Machine Learning - MT 2017
14, 15, 16. Neural Networks

Varun Kanade

University of Oxford

November 8, 13 & 15, 2017

Announcements

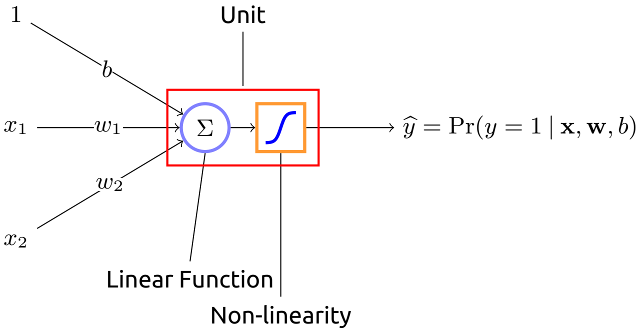
- Problem Sheet 3 due this week
- Problem Sheet 4 posted soon (class in Week 8)
- Practical 2 next week: Compare NBC & LR
- All subsequent lectures in LTA

Outline

Today, we'll study feedforward neural networks

- Multi-layer perceptrons
- Classification or regression settings
- Backpropagation to compute gradients
- Brief introduction to tensorflow and MNIST

Artificial Neuron : Logistic Regression

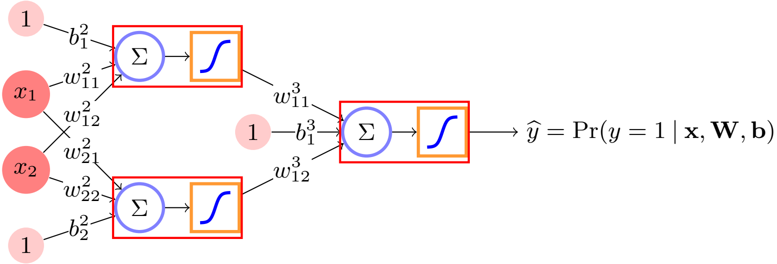


- A unit in a neural network computes a linear function of its input and is then composed with a non-linear activation function
- For logistic regression, the non-linear activation function is the sigmoid

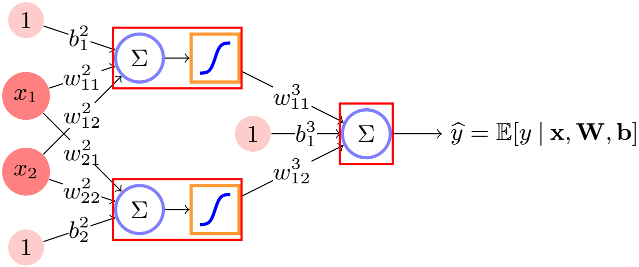
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- The separating surface is linear

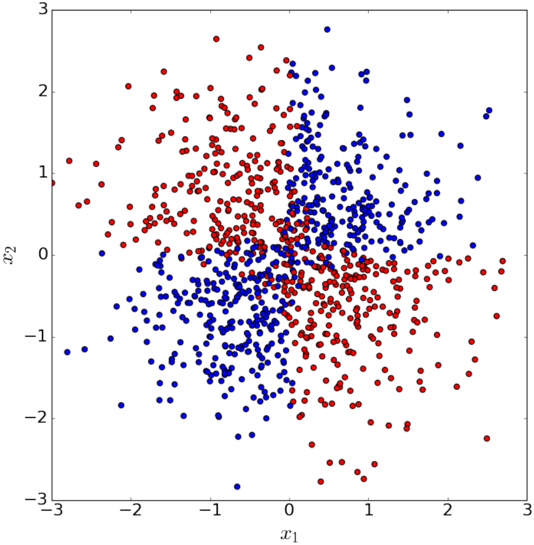
Multilayer Perceptron (MLP) : Classification



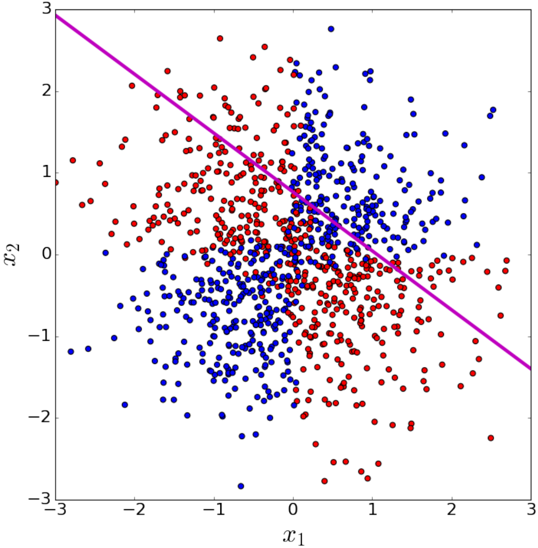
Multilayer Perceptron (MLP) : Regression



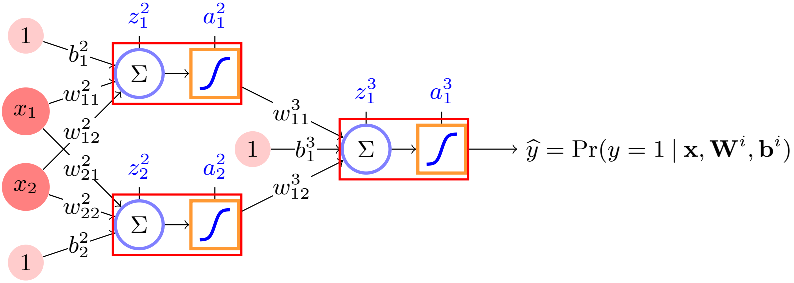
A Toy Example



Logistic Regression Fails Badly



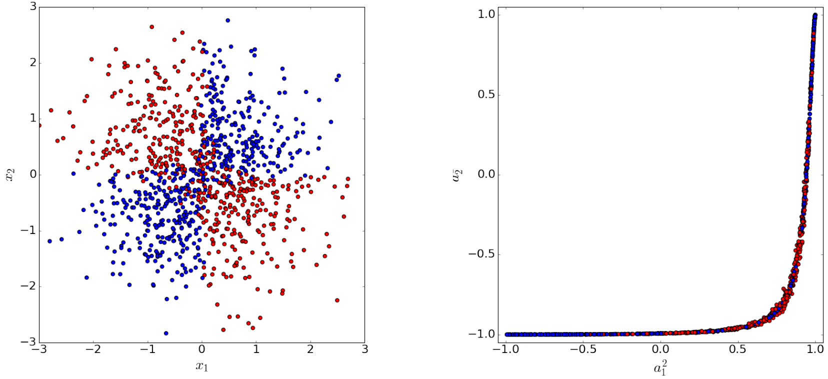
Solve using MLP



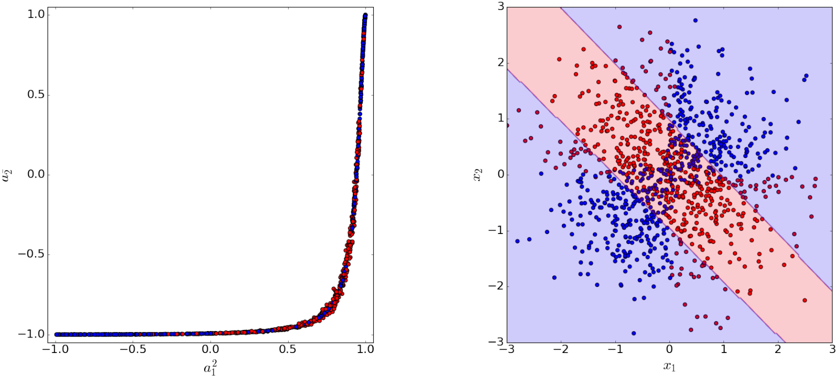
Let us use the notation:

$$\begin{aligned} \mathbf{a}^1 &= \mathbf{z}^1 = \mathbf{x} \\ \mathbf{z}^2 &= \mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2 \\ \mathbf{a}^2 &= \tanh(\mathbf{z}^2) \\ \mathbf{z}^3 &= \mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3 \\ \mathbf{y} = \mathbf{a}^3 &= \sigma(\mathbf{z}^3) \end{aligned}$$

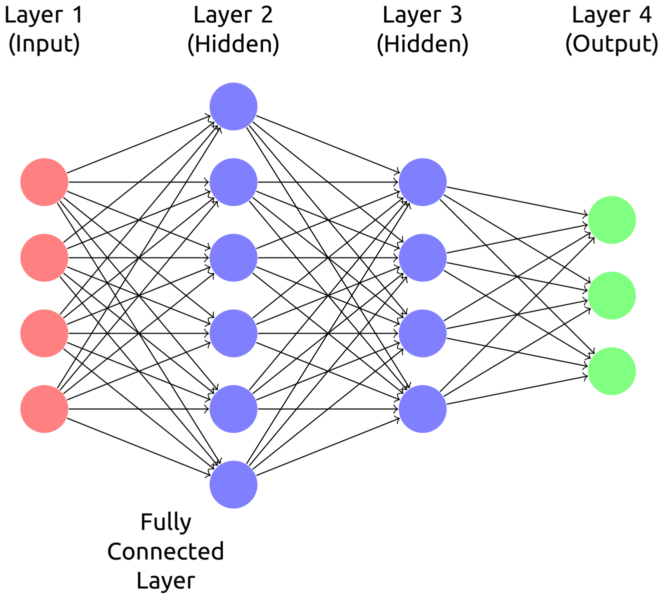
Scatterplot Comparison (x_1, x_2) vs (a_1^2, a_2^2)



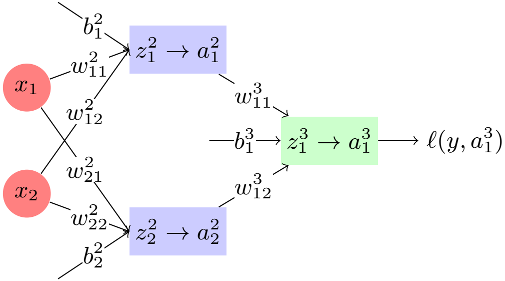
Decision Boundary of the Neural Net



Feedforward Neural Networks



Computing Gradients on Toy Example



Want the derivatives

$$\frac{\partial \ell}{\partial w_{11}^2}, \frac{\partial \ell}{\partial w_{12}^2}$$
$$\frac{\partial \ell}{\partial w_{21}^2}, \frac{\partial \ell}{\partial w_{22}^2}$$
$$\frac{\partial \ell}{\partial w_{11}^3}, \frac{\partial \ell}{\partial w_{12}^3}$$
$$\frac{\partial \ell}{\partial b_1^2}, \frac{\partial \ell}{\partial b_2^2}, \frac{\partial \ell}{\partial b_1^3}$$

Would suffice to compute $\frac{\partial \ell}{\partial z_1^3}$, $\frac{\partial \ell}{\partial z_1^2}$, $\frac{\partial \ell}{\partial z_2^2}$

Computing Gradients on Toy Example

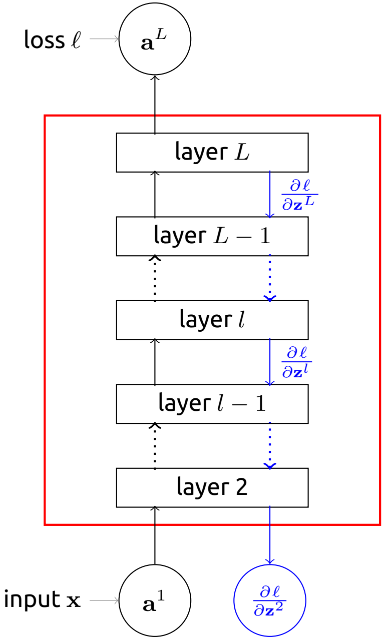
Let us compute the following:

1. $\frac{\partial \ell}{\partial a_1^3} = -\frac{y}{a_1^3} + \frac{1-y}{1-a_1^3} = \frac{a_1^3 - y}{a_1^3(1-a_1^3)}$
2. $\frac{\partial a^3}{\partial z_1^3} = a_1^3 \cdot (1 - a_1^3)$
3. $\frac{\partial z_1^3}{\partial \mathbf{a}^2} = [w_{11}^3, w_{12}^3]$
4. $\frac{\partial \mathbf{a}^2}{\partial \mathbf{z}^2} = \begin{bmatrix} 1 - \tanh^2(z_1^2) & 0 \\ 0 & 1 - \tanh^2(z_2^2) \end{bmatrix}$

Then we can calculate

$$\frac{\partial \ell}{\partial z_1^3} = \frac{\partial \ell}{\partial a_1^3} \cdot \frac{\partial a_1^3}{\partial z_1^3} = a_1^3 - y$$

$$\frac{\partial \ell}{\partial \mathbf{z}^2} = \left(\frac{\partial \ell}{\partial a_1^3} \cdot \frac{\partial a_1^3}{\partial z_1^3} \right) \cdot \frac{\partial z_1^3}{\partial \mathbf{a}^2} \cdot \frac{\partial \mathbf{a}^2}{\partial \mathbf{z}^2} = \frac{\partial \ell}{\partial z_1^3} \cdot \frac{\partial z_1^3}{\partial \mathbf{a}^2} \cdot \frac{\partial \mathbf{a}^2}{\partial \mathbf{z}^2}$$


loss ℓ
$\mathbf{a}^L$
layer L
$\frac{\partial \ell}{\partial \mathbf{z}^L}$
layer $L-1$
layer l
$\frac{\partial \ell}{\partial \mathbf{z}^l}$
layer $l-1$
layer 2
input $\mathbf{x}$
$\mathbf{a}^1$
$\frac{\partial \ell}{\partial \mathbf{z}^2}$

Each layer consists of a linear function and non-linear activation

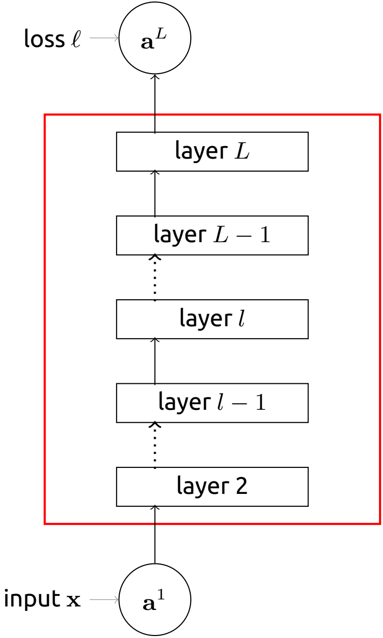
Layer l consists of the following:

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$$
$$\mathbf{a}^l = f_l(\mathbf{z}^l)$$

where f_l is the non-linear activation in layer l.

If there are n_l units in layer l, then $\mathbf{W}^l$ is $n_l \times n_{l-1}$

Backward pass to compute derivatives



Forward Equations

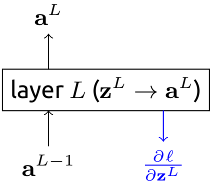
(1) $\mathbf{a}^1 = \mathbf{x}$ (input)

(2) $\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$

(3) $\mathbf{a}^l = f_l(\mathbf{z}^l)$

(4) $\ell(\mathbf{a}^L, y)$

Output Layer



$\mathbf{a}^L$

layer L ($\mathbf{z}^L \rightarrow \mathbf{a}^L$)

$\mathbf{a}^{L-1}$

$\frac{\partial \ell}{\partial \mathbf{z}^L}$

$$\mathbf{z}^L = \mathbf{W}^L \mathbf{a}^{L-1} + \mathbf{b}^L$$

$$\mathbf{a}^L = f_L(\mathbf{z}^L)$$

Loss: $\ell(y, \mathbf{a}^L)$

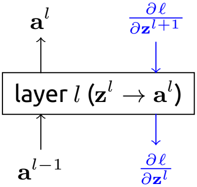
$$\frac{\partial \ell}{\partial \mathbf{z}^L} = \frac{\partial \ell}{\partial \mathbf{a}^L} \cdot \frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}$$

If there are n_L (output) units in layer L, then $\frac{\partial \ell}{\partial \mathbf{a}^L}$ and $\frac{\partial \ell}{\partial \mathbf{z}^L}$ are row vectors with n_L elements and $\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}$ is the $n_L \times n_L$ Jacobian matrix:

$$\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L} = \begin{bmatrix} \frac{\partial a_1^L}{\partial z_1^L} & \frac{\partial a_1^L}{\partial z_2^L} & \cdots & \frac{\partial a_1^L}{\partial z_{n_L}^L} \\ \frac{\partial a_2^L}{\partial z_1^L} & \frac{\partial a_2^L}{\partial z_2^L} & \cdots & \frac{\partial a_2^L}{\partial z_{n_L}^L} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial a_{n_L}^L}{\partial z_1^L} & \frac{\partial a_{n_L}^L}{\partial z_2^L} & \cdots & \frac{\partial a_{n_L}^L}{\partial z_{n_L}^L} \end{bmatrix}$$

If f_L is applied element-wise, *e.g.*, sigmoid then this matrix is diagonal

Back Propagation



$\mathbf{a}^l$

$\frac{\partial \ell}{\partial \mathbf{z}^{l+1}}$

layer l ($\mathbf{z}^l \rightarrow \mathbf{a}^l$)

$\mathbf{a}^{l-1}$

$\frac{\partial \ell}{\partial \mathbf{z}^{l}}$

$\mathbf{a}^l$ (the inputs into layer $l+1$)

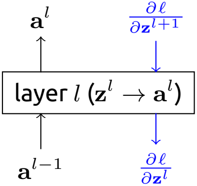
$\mathbf{z}^{l+1} = \mathbf{W}^{l+1}\mathbf{a}^l + \mathbf{b}^{l+1}$ ($w^{l+1}_{j,k}$ weight on connection from k^{th} unit in layer l to j^{th} unit in layer $l+1$)

$\mathbf{a}^l = f(\mathbf{z}^l)$ (f is a non-linearity)

$\frac{\partial \ell}{\partial \mathbf{z}^{l+1}}$ (derivative passed from layer above)

$$\begin{aligned}
\frac{\partial \ell}{\partial \mathbf{z}^{l}} &= \frac{\partial \ell}{\partial \mathbf{z}^{l+1}} \cdot \frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^{l}} \\
&= \frac{\partial \ell}{\partial \mathbf{z}^{l+1}} \cdot \frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{a}^{l}} \cdot \frac{\partial \mathbf{a}^{l}}{\partial \mathbf{z}^{l}} \\
&= \frac{\partial \ell}{\partial \mathbf{z}^{l+1}} \cdot \mathbf{W}^{l+1} \cdot \frac{\partial \mathbf{a}^{l}}{\partial \mathbf{z}^{l}}
\end{aligned}$$

Gradients with respect to parameters



$\mathbf{a}^l$

$\frac{\partial \ell}{\partial \mathbf{z}^{l+1}}$

layer l ($\mathbf{z}^l \to \mathbf{a}^l$)

$\mathbf{a}^{l-1}$

$\frac{\partial \ell}{\partial \mathbf{z}^{l}}$

$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$ ($w^l_{j,k}$ weight on connection from k^{th} unit in layer l-1 to j^{th} unit in layer l)

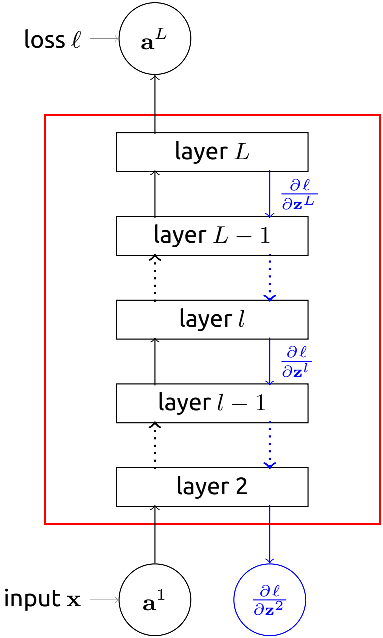
$\frac{\partial \ell}{\partial \mathbf{z}^l}$ (obtained using backpropagation)

Consider $\frac{\partial \ell}{\partial w^l_{ij}} = \frac{\partial \ell}{\partial z^l_i} \cdot \frac{\partial z^l_i}{\partial w^l_{ij}} = \frac{\partial \ell}{\partial z^l_i} \cdot a^{l-1}_j$

$\frac{\partial \ell}{\partial b^l_i} = \frac{\partial \ell}{\partial z^l_i}$

More succinctly, we may write: $\frac{\partial \ell}{\partial \mathbf{W}^l} = \left(a^{l-1} \frac{\partial \ell}{\partial \mathbf{z}^l} \right)^{\mathsf{T}}$

$\frac{\partial \ell}{\partial \mathbf{b}^l} = \frac{\partial \ell}{\partial \mathbf{z}^l}$


loss ℓ
$\mathbf{a}^L$
layer L
$\frac{\partial \ell}{\partial \mathbf{z}^L}$
layer $L-1$
layer l
$\frac{\partial \ell}{\partial \mathbf{z}^l}$
layer $l-1$
layer 2
input $\mathbf{x}$
$\mathbf{a}^1$
$\frac{\partial \ell}{\partial \mathbf{z}^2}$

Forward Equations

(1) $\mathbf{a}^1 = \mathbf{x}$ (input)

(2) $\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l$

(3) $\mathbf{a}^l = f_l(\mathbf{z}^l)$

(4) $\ell(\mathbf{a}^L, y)$

Back-propagation Equations

(1) Compute $\frac{\partial \ell}{\partial \mathbf{z}^L} = \frac{\partial \ell}{\partial \mathbf{a}^L} \cdot \frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}$

(2) $\frac{\partial \ell}{\partial \mathbf{z}^l} = \frac{\partial \ell}{\partial \mathbf{z}^{l+1}} \cdot \mathbf{W}^{l+1} \cdot \frac{\partial \mathbf{a}^l}{\partial \mathbf{z}^l}$

(3) $\frac{\partial \ell}{\partial \mathbf{W}^l} = \left(\mathbf{a}^{l-1} \frac{\partial \ell}{\partial \mathbf{z}^l} \right)^{\mathsf{T}}$

(4) $\frac{\partial \ell}{\partial \mathbf{b}^l} = \frac{\partial \ell}{\partial \mathbf{z}^l}$

Computational Questions

What is the running time to compute the gradient for a single data point?

- As many matrix multiplications as there are fully connected layers
- Performed twice during forward and backward pass

What is the space requirement?

- Need to store vectors $\mathbf{a}^l$, $\mathbf{z}^l$, and $\frac{\partial \ell}{\partial \mathbf{z}^l}$ for each layer

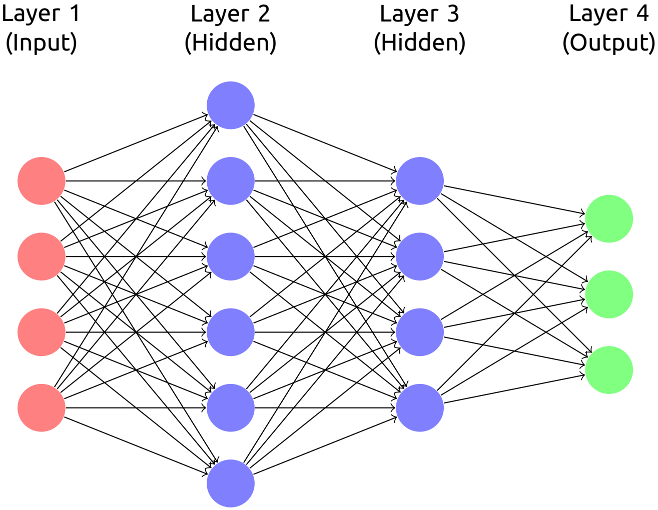
Can we process multiple examples together?

- Yes, if we minibatch, we perform tensor operations
- Make sure that all parameters fit in GPU memory

Training Deep Neural Networks

- Back-propagation gives gradient
- Stochastic gradient descent is the method of choice
- Regularisation
  - How do we add ℓ_1 or ℓ_2 regularisation?
  - Don't regularise bias terms
- How about convergence?
- What did we learn in the last 10 years, that we didn't know in the 80s?

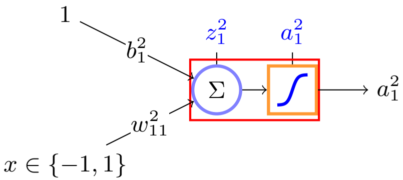
Training Feedforward Deep Networks



Why do we get non-convex optimisation problem?

All units in a layer are symmetric, hence invariant to permutations

A toy example



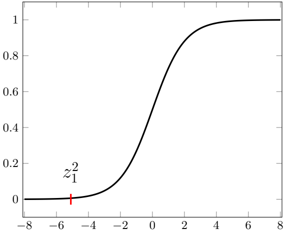
Target is $y = \frac{1-x}{2}$

Squared Loss Function

$$\ell(a_1^2, y) = (a_1^2 - y)^2$$

$$\frac{\partial \ell}{\partial z_1^2} = 2(a_1^2 - y) \cdot \frac{\partial a_1^2}{\partial z_1^2} = 2(a_1^2 - y)\sigma'(z_1^2)$$

If $x = -1, w_{11}^2 \approx 5, b_1^2 \approx 0$, then $\sigma'(z_1^2) \approx 0$

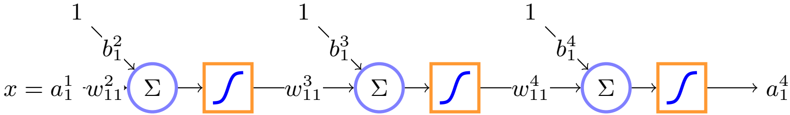


Cross-Entropy Loss Function

$$\ell(a_1^2, y) = -(y \log a_1^2 + (1 - y) \log(1 - a_1^2))$$

$$\frac{\partial \ell}{\partial z_1^2} = \frac{a_1^2 - y}{a_1^2(1 - a_1^2)} \cdot \frac{\partial a_1^2}{\partial z_1^2} = (a_1^2 - y)$$

Propagating Gradients Backwards



- Cross entropy loss: $\ell(a_1^4, y) = -(y \log a_1^4 + (1-y)\log(1-a_1^4))$
- $\frac{\partial \ell}{\partial z_1^4} = a_1^4 - y$
- $\frac{\partial \ell}{\partial z_1^3} = \frac{\partial \ell}{\partial z_1^4} \cdot \frac{\partial z_1^4}{\partial a_1^3} \cdot \frac{\partial a_1^3}{\partial z_1^3} = (a_1^4 - y) \cdot w_{11}^4 \cdot \sigma'(z_1^3)$
- $\frac{\partial \ell}{\partial z_1^2} = \frac{\partial \ell}{\partial z_1^3} \cdot \frac{\partial z_1^3}{\partial a_1^2} \cdot \frac{\partial a_1^2}{\partial z_1^3} = (a_1^4 - y) \cdot w_{11}^4 \cdot \sigma'(z_1^3) \cdot w_{11}^3 \cdot \sigma'(z_1^2)$
- Saturation: When the output of an artificial neuron is in the 'flat' part, *e.g.*, where $\sigma'(z) \approx 0$ for sigmoid
- Vanishing Gradient Problem: Multiplying several $\sigma'(z_i^l)$ together makes the gradient ≈ 0, when we have a large number of layers
- For example, when using sigmoid activation, $\sigma'(z) \in [0, 1/4]$

Avoiding Saturation

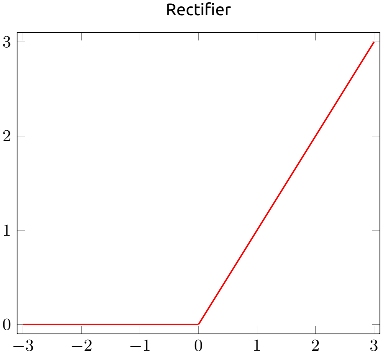
Use *rectified linear units*

Rectifier non-linearity
$f(z) = \max(0, z)$

Rectified Linear Unit (ReLU)
$\max(0, \mathbf{a} \cdot \mathbf{w} + b)$

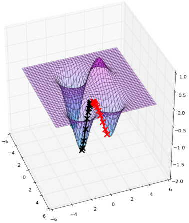
You can also use $f(z) = |z|$

Other variants
leaky ReLUs, parametric ReLUs



Initialising Weights and Biases

Initialising is important when minimising non-convex functions. We may get very different results depending on where we start the optimisation.



Suppose we were using a *sigmoid unit*, how would you initialise the weights?

- Suppose $z = \sum_{i=1}^{D} w_i a_i$
- E.g., choose $w_i \in [-\frac{1}{\sqrt{D}}, \frac{1}{\sqrt{D}}]$ at random

What if it were a ReLU unit?

- You can initialise similarly

How about the biases?

- For sigmoid, can use 0 or a random value around 0
- For ReLU, should use a small positive constant

Avoiding Overfitting

Deep Neural Networks have a lot of parameters

- Fully connected layers with $n_1, n_2, .., n_L$ units have at least $n_1 n_2 + n_2 n_3 + \cdots + n_{L-1} n_L$ parameters

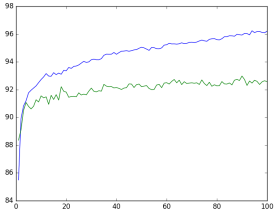
- For Problem Sheet 4, you will be asked to train an MLP for digit recognition with 2 million parameters and only 60,000 training images

- For image detection, one of the most famous models, the neural net used by Krizhevsky, Sutskever, Hinton (2012) has 60 *million* parameters and 1.2 *million* training images

- How do we prevent deep neural networks from overfitting?

Early Stopping

Maintain validation set and stop training when error on validation set stops decreasing.



What are the computational costs?

- Need to compute validation error
- Can do this every few iterations to reduce overhead

What are the advantages?

- If validation error flattens, or starts increasing can stop optimisation
- Prevents overfitting

See paper by Hardt, Recht and Singer (2015)

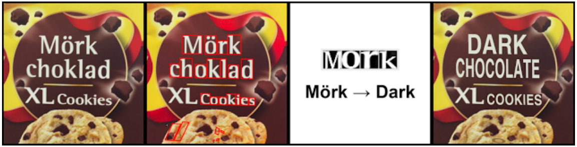
Add Data: Modified Data

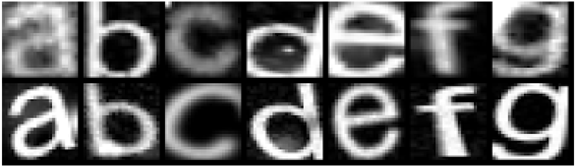
Typically, getting additional data is either impossible or expensive

Fake the data!

Images can be translated slight, rotated slightly, change of brightness, etc.

Google Offline Translate trained on entirely fake data!



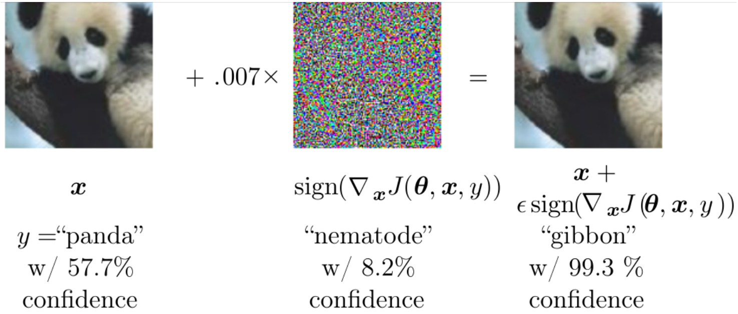


Google Research Blog

Add Data: Adversarial Training

Take trained (or partially trained model)

Create examples by modifications “imperceptible to the human eye”, but where the model fails


$+ .007 \times$
$=$
$\boldsymbol{x}$
$y =$“panda”
w/ 57.7%
confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“nematode”
w/ 8.2%
confidence
$\boldsymbol{x} +$
$\epsilon \, \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
“gibbon”
w/ 99.3 %
confidence

Szegedy *et al.* and Goodfellow *et al.*

Other Ideas to Reduce Overfitting

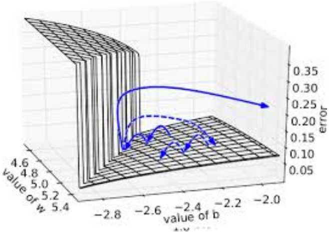
Hard constraints on weights

Gradient Clipping

Inject noise into the system

Enforce sparsity in the neural network

Unsupervised Pre-training



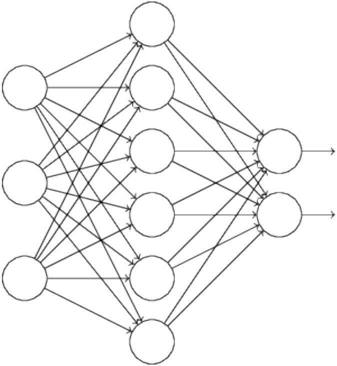
(Bengio *et al.*)

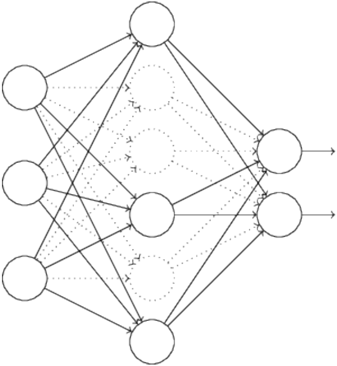
Bagging (Bootstrap Aggregation)

Bagging (Leo Breiman - 1994)

- Given dataset $\mathcal{D} = \langle (\mathbf{x}_i, y_i) \rangle_{i=1}^{N}$, sample $\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_k$ of size N from $\mathcal{D}$ with replacement
- Train classifiers $f_1, \ldots, f_k$ on $\mathcal{D}_1, \ldots, \mathcal{D}_k$
- When predicting use majority (or average if using regression)
- Clearly this approach is not practical for deep networks

Dropout

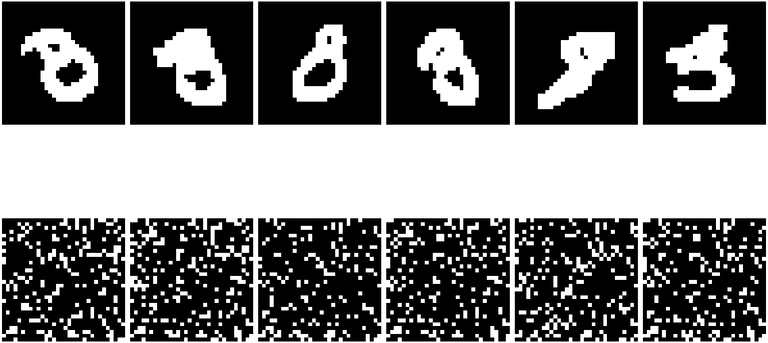




- For input x each hidden unit with probability $1/2$ independently
- Every input, will have a potentially different mask
- Potentially exponentially different models, but have "same weights"
- After training whole network is used by halving all the weights

Srivastava, Hinton, Krizhevsky, 2014

Errors Made by MLP for Digit Recognition

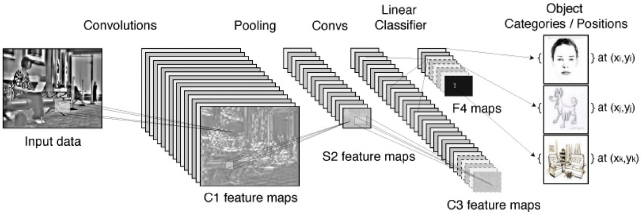


Avoiding Overfitting

- Use parameter sharing a.k.a weight tying in the model
- Exploit invariances to translation, rotation, *etc.*
- Exploit locality in images, audio, text, *etc.*
- Convolutional Neural Networks (convnets)

Convolutional Neural Networks (convnets)

(Fukushima, LeCun, Hinton 1980s)



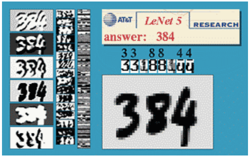
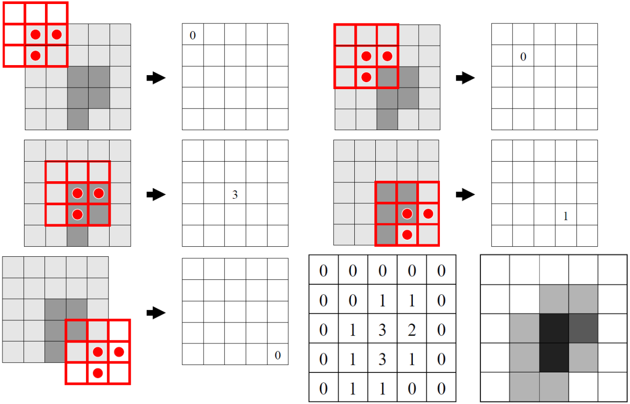


Image Convolution



0	0	0	0	0
0	0	1	1	0
0	1	3	2	0
0	1	3	1	0
0	1	1	0	0

Source: L. W. Kheng

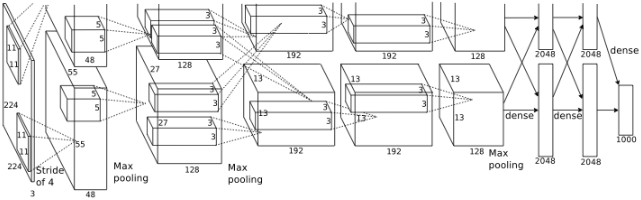
Convolution

In general, a convolution filter f is a tensor of dimension $W_f \times H_f \times F_l$, where F_l is the number of channels in the previous layer

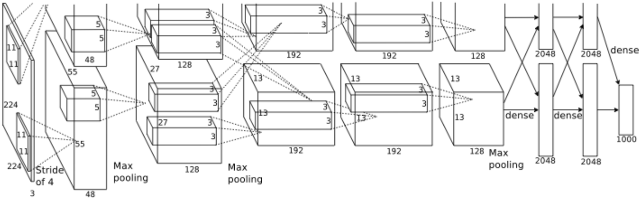
Strides in x and y directions dictate which convolutions are computed to obtain the next layer

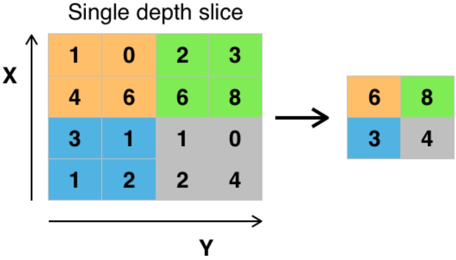
Zero-padding can be used if required to adjust layer sizes and boundaries

Typically, a convolution layer will have a large number of filters, the number of channels in the next layer will be the same as the number of filters used



Source: Krizhevsky, Sutskever, Hinton (2012)

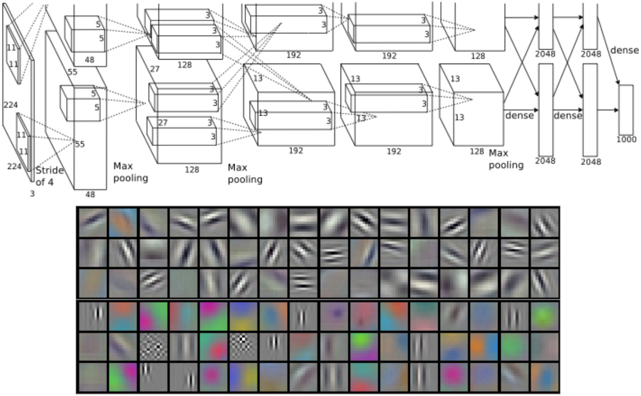



Single depth slice

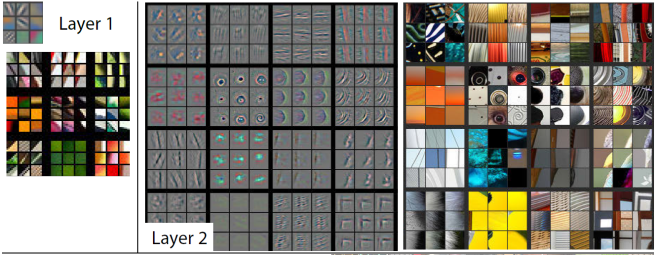
1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

6	8
3	4

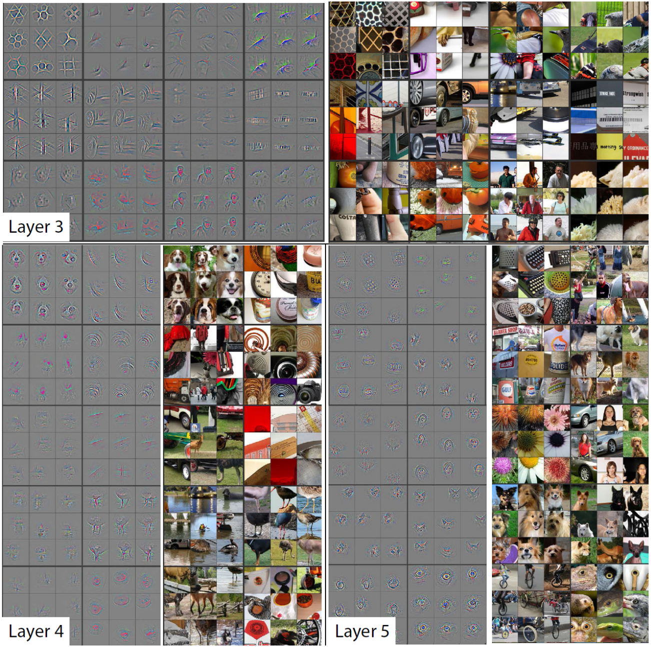
Sources: Krizhevsky, Sutskever, Hinton (2012); Wikipedia



Source: Krizhevsky, Sutskever, Hinton (2012)



Source: Zeiler and Fergus (2013)



Source: Zeiler and Fergus (2013)

Convolutional Layer

Suppose that there is no zero padding and strides in both directions are 1

$$z^{l+1}_{i',j',f'} = b_{f'} + \sum_{i=1}^{W_{f'}} \sum_{j=1}^{H_{f'}} \sum_{f=1}^{F_l} a^l_{i'+i-1,j'+j-1,f} w^{l+1,f'}_{i,j,f}$$

$$\frac{\partial z^{l+1}_{i',j',f'}}{\partial w^{l+1,f'}_{i,j,f}} = a^l_{i'+i-1,j'+j-1,f}$$

$$\frac{\partial \ell}{\partial w^{l+1,f'}_{i,j,f}} = \sum_{i',j'} \frac{\partial \ell}{\partial z^{l+1}_{i',j',f'}} \cdot a^l_{i'+i-1,j'+j-1,f}$$

Convolutional Layer

Suppose that there is no zero padding and strides in both directions are 1

$$z^{l+1}_{i',j',f'} = b_{f'} + \sum_{i=1}^{W_{f'}} \sum_{j=1}^{H_{f'}} \sum_{f=1}^{F_l} a^{l}_{i'+i-1,j'+j-1,f} w^{l+1,f'}_{i,j,f}$$

$$\frac{\partial z^{l+1}_{i',j',f'}}{\partial a^{l}_{i,j,f}} = w^{l+1,f'}_{i-i'+1,j-j'+1,f}$$

$$\frac{\partial \ell}{\partial a^{l}_{i,j,f}} = \sum_{i',j',f'} \frac{\partial \ell}{\partial z^{l+1}_{i',j',f'}} \cdot w^{l+1,f'}_{i-i'+1,j-j'+1,f}$$

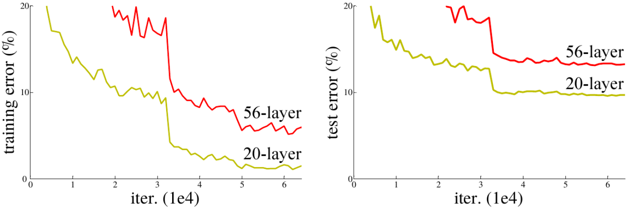
Max-Pooling Layer

Let $\Omega(i', j')$ be the set of (i, j) pairs in the previous layer that are involved in the maxpool

$$s_{i',j'}^{l+1} = \max_{i,j \in \Omega(i',j')} a_{i,j}^{l}$$

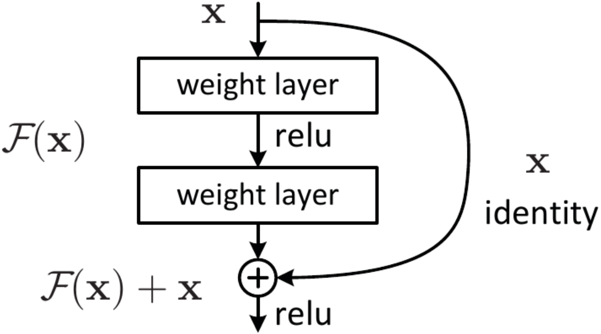
$$\frac{\partial s_{i',j'}^{l+1}}{\partial a_{i,j}^{l}} = \mathbb{I}\left((i,j) = \operatorname*{argmax}_{\tilde{i},\tilde{j} \in \Omega(i',j')} a_{\tilde{i},\tilde{j}}^{l} \right)$$

ResNets : Why more layers can give worse models



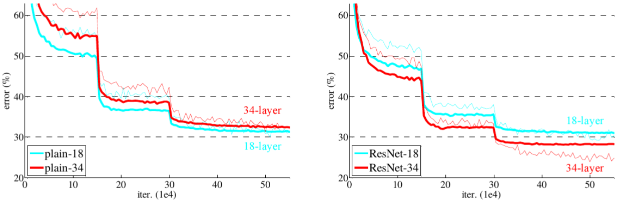
Source: He, Zhang, Ren and Sun (2015)

ResNets Architecture



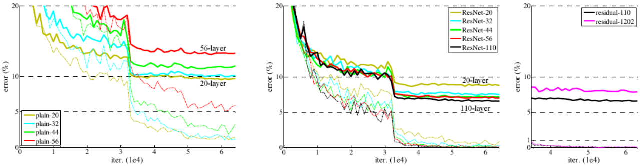
Source: He, Zhang, Ren and Sun (2015)

ResNets Results



Source: He, Zhang, Ren and Sun (2015)

ResNets Results



Source: He, Zhang, Ren and Sun (2015)

Next Week

- Practial will be about training neural networks on MNIST dataset
- Time permitting, implement one problem on the sheet in tensorflow
- Start Unsupervised Learning
- Revise eigenvectors, eigenvalues (Problem 5 on Sheet 3)