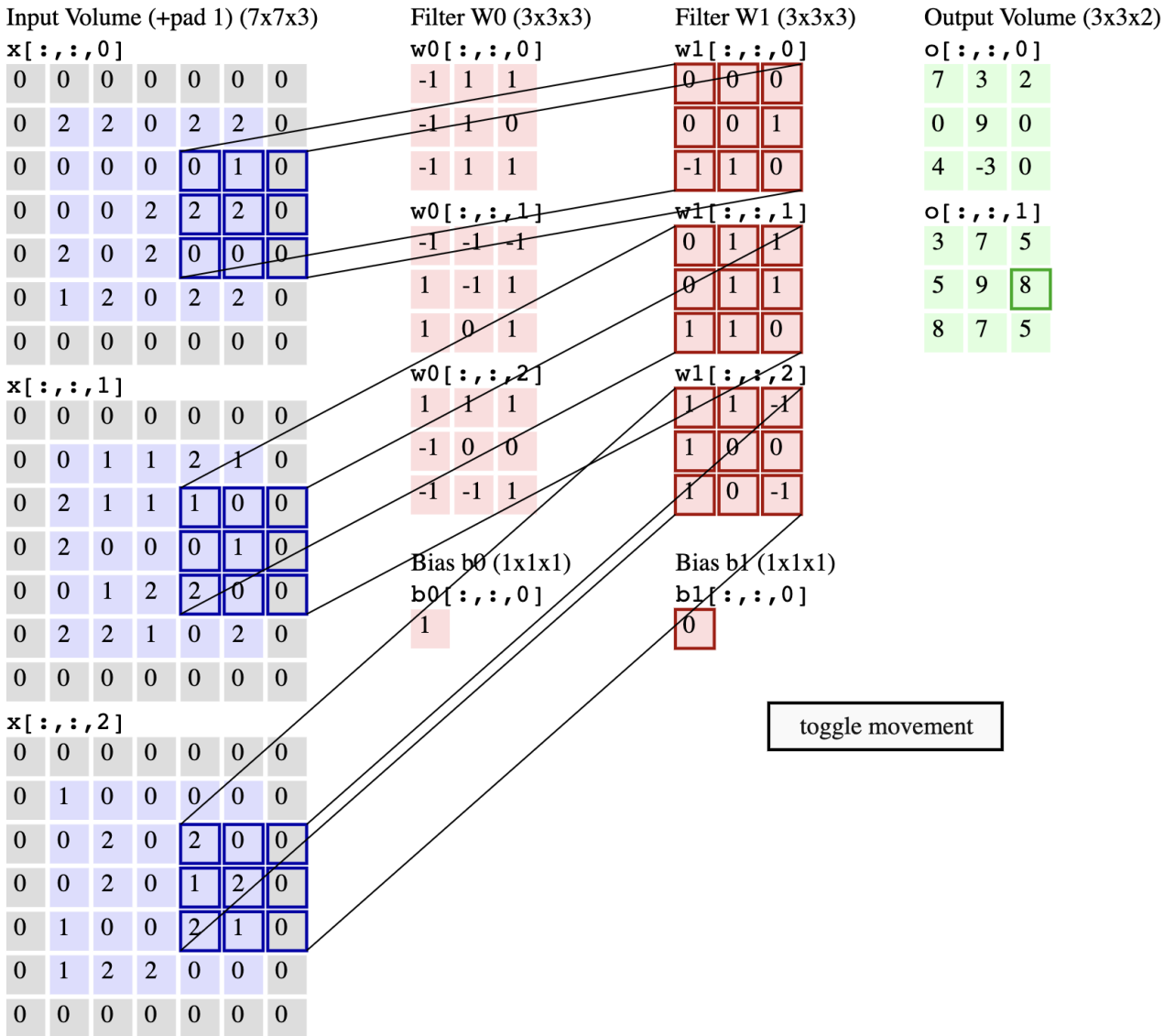


To extend to include multiple independent filter channels: filter tensor now has dimensionality $M \times M \times C \times C_{\text{OUT}}$ where C_{OUT} is the number of output channels



Example

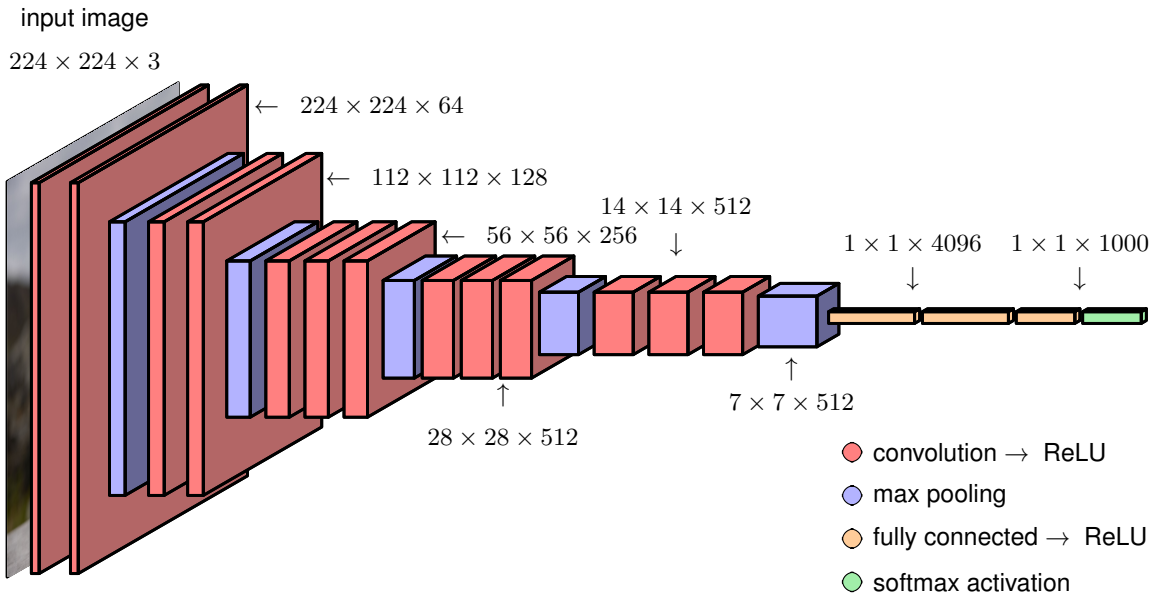
Two filters, zero padding, stride one[1]



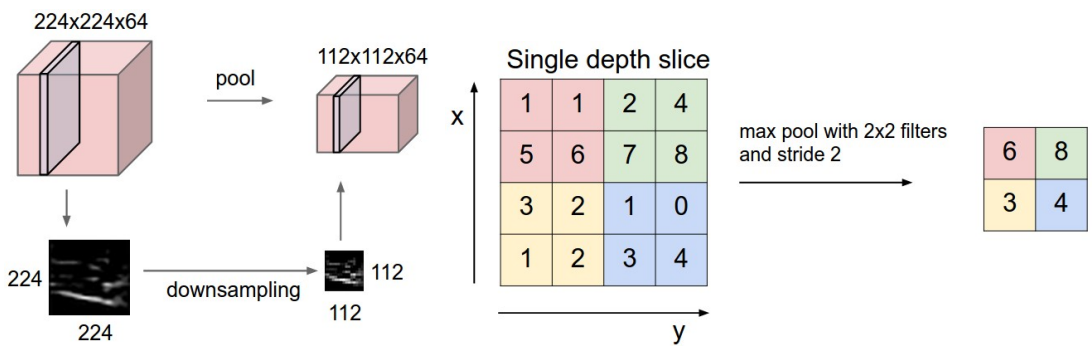
[1] https://cs231n.github.io/convolutional-networks/

Deep Convnets

To extend the architecture by considering multiple layers: VGG-16 model

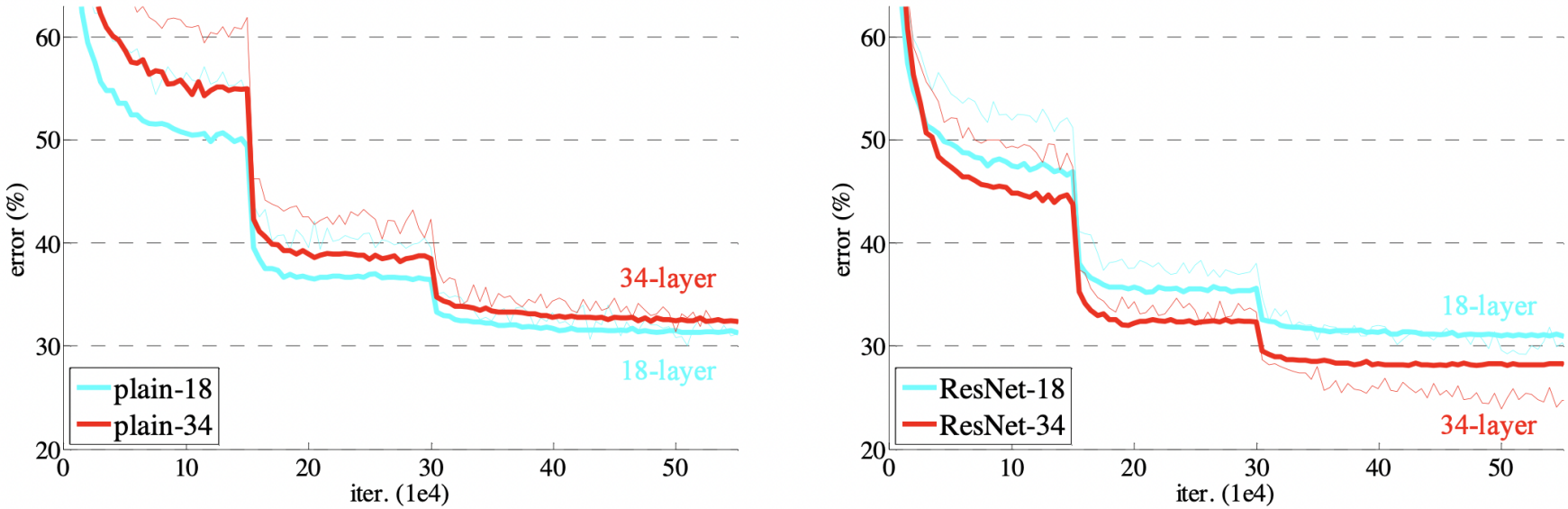


Pooling[2]



[2] https://cs231n.github.io/convolutional-networks/

ResNets: Why more layers can give worse models

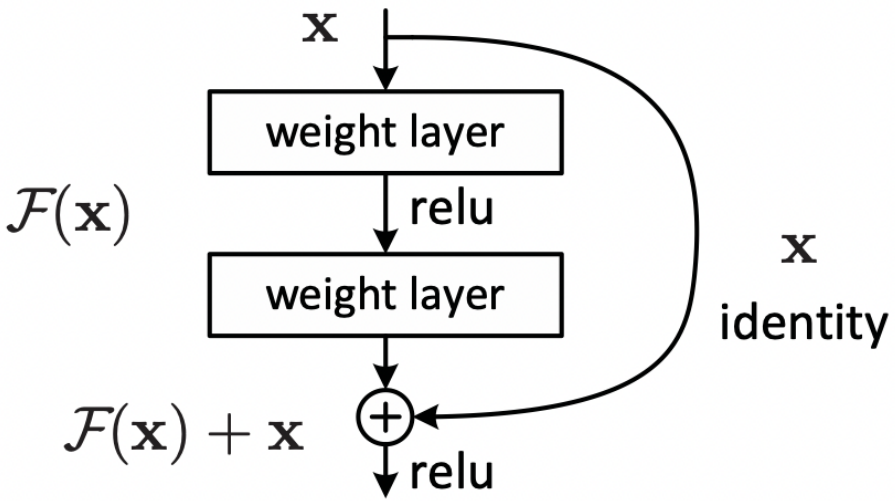


Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers

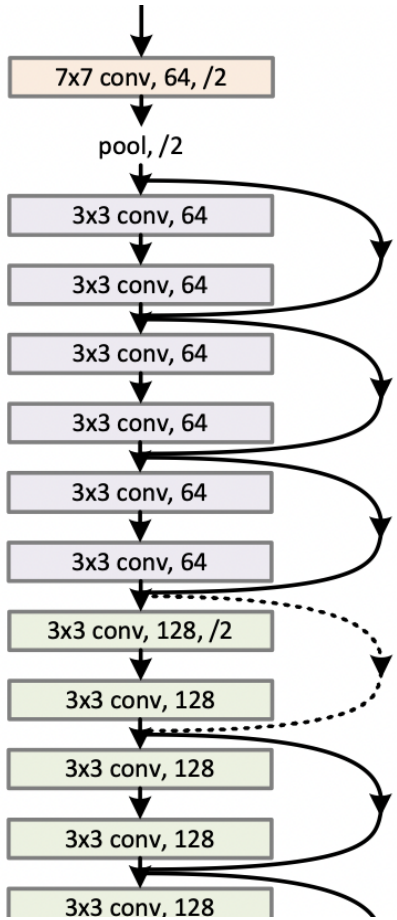
See paper by He, Zhang, Ren and Sun (2015)

ResNets

ResNet block



The 34-layer residual (the dotted shortcuts increase dimensions)



See paper by He, Zhang, Ren and Sun (2015)

Recap

Feedforward Neural Networks

Training Deep Neural Networks

Convolutional Neural Networks

END LECTURE