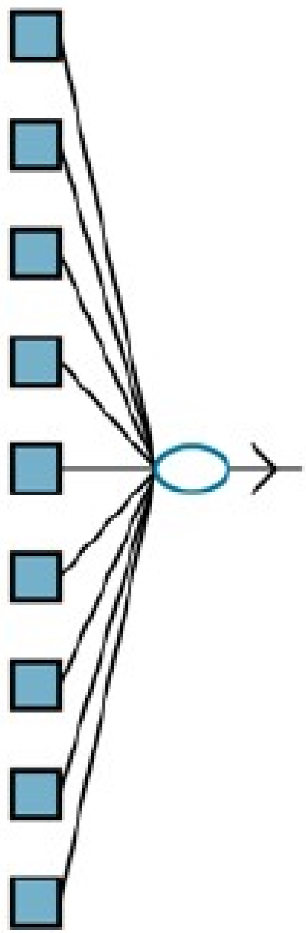
CSE 471: MACHINE LEARNING

Deep Learning

Outline

- Feedforward neural network
  - Multi layer perceptron (MLP)
- Convolutional neural network
- Recurrent neural network
-

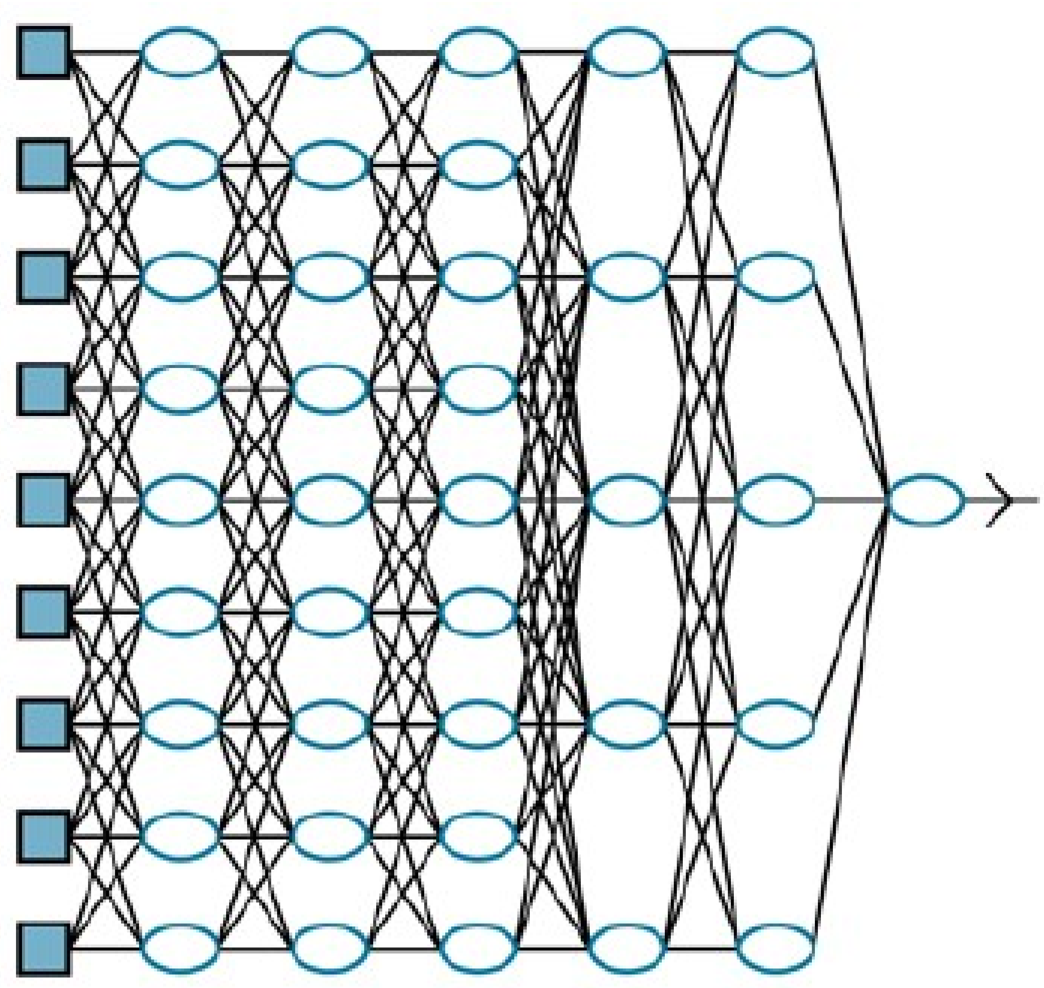
Perceptron



Multi layer perceptron

Deep learning network

A deep learning network has longer computation paths, allowing each variable to interact with all the others.



Feedforward Network

$$a_j = g_j\left(\sum_i w_{i,j} a_i\right) \equiv g_j(in_j),$$

- g_i – Nonlinear activation function
- There is an intercept *b*. *Think of it as weight given to a fixed +1 activation node.*

$$a_j = g_j(\mathbf{w}^\top \mathbf{x})$$

Nonlinearity (Sigmoid)

$$\sigma(x) = 1/(1 + e^{-x})$$

Nonlinearity

Sigmoid

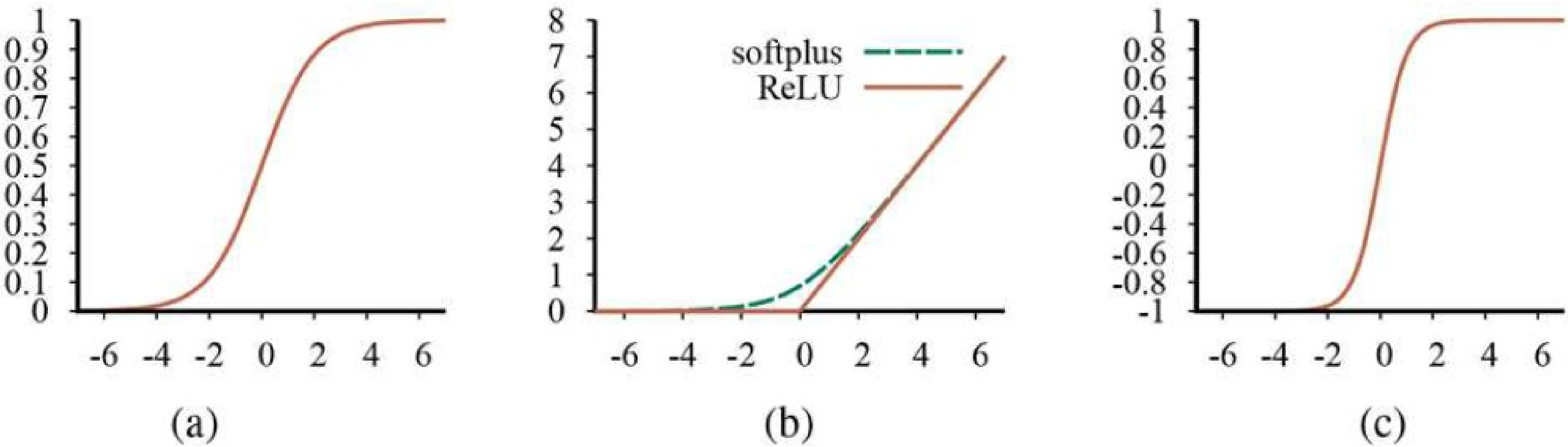
$$\sigma(x) = 1/(1 + e^{-x})$$

ReLU(x) = max(*0*, *x*)

softplus(x) = log(1 + e^{x})

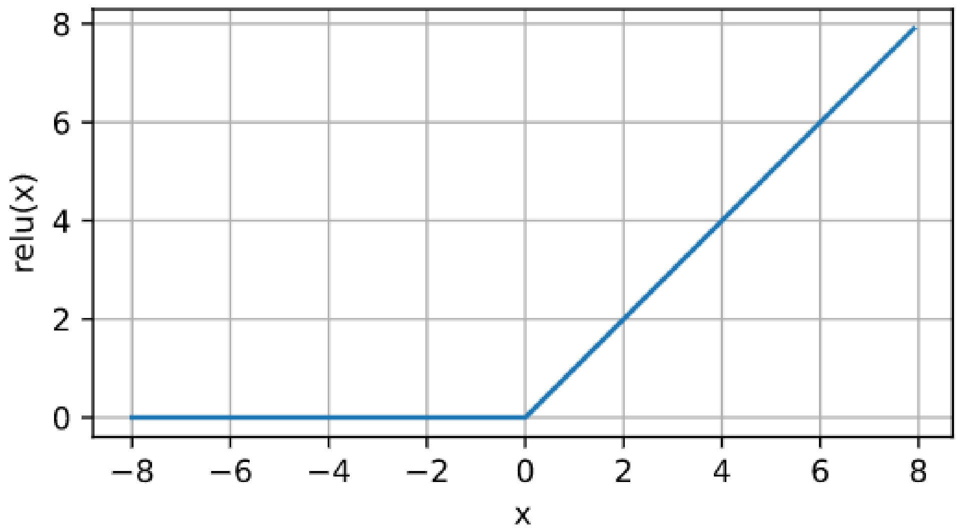
$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} = 2\sigma(2x) - 1$$

Nonlinearity

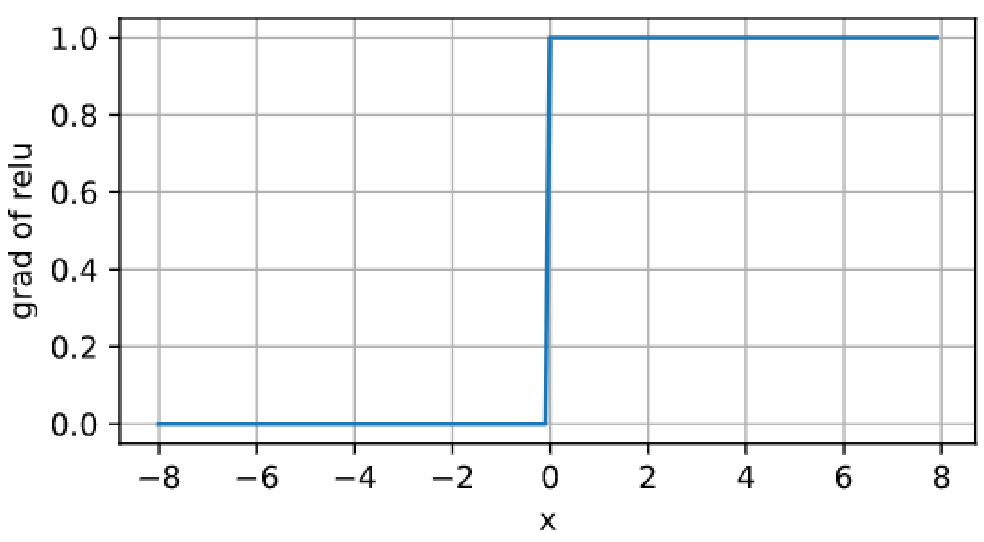


Activation functions commonly used in deep learning systems: (a) the logistic or sigmoid function; (b) the ReLU function and the softplus function; (c) the tanh function.

ReLU

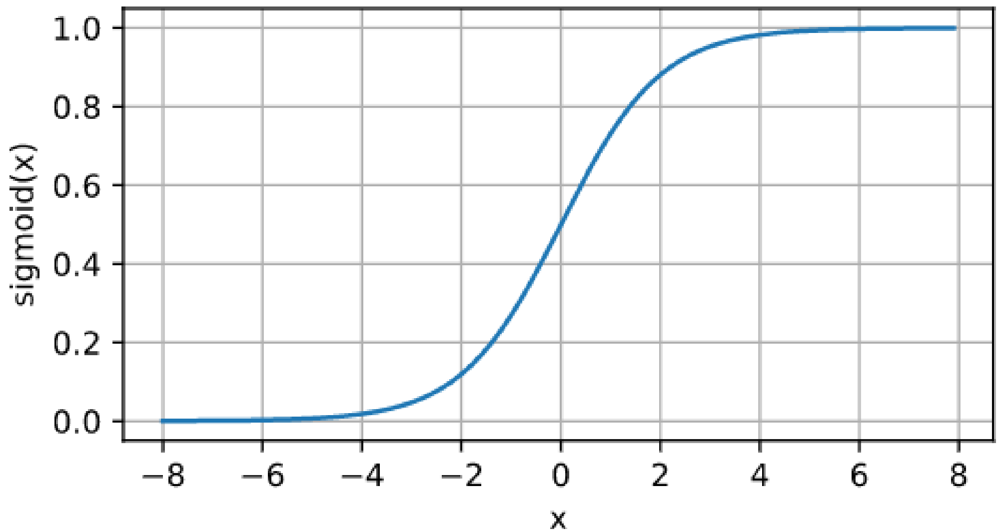


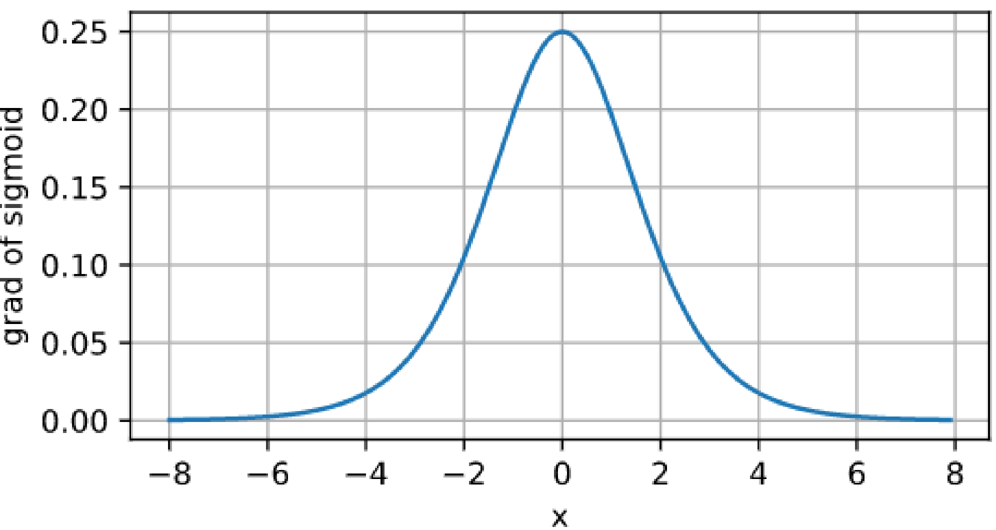
relu(x)
8
6
4
2
0
−8
−6
−4
−2
0
2
4
6
8
x



grad of relu
1.0
0.8
0.6
0.4
0.2
0.0
−8
−6
−4
−2
0
2
4
6
8
x

Sigmoid

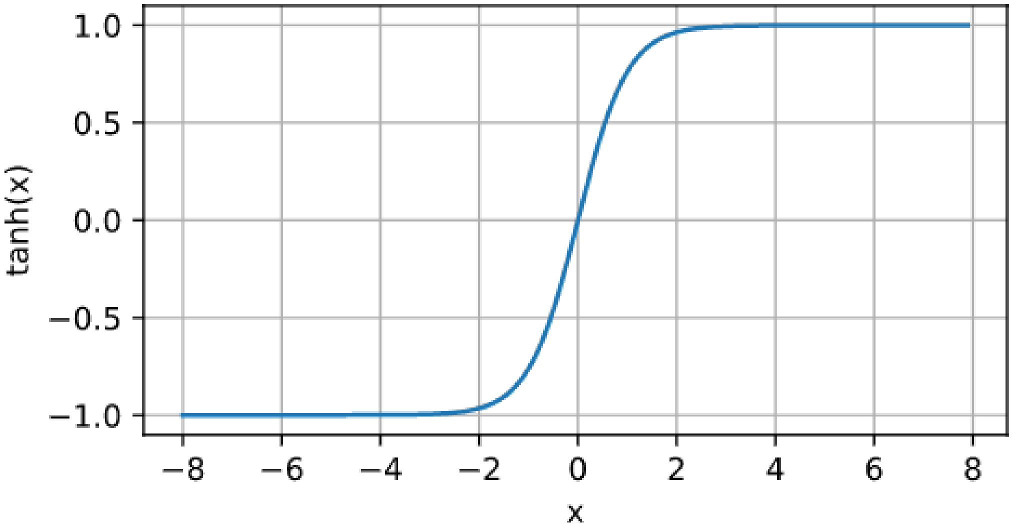




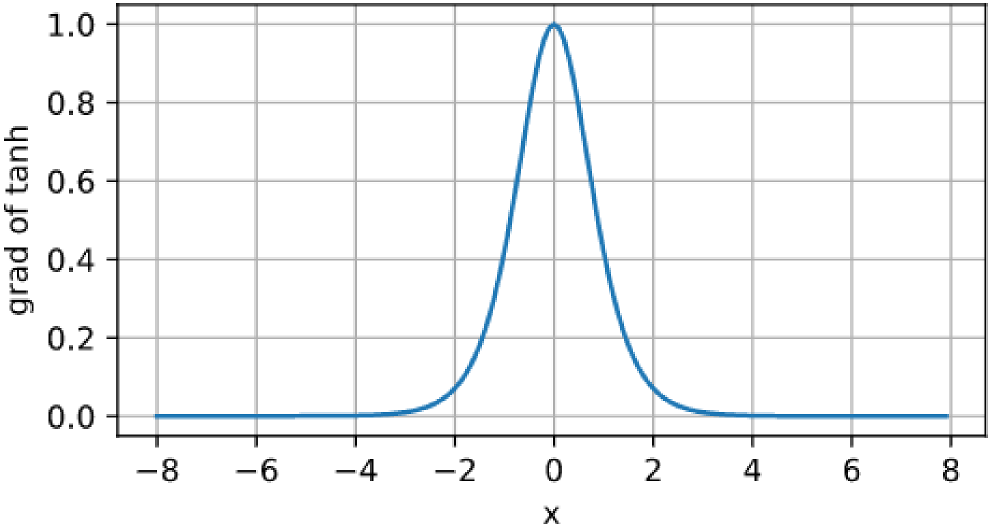
$$\frac{d}{dx}\mathrm{sigmoid}(x) = \frac{\exp(-x)}{(1+\exp(-x))^2}$$

$$= \mathrm{sigmoid}(x)\,(1 - \mathrm{sigmoid}(x))$$

Tanh



$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}.$$

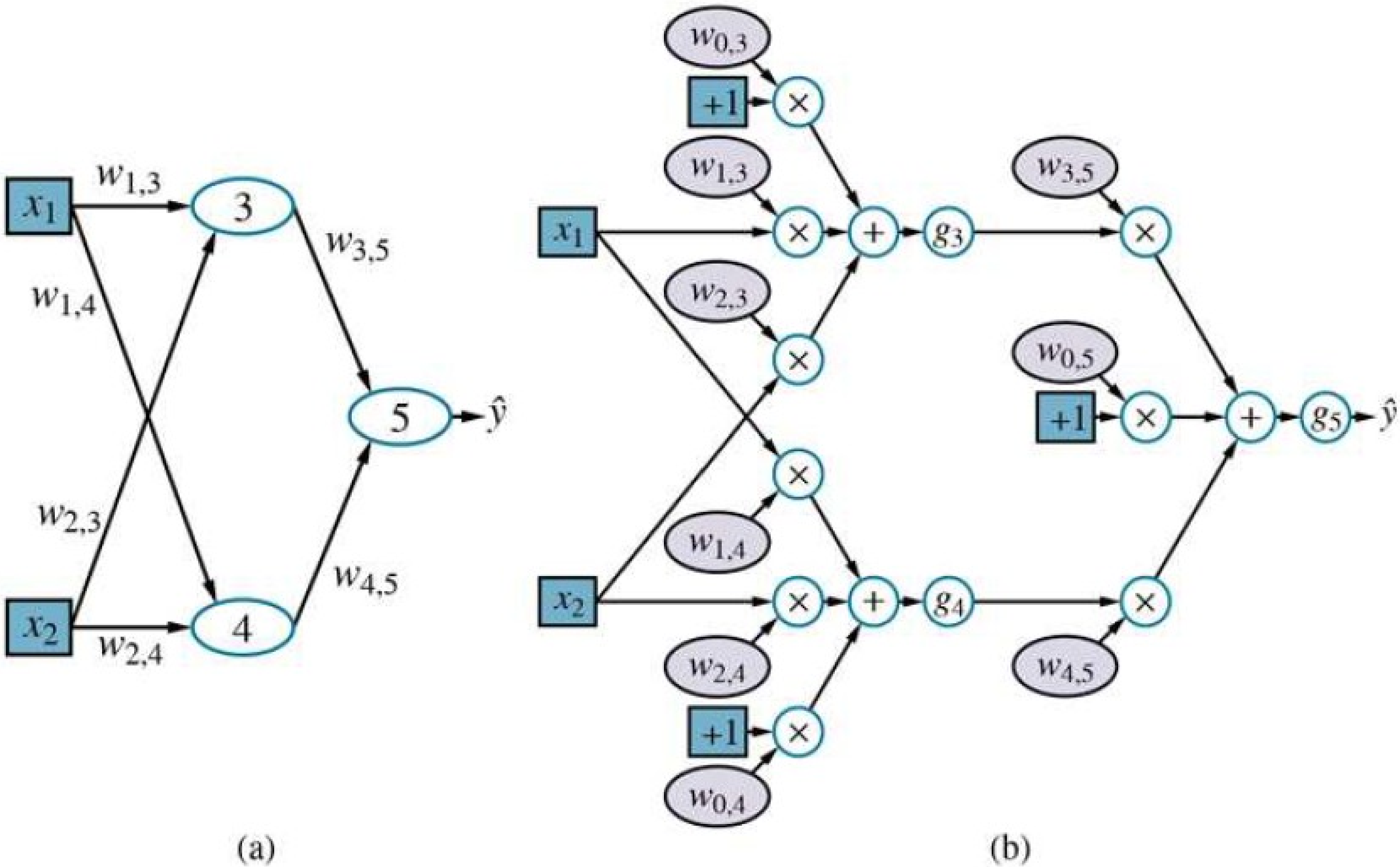


$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$

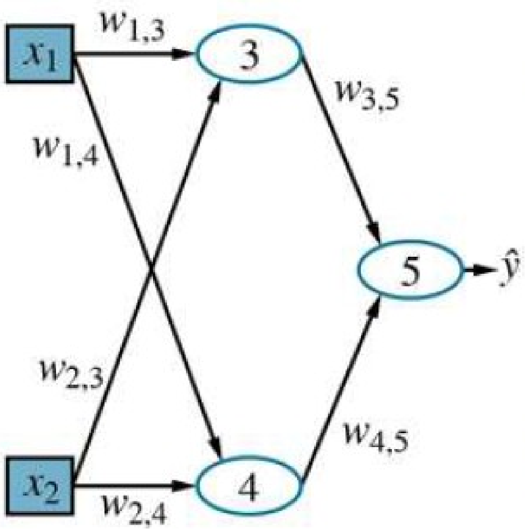
Issue of ReLU not being differentiable at x = 0

- https://www.mldawn.com/relu-activation-function/

Computation Graph



Forward computation



$$
\begin{aligned}
\hat{y} &= g_5(in_5) = g_5(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4) \\
&= g_5(w_{0,5} + w_{3,5}g_3(in_3) + w_{4,5}g_4(in_4)) \\
&= g_5(w_{0,5} + w_{3,5}g_3(w_{0,3} + w_{1,3}x_1 + w_{2,3}x_2) \\
&\qquad + w_{4,5}g_4(w_{0,4} + w_{1,4}x_1 + w_{2,4}x_2))
\end{aligned}
$$

$$
\boxed{h_{\mathbf{w}}(\mathbf{x}) = \mathbf{g}^{(2)}(\mathbf{W}^{(2)}\mathbf{g}^{(1)}(\mathbf{W}^{(1)}\mathbf{x}))}
$$

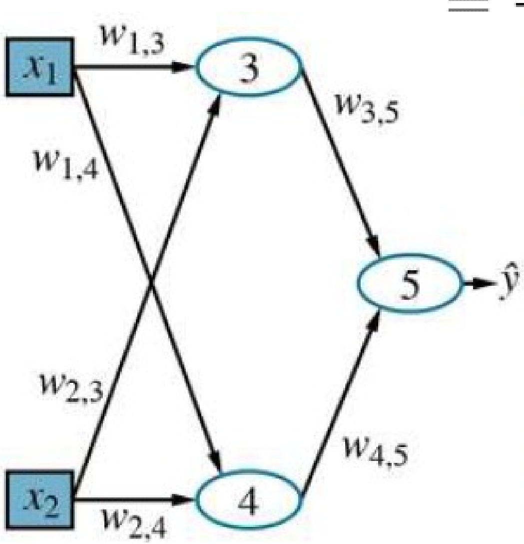
Gradients

$$\frac{\partial}{\partial w_{3,5}} Loss(h_{\mathbf{w}}) = \frac{\partial}{\partial w_{3,5}}(y-\hat{y})^2 = -2(y-\hat{y})\frac{\partial \hat{y}}{\partial w_{3,5}}$$

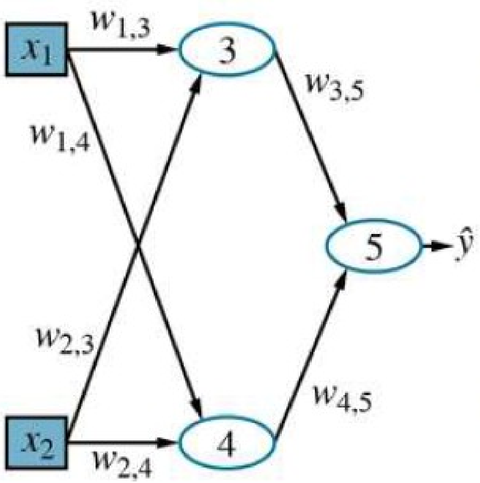
$$= -2(y-\hat{y})\frac{\partial}{\partial w_{3,5}} g_5(in_5) = -2(y-\hat{y})g_5'(in_5)\frac{\partial}{\partial w_{3,5}} in_5$$

$$= -2(y-\hat{y})g_5'(in_5)\frac{\partial}{\partial w_{3,5}}\left(w_{0,5} + w_{3,5}a_3 + w_{4,5}a_4\right)$$

$$= -2(y-\hat{y})g_5'(in_5)a_3.$$



Gradients

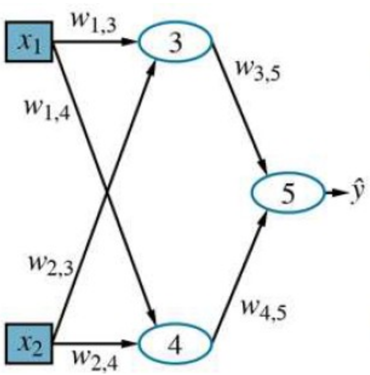


$$\frac{\partial}{\partial w_{3,5}} Loss(h_{\mathbf{w}}) = -2(y - \hat{y}) g_5'(in_5) a_3$$

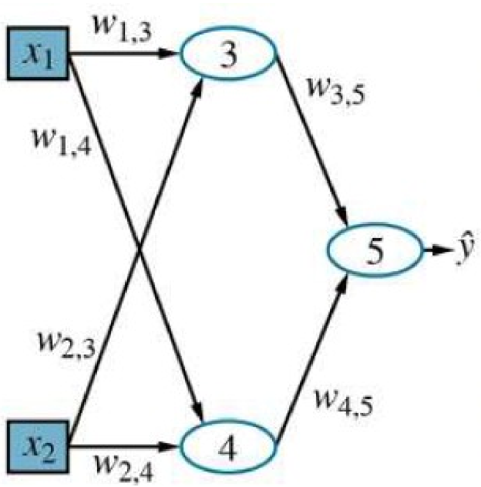
If we define $\Delta_5 = 2(\hat{y} - y) g_5'(in_5)$ as a sort of "perceived error" at the point where unit 5 receives its input, then the gradient with respect to $w_{3,5}$ is just $\Delta_5 a_3$. This makes perfect sense: if Δ_5 is positive, that means $\hat{y}$ is too big (recall that g' is always nonnegative); if a_3 is also positive, then increasing $w_{3,5}$ will only make things worse, whereas if a_3 is negative, then increasing $w_{3,5}$ will reduce the error. The magnitude of a_3 also matters: if a_3 is small for this training example, then $w_{3,5}$ didn't play a major role in producing the error and doesn't need to be changed much.

Gradients

$$
\begin{aligned}
\frac{\partial}{\partial w_{1,3}} Loss(h_{\mathbf{w}}) &= -2(y-\hat{y})g_5'(in_5)\frac{\partial}{\partial w_{1,3}}(w_{0,5}+w_{3,5}a_3+w_{4,5}a_4) \\
&= -2(y-\hat{y})g_5'(in_5)w_{3,5}\frac{\partial}{\partial w_{1,3}}a_3 \\
&= -2(y-\hat{y})g_5'(in_5)w_{3,5}\frac{\partial}{\partial w_{1,3}}g_3(in_3) \\
&= -2(y-\hat{y})g_5'(in_5)w_{3,5}g_3'(in_3)\frac{\partial}{\partial w_{1,3}}in_3 \\
&= -2(y-\hat{y})g_5'(in_5)w_{3,5}g_3'(in_3)\frac{\partial}{\partial w_{1,3}}(w_{0,3}+w_{1,3}x_1+w_{2,3}x_2) \\
&= -2(y-\hat{y})g_5'(in_5)w_{3,5}g_3'(in_3)x_1 .
\end{aligned}
$$



Gradients

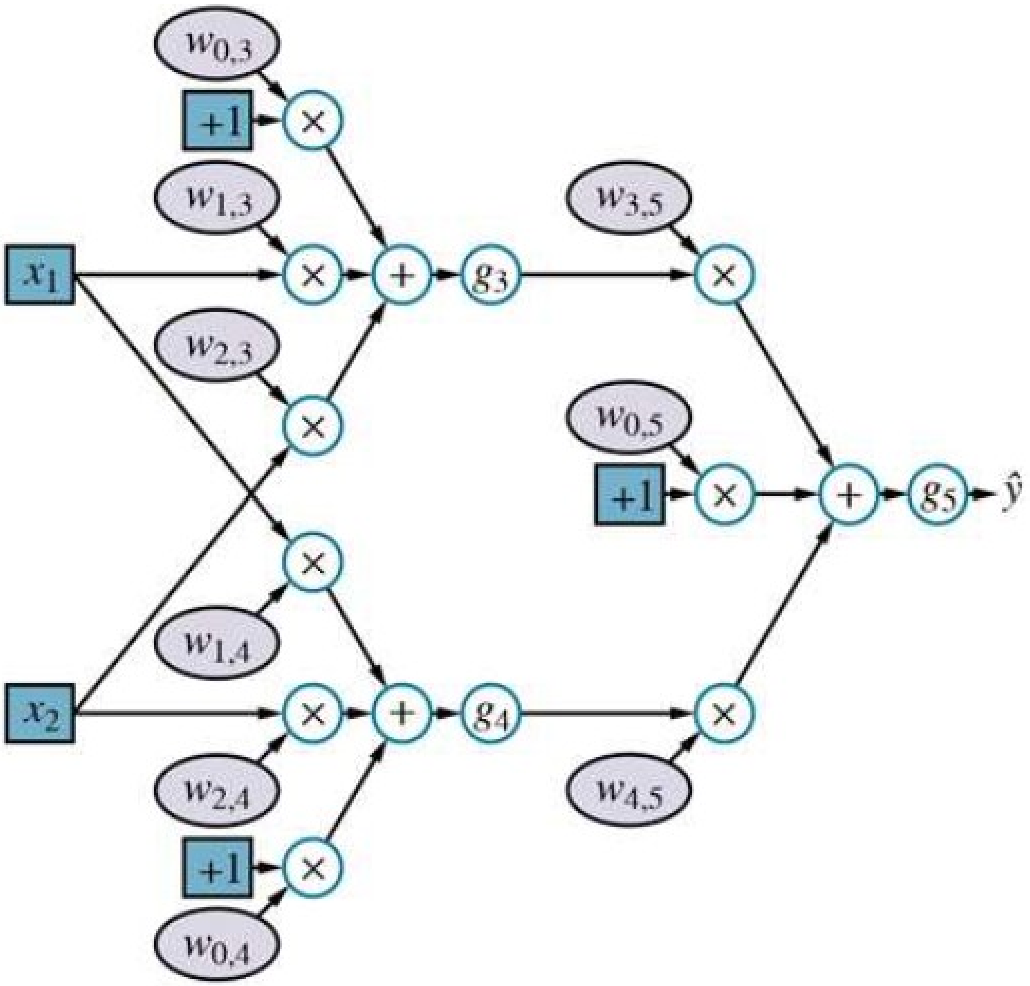


$$\frac{\partial}{\partial w_{1,3}} Loss(h_{\mathbf{w}}) = -2(y - \hat{y})g_5'(in_5)w_{3,5}g_3'(in_3)x_1$$

If we also define $\Delta_3 = \Delta_5 w_{3,5} g_3'(in_3)$, then the gradient for $w_{1,3}$ becomes just $\Delta_3 x_1$. Thus, the perceived error at the input to unit 3 is the perceived error at the input to unit 5, multiplied by information along the path from 5 back to 3. This phenomenon is completely general, and gives rise to the term **back-propagation** for the way that the error at the output is passed back through the network.

Home work

- $w_{0,3} = 2$, $w_{0,4} = -1$, $w_{0,5} = 1$
- $w_{1,3} = 2$, $w_{1,4} = 4$, $w_{2,3} = 3$
- $w_{2,4} = 1$, $w_{3,5} = 2$, $w_{4,5} = 3$
- $x_1 = -3$, $x_2 = 5$
- $y = 0$
- Loss = binary cross entropy
- Non-linearity = ReLU
- y_hat = ?
- Calculate the gradients
  - w.r.t the w's.



Cross-entropy loss

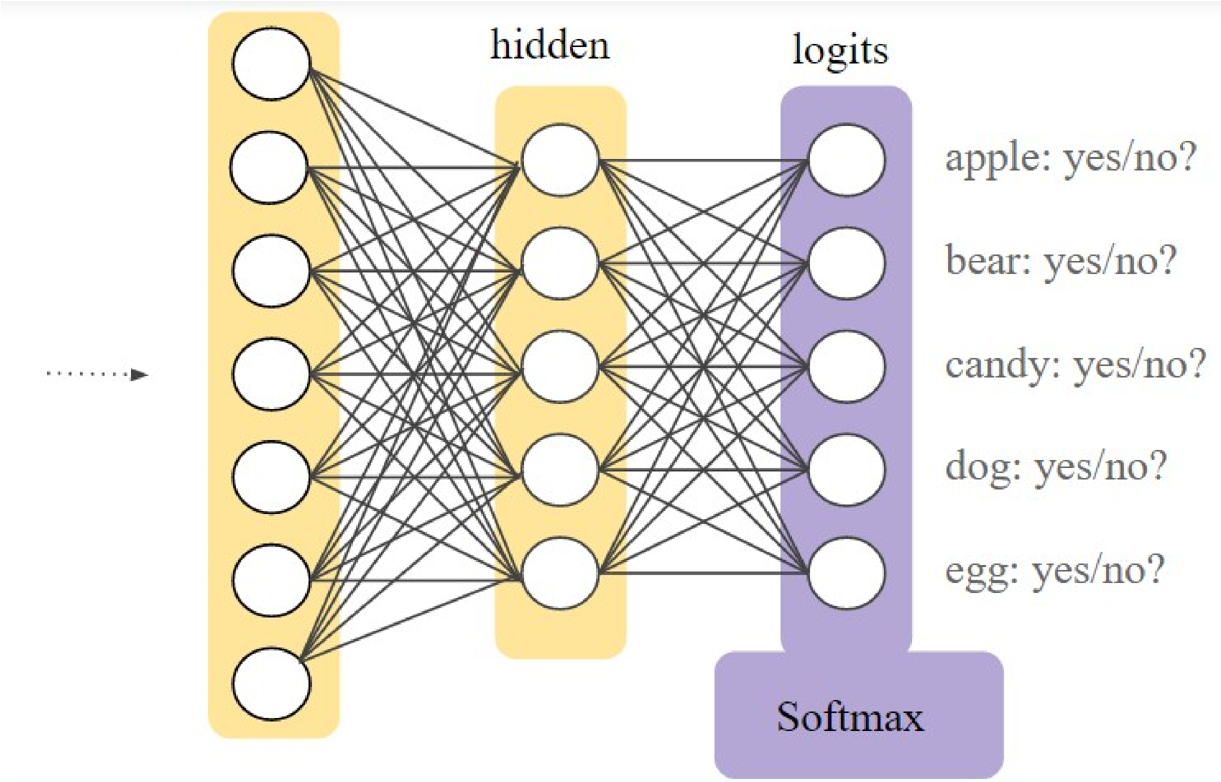
Loss for single data point

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j$$

Loss for *n* data point

$$\sum_{i=1}^{n} l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})$$

Soft-max



$$\text{softmax}(\mathbf{in})_k = \frac{e^{in_k}}{\sum_{k'=1}^{d} e^{in_{k'}}}$$

Shallow vs. Deep network's Performance

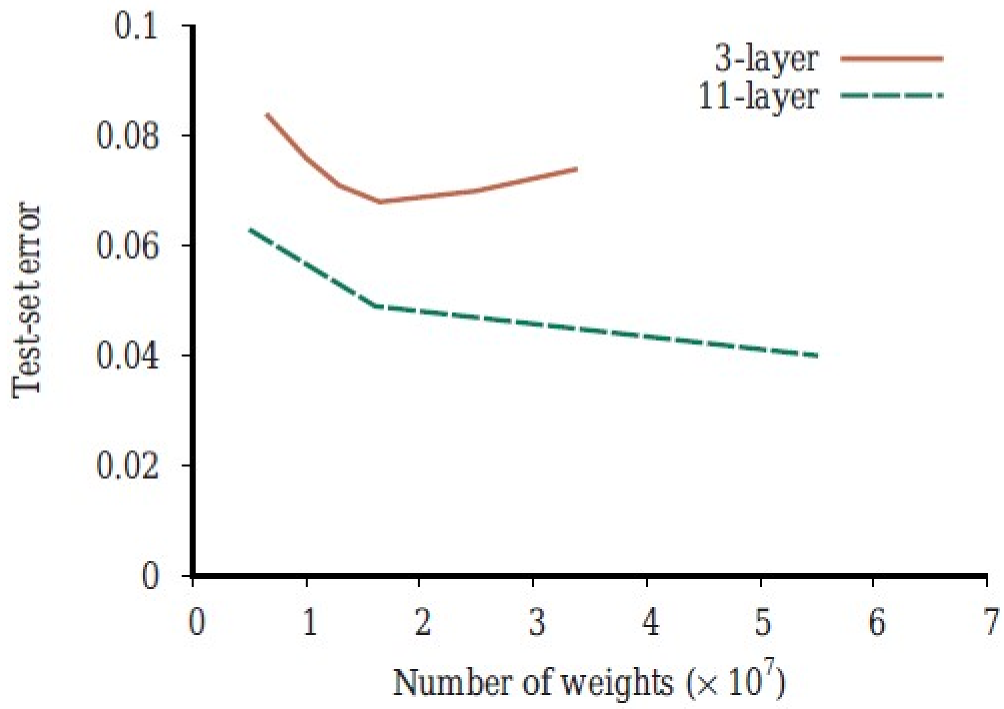


Figure 21.7 Test-set error as a function of layer width (as measured by total number of weights) for three-layer and eleven-layer convolutional networks. The data come from early versions of Google's system for transcribing addresses in photos taken by Street View cars (Goodfellow *et al.*, 2014).

Batch Normalization

$$\mathrm{BN}(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}}}{\hat{\boldsymbol{\sigma}}_{\mathcal{B}}} + \boldsymbol{\beta}$$

Sample mean

Shift parameter

Element-wise
scale parameter

Sample Standard
Deviation

$$\hat{\boldsymbol{\mu}}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbf{x} \text{ and } \hat{\boldsymbol{\sigma}}_{\mathcal{B}}^2 = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} (\mathbf{x} - \hat{\boldsymbol{\mu}}_{\mathcal{B}})^2 + \epsilon$$

- ϒ and β are learned parameters
- Like dropout, the model sees the noise (in mean and SD) only during training
- At prediction time, population mean and standard deviations are used.

Batch Normalization

- Faster convergence
- Regularization
- Works best for moderate minibatch sizes in the 50–100 range.
  - Large minibatch regularizes less due to the more stable estimates,
  - Tiny minibatch destroys useful signal due to high variance.

$$\mathbf{h} = \phi(\mathrm{BN}(\mathbf{W}\mathbf{x} + \mathbf{b}))$$

Dropout

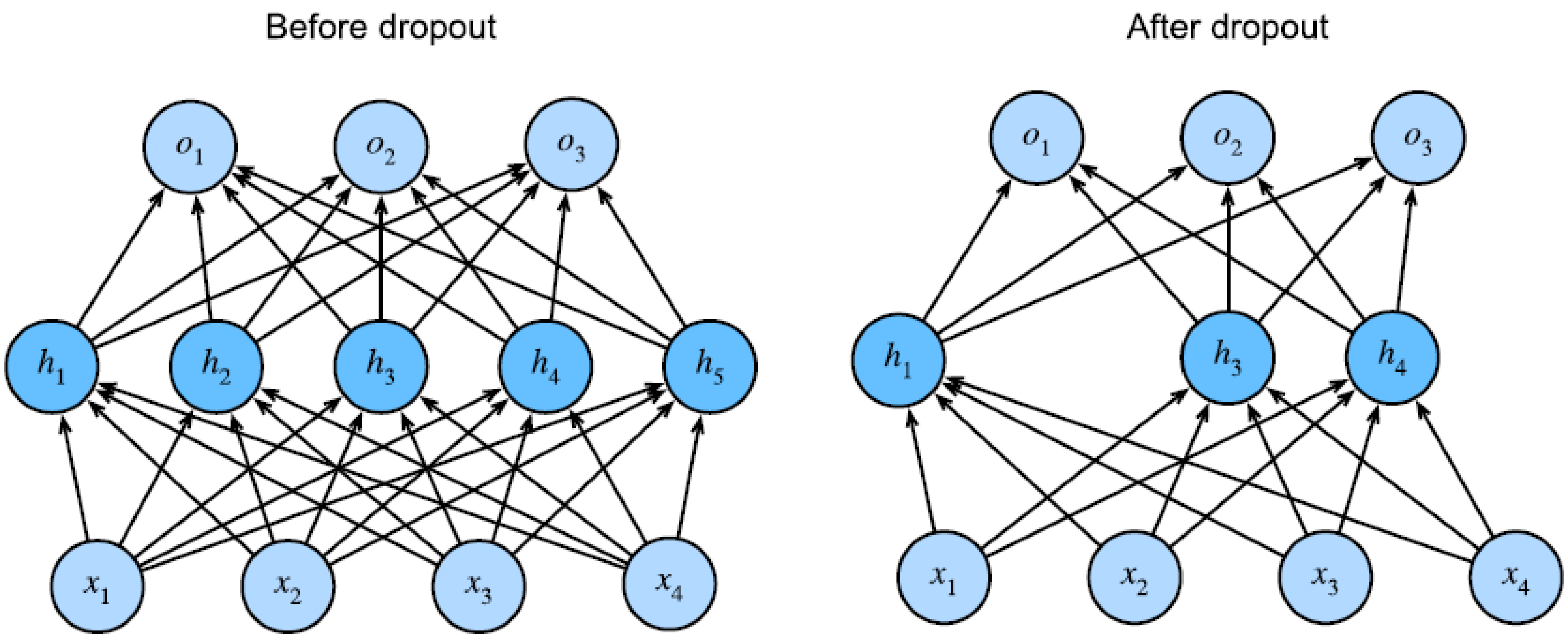


Fig. 4.6.1: MLP before and after dropout.

Dropout

- Drop out some neurons during training
  - On each iteration
  - Layer by layer
  - Different neurons will get dropped in different iterations
- Breaks up co-adaptation

Co-adaptation. *Neural network overfitting is characterized by a state in which each layer relies on a specific pattern of activations in the previous layer.*

Dropout

- Need to normalize the activation of the retained nodes
- Each intermediate activation *h* is replaced by a random variable h'
- Expectation remains unchanged, i.e., $E[h'] = h$.

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases}$$

Dropout

- Used only in the training phase
- Inference uses all the neurons
- Exception
  - Dropout at test time → heuristic for estimating the uncertainty of neural network predictions
    - if the predictions agree across many different dropout outputs, then we might say that the network is more confident.

Gradients

- Vanishing gradient
  - Derivatives can be very close to 0
  - Changing the weights may have negligible effects.
- Sigmoid activation is succeptible to vanishing gradient
- ReLU is more resilient

Xavier Initialization

- Let
  - O_i – output for some fully-connected layer (without nonlinearities)
  - There are n_{in} inputs x_i with associated weights w_{ij}
  - Weights are drawn independently from the same distribution, with 0 mean, σ^2 variance
  - x_i's also have 0 mean, γ^2 variance
    - Independent of weights
    - Independent of each other

$$\boxed{o_i = \sum_{j=1}^{n_{\text{in}}} w_{ij} x_j}$$

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249–256).

Xavier Initialization

$$
\begin{aligned}
E[o_i] &= \sum_{j=1}^{n_{\text{in}}} E[w_{ij} x_j] \\
&= \sum_{j=1}^{n_{\text{in}}} E[w_{ij}] E[x_j] \\
&= 0, \\
\text{Var}[o_i] &= E[o_i^2] - (E[o_i])^2 \\
&= \sum_{j=1}^{n_{\text{in}}} E[w_{ij}^2 x_j^2] - 0 \\
&= \sum_{j=1}^{n_{\text{in}}} E[w_{ij}^2] E[x_j^2] \\
&= n_{\text{in}} \sigma^2 \gamma^2.
\end{aligned}
$$

- Variance can be kept fixed if
  - **$n_{in}\sigma^2 = 1$**
- Following same reasoning, during backprop. Gradients' variance can be kept fixed if
  - **$n_{out}\sigma^2 = 1$**
- Therefore, we try to achieve
  - **$0.5 \times (n_{in} + n_{out})\, \sigma^2 = 1$**

$$\boxed{\sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}}$$

Xavier Initialization

- Sampling weights from $N(0, \sigma^2)$
- Sampling weights from uniform distribution $U(-a, a)$

$$\boxed{U\left(-\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}\right)}$$

Though the assumption for nonexistence of nonlinearities in the above mathematical reasoning can be easily violated in neural networks, the Xavier initialization method turns out to work well in practice.

Gradient of cross entropy with softmax

- https://fanzengau.com/myblog/content/machine_learning/gradient_of_categorical_cross_entropy/gradient_of_categorical_cross_entropy.html