# Neural Networks

Note: Unless otherwise noted all references including images are from the required textbook, Machine Learning: A Probabilistic Perspective by Kevin P. Murphy.

# Feedforward Neural Networks

A **feedforward neural network**, aka **multi-layer perceptron (MLP)**, is a series of logistic regression models stacked on top of each other, with the final layer being either another logistic regression or a linear regression model, depending on whether we are solving a classification or regression problem.

# Feedforward Neural Networks

For example, if we have two layers, and we are solving a regression problem, the model has the form

$$
\begin{aligned}
p(y|\mathbf{x}, \boldsymbol{\theta}) &= \mathcal{N}(y|\mathbf{w}^T \mathbf{z}(\mathbf{x}), \sigma^2) \\
\mathbf{z}(\mathbf{x}) &= g(\mathbf{V}\mathbf{x}) = [g(\mathbf{v}_1^T \mathbf{x}), \ldots, g(\mathbf{v}_H^T \mathbf{x})]
\end{aligned}
$$

# Feedforward Neural Networks

Where

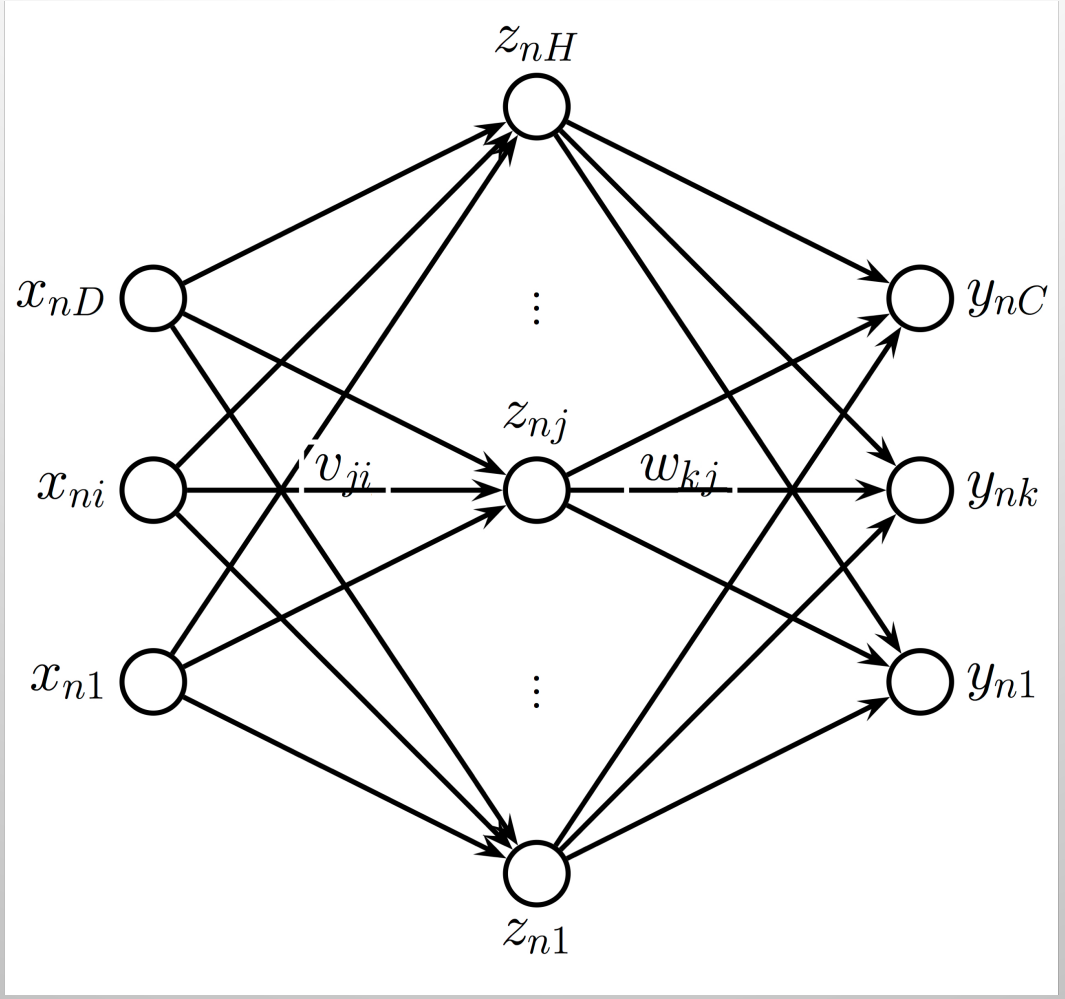
- $g$ is a non-linear **activation** or **transfer function** (commonly the logistic function),
- $\mathbf{z}(\mathbf{x}) = \phi(\mathbf{x}, \mathbf{V})$ is called the **hidden layer**,
- $H$ is the number of hidden units,
- $\mathbf{V}$ is the weight matrix from the inputs to the hidden nodes, and
- $\mathbf{w}$ is the weight vector from the hidden nodes to the output.

# Feedforward Neural Networks

It is important that $g$ be nonlinear otherwise the whole model collapses into a large linear regression model of the form $y = \mathbf{w}^T(\mathbf{Vx})$.

MLP is a universal approximator, meaning it can model any suitably smooth function, given enough hidden units, to any desired level of accuracy.

# Feedforward Neural Networks




A neural network with one hidden layer

# A Brief History of the Field

Neural networks have been the subject of great interest for many decades, due to the desire to understand the brain, and to build learning machines.

# A Brief History of the Field

The field is generally viewed as starting with McCulloch and Pitts (1943), who devised a simple mathematical model of the neuron in 1943, in which they approximated the output as a weighted sum of inputs passed through a threshold function, $y = \mathbb{I}(\sum_i w_i x_i > \theta)$, for some threshold $\theta$. This is similar to a sigmoidal activation function.

# A Brief History of the Field

Frank Rosenblatt invented the perceptron learning algorithm in 1957, which is a way to estimate the parameters of a McCulloch-Pitts neuron.

A very similar model called the adaline (for adaptive linear element) was invented in 1960 by Widrow and Hoff.

# A Brief History of the Field

In 1969, Minsky and Papert published a famous book called “Perceptrons” in which they showed that such linear models, with no hidden layers, were very limited in their power, since they cannot classify data that is not linearly separable. This considerably reduced interest in the field.

# A Brief History of the Field

In 1986, Rumelhart, Hinton and Williams (1986) discovered the backpropagation algorithm, which allows one to fit models with hidden layers.

The backpropagation algorithm was originally discovered by Bryson and Ho (1969), and independently by Werbos (1974); however, it was Rumelhart et al. (1986) that brought the algorithm to people's attention. This spawned a decade of intense interest in these models.

# A Brief History of the Field

In 1987, Sejnowski and Rosenberg created the famous **NETtalk** system, that learned a mapping from English words to phonetic symbols which could be fed into a speech synthesizer.

An audio demo of the system as it learns over time can be found at https://www.youtube.com/watch?v=gakJlr3GecE. The systems starts by “babbling” and then gradually learns to pronounce English words.

# A Brief History of the Field

In 1989, Yann Le Cun and others created the famous LeNet system (http://yann.lecun.com/exdb/lenet/)

In 1992, the support vector machine was invented by Boser et al. SVMs provide similar prediction accuracy to neural networks while being considerably easier to train. This spawned a decade of interest in kernel methods in general.

# A Brief History of the Field

In 2002, Geoff Hinton invented the contrastive divergence training procedure, which provided a way, for the first time, to learn deep networks, by training one layer at a time in an unsupervised fashion. This in turn has spawned renewed interest in neural networks over the last few years.

# The Backpropagation Algorithm

The **backpropagation** algorithm computes the gradient vector of the negative log-likelihood (NLL) by applying the chain rule of calculus.

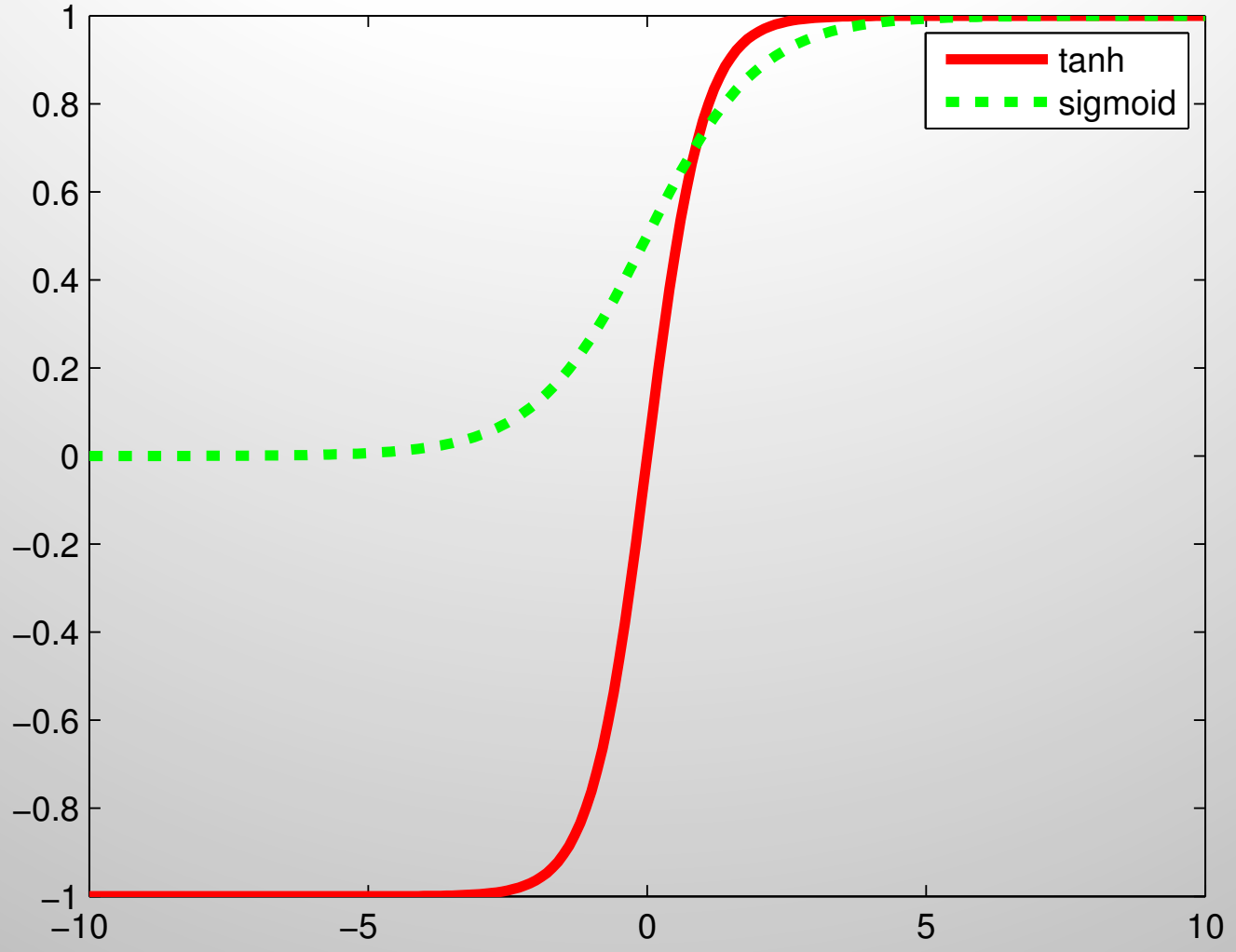
For notational simplicity, we shall assume a model with just one hidden layer. It is helpful to distinguish the pre- and post-synaptic values of a neuron, that is, before and after we apply the nonlinearity.

# The Backpropagation Algorithm

Let $\mathbf{x}_n$ be the $n^{\text{th}}$ input, $\boldsymbol{a}_n = \mathbf{V}\mathbf{x}_n$ be the pre-synaptic hidden layer, and $\boldsymbol{z}_n = g(\boldsymbol{a}_n)$ be the post-synaptic hidden layer, where $g$ is some activation (transfer) function.

We typically use $g(\mathbf{a}) = \text{sigm}(\mathbf{a})$, but we may also use $g(\mathbf{a}) = \tanh(\mathbf{a})$. When the input to sigm or tanh is a vector, we assume it is applied component-wise.

# The Backpropagation Algorithm




Two possible activation functions: tanh and sigm

# The Backpropagation Algorithm

We now convert this hidden layer to the output layer as follows. Let $\boldsymbol{b}_n = \mathbf{W}\boldsymbol{z}_n$ be the pre-synaptic output layer, and $\hat{\mathbf{y}}_n = h(\boldsymbol{b}_n)$ be the post-synaptic output layer, where $h$ is another nonlinearity.

We can write the overall model as follows:

$$\mathbf{x}_n \xrightarrow{\mathbf{V}} \mathbf{a}_n \xrightarrow{g} \mathbf{z}_n \xrightarrow{\mathbf{W}} \mathbf{b}_n \xrightarrow{h} \hat{\mathbf{y}}_n$$

# The Backpropagation Algorithm

In the regression case, with K outputs, the NLL is given by the squared error:

$$J(\boldsymbol{\theta}) = -\sum_n \sum_k (\hat{y}_{nk}(\boldsymbol{\theta}) - y_{nk})^2$$

In the classification case, with K classes, the NLL is given by the cross entropy:

$$J(\boldsymbol{\theta}) = -\sum_n \sum_k y_{nk} \log \hat{y}_{nk}(\boldsymbol{\theta})$$

# The Backpropagation Algorithm

Our task is to compute $\nabla_\theta J$. We will derive this for each $n$ separately; the overall gradient is obtained by summing over $n$.

Let us start by considering the output layer weights. We have

$$\nabla_{\mathbf{w}_k} J_n = \frac{\partial J_n}{\partial b_{nk}} \nabla_{\mathbf{w}_k} b_{nk} = \frac{\partial J_n}{\partial b_{nk}} \mathbf{z}_n$$

since $b_{nk} = \mathbf{w}_k^T \mathbf{z}_n$.

# The Backpropagation Algorithm

Assuming $h$ is the canonical link function for the output generalized linear model (GLM), then

$$\frac{\partial J_n}{\partial b_{nk}} \triangleq \delta_{nk}^w = (\hat{y}_{nk} - y_{nk})$$

which is the error signal.

# The Backpropagation Algorithm

So, the overall gradient is

$$\nabla_{\mathbf{w}_k} J_n = \delta_{nk}^{w} \mathbf{z}_n$$

which is:

pre-synaptic input to the output layer ($\mathbf{z}_n$) times the error signal ($\delta_{nk}^{w}$).

# The Backpropagation Algorithm

For the input layer weights, we have

$$\nabla_{\mathbf{v}_j} J_n = \frac{\partial J_n}{\partial a_{nj}} \nabla_{\mathbf{v}_j} a_{nj} \triangleq \delta_{nj}^{v} \mathbf{x}_n$$

where we exploited the fact that $a_{nj} = \mathbf{v}_k^T \boldsymbol{x}_n$.

# The Backpropagation Algorithm

All that remains is to compute the first level error signal $\delta_{nj}^{v}$. We have

$$\delta_{nj}^{v} = \frac{\partial J_n}{\partial a_{nj}} = \sum_{k=1}^{K} \frac{\partial J_n}{\partial b_{nk}} \frac{\partial b_{nk}}{\partial a_{nj}} = \sum_{k=1}^{K} \delta_{nk}^{w} \frac{\partial b_{nk}}{\partial a_{nj}}$$

# The Backpropagation Algorithm

Now

$$b_{nk} = \sum_j w_{kj} g(a_{nj})$$

so

$$\frac{\partial b_{nk}}{\partial a_{nj}} = w_{kj} g'(a_{nj})$$

where $g'(a) = \frac{d}{da} g(a)$.

# The Backpropagation Algorithm

For sigmoid units:

$$g'(a) = \frac{d}{da}\mathrm{sigm}(a) = \mathrm{sigm}(a)\big(1 - \mathrm{sigm}(a)\big),$$

and for tanh units:

$$g'(a) = \frac{d}{da}\tanh(a) = 1 - \tanh^2(a) = \mathrm{sech}^2(a)$$

Hence

$$\delta_{nj}^{v} = \sum_{k=1}^{K} \delta_{nk}^{w} w_{kj} g'(a_{nj})$$

# The Backpropagation Algorithm

Thus the layer 1 errors can be computed by passing the layer 2 errors back through the **W** matrix; hence the term “backpropagation”.

The key property is that we can compute the gradients locally: each node only needs to know about its immediate neighbors.

# The Backpropagation Algorithm

Putting it all together, we can compute all the gradients as follows:

- We first perform a forwards pass to compute $\mathbf{a}_n$, $\boldsymbol{z}_n$, $\mathbf{b}_n$, and $\widehat{\boldsymbol{y}}_n$.
- We then compute the error for the output layer, $\delta_{nk}^{w} = \widehat{\boldsymbol{y}}_n - \boldsymbol{y}_n$.
- We then pass $\delta_{nk}^{w}$ backwards through **W** to compute the error for the hidden layer, $\delta_{nk}^{v}$.
- We then compute the overall gradient as $\nabla_\theta J(\theta) = \sum_n [\delta_n^v \boldsymbol{x}_n, \delta_n^w \boldsymbol{z}_n]$.

# Convolutional Neural Networks

A form of MLP which is particularly well suited to 1d signals like speech or text, or 2d signals like images, is the **convolutional neural network**. This is an MLP in which the hidden units have local **receptive fields** (as in the primary visual cortex), and in which the weights are tied or shared across the image, in order to reduce the number of parameters.

# Convolutional Neural Networks

Intuitively, the effect of such spatial parameter tying is that any useful features that are "discovered" in some portion of the image can be re-used everywhere else without having to be independently learned. The resulting network then exhibits **translation invariance**, meaning it can classify patterns no matter where they occur inside the input image.

# Other Kinds of Neural Networks

Other network topologies are possible besides the ones we discussed. For example, we can have skip arcs that go directly from the input to the output, skipping the hidden layer; we can have sparse connections between the layers; etc. However, the MLP always requires that the weights form a directed acyclic graph.

# Other Kinds of Neural Networks

If we allow feedback connections, the model is known as a recurrent neural network; this defines a nonlinear dynamical system, but does not have a simple probabilistic interpretation. Such RNN models are currently the best approach for language modeling.

# Other Kinds of Neural Networks

If we allow symmetric connections between the hidden units, the model is known as a Hopfield network or associative memory; its probabilistic counterpart is known as a Boltzmann machine and can be used for unsupervised learning.