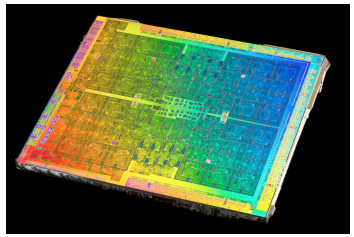
Machine Learning

3 and 5 December 2019

Gerton Lunter, WIMM

A brief history of Machine Learning

- 1947: transistor
- 1954: FORTRAN
- 1958: first IC
- 1969: UNIX
- 1976: Apple I
- 1981: IBM PC; MS-DOS
- 2007: NVIDIA's CUDA



- 1951: Marvin Minsky builds first neural network
- 1953: "Machine Learning" coined (Arthur Samuel, IBM)
  Program for playing Checkers; alpha-beta pruning
- 1957: Perceptron (Frank Rosenblatt)
- 1969: Marvin Minsky and Seymour Papert, book "Perceptron"
  describes limitations of NNs. Beginning of "AI Winter"
- 1970: Seppo Linnainmaa introduces backpropagation
- 1982: John Hopfield introduces "Hopfield networks"
- 1986: Backpropagation used by Geoff Hinton and others
- 1995: Tin Kam Ho describes random forests
- 1995: Cortes & Vapnik describe Support Vector Machines
- 1997: Sepp Hochreiter & Jürgen Schmidhuber invent LSTM
- 1998: Yann LeCun introduces the MNIST dataset
- 2009: ImageNet dataset introduced, "catalyst for AI boom"
- 2016: Computer beats human in GO

Deep learning – nuts and bolts

GMS DTC, November 2019

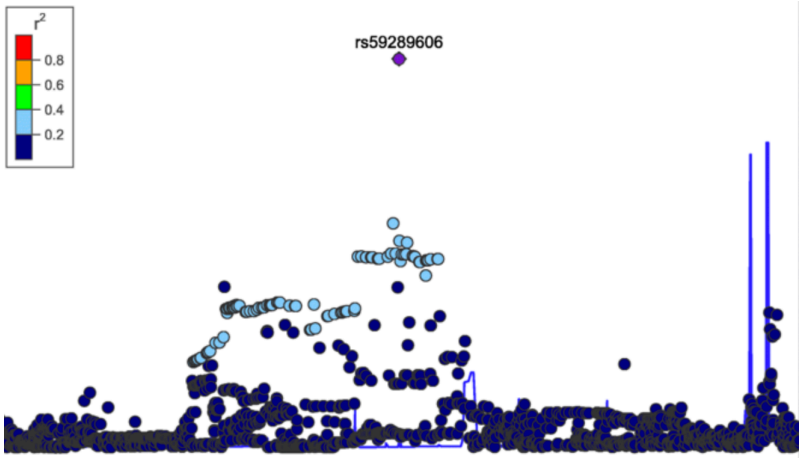
Gerton Lunter
WIMM

Neural networks

Neural networks vs. statistics

Statistics

- Simple relationships
  - Linear regression: $y = A x + \varepsilon$
  - Logistic regression: $\log(p/1-p) = A x + \varepsilon$
- Careful modeling of the error ε (e.g. covariance structure)
- Careful modeling of uncertainty (estimates as well as prediction)
- Works well with little to modest amounts of data
  - For large amounts of data, model misfit leads to spurious precision
- Equal focus on testing ($A = 0$ vs $A \neq 0$) and prediction ($\hat{y} = A x$)



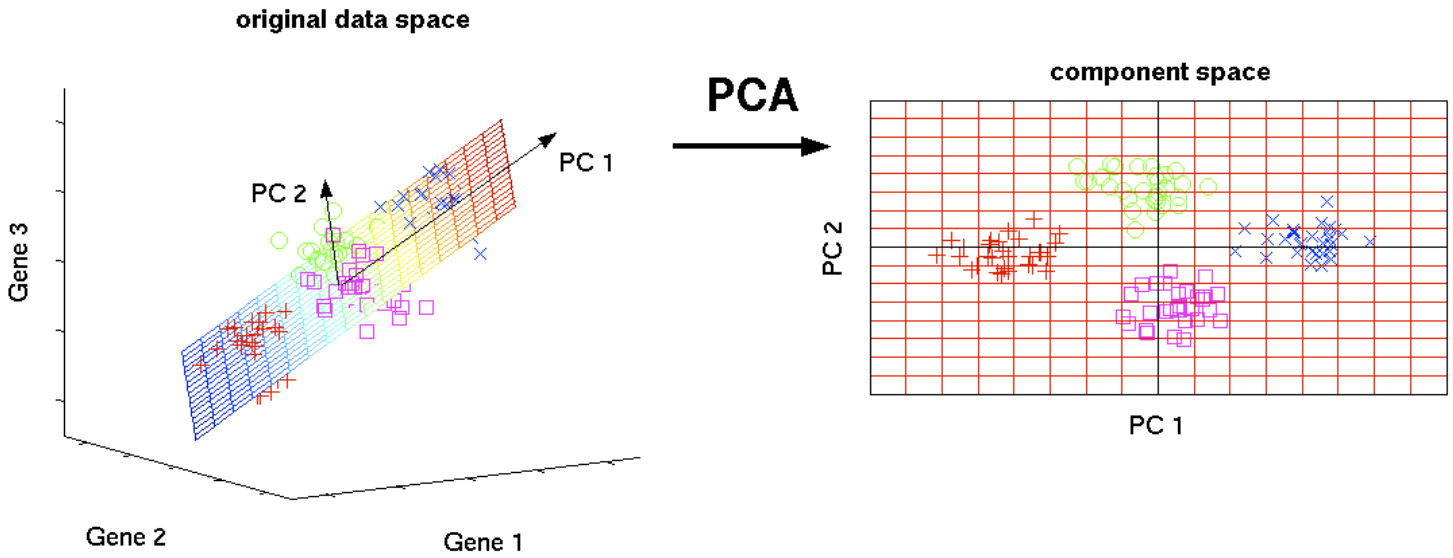
Neural networks

- Complex (nonlinear) relationships
- Many applications
  - Regression and classification,
    data generation (e.g. image captioning, super-resolution),
    control (game play, robot, self-driving car)
- Superficial modeling of ε and uncertainty
- Best results with very large (size & dimension) data sets
- Focus on prediction, not testing

The girl with blue hair stands under the umbrella.
NP: a woman, an umbrella, a man, a person, a girl, umbrellas, that, a little girl, a cell phone.
VP: holding, wearing, is holding, holds, carrying.
PP: with, on, of, in, under.
A woman is holding an umbrella.

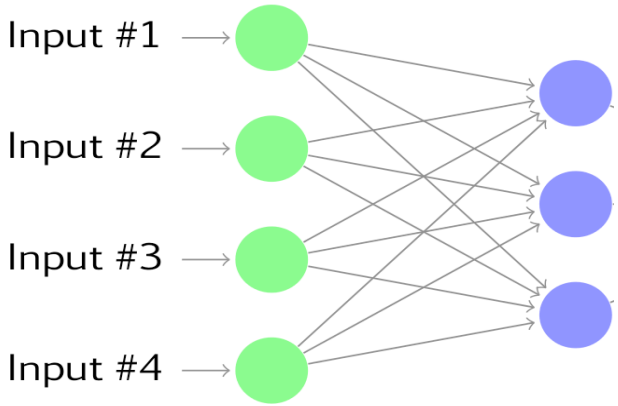
Remi Lebret et. al. ICML 2015

Principal component analysis as a linear neural network



Principal component analysis as a linear neural network

- For a dataset $\mathbf{A} \subset \mathbf{R}^n$, PCA finds a projection from $\mathbf{R}^n$ to $\mathbf{R}^k$ ($k \ll n$) that retains as much of the variance of the data **A** as possible.

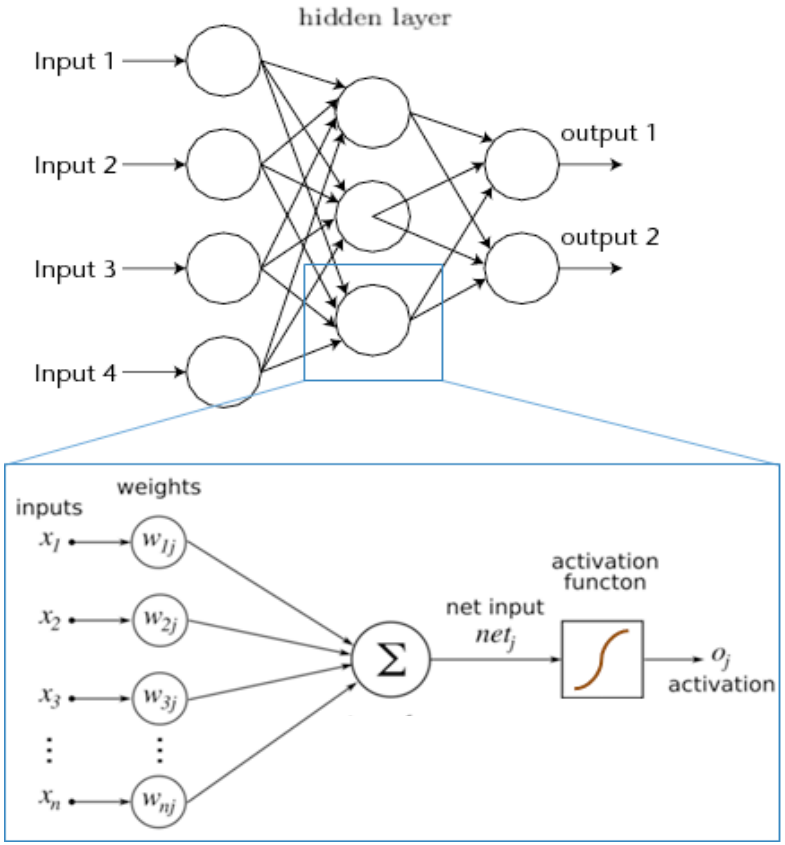


(Mathematically: take first k eigenvectors of **A** as a matrix)

Neural network

A neural network is similar to PCA, with key differences:

- Addition of a *nonlinear* "activation function"
- *Multiple* layers (and *many* parameters)
- No simple algorithm to find "best" coefficients



Classification problem: traffic signs

Multi-Column Deep Neural Network for Traffic Sign Classification

Dan Cireşan, Ueli Meier, Jonathan Masci and Jürgen Schmidhuber

IDSIA - USI - SUPSI — Galleria 2, Manno - Lugano 6928, Switzerland

Abstract

We describe the approach that won the final phase of the German traffic sign recognition benchmark. Our method is the only one that achieved a better-than-human recognition rate of 99.46%. We use a fast, fully parameterizable GPU implementation of a Deep Neural Network (DNN) that does not re-

Speech recognition

ACHIEVING HUMAN PARITY IN CONVERSATIONAL SPEECH RECOGNITION

W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu and G. Zweig

Microsoft Research
Technical Report MSR-TR-2016-71

ABSTRACT

Conversational speech recognition has served as a flagship speech recognition task since the release of the DARPA Switchboard corpus in the 1990s. In this paper, we measure the human error rate on the widely used NIST 2000 test set, and find that our latest automated system has reached human parity. The error rate of professional transcriptionists is 5.9% for the Switchboard portion of the data, in which newly ac-

Game playing - Go

nature International weekly journal of science

Home | News & Comment | Research | Careers & Jobs | Current Issue | Archive | Audio & Video | For

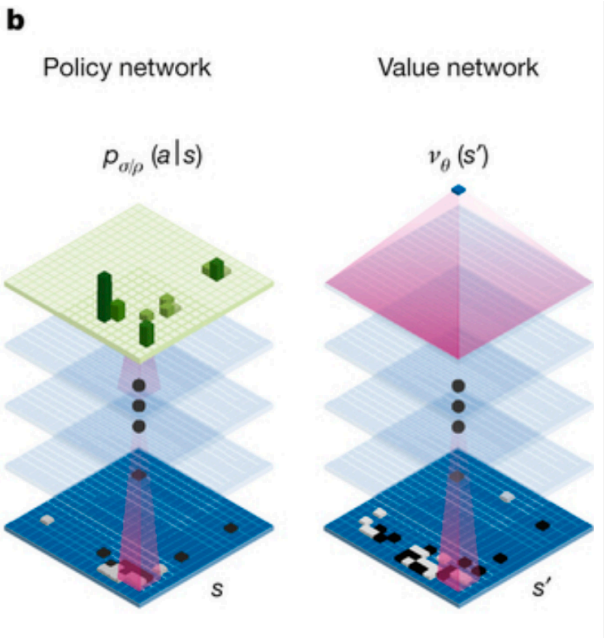
Archive > Volume 529 > Issue 7587 > Articles > Article

NATURE | ARTICLE

日本語要約

Mastering the game of Go with deep neural networks and tree search

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis



The building blocks

Setup

Multiple linear regression

$$\hat{y}_i = \sum_k \theta_k x_{ik}$$

Setup

Multiple linear regression

$$\hat{y}_i = \sum_k \theta_k x_{ik}$$

$$y_i = \hat{y}_i + \epsilon_i; \quad \epsilon_i \sim \mathcal{N}(0,1)$$

$$-\log p(y|x) = -\log \prod_i \frac{e^{-(y_i-\hat{y}_i)^2/2}}{\sqrt{2\pi}}$$

$$= \sum_i \frac{1}{2}||y_i - \hat{y}_i||^2 + C$$

Max likelihood: $\hat{\theta} = (X^T X)^{-1} X^T y$

Neural network

$\hat{y}_i = F_\theta(x_i)$ neural network

$p(y_i|x_i,\theta) = p(y_i|\hat{y}_i)$ error model

$$-\log p(y|x) = -\sum_i \log p(y_i|\hat{y}_i)$$

$$= \sum_i \frac{1}{2}||y_i - \hat{y}_i||^2 \quad \text{(e.g.)}$$

Max l'hd: $\hat{\theta} = \text{argmin}_\theta \sum_i ||y_i - F_\theta(x_i)||^2$

Key questions

$\hat{y}_i = F_\theta(x_i)$ neural network

$p(y_i|x_i, \theta) = p(y_i|\hat{y}_i)$ error model

$L(y_i|\hat{y}_i) = -\log p(y_i|\hat{y}_i)$ loss function

- How to choose p
- How to choose F_θ
- How to fit the model efficiently
- How to compute gradient of F_θ
- How to deal with symmetries in the problem
- How to account for uncertainty in the parameters
- How to deal with overtraining
- How to interpret the model

The error model

Error model

- Same considerations as in "normal" statistical modeling

$$p(y_i|x_i,\theta) = p(y_i|\hat{y}_i) \quad \text{error model}$$

- Try to find "generative process" for errors
  - Probabilities multiply; take negative logarithm to get *additive* loss function
  - Scale and constant terms don't matter. Must be differentiable.

  - **Predicting** temperature. Error = Gaussian, loss function = **sum of squared differences** = Euclidian distance
  - **Predicting** count(s). Likelihood e.g. (overdispersed) Poisson, (beta-)binomial, negative binomial, …
  - **Classification**. Likelihood is multinomial, loss function is **cross entropy**

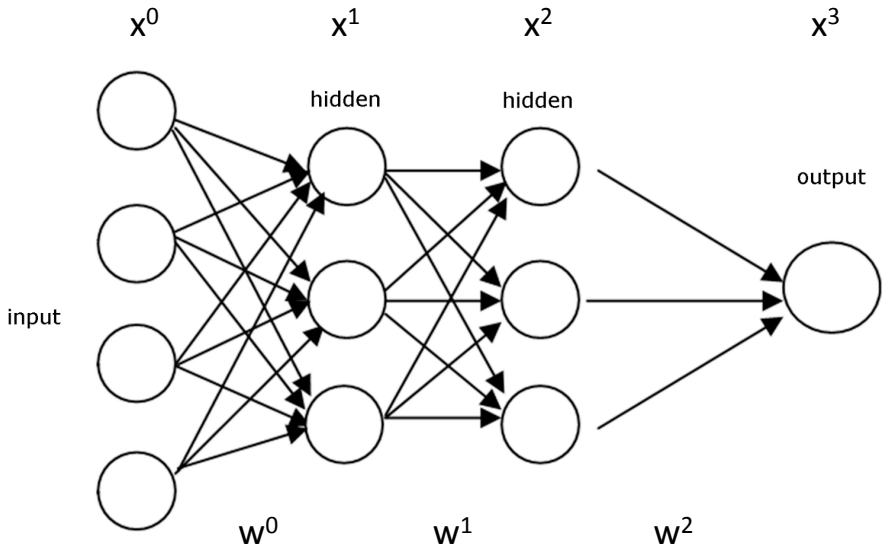
- Should model observation process (measurement error, count model) but also prediction error
  - Similar to linear modeling when the actual relationship has a nonlinear component

The neural network architecture - F_θ

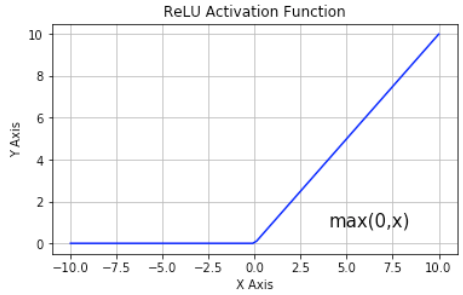
Neural network architecture

- **Fully connected network**

$$x_j^{i+1} = f\left(\sum_k w_{jk}^i x_k^i\right)$$



- f: *activation function,* introducing nonlinearity
  - Common choice: ReLU (Rectified Linear Unit)



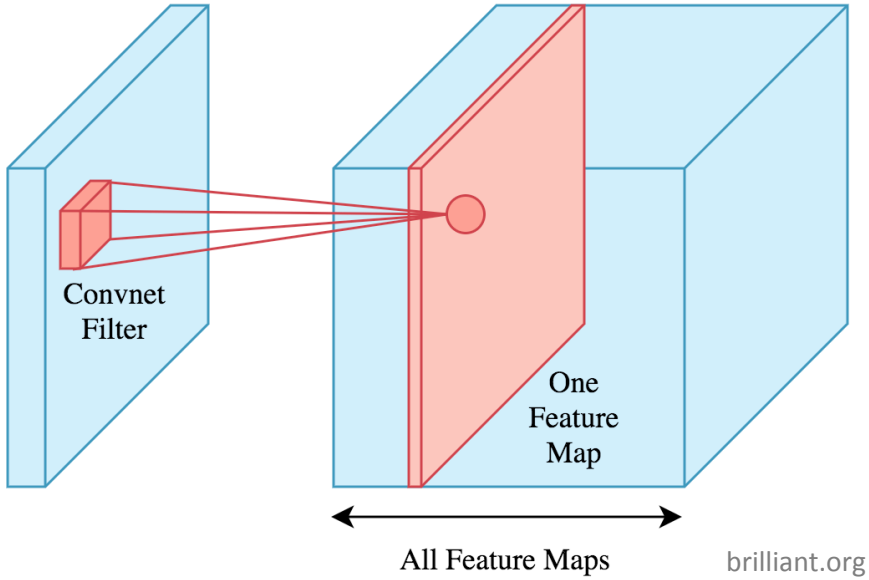
Universal approximation theorem:
Any function can be approximated with a (deep, wide) NN

Balázs Csanád Csáji (2001) Approximation with Artificial Neural Networks (MSc thesis)

Convolutional Neural Network

- Often, model has translation symmetry
  - "Meaning" of a signal does not change if you shift it
  - Sound (1D), image (2D), DNA (1D), weather patterns (2D), physics (3D), …

- **Convolutional Neural Network** (Kunihiko Fukushima, 1980): use same weights for all "positions"
  - Find edges, binding motifs, …, *wherever* they occur
  - Fewer weights → less memory, parallelizable
  - More training examples per weight → better learning
  - Required symmetry baked in → better generalization



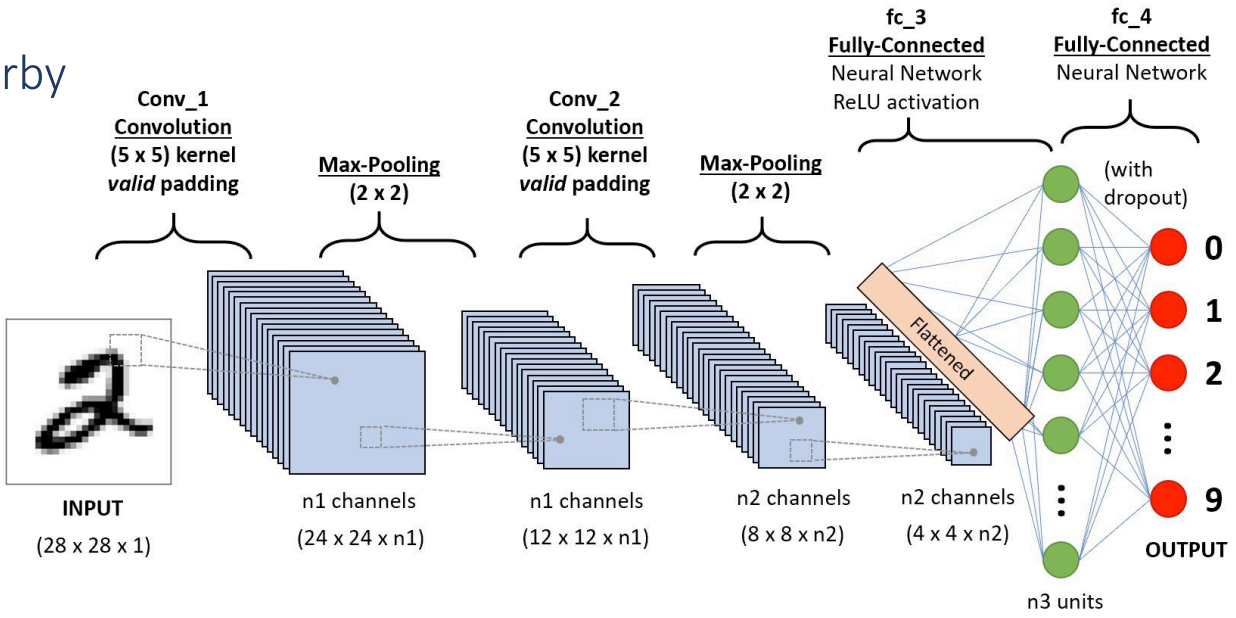
brilliant.org

Pooling

- Output dimension of CNN is same as input [ignoring padding at edges]
  - E.g. 16 x 16 pixels -> 16 x 16 x *N* feature maps

- Often you want position-independent output; reduced resolution at intermediate layers

- **Pooling** (J Weng, 1993): combine nearby "positions" into single output
  - Max pooling: use max to combine
  - Average pooling also used



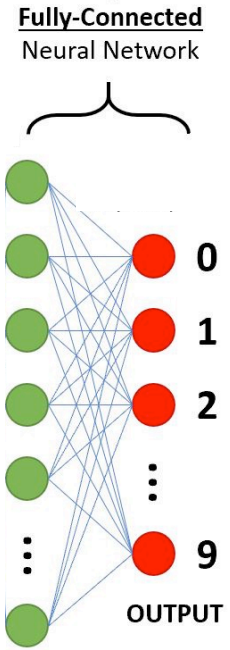
towardsdatascience.com

valid padding: only use contributions where inputs stay inside bounds.

zero padding: assume that out-of-bounds inputs are zero

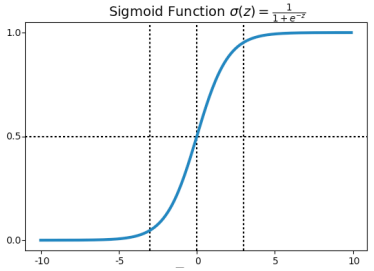
Output layer

- Output layer often *fully connected*, with a special activation function
  - Classification: softmax
  - Non-exclusive features: sigmoid



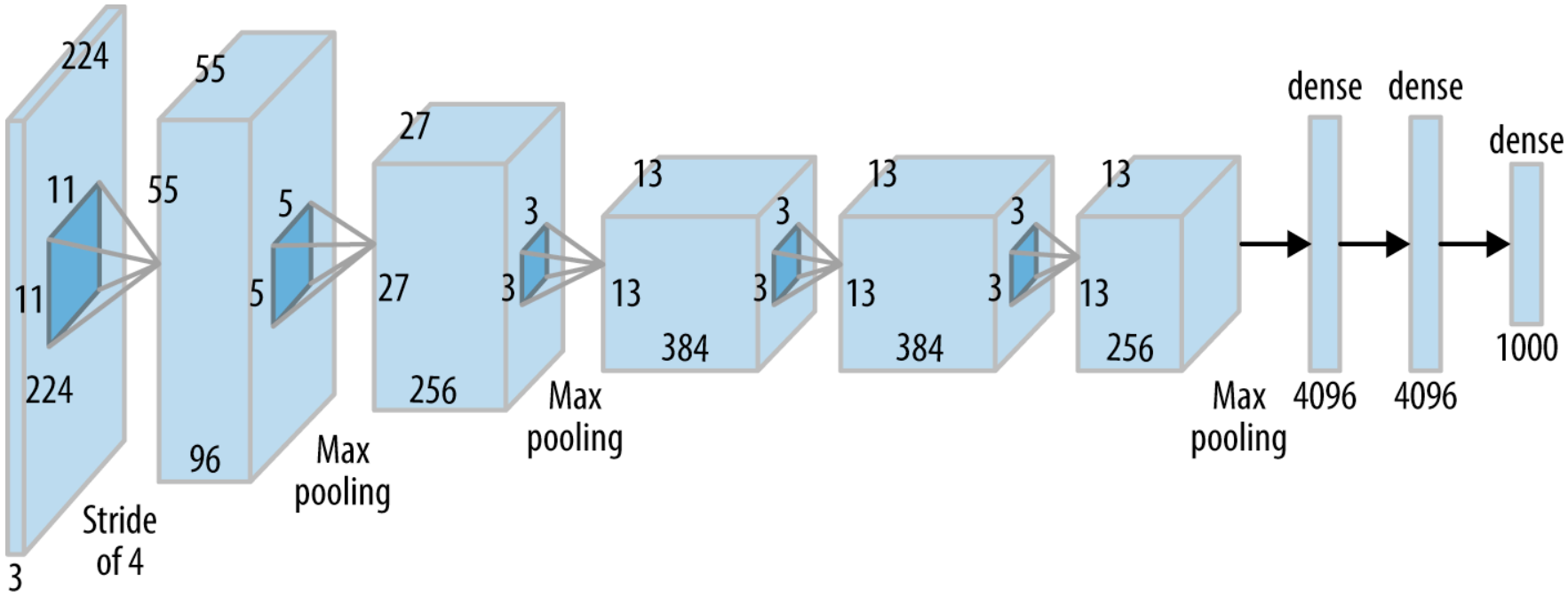
$$y_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \qquad \text{softmax;} \quad \sum_i y_i = 1$$

$$y_i = \frac{\exp(x_i)}{1 + \exp(x_i)} \qquad \text{sigmoid;} \quad 0 < y_i < 1$$



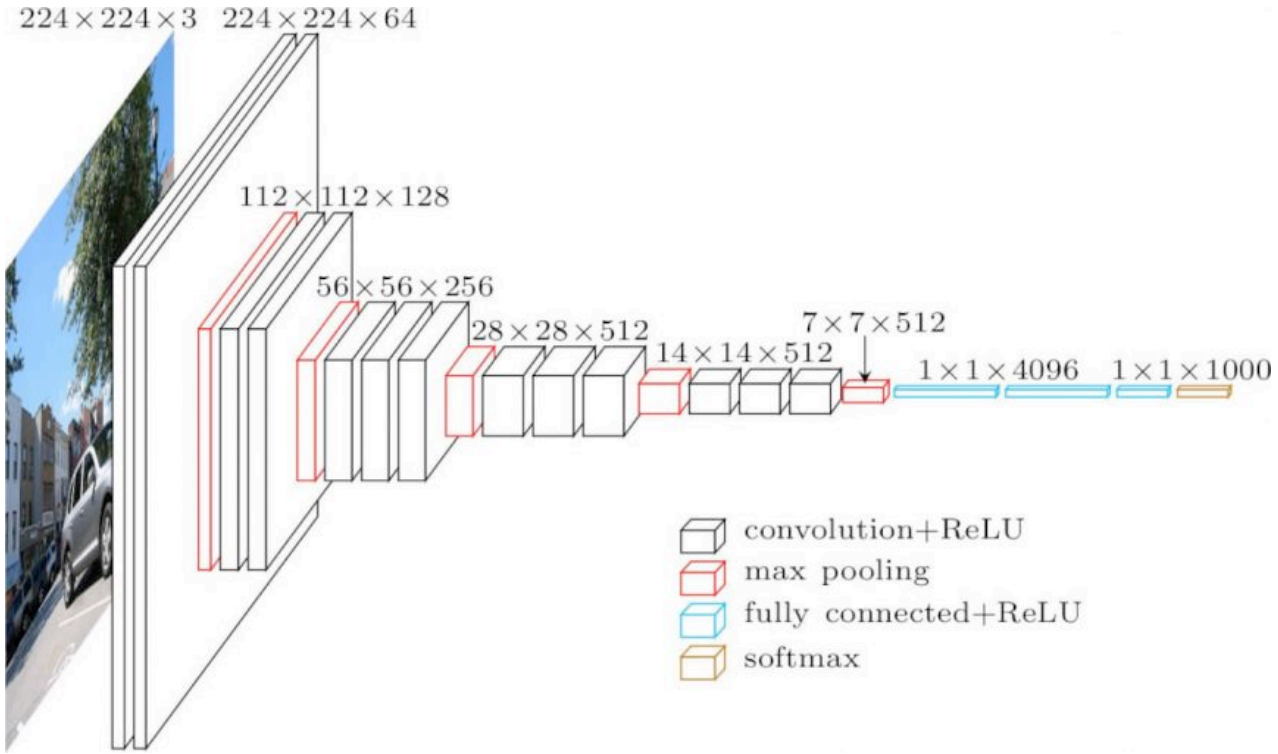
Example: AlexNet (2012)

- First time a CNN was used for image classification
- Top-5 ImageNet test error **15.4%**. Next best entry 26.2%



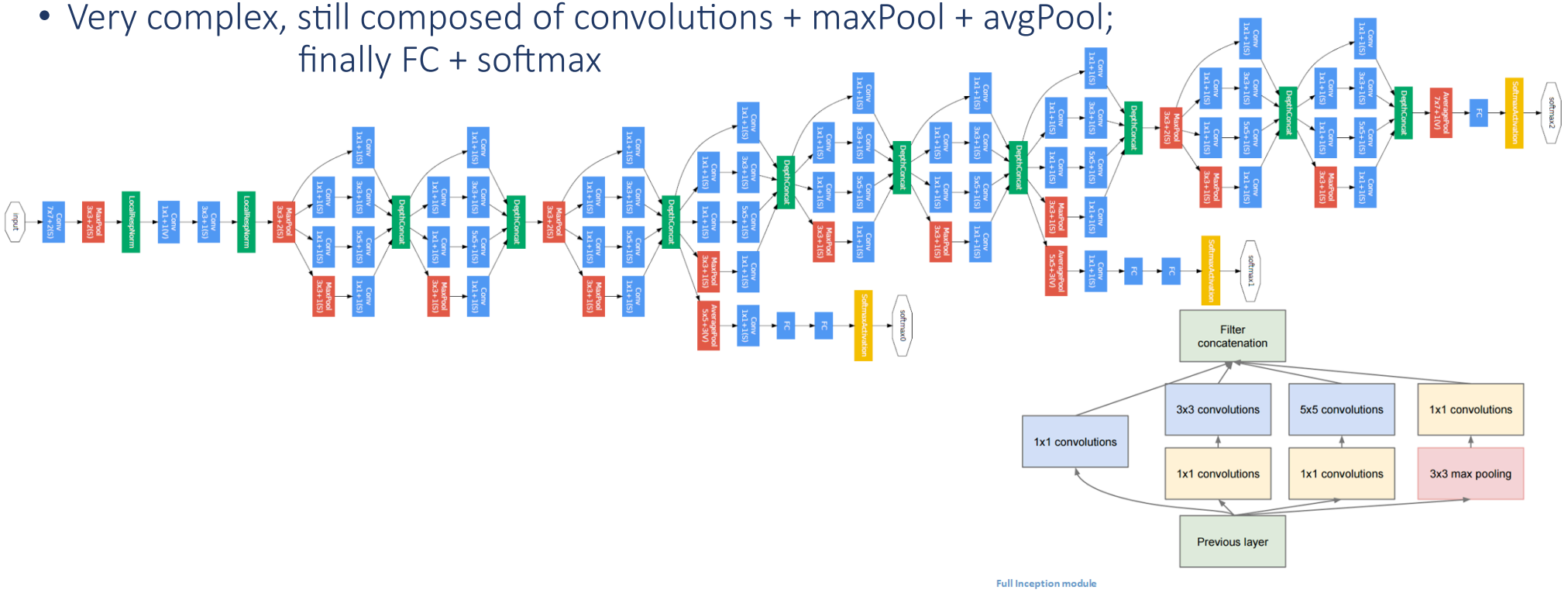
Example: VGG16 (2014)

- ImageNet 2014: **7.3%** error
- Smaller filters, more depth – fewer parameters, better performance



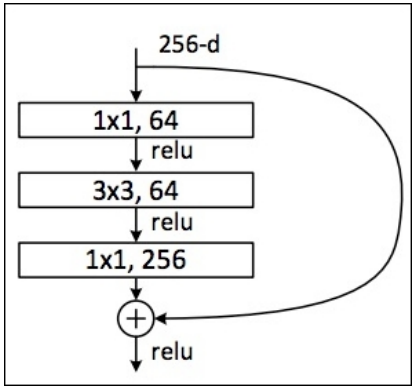
Example: GoogLeNet (2015)

- ImageNet 2015: **6.7%** error rate
- “Inception” module
- Very complex, still composed of convolutions + maxPool + avgPool; finally FC + softmax

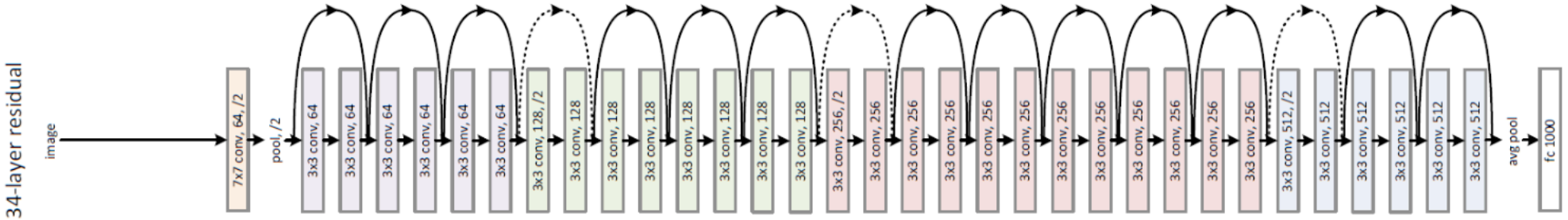


Residual networks (2015)

- More layers are beneficial (AlexNet -> VGG -> GoogLeNet)
- But: learning becomes more difficult. Information may get lost by a layer.
- Solution: **Residual network**.
  Add bypass channel; layer adds a small perturbation



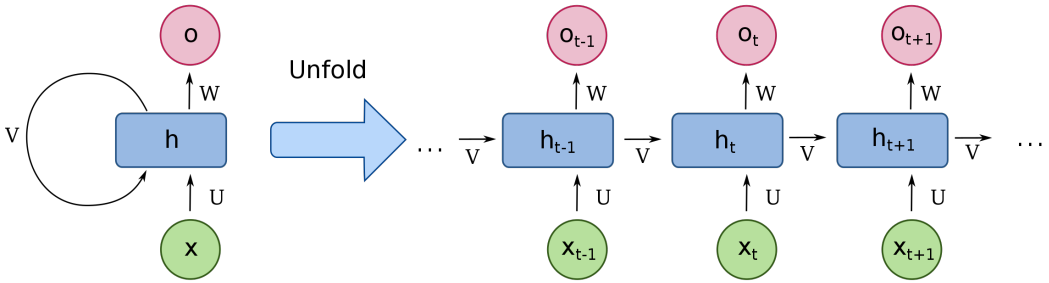
- ResNet (2015): **3.6% error** (human: 5-10%)



More architecture

- CNNs combine information locally. What if relevant info is "far" away, or at variable distance?
  - Language; DNA sequences

- **Recurrent network (RNN)**: memorize and transmit information "sideways"
  - Memory units: LSTM, GRU
  - Value in layer depends on previous layer, and neighbor in own layer
  - Tends to slow down computation, and makes learning unstable
  - Old idea (Schmidhuber 1997), used in many state-of-art models, but is getting out of favor in modern models

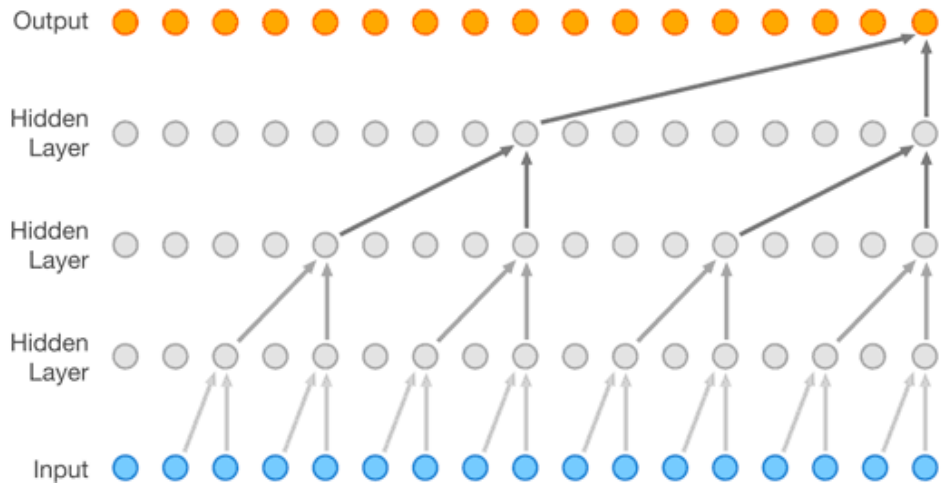


More architecture

- CNNs combine information locally. What if relevant info is “far” away, or at variable distance?
  - Language; DNA sequences

- **Dilated convolutional networks**
  - Output influenced by many inputs
  - Number of weights manageable
  - Fast calculation on standard GPUs
  - Influence of each input on output is *irregular* – neighbouring inputs may interact at different levels depending on their location

- Famous application: WaveNet
  - arXiv:1609.03499 (2016)



More architecture

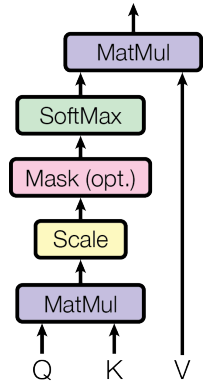
- CNNs combine information locally. What if relevant info is "far" away, or at variable distance?
  - Language; DNA sequences

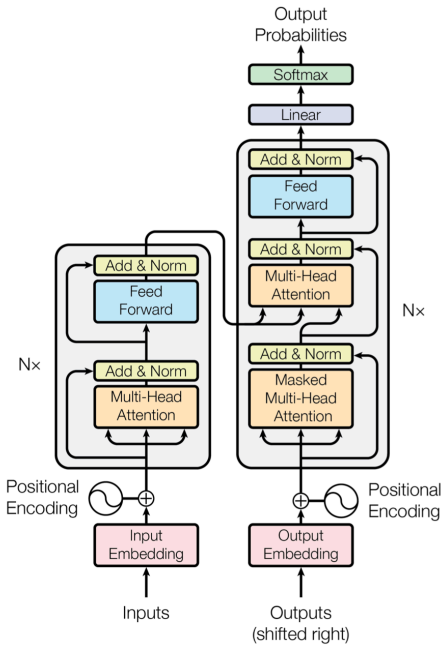
- **Attention mechanism**
  - Encode "what" you're interested in
  - Inner product: look for relevant location
  - Copy (other) info across from that location

  Position-independent. Works well for translation

  arXiv:1706.03762 (2017)

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

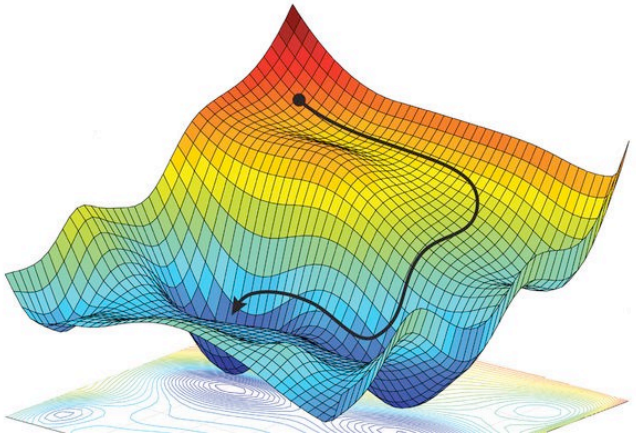


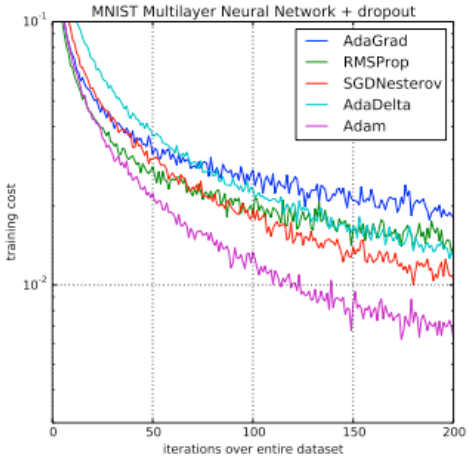


Fitting the model (learning)

Fitting the model

- Find θ minimizing $S_\theta := \sum_i S[\, y_i,\ F_\theta(x_i)\,]$
- Gradient descent: $\theta_{t+1} = \theta_t - \varepsilon \nabla_\theta [S_\theta]$
  - In practice, S is evaluated on small data batches at a time ("minibatches"), each followed by parameter update: *stochastic* gradient descent (SGD)
  - Sensitive to choice of ε ("learning rate")
- Several methods to change learning rate adaptively
  - **Adam** is a good all-round choice
  - Kingma and Ba, arXiv:1412.6980 (2015)



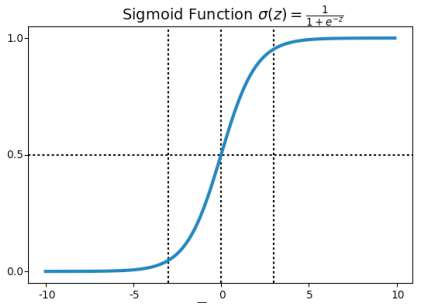


Two problems in learning deep neural networks

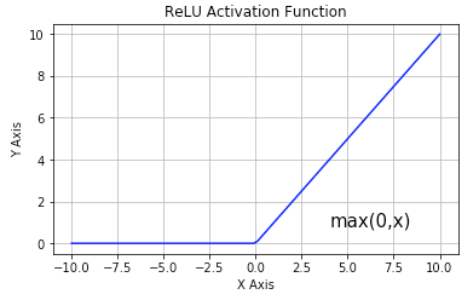
- Vanishing gradient problem
  - Derivatives w.r.t. parameters can become very small as you stack many layers

$$f_1(f_2(f_3(f_4(x))))' = f_1'(x_1) \times f_2'(x_2) \times f_3'(x_3) \times f_4'(x)$$
$$x_1 = f_2(f_3(f_4(x)))$$
$$x_2 = f_3(f_4(x)) \text{ etc.}$$

  - Particular problem with sigmoid (tanh) function: derivatives at most ¼ (or 1), very small away from origin



- Solution
  - **ReLU**: derivative either 0 or 1, on a large range



Two problems in learning deep neural networks

- What constrains the "scale" of intermediate layers?
  - ReLU: scale of intermediate layers completely arbitrary
  - Runaway activations can again lead to vanishing (or exploding) gradient problem

- First solution: Weight initialization
  - Idea: Try to keep *distribution* of output constant across layers (mean 0, variance 1)
  - For tanh activation: **Xavier initialization** (Xavier Glorot & Bengio 2010)
  - For ReLU activation: **He initialization** (Kaiming He et al. 2015; arXiv 1502.01852)

- Second solution: Batch normalization
  - Made training deep neural networks possible
  - Idea: simply ***normalize*** outputs to have mean 0 and variance 1 (across a ***batch*** of inputs)
  - When predicting (rather than learning), use mean and variance estimated from all training data, not just batch
  - Reduces the dependence on weight initialization
  - Ioffe and Szegedy 2015; arXiv 1502.03167

How to compute the gradient $\nabla_\theta[S_\theta]$

Computing the gradient – in practice

- In practice F_θ can be hugely complicated (10-100 layers, >1M parameters and variables)
- Computing $F_\theta(x)$ and $S_\theta := \sum_i S[\, y_i, F_\theta(x_i)\,]$ is one thing, but what about $\nabla_\theta [S_\theta]$?

- Key features of deep learning frameworks (TensorFlow, pyTorch):
  - Fast computation (CPU, GPU, TPU)
  - Built-in components (optimization, visualisation, common layers, …)
  - **Automatic differentiation**
    - Exact, fast
    - A computation is represented by a graph – delayed execution
    - Basic functions "know" their derivative
    - Automatic application of the chain rule
    - Backpropagation (Werbos 1975)
      - dynamic programming algorithm for computing $\nabla_\theta f$ for high-dimensional θ

Dealing with symmetries

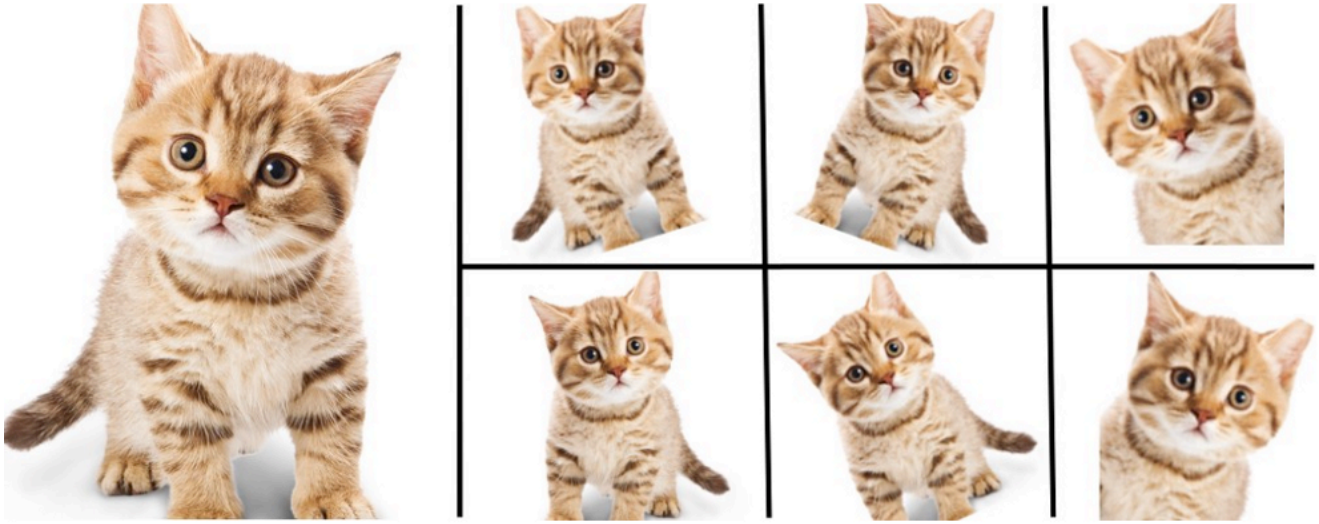
Symmetries

- When the problem has symmetries, model should have them too
  - E.g. image classification: translation, scaling, rotation, color changes, saturation, contrast
  - E.g. speech recognition: time invariance, speed of speech, high/low voice
  - E.g. genomics: reverse complement symmetry
- Why is it helpful if the model has the same symmetries?
  - Model behaves predictably - no change in prediction with “irrelevant” change in input
- But symmetries is just one aspect of the problem. It can be learnt.
  - Yes, but data used learning a symmetry we know, could be better spent learning things we don’t know

Symmetries

- Approach 1: Data augmentation: make “new” data by applying symmetries to data
  - E.g. apply scaling, rotation, translation + cropping to images
  - E.g. add reverse-complemented DNA sequences to dataset

  - Very easy to implement, and improves results, but does not lead to exact symmetric models



Symmetries

- Approach 2: Explicitly make the model "symmetric"
  - Convolutional neural network: explicitly enforce translational symmetry
    - Massive reduction of parameters compared to fully connected
  - More general approach leads to "equivariant networks" (Lunter and Brown, 10.1093/bioinformatics/bty964)
    - Developed to deal with reverse-complement symmetry

$$y = F_n F_{n-1} \cdots F_1 \, \phi_0 \, x \qquad (F_i : R_{i-1} \to R_i)$$

$$\phi_i : \ : R_i \to R_i \qquad (\textit{action} \text{ of element of symmetry group on } R_i)$$

Equivariance: $F_{i+1} \, \phi_i \, x = \phi_{i+1} \, F_{i+1} \, x \qquad (x \in R_i)$

$$\begin{array}{ccccccc} R_0 & \xrightarrow{F_1} & R_1 & \xrightarrow{F_2} & \cdots & \xrightarrow{F_n} & R_n \\ \downarrow{\scriptstyle \phi_0} & & \downarrow{\scriptstyle \phi_1} & & & & \downarrow{\scriptstyle \phi_n} \\ R_0 & \xrightarrow{F_1} & R_1 & \xrightarrow{F_2} & \cdots & \xrightarrow{F_n} & R_n \end{array}$$

Accounting for uncertainty in θ

Uncertainty in parameters

- Statistics gives methods to determine posterior belief in hypothesis $\theta \neq 0$, or uncertainty in θ. No such luck for neural networks
- Uncertainty in parameters tend to be ignored: overfitting is a real danger.
- Hyperparameter search makes the problem worse
  - Trying many different architectures and choosing the best. "Winner's curse"
- Techniques for reducing the problem:
  - Regularization
  - Dropout

Uncertainty in parameters - regularization

- Regularization: Add a penalty term for "large" parameter values
  - $\alpha \Sigma_i \theta_i^2$ - L2 regularization
  - $\beta \Sigma_i |\theta_i|$ - L1 regularization
- Relation to putting a prior on parameters
- Effect:
  - L2: smaller parameter estimates
  - L1: sparse parameter estimates (many 0s) -- but not necessarily when using stochastic gradient descent
- With appropriate choice of α, β can lead to better generalization and less overtraining
  - Drawback: need to set more (hyper)parameters.

Uncertainty in parameters - dropout

- Randomly leave out a fraction of neurons during training. Improves model significantly.
  - arXiv:1207.0580 (2012)
- Dropout can be regarded as (approximate) Bayesian inference
  - Regularization by putting an (implicit) prior on parameters
  - arXiv:1506.02557 (2015) [Kingma Salimans & Welling], arXiv:1506.02142 (2015) [Gal & Ghahramani]
- When predicting, multiply weights by dropout probability
  - Better results are obtained when re-doing dropout and averaging results; arXiv:1506.02142
- When using a symmetric model, it pays to make dropout symmetric too
  - Lunter and Brown, 10.1093/bioinformatics/bty964

Avoiding / detecting overtraining

Detecting overtraining

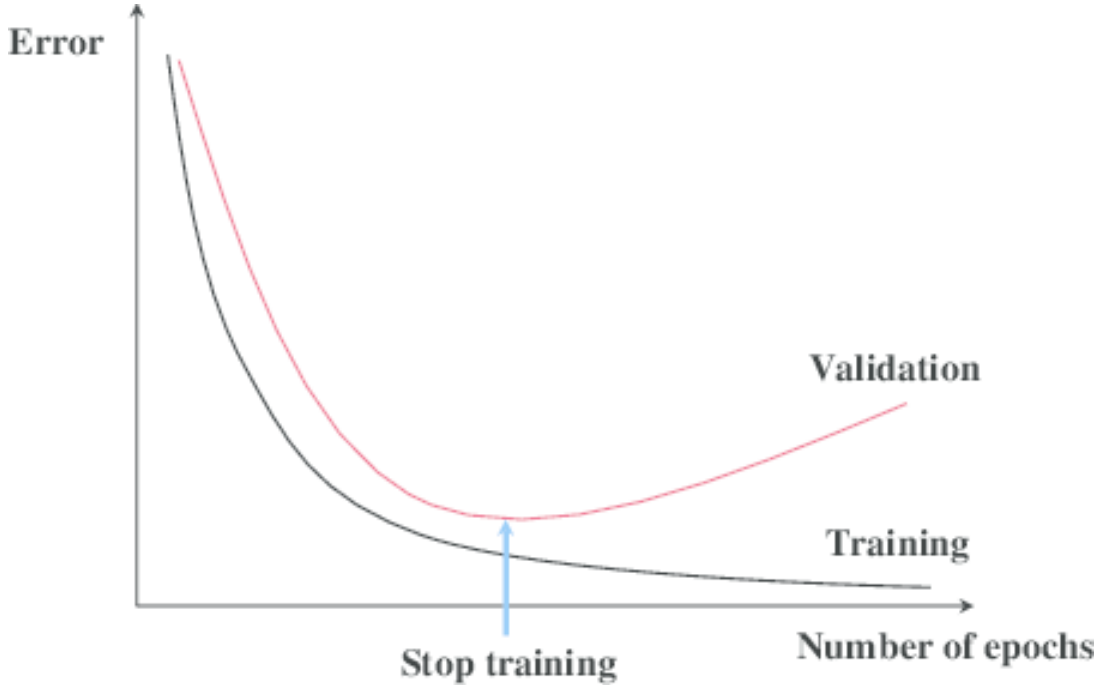
Overtraining occurs when the model “memorizes” examples,
rather than generalizes from relevant features. To be avoided!

- Split data into training (2/3), validation (1/6) and test (1/6)
  - Use training data to train the models
  - Use validation data to assess models, and perform hyperparameter search
  - Use test data to assess final model

- Standard practice in ML. Make sure test data is actually independent!
  - No hidden copies: related individuals (family members), related genes (paralogs), …

Avoiding overtraining

- Early stopping



(epoch = one cycle of training on all data)

Final thoughts

What explains the success of deep learning?

My 2¢…

- Development of GPUs (thank you gamers!), allowing larger models
- Development of automatic differentiation frameworks
  - Speeds up experimentation with complex models
- Discovery that ReLU & friends + maxPool superior to tanh, sigmoid + avgPool
  - Fewer nodes contribute to any specific prediction
- Techniques to train deep networks
  - Batch normalization
  - Adaptive stochastic gradient descent algorithms
  - Dropout
  - Transfer learning
  - Residual networks
- Architecture:
  - Deeper networks improves results
  - Let relevant parts of data interact: dilated networks, attention, LSTM, causal networks for prediction, …
  - Reduce number of parameters (AlexNet->VGG16->GoogLeNet; CNNs)
  - …

Tutorial

- https://github.com/jpwhalley/GMS_Stats_Course/6_Machine_Learning_Applications