POLITECNICO
MILANO 1863

Deep Learning: Theory, Techniques & Applications

- Neural Network Training: Overfitting -

Prof. Matteo Matteucci – *matteo.matteucci@polimi.it*

Department of Electronics, Information and Bioengineering
Artificial Intelligence and Robotics Lab - Politecnico di Milano

Neural Networks are Universal Approximators

"A single hidden layer feedforward neural network with S shaped activation functions can approximate any measurable function to any desired degree of accuracy on a compact set "

Universal approximation theorem (Kurt Hornik, 1991)

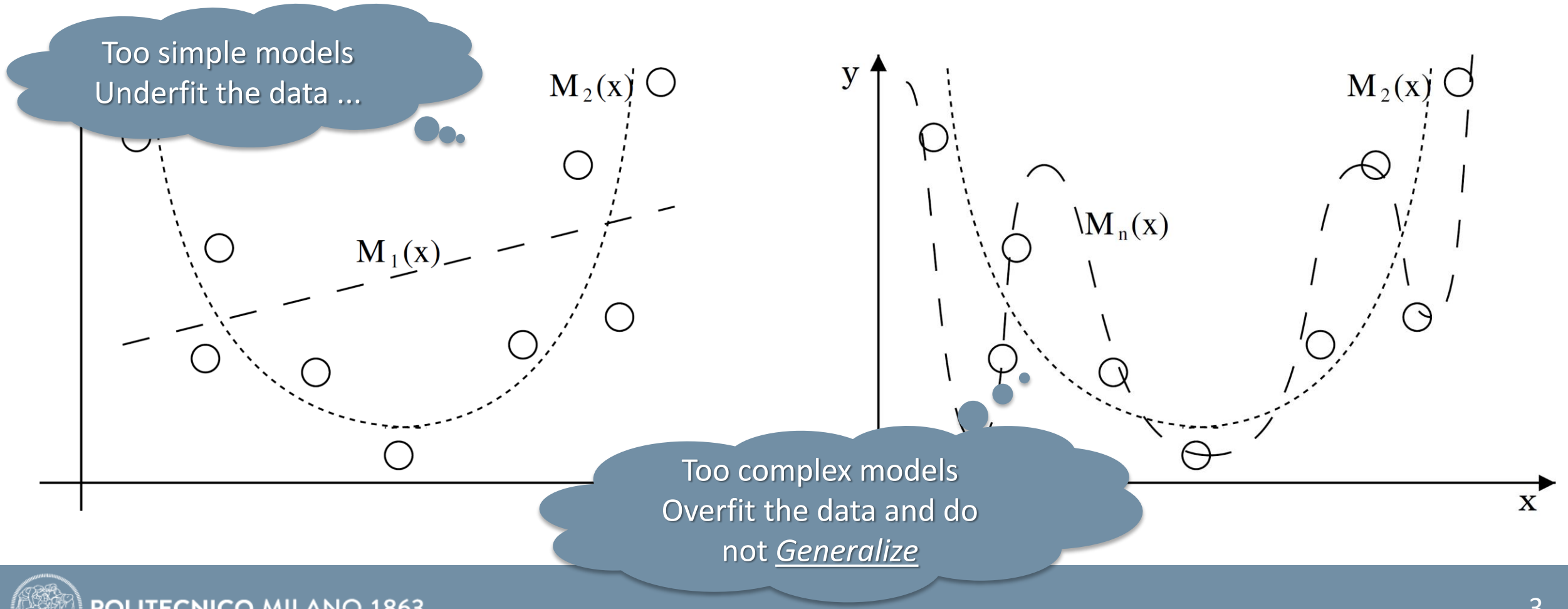
Regardless of what function we are learning, a single layer can do it …

- … but it doesn't mean we can find the necessary weights!
- … but an exponential number of hidden units may be required
- … but it might be useless in practice if it does not generalize!

"Entia non sunt multiplicanda praeter necessitatem"
William of Ockham (c 1285 – 1349)

Model Complexity

Inductive Hypothesis: A solution approximating the target function over a sufficiently large set of training examples will also approximate it over unobserved examples



How to Measure Generalization?

Training error/loss is not a good indicator of performance on future data:

- The classifier has been learned from the very same training data, any estimate based on that data will be optimistic
- New data will probably not be exactly the same as training data
- You can find patterns even in random data

We need to test on an independent new test set

- Someone provides you a new dataset
- Split the data and hide some of them for later evaluation
- Perform random subsampling (with replacement) of the dataset

In classification you should preserve class distribution, i.e., stratified sampling!

Cross-validation Variations

Cross-validation uses training data itselves to estimate the error on new data

- When enough data available use an hold out set and perform validation

FIGURE 5.1. *A schematic display of the validation set approach. A set of n observations are randomly split into a training set (shown in blue, containing observations 7, 22, and 13, among others) and a validation set (shown in beige, and containing observation 91, among others). The statistical learning method is fit on the training set, and its performance is evaluated on the validation set.*

Cross-validation Variations

Cross-validation uses training data itselves to estimate the error on new data

- When enough data available use an hold out set and perform validation
- When not too many data available use leave-one-out cross-validation (LOOCV)

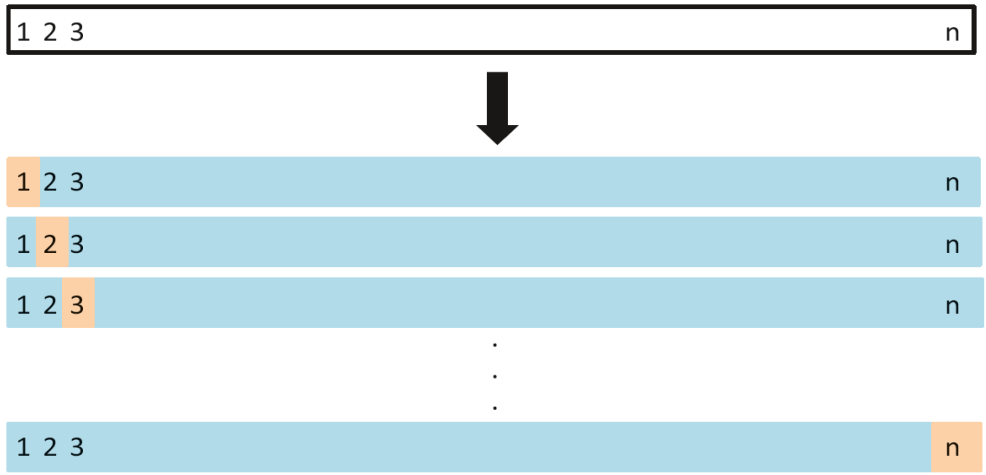


FIGURE 5.3. *A schematic display of LOOCV. A set of n data points is repeatedly split into a training set (shown in blue) containing all but one observation, and a validation set that contains only that observation (shown in beige). The test error is then estimated by averaging the n resulting MSE's. The first training set contains all but observation 1, the second training set contains all but observation 2, and so forth.*

Cross-validation Variations

Cross-validation uses training data itselves to estimate the error on new data

- When enough data available use an hold out set and perform validation
- When not too many data available use leave-one-out cross-validation (LOOCV)
- Use k-fold cross-validation for a good trade-off (sometime better than LOOCV)

$$\hat{e}_k = \frac{1}{N} \sum_{n_k \in N_k} E(x_{n_k}|w)$$

$$\hat{E} = \frac{1}{K} \sum_{k}^{K} \hat{e}_k$$

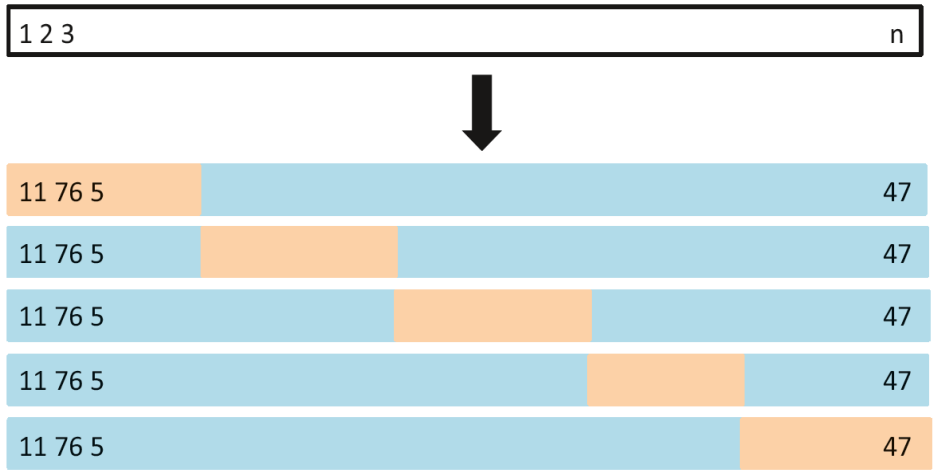


FIGURE 5.5. *A schematic display of 5-fold CV. A set of n observations is randomly split into five non-overlapping groups. Each of these fifths validation set (shown in beige), and the remainder as a trainin blue). The test error is estimated by averaging the five resulting M*

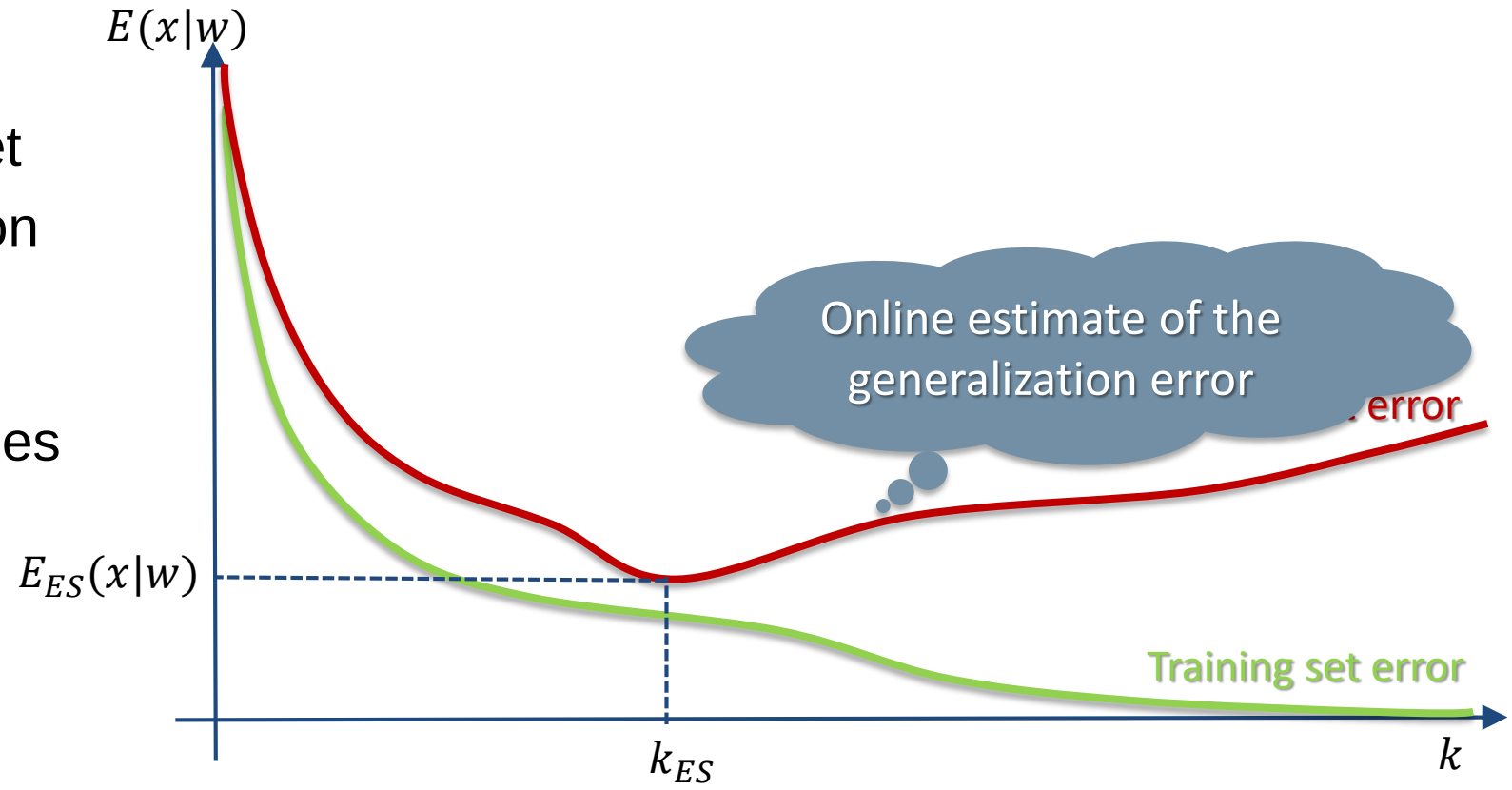
What do I do with all these models?

Note on Ensemble Methods

Early Stopping: Limiting Overfitting by Cross-validation

Overfitting networks show a monotone *training error* trend (on average with SGD) as the number of gradient descent iterations k, but they lose generalization at some point ...

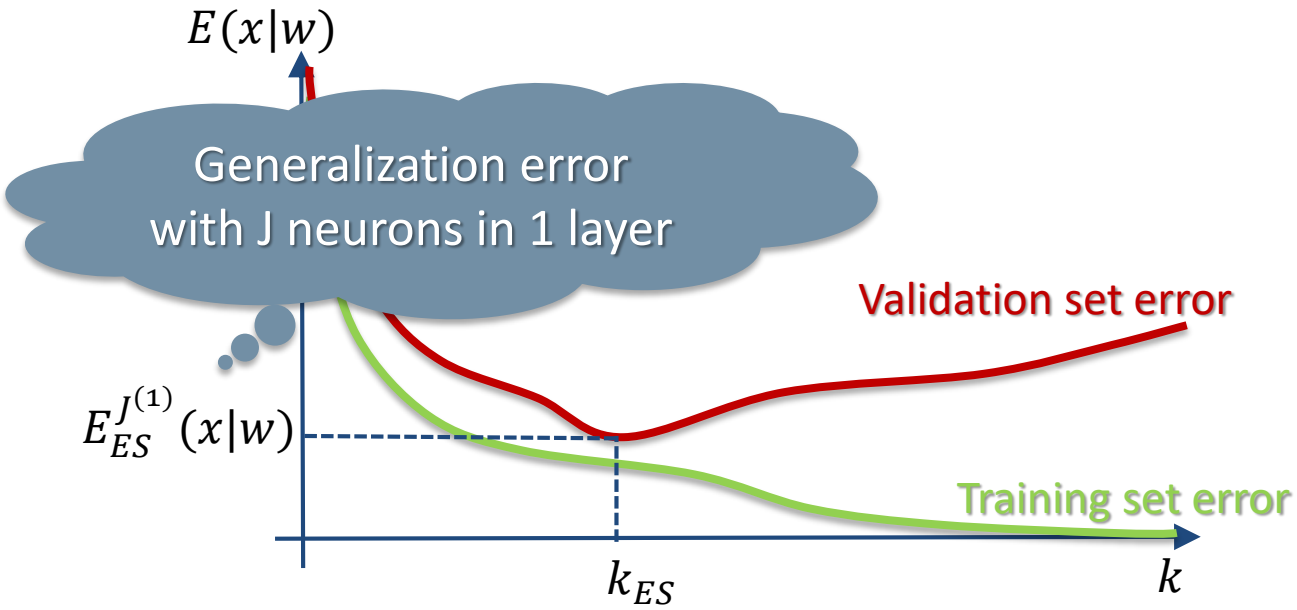
- Hold out some data
- Train on the training set
- Perform cross-validation on the hold out set
- Stop the train when validation error increases

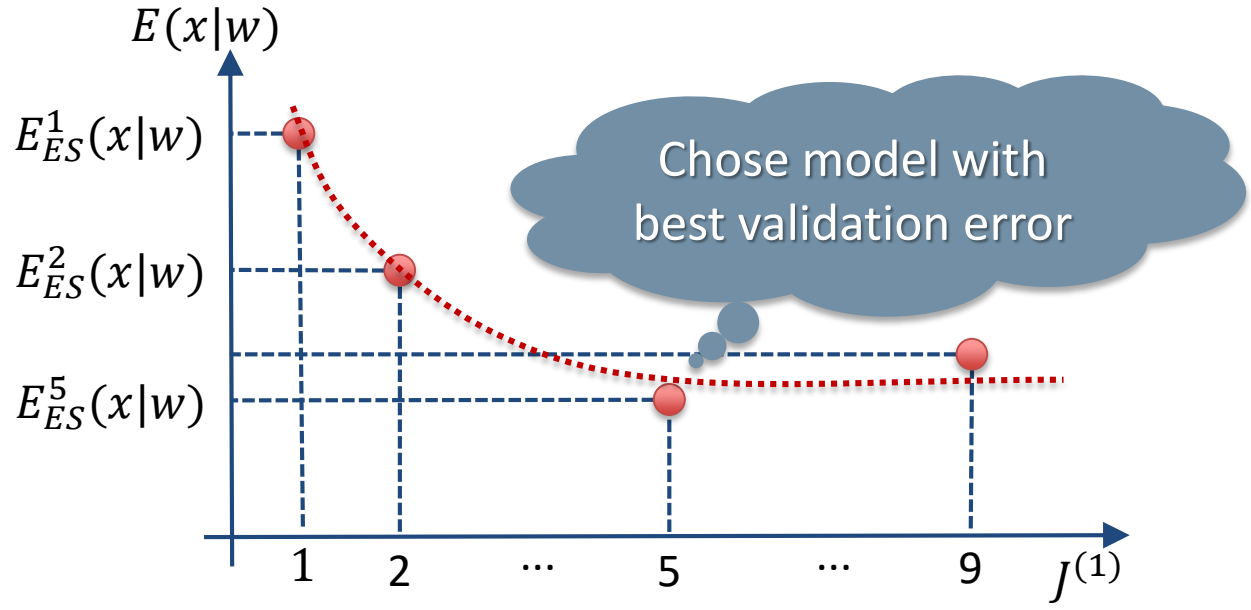


Cross-validation and Hyperparameters Tuninig

Model selection and evaluation happens at different levels:

- Parameters level, i.e, when we learn the weights w for a neural network
- Hyperparameters level, i.e., when we chose the number of layers L or the number of hidden neurons $J^{(l)}$ or a give layer
- Meta-learning, i.e., when we learn from data a model to chose hyperparameters





Weight Decay: Limiting Overfitting by Weights Regularization

Regularization is about constraining the model «freedom», based on a-priori assumption on the model, to reduce overfitting.

So far we have maximized the data likelihood:

Maximum Likelihood

$$w_{MLE} = argmax_w \, P(D|w)$$

We can reduce model «freedom» by using a Bayesian ap

Make assumption on parameters (a-priori) distribution

Maximum A-Posteriori

$$w_{MAP} = argmax_w \, P(w|D)$$
$$= argmax_w \, P(D|w) \cdot P(w)$$

Small weights have been observed to improve generalization of neural networks:

$$P(w) \sim N(0, \sigma_w^2)$$

Weight Decay: Limiting Overfitting by Weights Regularization

$$\hat{w} = argmax_w\, P(w|D) = argmax_w\, P(D|w)\, P(w)$$

$$= argmax_w \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(t_n - g(x_n|w)\right)^2}{2\sigma^2}} \prod_{q=1}^{Q} \frac{1}{\sqrt{2\pi}\sigma_w} e^{-\frac{\left(w_q\right)^2}{2\sigma_w^2}}$$

$$= argmin_w \sum_{n=1}^{N} \frac{\left(t_n - g(x_n|w)\right)^2}{2\sigma^2} + \sum_{q=1}^{Q} \frac{\left(w_q\right)^2}{2\sigma_w^2}$$

$$= argmin_w \underbrace{\sum_{n=1}^{N} \left(t_n - g(x_n|w)\right)^2}_{\text{Fitting}} + \underbrace{\gamma \sum_{q=1}^{Q} \left(w_q\right)^2}_{\text{Regularization}}$$

Here it comes another loss function!!!

Recall Cross-validation and Hyperparameters Tuninig

You can use cross-validaton to select the proper γ:

- Split data in training and validation sets
- Minimize for different values of γ

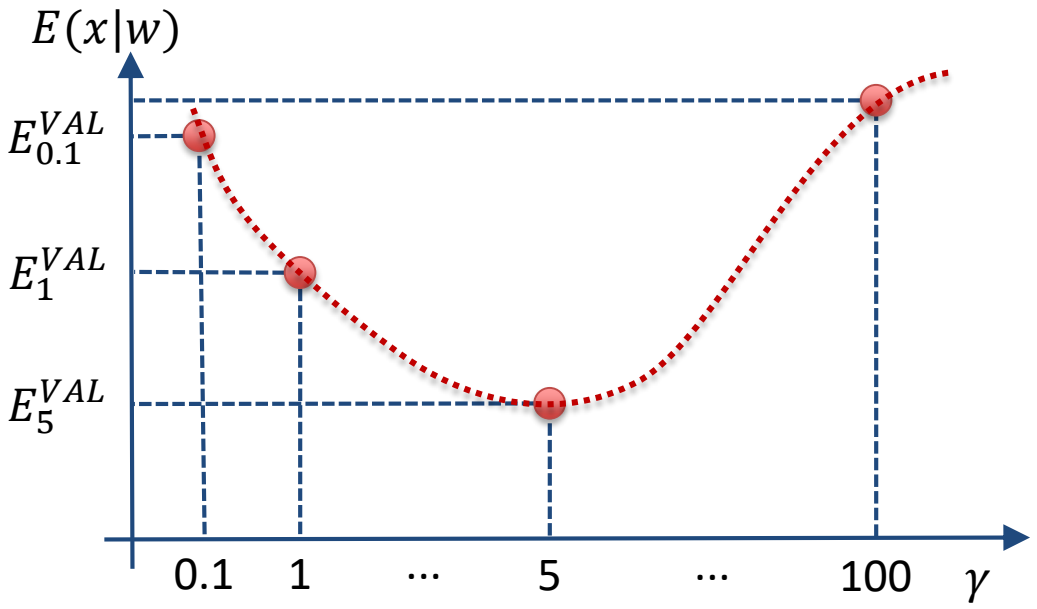
$$E_{\gamma}^{TRAIN} = \sum_{n=1}^{N_{TRAIN}} \left(t_n - g(x_n|w)\right)^2 + \gamma \sum_{q=1}^{Q} \left(w_q\right)^2$$

- Evaluate the model

$$E_{\gamma}^{VAL} = \sum_{n=1}^{N_{VAL}} \left(t_n - g(x_n|w)\right)^2$$

- Chose the γ^* with the best validation error
- Put back all data together and minimize

$$E_{\gamma^*} = \sum_{n=1}^{N} \left(t_n - g(x_n|w)\right)^2 + \gamma^* \sum_{q=1}^{Q} \left(w_q\right)^2$$

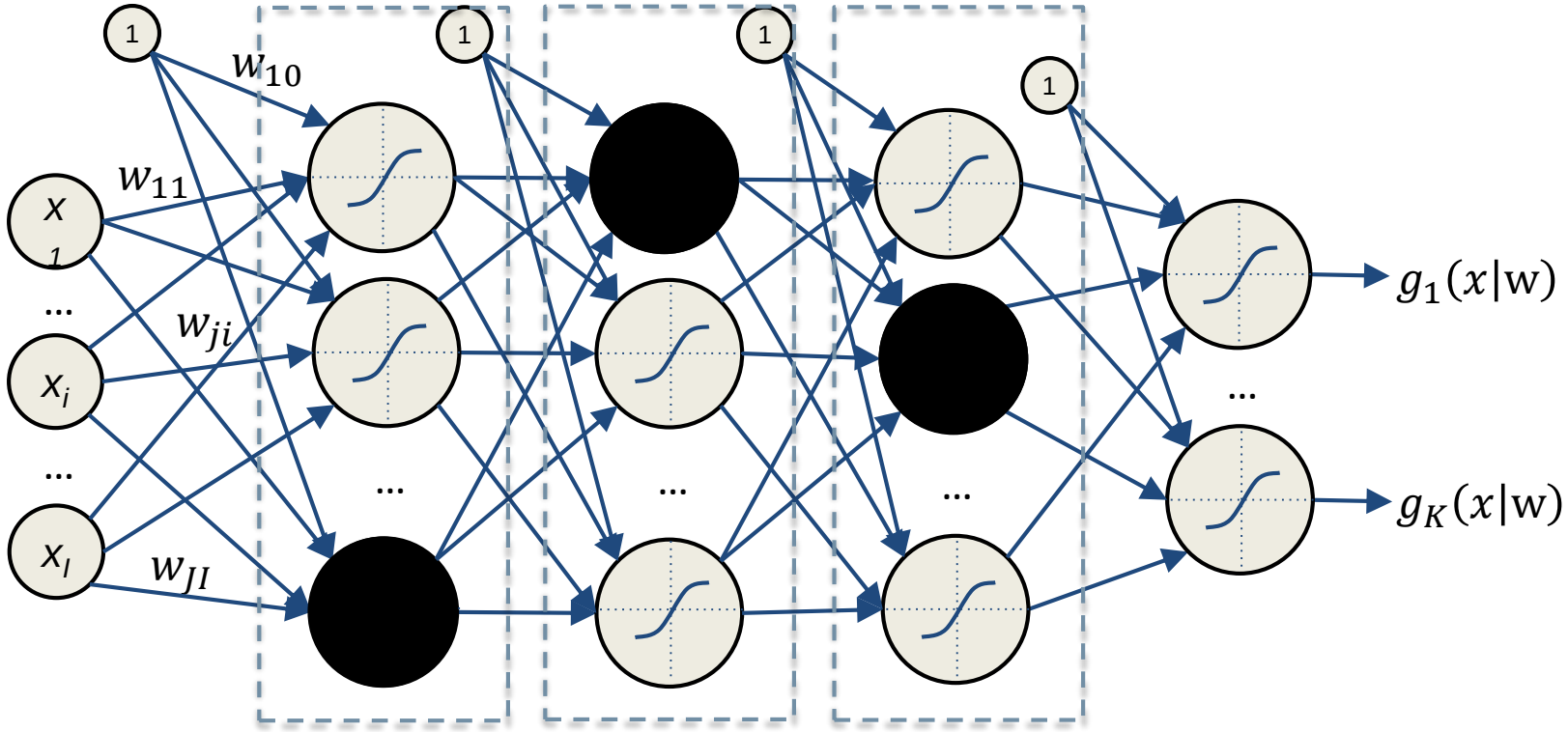


Chose $\gamma^* = 5$ with best validation error

Dropout: Limiting Overfitting by Stochastic Regularization

By turning off randomly some neurons we force them to learn an independent feature preventing hidden units to rely on other units (co-adaptation):

- Each hidden unit is set to zero with $p_j^{(l)}$ probability, e.g., $p_j^{(l)} = 0.3$



$$m^{(l)} = [m_1^{(l)}, \dots m_{J^{(l)}}^{(l)}]$$

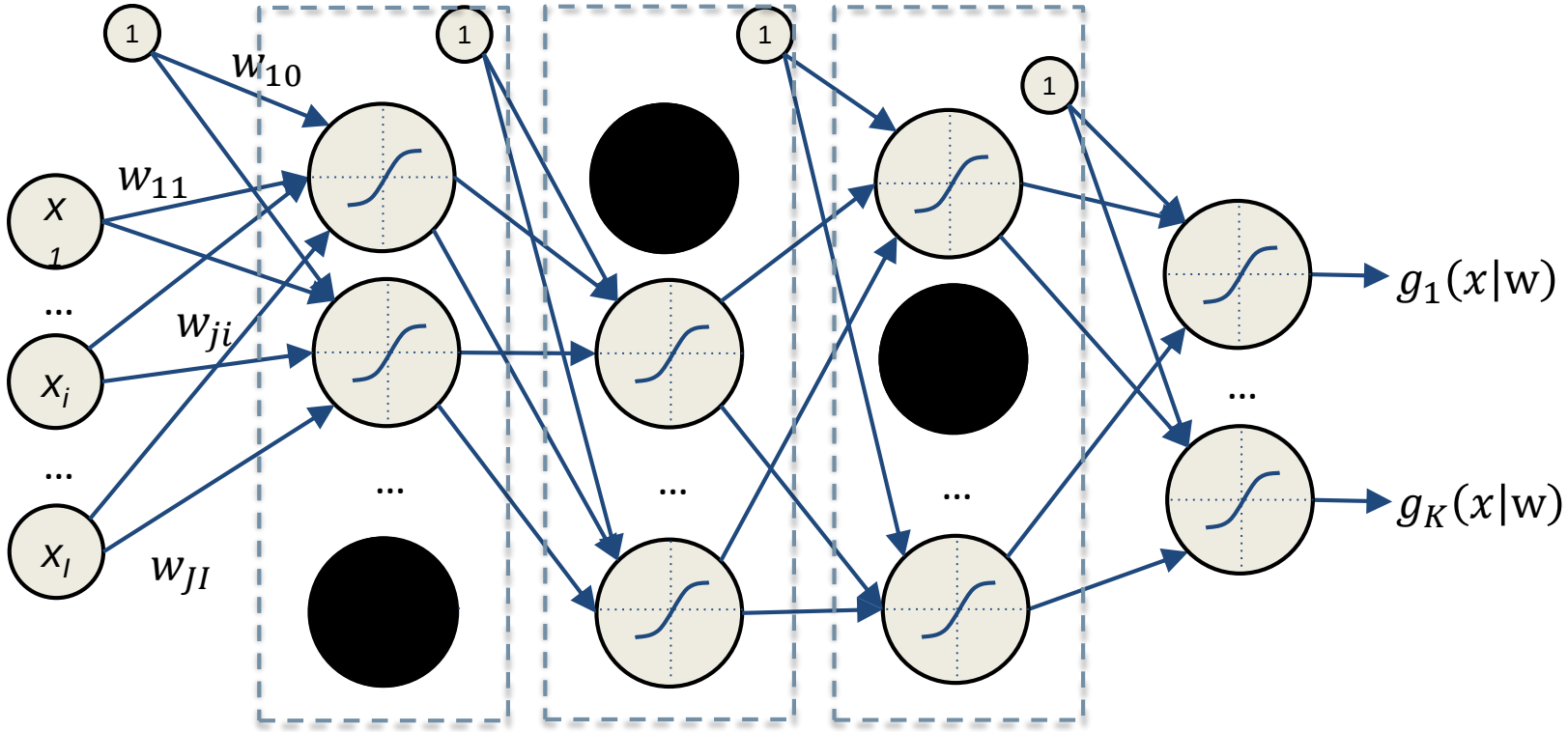
$$m_j^{(l)} \sim Be\left(p_j^{(l)}\right)$$

$$h^{(l)}(W^{(l)} h^{(l-1)} \odot m^{(l)})$$

Dropout: Limiting Overfitting by Stochastic Regularization

By turning off randomly some neurons we force them to learn an independent feature preventing hidden units to rely on other units (co-adaptation):

- Each hidden unit is set to zero with $p_j^{(l)}$ probability, e.g., $p_j^{(l)} = 0.3$



$$m^{(l)} = [m_1^{(l)}, \dots m_{J^{(l)}}^{(l)}]$$

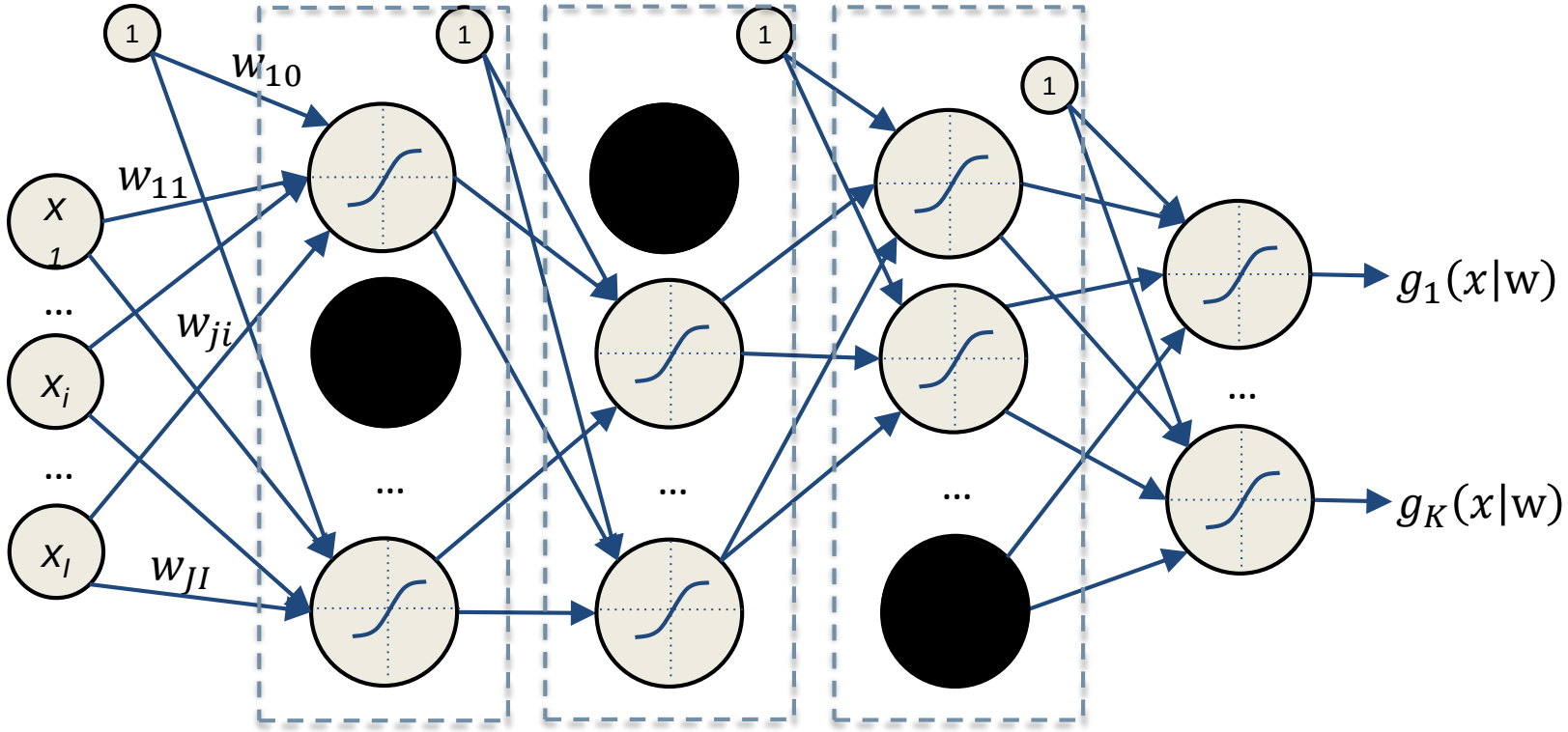
$$m_j^{(l)} \sim Be\left(p_j^{(l)}\right)$$

$$h^{(l)}(W^{(l)} h^{(l-1)} \odot m^{(l)})$$

Dropout: Limiting Overfitting by Stochastic Regularization

By turning off randomly some neurons we force them to learn an independent feature preventing hidden units to rely on other units (co-adaptation):

- Each hidden unit is set to zero with $p_j^{(l)}$ probability, e.g., $p_j^{(l)} = 0.3$



$$m^{(l)} = [m_1^{(l)}, \dots m_{J^{(l)}}^{(l)}]$$

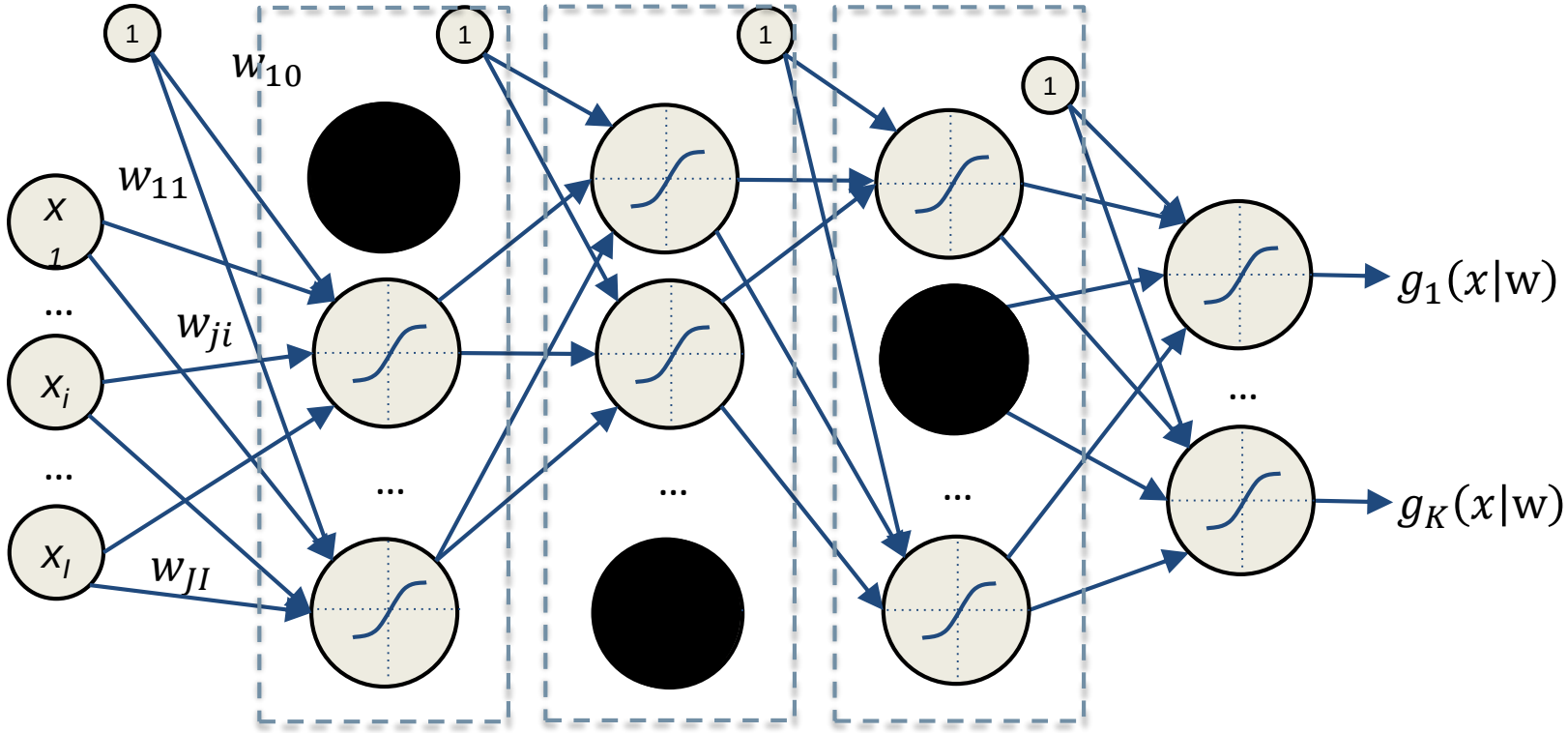
$$m_j^{(l)} \sim Be\left(p_j^{(l)}\right)$$

$$h^{(l)}(W^{(l)} h^{(l-1)} \odot m^{(l)})$$

Dropout: Limiting Overfitting by Stochastic Regularization

By turning off randomly some neurons we force them to learn an independent feature preventing hidden units to rely on other units (co-adaptation):

- Each hidden unit is set to zero with $p_j^{(l)}$ probability, e.g., $p_j^{(l)} = 0.3$



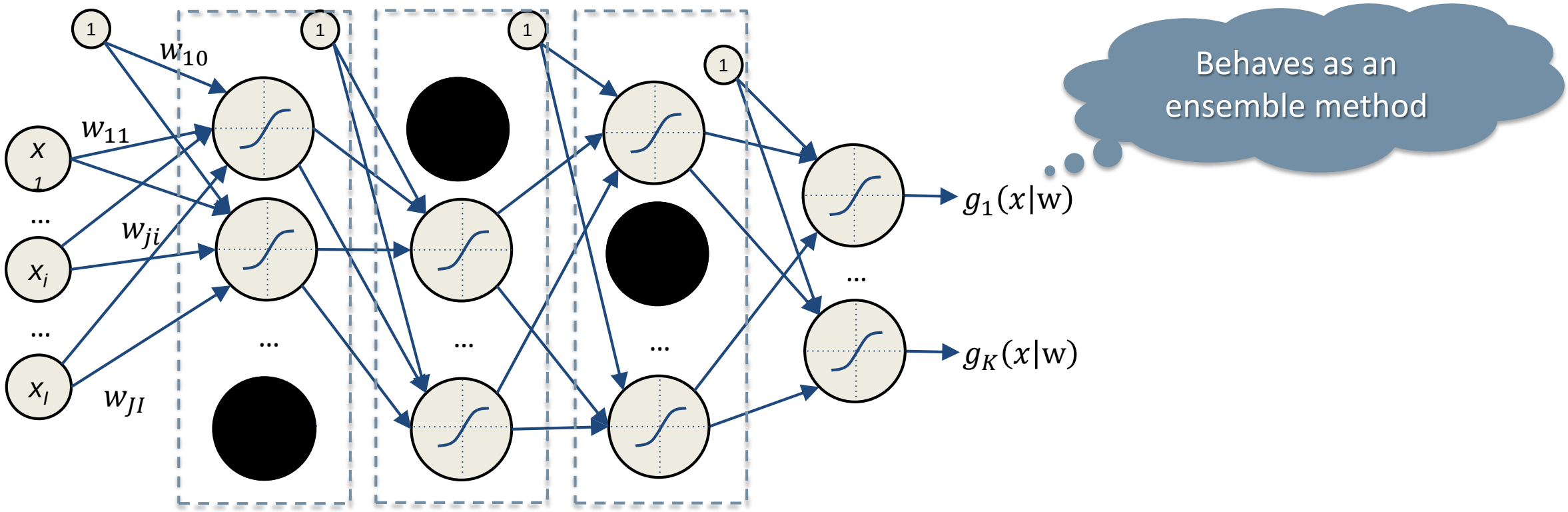
$$m^{(l)} = [m_1^{(l)}, \dots m_{J^{(l)}}^{(l)}]$$

$$m_j^{(l)} \sim Be\left(p_j^{(l)}\right)$$

$$h^{(l)}(W^{(l)} h^{(l-1)} \odot m^{(l)})$$

Dropout: Limiting Overfitting by Stochastic Regularization

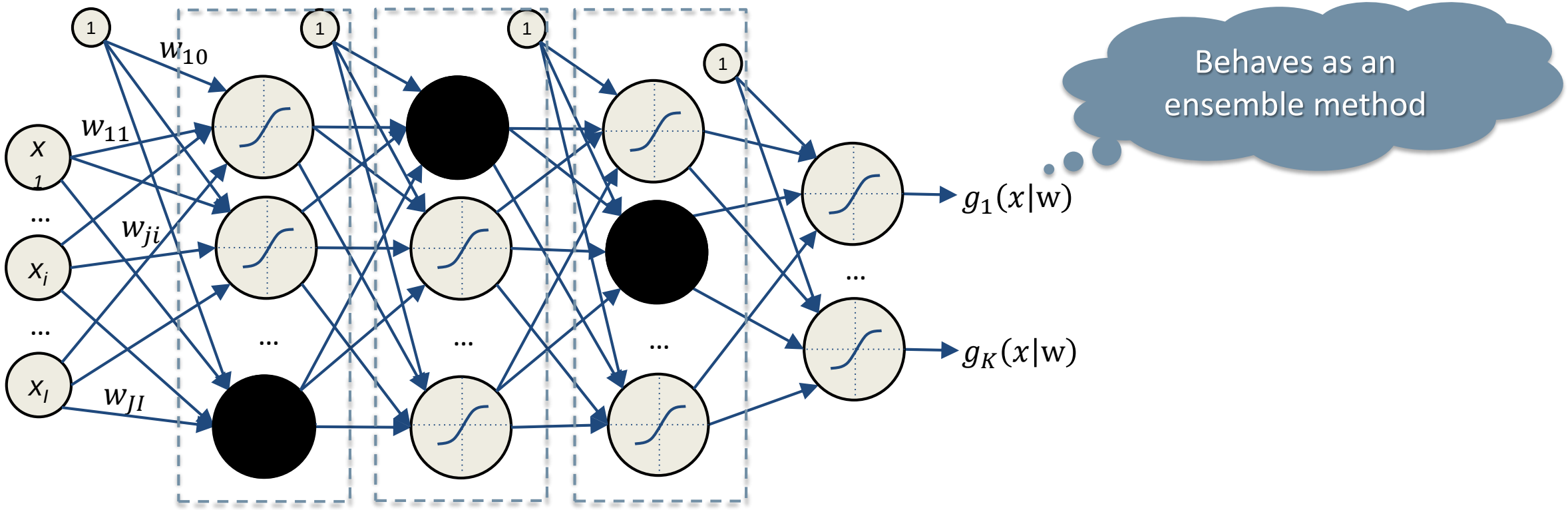
In Dropout we train a number of *weaker classifiers*, on the different mini- batch and then at test time we use them by averaging the responses of all ensemble members.



Dropout: Limiting Overfitting by Stochastic Regularization

In Dropout we train a number of *weaker classifiers*, on the different mini- batch and then at test time we use them by averaging the responses of all ensemble members.

At testing time we remove the masks and we average the output (by weight scaling)



POLITECNICO
MILANO 1863

Deep Learning: Theory, Techniques & Applications

- Neural Network Training: Tips & Tricks-

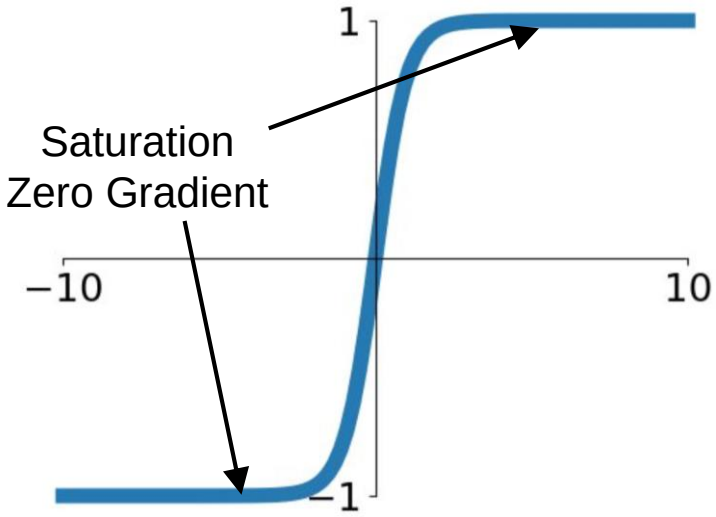
Prof. Matteo Matteucci – *matteo.matteucci@polimi.it*

Department of Electronics, Information and Bioengineering
Artificial Intelligence and Robotics Lab - Politecnico di Milano

Better Activation Functions

Activation functions such as Sigmoid or Tanh saturate

- Gradient is close to zero
- Backprop. requires gradient multiplications
- Gradient faraway from the output vanishes
- Learning in deep networks does not happen



$$\frac{\partial E(w_{ji}^{(1)}))}{\partial w_{ji}^{(1)}} = -2\sum_{n}^{N}\left(t_n - g_1(x_n, w)\right) \cdot g_1'\,(x_n, w) \cdot w_{1j}^{(2)} \cdot h_j'\left(\sum_{j=0}^{J} w_{ji}^{(1)} \cdot x_{i,n}\right) \cdot x_i$$

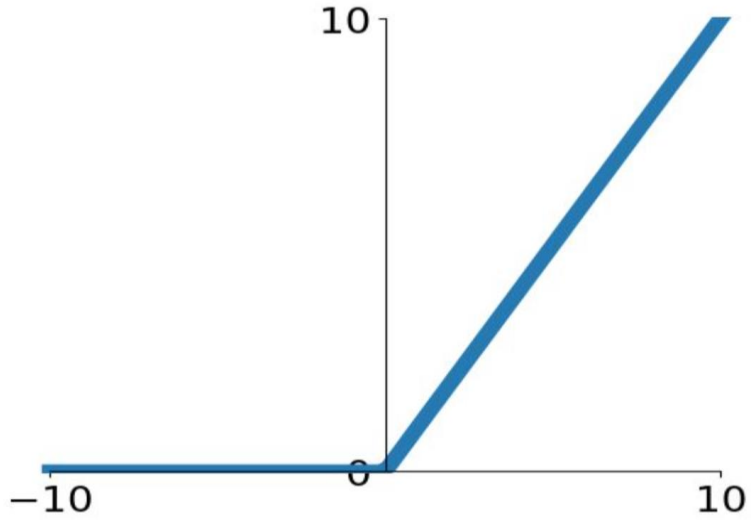
This is a well known problem in Recurrent Neural Networks, but it affects also deep networks, and it has hindered neural network training since ever ...

Rectified Linear Unit

The ReLU activation function has been introduced

$$g(a) = ReLu(a) = \max(0, a)$$

$$g'^{(a)} = 1_{a>0}$$



It has several advantages:

- Faster SGD Convergence (6x w.r.t sigmoid/tanh)
- Sparse activation (only part of hidden units are activated)
- Efficient gradient propagation (no vanishing or exploding gradient problems), and Efficient computation (just thresholding at zero)
- Scale-invariant:

$$\max(0, ax) = a \max(0, x)$$

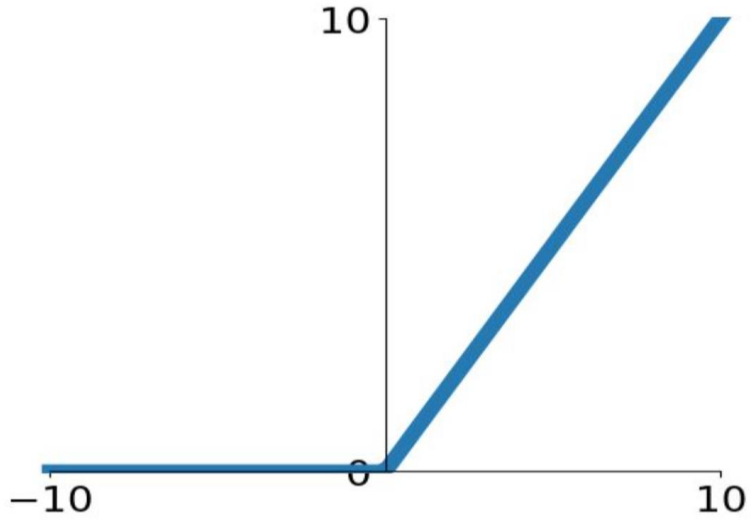
22

Rectified Linear Unit

The ReLU activation function has been introduced

$$g(a) = ReLu(a) = \max(0, a)$$

$$g'^{(a)} = 1_{a>0}$$



It has potential sisadvantages:

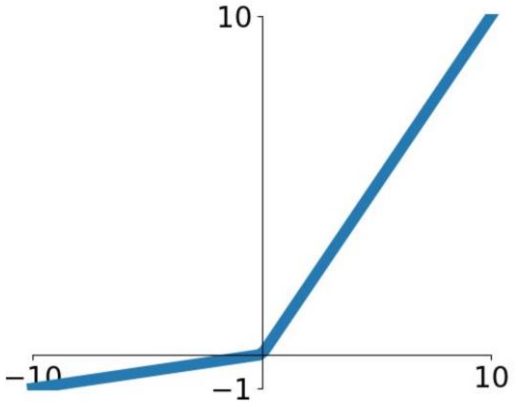
- Non-differentiable at zero: however it is differentiable anywhere else
- Non-zero centered output
- Unbounded: Could potentially blow up
- Dying Neurons: ReLU neurons can sometimes be pushed into st… which they become inactive for essentially all inputs. No gradients flow backward through the neuron, and so the neuron becomes stuck and "dies".

Decreased model capacity, it happens with high learning rates

Rectified Linear Unit (Variants)

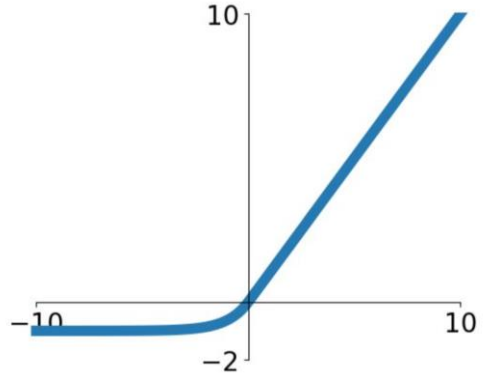
Leaky ReLU: fix for the “dying ReLU” problem

$$f(x) = \begin{cases} x & if\ x \geq 0 \\ 0.01x & otherwise \end{cases}$$



ELU: try to make the mean activations closer to zero which speeds up learning. Alpha is tuned by hand*by hand*

$$f(x) = \begin{cases} x & if\ x \geq 0 \\ \alpha(e^x - 1) & otherwise \end{cases}$$



Weights Initialization

The final result of gradient descent is highly affected by weight initialization:

- Zeros: it does not work! All gradient would be zero, no learning will happen
- Big Numbers: bad idea, if unlucky might take very long to converge
- $w \sim N(0, \sigma^2 = 0.01)$: good for small networks, but it might be a problem for deeper neural networks

In deep netowrks:

- If weights start too small, then gradient shrinks as it passes through each layer
- If the weights in a network start too large, then gradient grows as it passes through each layer until it's too massive to be useful

Some proposal to solve this Xavier initialization or He initialization …

Xavier Initialization

Suppose we have an input x with I components and a linear neuron with random weights w. Its output is

$$h_j = w_{j1}x_1 + \cdots + w_{ji}x_I + \cdots + w_{jI}\,x_I$$

We can derive that $w_{ji}x_i$ is going to have variance

$$Var(w_{ji}x_i) = E[x_i]^2 Var(w_{ji}) + E\left[w_{ji}\right]^2 Var(x_i) + Var(w_{ji})Var(x_i)$$

Now if our inputs and weights both have mean 0, that simplifies to

$$Var(w_{ji}x_i) = Var(w_{ji})Var(x_i)$$

If we assume all w_i and x_i are i.i.d. we obtain

$$Var\left(h_j\right) = Var\left(w_{j1}x_1 + \cdots + w_{ji}x_I + \cdots + w_{jI}\,x_I\right) = nVar(w_i)Var(x_i)$$

The variance of the output is the variance of the input, but scaled by $nVar(w_i)$.

Xavier Initialization

If we want the variance of the input and the out to be the same

$$nVar(w_j) = 1$$

Linear assumption seem too much, but in practice it works!

For this reason Xavier proposes to initialize $w \sim N\left(0, \frac{1}{n_{in}}\right)$

Performing similar reasoning for the gradient Glorot & Bengio found

$$n_{out}Var(w_j) = 1$$

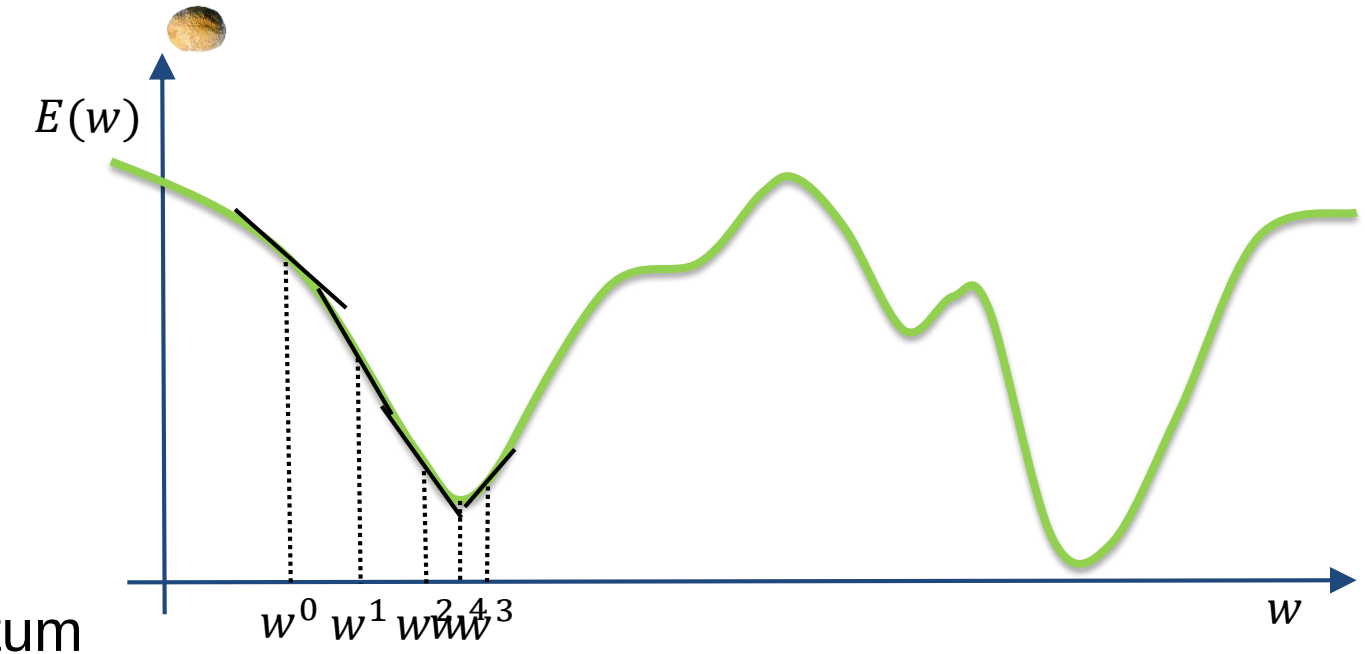
To accommodate for this and for the Xavier constraint they propose $w \sim N\left(0, \frac{2}{n_{in}+n_{out}}\right)$

More recently He proposed, for rectified linear units, $w \sim N\left(0, \frac{2}{n_{in}}\right)$

Recall about Backpropagation

Finding the weighs of a Neural Network is a non linear minimization process

$$argmin_w \; E(w) = \sum_{n=1}^{N} (t_n - g(x_n, w))^2$$



We iterate from a initial configuration

$$w^{k+1} = w^k - \eta \left. \frac{\partial E(w)}{\partial w} \right|_{w^k}$$

To avoid local minima can use momentum

$$w^{k+1} = w^k - \eta \left. \frac{\partial E(w)}{\partial w} \right|_{w^k} - \alpha \left. \frac{\partial E(w)}{\partial w} \right|_{w^{k-1}}$$

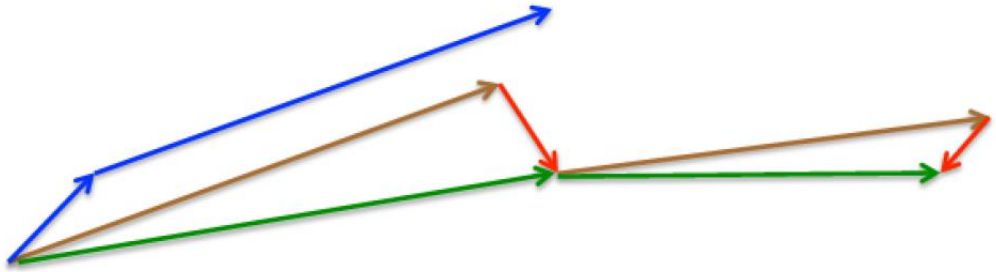
Several variations exists beside these two …

More about Gradient Descent

Nesterov Accelerated gradient: first make a jump as the momentum, then adjust

$$w^{k+\frac{1}{2}} = w^k - \alpha \left.\frac{\partial E(w)}{\partial w}\right|_{w^{k-1}}$$

$$w^{k+1} = w^k - \eta \left.\frac{\partial E(w)}{\partial w}\right|_{w^{k+\frac{1}{2}}}$$



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

Adaptive Learning Rates

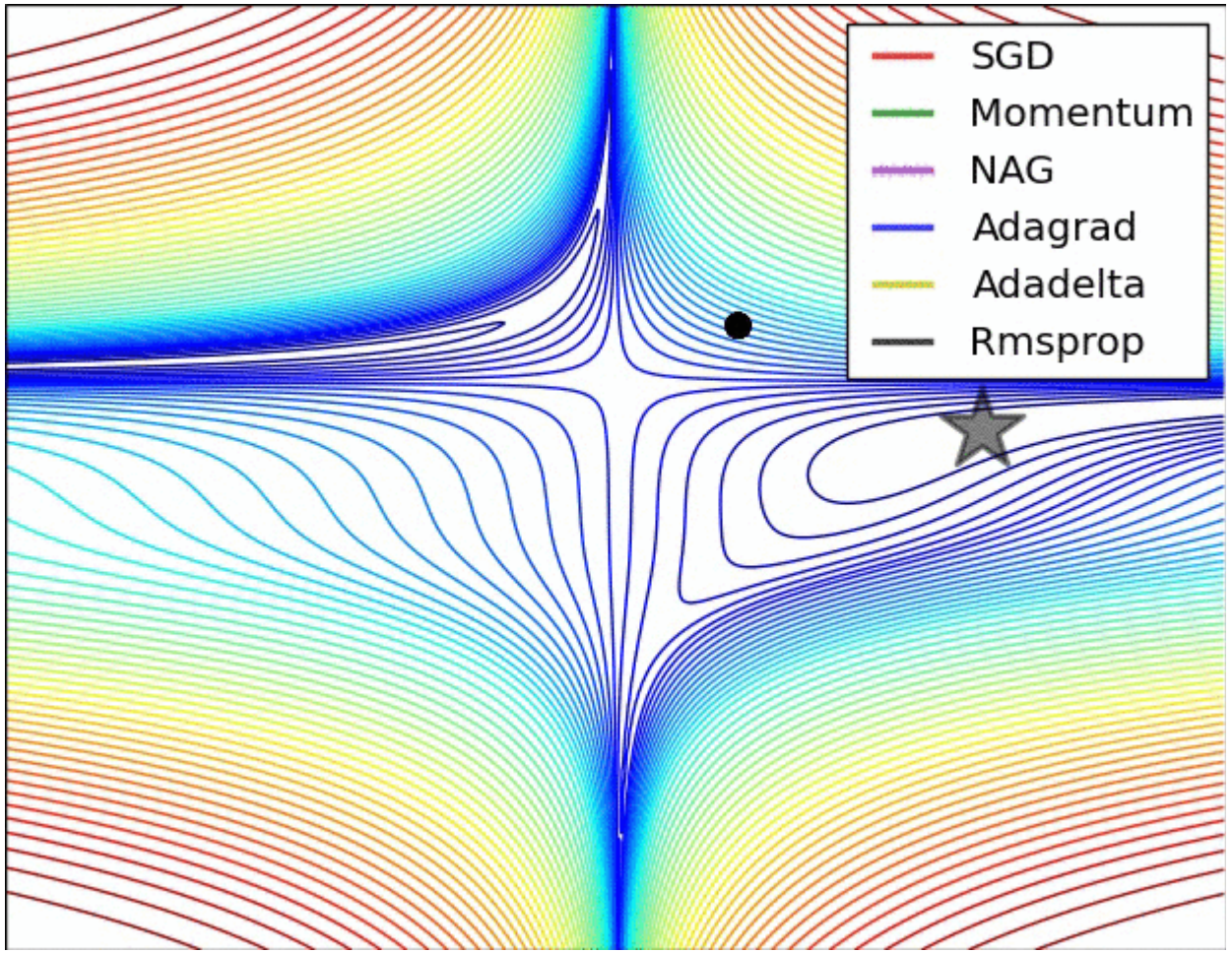
Neurons in each layer learn differently

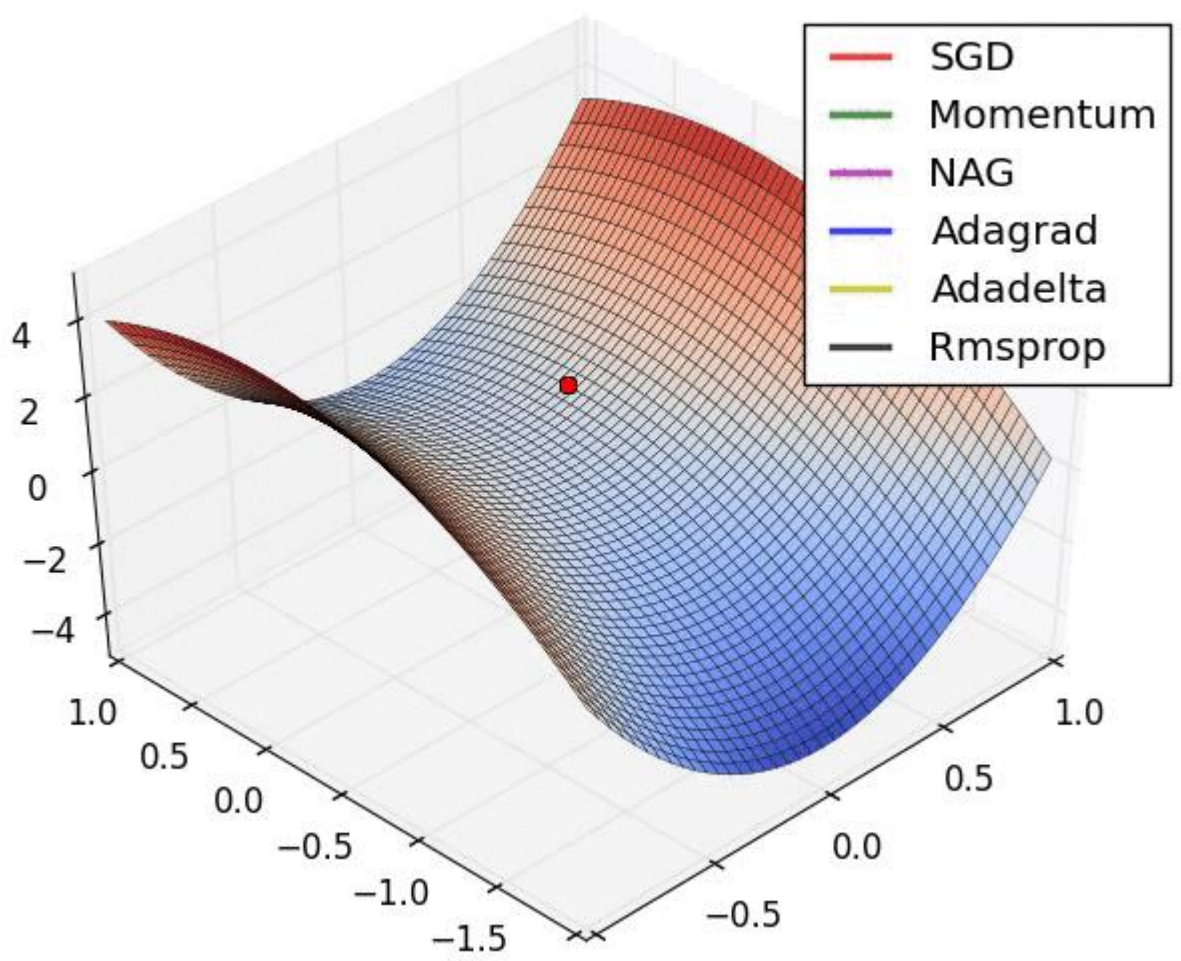
- Gradient magnitudes vary across layers
- Early layers get “vanishing gradients”
- Should ideally use separate adaptive learning rates

Several algoritm proposed:

- Resilient Propagation (Rprop – Riedmiller and Braun 1993)
- Adaptive Gradient (AdaGrad – Duchi et al. 2010)
- RMSprop (SGD + Rprop – Teileman and Hinton 2012)
- AdaDelta (Zeiler et at. 2012)
- Adam (Kingma and Ba, 2012)
- …

Learning Rate Matters





POLITECNICO
MILANO 1863

Deep Learning: Theory, Techniques & Applications

- Neural Network Training: Overfitting -

Prof. Matteo Matteucci – *matteo.matteucci@polimi.it*

Department of Electronics, Information and Bioengineering
Artificial Intelligence and Robotics Lab - Politecnico di Milano