COMP9444: Neural Networks and Deep Learning

Week 1d. Backpropagation

Raymond Louie

School of Computer Science and Engineering

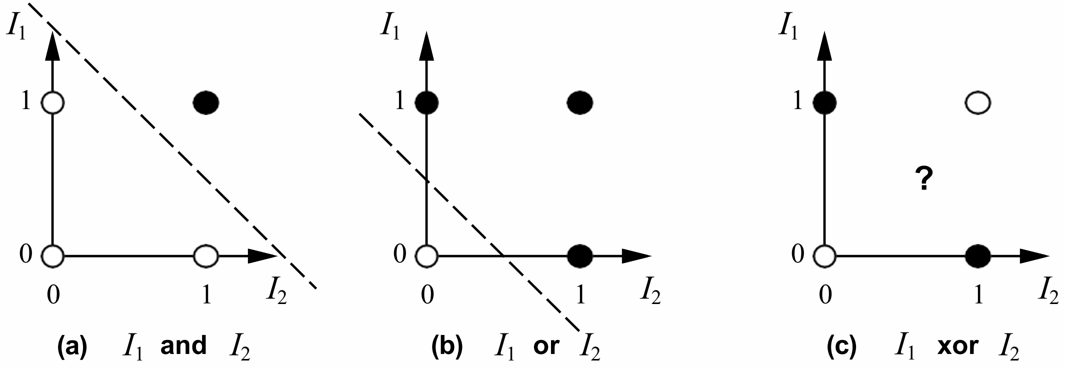
Feb 19, 2025

Outline

- Multi-Layer Neural Networks
- Continuous Activation Functions (3.10)
- Gradient Descent (4.3)
- Backpropagation (6.5.2)
- Examples
- Momentum and Adam

Recall: Limitations of Perceptrons

Problem: many useful functions are not linearly separable (e.g. XOR)



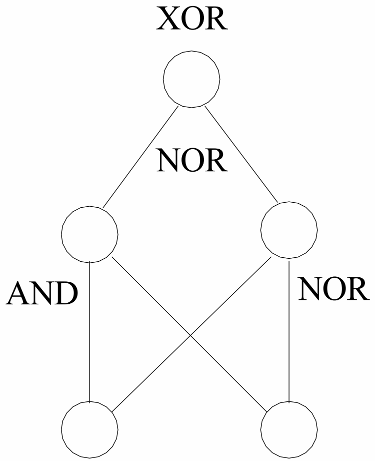
(a) I_1 **and** I_2 **(b)** I_1 **or** I_2 **(c)** I_1 **xor** I_2

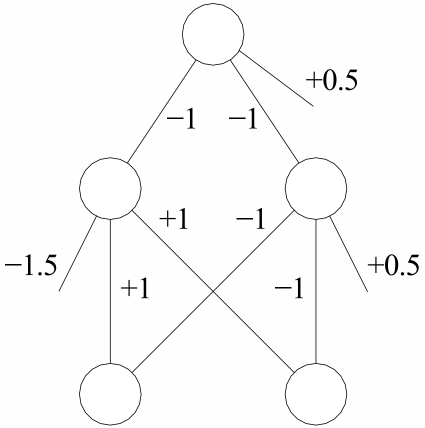
Possible solution:

x_1 XOR x_2 can be written as: (x_1 AND x_2) NOR (x_1 NOR x_2)

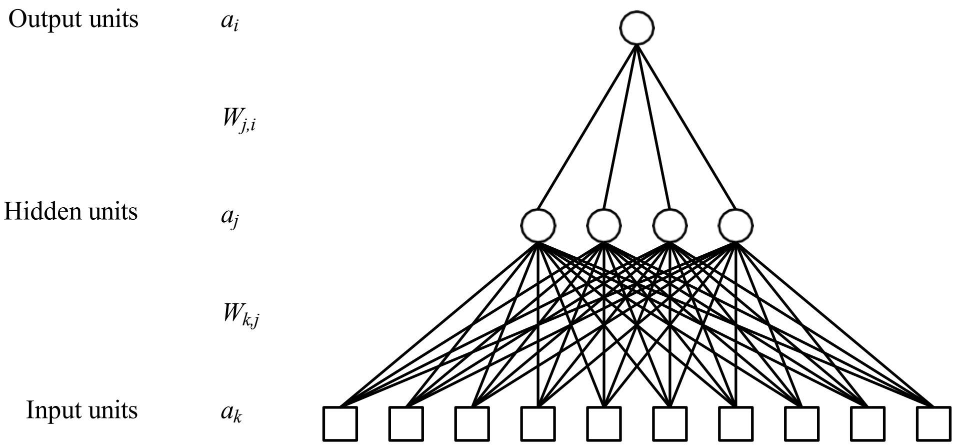
Recall that AND, OR and NOR can be implemented by perceptrons.

Multi-Layer Neural Networks



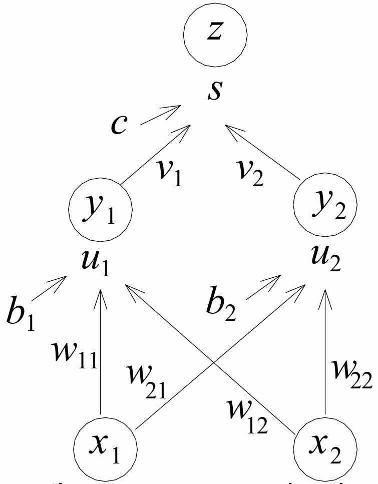


Two-Layer Neural Network



Problem: How can we train it to learn a new function?

Neural Network Equations



$$\begin{aligned} u_1 &= b_1 + w_{11}x_1 + w_{12}x_2 \\ y_1 &= g(u_1) \\ s &= c + v_1 y_1 + v_2 y_2 \\ z &= g(s) \end{aligned}$$

We sometimes use w as a shorthand for any of the trainable weights $\{c, v_1, v_2, b_1, b_2, w_{11}, w_{21}, w_{12}, w_{22}\}$.

NN Training as Cost Minimization

- We define an **error** function or **loss** function E to be (half) the sum over all input patterns of the square of the difference between actual output (**z**) and **target** output (**t**)

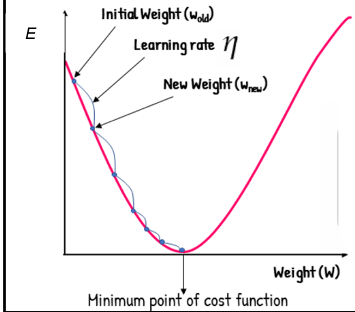
$$\boxed{E = \frac{1}{2}\sum_i (z_i - t_i)^2}$$

- i: it's training sample
- If we think of E as height, it defines an error **landscape** on the weight space.
- The aim is to find a set of weights for which E is very low.

Gradient Descent (4.3)

$$E = \frac{1}{2}\sum_i (z_i - t_i)^2$$

- The aim is to find a set of weights for which E is very low.



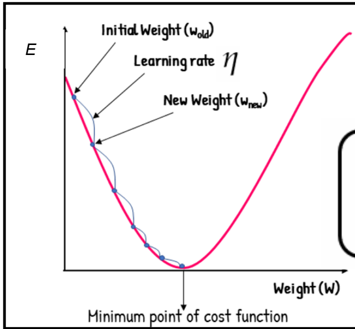
Do we increase or decrease the weight?

https://www.linkedin.com/pulse/understanding-gradient-descent-algorithm-its-role-linear-mhango-kjbvf

Gradient Descent (4.3)

$$\boxed{E = \frac{1}{2}\sum_i (z_i - t_i)^2}$$

- The aim is to find a set of weights for which E is very low.



$$\boxed{\overset{\text{new weight}}{w} \leftarrow \overset{\text{old weight}}{w} - \eta \frac{\partial E}{\partial w}}$$

https://www.linkedin.com/pulse/understanding-gradient-descent-algorithm-its-role-linear-mhango-kjbvf

Gradient Descent (4.3)

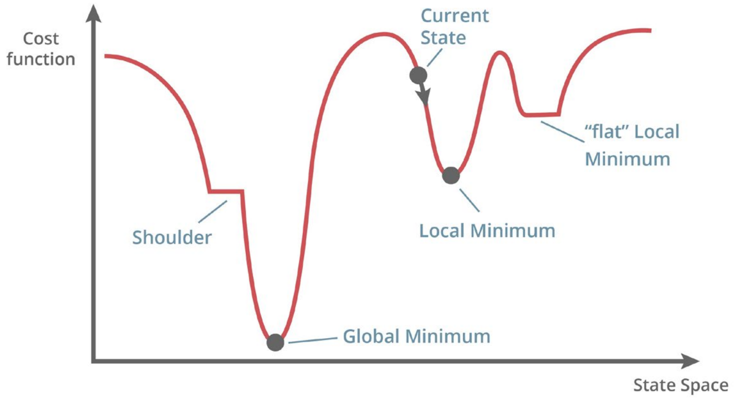
$$\boxed{E = \frac{1}{2}\sum_i (z_i - t_i)^2}$$

- The aim is to find a set of weights for which E is very low.
- If the functions involved are smooth, we can use multi-variable calculus to adjust the weights in such a way as to take us in the steepest downhill direction.

$$\boxed{w \leftarrow w - \eta \frac{\partial E}{\partial w}}$$

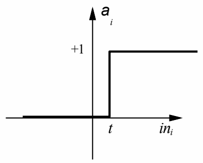
- Parameter η is called the *learning rate*.

Local Search in Weight Space

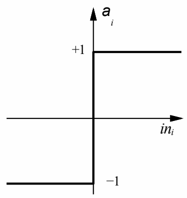


Problem: because of the step function, the landscape will not be smooth, but will instead consist almost entirely of flat local regions and “shoulders”, with occasional discontinuous jumps.

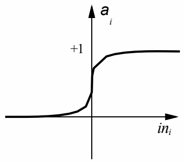
Continuous Activation Functions (3.10)



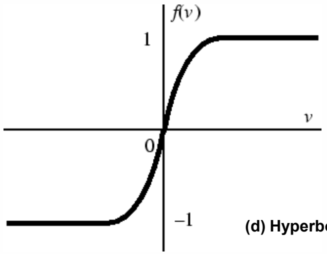
(a) Step function



(b) Sign function



(c) Sigmoid function



(d) Hyperbolic function

Key Idea: Replace the (discontinuous) step function with a differentiable function, such as the sigmoid:

$$\boxed{g(s) = \frac{1}{1 + e^{-s}}}$$

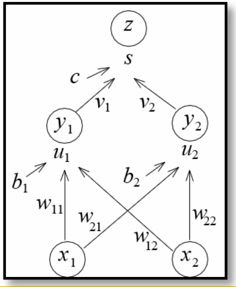
or hyperbolic tangent

$$\boxed{g(s) = \tanh(s) = \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}} = 2\left(\frac{1}{1 + e^{-2s}}\right) - 1}$$

Chain Rule (6.5.2)

- If, say $\boxed{\begin{aligned} y &= y(u) \\ u &= u(x) \end{aligned}}$ Then $\boxed{\dfrac{\partial y}{\partial x} = \dfrac{\partial y}{\partial u}\dfrac{\partial u}{\partial x}}$

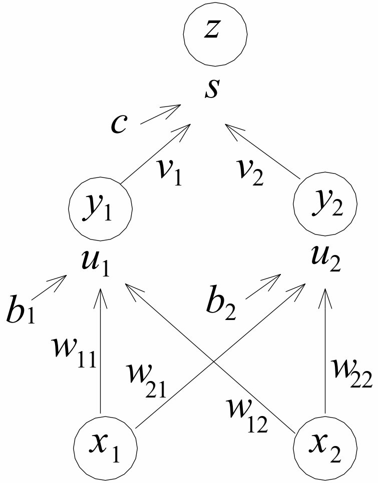
- This principle can be used to compute the partial derivatives in an efficient and localized manner. Note that the transfer function must be differentiable (usually sigmoid, or tanh).



Note: if $z(s) = \dfrac{1}{1+e^{-s}}, \longrightarrow z'(s) = z(1-z).$

if $z(s) = \tanh(s), \longrightarrow z'(s) = 1 - z^2.$

Forward Pass



This is what we know after the forward pass

$$\begin{aligned} u_1 &= b_1 + w_{11}x_1 + w_{12}x_2 \\ u_2 &= b_2 + w_{21}x_1 + w_{22}x_2 \\ y_1 &= g(u_1), y_2 = g(u_2) \\ s &= c + v_1y_1 + v_2y_2 , z = g(s) \\ E &= \frac{1}{2}\sum(z-t)^2 \end{aligned}$$

$$z(s) = \frac{1}{1+e^{-s}}, \longrightarrow z'(s) = z(1-z).$$

$$z(s) = \tanh(s), \longrightarrow z'(s) = 1 - z^2.$$

Chain Rule (6.5.2)

- If, say $\boxed{\begin{aligned} y &= y(u) \\ u &= u(x) \end{aligned}}$ Then $\boxed{\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}}$

?

$\frac{\partial E}{\partial s}$

$\frac{\partial E}{\partial v_1}$

$\frac{\partial E}{\partial w_{ij}}$

1 $\frac{\partial s}{\partial y_j}$

2 $\frac{\partial E}{\partial z}$

4 $\frac{\partial z}{\partial s}$

6 $\frac{\partial s}{\partial v_2}$

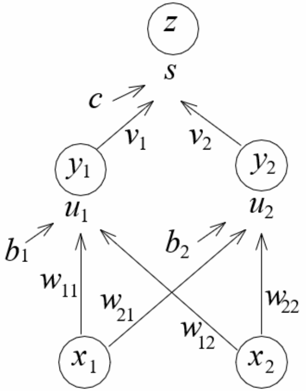
3 $\frac{\partial u_j}{\partial w_{ij}}$

5 $\frac{\partial y_j}{\partial u_j}$

7 $\frac{\partial s}{\partial v_1}$

Backpropagation – calculate weights for second layer

$$\boxed{\frac{\partial E}{\partial v_1} = \boxed{\frac{\partial E}{\partial s}} \cdot \boxed{\frac{\partial s}{\partial v_1}}, \quad \frac{\partial E}{\partial v_2} = \frac{\partial E}{\partial s} \cdot \frac{\partial s}{\partial v_2}, \quad \frac{\partial E}{\partial c} = \frac{\partial E}{\partial s} \cdot \frac{\partial s}{\partial c}}$$



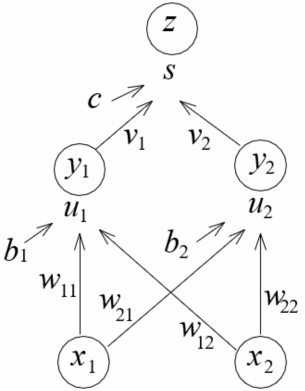
$$\frac{\partial E}{\partial z} = ? \qquad \frac{\partial z}{\partial s} = ?$$

$$\frac{\partial E}{\partial s} = \frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial s}$$

$$\frac{\partial s}{\partial v_1} = ? \qquad \frac{\partial s}{\partial v_2} = ? \qquad \frac{\partial s}{\partial c} = ?$$

Backpropagation – calculate weights for second layer

$$\frac{\partial E}{\partial v_1} = \boxed{\frac{\partial E}{\partial s}} \cdot \boxed{\frac{\partial s}{\partial v_1}}, \quad \frac{\partial E}{\partial v_2} = \frac{\partial E}{\partial s} \cdot \frac{\partial s}{\partial v_2}, \quad \frac{\partial E}{\partial c} = \frac{\partial E}{\partial s} \cdot \frac{\partial s}{\partial c}$$



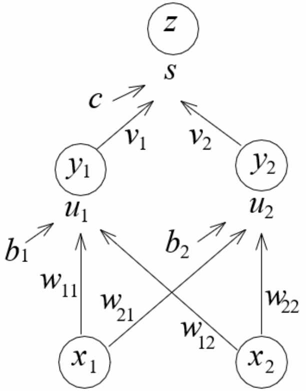
$$\frac{\partial E}{\partial z} = z - t \qquad \frac{\partial z}{\partial s} = g'(s)$$

$$\frac{\partial E}{\partial s} = \frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial s}$$

$$\frac{\partial s}{\partial v_1} = y_1, \quad \frac{\partial s}{\partial v_2} = y_2, \quad \frac{\partial s}{\partial c} = 1$$

Backpropagation – calculate weights for first layer

$$\frac{\partial E}{\partial w_{ij}} = \boxed{\frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial s}} \cdot \boxed{\frac{\partial s}{\partial y_j} \cdot \frac{\partial y_j}{\partial u_j}} \cdot \boxed{\frac{\partial u_j}{\partial w_{ij}}}$$



$$\frac{\partial E}{\partial s} = \frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial s} \quad ?$$

$$\frac{\partial s}{\partial y_1} = ? \quad \frac{\partial s}{\partial y_2} = ?$$

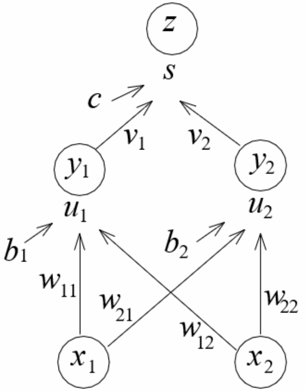
$$\frac{\partial y_1}{\partial u_1} = ? \quad \frac{\partial y_2}{\partial u_2} = ?$$

$$\frac{\partial u_1}{\partial w_{11}} = ? \quad \frac{\partial u_1}{\partial w_{12}} = ? \quad \frac{\partial u_1}{\partial b_1} = ?$$

$$\frac{\partial u_2}{\partial w_{21}} = ? \quad \frac{\partial u_2}{\partial w_{22}} = ? \quad \frac{\partial u_2}{\partial b_2} = ?$$

Backpropagation – calculate weights for first layer

$$\frac{\partial E}{\partial w_{ij}} = \boxed{\frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial s}} \cdot \boxed{\frac{\partial s}{\partial y_j} \cdot \frac{\partial y_j}{\partial u_j}} \cdot \boxed{\frac{\partial u_j}{\partial w_{ij}}}$$



$$\frac{\partial E}{\partial s} = \frac{\partial E}{\partial z} \cdot \frac{\partial z}{\partial s}$$ already calculated

$$\frac{\partial s}{\partial y_1} = v_1, \quad \frac{\partial s}{\partial y_2} = v_2$$

$$\frac{\partial y_1}{\partial u_1} = g'(u_1), \quad \frac{\partial y_2}{\partial u_2} = g'(u_2)$$

$$\frac{\partial u_1}{\partial w_{11}} = x_1, \quad \frac{\partial u_1}{\partial w_{12}} = x_2, \quad \frac{\partial u_1}{\partial b_1} = 1$$
$$\frac{\partial u_2}{\partial w_{21}} = x_1, \quad \frac{\partial u_2}{\partial w_{22}} = x_2, \quad \frac{\partial u_2}{\partial b_2} = 1$$

Training Tips

- ➛ re-scale inputs and outputs to be in the range 0 to 1 or -1 to 1
  - → otherwise, backprop may put undue emphasis on larger values
- ➛ replace missing values with mean value for that attribute
- ➛ weight initialization
  - → for shallow networks, initialize weights to small random values
  - → for deep networks, more sophisticated strategies to counter exploding or vanishing gradients
- ➛ three different ways to prevent overfitting:
  - → limit the number of hidden nodes or connections
  - → limit the training time, using a validation set
  - → weight decay
- ➛ adjust learning rate (and other parameters) to suit the particular task
- ➛ on-line, batch, mini-batch

Types of training

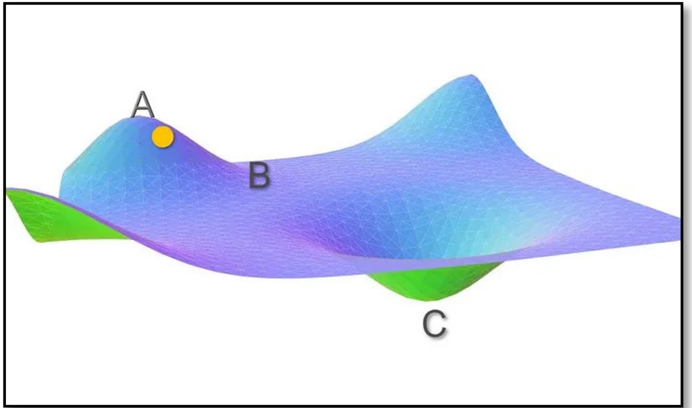
Initial definitions

- ➢ **Epoch:** One complete pass of the training set
- ➢ **Batch size:** Number of samples from training dataset used
- ➢ **Iteration:** One update of the model parameters. Thus, no. of iterations to complete one epoch depends on training dataset and batch size (e.g., For example, if you have 1000 examples in training set and use a batch size of 100, you'd have 10 iterations per epoch)

Types of training

- ➢ **Stochastic Gradient Descent/online:** one training sample. Cons: Time
- ➢ **Batch:** all data. Cons: Resources (for large data)
- ➢ **Mini-batch:** use a subset of the data.

Momentum (8.3)



- Gradient at B is very low, but we want to get to C
- What do we do?

Momentum (8.3)

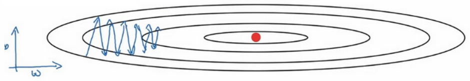
- If the landscape is shaped like a “rain gutter”, weights will tend to oscillate without much improvement. We can add a momentum factor

$$\boxed{\begin{aligned} \delta w &\leftarrow a\ \delta w - \eta \frac{\partial E}{\partial w} \\ w &\leftarrow w + \delta w \end{aligned}}$$

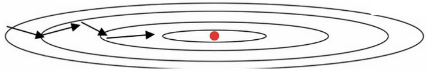
0<=a<=1

- Hopefully, this will dampen sideways oscillations but amplify downhill motion by $\frac{1}{1-\alpha}$. Momentum can also help to escape from local minima, or move quickly across flat regions in the loss landscape.
- When momentum is used, we generally reduce the learning rate at the same time, in order to compensate for the implicit factor of $\frac{1}{1-\alpha}$.

RMSprop (8.5.2)



Gradient descent



RMSprop

$$v \leftarrow \beta v + (1-\beta)\left(\frac{\partial E}{\partial w}\right)^2$$

$$w \leftarrow w - \frac{\eta}{\sqrt{v+\epsilon}}\frac{\partial E}{\partial w}$$

0<=b<=1

- Adaptive learning rates
- Scales the learning rate inversely to the moving average of past squared gradients.
- If gradients are large, the denominator increases, reducing the effective learning rate. If gradients are small, the learning rate remains relatively high. This prevents drastic weight updates and stabilizes training.

Adaptive Moment Estimation (Adam)

- Each parameter is adjusted according to:

$$\boxed{w \leftarrow w - \frac{\eta}{\sqrt{\hat{v} + \epsilon}} \hat{m}}$$

- Maintain a running average of the gradients (m_t) and squared gradients (v_t) for each weight in the network.

$$m \leftarrow \beta_1 m + (1 - \beta_1) \frac{\partial E}{\partial w}$$

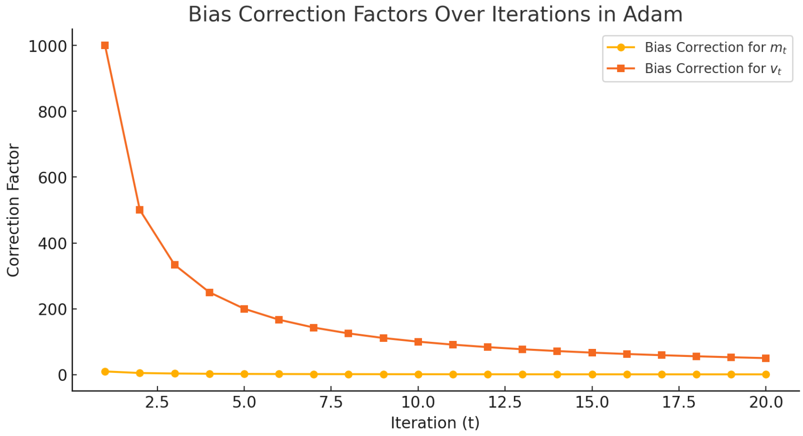
$$v \leftarrow \beta_2 v + (1 - \beta_2) \left(\frac{\partial E}{\partial w} \right)^2$$

- To speed up training in the early stages, compensating for the fact that m_t, v_t are initialized to zero, we rescale as follows:

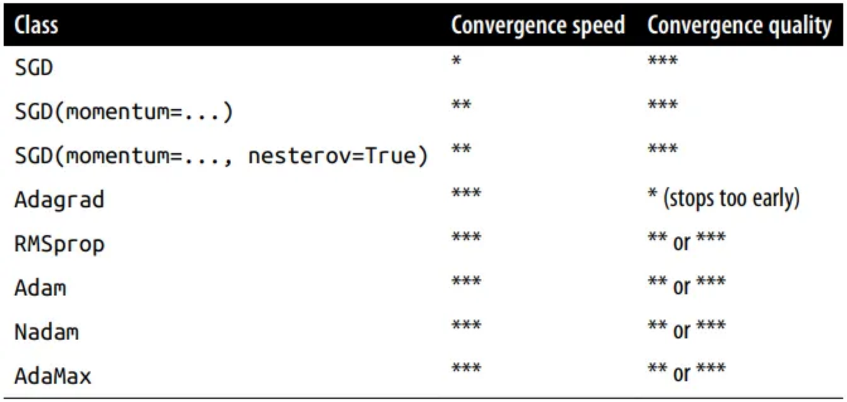
$$\boxed{\hat{m} \leftarrow \frac{m}{1 - \beta_1^t}, \quad \hat{v} \leftarrow \frac{v}{1 - \beta_2^t}}$$

$0 <= b_1 <= 1$
$0 <= b_2 <= 1$

Adaptive Moment Estimation (Adam)



Optimizer summary



Class	Convergence speed	Convergence quality
SGD	*	***
SGD(momentum=...)	**	***
SGD(momentum=..., nesterov=True)	**	***
Adagrad	***	* (stops too early)
RMSprop	***	** or ***
Adam	***	** or ***
Nadam	***	** or ***
AdaMax	***	** or ***

Optimizer comparison (* is bad, ** is average, and *** is good). (Image credits: Hands on Machine Learning by Geron Aurelien, page 359.)

Second Order Methods

- Some optimization methods involve computing *second order* partial derivatives of the loss function with respect to each *pair* of weights:

$$\boxed{\frac{\partial^2 E}{\partial w_i \partial w_j}}$$

  - ➢ Conjugate Gradients
    - → approximate the landscape with a quadratic function (paraboloid) and jump to the minimum of this quadratic function
  - ➢ Natural Gradients (Amari, 1995)
    - → use methods from information geometry to find a "natural" re-scaling of the partial derivatives
- These methods are not normally used for deep learning, because the number of weights is too high. In practice, the Adam optimizer tends to provide similar benefits with low computational cost.

Two-Layer NN’s – Applications

- Medical Dignosis
- Autonomous Driving
- Game Playing
- Credit Card Fraud Detection
- Handwriting Recognition
- Financial Prediction

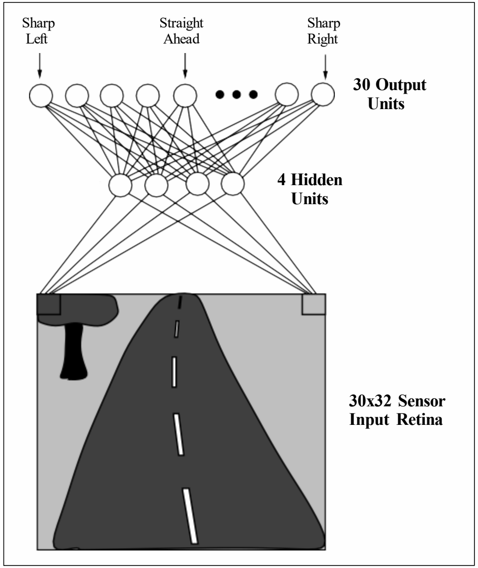
Example: Pima Indians Diabetes Dataset

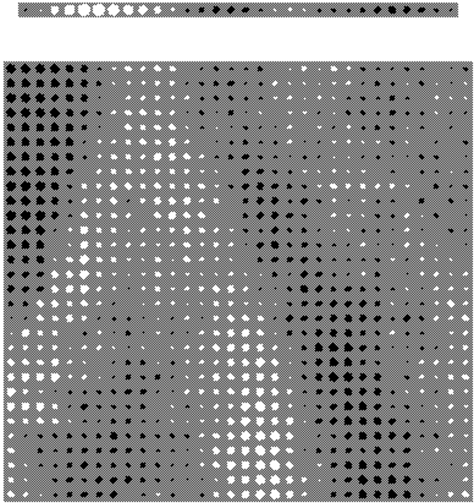
	Attribute	mean	stdv
1.	Number of times pregnant	3.8	3.4
2.	Plasma glucose concentration	120.9	32.0
3.	Diastolic blood pressure (mm Hg)	69.1	19.4
4.	Triceps skin fold thickness (mm)	20.5	16.0
5.	2-Hour serum insulin (mu U/ml)	79.8	115.2
6.	Body mass index (weight in kg/(height in m)^2)	32.0	7.9
7.	Diabetes pedigree function	0.5	0.3
8.	Age (years)	33.2	11.8

Based on these inputs, try to predict whether the patient will develop Diabetes (1) or Not (0).

ALVINN (Pomerleau 1991, 1993)

ALVINN

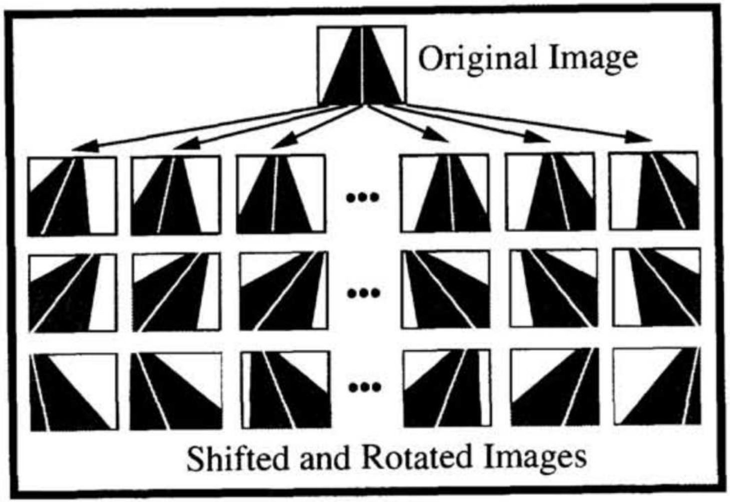

Sharp Left
Straight Ahead
Sharp Right
30 Output Units
4 Hidden Units
30x32 Sensor Input Retina



ALVINN

- Autonomous Land Vehicle In a Neural Network
- Later version included a sonar range finder
  - 8×32 range finder input retina
  - 29 hidden units
  - 45 output units
- Supervised Learning, from human actions (Behavioral Cloning)
  - Replay Memory – experiences are stored in a database and randomly shuffled for training
  - Data Augmentation – additional “transformed” training items are created, in order to cover emergency situations
- drove autonomously from coast to coast

Data Augmentation


Original Image
...
...
...
Shifted and Rotated Images