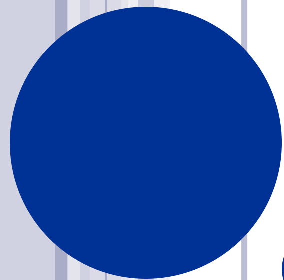
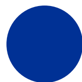
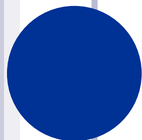
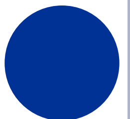




Deep Learning

Elisabetta Fersini

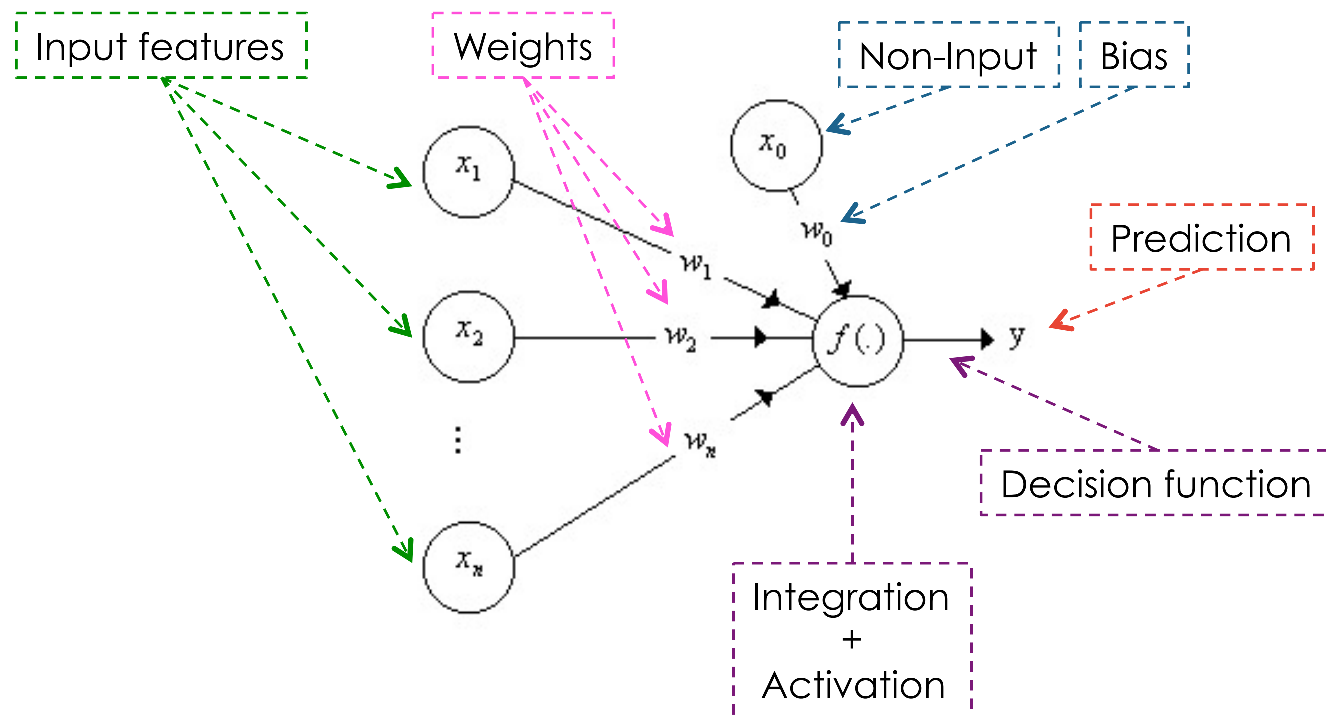
elisabetta.fersini@unimib.it

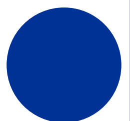
Now back to linear models…



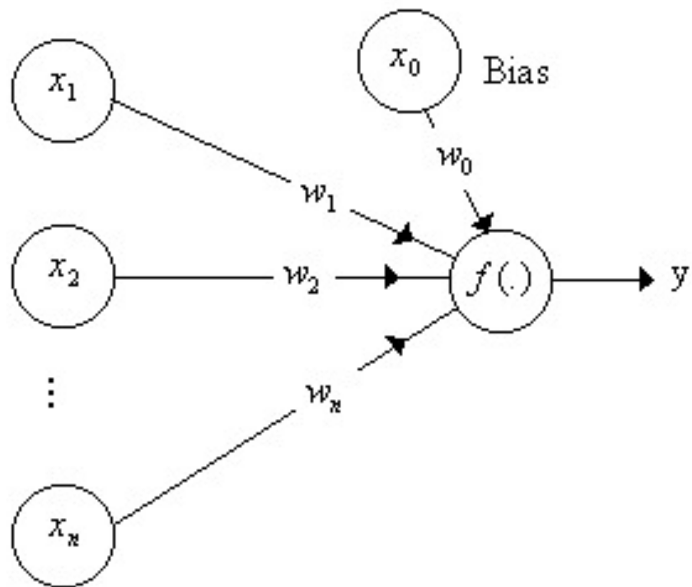
PERCEPTRON

- A perceptron takes a vector of inputs x = (x_1, x_2, . . . , x_n), weights each feature, and outputs a binary variable, "+1" or "-1", according to an activation function (i.e. depending on whether a weighted sum exceeds some pre-determined threshold).





PERCEPTRON



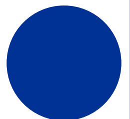
Activation

Integration

f(x,w)=sign(w_1 * x_1 + … + w_n * x_n + w_0 * x_0)

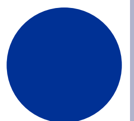
x_0=1; w_0 alters the position of the decision boundary

Remark: If w_0 is negative, then the weighted combination of inputs must produce a positive value greater than $|b|$ to push the classifier over the 0 threshold



PERCEPTRON

- This means that w_0 is (usually) a parameter to be estimated



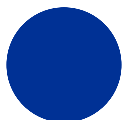
LIMITATIONS OF LINEAR MODELS

- The hypothesis class of linear (and log-linear) models is severely restricted:
  - It cannot represent the XOR function

$$\text{xor}(0,0) = 0$$
$$\text{xor}(1,0) = 1$$
$$\text{xor}(0,1) = 1$$
$$\text{xor}(1,1) = 0.$$

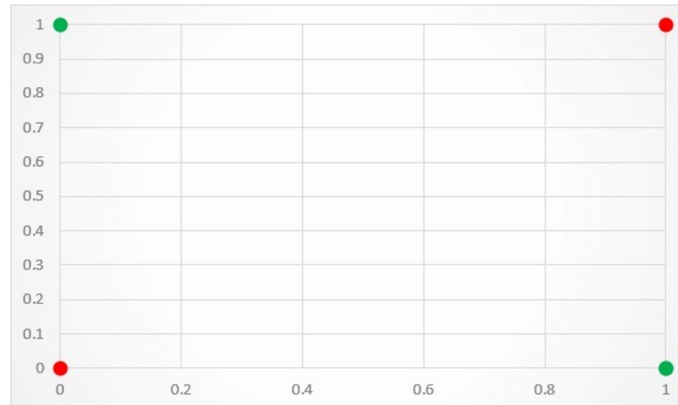
  - There is no $w \in R^2$ and $b \in R$ such that:

$$(0,0) \cdot \boldsymbol{w} + b < 0$$
$$(0,1) \cdot \boldsymbol{w} + b \geq 0$$
$$(1,0) \cdot \boldsymbol{w} + b \geq 0$$
$$(1,1) \cdot \boldsymbol{w} + b < 0.$$



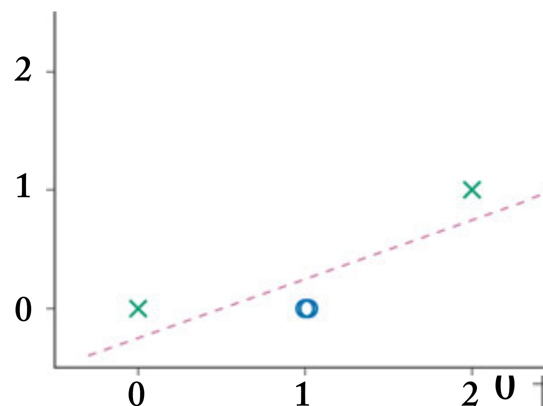
LIMITATIONS OF LINEAR MODELS

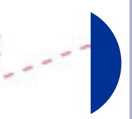
- There is no line that can separete our samples:



- However, if we transform our input using a non linear function

$$\phi(x_1, x_2) = [x_1 \times x_2, x_1 + x_2]$$



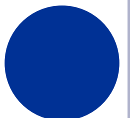


LIMITATIONS OF LINEAR MODELS

- In general, we train a linear classifier over a dataset that is not linearly separable by defining a function that will *linearize* the data

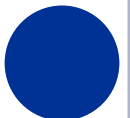
- However, in most of the cases the dimension of the feature space is much higher that the original input space
  - And we need to define a mapping function Φ

- Support Vector Machines *approach this problem by defining a set of mappings*
  - *Each of them map the data into a very high dimensional space*



LIMITATIONS OF LINEAR MODELS

- One example is the polynomial mapping $\phi(\boldsymbol{x}) = (\boldsymbol{x})^d$
- For d=2, we obtain $\phi(x_1, x_2) = (x_1 x_1, x_1 x_2, x_2 x_1, x_2 x_2)$
  - i.e. all the combinations of the two variables
- Although we are now able to train a linear classifier for the XOR problem, we have a polynomial increase in the number of parameters
- Let's assume that we have a classification problem with 784 input variables
  - With a simple polynomial mapping we will move from an input space of 784 to a feature space of 784^2=614,656



Limitations of Linear Models

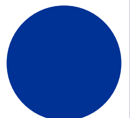
- A different approach is to define a trainable non-linear mapping function, and train it with the linear classifier

  - The mapping function can take the form of a parameterized linear model, followed by a non-linear activation function *g* that is applied to each of the output dimensions:

$$\hat{\boldsymbol{y}} = \phi(\boldsymbol{x})\boldsymbol{W} + \boldsymbol{b}$$

$$\phi(\boldsymbol{x}) = g(\boldsymbol{x}\boldsymbol{W}' + \boldsymbol{b}')$$

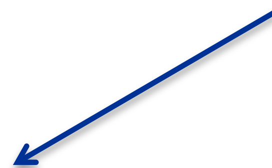
  - By taking g(x)=max(0,x) and $W' = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ and b'=(-1,0) we obtain the same results as before

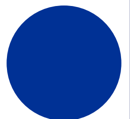


LIMITATIONS OF LINEAR MODELS

- The entire expression $g(\boldsymbol{x}\boldsymbol{W}' + \boldsymbol{b}')\boldsymbol{W} + \boldsymbol{b}$ is differentiable even if not convex.
  - we can apply any gradient-based estimation to learn simultaneously both the representation function and the linear classifier on top
  - This corresponds to derive what we can call Multi-Layer Perceptron

$$\hat{\boldsymbol{y}} = \phi(\boldsymbol{x})\boldsymbol{W} + \boldsymbol{b}$$
$$\phi(\boldsymbol{x}) = g(\boldsymbol{x}\boldsymbol{W}' + \boldsymbol{b}').$$



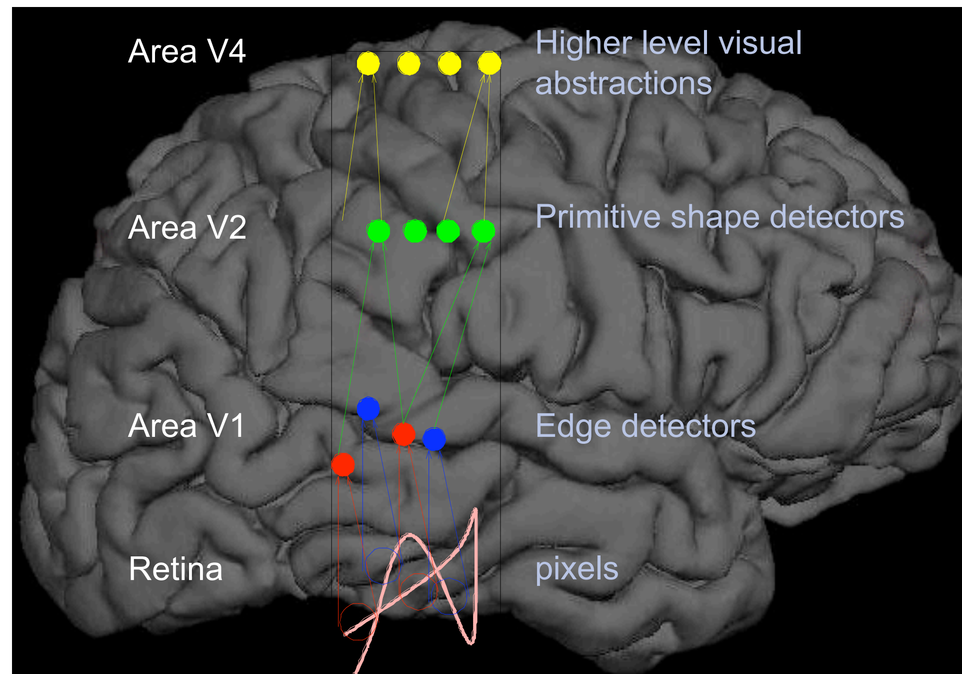


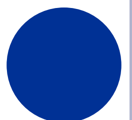
Artificial Neural Networks (ANNs)

The basics

ANNs incorporate the two fundamental components of biological neural nets:

- Neurones (nodes)
- Synapses (weights)



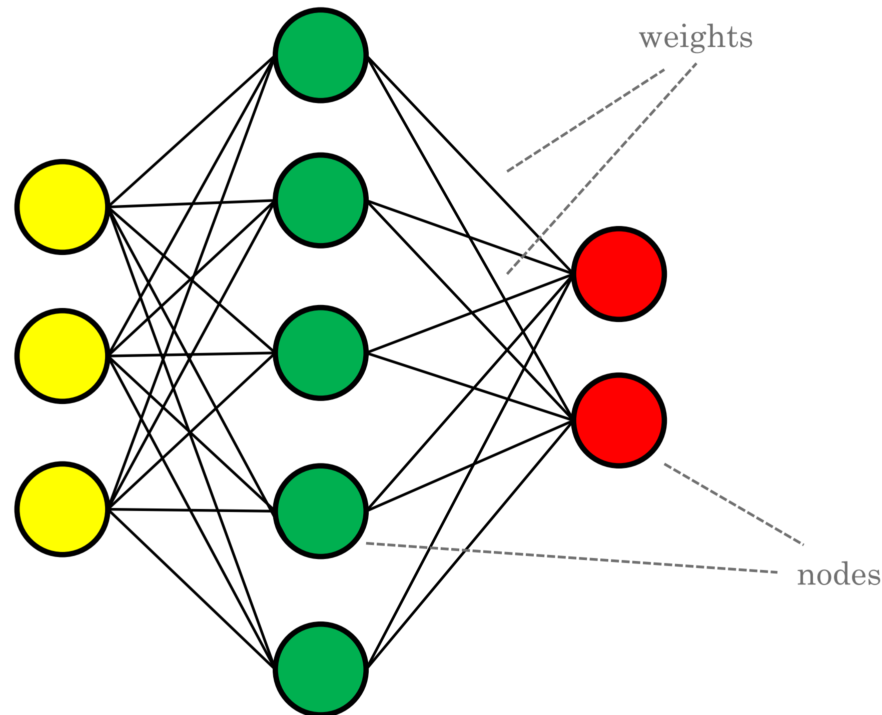


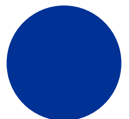
Artificial Neural Networks (ANNs)

The basics

ANNs incorporate the two fundamental components of biological neural nets:

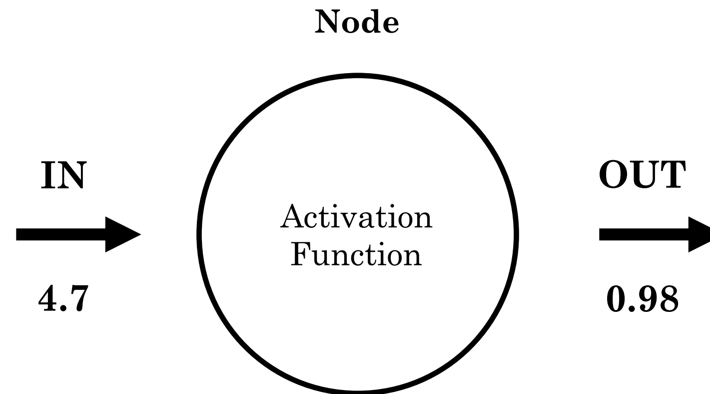
- Neurones (nodes)
- Synapses (weights)



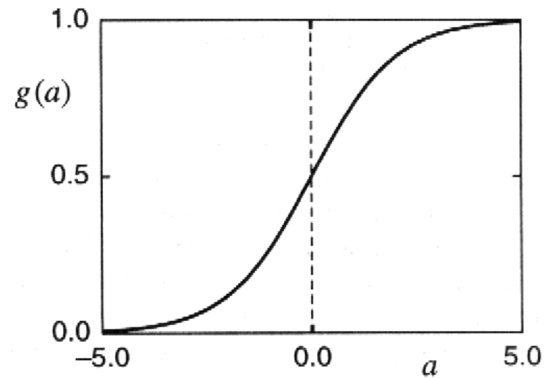


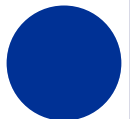
Artificial Neural Networks (ANNs)

Structure of a Node



Activation function limits node output:

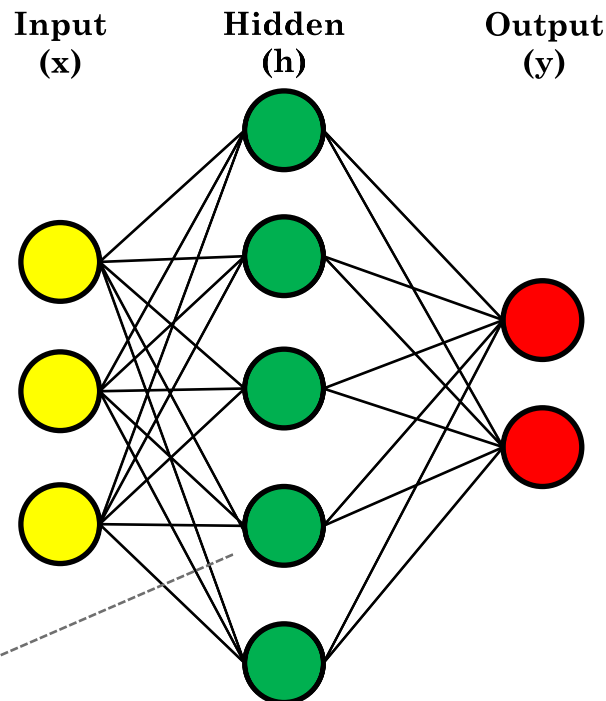




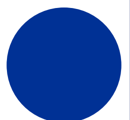
FEED-FORWARD NEURAL NETWORKS

A **feed-forward neural network** is an artificial neural network where:

- Information flow is unidirectional
- Information is distributed
- Information processing is parallel

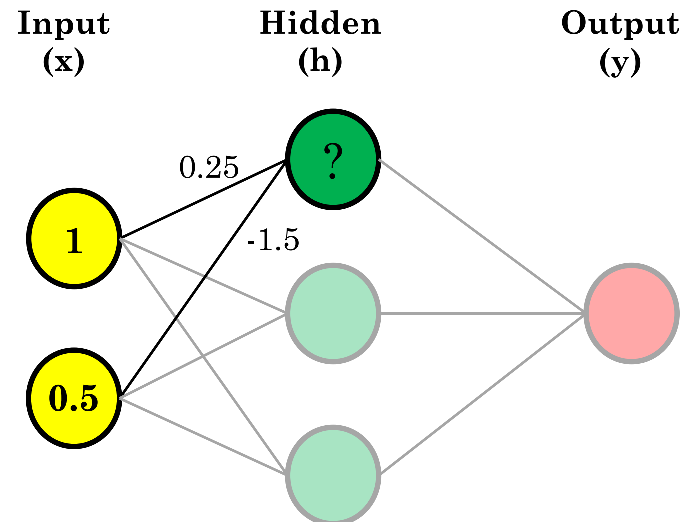


Internal representation (interpretation) of data



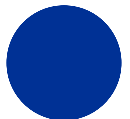
Feed-forward Neural Networks
Forward Propagation

Feeding the data through the network:



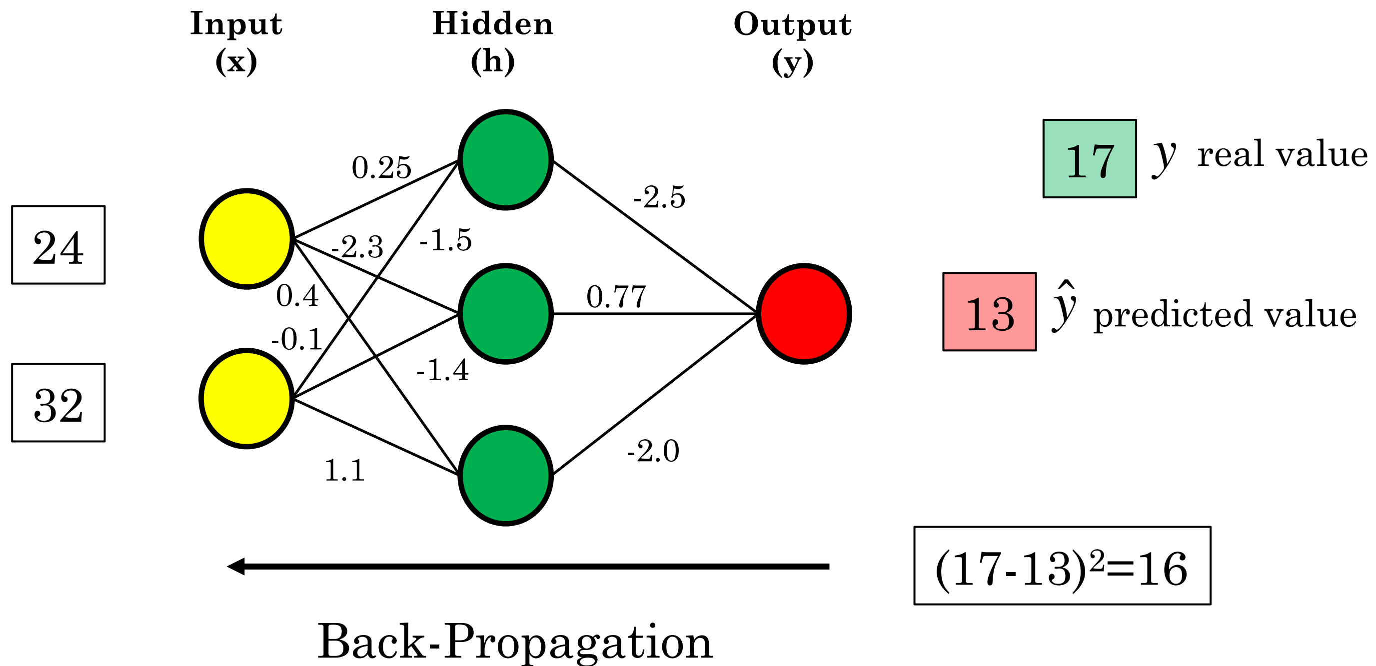
$(1 \times 0.25) + (0.5 \times (-1.5)) = 0.25 + (-0.75) \quad = \mathbf{-0.5}$

Activation function: $\frac{1}{1+e^{0.5}} = 0.3775$

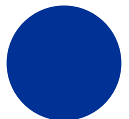


Feed-forward Neural Networks

Back-Propagation

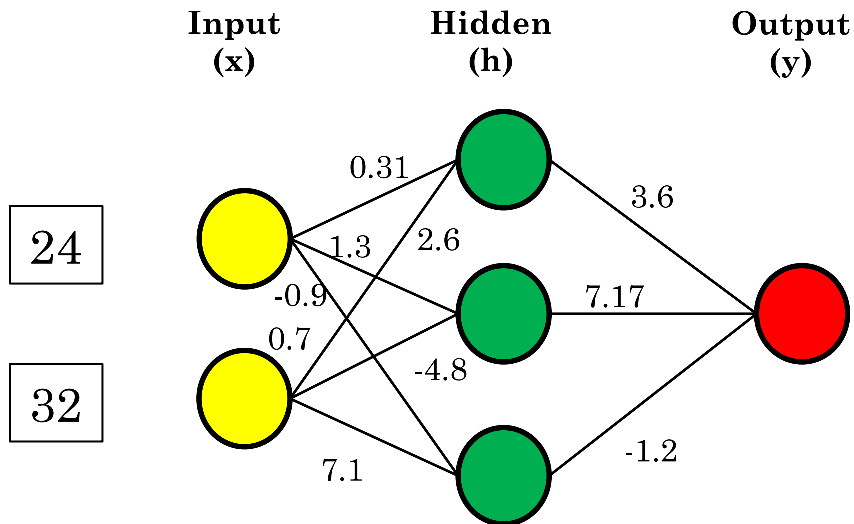


$$C(w) = \frac{1}{2}\sum_{h}^{H}\sum_{i}^{N}\left(\hat{y}_i^h - y_i^h\right)^2$$

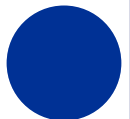


Feed-forward Neural Networks

Back-Propagation



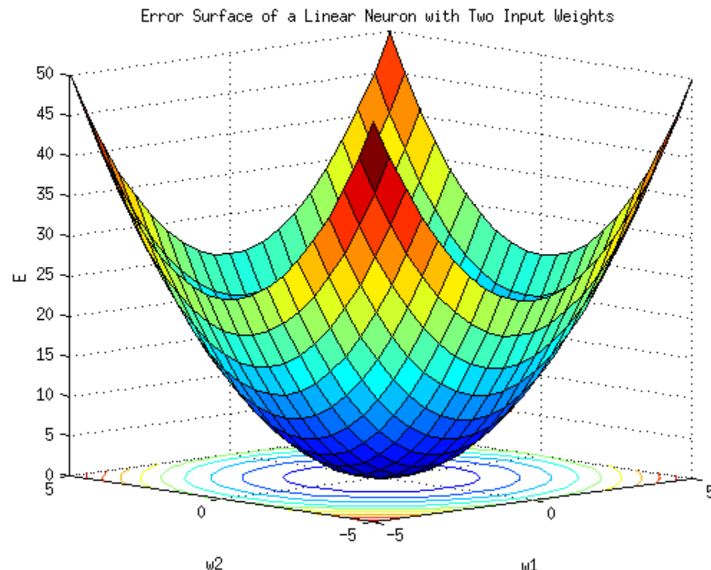
$$C(w) = \frac{1}{2} \sum_{h}^{H} \sum_{i}^{N} \left(\hat{y}_i^h - y_i^h \right)^2$$



FEED-FORWARD NEURAL NETWORKS
TRAINING

- Backpropagation
  - Requires training set (input / output pairs)
  - Starts with small random weights
  - Error is used to adjust weights

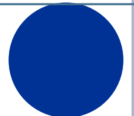
→ **Gradient descent** on error landscape



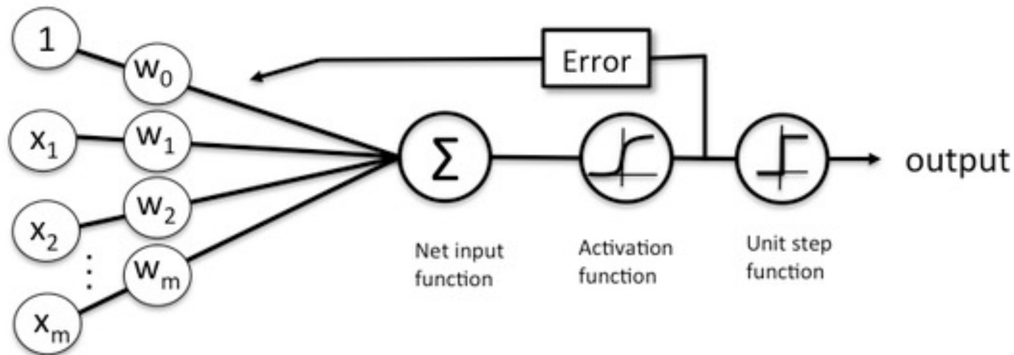
The ***cost function C*** depends on the task!

$$w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}} + \xi(t)$$

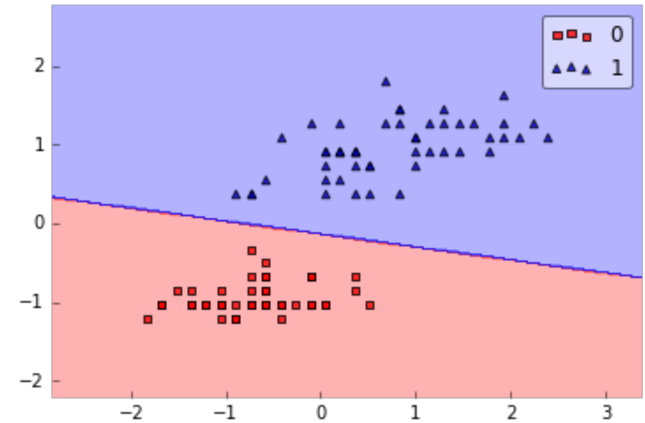
NLP: most of the cost functions are based on probability distributions underlying language composition



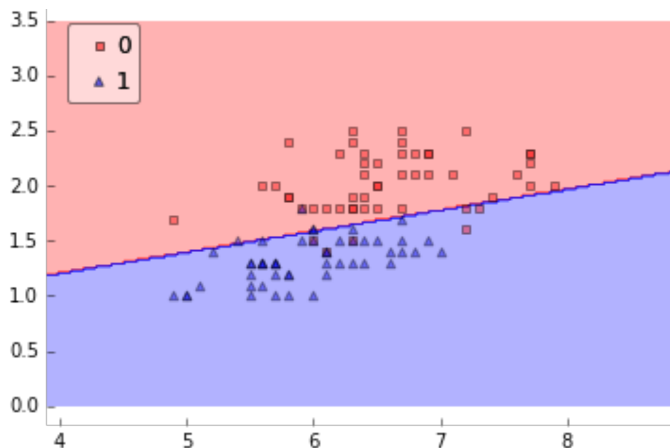
NON-LINEAR TRANSFORMATIONS

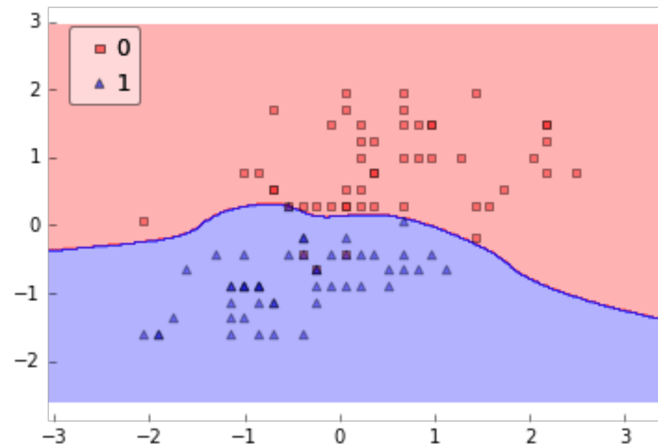


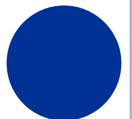
Schematic of a logistic regression classifier.



If classes can be linearly separated, **this works fine…**



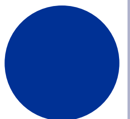




Activation functions

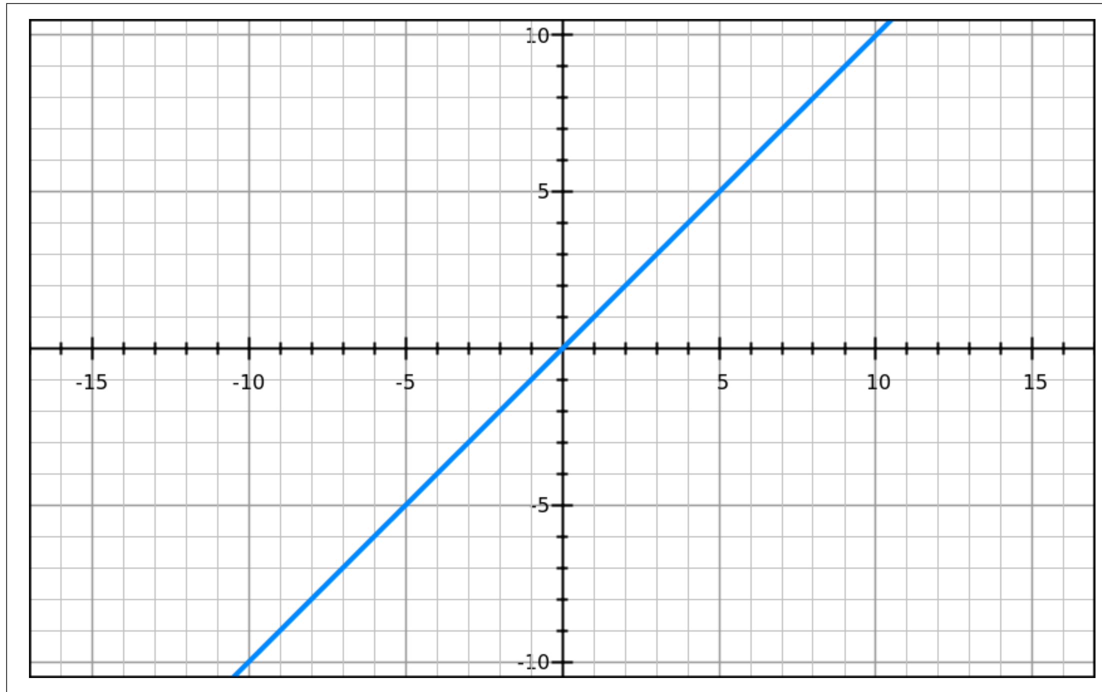
- An activation function is extremely important in neural networks:
  - They tell us if a neuron is activated or not
  - They represent a non-linear transformation

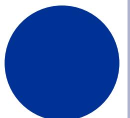
1. Linear
2. Binary Step Function
3. Sigmoid
4. Tanh
5. ReLU
6. Leaky ReLU



Activation functions

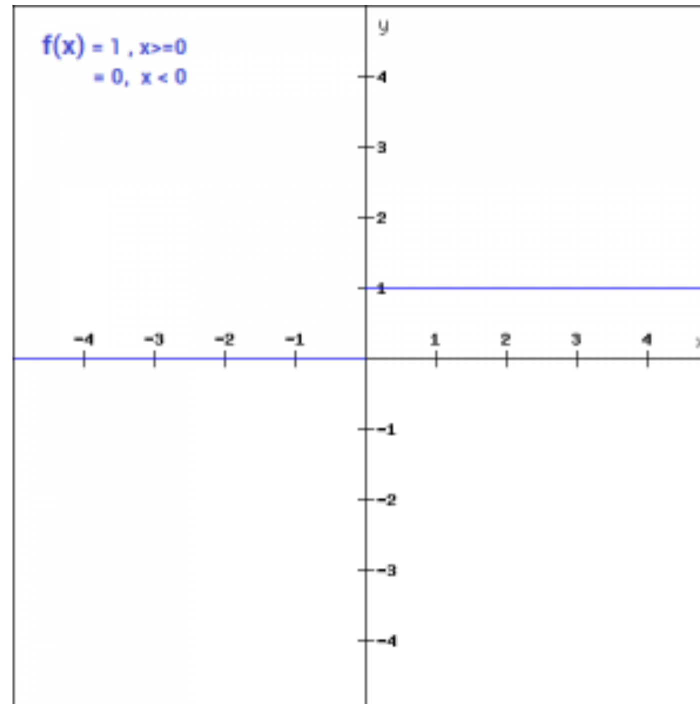
- Linear Function: A linear transform is basically the identity function

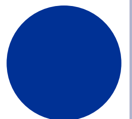




Activation functions

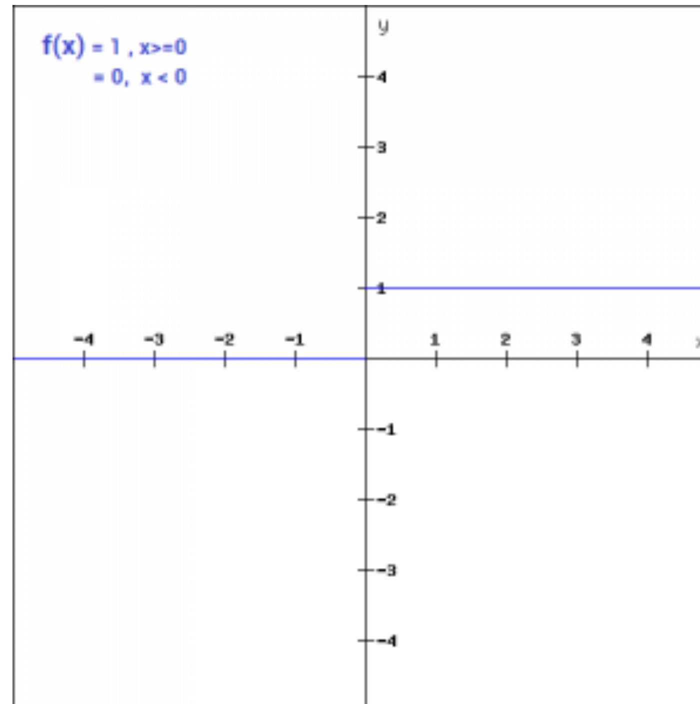
- Binary Step Function: it can be viewed as a threshold based classifier i.e. whether or not the neuron should be activated
  - If the value Y is above a given threshold value then activate the neuron else leave it deactivated.

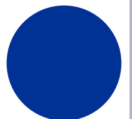




Activation functions

- Binary Step Function:
  - It is extremely simple
  - It can be used while creating a binary classifier.

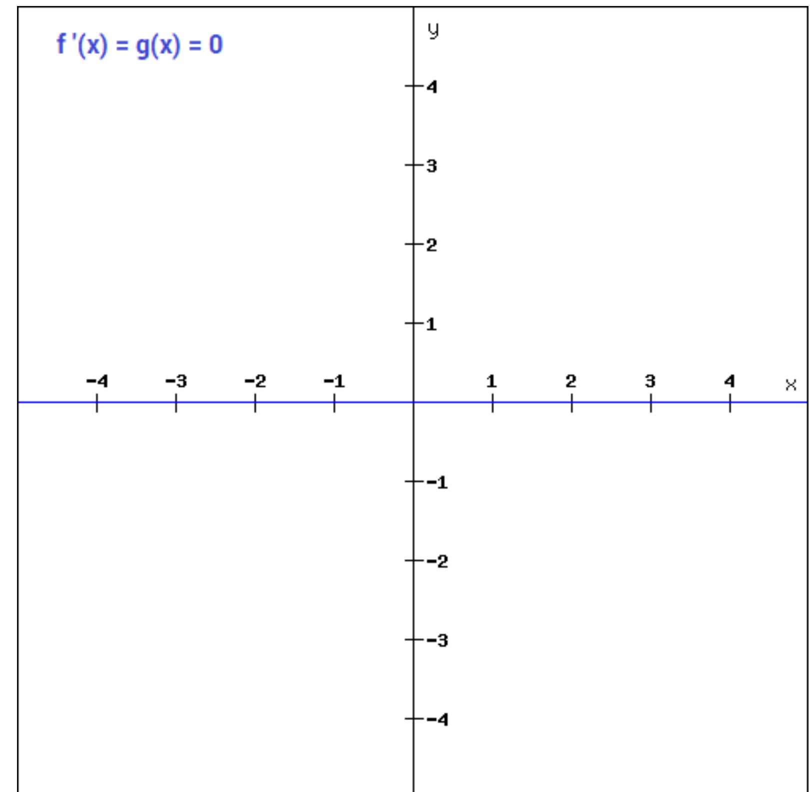


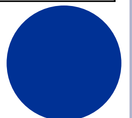


Activation functions

- What's the problem with the binary step function?

- The gradient of the step function is zero.
  - This makes the step function not so useful
  - The gradient of the step function reduces it all to zero with no improvement of the model





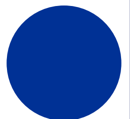
ACTIVATION FUNCTIONS

- Sigmoid: (also calle *logistic function)* is an S-shaped function that maps each value of x into the interval [0,1]

$$\frac{1}{(1+e^{-x})}$$

- It is a smooth function and is continuously differentiable.

- The biggest advantage over step linear function is that it is non-linear
  - When we have multiple neurons having sigmoid function as activation function – the output is non linear as well.

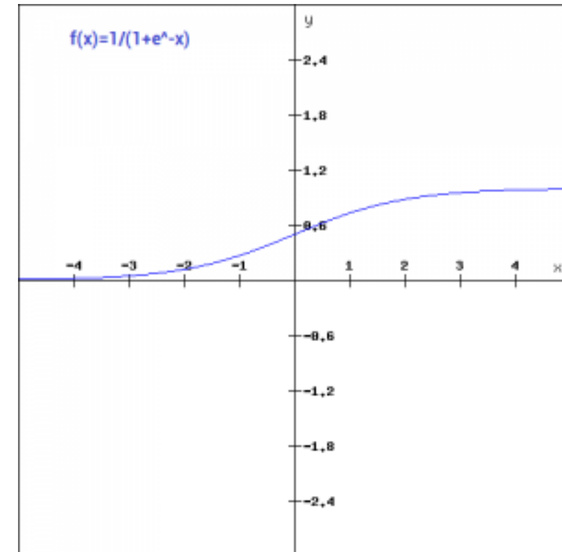


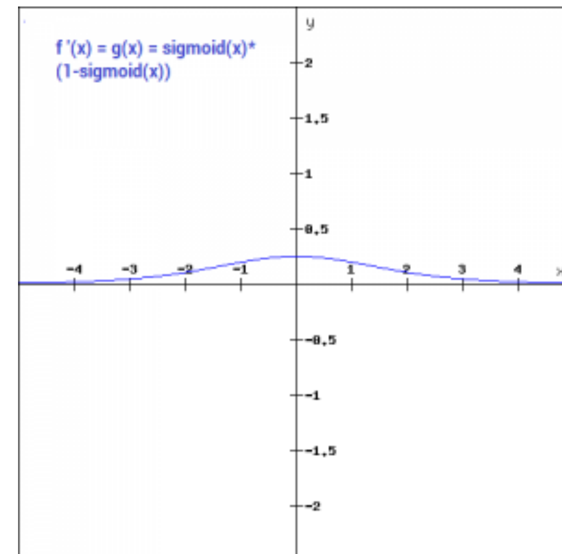
ACTIVATION FUNCTIONS

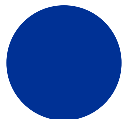
Sigmoid

- The gradient is very high between the values of -3 and 3 but gets much flatter in other regions.

- **What does it mean?**



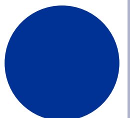




Activation functions

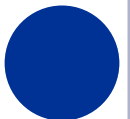
- Sigmoid is dependent on x.
  - This means that during backpropagation we can easily use this function.
  - Therefore, the error can be backpropagated and the weights can be accordingly updated.

- **So what is the problem with the sigmoid?**



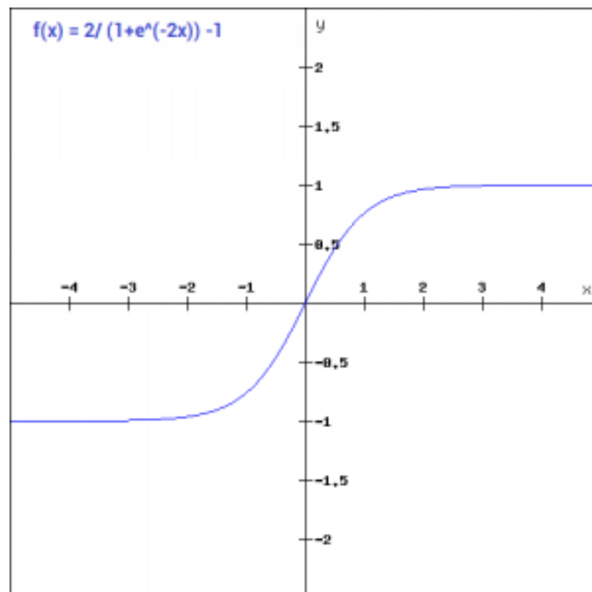
Activation functions

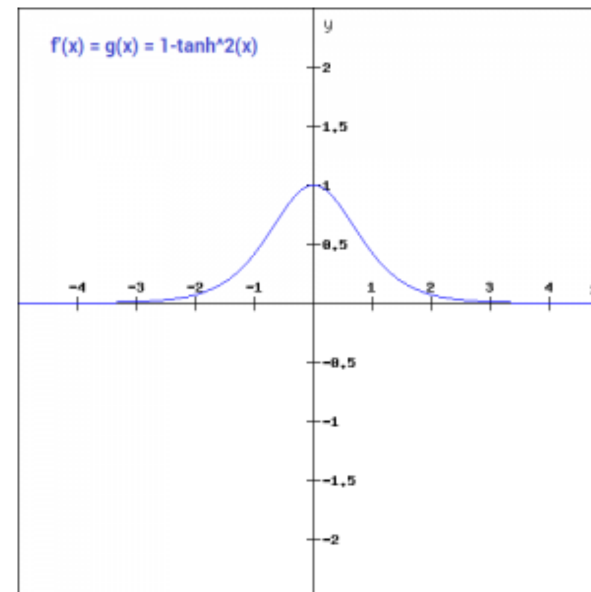
- Tanh: It is actually just a scaled version of the sigmoid function
  - It solves the problem of values all being of the same sign (around the origin).
  - All other properties are the same as that of the sigmoid function.
  - It is continuous and differentiable at all points.
  - The function as you can see is non linear so we can easily backpropagate the errors.
  - It ranges into the interval [-1,1]

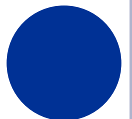


ACTIVATION FUNCTIONS

- The gradient of the tanh function is steeper as compared to the sigmoid function.
- **But…**

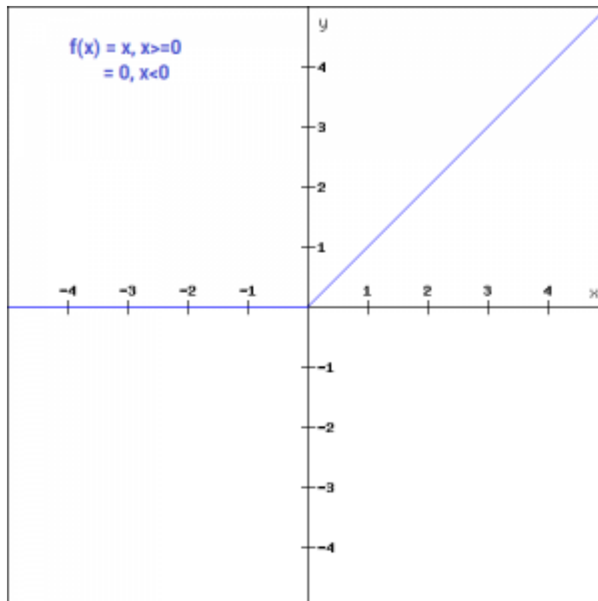






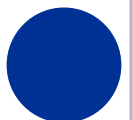
ACTIVATION FUNCTIONS

- ReLU (Rectified linear unit): It is the most widely used activation function
  - it is non linear
  - We can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
  - Main advantage over other activation functions:
    - it does not activate all the neurons at the same time.



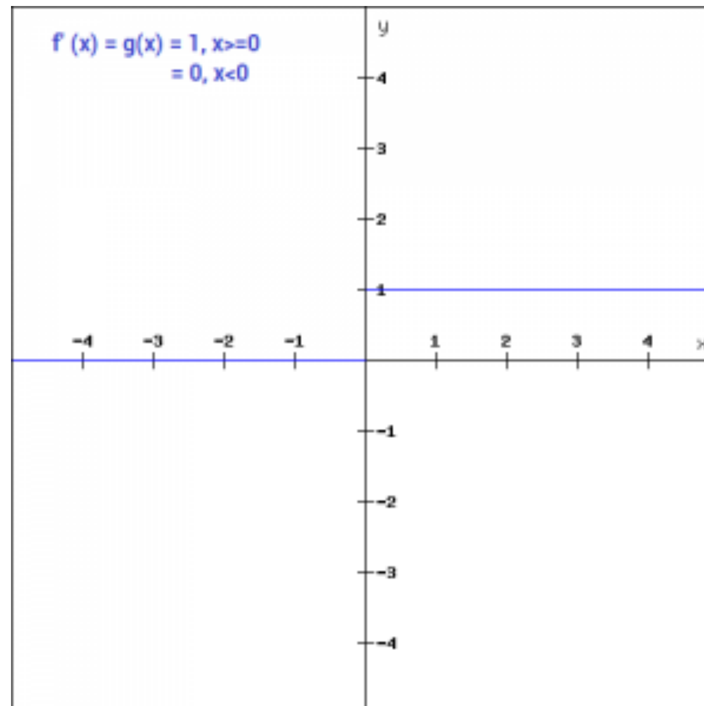
If the input is negative it will convert it to zero and the neuron does not get activated.

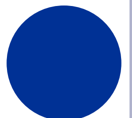
- This means that at a time only a few neurons are activated making the network sparse making it efficient and easy for computation.



ACTIVATION FUNCTIONS

- If we look at the gradient it is …very nice
  - ReLU also falls a prey to the gradients moving towards zero
  - The gradient is zero for x<0, which means the weights are not updated during the back-propagation



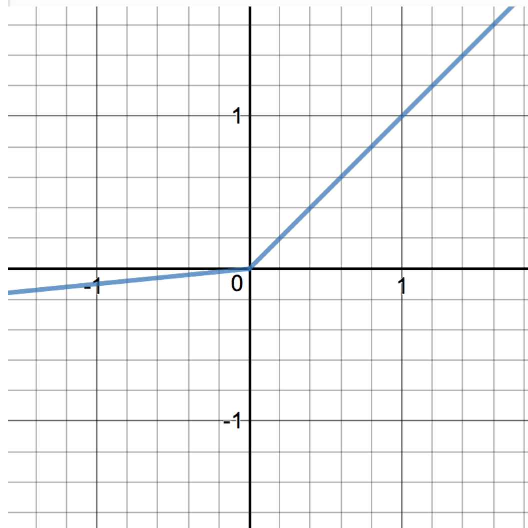


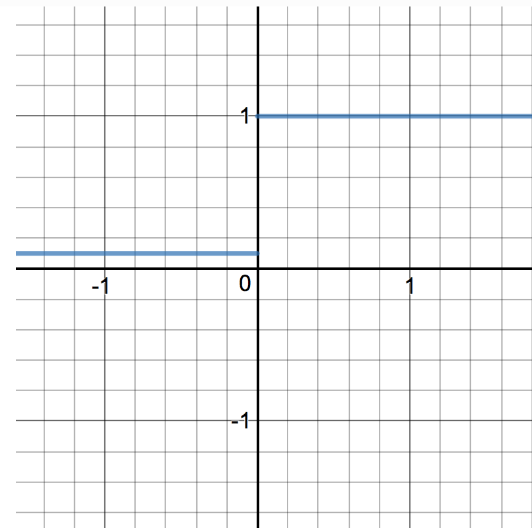
ACTIVATION FUNCTIONS

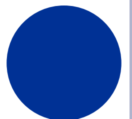
- Leaky ReLU: it is the improved version of the ReLU function even if in practice is not widely used
  - Instead of defining the Relu function as 0 for x less than 0, we define it as a small linear component of x
  - It will have a negative slope (0.01)

$$R(z) = \begin{Bmatrix} z & z > 0 \\ \alpha z & z <= 0 \end{Bmatrix}$$

$$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ \alpha & z < 0 \end{Bmatrix}$$



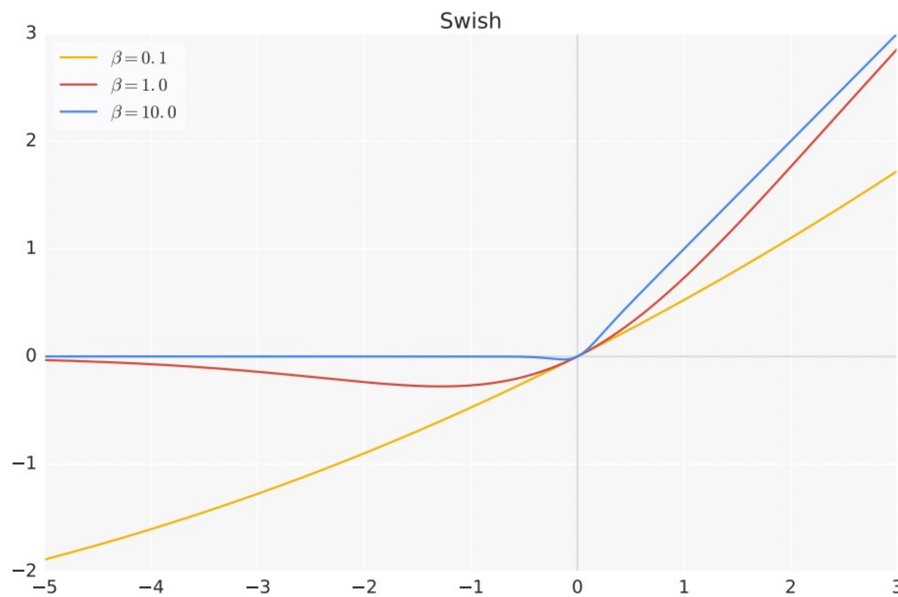


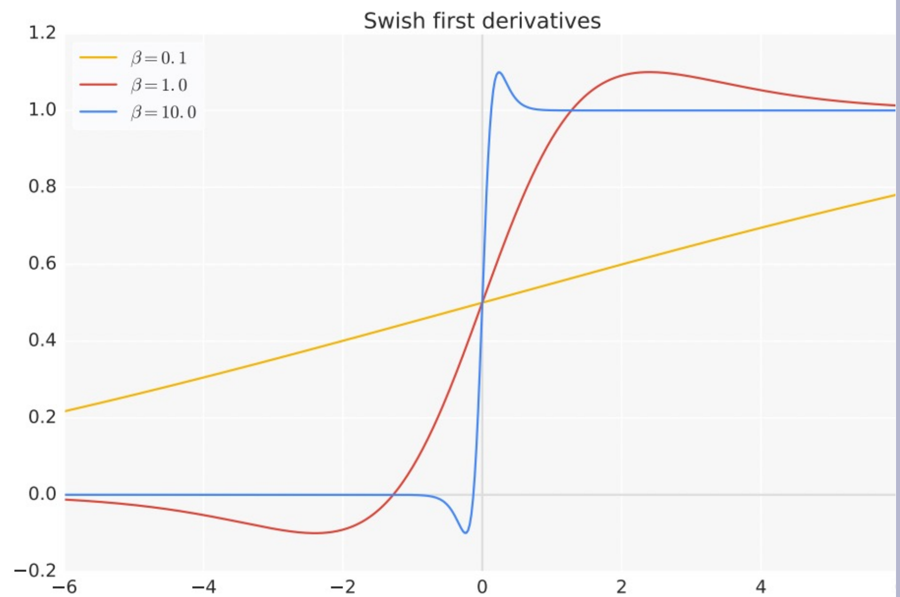


Activation functions

- Swhish: smooth and non-monotonic function

$$f(x) = x \cdot sigmoid(\beta x)$$





And now?

$$f(x)=\begin{cases} \alpha x,\ if x<0 \\ x,\ if x\geq 0 \end{cases}$$

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

$$f(x)=\begin{cases} \alpha\left(e^{\frac{x}{b}}-1\right),\ if x<0 \\ \frac{a}{b}x,\quad if x\geq 0 \end{cases} \alpha, b>0$$

$$f(x)=Scale\begin{cases} \alpha e^{x}-a,\ if x<0 \\ x,\quad if x\geq 0 \end{cases}$$

$$f(x,y)=\begin{cases} 0, if x\leq 0 and y\leq 0 \\ x, if x>0 and y\leq 0 \\ -y, if x\leq 0 and y>0 \\ x-y, if x>0 and y>0 \end{cases}$$

$$f(x)=\begin{cases} x,\quad if x>0 \\ \alpha\left(2^{\frac{x}{ln(2)}}-1\right),\ if x\leq 0 \end{cases}$$

$$f(x)=\begin{cases} Tan\,h'(\lambda^{+})(x-\lambda^{+})+Tanh(\lambda^{+}), x\geq\lambda^{+} \\ Tanh(x), \lambda^{-}<x<\lambda^{+} \\ Tanh'(\lambda^{-})(x-\lambda^{-})+Tanh(\lambda^{-}), x\leq\lambda^{-} \end{cases}$$

$$f(x)=\begin{cases} -\alpha\left(e^{\frac{-x}{b}}-1\right), if x\geq 0 \\ c\left(e^{\frac{x}{d}}-1\right), otherwise \end{cases}$$

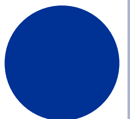
$$f(x)=\begin{cases} \alpha_{1}x-\alpha_{1}+1, if 1<x<\infty \\ x, if 0\leq x\leq 1 \\ \alpha_{2}x, if -\infty<x<0 \end{cases}$$

$$f(x)=\begin{cases} 0,\quad if x\leq 0 \\ \frac{x}{1+e^{-x}},\ if x>0 \end{cases}$$

$$f(x)=ln(1+e^{x})$$

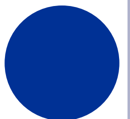
$$f(x)=\frac{x}{(1+|x|)}$$

READING



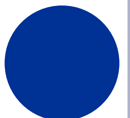
PRE-TRAINING STEPS

- Some practical suggestions:
  - Selection of Data
  - Data Preprocessing
  - Initial parameter settings



PRE-TRAINING STEPS

- Selection of data
  - It is generally difficult to incorporate prior knowledge into a neural network
    - therefore the network will only be as good as the data that is used to train it.
  - It is not always easy to be sure that the input space is adequately sampled by the training data.
    - For simple problems, in which the dimension of the input vector is small, and each element of the input vector can be chosen independently, we can sample the input space using a grid…but this can be done when the dimension of the input space is "small"

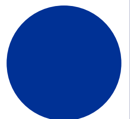


PRE-TRAINING STEPS

- How can we be sure that the input space has been adequately sampled by the training data?

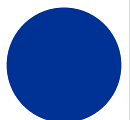
  - This is difficult to do prior to training

  - In addition, we can use techniques that indicate when a network is being used outside the range of the data with which it was trained.
    - This will not improve the network performance, but it will prevent us from using a network in situations where it is not reliable.



Pre-training steps

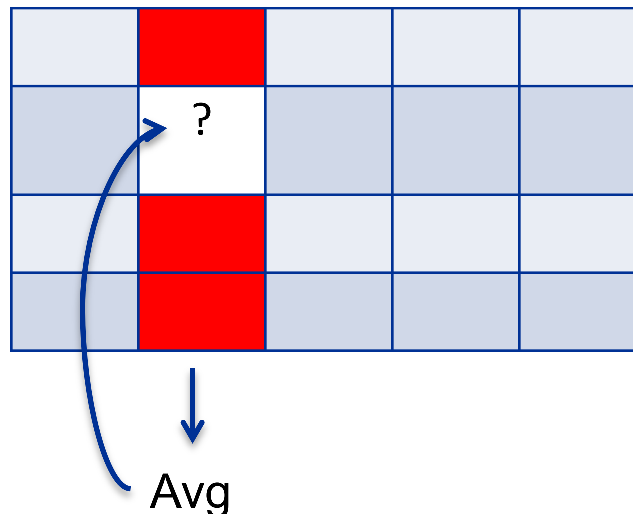
- The main purpose of the data preprocessing stage is to facilitate network training.
  - Another practical issue to consider is *missing data*
    - *According to the type of missing value, you can select between two choices*
      - *If your missing is a "value of interest", just put a 0 and let the network to learn the missing*
      - *If your missing is a "a real missing", it is better to replace it according to your knowledge*

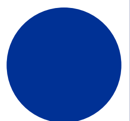


Pre-training Steps

Unsupervised Settings

- Missing Data: Most Common (MC) value
  - If the missing value is continuous
    - Replace it with the mean value of the attribute for the dataset
  - If the missing value is discrete
    - Replace it with the most frequent value of the attribute for the dataset
  - Simple and fast to compute
  - Assumes that each attribute presents a normal distribution

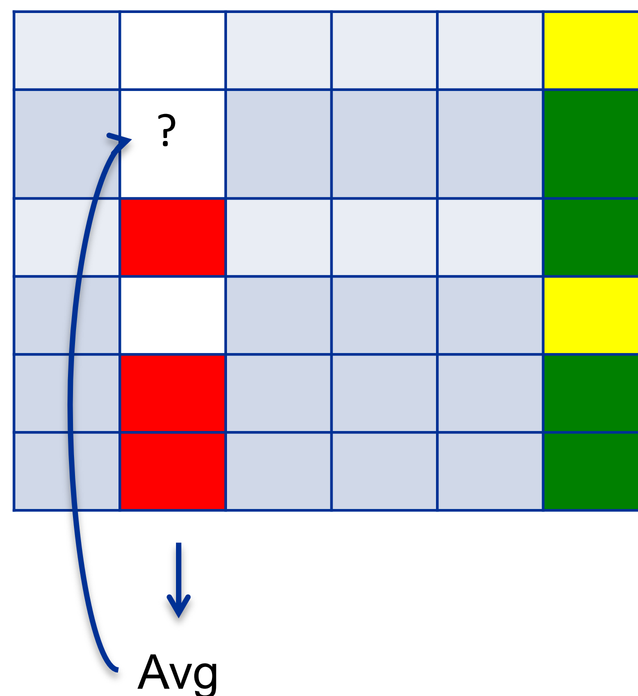


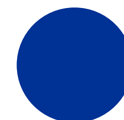


PRE-TRAINING STEPS

Supervised Settings

- Concept Most Common (CMC) value
- Refinement of the MC policy
- The MV is replaced with the *mean*/most frequent value computed from the instances *belonging to the same class*
- Assumes that the distribution for an attribute of all instances from the same class is normal

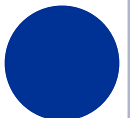




PRE-TRAINING STEPS

- Recently…

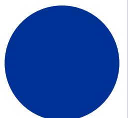
Śmieja, M., Struski, Ł., Tabor, J., Zieliński, B., & Spurek, P. (2018). Processing of missing data by neural networks. In Advances in Neural Information Processing Systems (pp. 2719-2729).



PRE-TRAINING STEPS

- Before training the network, we need to initialize the weights and biases.

- Traditional strategies for traditional ANN:
  1. Initializing all weights to 0
  - When you set all weight to 0, the derivative with respect to loss function is the same for every w
    - all the weights have the same values in the subsequent iteration
    - This makes the hidden units symmetric and continues for all the n iterations of the training.
    - Setting weights to zero makes your network no better than a linear model.



PRE-TRAINING STEPS

2. Initializing weights randomly
   - Initializing weights randomly, following standard normal distribution
   - while working with a deep network, this can potentially lead to 2 issues:

   a) Vanishing gradients: weights receives an update proportional to the partial derivative of the error function with respect to the current weight.

      - With sigmoid and tanh, if your weights are large, then the gradient will be really small, preventing the weights from changing their value. This is because abs(dW) will increase very slightly or possibly get smaller and smaller every iteration.

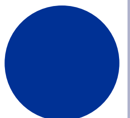
      - With ReLU vanishing gradients are generally not a problem as the gradient is inputs less than 0 and 1 for positive inputs.

PRE-TRAINING STEPS

b) Exploding gradients:

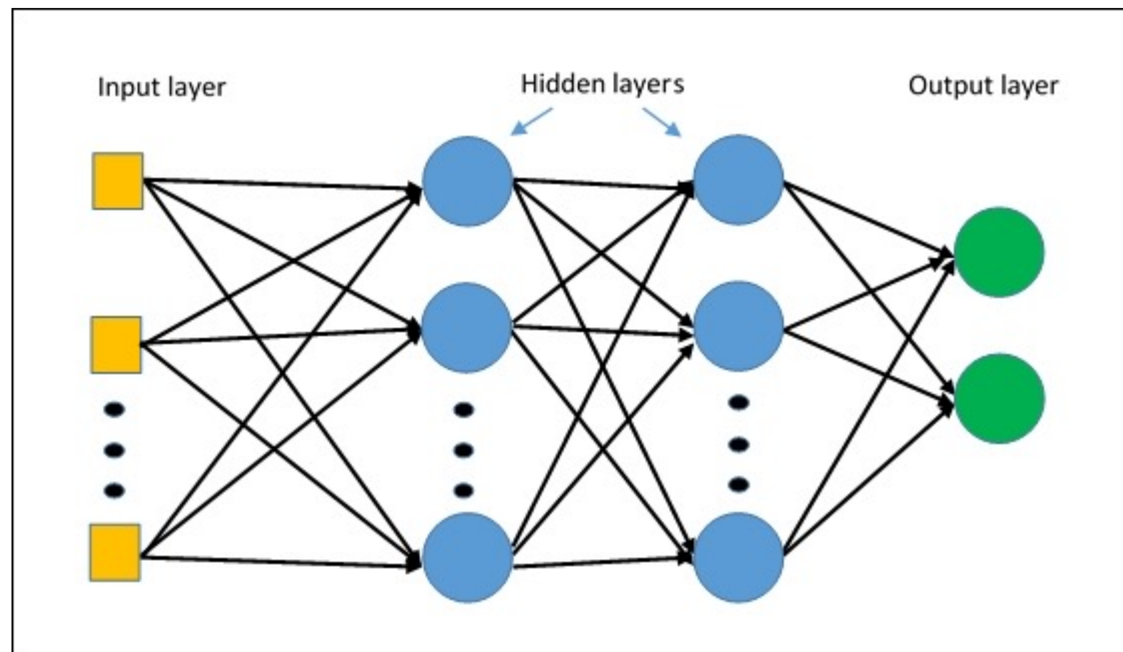
- Let's consider non-negative and large weights for a sigmoid.
- When these weights are multiplied along the layers, they cause a large change in the cost.
  - The gradients become large, implying that the changes in W will be in huge steps
- This may result in oscillating around the minima or even overshooting the optimum again and again and the model will never learn!

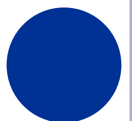
- In order to avoid this issues, there are some heuristics that can be adopted for initializing weights according to the activation functions that we have



Multi-Layers Neural Networks

- In principle, we could **enlarge** the neural network as much as we want…





Multi-Layers Neural Networks

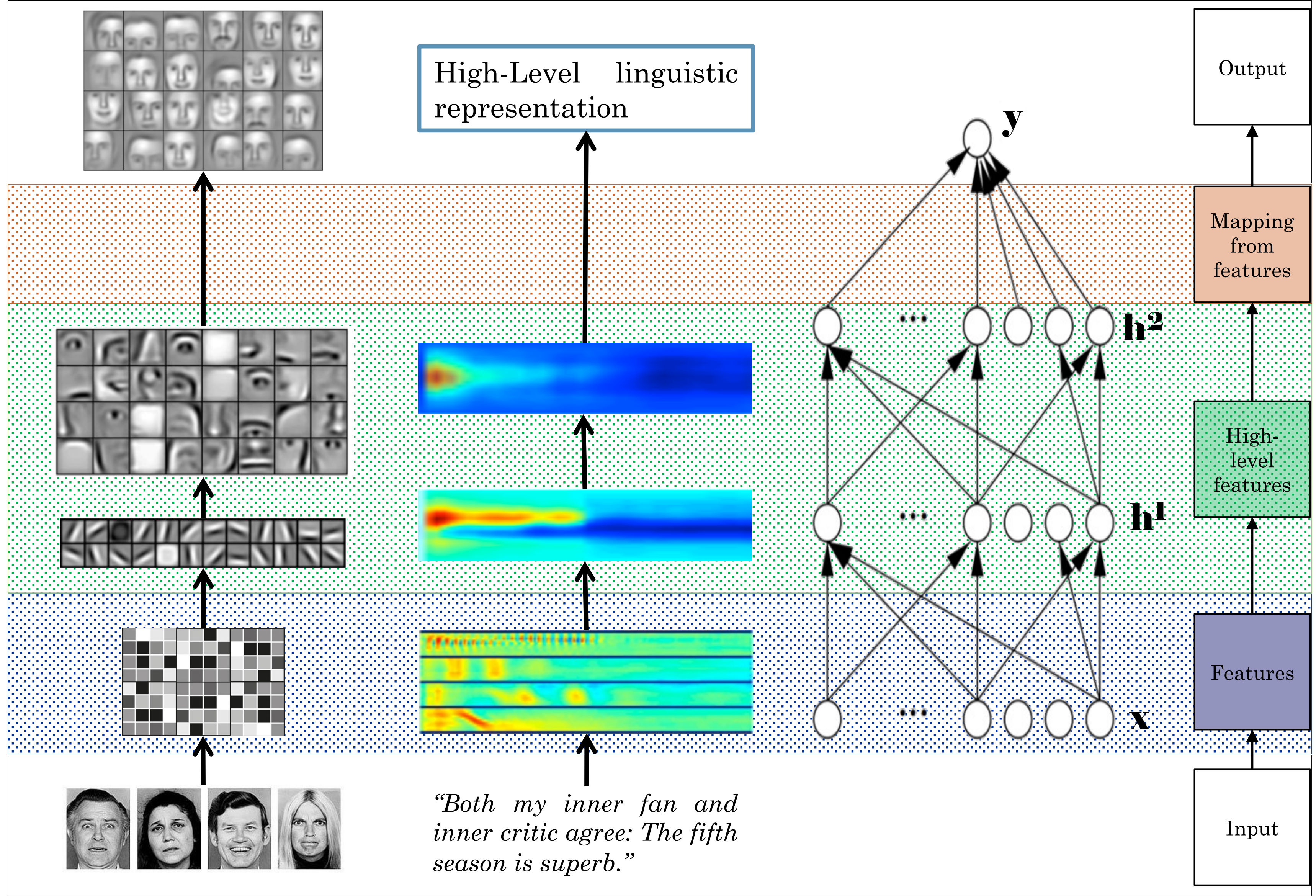
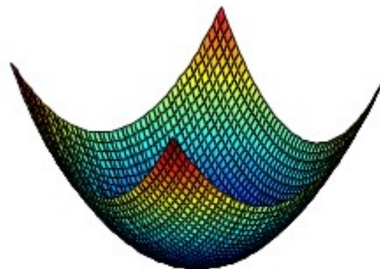


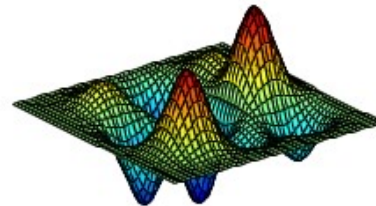
Image | Text | Model

Multi-Layers Neural Networks

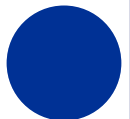
- **Pros**
  - Very flexible → we can select activation functions according to our goal.
  - Highly parallel → we can compute each activation function (neuron) separately.
- **Cons**
  - It requires labelled training data.
  - The learning time does not scale well with increasing number of layers and neurons.
  - It can get stuck in poor local optima (non-convex loss function).



Unique optimum: global/local



Multiple local optima



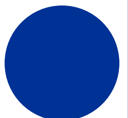
Multi-Layers Neural Networks

What was actually wrong with backpropagation in 1986?

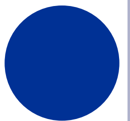
- We all drew the wrong conclusions about why it failed. The real reasons were:

1. Our labeled datasets were thousands of times too small.
2. Our computers were millions of times too slow.
3. We initialized the weights in a stupid way.
4. We used the wrong type of non-linearity.

What Was Actually Wrong With Backpropagation in 1986?
Slide by Geoff Hinton, all rights reserved.



So what is DEEP LEARNING now?

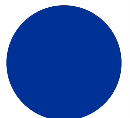


2006: The Deep Breakthrough

- Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). **A fast learning algorithm for deep belief nets**. *Neural computation*.

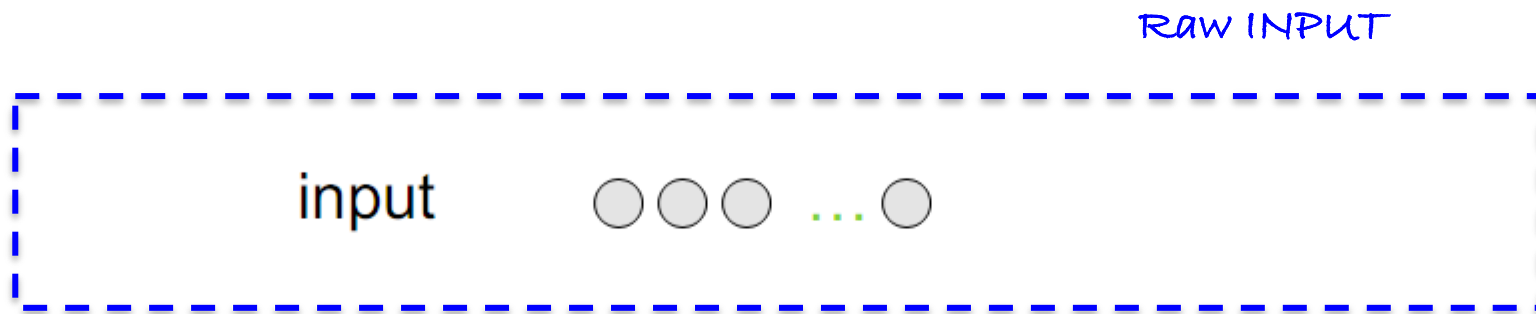
- Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). **Greedy layer-wise training of deep networks**. *Advances in neural information processing systems*.

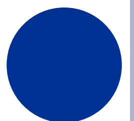
- Poultney, C., Chopra, S., & Cun, Y. L. (2006). **Efficient learning of sparse representations with an energy-based model**. *Advances in neural information processing systems*.



LAYER-WISE UNSUPERVISED PRE-TRAINING

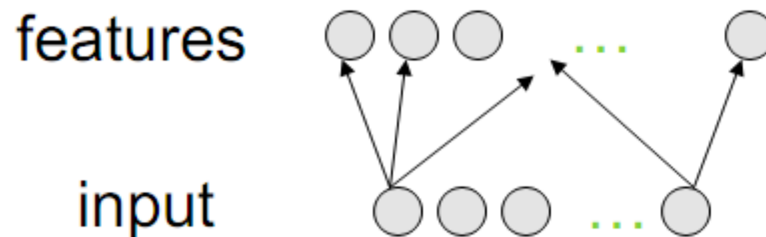
Raw INPUT

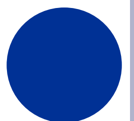




Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

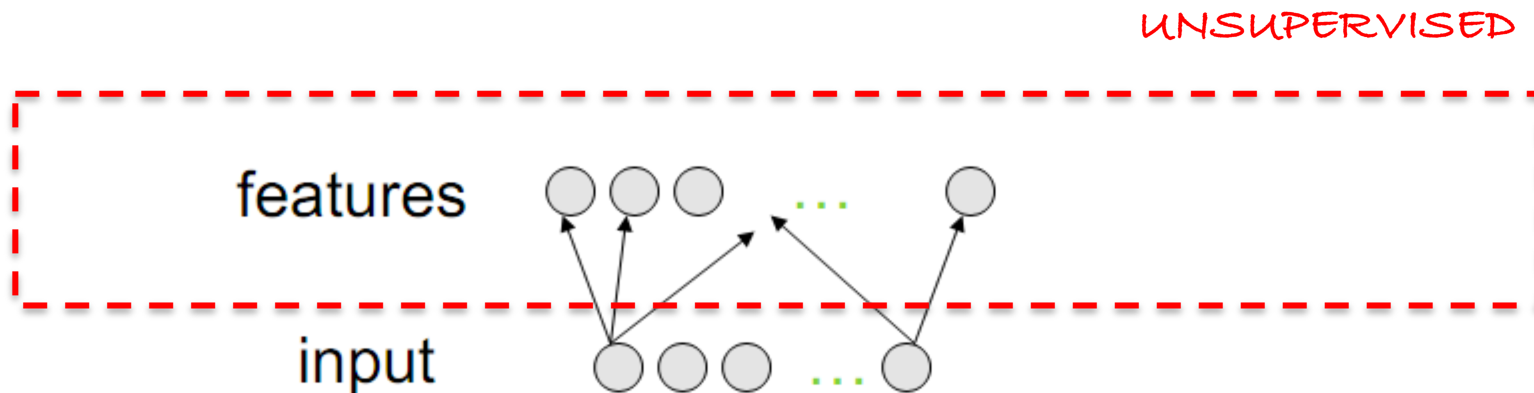
LAYER-WISE UNSUPERVISED PRE-TRAINING

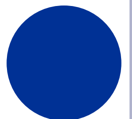




Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

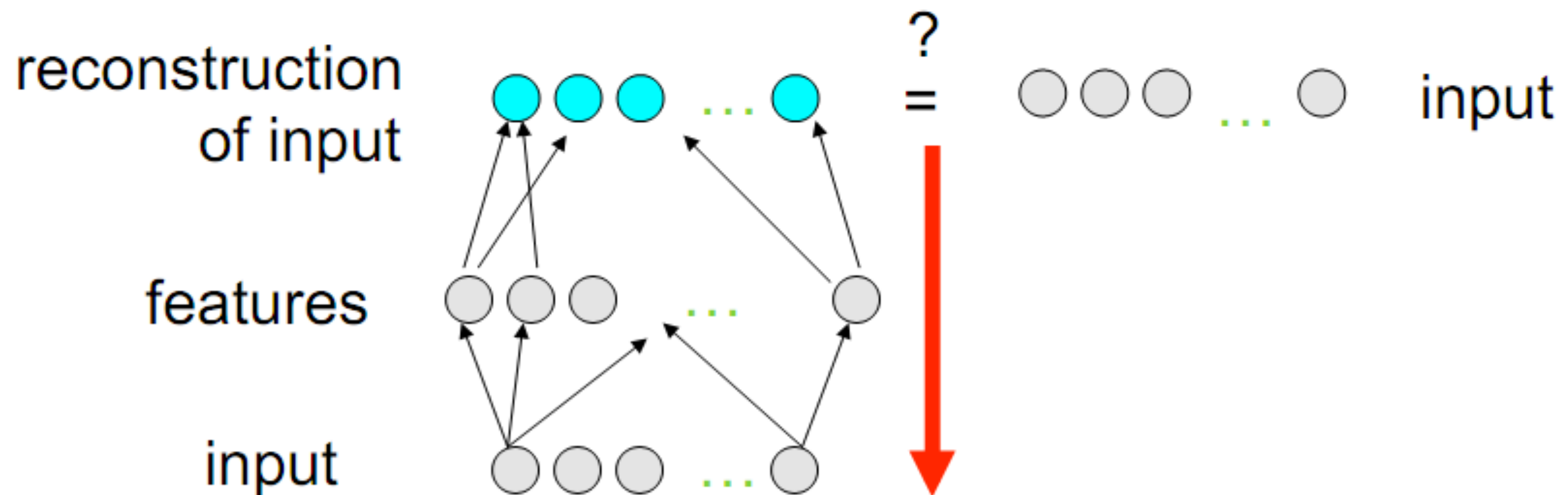
LAYER-WISE UNSUPERVISED PRE-TRAINING





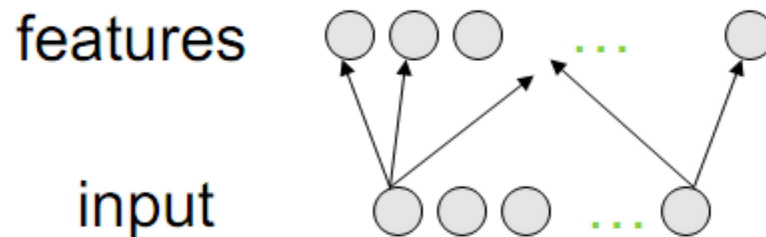
Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

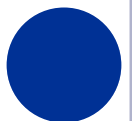
Layer-wise Unsupervised Pre-training



Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

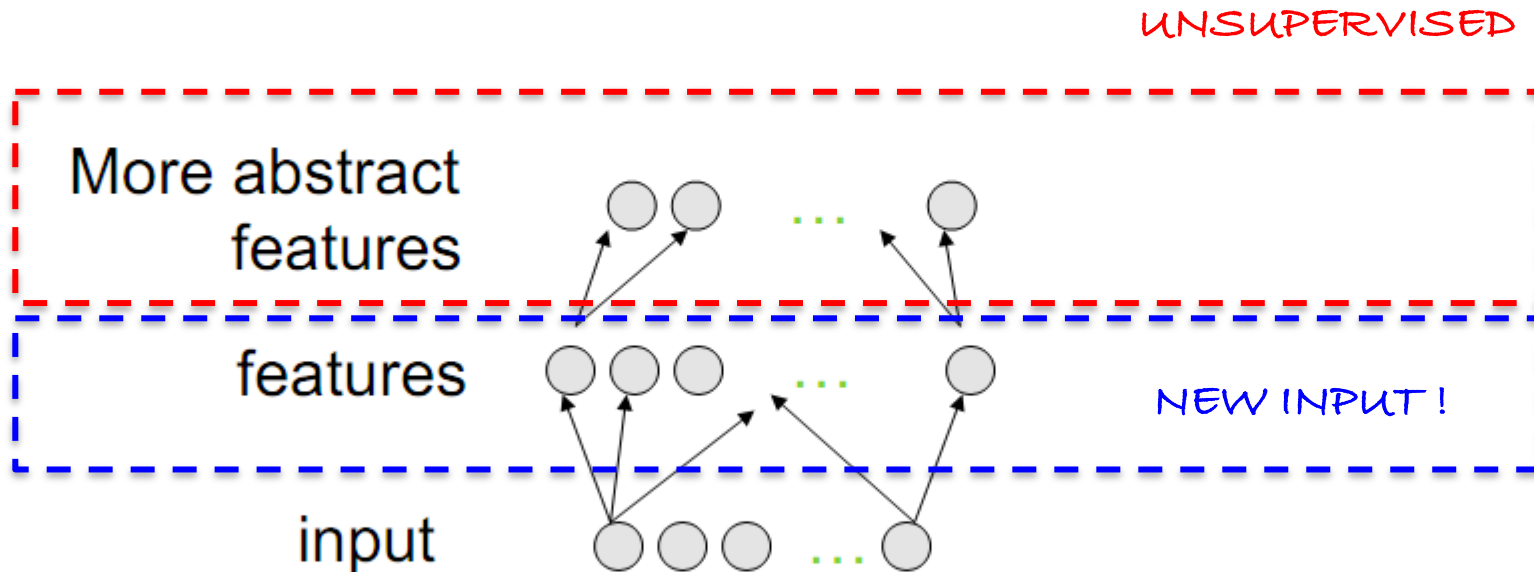
LAYER-WISE UNSUPERVISED PRE-TRAINING

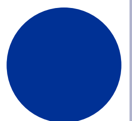




Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

Layer-wise Unsupervised Pre-training





Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

Layer-wise Unsupervised Pre-training

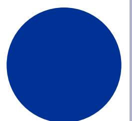
reconstruction

?
=

More abstract features

features

input



Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

Layer-wise Unsupervised Pre-training

reconstruction

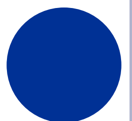
?

=

More abstract features

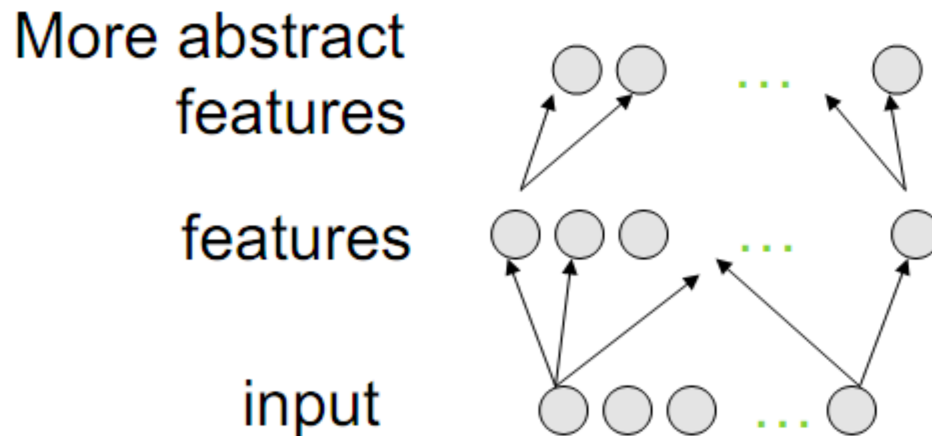
features

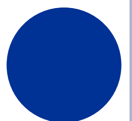
input



Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

LAYER-WISE UNSUPERVISED PRE-TRAINING





Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

Layer-wise Unsupervised Pre-training

UNSUPERVISED

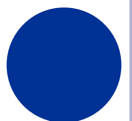
Even more abstract features

More abstract features

NEW INPUT !

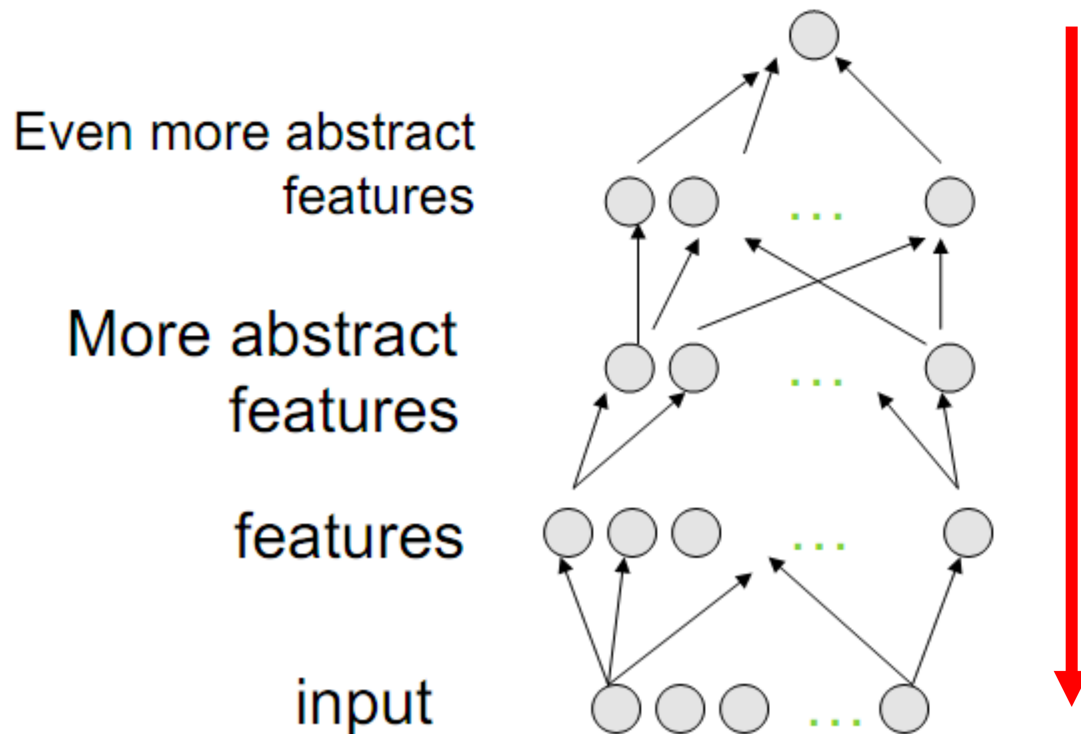
features

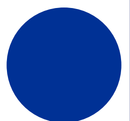
input



Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

SUPERVISED FINE-TUNING





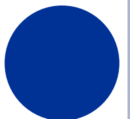
Socher, Richard, Yoshua Bengio, and Christopher D. Manning. "Deep learning for NLP (without magic)." *Tutorial Abstracts of ACL 2012*. Association for Computational Linguistics, 2012.

Layer-wise Unsupervised Pre-training

- The main intuition

$$\text{regularizer} = -\log P(\theta).$$

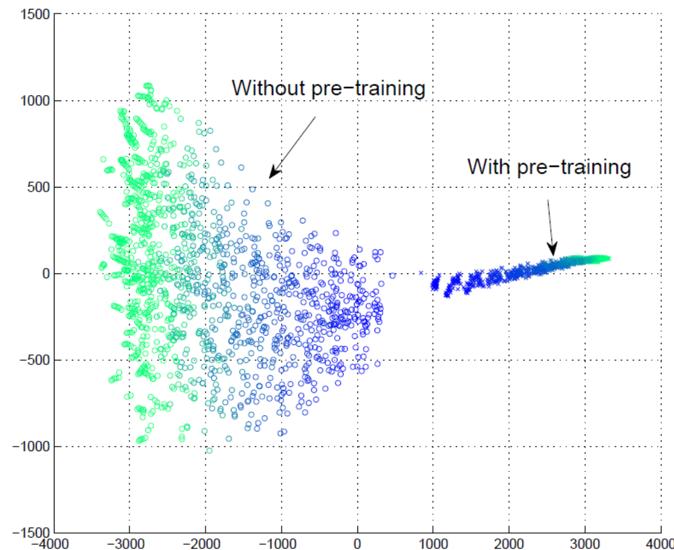
$$P_{\text{pre-training}}(\theta) = \sum_k 1_{\theta \in R_k} \pi_k / v_k. \qquad P_{\text{no-pre-training}}(\theta) = \sum_k 1_{\theta \in R_k} r_k / v_k.$$



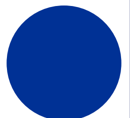
Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning?. *Journal of Machine Learning Research*, *11*(Feb), 625-660.

Layer-wise Unsupervised Pre-training

- Why is unsupervised pre-training working so well?
  - Regularization hypothesis:
    - Representations good for P(x) are good for P(y|x)



Visualization of Model Trajectories During Learning



Erhan, D., Bengio, Y., Courville, A., Manzagol, P. A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning?. *Journal of Machine Learning Research*, *11*(Feb), 625-660.