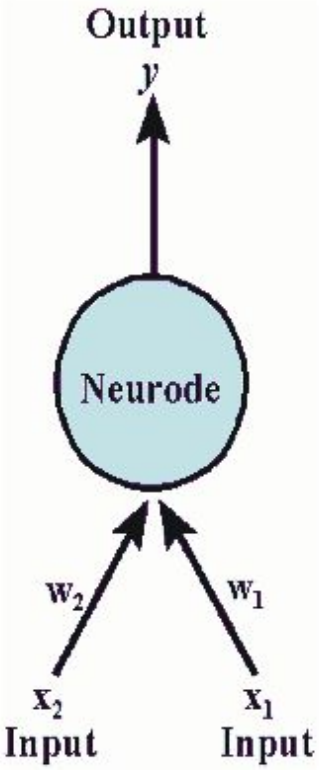
Longitudinal data analysis in R: Deep learning & transformers

Nelson Nazzicari

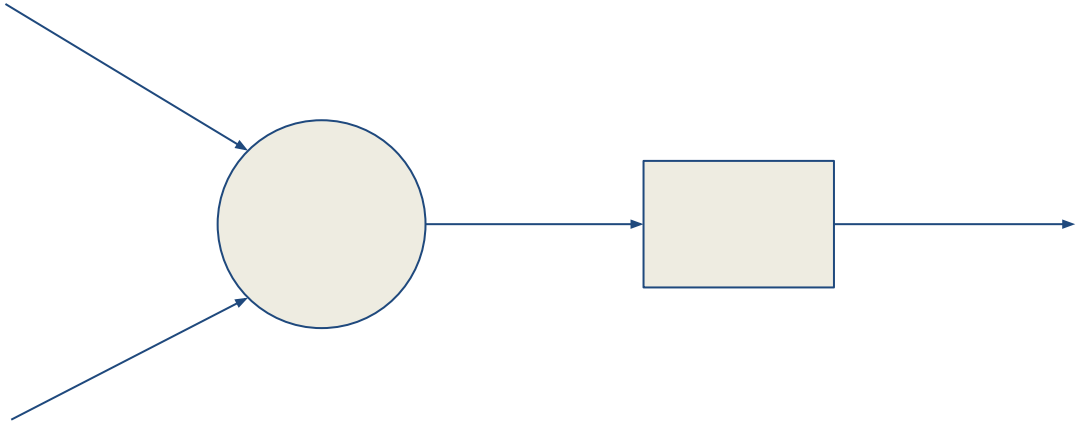
(Bioinformatician & data scientist) **CREA**, Lodi (Italy)

How did it begin?

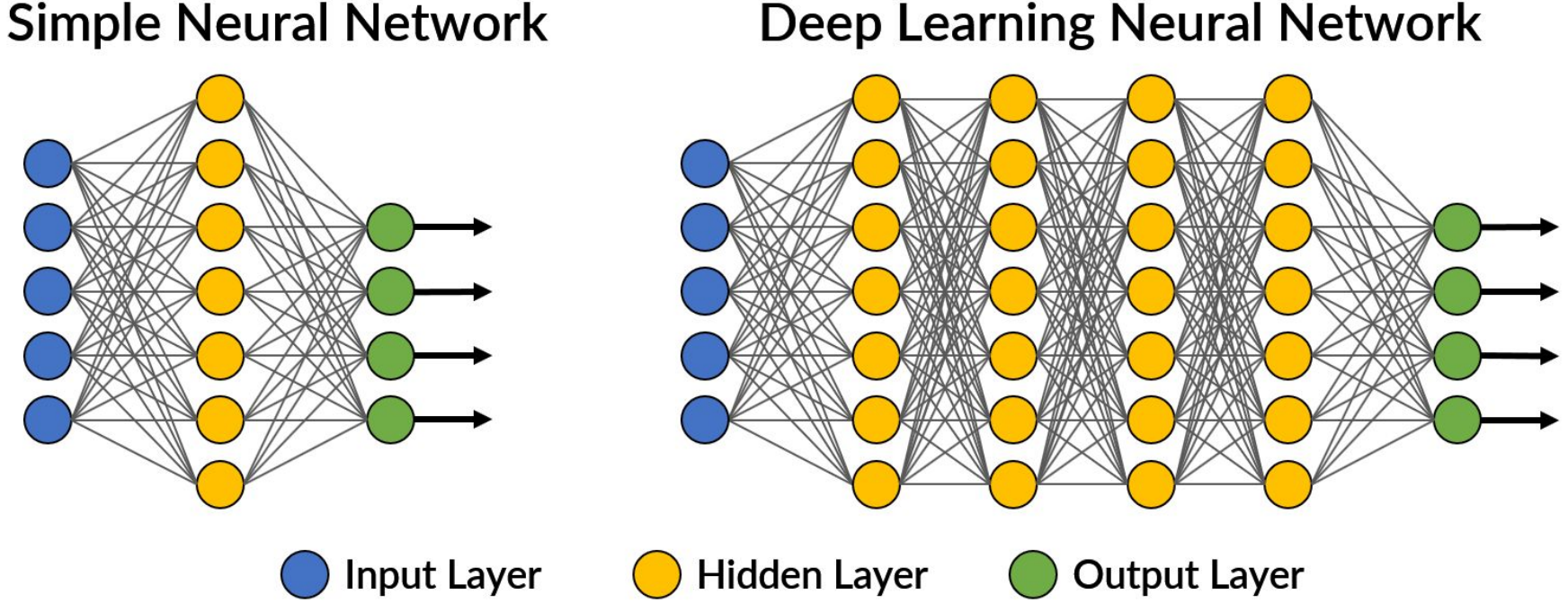
- **1943, McCulloch & Pitts:**
  - first mathematical model of a neural network
  - only binary input and output (0/1),
  - only used the threshold step activation function
  - did not incorporate weighting the different inputs
- **1950s**: pioneers started asking whether computers could be made to “think” → (A.I.)
- **1957, Rosenblatt**: introducing the **perceptron**:
  - weighting inputs
  - additional activation functions (e.g. sigmoid)



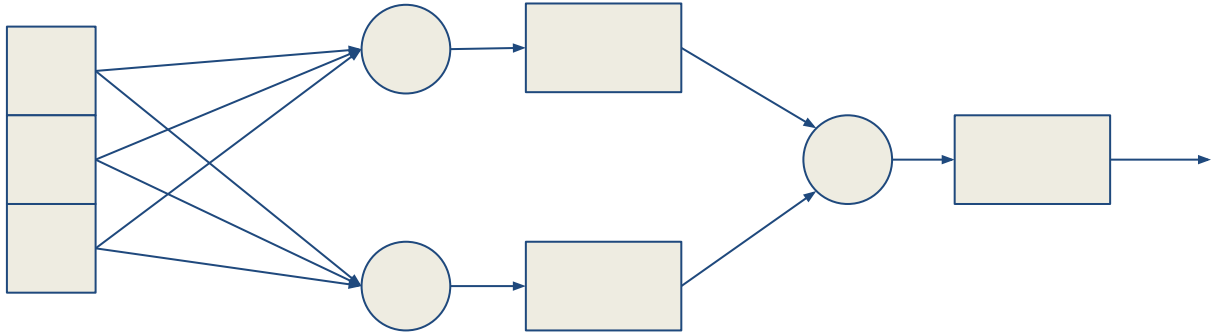
Deep learning an introduction



Deep learning an introduction

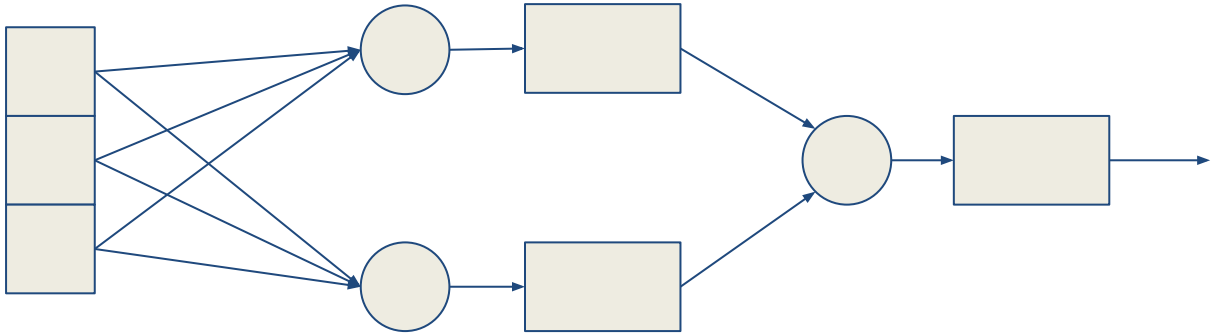


How to learn anything?



Loss function

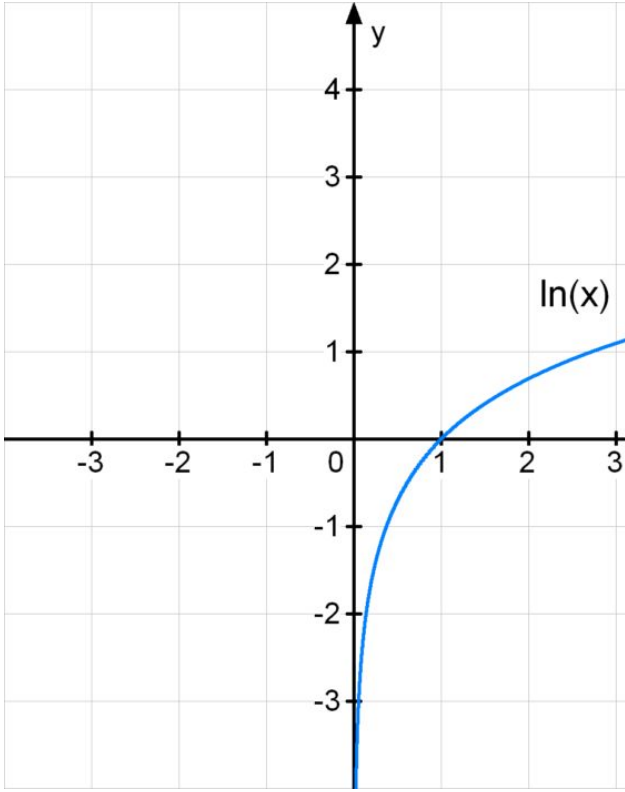
- Measure the distance from true value and prediction
  - Like a residual!
- **Theta**
- Loss(**Theta, X_i, y_i**)
- For regression: RMSE is your friend (differentiable)



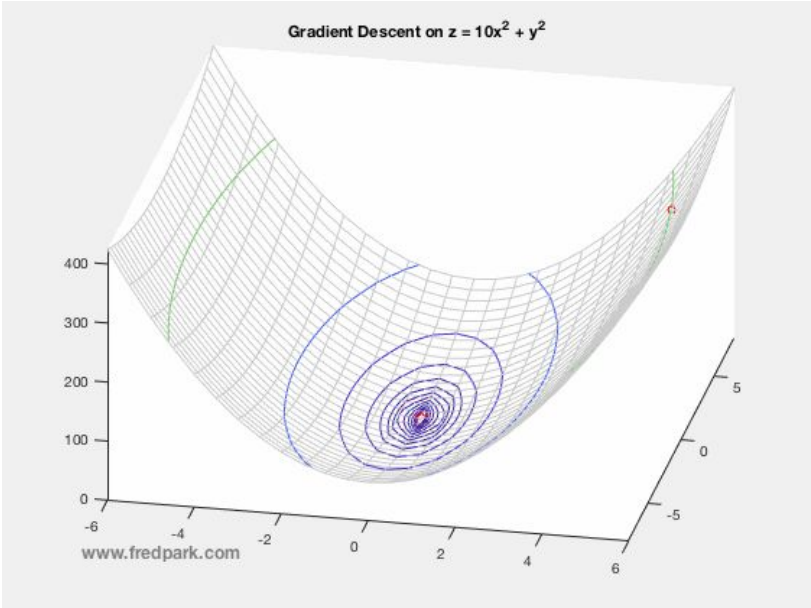
Loss function for binary classification

Binary Cross Entropy

$$BCE(y, \hat{y}) = - (y \ln(\hat{y}) + (1-y) \ln(1-\hat{y}))$$



Gradient descent



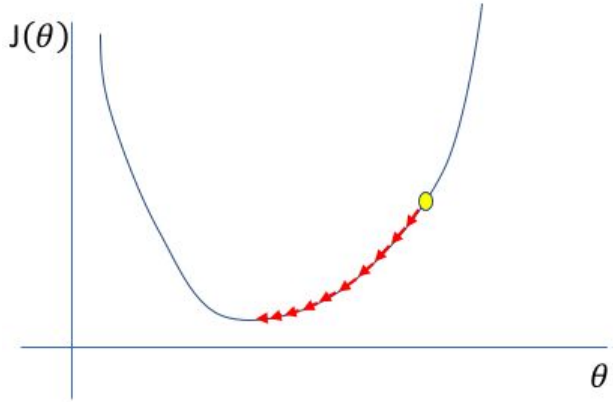
- TARGET: minimize the cost function J(**Theta**)
- ITERATIVELY:
  - Evaluate the gradient of J(**Theta**)
  - **Delta** = - learning_rate * gradient
  - **Theta** = **Theta** + **Delta**

With

- **Theta** : current solution
- **Delta** : step in solution space
- J(**Theta**) : mean(Loss(**Theta, X_1, y_1**) + Loss(**Theta, X_2, y_2**) + …)
- Learning rate…

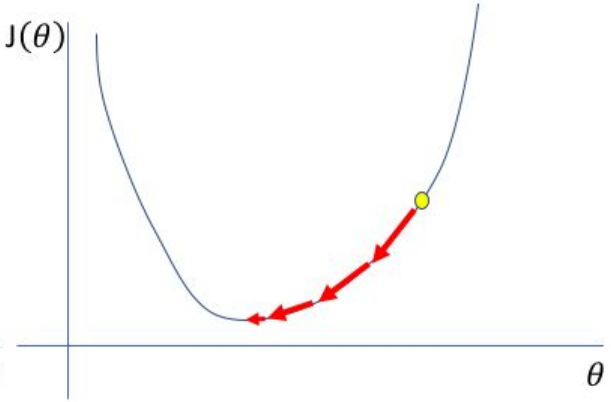
Learning rate

Too low



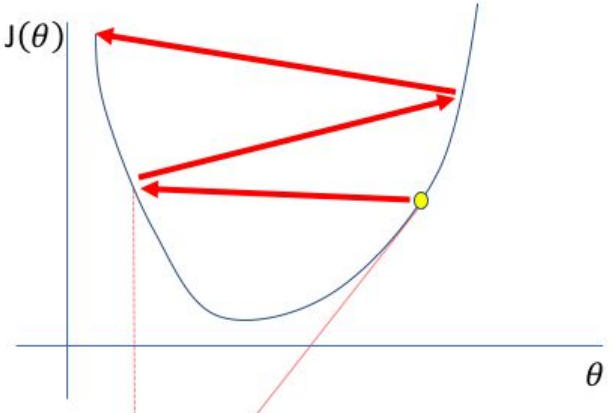
A small learning rate requires many updates before reaching the minimum point

Just right



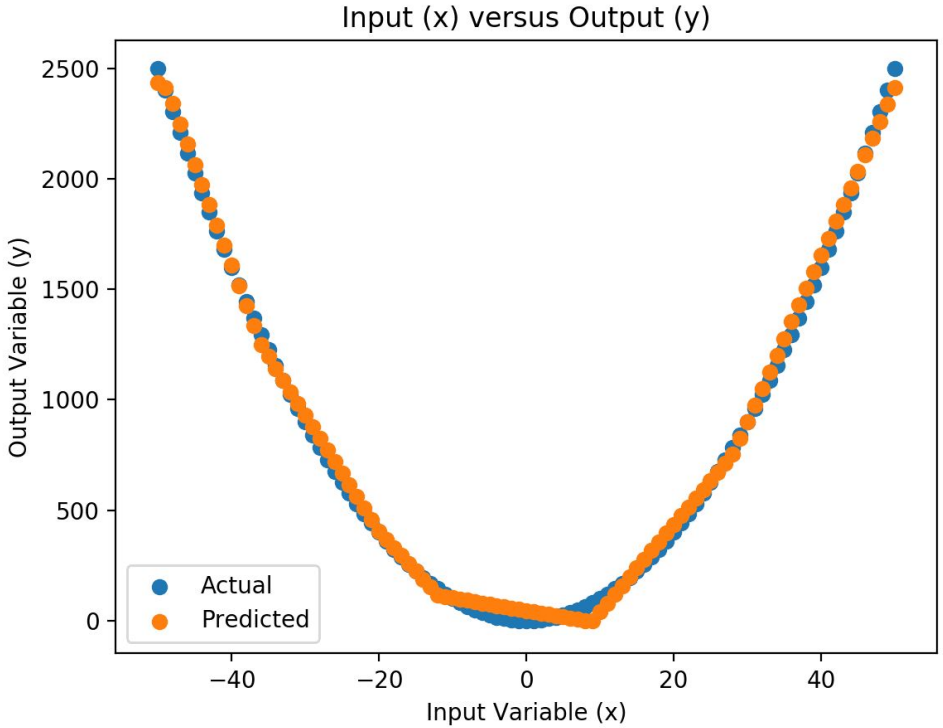
The optimal learning rate swiftly reaches the minimum point

Too high



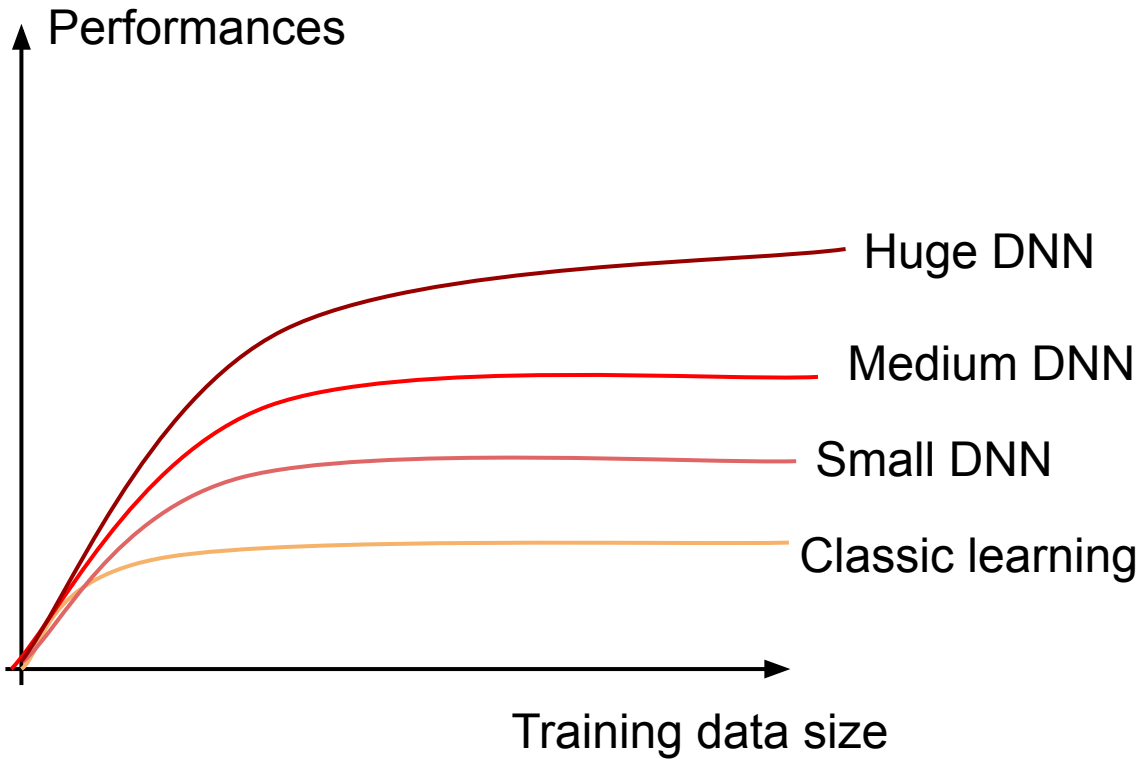
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Function approximators

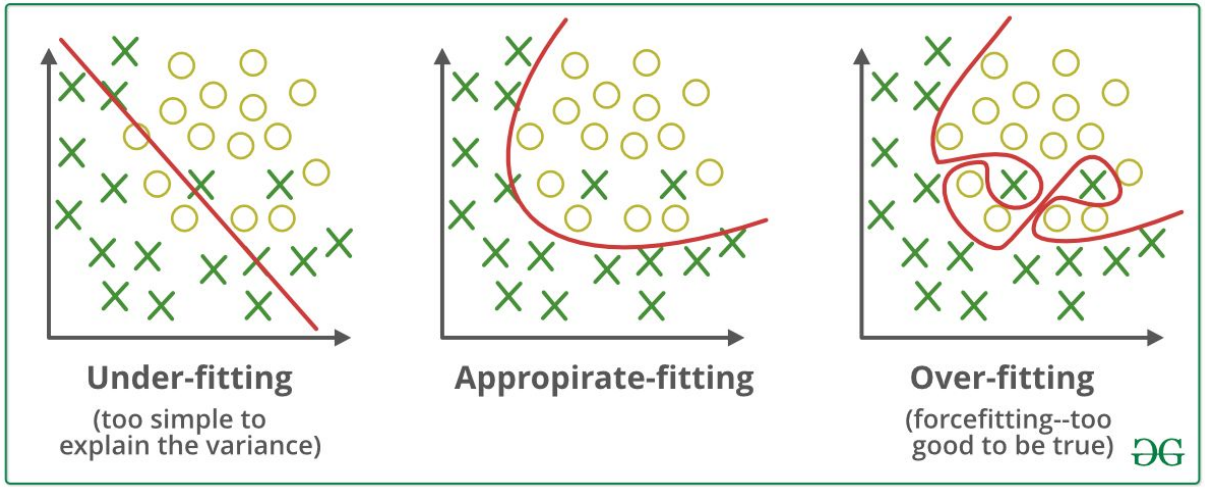


- (known) **quadratic function** (blue line)
- approximated with a **neural network model** [2 hidden layers with 10 nodes each] (orange line)

Size matters



The peril of overfitting



Key concepts

- Weights
- Forward and back propagation
- Loss function, cost function
- Gradient Descent/Optimizers
- Universal function approximation
- Data hungry/overfitting

Notebook on neural networks

Network optimization: EfficientNet

https://arxiv.org/abs/1905.11946

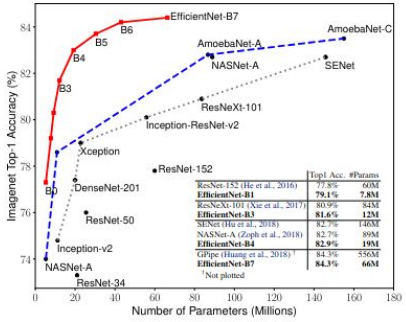
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan [1] Quoc V. Le [1]

Abstract

Convolutional Neural Networks (ConvNets) are commonly developed at a fixed resource budget, and then scaled up for better accuracy if more resources are available. In this paper, we systematically study model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Based on this observation, we propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective *compound coefficient*. We demonstrate the effectiveness of this method on scaling up MobileNets and ResNet.

To go even further, we use neural architec-



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) [1]	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

[1] Not plotted

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

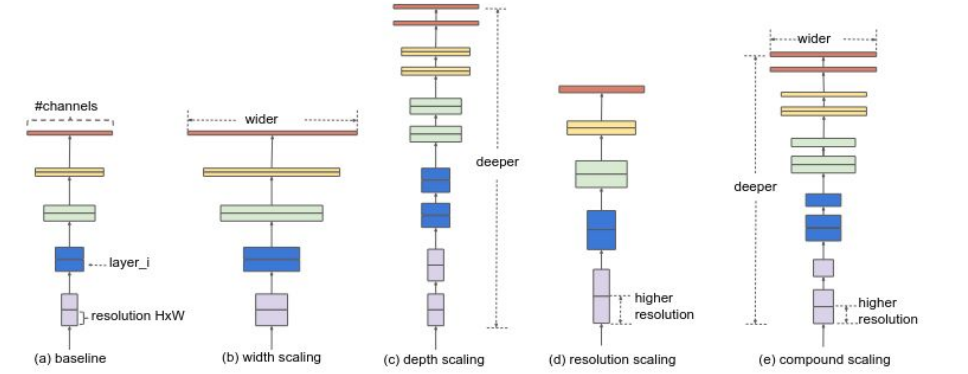
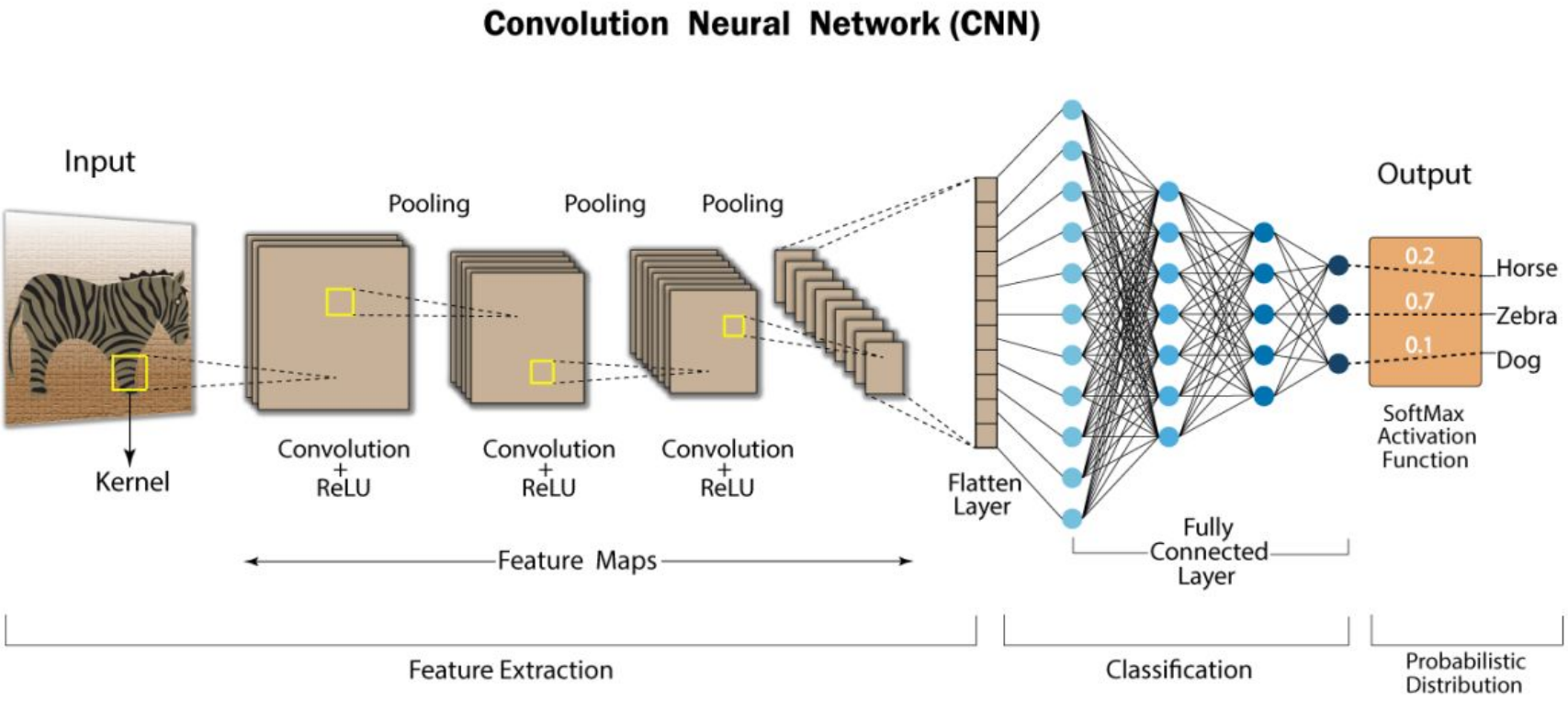


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

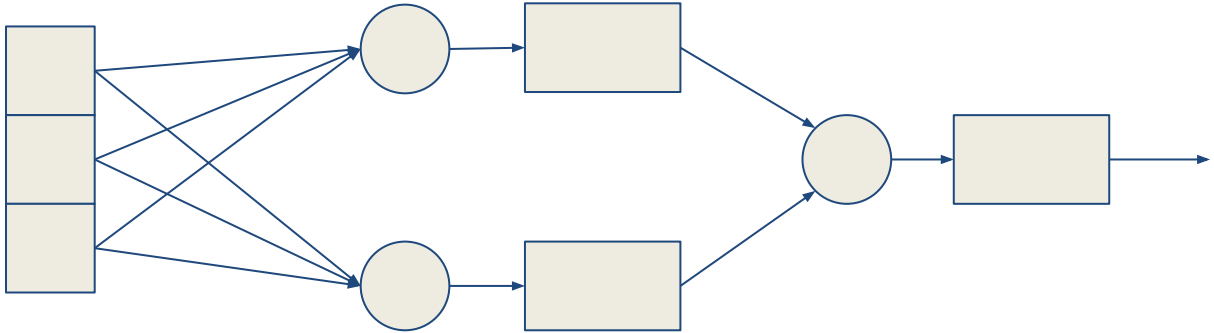
Convolutional neural networks



“Traditional” approach to longitudinal data

RNNs (Recurrent Neural Networks)

- LSTMs/GRUs (Long Short-Term Memory / Gated Recurrent Units)
- Limitations: vanishing gradients, slow training, difficulty modeling long-term dependencies.



The rise of transformers

- Transformers don't process sequentially, they use self-attention to look at the entire sequence at once.
- Originally developed for NLP (Natural Language Processing), but now used for time series, longitudinal data, etc.
- "Attention is All You Need" https://arxiv.org/abs/1706.03762
- Strengths: parallelizable, better modeling of long-range dependencies
- Always data-hungry
- Can be computationally expensive
- Beware of the hyperparameters flood

Key concept: tokens

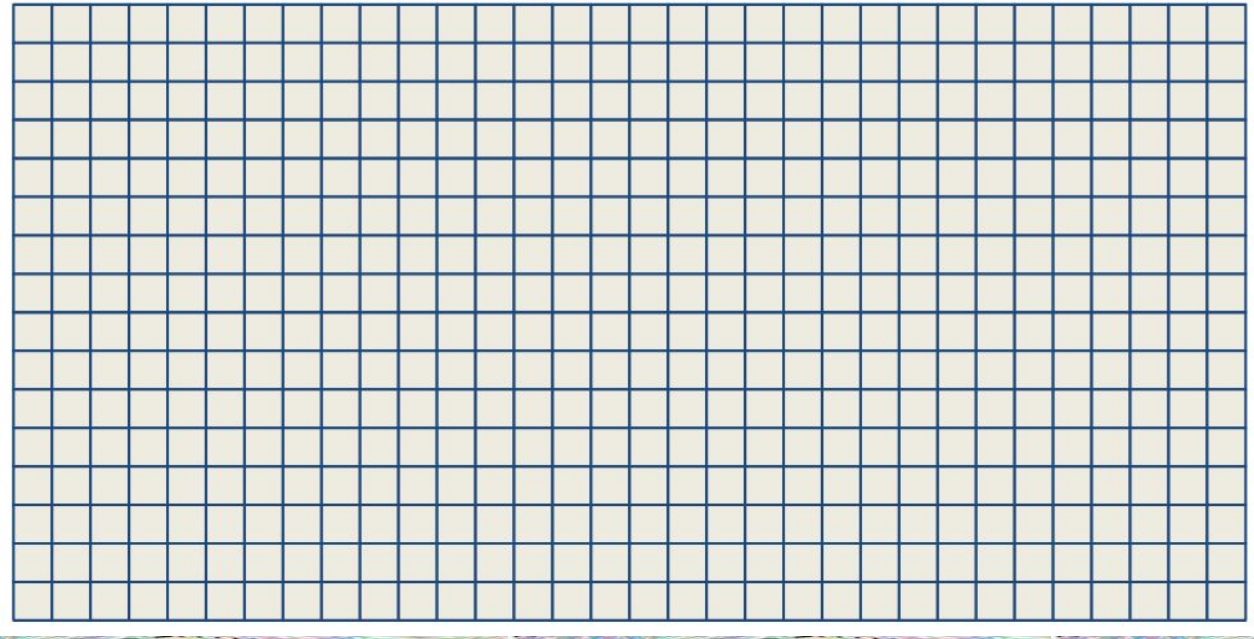
- Mapping input to a dictionary of known tokens (100k+)
- https://platform.openai.com/tokenizer

Key concept: embedding

- Mapping tokens to high dimensional space
  - chatGPT 3.0 → 12,288 dimensions
  - More recent models: undisclosed

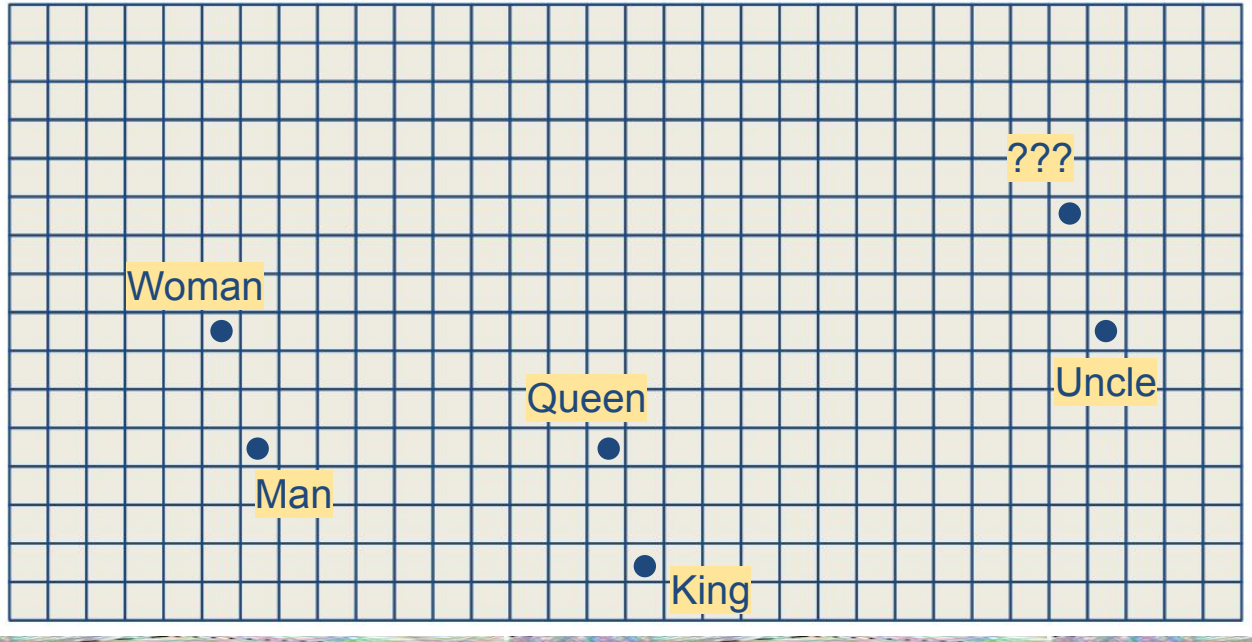
Key concept: embedding

- Mapping tokens to high dimensional space
  - chatGPT 3.0 → 12,288 dimensions
  - More recent models: undisclosed



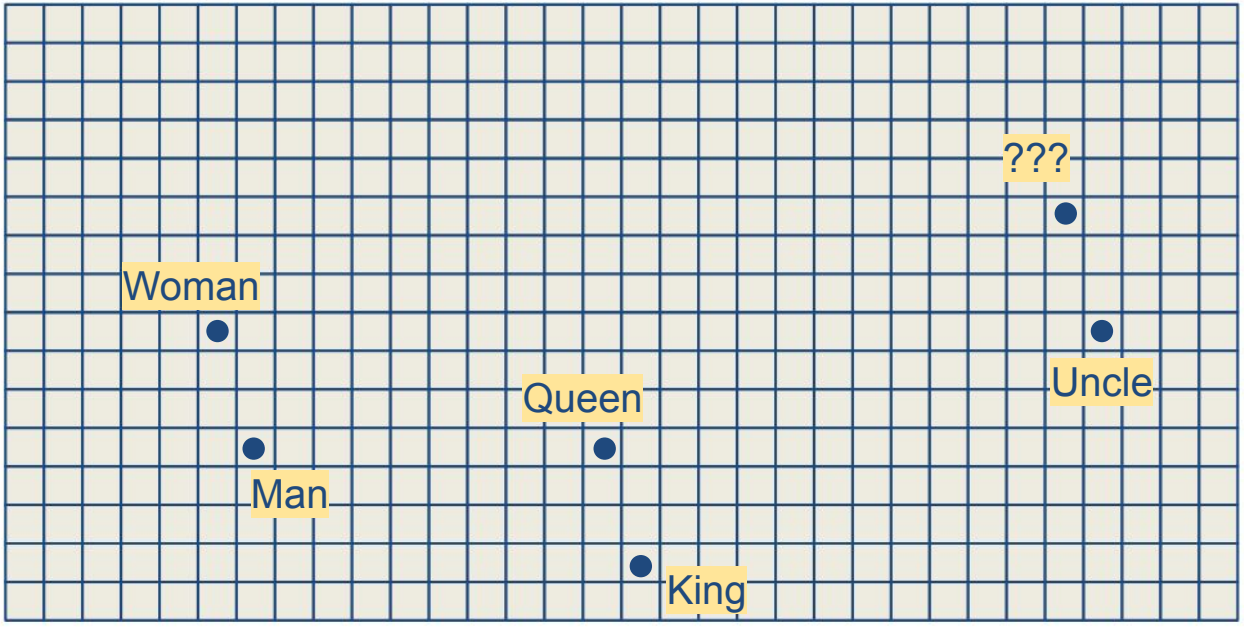
Key concept: embedding

- Mapping tokens to high dimensional space
  - chatGPT 3.0 → 12,288 dimensions
  - More recent models: undisclosed



Key concept: embedding

- Mapping tokens to high dimensional space
  - chatGPT 3.0 → 12,288 dimensions
  - More recent models: undisclosed



Man - Woman + Uncle = ???

Key concept: self attention

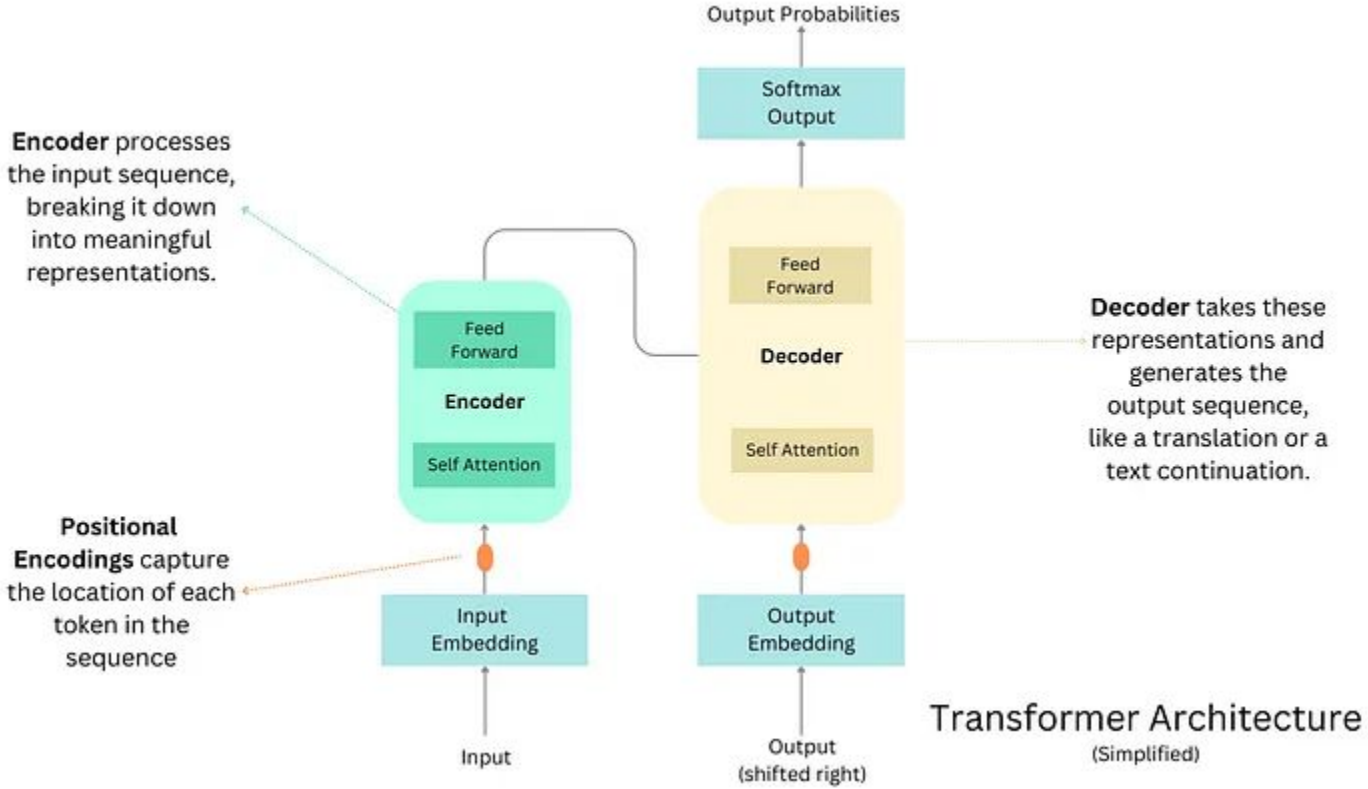
The cat run through the door because it was open

Key concept: self attention

The cat run through the door because it was open

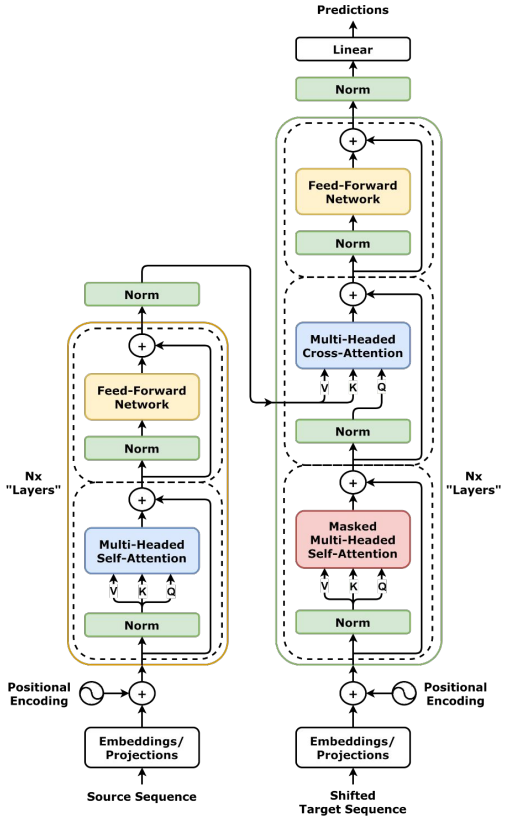
The cat run through the door because it was hungry

Transformer architecture



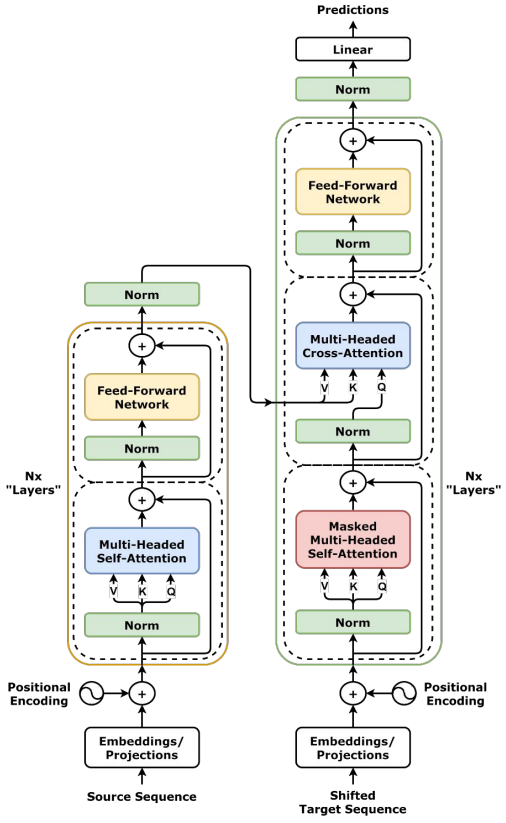
From https://medium.com/@theaveragegal/transformer-architecture-simplified-3fb501d461c8

Transformer architecture



Transformer architecture

- Input embeddings (including positional encodings — important for sequences!)
- Self-attention: each element attends to others with learned weights
- V (value), K (key), Q (query)
- Feed-forward networks applied to each element
- Layer normalization and residual connections for stability
- Context size is fixed



Transformers for sequential/longitudinal data

Adaptations to handle time information:

- Positional encoding
- Continuous-time embeddings (e.g., in patient data where time between visits is variable)

Examples of applications:

- Time Series Forecasting (e.g., stock prices, weather)
- Healthcare (e.g., modeling patient trajectories over multiple visits — models like Transformer for EHR data)
- Sensor data analysis (e.g., predictive maintenance)

Notebook on transformer architecture

Transformers for sequential/longitudinal data

- Time-series transformers (Informer, TimesNet, PatchTST, etc.)
- Clinical transformers (e.g., BEHRT, Med-BERT)
- Specialized designs: sometimes replace vanilla attention with local/global attention to handle very long sequences
- Model: ProtBERT (meta) / AlphaFold (deepmind)
    - These models treat amino acid sequences as text, where each amino acid is like a word. They use transformers (like BERT or GPT-style architectures) to model the dependencies between residues.

Notebook on prot BERT