Feedforward Neural Networks

Carlos Valle

Departamento de Informática
Universidad Técnica Federico Santa María

cvalle@inf.utfsm.cl

October 19, 2018

Overview

Artificial Neural Networks

- ANN were inspired by scientists who attempt to answer questions such as:
  - What makes the human brain such a formidable machine in processing cognitive thought?
  - What is the nature of this thing called intelligence?
  - And, how do humans solve problems?
- There are many different theories and models for how the mind and brain work.

Connectionism

- One such theory, called connectionism, uses analogues of neurons and their connections together with the concepts of activation functions, and the ability to modify those connections to create learning algorithms.
- Now, ANNs are treated more abstractly, as a network of highly interconnected nonlinear computing elements.
- Problems of speech recognition, handwritten character recognition, face recognition, and robotics are important applications of ANNs.

Why neural networks?

- Essential motivation: automatic extraction of characteristics relevant to solve a learning task.

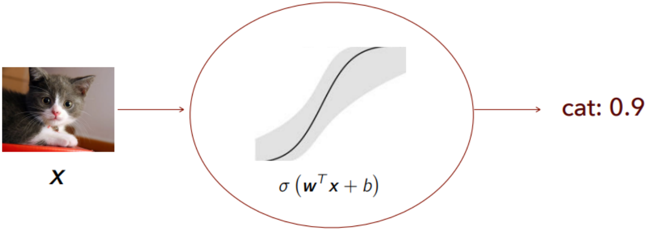


Figure: Logistic Regression Model

Why neural networks? (2)

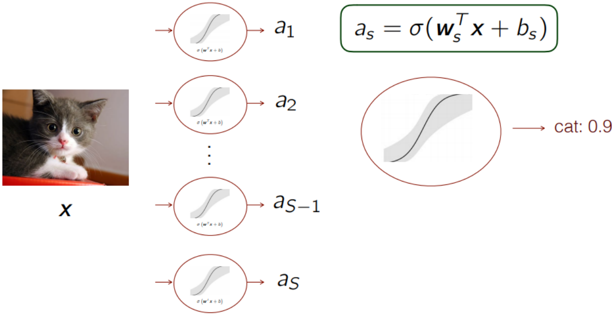
- Note that using the logistic regresion model, $f(\mathbf{x})$ is given by

$$f(\mathbf{x}) = \sigma(w^T\mathbf{x} + b) = \sigma\left(\sum_{i=1}^{I} w^{(i)}x^{(i)} + b\right)$$

- Thus, we would state that $f(\mathbf{x})$ depends on $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(I)}$.
- In practice, we do not know what the relevant attributes are for solve the problem by linear methods.

Why neural networks? (3)

- Idea: The determination of each attribute (representation) is itself same a learning problem.



$$a_s = \sigma(\mathbf{w}_s^T \mathbf{x} + b_s)$$

Neural Network Architecture

- The resulting model is given by

$$f(\mathbf{x}) = \sigma\left(\sum_{s=1}^{S} w_{out,s}\sigma\left(\mathbf{w}_s^T\mathbf{x} + b_s\right) + b_{out}\right)$$

- Equivalently,

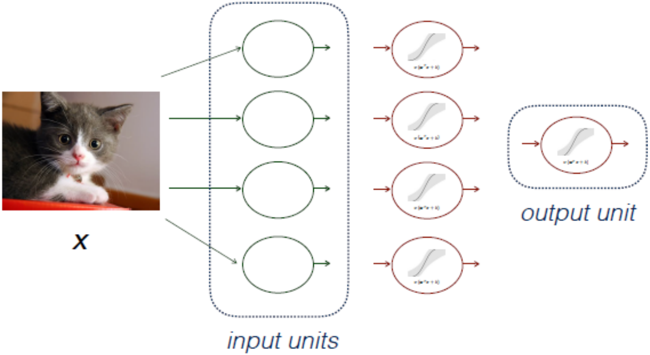
$$f(\mathbf{x}) = \sigma\left(w_{out}^T a + b_{out}\right),$$

- where $a = (a_1, \ldots, a_S)^T$
- And $a_s = \sigma(w_s^T\mathbf{x} + b_s)$
- Thus, this model has (N+1)(I+1) free parameters.

Neurons and Layers

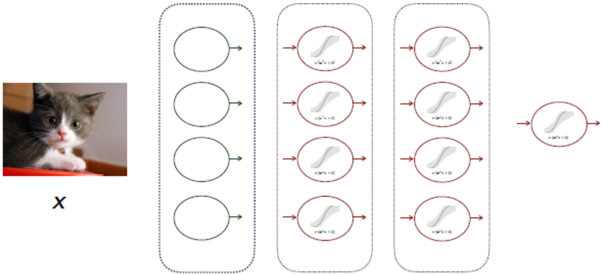
- Neurons: Each of the computation units used in the model (to learn an attribute) will be called neuron.
- Input Neurons: Each original feature feeds an specialized unit (neuron) called input neuron.
- The unit that produces the final output is called the output neuron.
- Hidden neurons are located between the input and output. They learn the underlying representation.

Neurons and Layers (2)



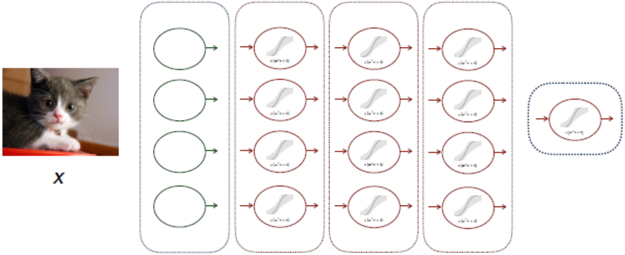
Neurons and Layers (3)

- Extending this idea, we may need to add a level additional processing aimed at learning the necessary attributes for learning the attributes necessary to learn the output.



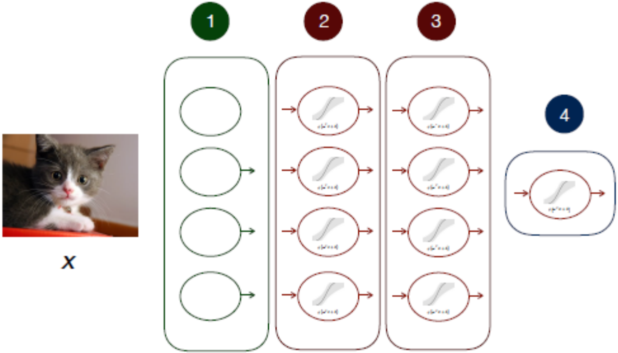
Neurons and Layers (4)

- And so on.



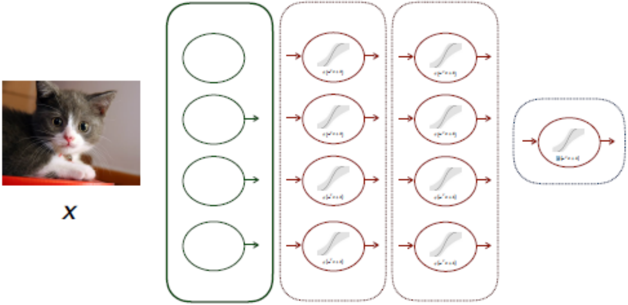
Neurons and Layers (5)

- We thus obtain the input layer (1), the output layer (4) and the hidden (2,3) layers.
- The traditional numbering is incremental, counting the input number as 1.



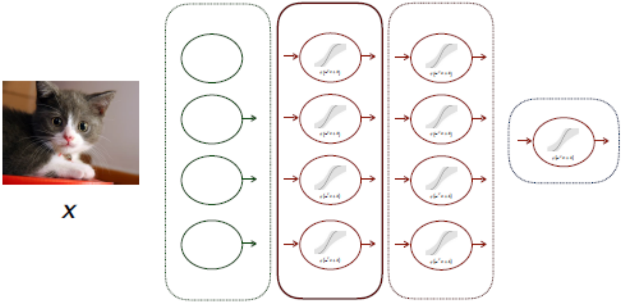
Layers = transformations

- Mathematically, each level computes a transformation of the representation obtained in the previous level.
- $a^{(1)} = \mathbf{x}$



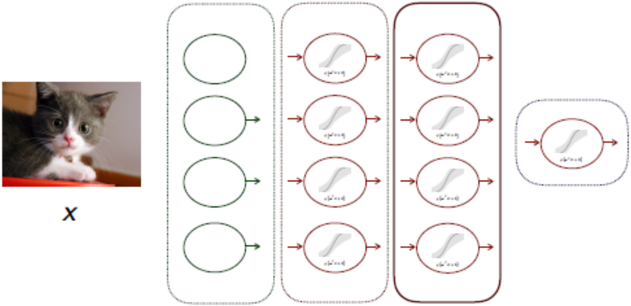
Layers = transformations (2)

- Mathematically, each level computes a transformation of the representation obtained in the previous level.
- $a^{(2)} = H^{(1)}(a^{(1)})$



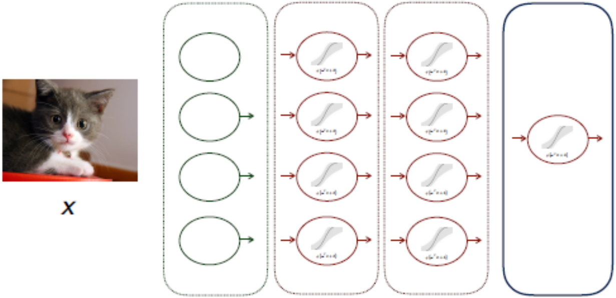
Layers = transformations (3)

- Mathematically, each level computes a transformation of the representation obtained in the previous level.
- $a^{(3)} = H^{(2)}(a^{(2)})$



Layers = transformations (4)

- Mathematically, each level computes a transformation of the representation obtained in the previous level.
- $a^{(4)} = H^{(3)}(a^{(3)})$



Layers = transformations (4)

- $a^{(\ell)} = (a_1^{(\ell)}, a_2^{(\ell)}, \ldots, a_{s_\ell}^{(\ell)})^T \in \mathbb{R}^{s_\ell}$
- Now we have $H(0) \circ H(1) \circ \cdots \circ H(L-1)$

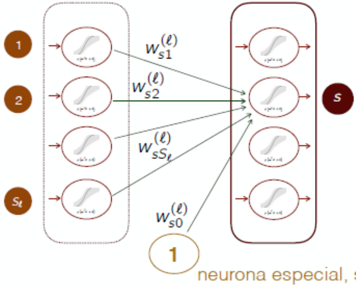
Transformations: Linear combinations + non-linear functions

- Each neuron linearly combines the attributes generated in the previous level and then transforms the total signal calculated using a non-linear function.
- $a^{(\ell+1)} = H^{(\ell)}(a^{(\ell)})$
- $w^{(\ell)} = (w_1^{(\ell)}, w_2^{(\ell)}, \dots, w_{s+1}^{(\ell)})^T \in \mathbb{R}^{s_{t-1} \times s_t}$
- $w_0^{(\ell)} = (w_{10}^{(\ell)}, w_{20}^{(\ell)}, \dots, w_{s+10}^{(\ell)})^T \in \mathbb{R}^{\ell+1}$
- $a^{(\ell+1)} = H^{(\ell)}(a^{(\ell)}) = \sigma(W^{(\ell)} a^{(\ell)} + w_0^{(\ell)})$

Transformations: Linear combinations + non-linear functions

- The transformation made by each neuron is determined by parameters that are interpreted as connection weights between the units.

$$\boldsymbol{a}^{(\ell+1)} = H^{(\ell)}(\boldsymbol{a}^{(\ell)})$$



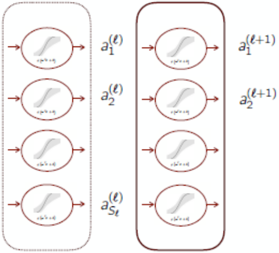
$$\boldsymbol{w}_s^{(\ell)} = (w_{s1}^{(\ell)}, w_{s2}^{(\ell)}, \ldots, w_{sS_\ell}^{(\ell)})^\top \in \mathbb{R}^{S_\ell}$$

$$\begin{aligned} a_s^{(\ell+1)} &= \sigma(\boldsymbol{w}_s^{(\ell)\top}\boldsymbol{a}^{(\ell)} + b_s^{(\ell)}) \\ &= \sigma\left(\sum_{t=1}^{S_\ell} w_{st}^{(\ell)} a_s^{(\ell)} + w_{s0}^{(\ell)}\right) \end{aligned}$$

Transformations: Linear combinations + non-linear functions

- In matrix form ...

$$\boldsymbol{a}^{(\ell+1)} = H^{(\ell)}(\boldsymbol{a}^{(\ell)})$$



$$\boldsymbol{W}^{(\ell)} = (\boldsymbol{w}_1^{(\ell)}, \boldsymbol{w}_2^{(\ell)}, \ldots, \boldsymbol{w}_{S_{\ell+1}}^{(\ell)})^\top \in \mathbb{R}^{S_{\ell+1} \times S_\ell}$$

$$\boldsymbol{w}_0^{(\ell)} = (w_{10}^{(\ell)}, w_{20}^{(\ell)}, \ldots, w_{S_{\ell+1}0}^{(\ell)})^\top \in \mathbb{R}^{S_{\ell+1}}$$

$$\boldsymbol{a}^{(\ell+1)} = H^{(\ell)}(\boldsymbol{a}^{(\ell)}) = \sigma(\boldsymbol{W}^{(\ell)}\boldsymbol{a}^{(\ell)} + \boldsymbol{w}_0^{(\ell)})$$

Composition of transformations (forward pass)

Algorithm 1 Batch gradient descent algorithm

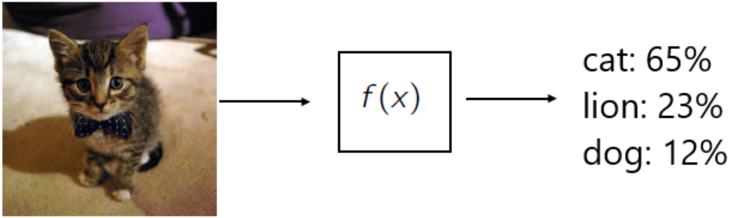
Require: $\mathbf{x}$

Ensure: $f_{ANN}(\mathbf{x})$

1: $a^{(1)} = \mathbf{x}$
2: **for** $l = 1$ to $L - 1$ **do**
3: $\quad a^{(\ell+1)} = \sigma(W^{(\ell)} a^{(\ell)} + w_0^{(\ell)})$
4: **end for**

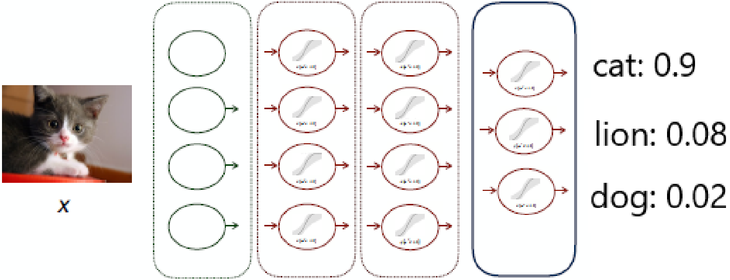
Extending for multi-class settings

- We might need that the number of neurons in the output layer be < 1.
- For example, in multi-class classification we would like to compute the probability of each class.



Extending for multi-class settings

- This solution could produce inconsistent results.
- Later, we will define special layers to solve this problem.



Hypothesis space of a Neural Network

Definition

Given a learning task, $f : \mathcal{X} \rightarrow \mathcal{Y}, \mathcal{X} \subset \mathbb{R}^I, \mathcal{Y} \subset \mathbb{R}^K$, a feed forward neural network (FFN) is a learning function in the hypothesis space $\mathcal{H}_{S_2}^{S_1} \circ \mathcal{H}_{S_3}^{S_2} \circ \cdots \circ \mathcal{H}_{S_{L-1}}^{S_L}$ with $S_1 = I$, $S_L = K$ and

$$\mathcal{H}_S^T = \{H : \mathbb{R}^S \rightarrow \mathbb{R}^T : H(a) = \sigma(Wa + w_0), W \in \mathbb{R}^{T \times S}, \mathbf{w}_0 \in \mathbb{R}^T\}.$$

L is called the number of layers or depth of the network. S_ℓ it is called the number of neurons in the layer ℓ.

How many neurons and layers?

Theorem (Universal approximation theorem, Cybenko (1989))

For any learning task $f : \mathcal{X} \to \mathcal{Y}, \mathcal{X} \subset [0,1]^I, \mathcal{Y} \subset \mathbb{R}$ *, where* f *is a continuous function. There exist a feed forward neural network with 3 layers such as* $\forall \varepsilon > 0$*:*

$$|f(\mathbf{x}) - f_{ANN}(\mathbf{x})| \leq \varepsilon$$

- In real scenarios, optimal values are highly dependent on the problem.
- Specifically depends on the feature and the size of the dataset.
- We will return to this problem later.

Parameters versus hyperparameters

- Parameters of the model: Its value is determined by training the model, that is, from the observed error on the training examples.
- Examples:
  - w^{ℓ}_{sr}: weight from neuron r in layer ℓ to neuron s in layer $\ell + 1$.
  - w^{ℓ}_{s0}: bias of the neuron r in layer ℓ.
  - w^{ℓ}_{s}: weight vector of neuron s in layer ℓ.
  - W^{ℓ}: matrix weights of layer ℓ.

Parameters versus hyperparameters

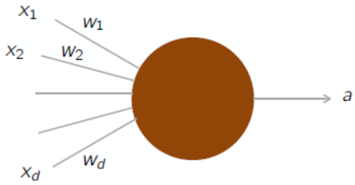
- Hyper-parameters of the model: In general, its value can not be determined from of the training data. An estimate of the future error (test) is needed.
- Examples:
  - L: weight from neuron r in layer ℓ to neuron s in layer $\ell + 1$.
  - S_ℓ: number of neurons of the layer ℓ.

Deep versus Shallow

- Before 2006 the preferred model had 1 hidden layer.
- Nowadays, recycling of attributes is the most popular idea.
  - Each neuron of a level can use all the attributes obtained in the previous level to work.
  - Each attribute generated by a neuron in a The layer can be used by all neurons in the next layer.
  - The deeper the network, the greater the recycling of attributes possible, i.e. it is possible to obtain more compact representations.
  - Layers as levels of abstraction to solve a problem: A greater number of layers of processing allows to build attributes of greater complexity from simpler attributes.

Activation functions

- Each neuron in the network learns a transformation (linear combination + non-linearity)
- $a = \sigma\left(\sum_{i=1}^{I} w_i x_i - b\right)$
- It is possible to model / choose the latter with different criteria



Activation Linear

- $a = \sigma\left(\sum_{i=1}^{I} w_i x_i - b\right)$
- $\sigma(\xi) = \xi$

Figure: Linear function

Activation sigmoidal

- $a = \sigma\left(\sum_{i=1}^{I} w_i x_i - b\right)$
- $\sigma(\xi) = \frac{1}{1+e-\xi}$

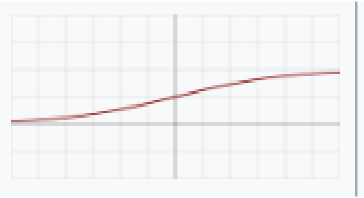


Figure: Sigmoid function

Activation tanh

- $a = \sigma\left(\sum_{i=1}^{I} w_i x_i - b\right)$
- $\sigma(\xi) = \frac{2}{1+e^{-2\xi}} - 1$

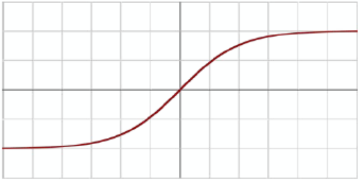


Figure: Tanh function

Activation ReLu (Rectifier Linear)

- $a = \sigma\left(\sum_{i=1}^{I} w_i x_i - b\right)$
- $\sigma(\xi) = \begin{cases} \xi & \xi \geq 0 \\ 0 & \xi < 0 \end{cases}$

Figure: ReLu function

Activation ReLu (Rectifier Linear)

- When an instance $\mathbf{x}$ is processed, a subset of the network units is activated. And the response is linear in the subset (path) of active neurons.
- We can view this as a classification tree with a exponentially large number of leaves and linear predictors on them.

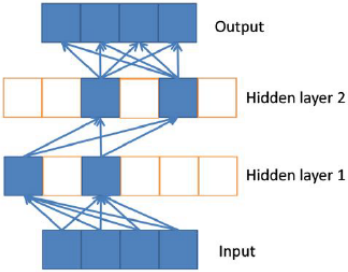


Figure: Linear function

Activation Softplus

- This approximation for the ReLu is differentiable (everywhere).

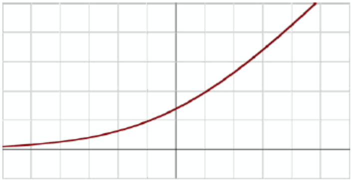


Figure: Softplus function

- Universal Approximation Theorem (Hornik, 1991)
- Let Σ be the family of functions $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ such that
  - σ is not constant.
  - σ is bounded.
  - σ is continuous.
- Kurt Hornik. Approximation Capabilities of Multilayer Feedforward Networks. Neural Networks, Vol. 4, pp. 251-257. 1991

Universal Approximation Theorem (Hornik, 1991)

Theorem (Universal approximation theorem. Hornik, 1991))

For any learning task $f : \mathcal{X} \to \mathcal{Y}, \mathcal{X} \subset \mathbb{R}^I, \mathcal{Y} \subset \mathbb{R}$ *, where* f *is a continuous function. There exist a feed forward neural network with 3 layers with* $\sigma \in \Sigma$*, such as* $\forall \varepsilon > 0$ *:*

$$|f(\mathbf{x}) - f_{ANN}(\mathbf{x})| \leq \varepsilon$$

Activation Functions

- Recent research: It is possible to use certain unbounded functions and still maintain the universal approximation property?
- S. Sonuda, N. Murata. Neural network with unbounded activation functions is universal approximator. 2015.
- In particular, the property is maintained for the 2 most popular transfer functions: ReLu and Softplus.
- In practice, the activation function can differ from layer to layer.

Activation Functions for the output layer

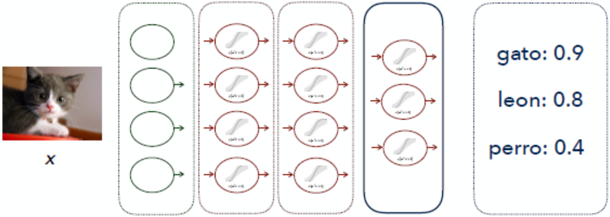
- Last layer (output) must use activation functions appropriate for the learning task that you want to solve.
- For example, in regression problems, where the output is continuous (position, speed, price, temperature, etc), an output to the range $[0, 1]$ may be too restrictive.
- The choice of the transfer function for the output layer is very related to the choice of an error function and with the interpretation / use we make of the network's response.
- In regression settings the linear activation function is the most used.

Activation Functions for the output layer

- In binary classification problems, the output layer is commonly compounded of 1 unit with sigmoid function.
- Thus, it models
$$p(y = c_1|\mathbf{x}) = f(\mathbf{x})$$
- And subsequently
$$p(y = c_2|\mathbf{x}) = 1 - f(\mathbf{x})$$

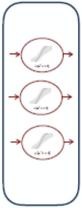
Activation Functions for the output layer

- In multi-class problems $(K > 2)$, the choice of sigmoidal functions for each unit of the output layer might lead to inconsistent results.



From neurons to layers

- Using K logistic functions, the output layer would be obtained as:



$$\boxed{\mathbf{a}^L = \sigma(\boldsymbol{W}^{(L-1)\top}\mathbf{a}^{(L-1)} + \mathbf{w}_0^{(L)})}$$

$$a_1^L = \sigma(\mathbf{w}_1^{(L-1)\top}\mathbf{a}^{(L-1)} + w_{10}^{(L)}) = \frac{1}{1+e^{-(\mathbf{w}_1^{(L-1)\top}\mathbf{a}^{(L-1)}+w_{10}^{(L)})}}$$

$$a_2^L = \sigma(\mathbf{w}_2^{(L-1)\top}\mathbf{a}^{(L-1)} + w_{20}^{(L)}) = \frac{1}{1+e^{-(\mathbf{w}_2^{(L-1)\top}\mathbf{a}^{(L-1)}+w_{20}^{(L)})}}$$

$$\vdots$$

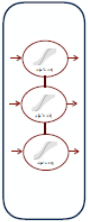
$$a_{S_L}^L = \sigma(\mathbf{w}_{S_L}^{(L-1)\top}\mathbf{a}^{(L-1)} + w_{S_L 0}^{(L)}) = \frac{1}{1+e^{-(\mathbf{w}_{S_L}^{(L-1)\top}\mathbf{a}^{(L-1)}+w_{S_L 0}^{(L)})}}$$

The softmax layer

- In classification problems with multiple categories (K¿ 2) the default choice is called softmax layer.

$$\boxed{\mathbf{a}^{L} = g_{\text{softmax}}(\mathbf{W}^{(L-1)\mathsf{T}}\mathbf{a}^{(L-1)} + \mathbf{w}_0^{(L)})}$$

Layer transformation



$$a_s^{(L)} = \frac{exp\left(\mathbf{w}_s^{(L-1)\mathsf{T}}\mathbf{a}^{(L-1)} + \mathbf{w}_{s0}^{(L-1)}\right)}{\sum_t exp\left(\mathbf{w}_t^{(L-1)\mathsf{T}}\mathbf{a}^{(L-1)} + \mathbf{w}_{t0}^{(L-1)}\right)}$$

$$\sum_s a_s^{(L)} = 1$$

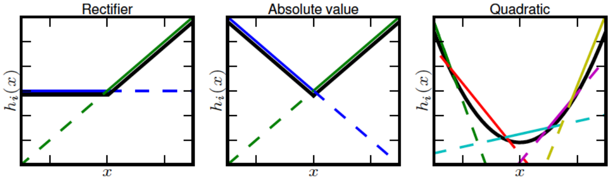
This normalization of the outputs ensures that the probabilities always sum 1.

The maxout layer

- Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A. C., and Bengio, Y. (2013). Maxout Networks. ICML (3), 28, 1319-1327.
- This layer is commonly used as a hidden layer.
- The maxout layer is given by
- $a^{(\ell+1)} = \sigma_{maxout}\left(W^{\ell^T} a^{(\ell)} + w_0^{(\ell)}\right),$
- where $\sigma_{maxout}\left(\mathbf{p}\right) = \max_k p_k$

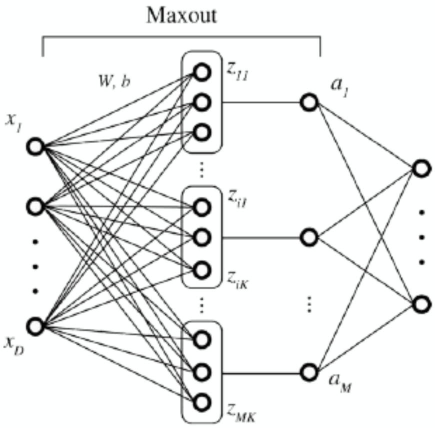
The maxout layer

- Each neuron in the layer learns a linear combination of its input signal.
- A maxout layer can be seen as a linear model by parts, which learns the shape of the activation function needed.



The maxout layer

- Each neuron in the layer learns a linear combination of its input signal.



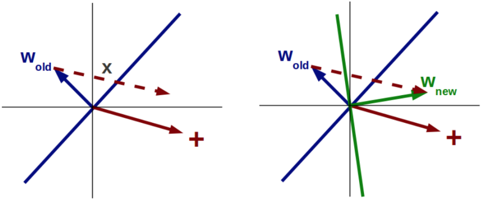
Primal Perceptron algorithm

Algorithm 2 Primal perceptron algorithm

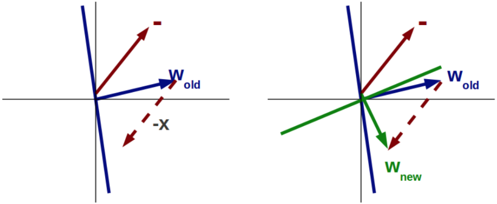
Given a training set S
$w_0 \leftarrow 0; b_0 \leftarrow 0; p \leftarrow 0$
repeat
  for $m = 1$ to M **do**
    if $y_m(\mathbf{w}^T \mathbf{x}_m + b_p) \leq 0$ **then**
      $w_{p+1} \leftarrow w_p + \eta y_m x_m$
      $b_{p+1} \leftarrow b_p + \eta y_m$
      $p \leftarrow p + 1$
    end if
  end for
until No mistakes are made within the loop
Output: $(\mathbf{w}_p, b_p)$

What does the weight update is doing?

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \mathbf{x}$$



$$\mathbf{w}_{new} = \mathbf{w}_{old} - \mathbf{x}$$



How to adjust the weights?

- Revisiting out DNN (Deep Neural Network) architecture, we need an optimization method that determines the best values for the network weights.
- $$\min_W R_{emp}(f_W) = \min_W \frac{1}{M} \sum_{m=1}^{M} \ell(f_{ANN}(\mathbf{x}_m), y_m)$$

- Recall that if we have a binary classification problem and we use a sigmoidal output, the output of the network is directly interpretable as:

$$f(\mathbf{x}) = p(y = c_1|x) = p(y = 1|x)$$

- The log-likelihood of the probabilistic model is:

$$\begin{aligned}
\mathcal{L}(S) = \ln \prod_m p(y_m|x_m) &= \sum_m \ln p(y_m|x_m) \\
&= \sum_{m:y_m=1} \ln f(\mathbf{x}_m) + \sum_{m:y_m=0} \ln(1 - f(\mathbf{x}_m)) \\
&= \sum_m y_m \ln f(\mathbf{x}_m) + (1 - y_m) \ln(1 - f(\mathbf{x}_m))
\end{aligned}$$

How to choose ℓ?

- Maximizing $\mathcal{L}(S)$ is equivalent to:

$$\begin{aligned}
\max \mathcal{L}(S) &= \max \sum_m y_m \ln f(\mathbf{x}_m) + (1 - y_m) \ln(1 - f(\mathbf{x}_m)) \\
&= \min \sum_m \ell(f(\mathbf{x}_m), y_m)
\end{aligned}$$

- where $\ell(f(\mathbf{x}_m), y_m) = -y_m \ln f(\mathbf{x}_m) + (1 - y_m) \ln(1 - f(\mathbf{x}_m))$
- ℓ is called the cross-entropy loss

How to choose ℓ?

- We can extend the cross-entropy loss to multi-class problem with K classes
- Using $T(y) = ((T(y))_1, \ldots, (T(y))_K)^T$, where $(T(y))_k = 1$ If $y = k$, 0 otherwise.
- Thus, $(T(y))^{(k)} = I(y = k)$ and $(T(y))^{(K)} = 1 - \sum_{k=1}^{K-1} (T(y))^{(k)}$,
- where $I(\cdot)$ is the indicator function.
- Here, $f(\mathbf{x}_m) = (f(\mathbf{x}_m)^{(1)}, \ldots, f(\mathbf{x}_m)^{(K)})$ is a vector of size K.
- Here, $\ell(f(\mathbf{x}_m), T(y_m) =$ $-\sum_k \left[T(y_m)^{(k)} \ln f(\mathbf{x}_m)^{(k)} + (1 - T(y_m)^{(k)}) \ln(1 - f(\mathbf{x}_m)^{(k)})\right]$

How to choose ℓ?

- For regression settings we model $f(\mathbf{x}) = E(y|x)$
- If we assume that $E(y|x)$ is normally distributed

$$
\begin{aligned}
\mathcal{L}(S) = \ln \prod_m p(y_m|x_m) &= \sum_m \ln p(y_m|x_m) \\
&= \sum_m \ln \left(\text{const} \cdot \exp\left(-\frac{y_m - f(\mathbf{x}_m)}{2} \right) \right) \\
&= \sum_m -(y_m - f(\mathbf{x}_m))^2 + \text{const}
\end{aligned}
$$

How to choose ℓ?

- Maximizing $\mathcal{L}(S)$ is equivalent to:

$$\begin{aligned} \max \mathcal{L}(S) &= \max -(y_m - f(\mathbf{x}_m))^2 + \text{const} \\ &= \min \sum_m (y_m - f(\mathbf{x}_m))^2 \\ &= \sum_m \ell(f(\mathbf{x}_m), y_m) \end{aligned}$$

- where $\ell(f(\mathbf{x}_m), y_m) = (y_m - f(\mathbf{x}_m))^2$

Back-propagation algorithm

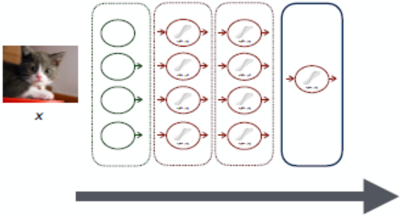
Algorithm 3 Back-propagation algorithm

```
Initialize the network weights
while stop criteria not met do
    for each example (x_m, y_m) do
        Compute forward pass(x_m, y_m)
        Compute the error E = E(x_m, y_m)
        Compute backward pass(E)
    end for
end while
```

Forward pass

Forward Pass

$a^{(1)} = x$;
for $\ell = 1, \ldots, L-1$ **do**
  $a^{(\ell+1)} = \sigma(W^{(\ell)\top} a^{(\ell)} + w_0^{(\ell)})$;
end
return $a^{(L)}$



Forward Pass
= Predict

Computing the error for the output layer

- For regression settings:

$$
\begin{aligned}
E = E(\mathbf{x}_m, y_m) &= \frac{1}{2}\sum_{k=1}^{K}(a_k^{(L)} - y_k)^2 \\
= E(\mathbf{x}_m, y_m) &= \frac{1}{2}\sum_{k=1}^{K}(f_{ANN}(\mathbf{x})_k - y_k)^2
\end{aligned}
$$

- Thus,

$$
\frac{\partial E}{\partial a_S^{(L)}} = (a_s^{(L)} - y_s) = (f_{ANN}(\mathbf{x})_S - y_S)
$$

Computing the error for the output layer

- For classification settings:

$$E(\mathbf{x}_m, y_m) = \frac{1}{2}\sum_{k=1}^{K}(y_k \ln a_k^{(L)} + (1-y_k)\ln(1-a_k^{(L)}))$$

- Thus,

$$\frac{\partial E}{\partial a_S^{(L)}} = \frac{(y_s - a_s^{(L)})}{a_s^{(L)}(1-a_s^{(L)})}$$

How to implement the backward pass?

- We will use gradient descent + chain rule

  1: **for** $t = 1, \ldots, T$ **do**
  2: $\quad W^{(t)} \longleftarrow W^{(t-1)} - \eta \frac{\partial E}{\partial W}$
  3: **end for**

- $W^{(t)} \xrightarrow[t \longrightarrow \infty]{} W^{*}$
- So, we need the partial derivatives with respect to each weight:
- For each layer ℓ,
- $w_s^{(\ell)} \longleftarrow w_s^{(\ell)} - \eta \frac{\partial E}{\partial w_s^{(\ell)}}$
- $w_{s0}^{(\ell)} \longleftarrow w_{s0}^{(\ell)} - \eta \frac{\partial E}{\partial w_{s0}^{(\ell)}}$

Any questions?