Introduction to neural networks

Herman Kamper

http://www.kamperh.com/

Introduction to neural networks

Preliminaries

Herman Kamper

http://www.kamperh.com/

Vector and matrix derivatives recap (Denominator layout)

$$\frac{\partial J}{\partial \underline{W}}$$

Derivative of a scalar function $f : \mathbb{R}^N \to \mathbb{R}$ with respect to vector $\mathbf{x} \in \mathbb{R}^N$:

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \triangleq \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_N} \end{bmatrix}$$

Derivative of a vector function $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$, where $\boldsymbol{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) & f_2(\mathbf{x}) & \cdots & f_M(\mathbf{x}) \end{bmatrix}^\top$, with respect to vector $\mathbf{x} \in \mathbb{R}^N$:

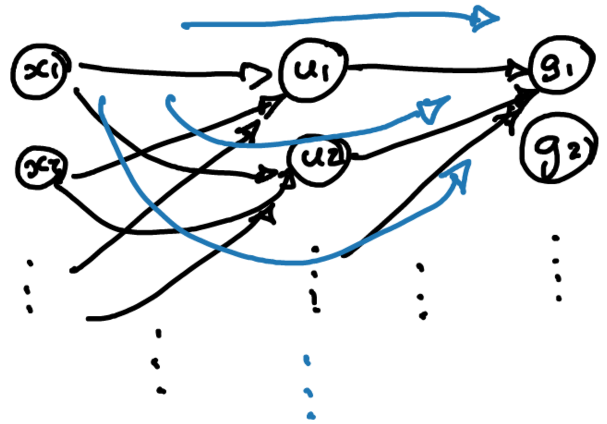
$$\frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial \mathbf{x}} \triangleq \begin{bmatrix} \frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial x_1} \\ \frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial \boldsymbol{f}(\mathbf{x})}{\partial x_N} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_M(\mathbf{x})}{\partial x_1} \\ \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_M(\mathbf{x})}{\partial x_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_1(\mathbf{x})}{\partial x_N} & \frac{\partial f_2(\mathbf{x})}{\partial x_N} & \cdots & \frac{\partial f_M(\mathbf{x})}{\partial x_N} \end{bmatrix}$$

Transpose of Jacobian

Derivative of a scalar function $f : \mathbb{R}^{M \times N} \to \mathbb{R}$
with respect to matrix $\mathbf{X} \in \mathbb{R}^{M \times N}$:

$$\frac{\partial f(\mathbf{X})}{\partial \mathbf{X}} \triangleq \begin{bmatrix} \frac{\partial f(\mathbf{X})}{\partial X_{1,1}} & \frac{\partial f(\mathbf{X})}{\partial X_{1,2}} & \cdots & \frac{\partial f(\mathbf{X})}{\partial X_{1,N}} \\ \frac{\partial f(\mathbf{X})}{\partial X_{2,1}} & \frac{\partial f(\mathbf{X})}{\partial X_{2,2}} & \cdots & \frac{\partial f(\mathbf{X})}{\partial X_{2,N}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f(\mathbf{X})}{\partial X_{M,1}} & \frac{\partial f(\mathbf{X})}{\partial X_{M,2}} & \cdots & \frac{\partial f(\mathbf{X})}{\partial X_{M,N}} \end{bmatrix}$$

Using the above definitions, we can generalise the chain rule. Given $\mathbf{u}$ is a function of $\mathbf{x}$, and $\boldsymbol{g}$ in turn is a vector function of $\mathbf{u}$, the vector-by-vector chain rule states:



$$\frac{\partial \boldsymbol{g}(\mathbf{u})}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \frac{\partial \boldsymbol{g}(\mathbf{u})}{\partial \mathbf{u}}$$

Order matters!

$$\frac{\partial g_1}{\partial x_1} \leftarrow \begin{bmatrix} \circ & \\ & \end{bmatrix}$$

This generalised chain rule comes from the chain rule for multivariate functions. For scalars where g depends on u_1 and u_2, which in turn depends on x, we have:

$$\frac{\partial g}{\partial x} = \frac{\partial u_1}{\partial x} \frac{\partial g}{\partial u_1} + \frac{\partial u_2}{\partial x} \frac{\partial g}{\partial u_2}$$

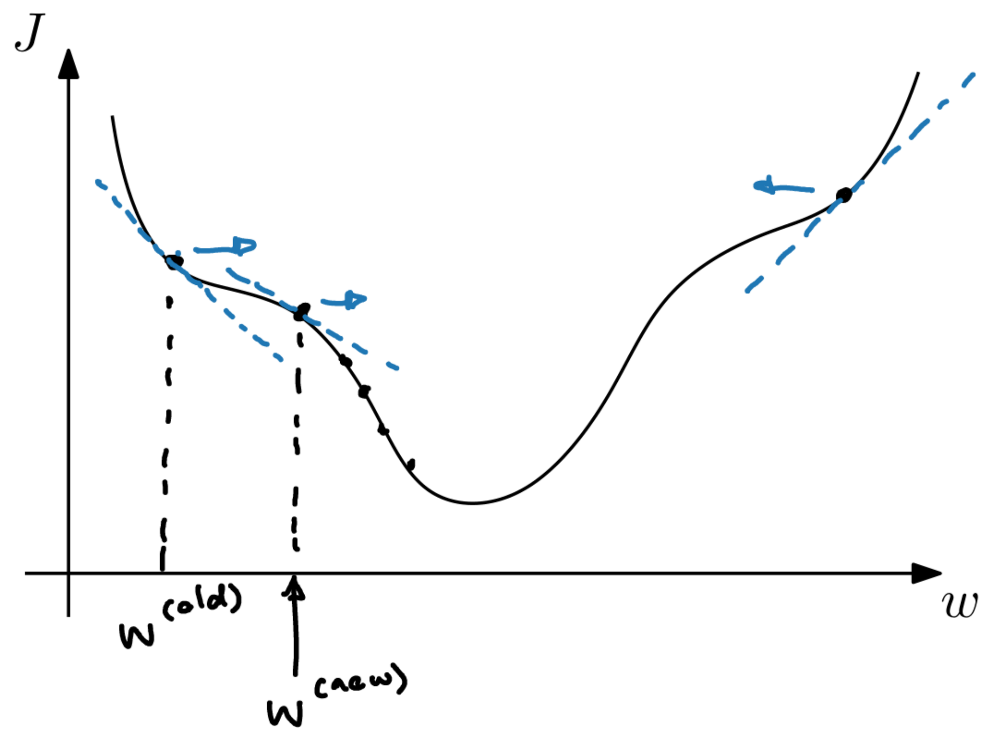
$$\begin{tikzcd} & u_1 \arrow[dr] & \\ x \arrow[ur] \arrow[dr] & & g \\ & u_2 \arrow[ur] & \end{tikzcd}$$

The chain rule of scalars:

$$\frac{dg}{dx} = \frac{du}{dx} \cdot \frac{dg}{du}$$

$$\begin{tikzcd} x \arrow[r] & u \arrow[r] & g \end{tikzcd}$$

Gradient descent recap



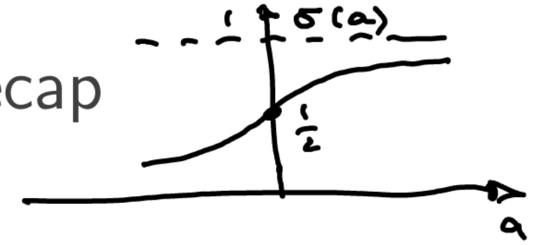
$$w \leftarrow w - \eta \frac{dJ}{dw}$$

Learning rate

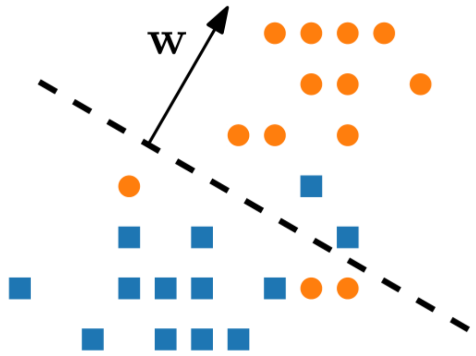
$$\underline{w} \leftarrow \underline{w} - \eta \frac{\partial J}{\partial \underline{w}}$$

$$\underline{w}^{(new)} = \underline{w}^{(old)} - \eta \left.\frac{\partial J}{\partial \underline{w}}\right|_{\underline{w}=\underline{w}^{(old)}}$$

Logistic regression, softmax regression, basis functions recap



- Prediction function: $f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \dfrac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}$



- Interpret function as: $f_{\mathbf{w}}(\mathbf{x}) = P_{\mathbf{w}}(y=1|\mathbf{x})$

- With labels $\underline{y \in \{0,1\}}$, minimise the negative log likelihood: (NLL)

$$J(\mathbf{w}) = -\log \prod_{n=1}^{N} P_{\mathbf{w}}(y^{(n)}|\mathbf{x}^{(n)})$$

$$= -\sum_{n=1}^{N} \left[y^{(n)} \log f_{\mathbf{w}}(\mathbf{x}^{(n)}) + (1-y^{(n)}) \log\left(1 - f_{\mathbf{w}}(\mathbf{x}^{(n)})\right)\right]$$

Softmax regression: $y \in \{1,2,\ldots,K\}$

- Gradient: $\dfrac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -\sum_{n=1}^{N} \left(y^{(n)} - f_{\mathbf{w}}(\mathbf{x}^{(n)})\right)\mathbf{x}^{(n)}$

Basis functions:

$f_{\underline{w}}(\underline{x}) = \sigma(\underline{w}^T \underline{\phi}(\underline{x}))$

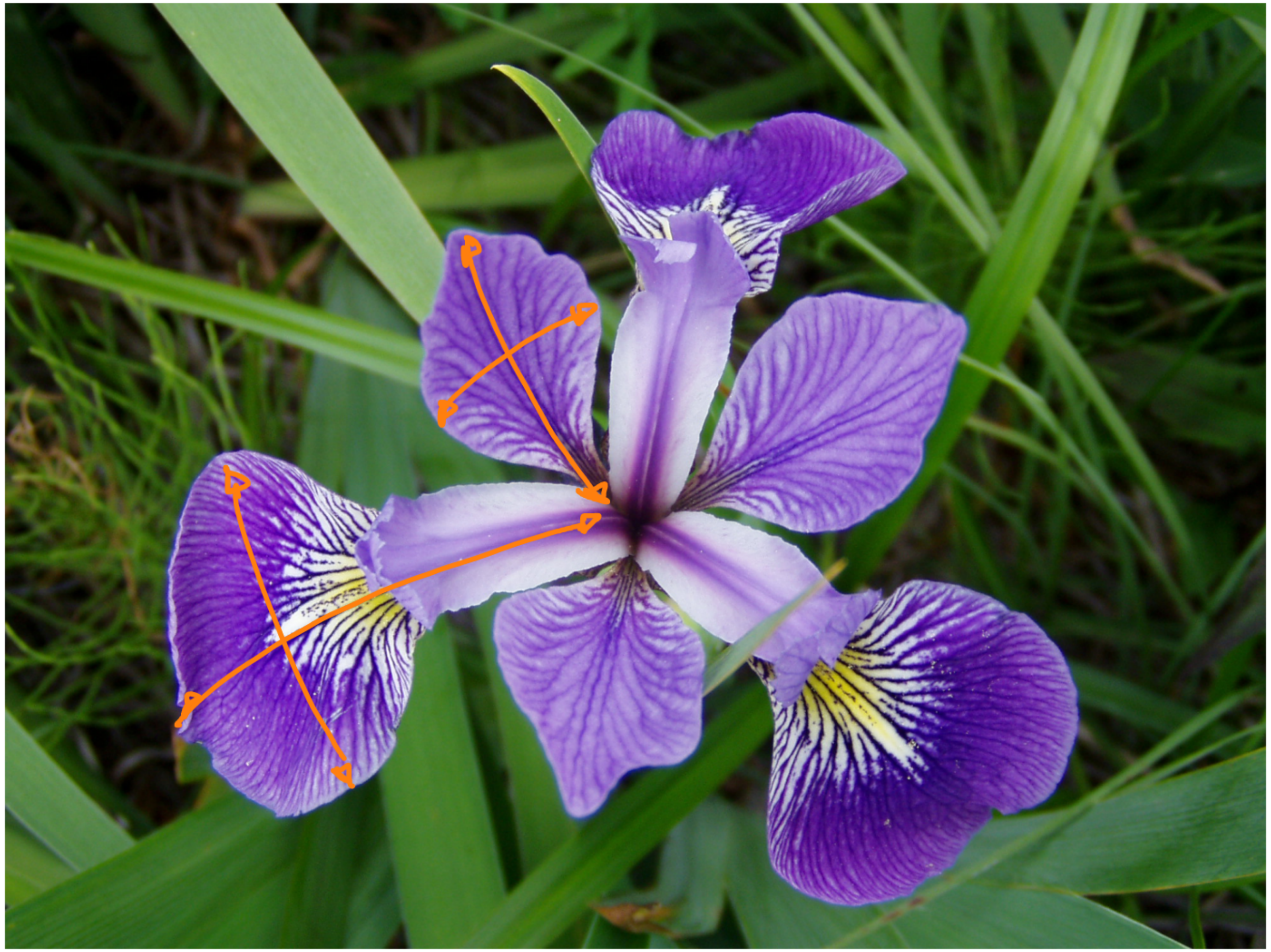
Introduction to neural networks

From logistic regression to neural networks

Herman Kamper

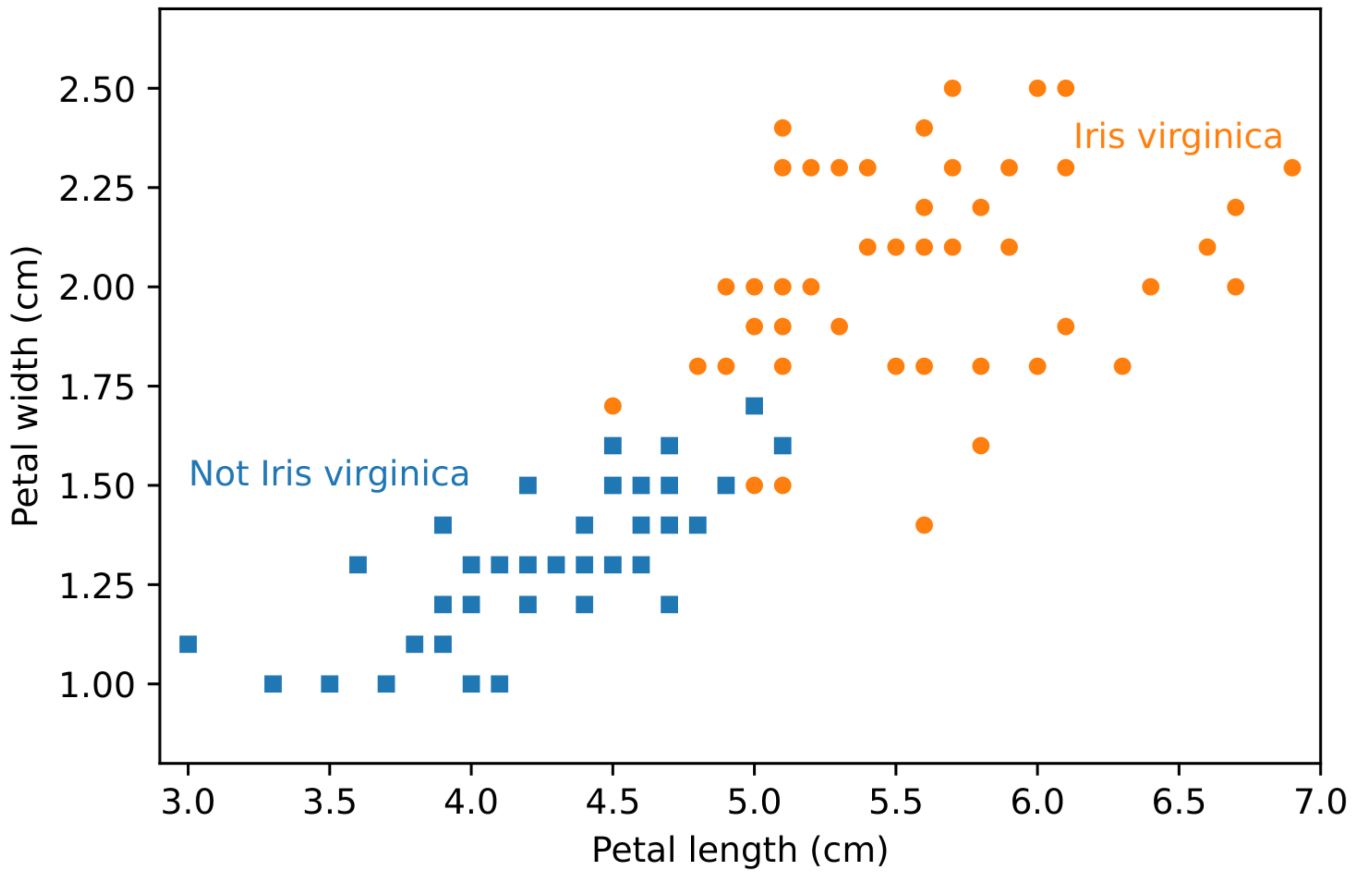
http://www.kamperh.com/

Binary classification of irises



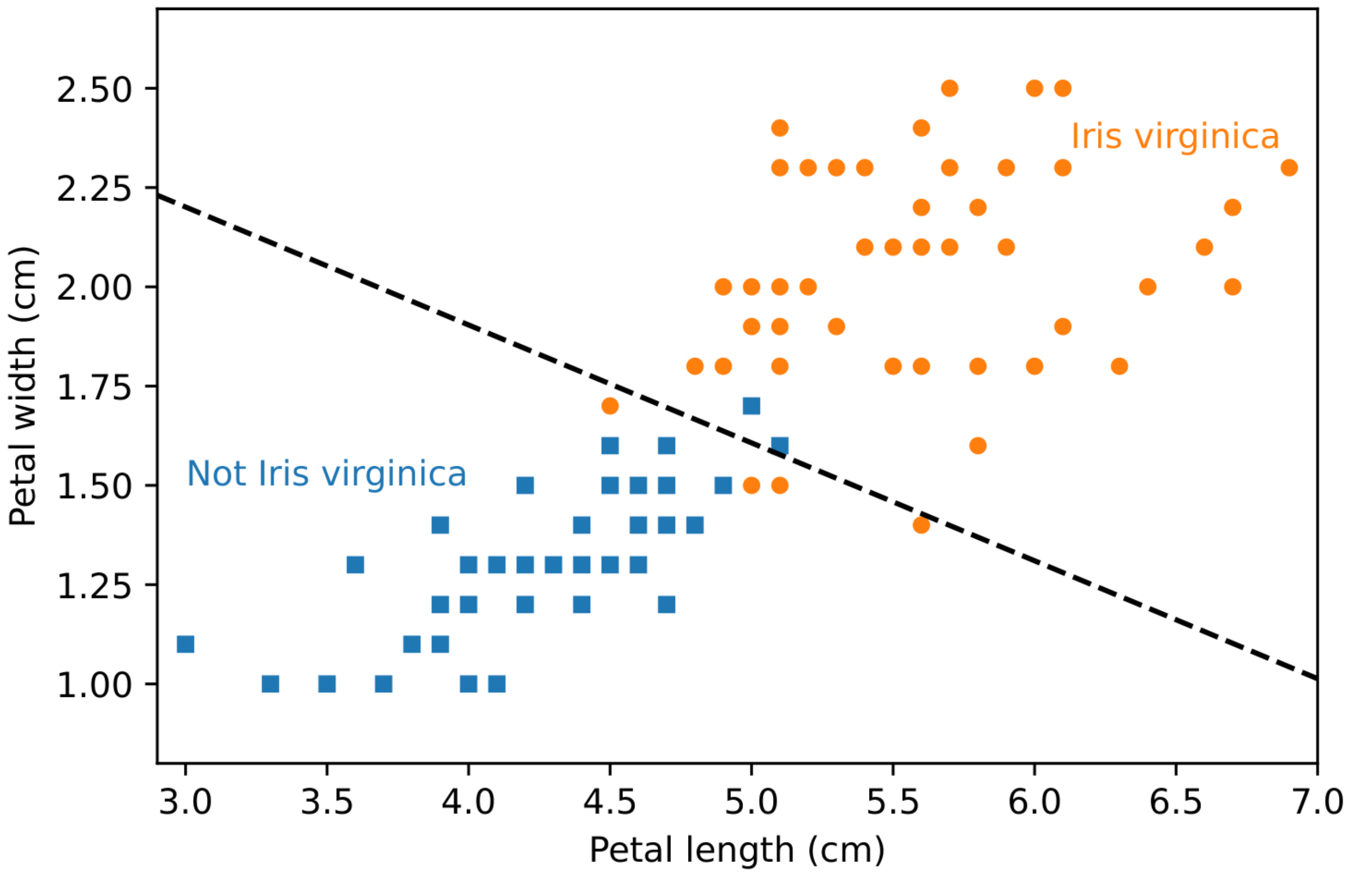
Source: `https://en.wikipedia.org/wiki/Iris_flower_data_set`

Logistic regression

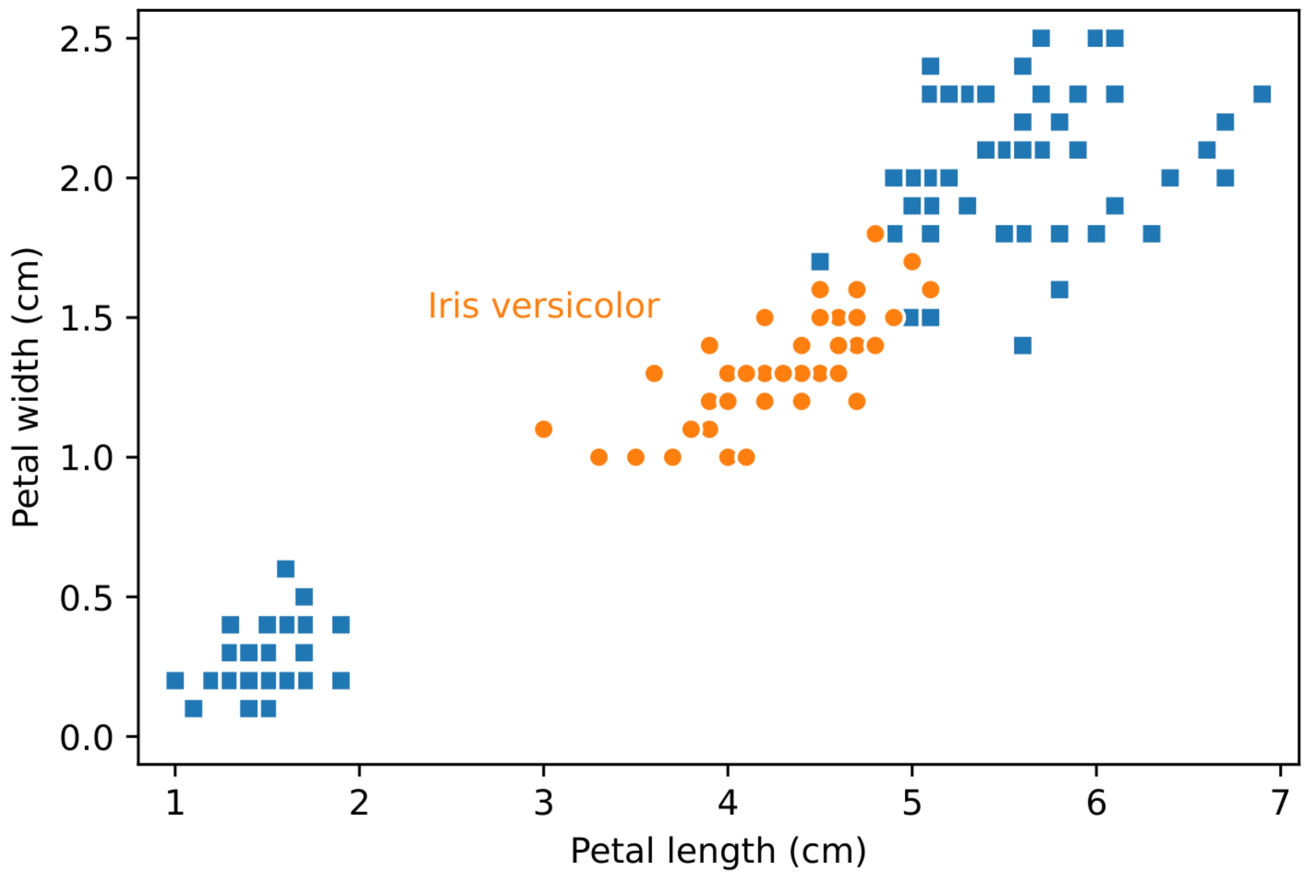


$\underline{x}$

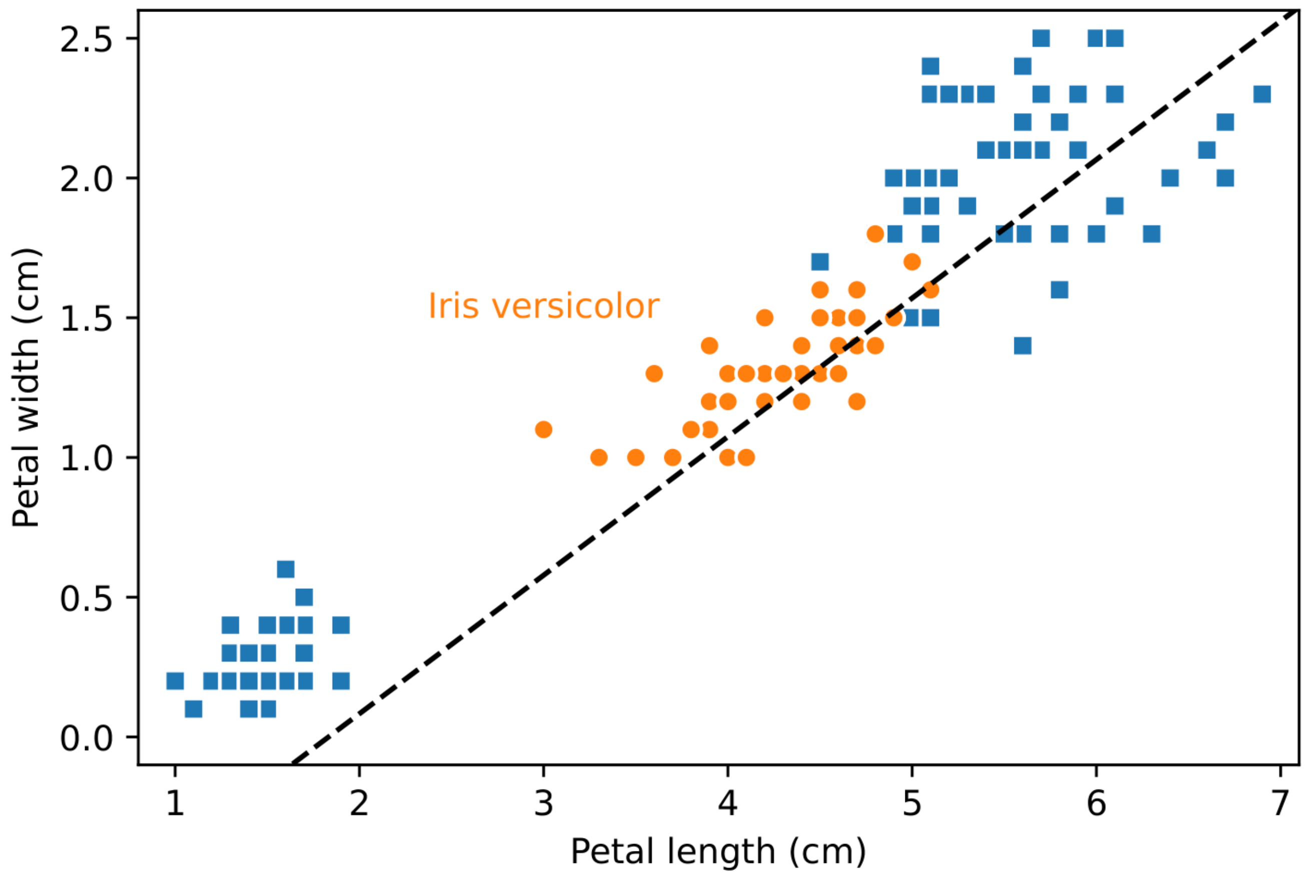
Logistic regression



Nonlinear logistic regression



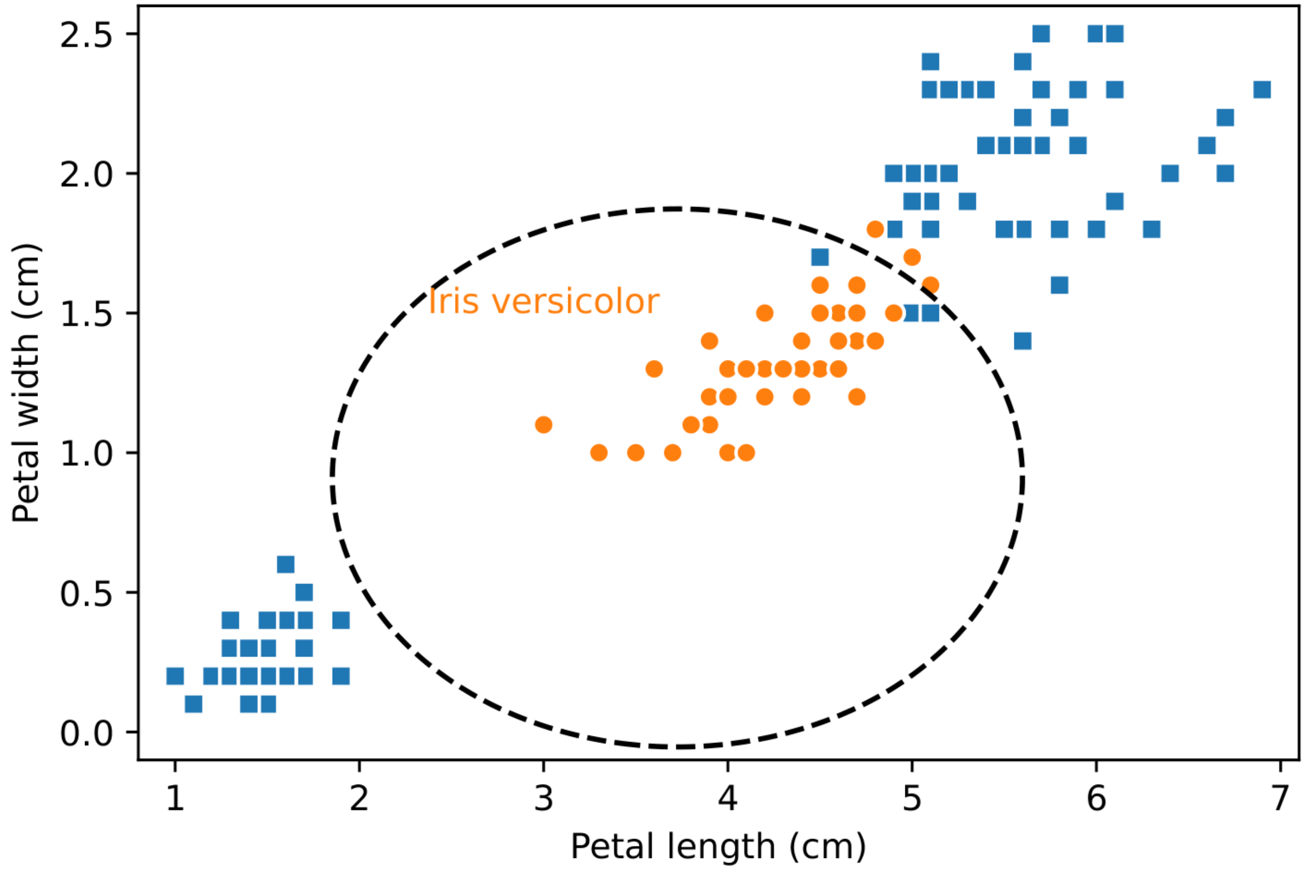
Nonlinear logistic regression



$\underline{x}$

$\underline{\phi}(\underline{x})$

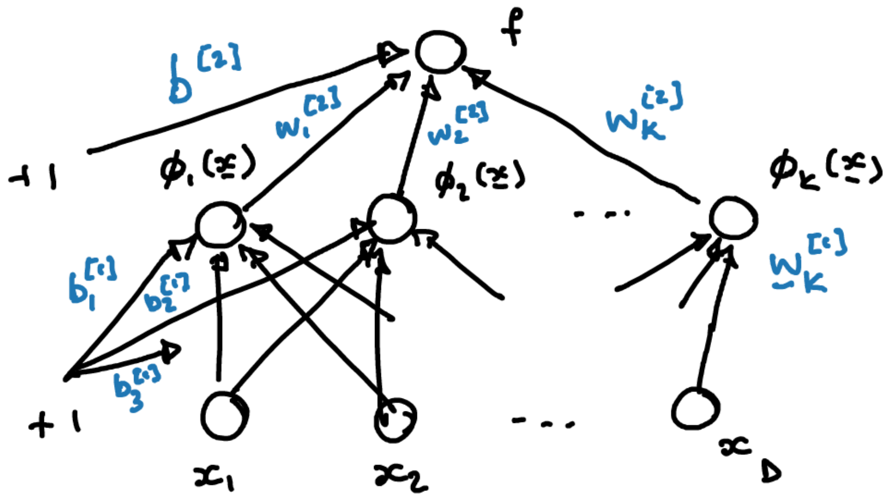
Nonlinear logistic regression



$$\underline{\phi}(\underline{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

$$\underline{f}_w(\underline{x}) = \sigma\left(\underline{w}^T \underline{\phi}(\underline{x})\right)$$

Binary logistic regression with basis functions as a neural network



Logistic regression with basis functions:

$$f_{\underline{w}}(\underline{x}) = \sigma(\underline{w}^T \underline{\phi}(\underline{x})) = \frac{1}{1+\exp(-\underline{w}^T\underline{\phi}(\underline{x}))}$$

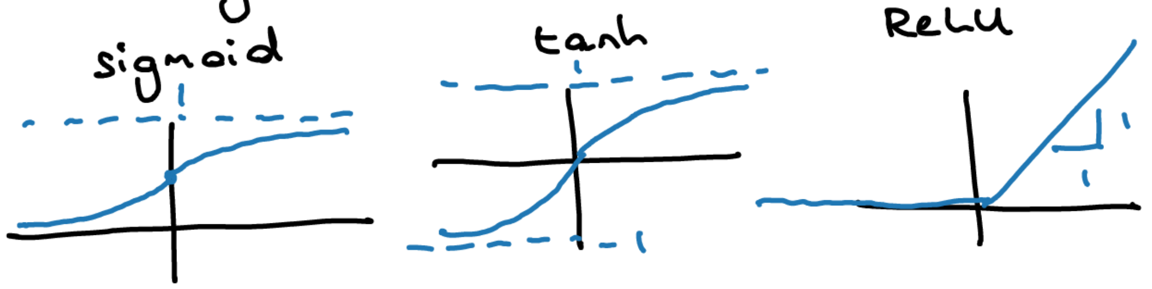
We can set:

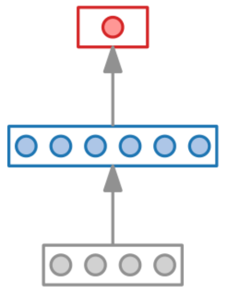
$$\phi_k(\underline{x}) = \sigma(\underline{w}_k^T \underline{x} + b_k)$$

or more generally:

$$\phi_k(\underline{x}) = g(\underline{w}_k^T \underline{x} + b_k)$$

where g is some non-linear function.





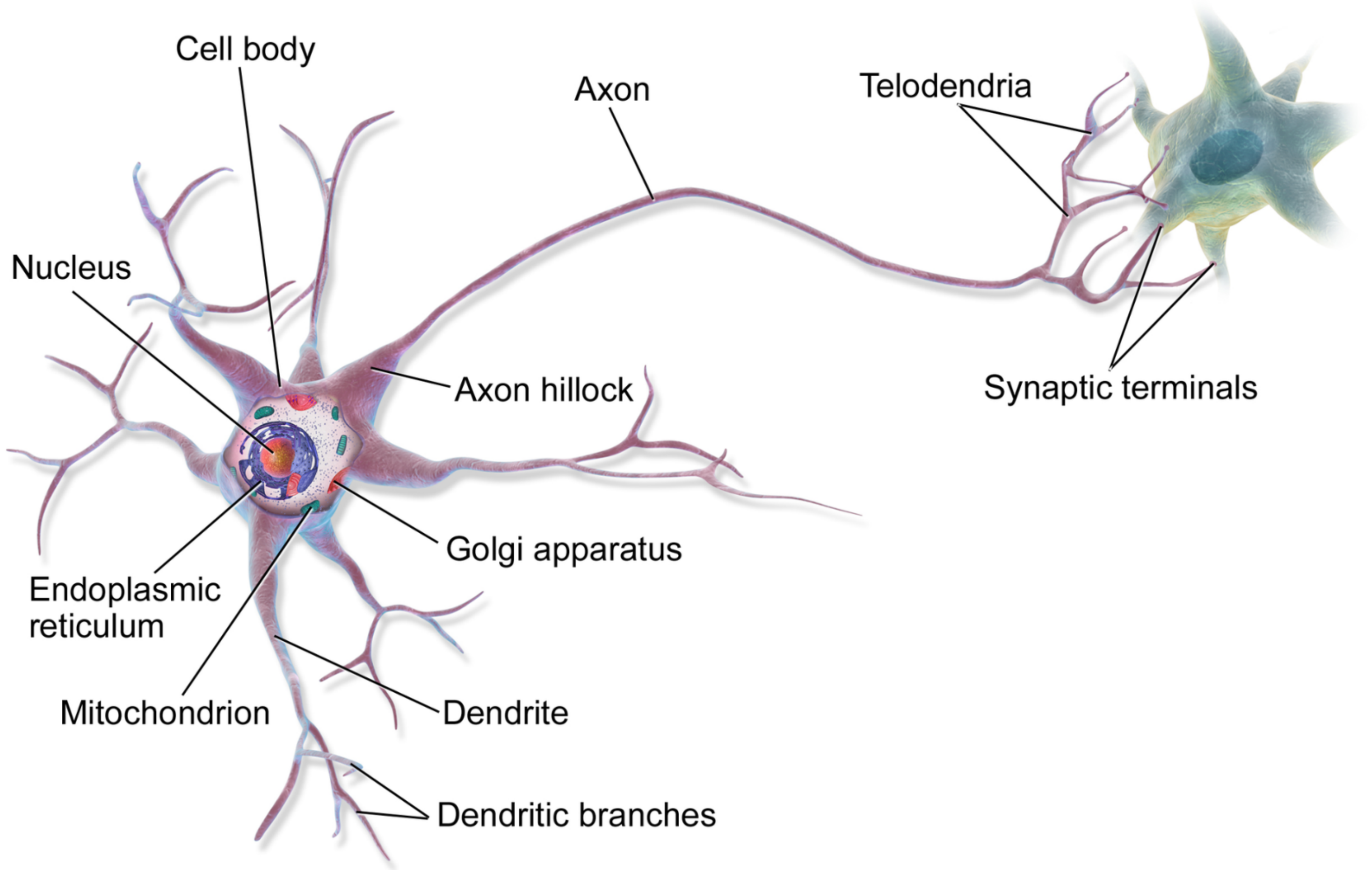
$$f_{\underline{\theta}}(\underline{x}) = \hat{y} = g(\underline{w}^{[2]T}\underline{a}^{[1]} + b^{[2]})$$

$$\underline{\phi}(\underline{x}) = \underline{a}^{[1]} = g(\underline{W}^{[1]}\underline{x} + \underline{b}^{[1]})$$

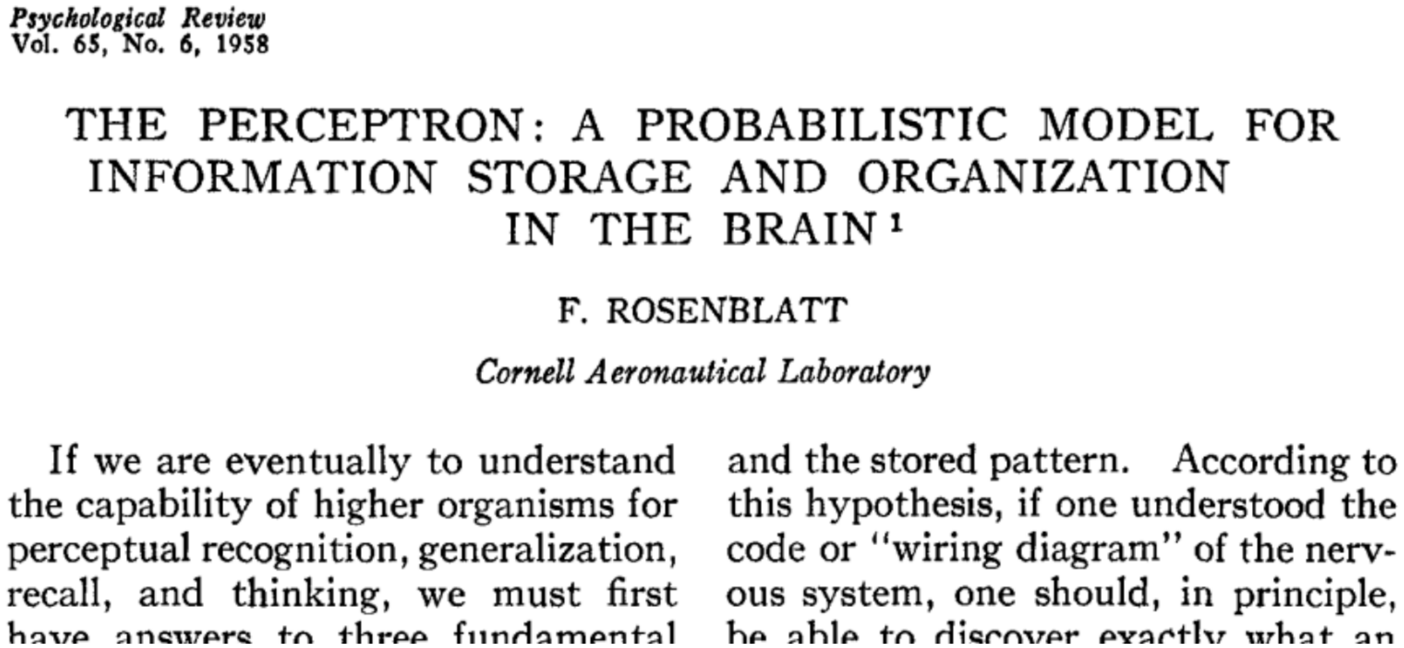
$$\underline{\theta} = \{\underline{W}^{[1]}, \underline{b}^{[1]}, \underline{w}^{[2]}, b^{[2]}\}$$

$\underline{x}$

Why is it called a (artificial) neural network?



Source: `https://en.wikipedia.org/wiki/Neuron#/media/File:Blausen_0657_MultipolarNeuron.png`


Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an

Introduction to neural networks

Backpropagation (without forks)

Herman Kamper

http://www.kamperh.com/

Example: Binary classification with a feedforward neural network

- How do we fit the parameters of our binary classification model?
  As usual: Use gradient descent to minimise the negative log likelihood.

$\hat{y}^{(n)} = f_{\underline{\theta}}(\underline{x}^{(n)})$

$\hat{y}^{(n)} \in [0, 1]$

- If we have a single training item $(\mathbf{x}^{(n)}, y^{(n)})$:

$$J(\boldsymbol{\theta}) = -\left[y^{(n)} \log \hat{y}^{(n)} + (1 - y^{(n)}) \log(1 - \hat{y}^{(n)})\right] = \begin{cases} -\log \hat{y}^{(n)} & \text{if } y^{(n)} = 1 \\ -\log(1 - \hat{y}^{(n)}) & \text{if } y^{(n)} = 0 \end{cases}$$

- Need $\frac{\partial J}{\partial \mathbf{u}}$, where $\mathbf{u}$ is each of the parameters:

$$\underline{\theta} = \left\{ \mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \mathbf{w}^{[2]}, b^{[2]} \right\}$$

$$\frac{\partial J}{\partial \underline{w}^{[2]}} = ?$$

- The *backpropagation algorithm* gives a principled procedure to obtain these gradients:
  Apply the chain rule while reusing previously computed results.

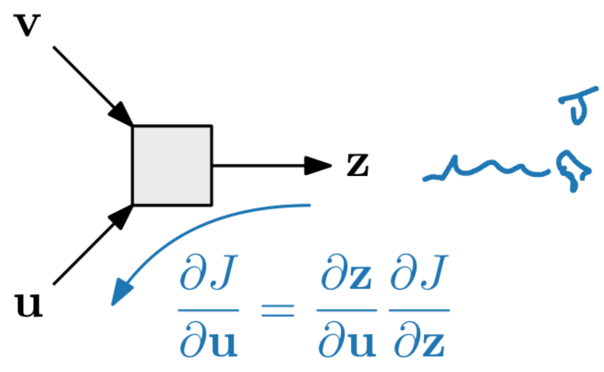
The backpropagation algorithm (without forks)

- Represent your neural network as a computational graph.
- **Forward pass:** Start at the inputs and calculate the output of each operation in the graph. Store these values.
- **Backward pass:** Start at the output of the graph and move backwards. For each operation:
  (a) Determine and calculate the derivative of the output variable w.r.t. each of the input variables to the operation.
  (b) For each input variable $\mathbf{u}$, set

$$\boldsymbol{\delta}_{\mathbf{u}} = \frac{\partial J}{\partial \mathbf{u}} = \frac{\partial \mathbf{z}}{\partial \mathbf{u}} \frac{\partial J}{\partial \mathbf{z}}$$

  where $\mathbf{z}$ is the output of the operation taking $\mathbf{u}$ as input.
- Use the calculated derivates to do take a gradient step to update the parameters. Repeat from the forward pass.

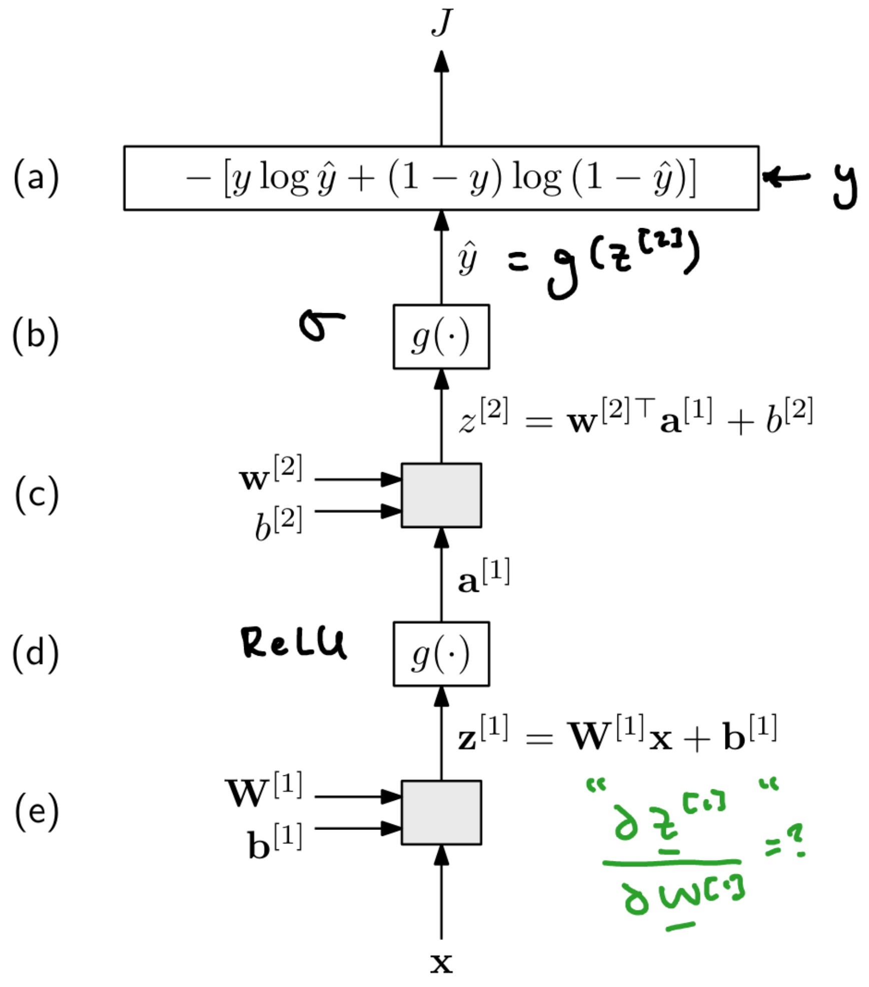
$\frac{\partial \underline{z}}{\partial \underline{v}}$ and $\frac{\partial \underline{z}}{\partial \underline{u}}$



$$\frac{\partial J}{\partial \mathbf{u}} = \frac{\partial \mathbf{z}}{\partial \mathbf{u}} \frac{\partial J}{\partial \mathbf{z}}$$

Example

Single training item: $(\underline{x}^{(n)}, y^{(n)})$


J
(a)
$-[y\log\hat{y} + (1-y)\log(1-\hat{y})]$
y
$\hat{y} = g(z^{[2]})$
(b)
σ
$g(\cdot)$
$z^{[2]} = \mathbf{w}^{[2]\top}\mathbf{a}^{[1]} + b^{[2]}$
(c)
$\mathbf{w}^{[2]}$
$b^{[2]}$
$\mathbf{a}^{[1]}$
(d)
ReLU
$g(\cdot)$
$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$
(e)
$\mathbf{W}^{[1]}$
$\mathbf{b}^{[1]}$
$\mathbf{x}$
$“\frac{\partial \underline{z}^{[1]}}{\partial \underline{W}^{[1]}}” = ?$

(a) $$\frac{\partial J}{\partial \hat{y}} = -\frac{\partial}{\partial \hat{y}}\left[y\log\hat{y} + (1-y)\log(1-\hat{y})\right]$$
$$= -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$
$$\delta_{\hat{y}} = \frac{\partial J}{\partial \hat{y}}\Bigg|_{\substack{\underline{x}=\underline{x}^{(n)}\\ y=y^{(n)}}} = -\frac{y^{(n)}}{\hat{y}^{(n)}} + \frac{1-y^{(n)}}{1-\hat{y}^{(n)}}$$

(b) $$\frac{\partial \hat{y}}{\partial z^{[2]}} = g'(z^{[2]})$$
$$\delta_{z^{[2]}} = \frac{\partial J}{\partial z^{[2]}}\Bigg|_{\substack{\underline{x}=\underline{x}^{(n)}\\ y=y^{(n)}}} = \frac{\partial \hat{y}}{\partial z^{[2]}}\cdot\frac{\partial J}{\partial \hat{y}}$$
$$= g'(z^{[2]})\cdot\delta_{\hat{y}}$$

(c) $$\frac{\partial z^{[2]}}{\partial \underline{w}^{[2]}} = \frac{\partial}{\partial \underline{w}^{[2]}}\left[\underline{w}^{[2]T}\underline{a}^{[1]} + b^{[2]}\right]$$
$$= \underline{a}^{[1]}$$
$$\delta_{\underline{w}^{[2]}} = \frac{\partial J}{\partial \underline{w}^{[2]}} = \frac{\partial z^{[2]}}{\partial \underline{w}^{[2]}}^{\checkmark}\cdot\frac{\partial J}{\partial z^{[2]}}^{\checkmark} = \underline{a}^{[1]}\cdot\delta_{z^{[2]}}$$
$$\delta_{b^{[2]}} = \frac{\partial J}{\partial b^{[2]}} = \frac{\partial z^{[2]}}{\partial b^{[2]}}^{\checkmark}\cdot\frac{\partial J}{\partial z^{[2]}}^{\checkmark}$$
$$\delta_{\underline{a}^{[1]}} = \frac{\partial J}{\partial \underline{a}^{[1]}} = \frac{\partial z^{[2]}}{\partial \underline{a}^{[1]}}^{\checkmark}\cdot\frac{\partial J}{\partial z^{[2]}}^{\checkmark}$$

(e) $\underline{\delta}_{\underline{W}^{[1]}} = \frac{\partial J}{\partial \underline{W}^{[1]}}$ " $= \frac{\partial \underline{z}^{[1]}}{\partial \underline{W}^{[1]}} \cdot \frac{\partial J}{\partial \underline{z}^{[1]}}$ "

$$= \underline{\delta}_{\underline{z}^{[1]}} \, \underline{x}^{T}$$

$$\underline{\delta}_{\underline{b}^{[1]}} = \frac{\partial J}{\partial \underline{b}^{[1]}} = \frac{\partial \underline{z}^{[1]}}{\partial \underline{b}^{[1]}} \cdot \underline{\delta}_{\underline{z}^{[1]}}$$

Why does this work?

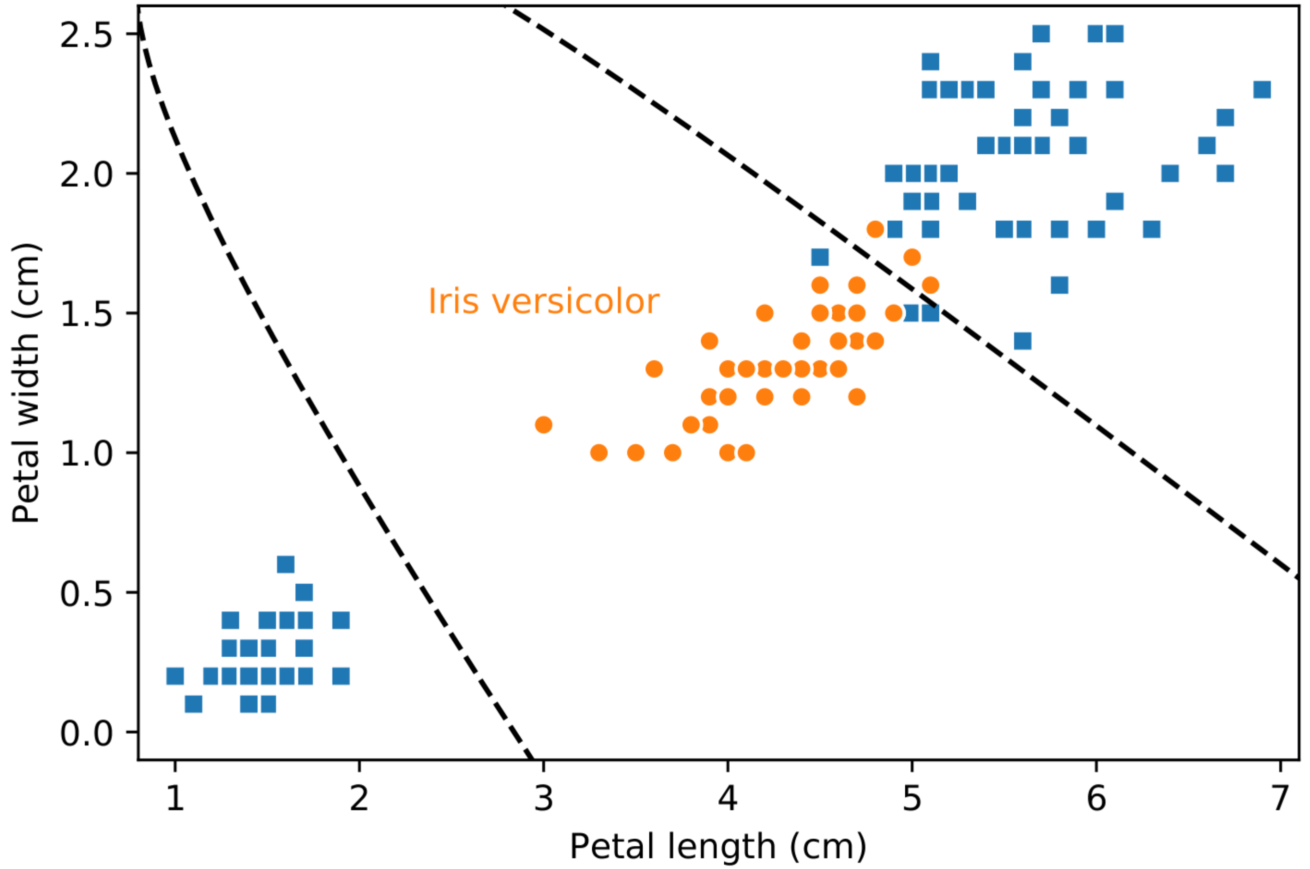
$$\underline{\delta}_{\underline{b}^{[1]}} = \frac{\partial J}{\partial \underline{b}^{[1]}} = \frac{\partial \underline{z}^{[1]}}{\partial \underline{b}^{[1]}} \cdot \frac{\partial J}{\partial \underline{z}^{[1]}} = \frac{\partial \underline{z}^{[1]}}{\partial \underline{b}^{[1]}} \cdot \frac{\partial \underline{a}^{[1]}}{\partial \underline{z}^{[1]}} \cdot \frac{\partial J}{\partial \underline{a}^{[1]}}$$

$$= \frac{\partial \underline{z}^{[1]}}{\partial \underline{b}^{[1]}} \cdot \frac{\partial \underline{a}^{[1]}}{\partial \underline{z}^{[1]}} \cdot \frac{\partial z^{[2]}}{\partial \underline{a}^{[1]}} \cdot \frac{\partial \hat{y}}{\partial z^{[2]}} \cdot \frac{\partial J}{\partial \hat{y}}$$

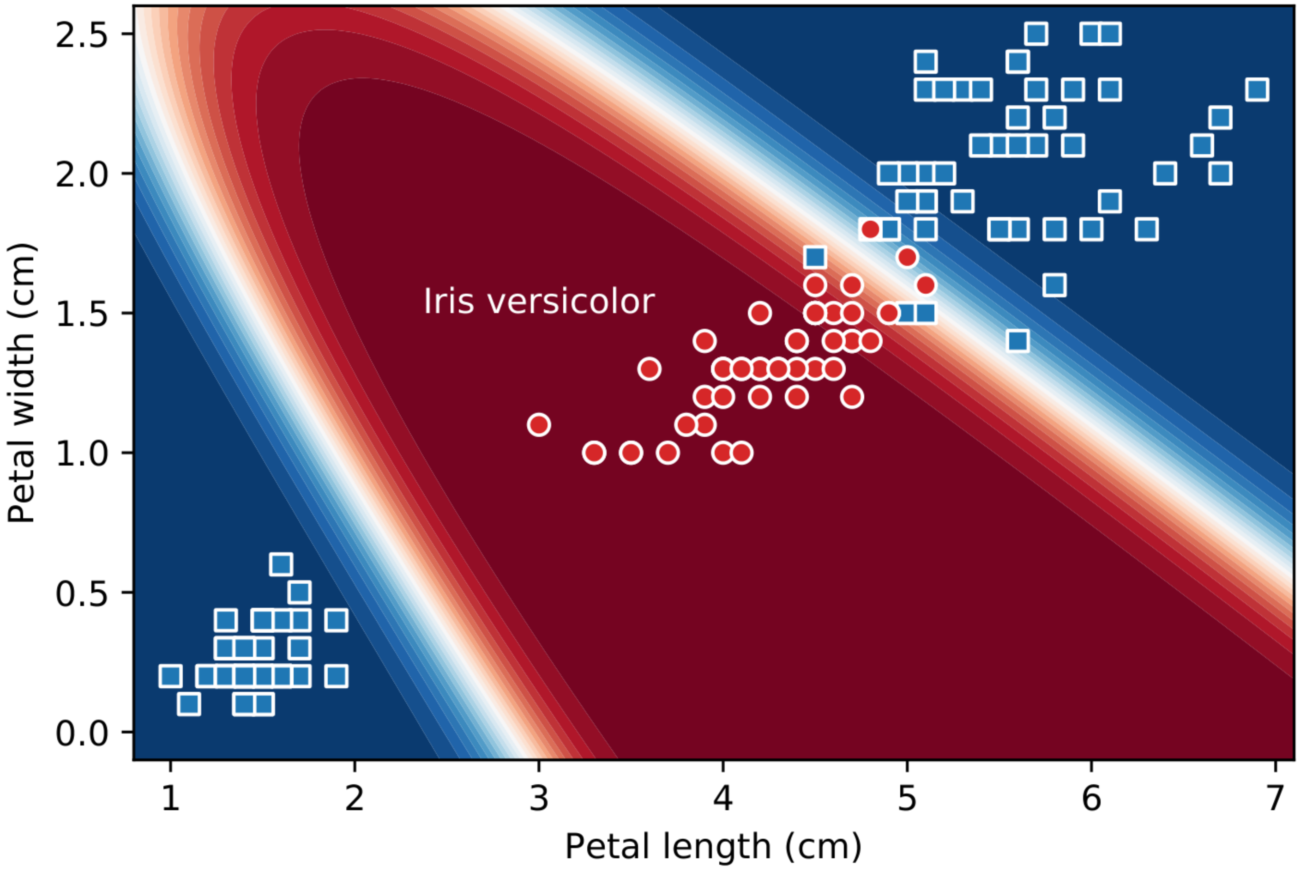
Why the bar?

$$\delta_{\hat{y}} = \frac{\partial J}{\partial \hat{y}} \Bigg|_{\substack{\underline{x} = \underline{x}^{(n)} \\ y = y^{(n)} \\ \underline{\theta} = \hat{\underline{\theta}}^{(m)}}} \qquad \hat{\underline{\theta}}^{(m+1)} = \hat{\underline{\theta}}^{(m)} - \eta \frac{\partial J}{\partial \underline{\theta}} \Bigg|_{\underline{\theta} = \hat{\underline{\theta}}^{(m)}}$$

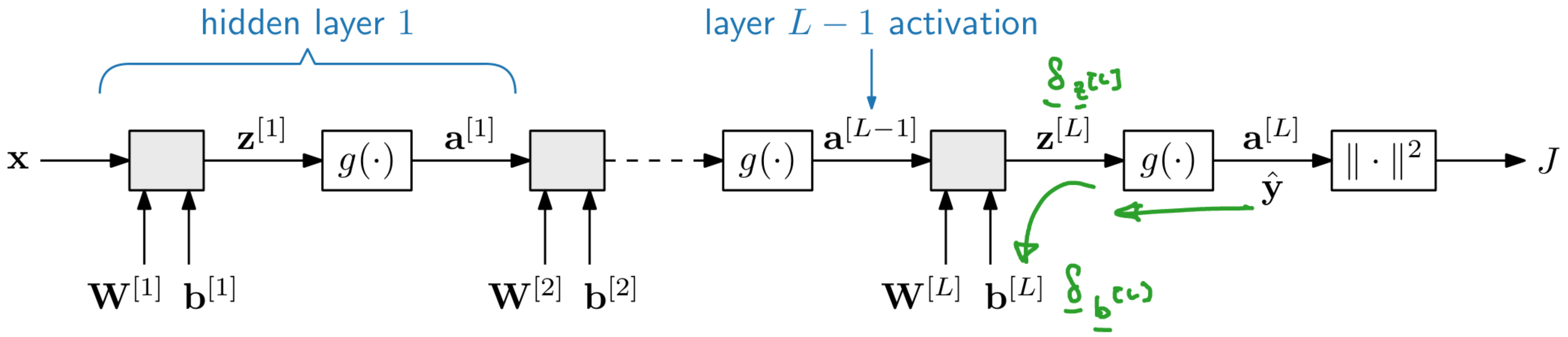
Binary classification of irises using our neural network

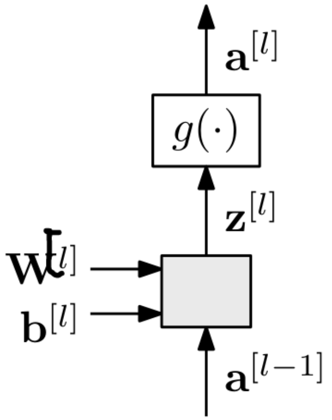


Binary classification of irises using our neural network



Multilayer feedforward neural network





$$
\begin{aligned}
\boldsymbol{\delta}_{\mathbf{z}^{[l]}} &= \boldsymbol{\delta}_{\mathbf{a}^{[l]}} \odot g'(\mathbf{z}^{[l]}) \\
\boldsymbol{\delta}_{\mathbf{b}^{[l]}} &= \boldsymbol{\delta}_{\mathbf{z}^{[l]}} \\
\boldsymbol{\delta}_{\mathbf{W}^{[l]}} &= \boldsymbol{\delta}_{\mathbf{z}^{[l]}}\, \mathbf{a}^{[l-1]\top} \\
\boldsymbol{\delta}_{\mathbf{a}^{[l-1]}} &= \mathbf{W}^{[l]\top} \boldsymbol{\delta}_{\mathbf{z}^{[l]}}
\end{aligned}
$$

These equations are sometimes combined:

$$
\boldsymbol{\delta}_{\mathbf{z}^{[l]}} = \left(\mathbf{W}^{[l+1]\top} \boldsymbol{\delta}_{\mathbf{z}^{[l+1]}}\right) \odot g'(\mathbf{z}^{[l]})
$$

Introduction to neural networks

Computational graphs and automatic differentiation

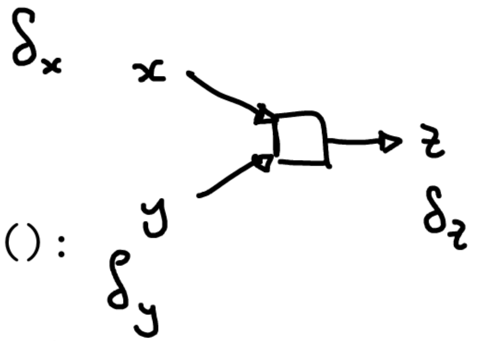
Herman Kamper

http://www.kamperh.com/

Why this (cool) graph formulation?

Adding additional structure is easy:

- As long as we know the derivative of a single operation, the gradient computation is fully specified by the graph.
- Each node just needs to know how to compute its output and how to compute the gradient w.r.t. its inputs. ~~the gradient w.r.t. its output~~.



```
class MultiplyGate():

    def forward(x, y):
        z = x*y
        self.x = x
        self.y = y
        return z

    def backward(delta_z):
        delta_x = self.y * delta_z
                  # dz/dx  * dJ/dz
        delta_y = self.x * delta_z
                  # dz/dy  * dJ/dz
        return [delta_x, delta_y]
```

$$\delta_x = \frac{\partial J}{\partial x}$$

Why then study backprop if the software can do it?

- In some very simple cases, you might not want to have to rely on (the bulky) PyTorch or Tensorflow.
  E.g. the gradients for word2vec is relatively straightforward.
- Sometimes you might want to introduce a new computational operation and then you might need to implement the gradient computation.
- More often: You are hacking parts of the gradient computation for an existing block and need to modify it.

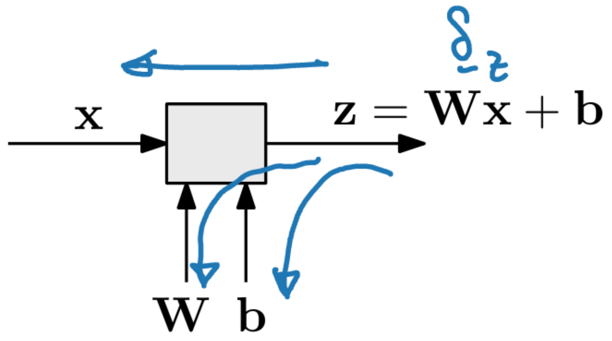
Introduction to neural networks

Backpropagation (now general)

Herman Kamper

http://www.kamperh.com/

A summary of derivatives for common blocks

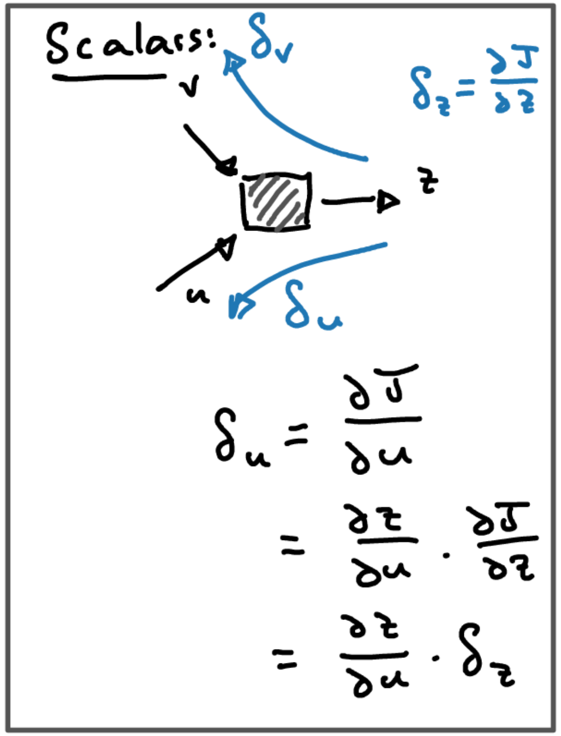


$$\delta_{\mathbf{b}} = \frac{\partial \mathbf{z}}{\partial \mathbf{b}} \delta_{\mathbf{z}} = \mathbf{I}\,\delta_{\mathbf{z}} = \delta_{\mathbf{z}}$$

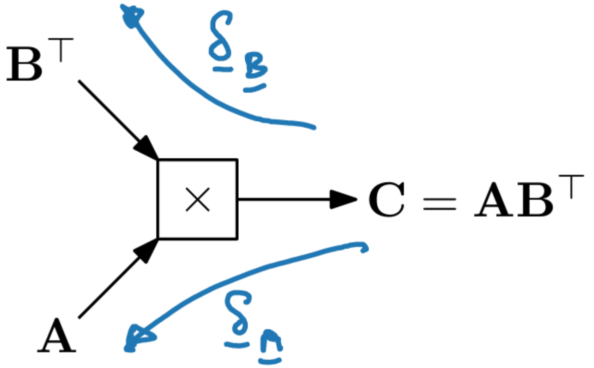
$$\delta_{\mathbf{W}} = \delta_{\mathbf{z}}\,\mathbf{x}^\top$$

$$\delta_{\mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \delta_{\mathbf{z}} = \mathbf{W}^\top \delta_{\mathbf{z}}$$

$$\underline{\delta}_{\underline{x}} = \mathrm{prod}\left(\frac{\partial \underline{z}}{\partial \underline{x}}, \underline{\delta}_{\underline{z}}\right)$$



$$\delta_u = \frac{\partial J}{\partial u} = \frac{\partial z}{\partial u} \cdot \frac{\partial J}{\partial z} = \frac{\partial z}{\partial u} \cdot \delta_z$$



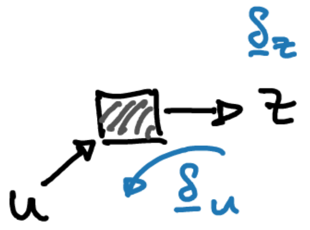
$\underline{\delta}_{\underline{B}}$
$\mathbf{B}^\top$
$\times$
$\mathbf{C} = \mathbf{A}\mathbf{B}^\top$
$\mathbf{A}$
$\underline{\delta}_{\underline{A}}$

$$\delta_{\mathbf{B}} = \delta_{\mathbf{C}}^\top \mathbf{A}$$

$$\underline{\delta}_{\underline{B}} = \mathrm{prod}\left(\frac{\partial \underline{C}}{\partial \underline{B}}, \underline{\delta}_{\underline{C}}\right)$$

$$\delta_{\mathbf{A}} = \delta_{\mathbf{C}}\mathbf{B}$$

A general notation



In all of the above, for arbitrary variable U going in to operation with output Z, the error signal $\boldsymbol{\delta}_{\mathsf{U}}$ is obtained as some kind of product between $\frac{\partial \mathsf{Z}}{\partial \mathsf{U}}$ and $\frac{\partial J}{\partial \mathsf{Z}} = \boldsymbol{\delta}_{\mathsf{Z}}$.

$$\delta_u = \frac{\partial J}{\partial u} = \frac{\partial z}{\partial u} \cdot \frac{\partial J}{\partial z}$$

- With vectors as inputs and outputs: $\boldsymbol{\delta}_{\mathbf{b}} = \frac{\partial \mathbf{z}}{\partial \mathbf{b}} \boldsymbol{\delta}_{\mathbf{z}}$
- But sometimes the order between $\frac{\partial \mathsf{Z}}{\partial \mathsf{U}}$ and $\boldsymbol{\delta}_{\mathsf{Z}}$ flips: $\boldsymbol{\delta}_{\mathbf{W}} = \boldsymbol{\delta}_{\mathbf{z}} \mathbf{x}^\top$
- Or we have to take the transpose: $\boldsymbol{\delta}_{\mathbf{B}} = \boldsymbol{\delta}_{\mathbf{C}}^\top \mathbf{A}$

Let's capture all of these with the new prod operator:

$$\begin{aligned}\boldsymbol{\delta}_{\mathsf{U}} = \frac{\partial J}{\partial \mathsf{U}} &= \operatorname{prod}\left(\frac{\partial \mathsf{Z}}{\partial \mathsf{U}}, \frac{\partial J}{\partial \mathsf{Z}}\right) \\ &= \operatorname{prod}\left(\frac{\partial \mathsf{Z}}{\partial \mathsf{U}}, \boldsymbol{\delta}_{\mathsf{Z}}\right)\end{aligned}$$

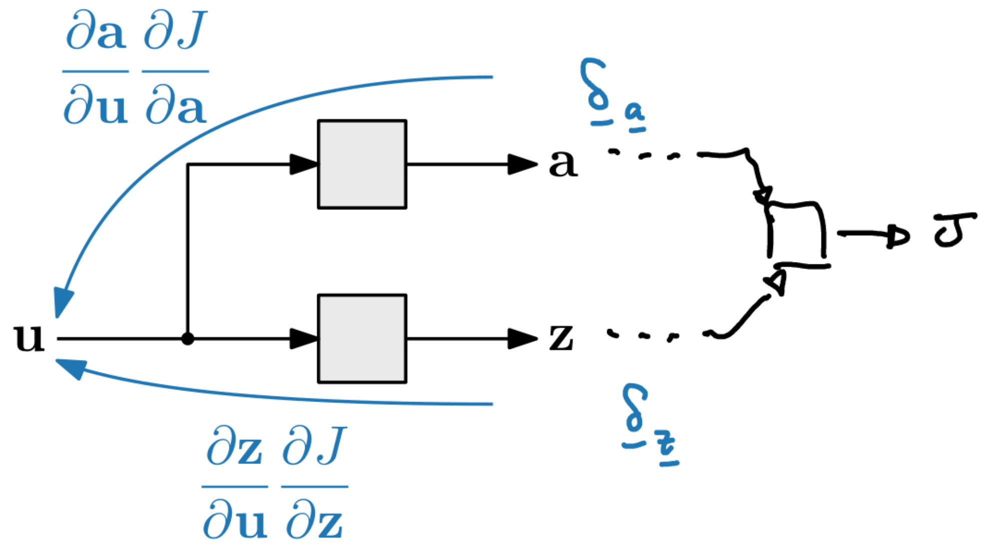
d2l.ai

About forks

$$\underline{\delta}_{\underline{u}} = \frac{\partial J}{\partial \underline{u}} = \frac{\partial \underline{z}}{\partial \underline{u}} \cdot \frac{\partial J}{\partial \underline{z}} + \frac{\partial \underline{a}}{\partial \underline{u}} \cdot \frac{\partial J}{\partial \underline{a}}$$

$$= \frac{\partial \underline{z}}{\partial \underline{u}} \cdot \underline{\delta}_{\underline{z}} + \frac{\partial \underline{a}}{\partial \underline{u}} \cdot \underline{\delta}_{\underline{a}}$$

"Error signal"
"Accumulator"



$\frac{\partial \mathbf{a}}{\partial \mathbf{u}} \frac{\partial J}{\partial \mathbf{a}}$
$\underline{\delta}_{\underline{a}}$
a
u
z
J
$\underline{\delta}_{\underline{z}}$
$\frac{\partial \mathbf{z}}{\partial \mathbf{u}} \frac{\partial J}{\partial \mathbf{z}}$

In general:

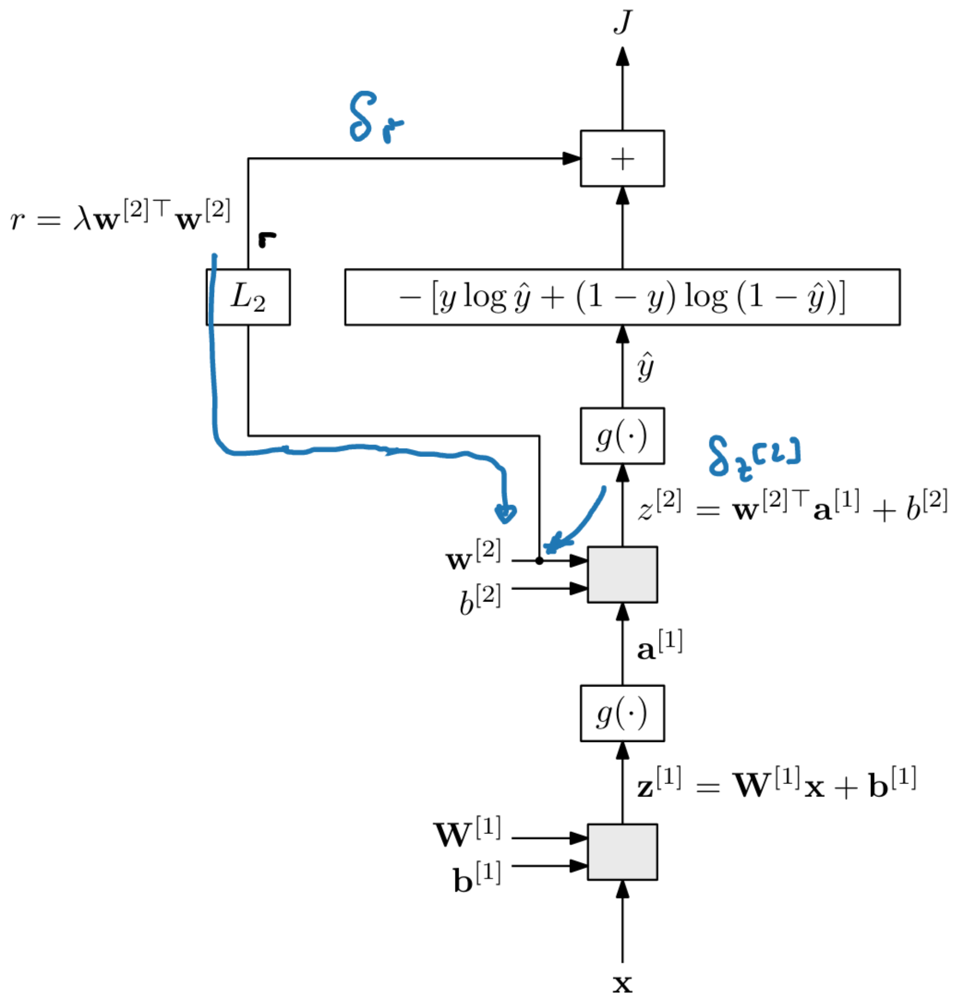
$$\underline{\delta}_{u} = \text{prod}\left(\frac{\partial z}{\partial u}, \underline{\delta}_{z}\right) + \text{prod}\left(\frac{\partial A}{\partial u}, \underline{\delta}_{A}\right)$$

Example: L_2 regularisation

$$J(\underline{\theta}) = -\left[y^{(n)} \log \hat{y}^{(n)} + (1-y^{(n)}) \cdot \log(1-\hat{y}^{(n)})\right] + \lambda \underline{w}^{[2]\top} \underline{w}^{[2]}$$

$$\delta_{\underline{w}^{[2]}} = \frac{\partial J}{\partial \underline{w}^{[2]}} = \frac{\partial z^{[2]}}{\partial \underline{w}^{[2]}} \cdot \delta_{z^{[2]}} + \frac{\partial r}{\partial \underline{w}^{[2]}} \cdot \delta_r$$

$$= \underline{a}^{[1]} \delta_{z^{[2]}} + 2\lambda \underline{w}^{[2]} \delta_r$$


J
$+$
δ_r
$r = \lambda \mathbf{w}^{[2]\top}\mathbf{w}^{[2]}$
r
L_2
$-[y \log \hat{y} + (1-y)\log(1-\hat{y})]$
$\hat{y}$
$g(\cdot)$
$\delta_{z^{[2]}}$
$z^{[2]} = \mathbf{w}^{[2]\top}\mathbf{a}^{[1]} + b^{[2]}$
$\mathbf{w}^{[2]}$
$b^{[2]}$
$\mathbf{a}^{[1]}$
$g(\cdot)$
$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$
$\mathbf{W}^{[1]}$
$\mathbf{b}^{[1]}$
$\mathbf{x}$

The backpropagation algorithm (now with forks)

- **Initialisation:**
  Set accumulators to zero for all input variables: $\boldsymbol{\delta}_{\mathsf{U}} \leftarrow \mathbf{0}$

- **Forward pass:** Start at the inputs and calculate the output of each operation in the graph. Store these values.

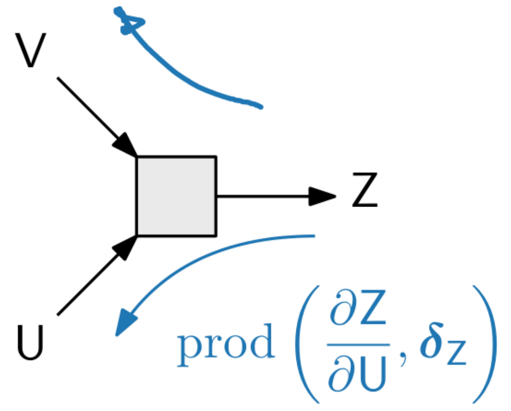
- **Backward pass:** Start at the output of the graph and move backwards. For each operation:

  (a) Determine and calculate the derivative of the output variable w.r.t. each of the input variables to the operation.

  (b) For each input variable U, add $\mathrm{prod}\left(\frac{\partial \mathsf{Z}}{\partial \mathsf{U}}, \frac{\partial J}{\partial \mathsf{Z}}\right)$ to its accumulator, i.e.

$$\boldsymbol{\delta}_{\mathsf{U}} \leftarrow \boldsymbol{\delta}_{\mathsf{U}} + \mathrm{prod}\left(\frac{\partial \mathsf{Z}}{\partial \mathsf{U}}, \boldsymbol{\delta}_{\mathsf{Z}}\right)$$

$\frac{\partial Z}{\partial V}$ and $\frac{\partial Z}{\partial U}$


V
Z
U
$\mathrm{prod}\left(\frac{\partial \mathsf{Z}}{\partial \mathsf{U}}, \boldsymbol{\delta}_{\mathsf{Z}}\right)$

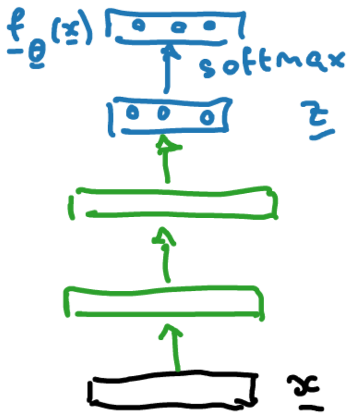
Introduction to neural networks

Relationship between negative log likelihood and cross entropy

Herman Kamper

http://www.kamperh.com/

Negative log likelihood and cross entropy



$$\hat{y} = \underline{f}_{\underline{\theta}}(\underline{x}) = \text{softmax}(\underline{z}) = \frac{1}{\sum_{j=1}^{K} e^{z_j}} \begin{bmatrix} e^{z_1} \\ e^{z_2} \\ \vdots \\ e^{z_K} \end{bmatrix}$$

$$y^{(n)} = \begin{array}{c} \left[\begin{array}{c} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{array}\right] \begin{array}{c} \\ \\ \\ \\ k \\ \\ \\ \\ \end{array} \end{array} \updownarrow K$$

$$\text{NLL:} \quad J(\underline{\theta}) = -\log \prod_{n=1}^{N} P_{\underline{\theta}}(y^{(n)} \mid \underline{x}^{(n)})$$

$$= -\sum_{n=1}^{N} \log P_{\underline{\theta}}(y^{(n)} \mid \underline{x}^{(n)})$$

$$= -\sum_{n=1}^{N} \sum_{k=1}^{K} y_k^{(n)} \log \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

$$= -\sum_{n=1}^{N} \sum_{k=1}^{K} y_k^{(n)} \log \hat{y}_k^{(n)}$$

$$\boxed{\text{Cross-entropy:} \quad H(\underline{p}, \underline{q}) = -\sum_{k=1}^{K} p_k \log q_k}$$

Introduction to neural networks

Neural networks in practice and NLP examples

Herman Kamper

http://www.kamperh.com/

The art of neural networks

- Sometimes useful to scale inputs.
- Instead of vanilla gradient descent, we often use advanced forms of mini-batch gradient descent (Adam is popular at the moment).
- Different initialisation strategies, e.g. `https://arxiv.org/abs/1811.00293`.
- Overfitting: Can combat using standard regularisation, but often rather just use dropout or rely on SGD with early stopping.
- Need to choose number of hidden layers and number of units per layer, and often many more hyperparameters.
- Often make architecture choices (e.g. skip connections) to deal with optimisation problems (e.g. exploding or vanishing gradients).

Named entity recognition

```
last night Paris Hilton wowed in a sequin gown
           PER   PER

Samuel Quinn was arrested in the Hilton Hotel  in Paris in April 1989
PER    PER                       LOC    LOC       LOC      DATE  DATE
```

Tag	Description	Example
PER	People, characters	**Shannon** is a giant of information theory.
ORG	Organisation	The **ICC** is the governing body of cricket.
LOC	Location	**Mt. Sanitas** is in **Sunshine Canyon**.
GPE	Geo-political (countries, states)	Petrol prices are going up in **South Africa**.
DATE	Days, months, years	Micah was born in **April**.

We want to classify the entity of `Paris` in the sentence:

$\texttt{anywhere}\ \underbrace{\texttt{in}\ \underline{\texttt{Paris}}\ \texttt{museums}}\ \texttt{are great}$

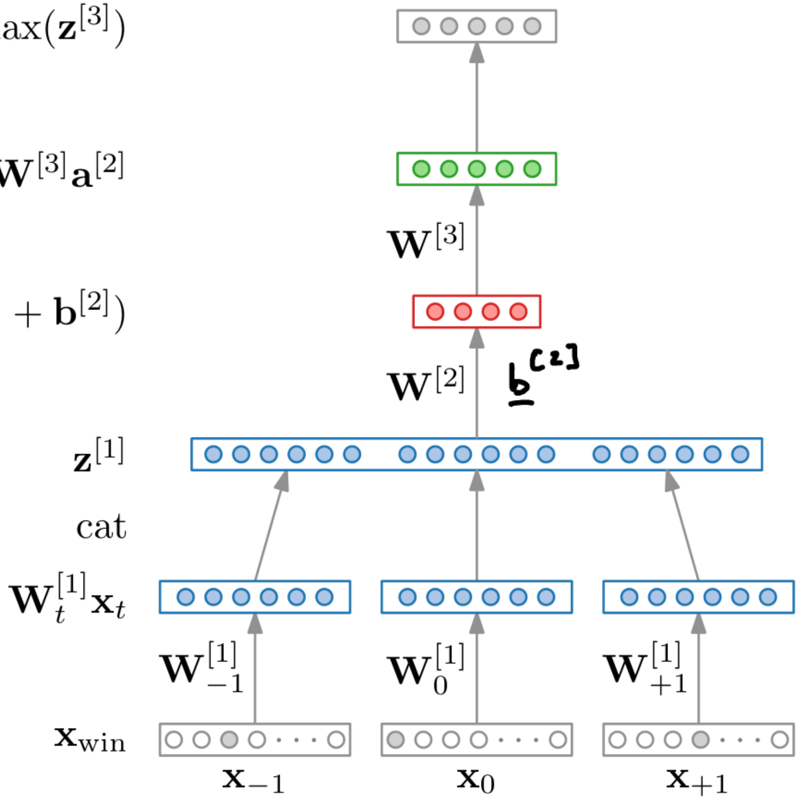
Use a window of words around the centre word:

$$\mathbf{x}_{\text{win}} = \left[-\mathbf{x}_{\texttt{in}}^\top- \quad -\mathbf{x}_{\texttt{Paris}}^\top- \quad -\mathbf{x}_{\texttt{museums}}^\top- \quad \right]^\top$$

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{\text{win}}) = \text{softmax}(\mathbf{z}^{[3]})$$

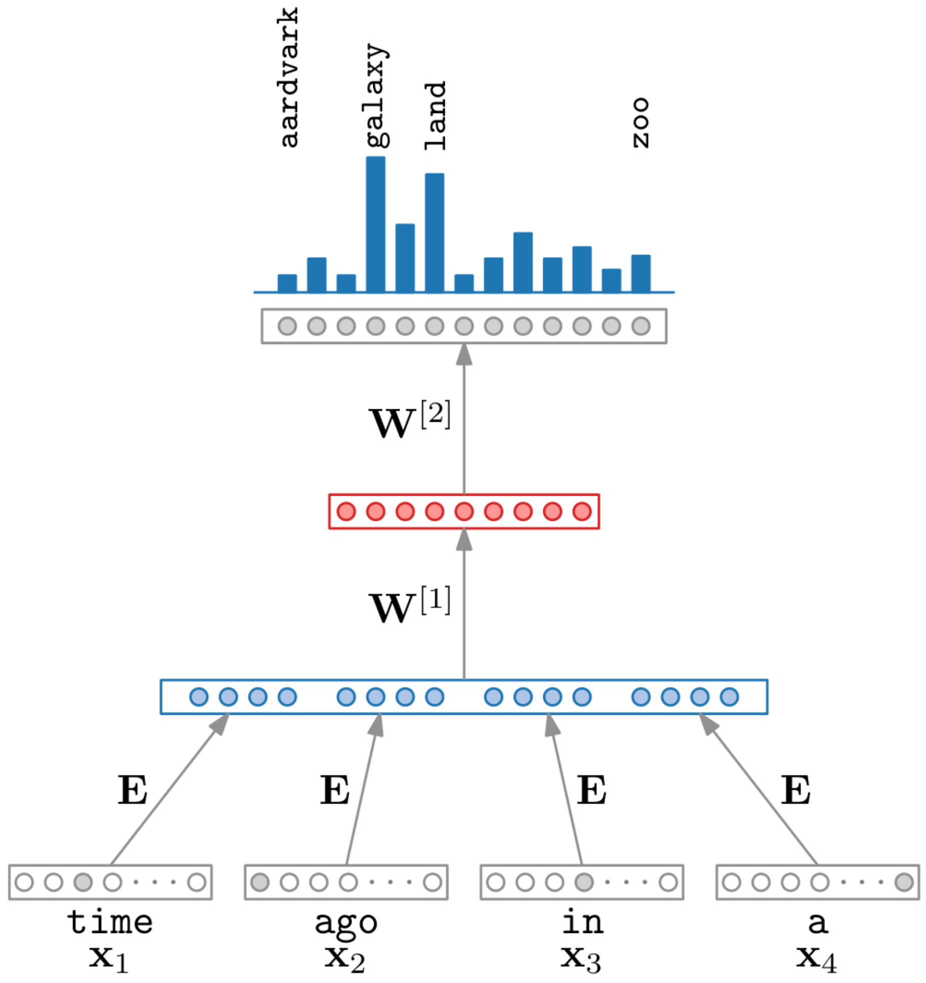
$$\mathbf{z}^{[3]} = \mathbf{W}^{[3]}\mathbf{a}^{[2]}$$

$$\mathbf{a}^{[2]} = g(\mathbf{W}^{[2]}\mathbf{z}^{[1]} + \mathbf{b}^{[2]})$$



Neural language models

$\texttt{A long long } \underbrace{\texttt{time ago in a}} \texttt{ ...}$



Output distribution

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{1:4}) = \hat{\mathbf{y}}$$
$$= \mathrm{softmax}(\mathbf{W}^{[2]}\mathbf{h} + \mathbf{b}^{[2]})$$
$$\in [0, 1]^{|\mathcal{V}|}$$

Hidden layer

$$\mathbf{h} = g(\mathbf{W}^{[1]}\mathbf{e} + \mathbf{b}^{[1]})$$

Concatenated word embeddings

$$\mathbf{e} = [\mathbf{e}_1; \mathbf{e}_2; \mathbf{e}_3; \mathbf{e}_4]$$

One-hot word vectors

Introduction to neural networks

What did I read

Herman Kamper

http://www.kamperh.com/

- H. Kamper, “Yet another introduction to backpropagation,” *Stellenbosch University*, 2022.
  `http://www.kamperh.com/notes/kamper_backprop22.pdf`

- A. Ng, “Multi-layer neural network,” *Stanford University*, 2013.
  `http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/`

- A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*, 2021.
  `https://d2l.ai/`

- M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*, 2020.
  `https://mml-book.github.io/book/mml-book.pdf`

- CS231n: Optimization 2, *Stanford University*.
  `https://cs231n.github.io/optimization-2/`

- C. Manning, “CS224N: Neural net learning, gradients by hand (matrix calculus) and algorithmically (the backpropagation algorithm),” *Stanford University*, 2022.
  `https://web.stanford.edu/class/cs224n/`