University of Minnesota
Driven to Discover℠

EE 5561: Image Processing and Applications

Lecture 17

Mehmet Akçakaya

Recap of Last Lecture

- Machine learning concepts
  - Tasks, performance measures
  - X: input data, Y: label
  - $f_{\boldsymbol{\theta}}$: function we want to estimate/learn, parametrized by $\boldsymbol{\theta}$
  - $\mathcal{L}(f_{\boldsymbol{\theta}}(X), Y)$ loss function
  - Sample mean loss $\arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{n} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$
  - Gradient descent
  - Generalization gap
  - Multi-layer networks
  - Activation functions
  - Neural networks

Minimizing the Loss

- Loss function is $$\arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{n} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$
- Optimization requires calculation of gradients with respect to **θ**
- Or more concretely, for our multi-layer feed-forward network

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \eta\left(\mathbf{W}^{[n]}\left(\ldots\eta\left(\mathbf{W}^{[2]}\eta(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]})\right) + \mathbf{b}^{[2]}\right)\cdots + \mathbf{b}^{[n]}\right)\Bigg)$$

with respect to $\{\mathbf{W}^{[n]}, \ldots, \mathbf{W}^{[1]}, \mathbf{b}^{[n]}, \ldots, \mathbf{b}^{[1]}\}$

- How to calculate these derivatives?
  - Chain rule!

Minimizing the Loss

- Consider $y = f(g(h(x)))$

  and let $u_1 = h(x), \qquad u_2 = g(u_1), \qquad y = f(u_2)$

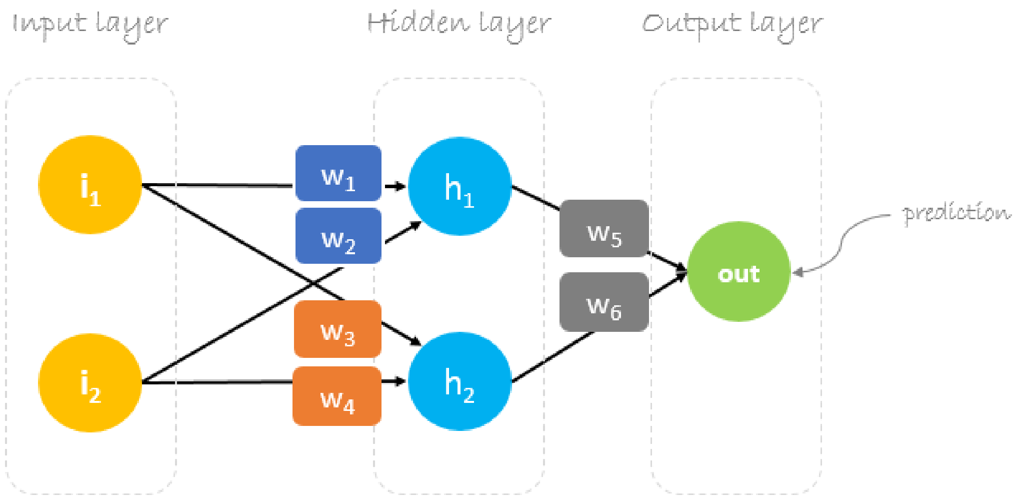
  then $$\left.\frac{\partial f}{\partial x}\right|_{x=c} = \left.\frac{\partial f}{\partial u_2}\right|_{u_2=g(u_1)} \left.\frac{\partial g}{\partial u_1}\right|_{u_1=h(c)} \left.\frac{\partial h}{\partial x}\right|_{x=c}$$

- In deep learning, chain rule is used in a procedure called backpropagation to calculate the gradient with respect to $\{\mathbf{W}^{[k]}, \mathbf{b}^{[k]}\}_k$ at each layer iteratively
- Basic idea

$$\left.\frac{\partial f}{\partial x}\right|_{x=c} = \left.\frac{\partial f}{\partial u_2}\right|_{u_2=g(h(c))} \left.\frac{\partial g}{\partial u_1}\right|_{u_1=h(c)} \left.\frac{\partial h}{\partial x}\right|_{x=c}$$

Backpropagation

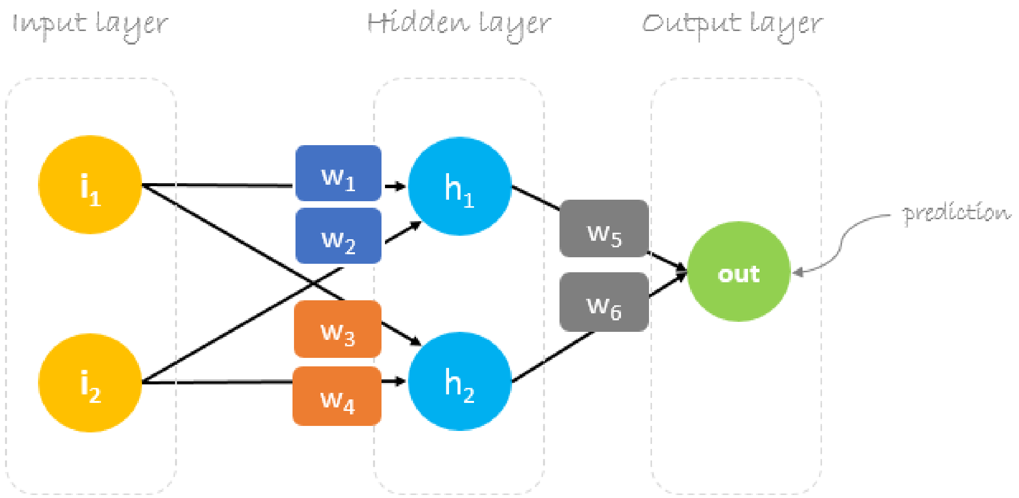
Simple linear multi-layer network



https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Simple linear multi-layer network



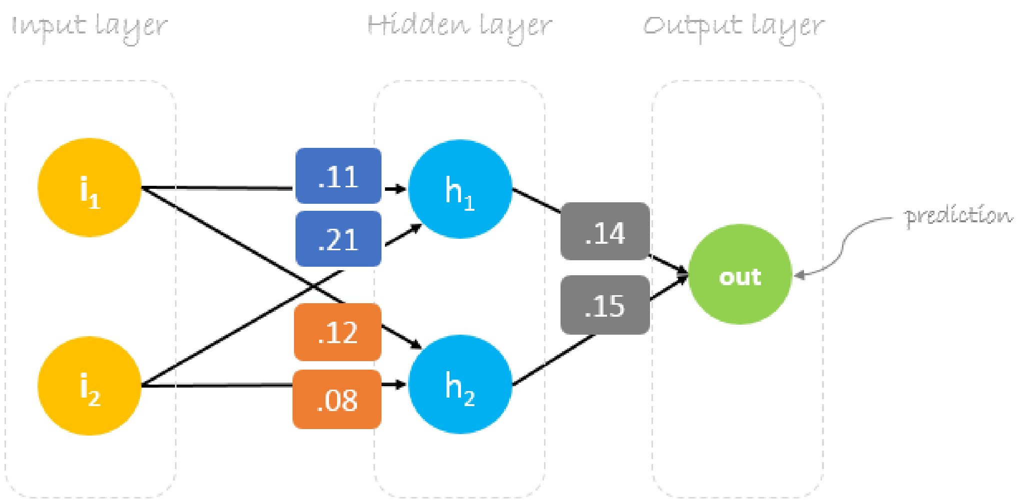
Assume initial weights

w1 = 0.11, w2 = 0.21, w3 = 0.12, w4 = 0.08, w5 = 0.14 and w6 = 0.15

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Simple linear multi-layer network

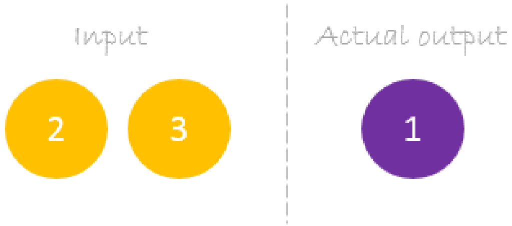


The updates depend on the database & loss function

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Let's use one sample in the dataset



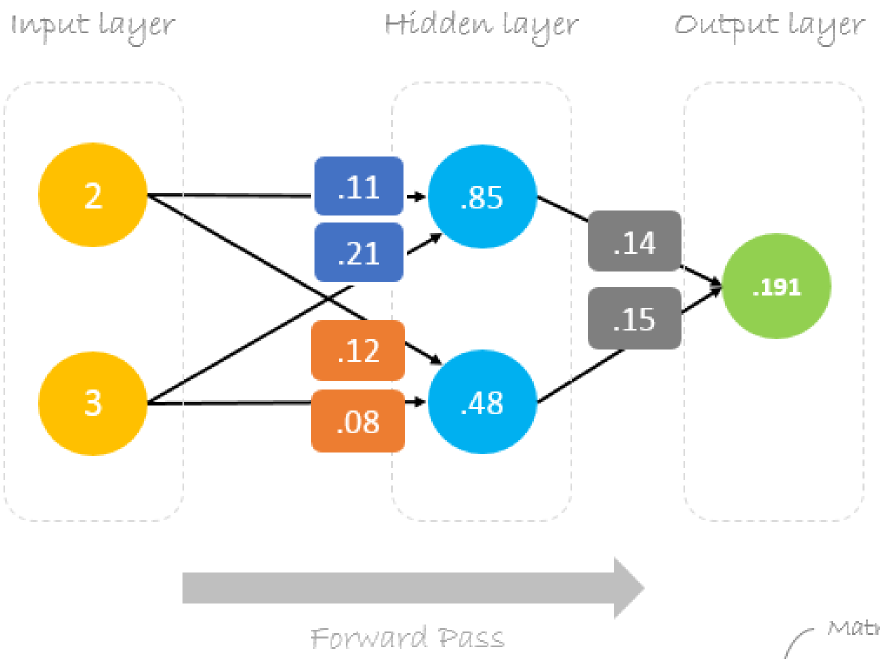
and use MSE loss

$$\text{MSE} = \frac{1}{2}\ (\text{predicted - label})^2 \triangleq (y - y^*)^2$$

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Forward pass



Matrix multiplication

$$[2 \quad 3] \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = [0.85 \quad 0.48] \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} = [0.191]$$

Details

2 x .11 + 3 x .21 = .85

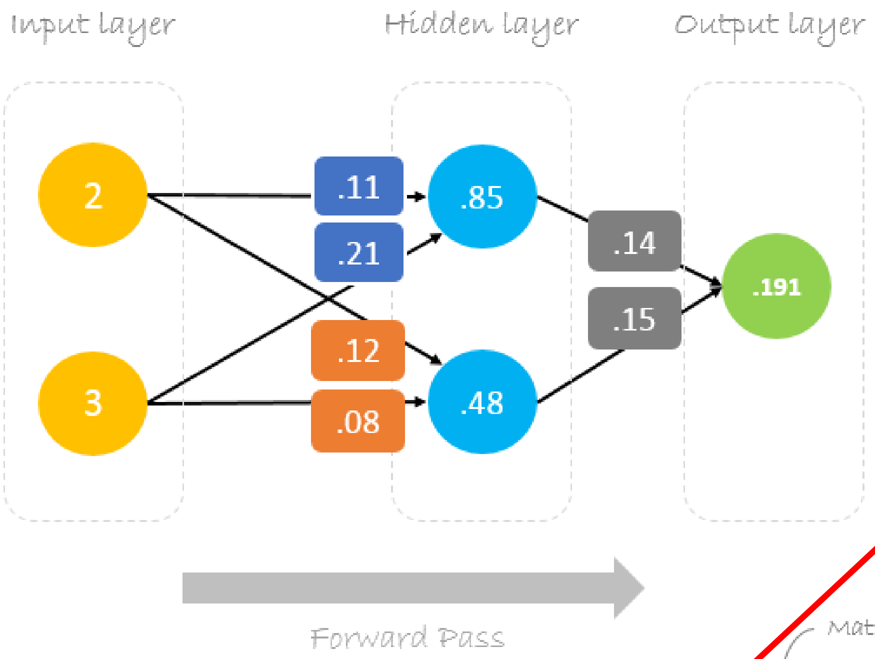
2 x .12 + 3 x .08 = .48

.85 x .14 + .48 x .15 = .191

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Forward pass



Pretty bad!

Matrix multiplication

$$[2 \quad 3] \cdot \begin{bmatrix}0.11 & 0.12\\ 0.21 & 0.08\end{bmatrix} = [0.85 \quad 0.48] \cdot \begin{bmatrix}0.14\\ 0.15\end{bmatrix} = [0.191]$$

Details

2 x .11 + 3 x .21 = .85

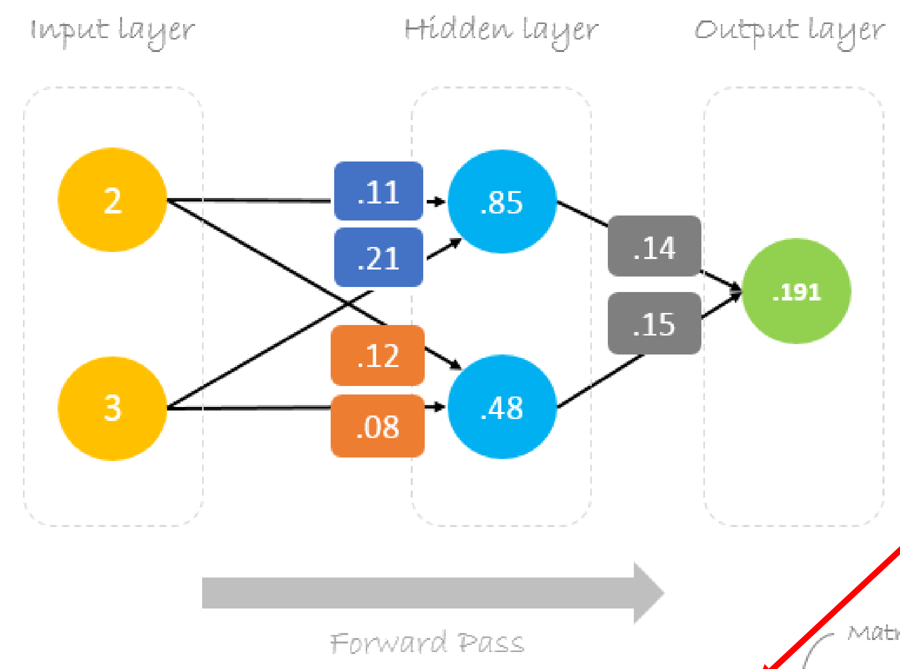
2 x .12 + 3 x .08 = .48

.85 x .14 + .48 x .15 = .191

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Forward pass



$$\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

Pretty bad!

Matrix multiplication

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = \begin{bmatrix} 0.85 & 0.48 \end{bmatrix} \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} = \begin{bmatrix} 0.191 \end{bmatrix}$$

Details

2 x .11 + 3 x .21 = .85

.85 x .14 + .48 x .15 = .191

2 x .12 + 3 x .08 = .48

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Backpropagate for update

Old weight

Derivative of Error with respect to weight

$$ {}^{*}W_x = W_x - a \left(\frac{\partial Error}{\partial W_x} \right) $$

New weight

Learning rate

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Backpropagate for update

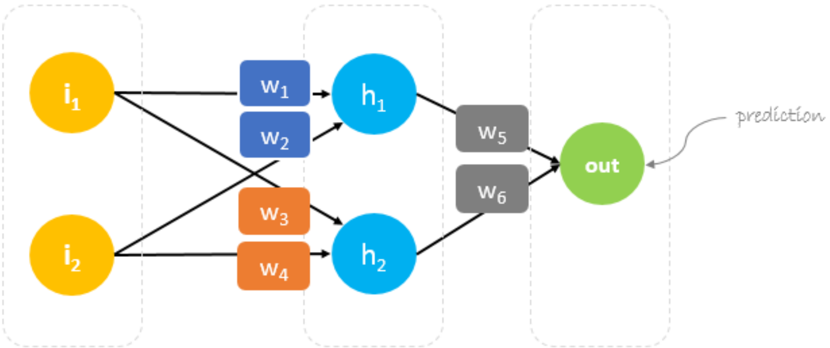
Derivative of Error with respect to weight

Old weight

$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

Learning rate



out

$$\downarrow$$

$$(h_1)\, w_5 + (h_2)\, w_6$$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$$\downarrow$$

$$(i_1\, w_1 + i_2\, w_2)\, w_5 + (i_1\, w_3 + i_2\, w_4)\, w_6$$

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

out

$(h_1)\, w_5 + (h_2)\, w_6$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$(i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$

$$\frac{\partial Error}{\partial y} = (y - y^*)$$

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

out

$$(h_1)\, w_5 + (h_2)\, w_6$$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$$(i_1 \boldsymbol{w_1} + i_2 \boldsymbol{w_2})\, \boldsymbol{w_5} + (i_1 \boldsymbol{w_3} + i_2 \boldsymbol{w_4})\, \boldsymbol{w_6}$$

$$\frac{\partial Error}{\partial y} = (y - y^*)$$

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial y} \frac{\partial y}{\partial W_6} = (y - y^*) h_2$$

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

out

$(h_1)\, w_5 + (h_2)\, w_6$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$(i_1 \boldsymbol{w_1} + i_2 \boldsymbol{w_2})\, \boldsymbol{w_5} + (i_1 \boldsymbol{w_3} + i_2 \boldsymbol{w_4})\, \boldsymbol{w_6}$

$$\frac{\partial Error}{\partial y} = (y - y^*)$$

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial y} \frac{\partial y}{\partial W_6} = (y - y^*) h_2$$

$$\frac{\partial Error}{\partial W_5} = \frac{\partial Error}{\partial y} \frac{\partial y}{\partial W_5} = (y - y^*) h_1$$

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

out

$(h_1)\, w_5 + (h_2)\, w_6$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$(i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$

$$\frac{\partial Error}{\partial y} = (y - y^*)$$

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial y}\frac{\partial y}{\partial W_6} = (y - y^*)h_2$$

$$\frac{\partial Error}{\partial W_5} = \frac{\partial Error}{\partial y}\frac{\partial y}{\partial W_5} = (y - y^*)h_1$$

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial y}\frac{\partial y}{\partial h_1}\frac{\partial h_1}{\partial W_1} = (y - y^*)W_5 i_1$$

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

out

$$(h_1)\, w_5 + (h_2)\, w_6$$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$$(i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

$$\frac{\partial Error}{\partial y} = (y - y^*)$$

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial y}\frac{\partial y}{\partial W_6} = (y - y^*)h_2$$

$$\frac{\partial Error}{\partial W_5} = \frac{\partial Error}{\partial y}\frac{\partial y}{\partial W_5} = (y - y^*)h_1$$

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial y}\frac{\partial y}{\partial h_1}\frac{\partial h_1}{\partial W_1} = (y - y^*)W_5 i_1$$

etc

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

out

$$(h_1)\, w_5 + (h_2)\, w_6$$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$$(i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

For learning rate a, $\Delta = (y - y^*)$

we have the following updates

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

out

$$(h_1)\, w_5 + (h_2)\, w_6$$

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

$$(i_1\, w_1 + i_2\, w_2)\, w_5 + (i_1\, w_3 + i_2\, w_4)\, w_6$$

For learning rate a, $\Delta = (y - y^*)$

we have the following updates

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} ah_1\Delta \\ ah_2\Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Let a = 0.05

we know $\Delta = (y - y^*) = 0.191 - 1 = -0.809$

Thus

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix} . \begin{bmatrix} 0.14 & 0.15 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Let a = 0.05

we know $\Delta = (y - y^*) = 0.191 - 1 = -0.809$

Thus

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(\text{-}0.809)\begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} \text{-}0.034 \\ \text{-}0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix} . \begin{bmatrix} 0.14 & 0.15 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} \text{-}0.011 & \text{-}0.012 \\ \text{-}0.017 & \text{-}0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Sanity check: With the new weights do a forward pass

The new prediction becomes y = 0.26 → better!

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

Let a = 0.05

we know $\Delta = (y - y^*) = 0.191 - 1 = -0.809$

Thus

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(\text{-}0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} \text{-}0.034 \\ \text{-}0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} . \begin{bmatrix} 0.14 & 0.15 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} \text{-}0.011 & \text{-}0.012 \\ \text{-}0.017 & \text{-}0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Sanity check: With the new weights do a forward pass

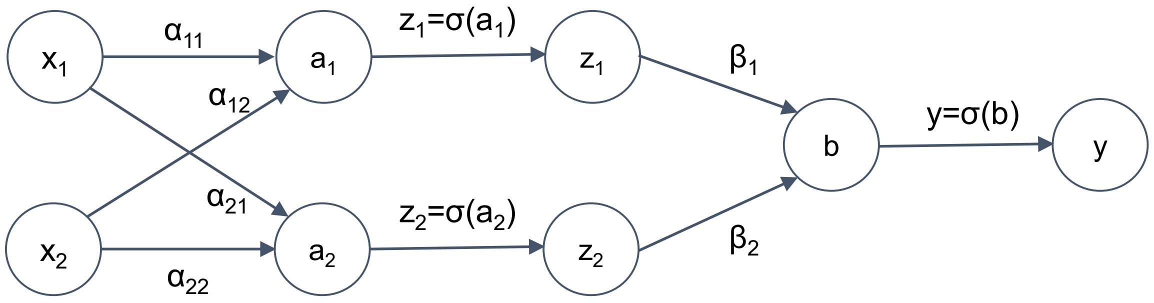
The new prediction becomes y = 0.26 → better!

REPEAT!

https://hmkcode.com/ai/backpropagation-step-by-step/

Backpropagation

- More realistic example

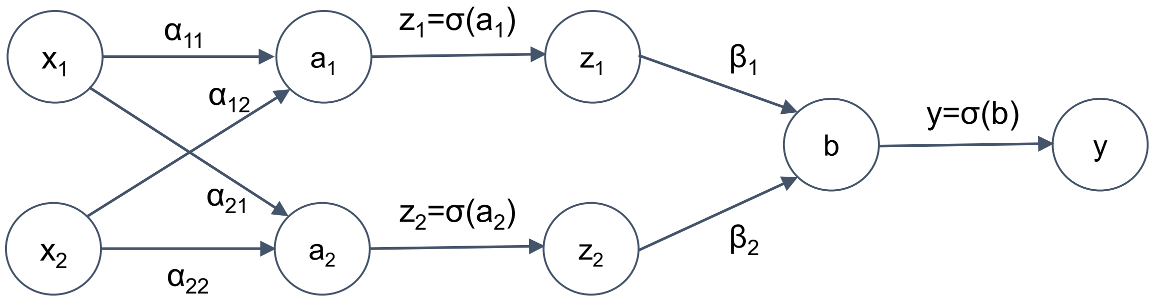


  - Let the activation σ be sigmoid $\sigma(z) = \dfrac{1}{1+e^{-z}}$

  - Need to learn α_{11}, α_{12}, α_{21}, α_{22}, β_1, β_2

Backpropagation

- More realistic example

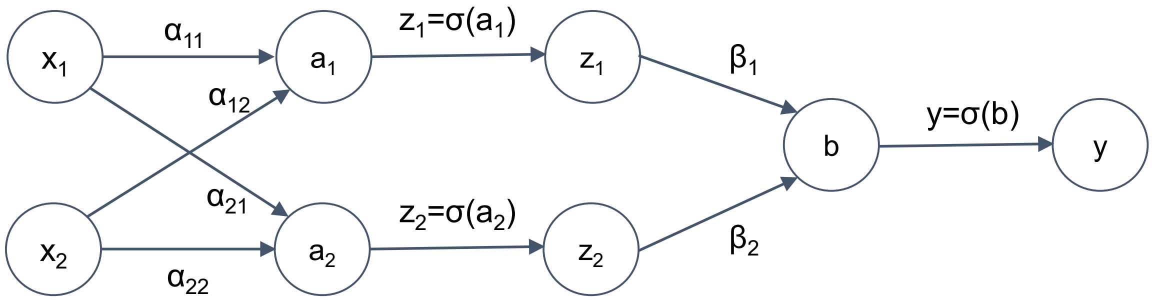


$$\sigma(z) = \frac{1}{1+e^{-z}}$$

  - We will consider logistic regression
    - Models probability for a binary classification problem
    - Outputs are 0 and 1

Backpropagation

- More realistic example



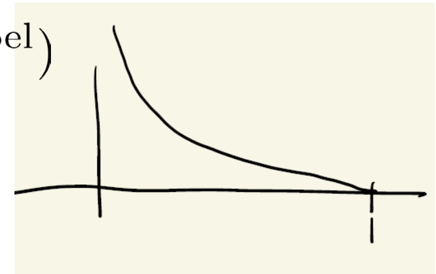
$$\sigma(z) = \frac{1}{1+e^{-z}}$$

  - We will consider logistic regression

$$\mathcal{L}(y, y^{\text{label}}) = \begin{cases} -\log(y) & \text{if } y^{\text{label}} = 1 \\ -\log(1-y) & \text{if } y^{\text{label}} = 0 \end{cases}$$

$y^{\text{label}} = 1$

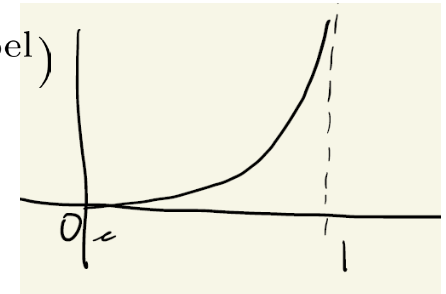
$\mathcal{L}(y, y^{\text{label}})$



y

$y^{\text{label}} = 0$

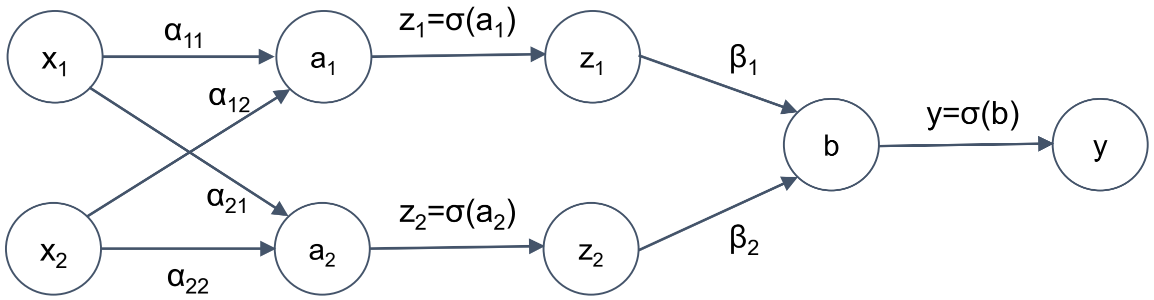
$\mathcal{L}(y, y^{\text{label}})$



y

Backpropagation

- More realistic example



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

  - We will consider logistic regression

$$\mathcal{L}(y, y^{\text{label}}) = \begin{cases} -\log(y) & \text{if } y^{\text{label}} = 1 \\ -\log(1-y) & \text{if } y^{\text{label}} = 0 \end{cases}$$

or

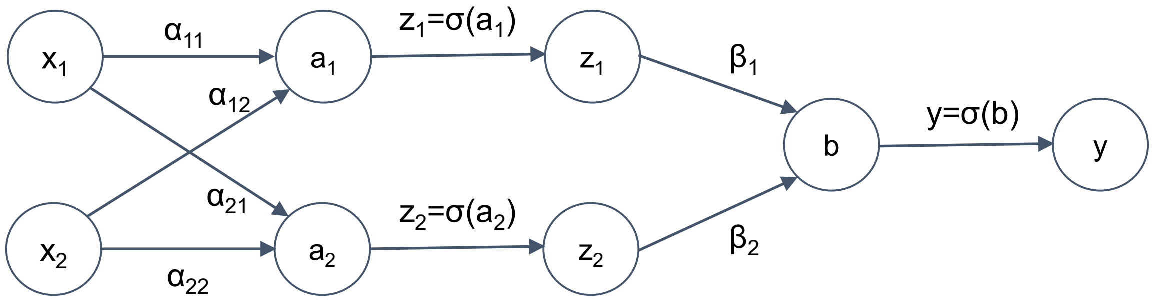
$$\mathcal{L}(y, y^{\text{label}}) = -y^{\text{label}} \log(y) - (1 - y^{\text{label}}) \log(1-y)$$

  - Again, we need to backpropagate

Backpropagation

- More realistic example

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



$$\mathcal{L}(y, y^{\text{label}}) = -y^{\text{label}} \log(y) - (1 - y^{\text{label}}) \log(1-y)$$

$$\frac{\partial \mathcal{L}}{\partial y} = -\frac{y^{\text{label}}}{y} + \frac{1-y^{\text{label}}}{1-y}$$

Forward pass

$$y = \frac{1}{1+e^{-b}}$$

$$b = \beta_1 z_1 + \beta_2 z_2$$

Backward pass

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial b} \qquad \frac{\partial y}{\partial b} = \frac{e^{-b}}{(1+e^{-b})^2} = y(1-y)$$

w.r.t. layer parameters

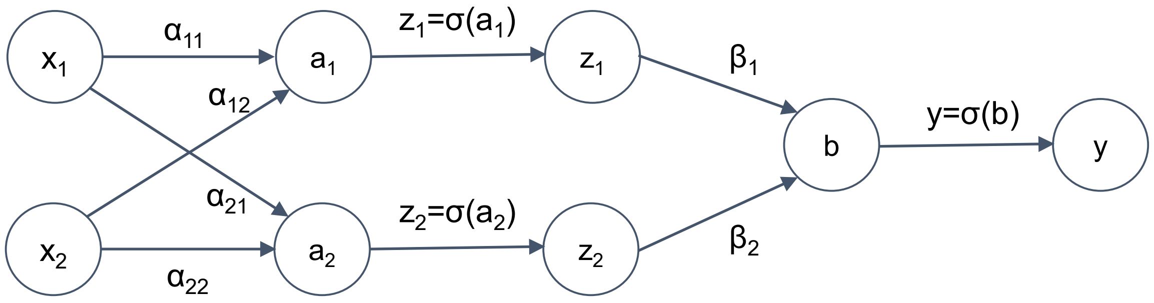
$$\frac{\partial \mathcal{L}}{\partial \beta_i} = \frac{\partial \mathcal{L}}{\partial b}\frac{\partial b}{\partial \beta_i} \qquad \frac{\partial b}{\partial \beta_i} = z_i$$

w.r.t. input to backpropagate

$$\frac{\partial \mathcal{L}}{\partial z_i} = \frac{\partial \mathcal{L}}{\partial b}\frac{\partial b}{\partial z_i} \qquad \frac{\partial b}{\partial z_i} = \beta_i$$

Backpropagation

- More realistic example



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

$$\mathcal{L}(y, y^{\text{label}}) = -y^{\text{label}} \log(y) - (1 - y^{\text{label}}) \log(1-y)$$

$$\frac{\partial \mathcal{L}}{\partial y} = -\frac{y^{\text{label}}}{y} + \frac{1-y^{\text{label}}}{1-y}$$

Forward pass

$$z_j = \frac{1}{1+e^{-a_j}}$$

$$a_j = \alpha_{j1}x_1 + \alpha_{j2}x_2$$

Backward pass

$$\frac{\partial \mathcal{L}}{\partial a_j} = \frac{\partial \mathcal{L}}{\partial z_j}\frac{\partial z_j}{\partial a_j} \qquad \frac{\partial z_j}{\partial a_j} = z_j(1-z_j)$$

w.r.t. layer parameters

$$\frac{\partial \mathcal{L}}{\partial \alpha_{jk}} = \frac{\partial \mathcal{L}}{\partial a_j}\frac{\partial a_j}{\partial \alpha_{jk}} \qquad \frac{\partial a_j}{\partial \alpha_{jk}} = x_k$$

w.r.t. input to backpropagate

$$\frac{\partial \mathcal{L}}{\partial x_k} = \frac{\partial \mathcal{L}}{\partial a_1}\frac{\partial a_1}{\partial x_k} + \frac{\partial \mathcal{L}}{\partial a_2}\frac{\partial a_2}{\partial x_k}$$

etc

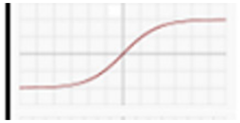
In Practice

- Most operators will be vector valued
  - Derivatives replaced with Jacobian matrices
- The software will do the differentiation for you
- Other derivatives

Rectified Linear Unit (ReLU)

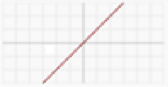
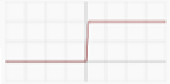
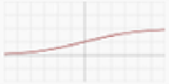
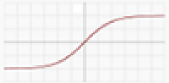
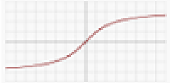
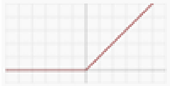
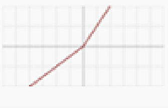
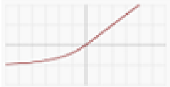
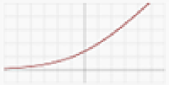
$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

TanH



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

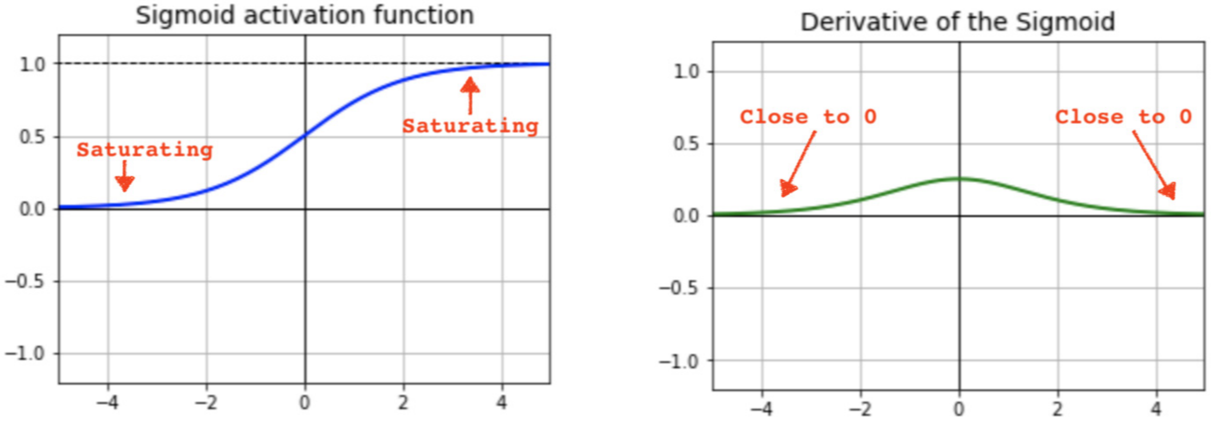
Derivatives of Activation Functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

http://srutisj.in/2017-11-19-activation-function/

In Practice

− What do these mean for multi-layer networks & backpropagation?



- If backpropagating over many layers, these will get multiplied
  - Small derivatives (exponentially decreasing with distance from last layer) → Numerical issues
  - Called “vanishing gradients”
  - Same problem for tanh
- For ReLU → Derivative is either 0 or 1
  - Solves this issue → Thus it’s very popular

In Practice

$$\arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{n} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$

- We can compute the gradient over
  - Entire training database

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \sum_{k=1}^{n} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$

  - Single data point

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$

  - A smaller subset (mini-batch)

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \sum_{k=k_0+1}^{k_0+n_0} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k) \quad \longleftarrow$$

This is what is usually done

Stochastic Gradient Descent (SGD)

- Stochastic gradient descent (SGD)
  - Randomly picks a mini-batch and uses this for gradient calculations and updates

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \sum_{k \in \mathcal{K}_j} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$

$\mathcal{K}_j$ randomly chosen index set of size n_0

  - Note the random index set itself is indexed over *j*
    - *j* has to vary between 1 to round(n/n_0) to cover the entire training dataset
    - Note mini-batches are chosen in a way to cover the entire dataset (sampling without replacement)
  - Two different terms:
    - Iteration: Each gradient update per mini-batch

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \sum_{k \in \mathcal{K}_j} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$

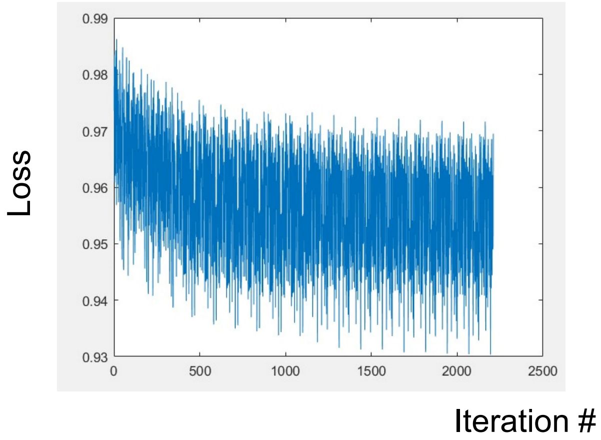
    - Epoch: When the entire dataset has been passed forward & backward once
      - n/n_0 iterations needed in an epoch

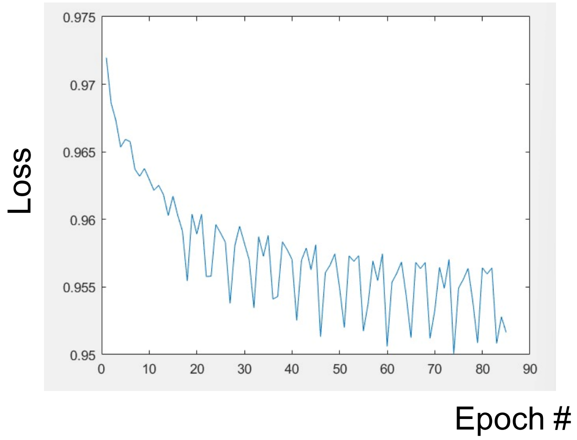
Stochastic Gradient Descent (SGD)

- Stochastic gradient descent (SGD)
  - Randomly picks a mini-batch and uses this for gradient calculations and updates

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \sum_{k \in \mathcal{K}_j} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k) \qquad \mathcal{K}_j \text{ randomly chosen index set of size } n_0$$

  - Note the random index set itself is indexed over *j*
    - *j* has to vary between 1 to round(n/n_0) to cover the entire training dataset
    - Note mini-batches are chosen in a way to cover the entire dataset (sampling without replacement)
  - Iteration vs. epoch





SGD

- Stochastic gradient descent (SGD)
  - In practice with images batch sizes of 32-64 (or higher now) are common
    - Depends on application!
  - η is called *learning rate* in deep learning applications (we used to call it step size)
  - SGD is much faster & more memory-efficient than GD
    - Can use optimized matrix operations in deep learning libraries
  - Reduces the variance of the parameter updates due to random selection
    - Can escape suboptimal local minima easier than GD
    - Converges faster than GD

More on SGD

- Stochastic gradient descent (SGD)
  - Many many variants exist…
  - One important concept is momentum
    - To avoid getting trapped in bad local minima
    - If the optimization surface is steeper than the other SGD will oscillate
    - Momentum dampens oscillations

Image 2: SGD without momentum

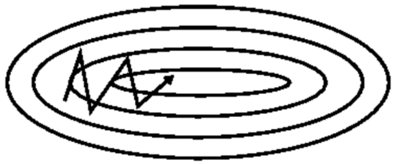


Image 3: SGD with momentum

    - Does this by adding a fraction of the update vector of the past time step to the current update vector

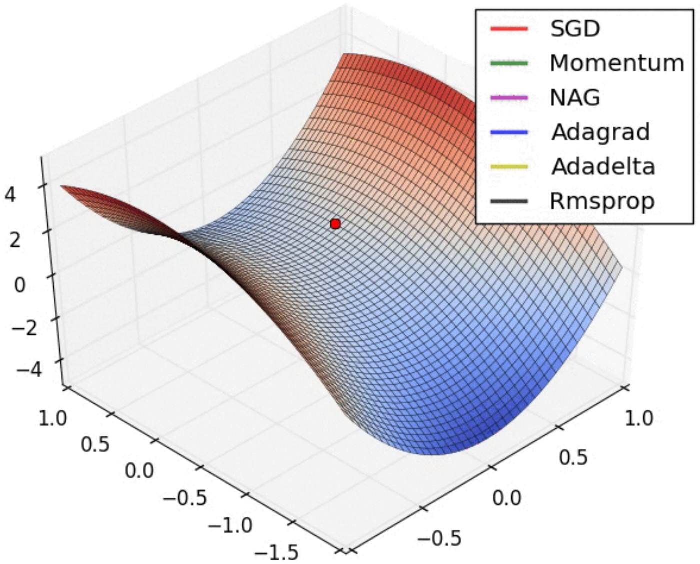
$$\mathbf{v}^{(t)} = \gamma \mathbf{v}^{(t-1)} + \eta \sum_{k \in \mathcal{K}_j} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k)$$

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)} - \mathbf{v}^{(t)}$$

https://ruder.io/optimizing-gradient-descent/

More on SGD

- Stochastic gradient descent (SGD)
  - Many many variants exist…
  - Other important concept: learning rate “schedules”
    - Hard to choose a good learning rate (very important hyperparameter in practice!)
    - High learning rate is good in early epochs, and low learning rate is better at later epochs
    - Also may need different learning rates for different parameters
    - AdaGrad, RMSProp, Adam …



This is a gif

http://ruder.io/optimizing-gradient-descent/

In Practice

- How do we initialize the weights in $\boldsymbol{\theta}$?
  - Mostly randomly with zero-mean
  - Either constant variance across layers
  - Or normalized with respect to number of channels/layers (Xavier initialization)
    - Most networks now have fixed number of channels/layer → approximately leads to the first setting

- Should we regularize the weights?

$$\arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{n} \mathcal{L}(f_{\boldsymbol{\theta}}(x_k), y_k) + \mathcal{R}(\boldsymbol{\theta})$$

  - We can replicate what we have done before (e.g. l_p norm regularizers)
  - We will see there are other methods for regularization
    - e.g. implicitly in the SGD via the random selection of indices

Recap

- Backpropagation
  - Concept & examples for clarification
  - In practice: All done automatically in the learning framework
- Practical training points
  - Optimization algorithms
  - Hyperparameters/initialization
  - Regularization
- Course announcement
  - Office hours shortened (11:00-11:45) via zoom tomorrow