Spatial Ecology

Artificial Neural Networks for Geodata

Antonio Fonseca

Our roadmap

Day 1:
- Into to Neural Nets
- Implementation (PyTorch)

Day 2:
- Autoencoders (AE), Variational AE and Generative Adversarial Nets
- Implementation (PyTorch)

Day 3:
- RNN, LSTM and Transformers
- Implementation (TensorFlow)

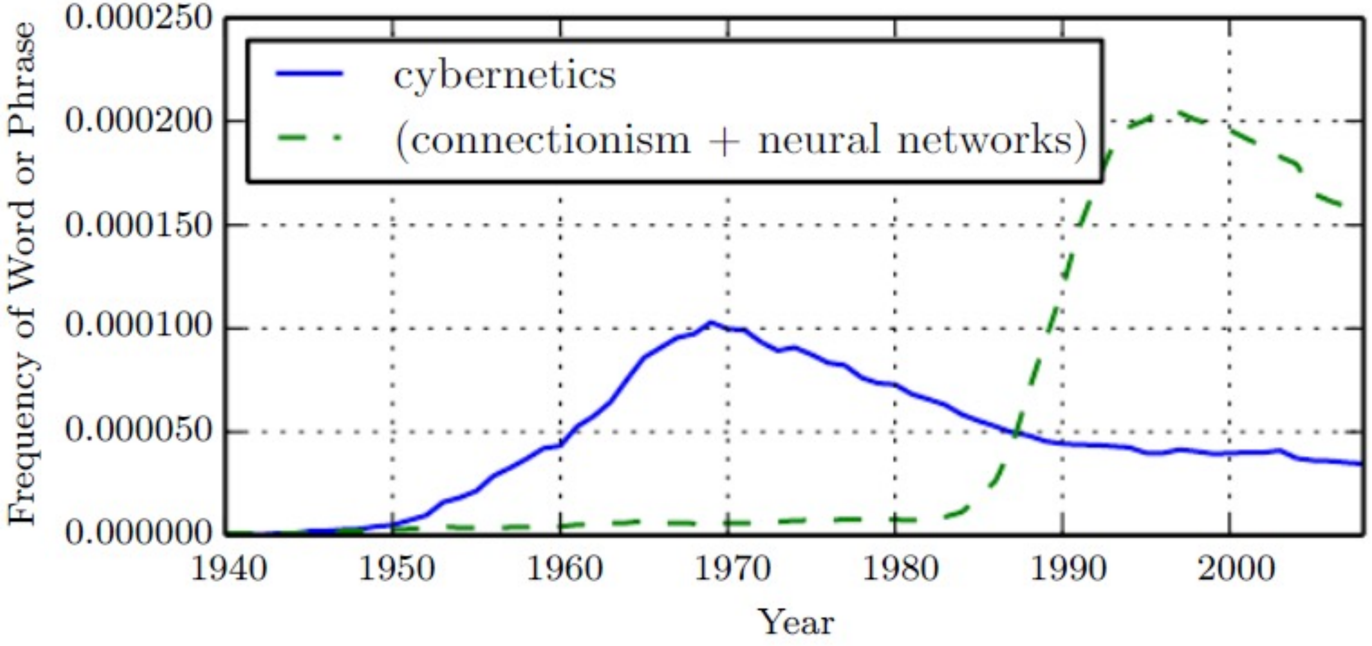
Agenda

1) Background in Artificial Neural Networks (ANNs)
- Biological inspiration
- Applications: Classification, Regression
- Components of ANNs

2) Using libraries to build an ANN (PyTorch)
- Opmitizers
- Dropout
- Early stop
- Regularization
- Deeper nets

Evolution of ANNs

Google Books (by Goodfellow et. al, 2016)



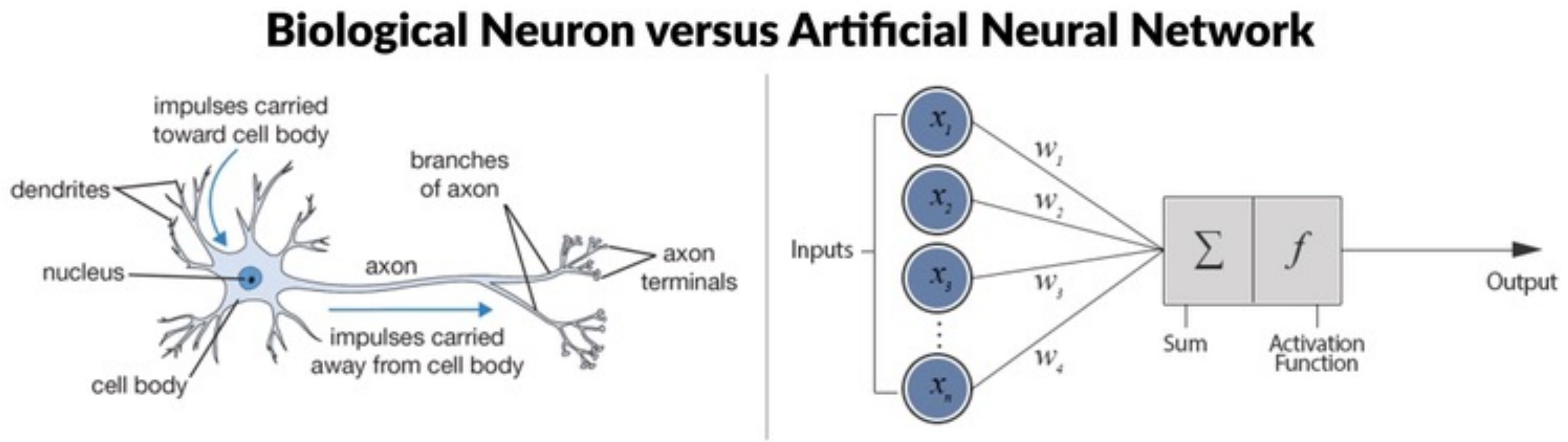
First wave

Second wave

Third wave

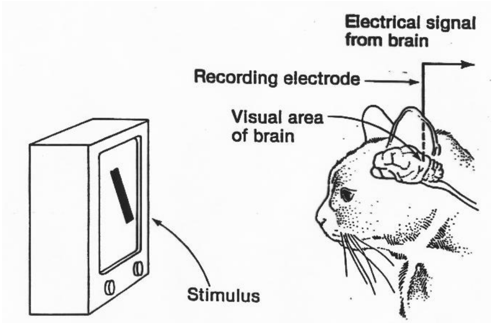
1) Biological Learning Theory (1943, 1949)
2) Perceptron (1958)
3) Backpropagation (1986)
4) Deep Learning (2006, 2007)

ANNs architecture

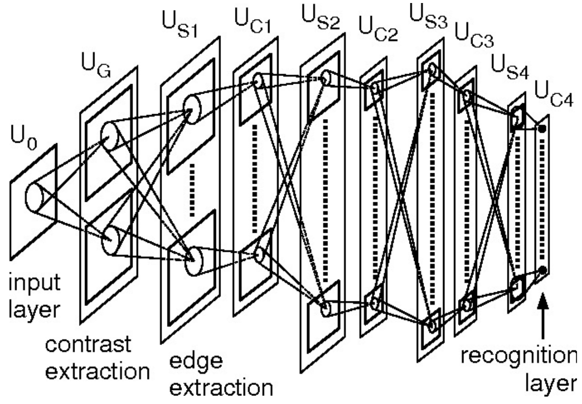


Brain “inspired” model

- Not enough info about brain processing…
- But we know the basics:



Hubel and Wiesel, 1959-1968



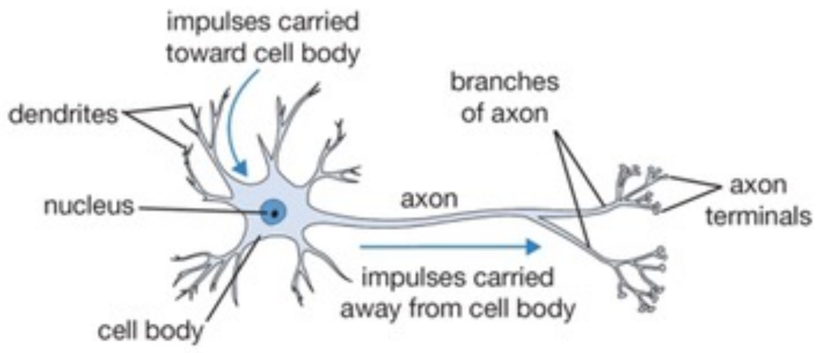
Fukushima, 1980

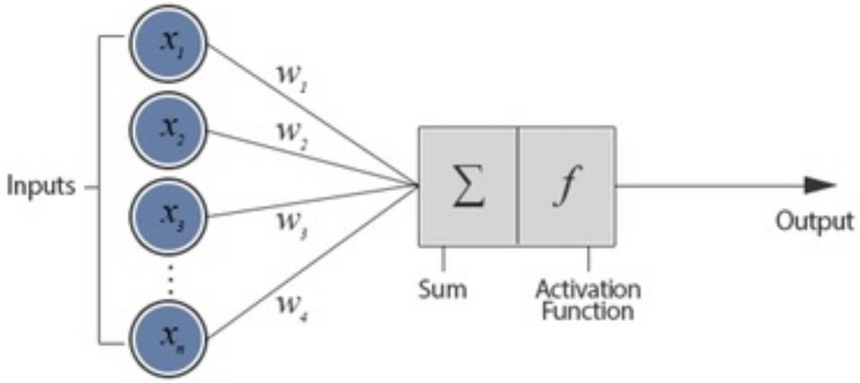
Learning algorithms

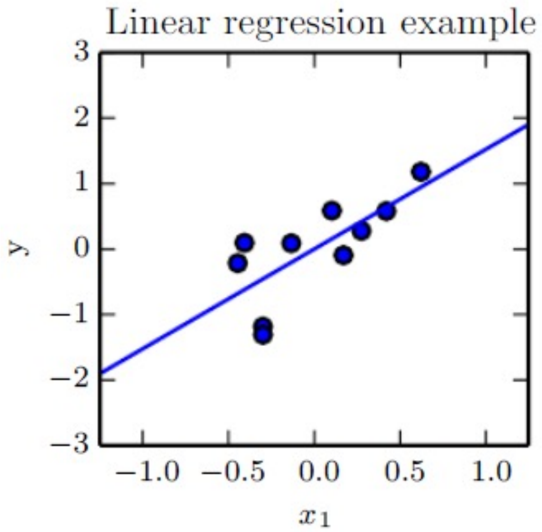
"A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P** , if its performance at tasks in **T** , as measured by **P** , improves with experience **E**."

Tasks (T)	**Performance (P)**	**Experience (E)**
Transcription		Supervised Learning
Machine Translation	Accuracy rate	
Classification		
Anomaly detection		Unsupervised Learning
Synthesis and sampling		
⋮	Adjusted R^2	
Regression	RMSE/MSE/MAE	Reinforcement Learning

Biological Neuron versus Artificial Neural Network







Task (T)

Input $x \in \mathbb{R}^n$

Weights $w \in \mathbb{R}^n$

$$\hat{y} = w^T x$$

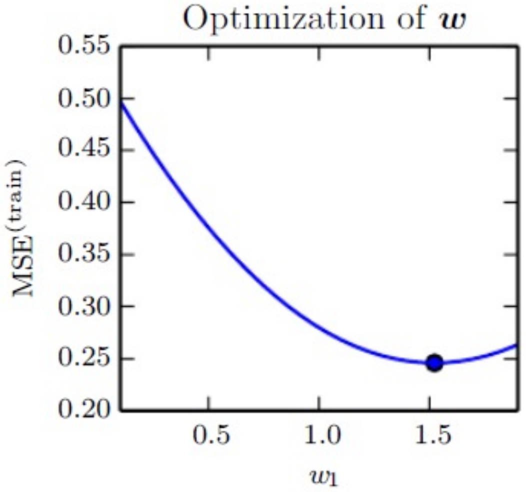
$$f(x, w) = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

Performance (P)

$$MSE_{test} = \frac{1}{m} \sum_i (\hat{y}_{test} - y_{test})_i^2$$

Training

$$\nabla_w \left(\frac{1}{m} \sum_i (w^T X_{train} - y_{train})_i^2 \right) = 0$$



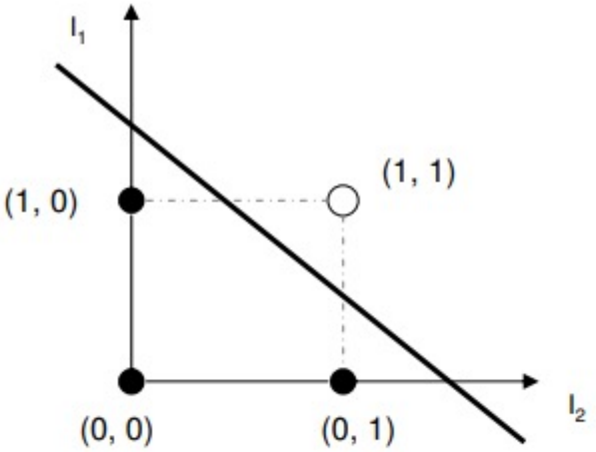
Dataset

$$(X, y) \begin{cases} (X_{train}, y_{train}) \\ (X_{test}, y_{test}) \end{cases}$$

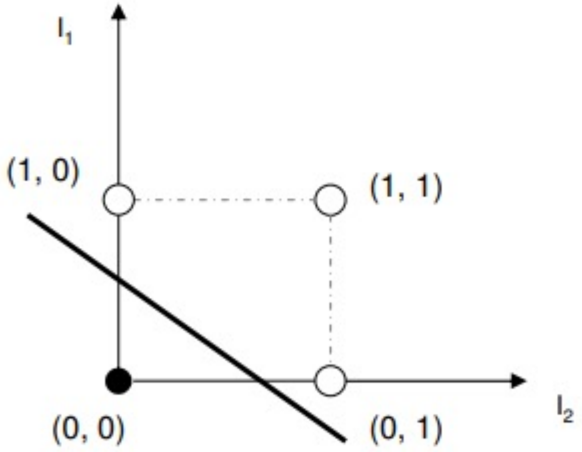
Solves linear problems

Can't solve the XOR problem

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

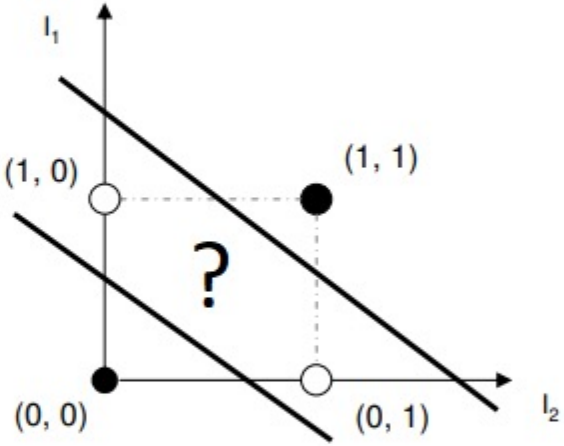


OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1

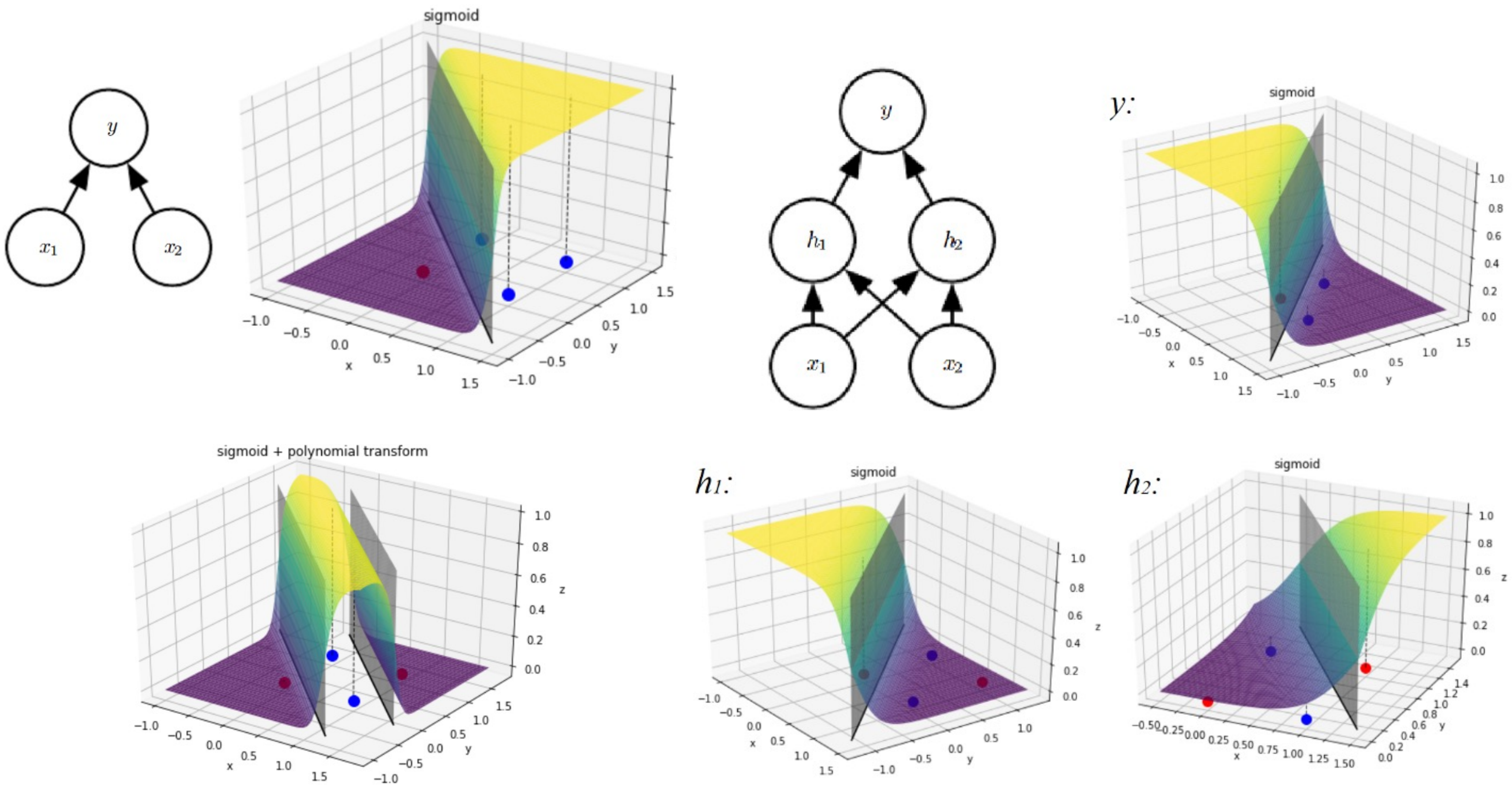


Perceptron

XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

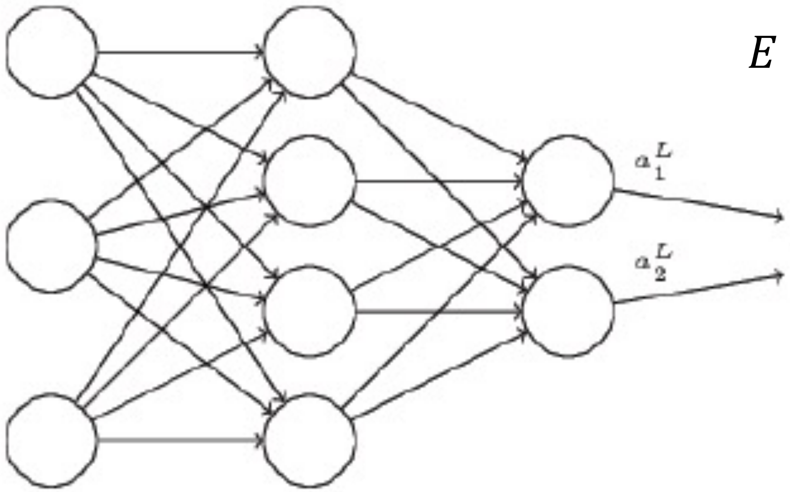


Multilayer Perceptron



sigmoid
sigmoid + polynomial transform
y:
h1:
h2:
sigmoid
sigmoid
sigmoid
y
x_1
x_2
h_1
h_2
x
y
z

Backpropagation



$$E = \frac{1}{2}\sum_i \left(a_i^L - y_i\right)^2$$

$$cost\ E = E(a^L)$$

1. **Input** x**:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

3. **Output error** δ^L**:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

$$z_j^l = \sum_i w_{ji}^l a_i^{l-1} + b_j^l \qquad a_j^l = \sigma\left(\sum_i w_{ji}^l a_i^{l-1} + b_j^l\right) = \sigma(z_j^l)$$

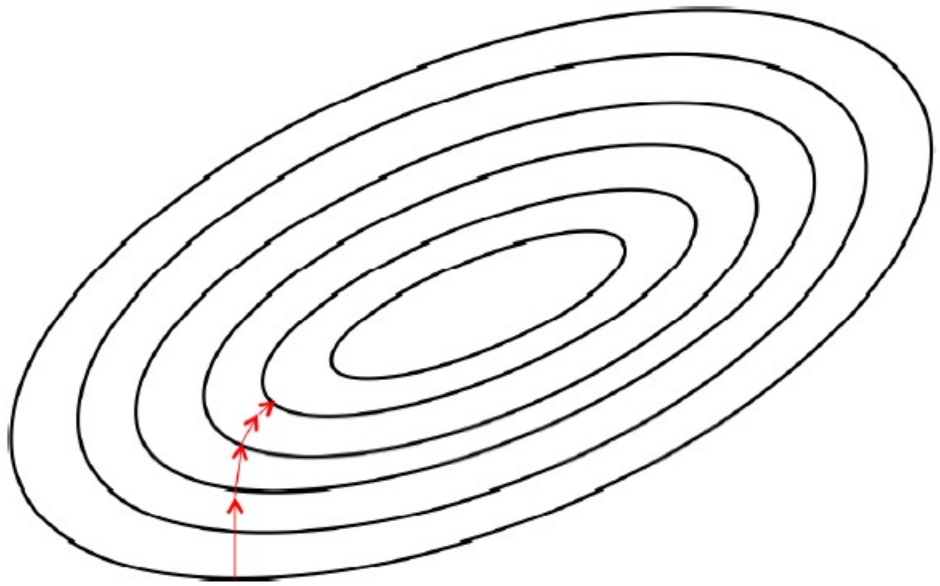
$$\delta_j^L \equiv \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial a_i^L}\frac{\partial a_i^L}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L}\sigma'(z_j^L) \qquad (1)$$

$$\delta_j^l \equiv \frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial z_i^{l+1}}\frac{\partial z_i^{l+1}}{\partial z_j^l} = \frac{\partial z_i^{l+1}}{\partial z_j^l}\delta_i^{l+1}$$

$$= \frac{\partial(\sum_i w_{ij}^{l+1} a_j^l + b_i^{l+1})}{\partial z_j^l}\delta_j^{l+1} = \sum_i w_{ij}^{l+1}\delta_i^{l+1}\sigma'(z_j^l) \qquad (2)$$

$$\frac{\partial E}{\partial w_{ji}^l} = \frac{\partial E}{\partial a_i^l}\frac{\partial a_i^l}{\partial z_j^l}\frac{\partial(w_{ji}^l a_i^{l-1})}{\partial w_{ji}^l}$$

Optimizers

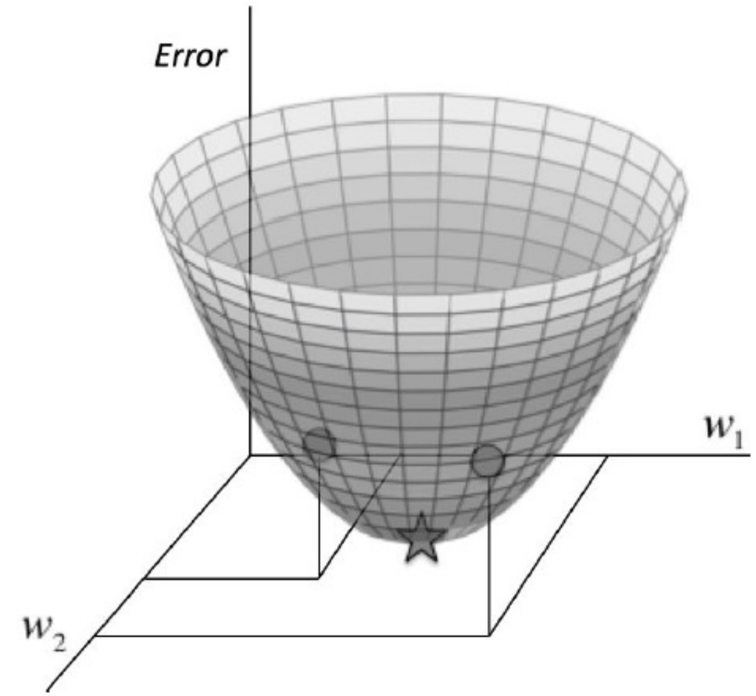


Gradient

$$\Delta w_k = -\frac{\partial E}{\partial w_k}$$

$$= -\frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + \Delta w_k$$



Stochastic gradient descent (**SGD**)

Optimizers

Practical test:
lr_val = [1; 0.1; 0.01]
momentum_val = 0
nesterov_val = ‘False'
decay_val = 1e-6

Hyperparameters

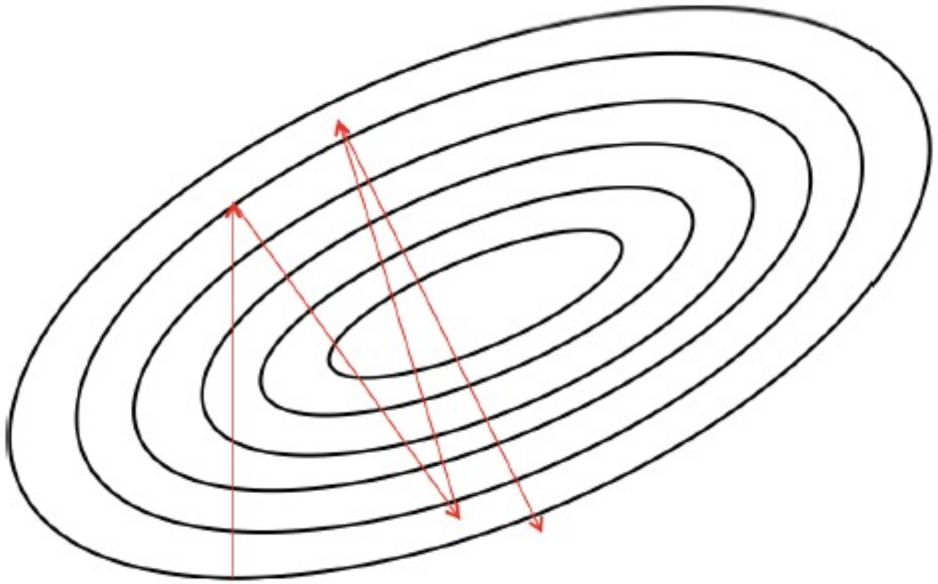
- Learning rate (α)

$$\Delta w_k = -\alpha \frac{\partial E}{\partial w_k}$$

$$= -\alpha \frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + \Delta w_k$$

Stochastic gradient descent (**SGD**)



Result of a large learning rate α

Optimizers

Hyperparameters

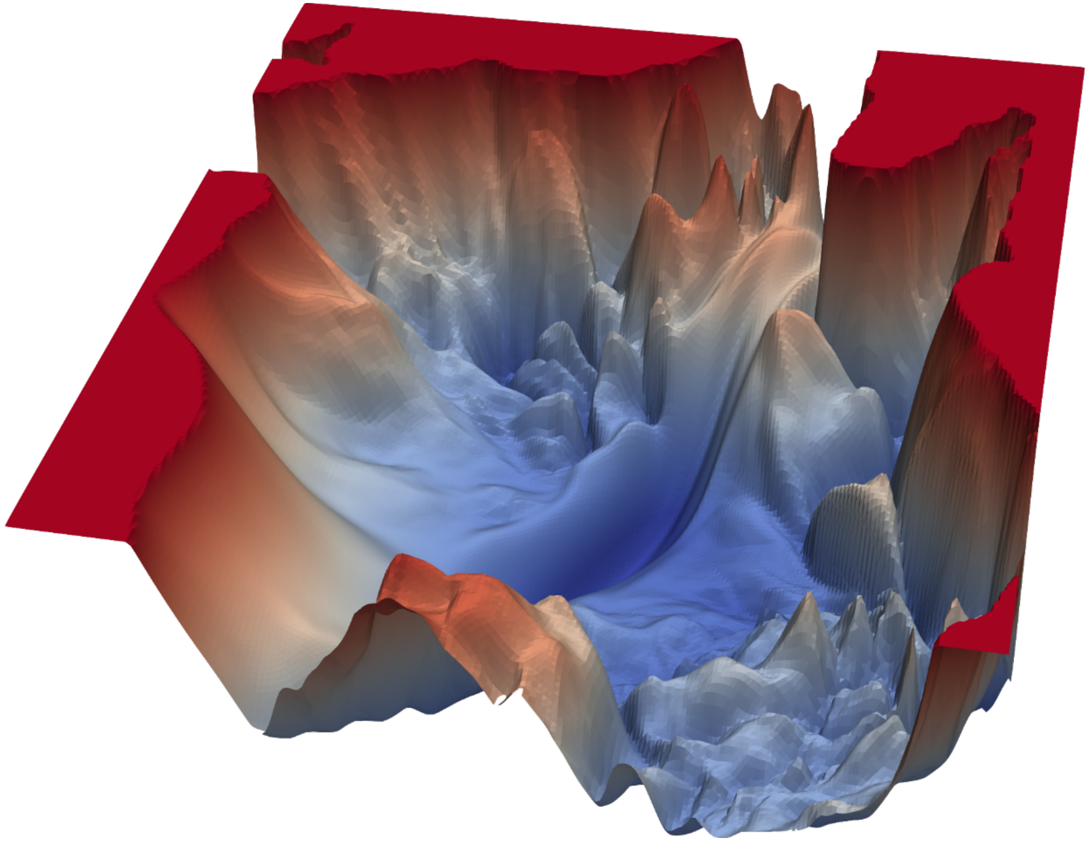
- Learning rate (α)

$$\Delta w_k = -\alpha \frac{\partial E}{\partial w_k}$$

$$= -\alpha \frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + \Delta w_k$$

Stochastic gradient descent (**SGD**)



Optimizers

Hyperparameters

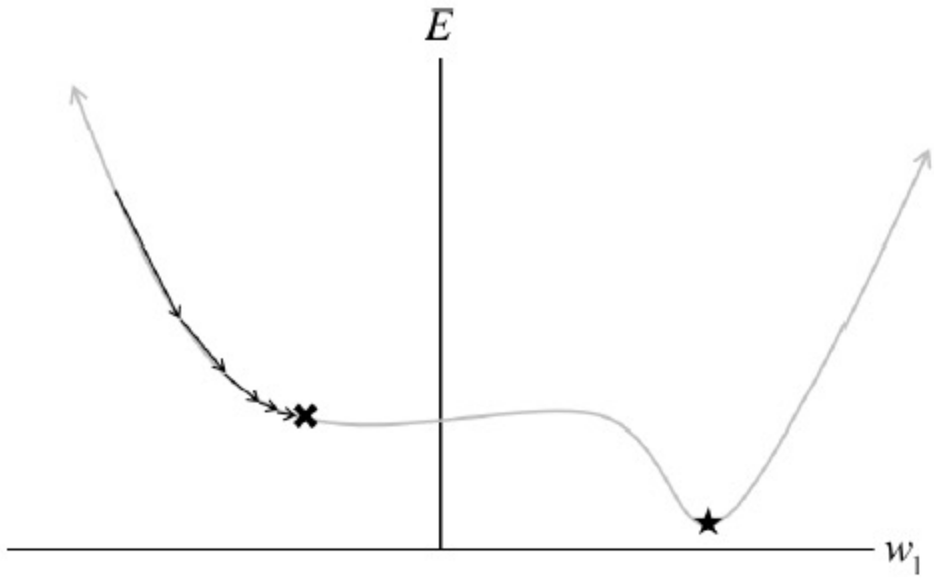
- Learning rate (α)

$$\Delta w_k = -\alpha \frac{\partial E}{\partial w_k}$$

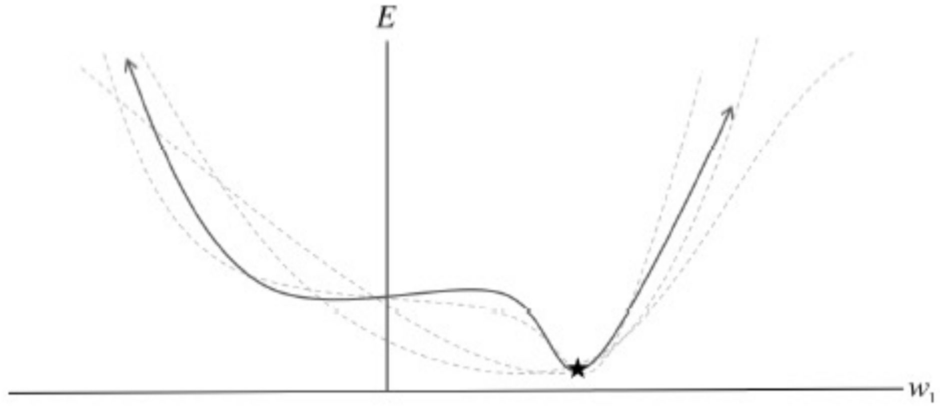
$$= -\alpha \frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + \Delta w_k$$

Stochastic gradient descent (**SGD**)



Local Minima



Multiple samples

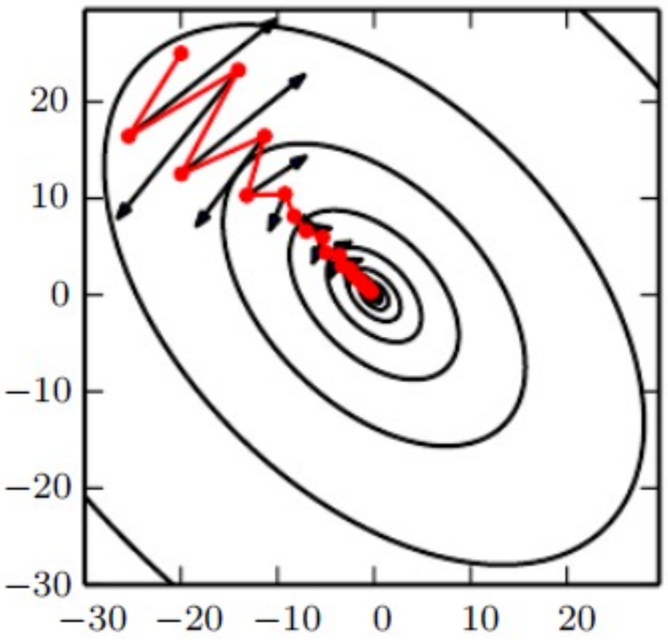
Optimizers

Hyperparameters

- Learning rate (α)
- Momentum (β)

$$v_{i+1} = v\beta - \alpha \frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + v$$



SGD

SGD+Momentum

Stochastic gradient descent with momentum (**SGD+Momentum**)

Optimizers

Hard to pick right hyperparameters

- Small learning rate: long convergence time
- Large learning rate: convergence problems

Adagrad: adapts learning rate to each parameter

$$\Delta w_{k,t} = -\alpha \frac{\partial E_t}{\partial w_{k,t}} = -\alpha \nabla_w E(w_t)$$

- Learning rate might decrease too fast
- Might not converge

$$g_{t,i} = \nabla_w E(w_{t,i})$$

$$G_{t+1,i} = G_{t,i} + g_{t,i} \odot g_{t,i}$$

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

Optimizers

RMSprop: decaying average of the past squared gradients

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

Decaying average

Adadelta

$$E[\Delta_w^2]_t = \gamma E[\Delta_w^2]_{t-1} + (1-\gamma)\Delta_w^2$$

$$\Delta w_t = \frac{\sqrt{E[\Delta_w^2]_t + \epsilon}}{\sqrt{G_{t,i} + \epsilon}} g_t$$

$$\Delta w_{k,t} = -\alpha \frac{\partial E_t}{\partial w_{k,t}} = -\alpha \nabla_w E(w_t) = -\alpha g_{t,i}$$

$$g_{t,i} = \nabla_w E(w_{t,i})$$

$$G_{t+1,i} = \gamma G_{t,i} + (1-\gamma) g_{t,i} \odot g_{t,i}$$

$$\Longrightarrow$$

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

Optimizers

ADAM: decaying average of the past squared gradients and momentum

RMSprop / Adadelta

$$g_{t,i} = \nabla_w E(w_{t,i})$$

$$G_{t+1,i} = \gamma G_{t,i} + (1-\gamma) g_{t,i} \odot g_{t,i}$$

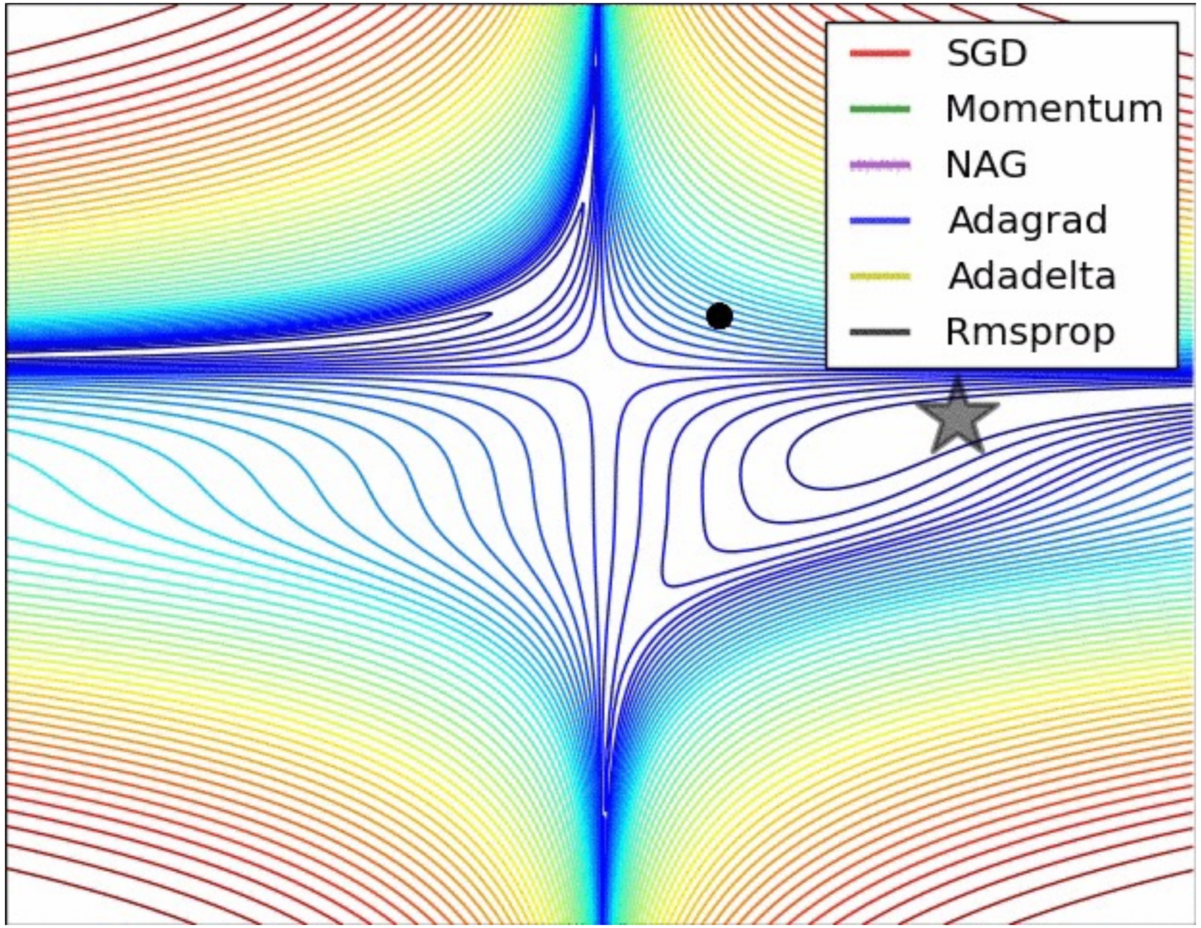
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

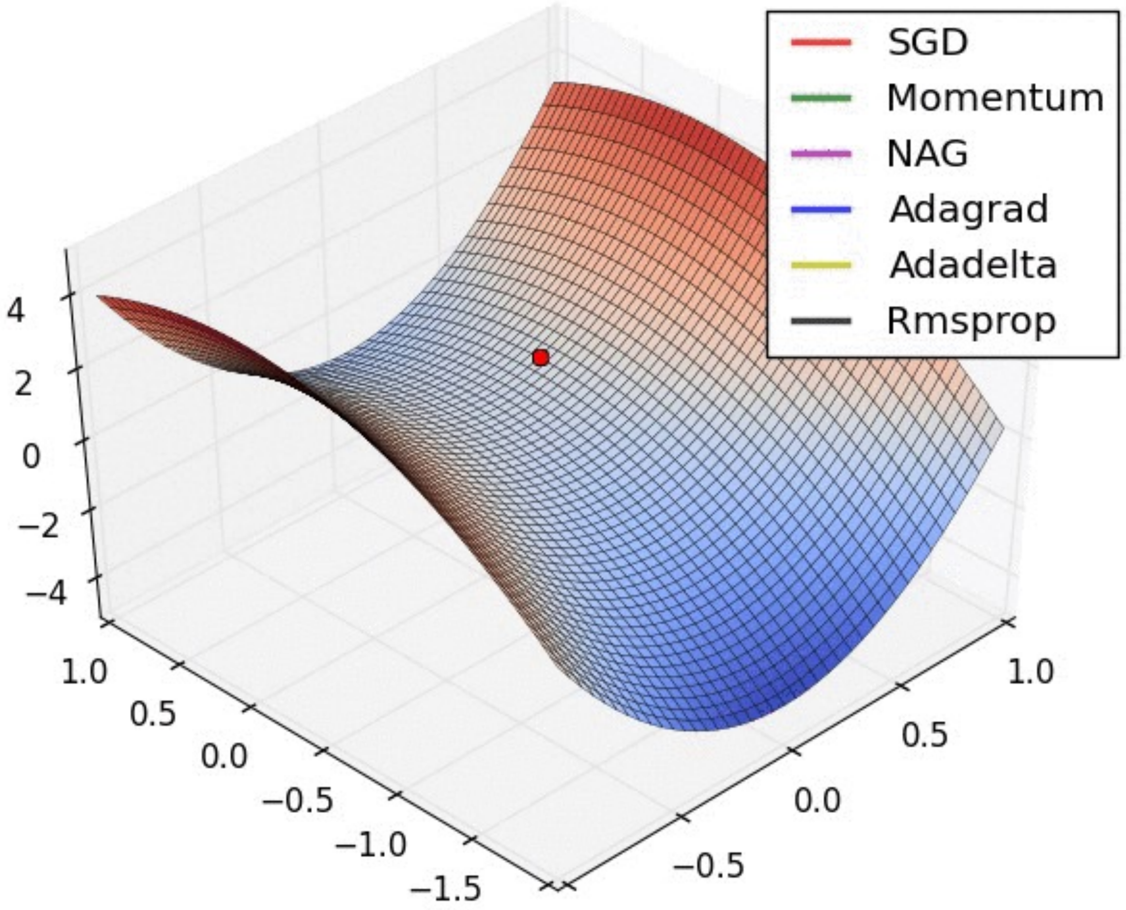
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$



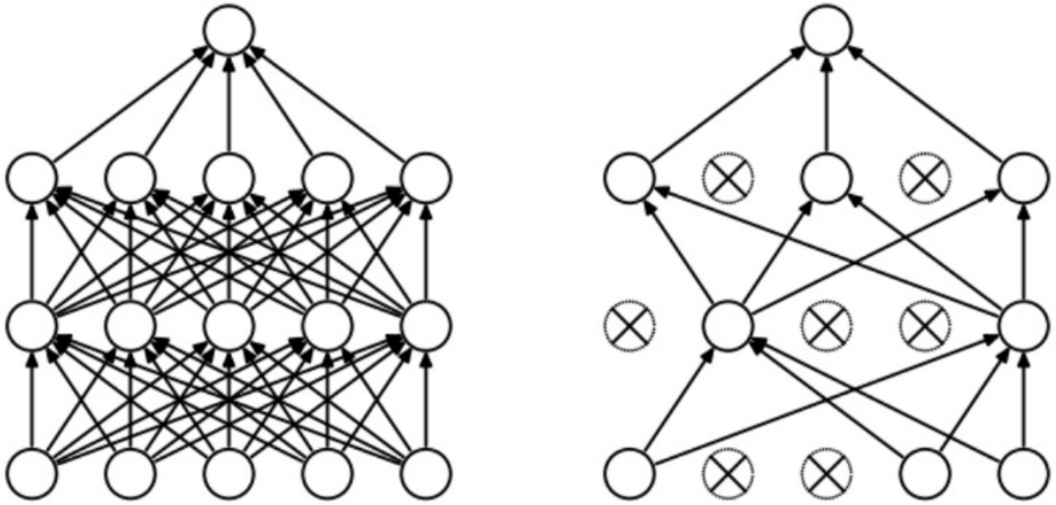


Which optimizer is the best?

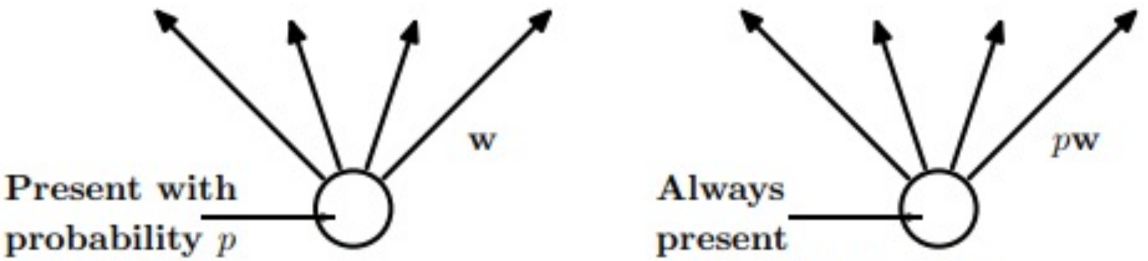
Regularization

Dropout: accuracy in the absence of certain information

- Prevent dependence on any one (or any small combination) of neurons



(a) Standard Neural Net (b) After applying dropout.



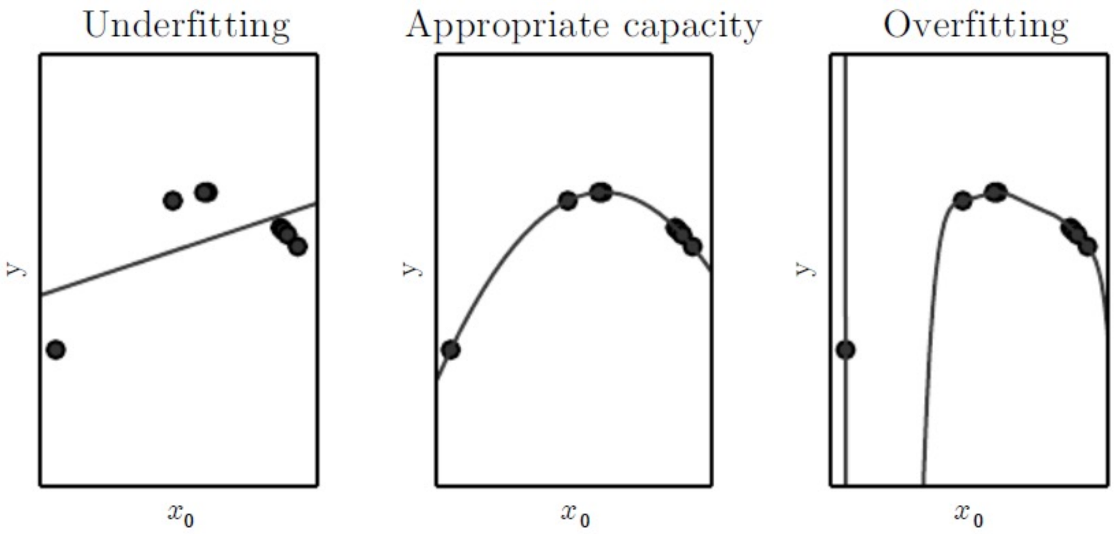
(a) At training time (b) At test time

Capacity, Overfitting and Underfitting

1) Make training error small
2) Make the gap between training and test error small

Capacity (fitting ability)

Low capacity ← $\hat{y} = \sum_{i}^{1} w_i x^i$ $\quad$ $\hat{y} = \sum_{i}^{2} w_i x^i$ $\quad$ $\hat{y} = \sum_{i}^{9} w_i x^i$ → High capacity



How training works

1. In each ***epoch***, randomly shuffle the training data
2. Partition the shuffled training data into ***mini-batches***
3. For each mini-batch, apply a single step of **gradient descent**
    - **Gradients** are calculated via ***backpropagation*** (the next topic)
4. Train for multiple epochs

Debugging a neural network

- What can we do?
  - Should we change the learning rate?
  - Should we initialize differently?
  - Do we need more training data?
  - Should we change the architecture?
  - Should we run for more epochs?
  - Are the features relevant for the problem (i.e. is the Bayes error rate reasonable)?
- Debugging is an art
  - We’ll develop good heuristics for choosing good architectures and hyper parameters