STELLENBOSCH UNIVERSITY **TECHPRENEURSHIP CENTRE**

MODULE M13: Introduction to Deep Learning

LAUNCHLAB
STELLENBOSCH UNIVERSITY

UNIVERSITEIT
iYUNIVESITHI
STELLENBOSCH
UNIVERSITY
100
1918·2018

launchlab.co.za

Who am I?

- Shane Josias: josias@sun.ac.za
- Junior lecturer (Applied Maths and School for DS and CT)
- PhD candidate - Machine learning with a computer vision focus
- Previously a software engineer in telecommunications

UNIVERSITEIT iYUNIVESITHI STELLENBOSCH UNIVERSITY 100 1918·2018

saam vorentoe · masiye phambili · forward together

SCHOOL FOR DATA SCIENCE AND COMPUTATIONAL THINKING

Applied Mathematics

Department of Mathematical Science

Intro to deep learning

- We will cover:
  - Applications of deep learning
  - What it means to “learn” a model.
  - Multilayer perceptron (fully connected neural nets)
  - Convolutions
  - Optimising a neural network
  - Practical example from scratch
- Some slides are adapted from Applied Math 792 (Computer Vision) given by Prof. Willie Brink

Deep learning use-cases

- Face recognition
- Image classification (crop health)
- Medical diagnosis (imaging)
- Cars: drivable area, lane keeping
- Speech recognition
- Machine translation
- Ads, search, social recommendations

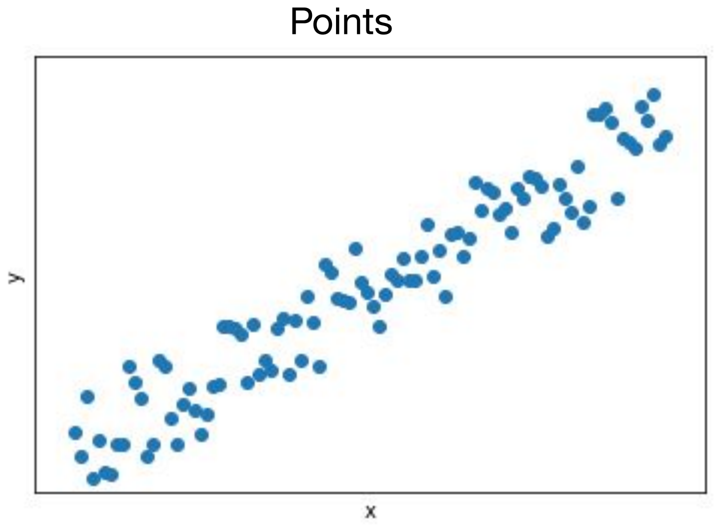
- What: fit powerful models to data
- How: NNs + optimisation
- Why: Data, hardware, community, tools, investment

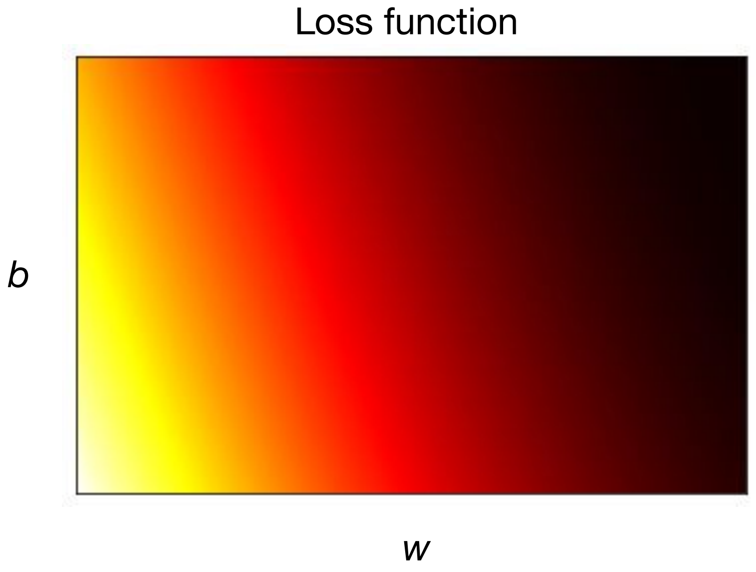
Linear regression

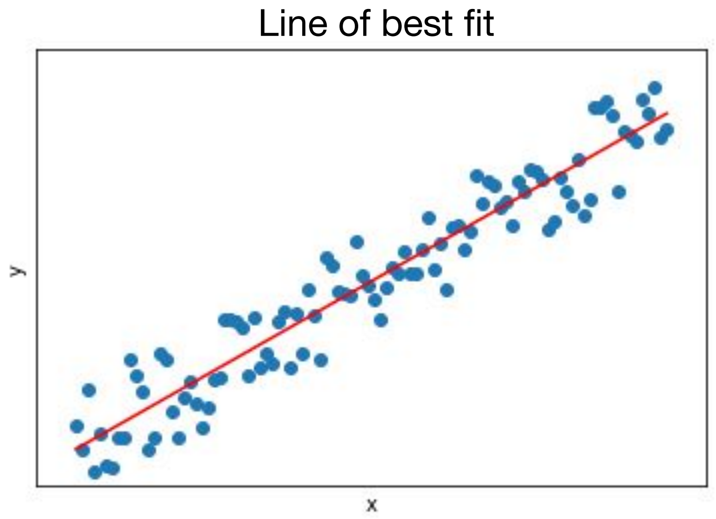
Model: $y = wx + b$ Training data: $(x_1, y_1), \ldots, (x_N, y_N)$

Parameters: $\mathbf{w} = [w \ b]^T$

Loss function: $L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} (wx_i + b - y_i)^2$

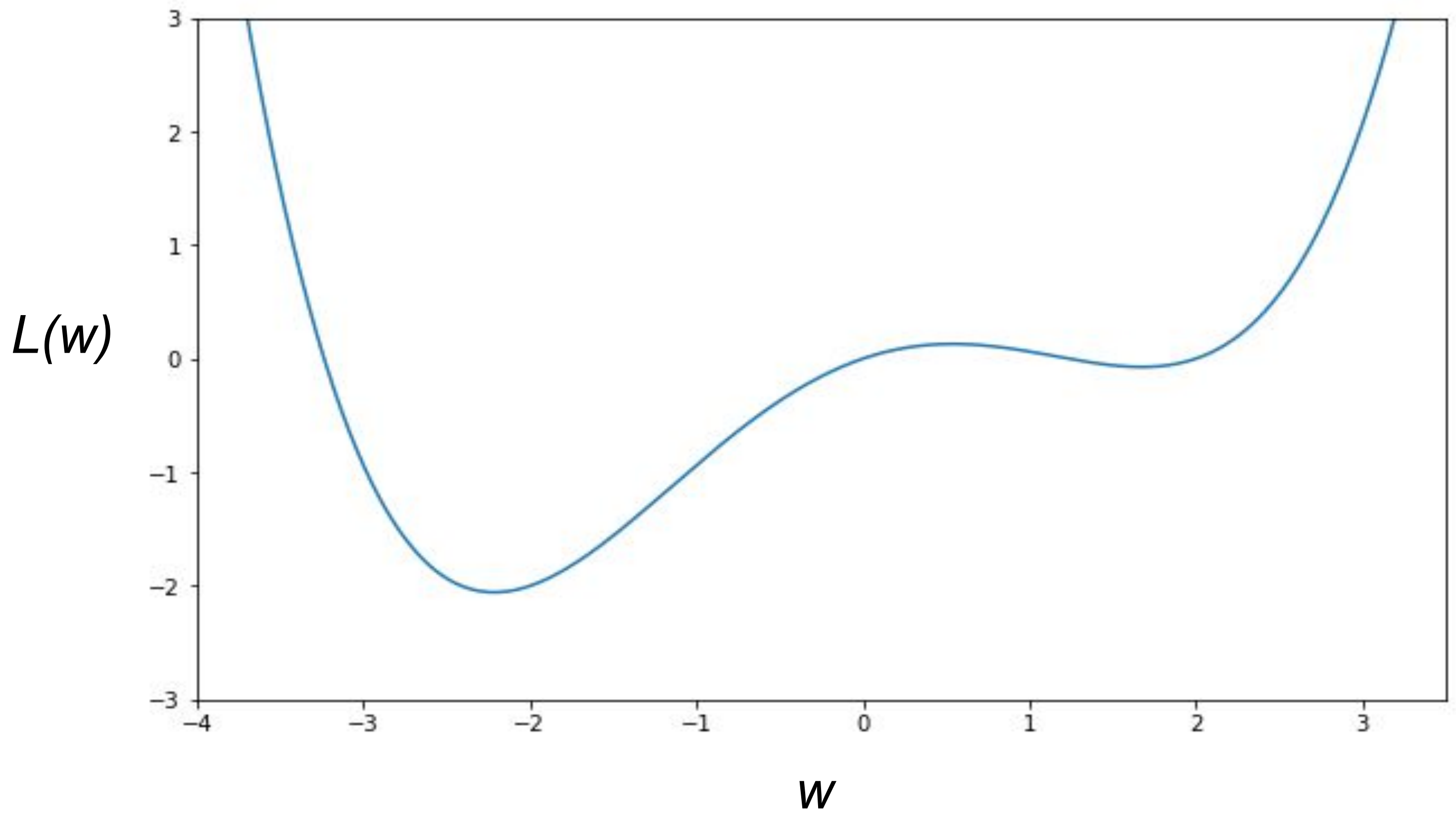






Gradient descent

Let's consider an objective function *L* over one parameter *w*:



We move in directions opposite to the gradient:

- If the gradient is positive, decrease the value of *w*
- If the gradient is negative, increase the value of w

Gradient descent

Loss must be differentiable w.r.t. all the model parameters

Gradient of the loss: $$\nabla L = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_n} \right]^T$$

Initialise $\mathbf{w}_0$

Iterate $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla L(\mathbf{w}_t)$

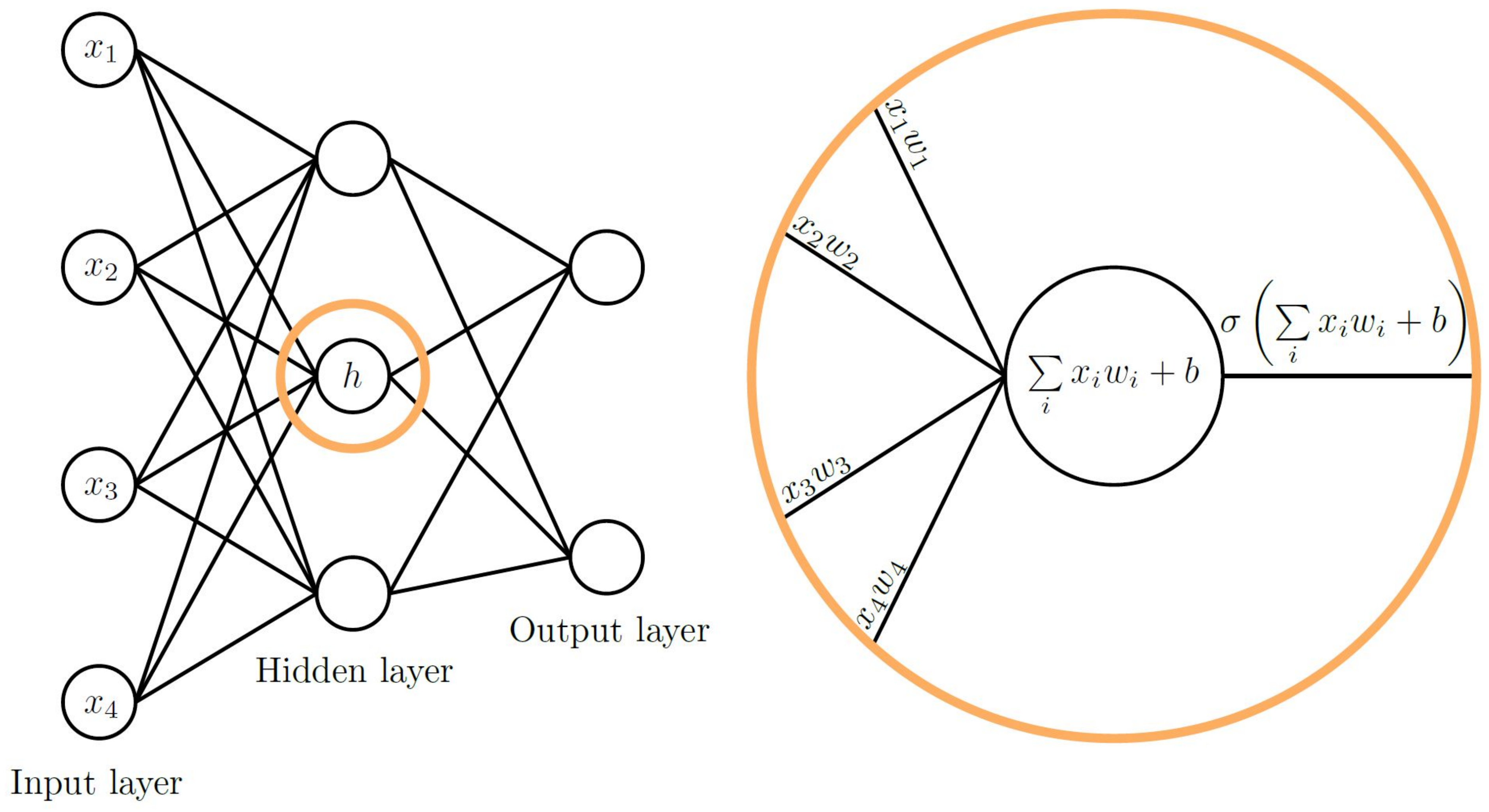
For a suitable learning rate, we can converge to a local minimum.

Feature engineering

- Converting raw input (e.g. pixels) into a more informative or discriminative feature vector (e.g. bag-of-visual-words)
  - Domain knowledge might be needed
- Feature selection strategies
  - Remove features with low variance (PCA)
  - Remove features that are highly correlated to others (not informative)
- Learning features:
  - With enough data, the machine can learn an optimal feature representation (NNs)

Multilayer perceptron

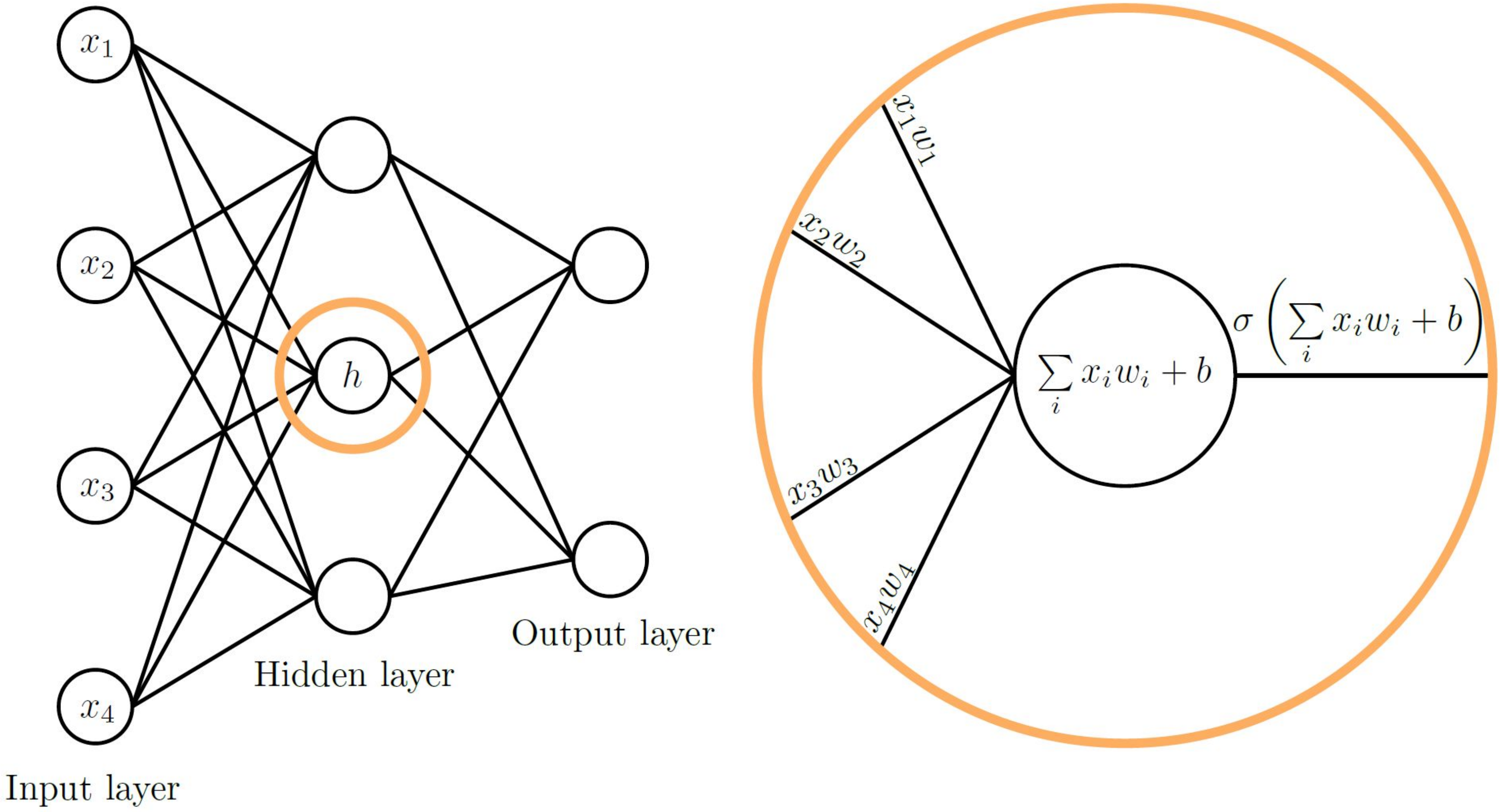
- Dot product between input **x** and **w**
- Same as doing matrix multiplication W**x**
- Apply activation function for nonlinearity
  - Sigmoid
  - Hyperbolic tangent
  - Rectified linear unit



Multilayer perceptron

Layer composition

- Feed outputs from one layer to another layer
- Complex functions as compositions of simple ones



Multilayer perceptron

Universal approximation theorem

> A network with a single, finite layer can represent any continuous function to an arbitrary degree of precision.

However:

- the layer can be impractically large, and in practice may fail to learn properly.
- adding layers increases complexity and expressiveness of the model while keeping total number of neurons required relatively low.

Softmax output

We use softmax to turn output of final layer into positive numbers that sum to 1.

If the output for one input sample is a vector with elements $z_1, z_2, \ldots, z_C$ then

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^{C} e^{z_j}}, \quad k = 1, 2, ..., C$$

Since the output for one sample is a vector of confidence scores, we one-hot encode labels from the training set:

0	0	1	0	0	0

one-hot labels

0.2	0.3	0.2	0.1	0.1	0.1

softmax output

Loss function

Remember we wanted to learn good weights for our models.

The loss function provides an objective to evaluate how good the current weights are.

Softmax cross-entropy for one sample:

$$L_i = -\sum_{k=1}^{C} y_k \log(\hat{y}_k)$$

Over all samples:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i$$

Learning

Now that we have a fully differentiable model and a differentiable loss function, we can optimise the weights using gradient descent.

Initialise $\mathbf{w}_0$

Iterate $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla L(\mathbf{w}_t)$

Why:

- Can compute gradients quickly and efficiently using backpropagation and auto-differentiation.
- Highly parallelisable: we can take advantage of GPU and TPU processing.
- Opportunities for regularisation by adding noise.

Stochastic gradient descent

The loss function and gradients are averaged over training samples.

We normally randomly sample m points from the data (of size N) and compute averages over those:

- $m = N$: batch gradient descent
- $m < N$: mini-batch gradient descent
- $m = 1$: stochastic gradient descent

Convolutions

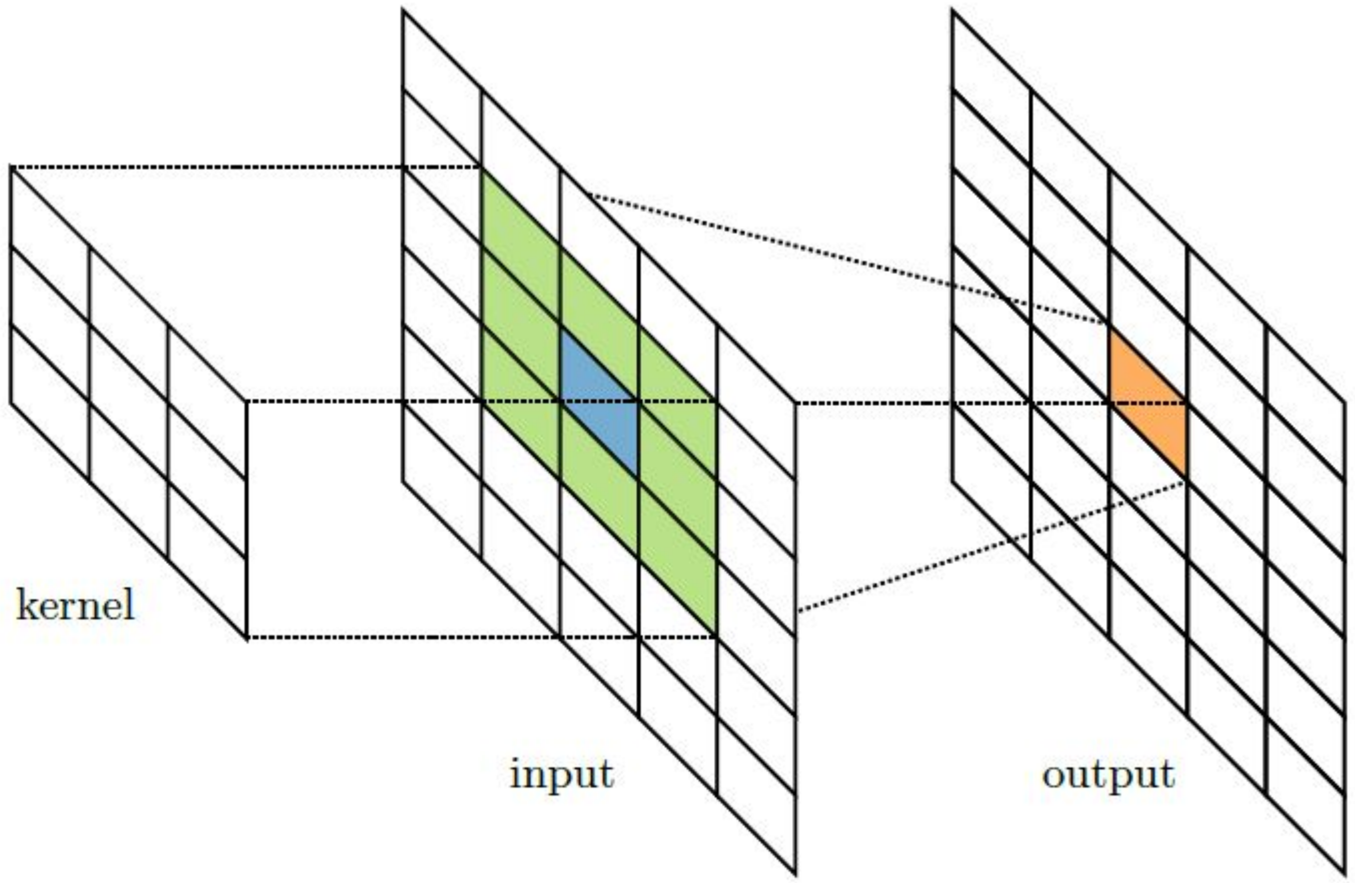


Illustration of two dimensional convolution

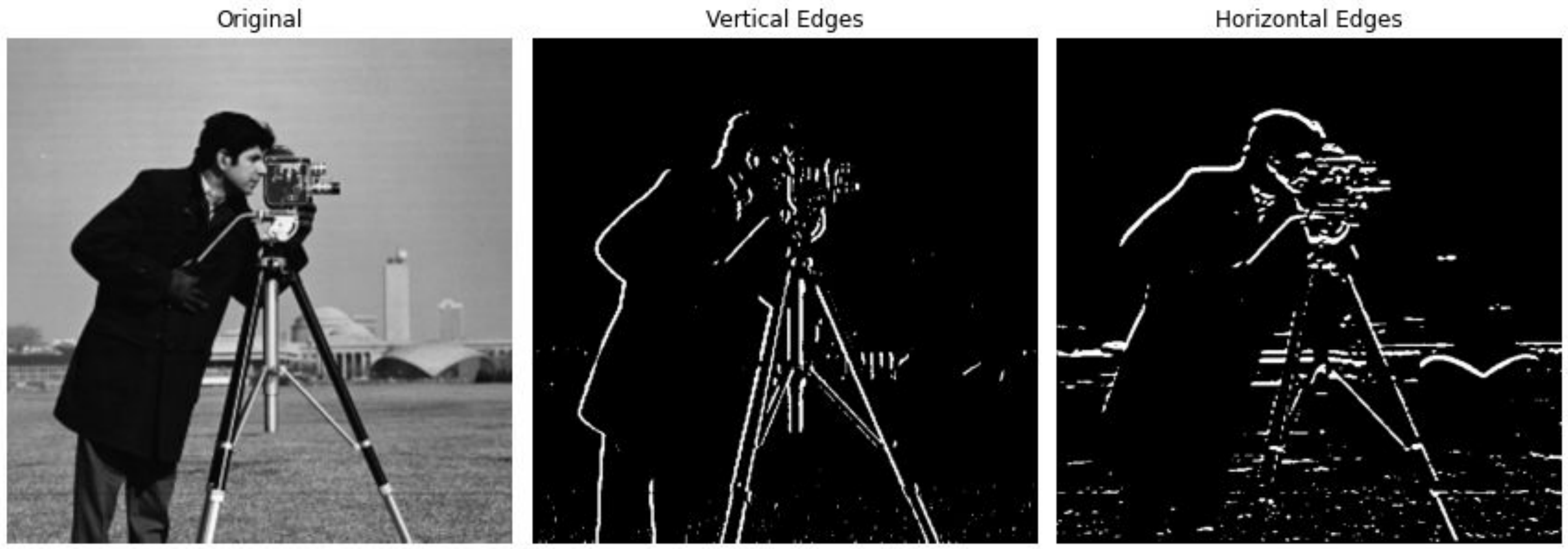
What can a kernel / filter do?

-1	0	1
-2	0	2
-1	0	1

Sobel x-filter

1	2	1
0	0	0
-1	-2	-1

Sobel y-filter

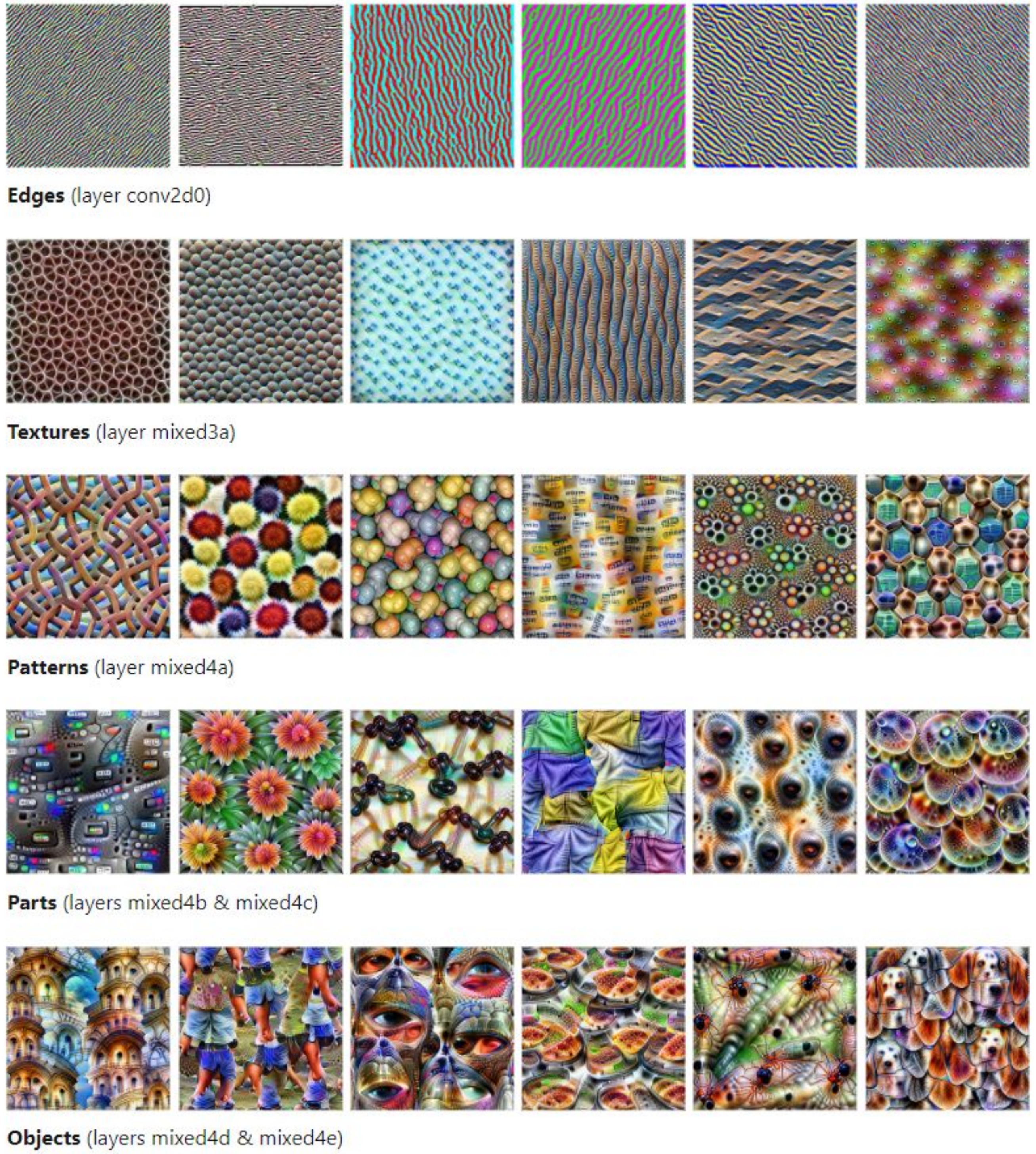


Applying sobel filters to an image to highlight edges.

What features does a NN learn?

Features in each subsequent layer of a NN

(https://distill.pub/2017/feature-visualization/)



Advantages of CNNs: Sparse connectivity

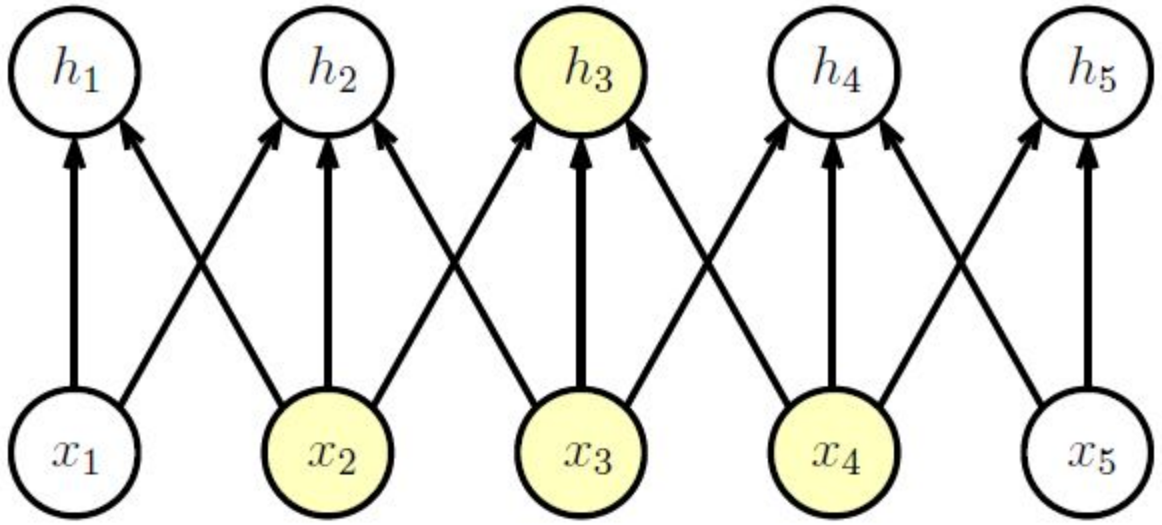


Illustration of sparse connectivity. x_2, x_3, and x_4 form the local receptive field of h_3.

Sparse connectivity leads to fewer computations and lower memory demands.

Advantages of CNNs

Parameter sharing: set of weights in the kernel is applied to all receptive fields of the output, instead of having a distinct weight for each input.

Equivariant to translation: features are detected regardless of their location in the image.

Pooling: adds regularisation and slight translation invariance.

Practical approach using pytorch

- Train a neural network from scratch on an image dataset
  - Create custom datasets and data loaders
  - Create a cnn
  - Train
    - Overfit on training data to check for bugs
  - Evaluate
    - Accuracy
    - Inspect output
  - Repeat but by fine-tuning a pre-trained model

Further Reading & Additional Resources

- To learn more about deep learning:
  - The deep learning book: https://www.deeplearningbook.org/
  - Chris Olah's Blog: https://colah.github.io/
  - Recipe for training neural nets: http://karpathy.github.io/2019/04/25/recipe/
- Additional frameworks:
  - Keras: https://keras.io/
  - Pytorch lightning: https://www.pytorchlightning.ai/
  - Hydra: https://hydra.cc/
    - I don't recommend this for the bootcamp project.
    - Useful to enable best practices for longer-term projects in pytorch
- A bit of both:
  - Fastai: https://www.fast.ai/
    - A good free online course and library that acts as a custom wrapper around pytorch