Multilayer networks & backprop

CS530
Chapman

Spring 2021

1

MORE ABOUT FROM SINGLE LAYER TO MULTILAYER NETWORKS

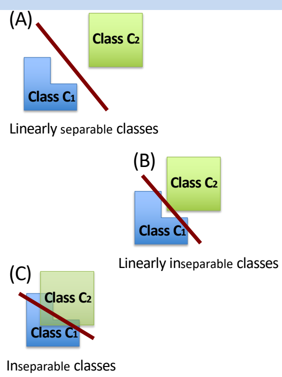
2

Who needs multi-layer networks?

So far, we focused more on 1-layer networks and saw that they can classify only linearly separable problems, either using the perceptron or delta rules.

What if our classes are not linearly separable, or not even perfectly separable using any hyper-curve?

The perceptron rule would never converge, forever striving to find a line that perfectly separates the classes. The delta rule would converge to some local minimum, resulting in a line that would separate the classes, to some extent and with errors.

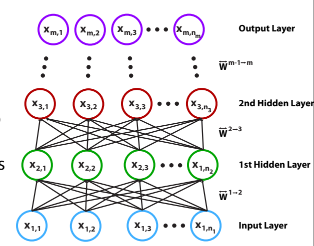


3

Who needs multi-layer networks?

More-general architectures can achieve non-linear separators and much more. Such a network has an input layer, whose neurons' states are the patterns input into the network. It has 1 or more hidden layers, called hidden because we do typically directly manipulate their inputs or outputs. Last, they possess an output layer, where the output classes are represented.

In the general sense, the network strives to transform the states of its input layer into states of its output layer that would match its classification goals.

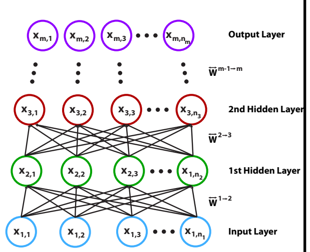


4

Who needs multi-layer networks?

If the network contains only linear neurons, it practically reduces to a single layer. The output would then just be a series of matrix multiplications of the inputs. And all those matrix multiplications would end up as a single resulting matrix, hence practically one layer.

To get more processing power than a single layer, non-linearities must be introduced into the network. This is done in the form of non-linear activation functions for at least some of the neurons.



5

How to represent multi-layer networks?

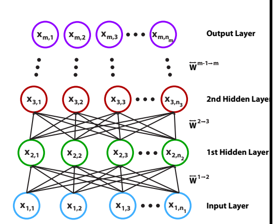
A general layer in a network does the following computation:

$\vec{x}^{k+1} = f\left(\vec{W}^{k\to k+1}\vec{x}^{k}\right).$

So the weights an n-layer network are an n-1 ordered series:

$\left(\vec{W}^{1\to 2}, \vec{W}^{2\to 3}, \cdots, \vec{W}^{n-1\to n+1}\right)$

Such a series of matrices is termed a **tensor**.

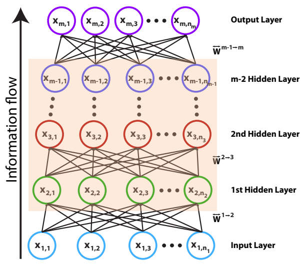


6

Function of hidden layers

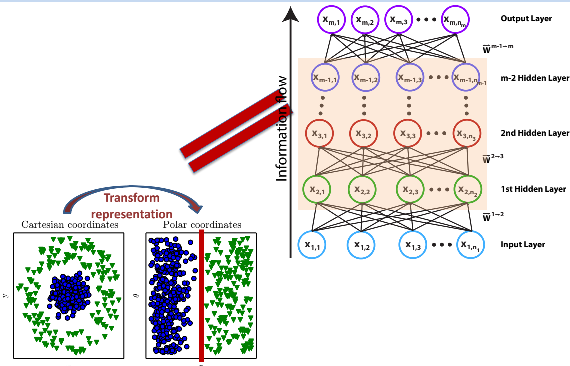
The hidden layers in a multilayer neural network serve as feature detectors. As the network weights are trained, the hidden-layer neurons take on salient features that characterize the training data.

They carry this out using nonlinear transformations on the input patterns. In the resulting non-linear feature space, the classification problem is typically more easily achieved than in the original space. In this respect, multilayer neural networks learn the nonlinear transformation kernels that they use on their own.



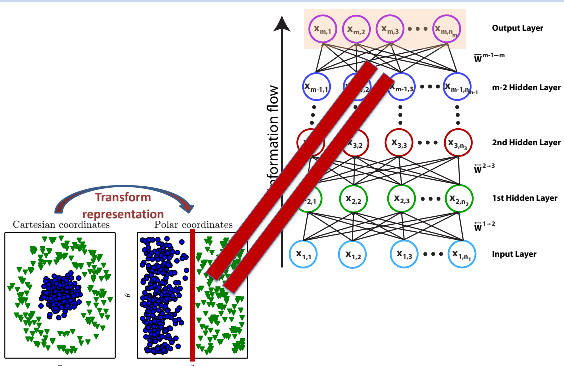
7

Function of hidden layers



8

Function of output layer



9

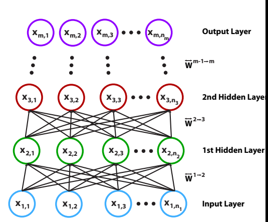
TRAINING MULTILAYER NETWORKS: THE BACKPROP

10

How to train multi-layer networks?

We can train multi-layer neural networks using gradient descent (or SGD), finding at least a local minimum. But how do we compute the gradient on such multi-layer networks?

Using back propagation (or the "backprop").

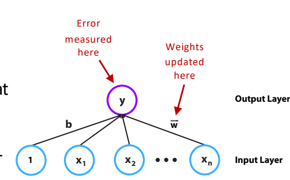


11

From single- to multi-layer networks

The perceptron and delta rules are effective error-correction methods using gradient descent for single-layer networks. But they both depend on:

1. Being able to measure the error between desired and actual output at the output neuron
2. The fact that only the weights into a single neuron contribute to the error for that neuron



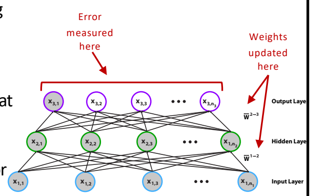
12

From single- to multi-layer networks

The perceptron and delta rules are effective error-correction methods using gradient descent for single-layer networks. But they both depend on:

1. Being able to measure the error between desired and actual output at the output neuron
2. The fact that only the weights into a single neuron contribute to the error for that neuron

In multilayer networks the error could be due to weights into that neuron from the previous layer. But it could also be due to all the weights from any downward layers. So which weights caused the error? This is a difficult credit-assignment problem.



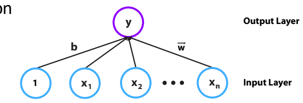
Any gray neuron could contribute to the error in $x_{3,1}$

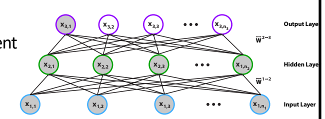
13

From single- to multi-layer networks

In sum:

- For single-layer networks, the gradient of the error (or loss) function can easily and directly be computed
- For multilayer networks computing the gradient is not so trivial.
- The backpropagation (backprop) algorithm computers the gradient of the error function
- And we then use (stochastic) gradient descent to decrease the error





Any gray neuron could contribute to the error in $x_{3,1}$

14

How does batch SDG work here?

1. We decide on a batch size, m', out of overall m samples
2. We permute (randomly shuffle) the training set
3. We divide it into $\left\lfloor \frac{m\prime}{m} \right\rfloor$ batches and run each to compute 1 gradient step (using backprop)
4. The remaining $m - \left\lfloor \frac{m'}{m} \right\rfloor \cdot m'$ samples are then integrated with the next iteration through the m samples

15

BACK PROPAGATION ALGORITHM (OR "THE BACKPROP")

16

Reminders

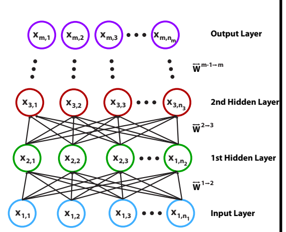
- In this network, each layer is a direct function of the layer below it

$$\vec{x}^{i+1} = f(\overleftrightarrow{W}^{i\to i+1} \cdot \vec{x}^{i})$$

- So,

$$\hat{\vec{y}} = \vec{x}^{m}$$

$$\vec{x}^{m} = f(\overleftrightarrow{w}^{m-1\to m} \cdot \vec{x}^{m-1})$$
$$= f(\overleftrightarrow{w}^{m-1\to m} \cdot f(\overleftrightarrow{w}^{m-2\to m-1} \cdot \vec{x}^{m-2}))$$
$$= f\Big(\overleftrightarrow{w}^{m-1\to m} \cdot f\big(\overleftrightarrow{w}^{m-2\to m-1} \cdot f(\cdots f(\overleftrightarrow{w}^{1\to 2} \cdot \vec{x}^{1})\cdots)\big)\Big)$$



17

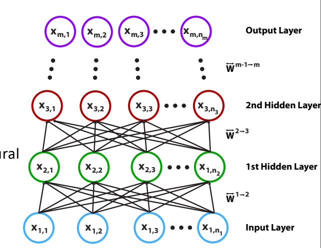
Reminders

- Collecting the weights (and biases) connecting all the layers into one 3D matrix, or tensor, we get:

$$\underline{\underline{w}} = [\overleftrightarrow{W}^{1\to 2}, \overleftrightarrow{W}^{2\to 3}, \dots, \overleftrightarrow{W}^{m-1\to m}]$$

- So, $\underline{\underline{w}}$ holds all the weights in the neural network
- Now, we define

$$g\left(\underline{\underline{w}}, \vec{x}^{1}\right) = f\Big(\overleftrightarrow{w}^{m-1\to m} \cdot f\big(\overleftrightarrow{w}^{m-2\to m-1} \cdot f(\cdots f(\overleftrightarrow{w}^{1\to 2} \cdot \vec{x}^{1}))\cdots\big)\Big)$$

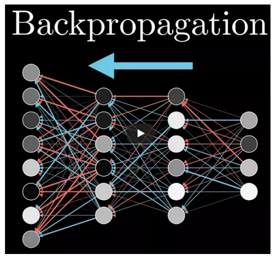


18

Animation intro to backprop

- Good (though not perfect) animation introduction to the backprop
- https://www.youtube.com/watch?v=Ilg3gGewQ5U

3Blue1Brown
3.63M subscribers



19