ENSAE

# Deep learning

Unpacking Transformers, LLMs and image generation

## Session 2

# Syllabus

| Session | Lecture | TP |
|---|---|---|
| 1 | Intro to DL<br>Gradient descent and backprop | Intro to micrograd |
| 2 | DL fundamentals I<br>• Backprop<br>• Loss functions<br>• Neural Probabilistic Language Model (Bengio 2003) | Bigram model and MLP for next-character prediction |
| 3 | DL fundamentals II<br>• Activation function<br>• Regularization<br>• Initialization<br>• Residual networks<br>• Normalization<br><br>Recurrent Neural Networks | Backprop ninja<br>MLP in pytorch<br>(*) add batchnorm to TP2<br>(*) MLP for MNIST (Fleuret 3) |
| 4 | Attention and Transformers | GPT from scratch<br>(*) Fleuret practical 5 and 6 |
| 5 | DL for computer vision: convnets, unets | Convnets for Fashion MNIST<br>(*) Fleuret practical 4 |
| 6 | VAE & Diffusion models | 1D diffusion model<br>2D diffusion model<br>(*) train on GPU<br><br>**Quiz** |

onfido

# Let’s venture into the variations of a deep networks

Network architecture and inductive bias

Loss function

Activation function

Regularization

Initialization

Residual networks

Batch norm, layer norm

# Activation functions

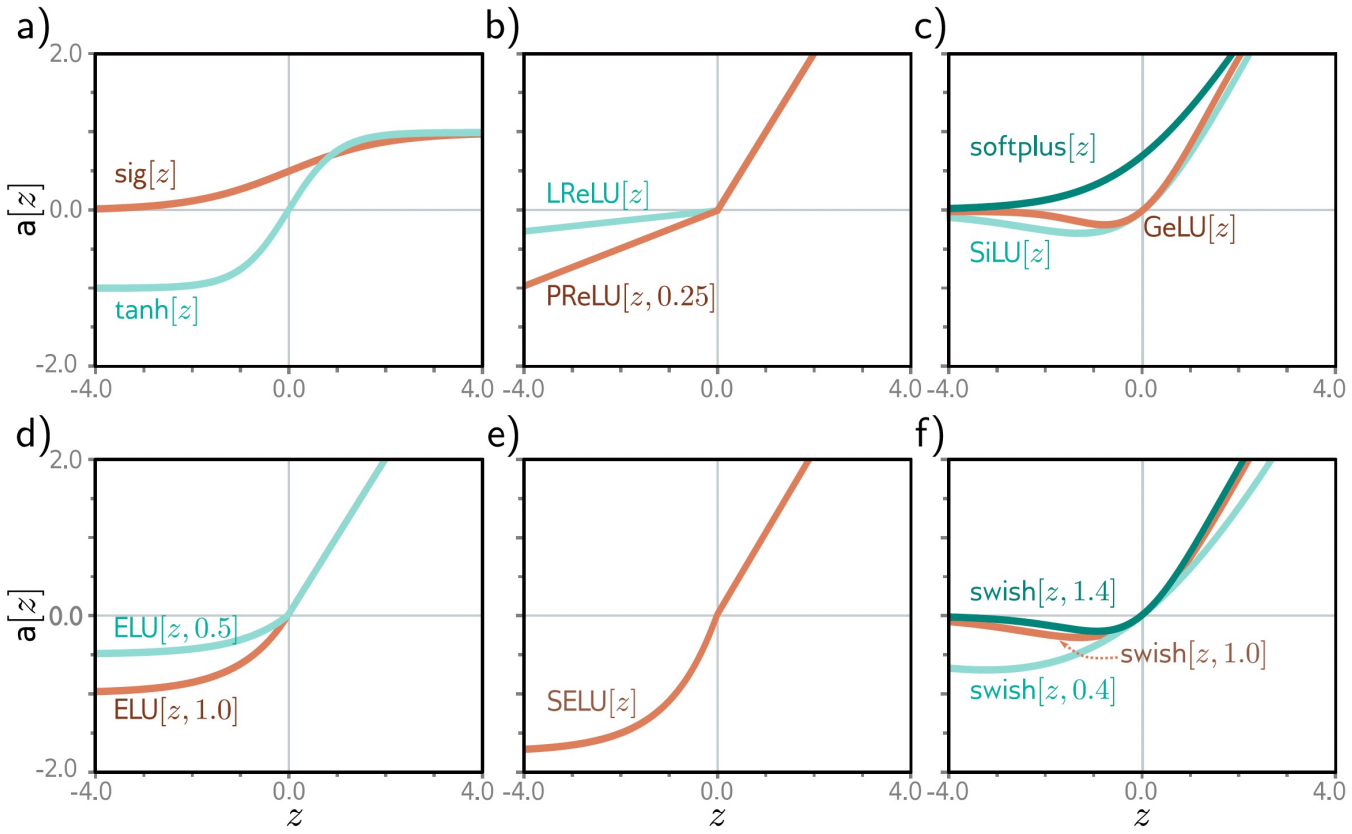



**Figure 3.13** Activation functions. a) Logistic sigmoid and tanh functions. b) Leaky ReLU and parametric ReLU with parameter 0.25. c) SoftPlus, Gaussian error linear unit, and sigmoid linear unit. d) Exponential linear unit with parameters 0.5 and 1.0, e) Scaled exponential linear unit. f) Swish with parameters 0.4, 1.0, and 1.4.

Source: Understanding Deep Learning, S. Prince, 2023

# Activation functions

Sigmoid: not blowing up activation

ReLU : not vanishing gradient

ReLU : More computationally efficient to compute than Sigmoid like functions since ReLU just needs to pick max(0,$x$) and not perform expensive exponential operations as in sigmoids

ReLU : In practice, networks with ReLU tend to show better convergence performance than sigmoid. (Krizhevsky et al.)

By default, use ReLU

ImageNet Classification with Deep Convolutional
Neural Networks, Krizhevsky, Sutskever and Hinton, NIPS 2012

# Let’s venture into the variations of a deep networks

Network architecture and inductive bias

Loss function

Activation function

Regularization

Initialization

Residual networks

Batch norm, layer norm

# Regularization

Regularization is about reducing the generalization gap between training and testing performance.

Implicit regularization is baked into SGD.

Explicit regularization is added through various methods (penalty term, data augmentation, dropout, etc.)

## Explicit regularization: adding a penalty term to the loss function

$$\hat{\omega} = \underset{\omega}{\text{argmax}} \left[ \prod_{i=1}^{N} Pr(y_i | x_i, \omega) \right]$$

$$\hat{\omega} = \underset{\omega}{\text{argmax}} \left[ \prod_{i=1}^{N} Pr(y_i | x_i, \omega) \Pr(\omega) \right]$$

$$\hat{\omega} = \underset{\omega}{\text{argmin}} \left[ - \sum_{i=1}^{N} \log[Pr(y_i | x_i, \omega)] + \log(\Pr(\omega)) \right]$$

$$\hat{\omega} = \underset{\omega}{\text{argmin}} \left[ - \sum_{i=1}^{N} \ell_i[x_i, y_i] + \lambda \cdot g(\omega) \right]$$

penalty term

## Explicit regularization: adding a penalty term to the loss function

$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}} \left[ -\sum_{i=1}^{N} \ell_i[x_i, y_i] + \lambda \cdot \sum \omega_j^2 \right]$$

L2 loss

# Regularization

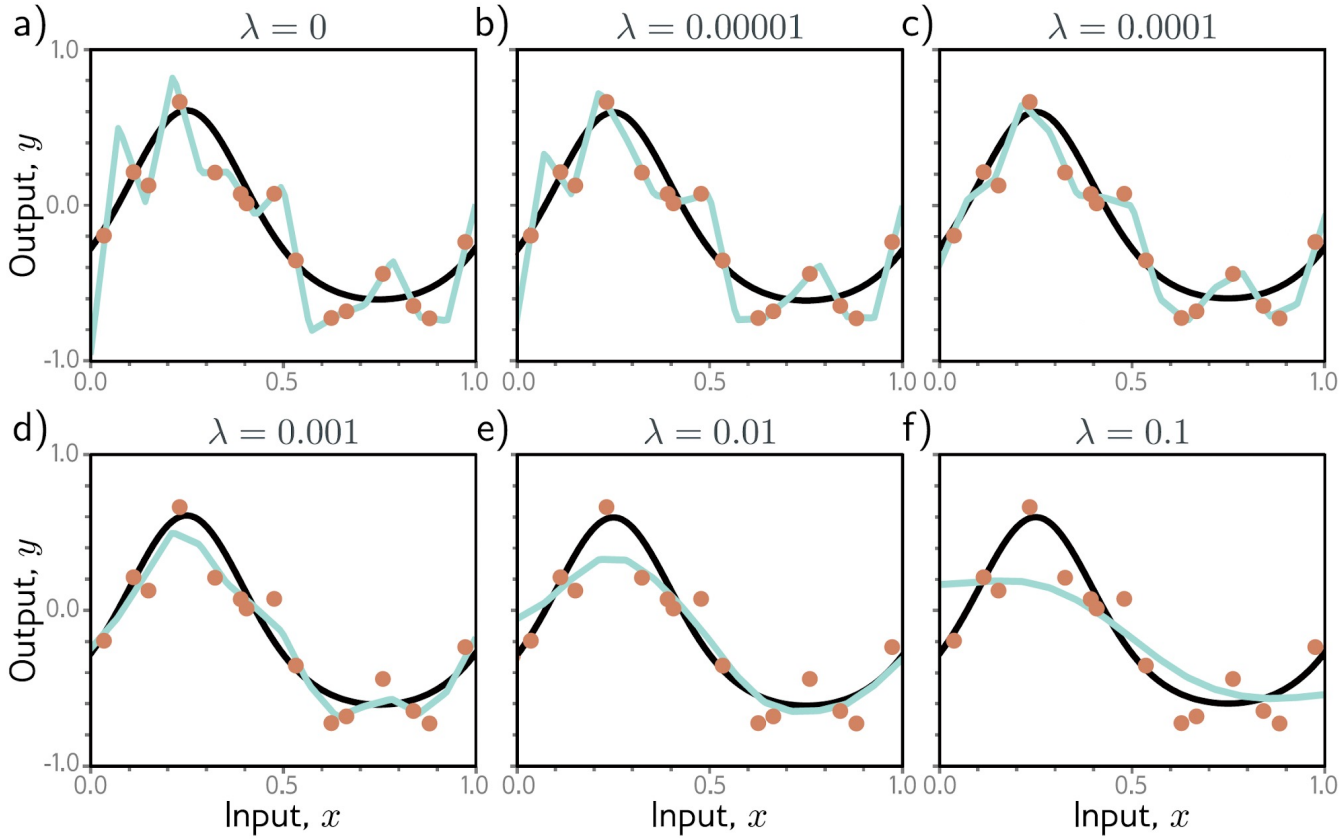



**Figure 9.2** L2 regularization in simplified network (see figure 8.4). a–f) Fitted functions as we increase the regularization coefficient $\lambda$. The black curve is the true function, the orange circles are the noisy training data, and the cyan curve is the fitted model. For small $\lambda$ (panels a–b), the fitted function passes exactly through the data points. For intermediate $\lambda$ (panels c–d), the function is smoother and more similar to the ground truth. For large $\lambda$ (panels e–f), the fitted function is smoother than the ground truth, so the fit is worse.

Source: Understanding Deep Learning, S. Prince, 2023

## Implicit regularization due to gradient descent

$$\omega_{t+1} = \omega_t - \eta \cdot \frac{\partial L}{\partial \omega}$$

$$L_{GD}[\omega] = L[\omega] + \frac{\eta}{4}\left\|\frac{\partial L}{\partial \omega}\right\|^2$$

implicit penalty

## Implicit regularization due to stochastic gradient descent

If we denote $L$ the average loss overall all samples and $L_B$ the average loss over all batches:

$$L_{SGD}[\omega] = L_{GD}[\omega] + \frac{\eta}{4B}\sum_{b=1}^{B}\left\|\frac{\partial L_B}{\partial \omega} - \frac{\partial L}{\partial \omega}\right\|^2$$

$$L_{SGD}[\omega] = L[\omega] + \frac{\eta}{4}\left\|\frac{\partial L}{\partial \omega}\right\|^2 + \frac{\eta}{4B}\sum_{b=1}^{B}\left\|\frac{\partial L_B}{\partial \omega} - \frac{\partial L}{\partial \omega}\right\|^2$$

*Larger steps regularize more*

*Smaller batches regularize more*

Larger learning rate regularize more

Smaller batches regularize more

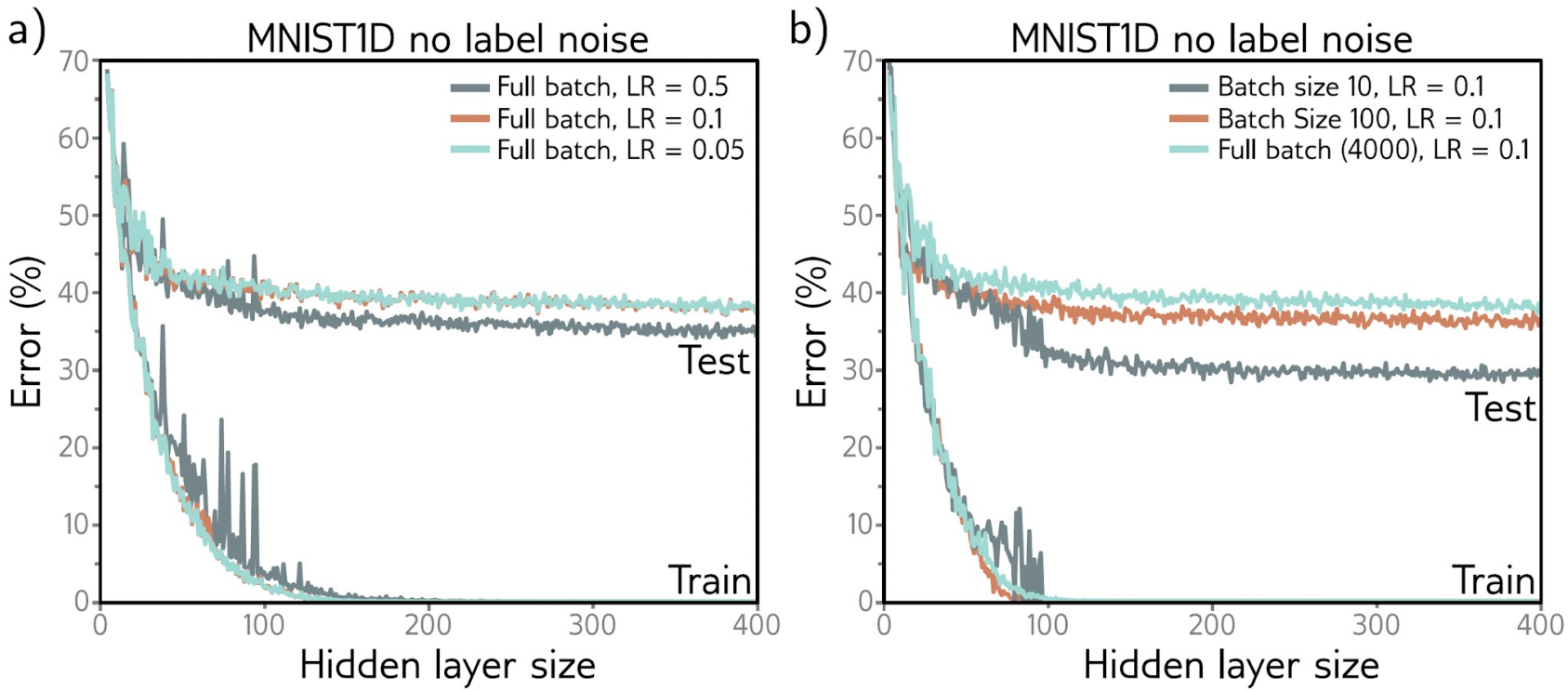



**Figure 9.5** Effect of learning rate and batch size for 4000 training and 4000 test examples from MNIST-1D (see figure 8.1) for a neural network with two hidden layers. a) Performance is better for large learning rates than for intermediate or small ones. In each case, the number of iterations is 6000× the learning rate, so each solution has the opportunity to move the same distance. b) Performance is superior for smaller batch sizes. In each case, the number of iterations was chosen so that the training data were memorized at roughly the same model capacity.

Source: Understanding Deep Learning, S. Prince, 2023

# Regularization

Heuristics to add regularization:

- Early stopping
- Ensembling
- Dropout
- Data augmentation

# Regularization: summary

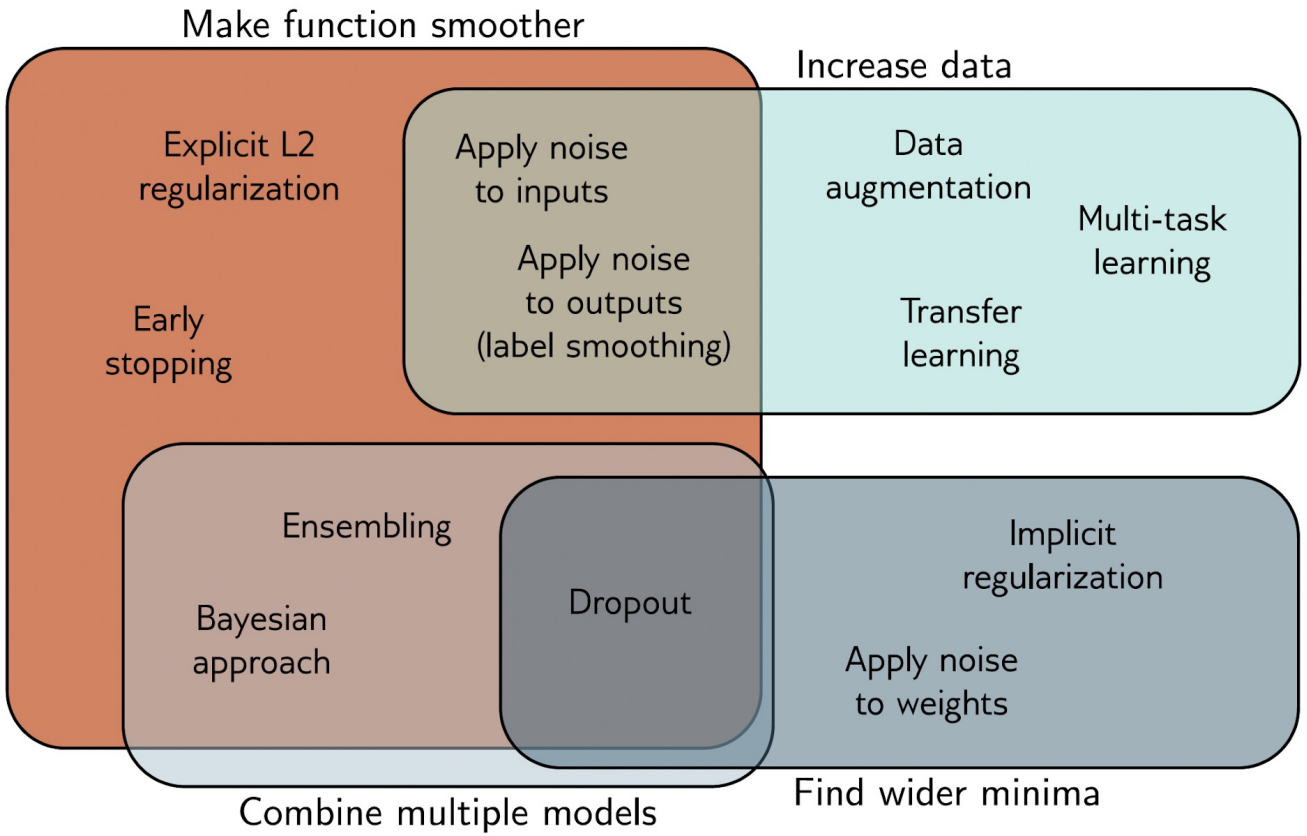



**Figure 9.14** Regularization methods. The regularization methods discussed in this chapter aim to improve generalization by one of four mechanisms. Some methods aim to make the modeled function smoother. Other methods increase the effective amount of data. The third group of methods combine multiple models and hence mitigate against uncertainty in the fitting process. Finally, the fourth group of methods encourages the training process to converge to a wide minimum where small errors in the estimated parameters are less important (see also figure 20.11).

Source: Understanding Deep Learning, S. Prince, 2023

# Let’s venture into the variations of a deep networks

Network architecture and inductive bias

Loss function

Activation function

Regularization

Initialization

Residual networks

Batch norm, layer norm

# Initialization: forward pass

At each layer $k$, given weights $\boldsymbol{\Omega}$ with variance $\boldsymbol{\sigma}_{\boldsymbol{\Omega}}^{\boldsymbol{2}}$ and pre-activations $\boldsymbol{f}_k$:

$$\boldsymbol{f}_k = \boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \cdot a[\boldsymbol{f}_{k-1}]$$

If $\boldsymbol{\sigma}_{\boldsymbol{\Omega}}^{\boldsymbol{2}}$ is too large → exploding gradients

If $\boldsymbol{\sigma}_{\boldsymbol{\Omega}}^{\boldsymbol{2}}$ is too small → vanishing gradients

# Initialization: forward pass

At each layer $k$, given weights $\boldsymbol{\Omega}$ with variance $\boldsymbol{\sigma_{\Omega}^2}$ and pre-activations $\boldsymbol{f}_k$:

$$\boldsymbol{\sigma_{f_{k+1}}^2 = \frac{1}{2} D_{h_k} \sigma_{\Omega}^2 \sigma_{f_k}^2}$$

where $\boldsymbol{D_{h_k}}$ is the dimensionality of the input layer $k$.

Hence the optimal variance of the weights is:

$$\boldsymbol{\sigma_{\Omega}^2 = \frac{2}{D_{h_k}}}$$

*He initialization*

# Initialization: backward pass

Similarly, the optimal variance of the weights for the backward pass is:

$$\sigma_{\Omega}^{2} = \frac{2}{D_{h_{k+1}}}$$

Overall, the optimal variance of the weights is:

$$\sigma_{\Omega}^{2} = \frac{4}{D_{h_k} + D_{h_{k+1}}}$$

Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, He et al., ICCV 2015

# Let’s venture into the variations of a deep networks

Network architecture and inductive bias

Loss function

Activation function

Regularization

Initialization

Residual networks

Batch norm, layer norm

# Residual networks

Gradient descent assumes that the function is smooth.

Unfortunately, the loss becomes less and less smooth with more depth (shattered gradients).

# Residual networks

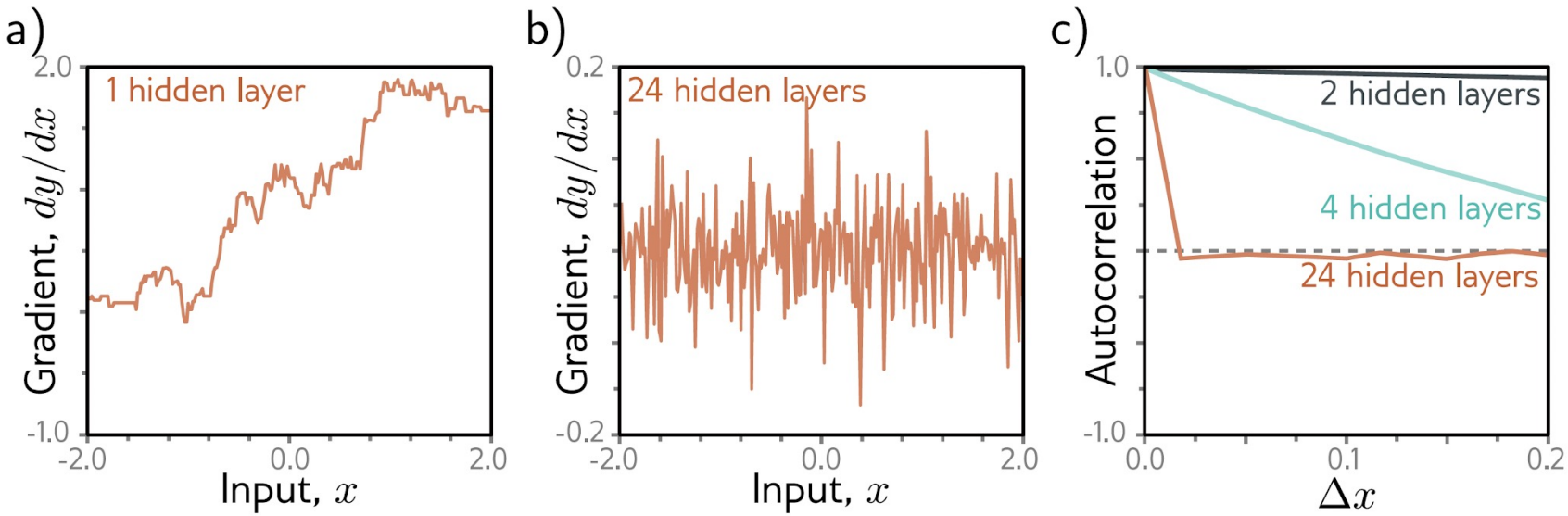



**Figure 11.3** Shattered gradients. a) Consider a shallow network with 200 hidden units and Glorot initialization (He initialization without the factor of two) for both the weights and biases. The gradient $\partial y/\partial x$ of the scalar network output $y$ with respect to the scalar input $x$ changes relatively slowly as we change the input $x$. b) For a deep network with 24 layers and 200 hidden units per layer, this gradient changes very quickly and unpredictably. c) The autocorrelation function of the gradient shows that nearby gradients become unrelated (have autocorrelation close to zero) for deep networks. This *shattered gradients* phenomenon may explain why it is hard to train deep networks. Gradient descent algorithms rely on the loss surface being relatively smooth, so the gradients should be related before and after each update step. Adapted from Balduzzi et al. (2017).

Source: Understanding Deep Learning, S. Prince, 2023

# Residual networks

*More depth is not always better!*

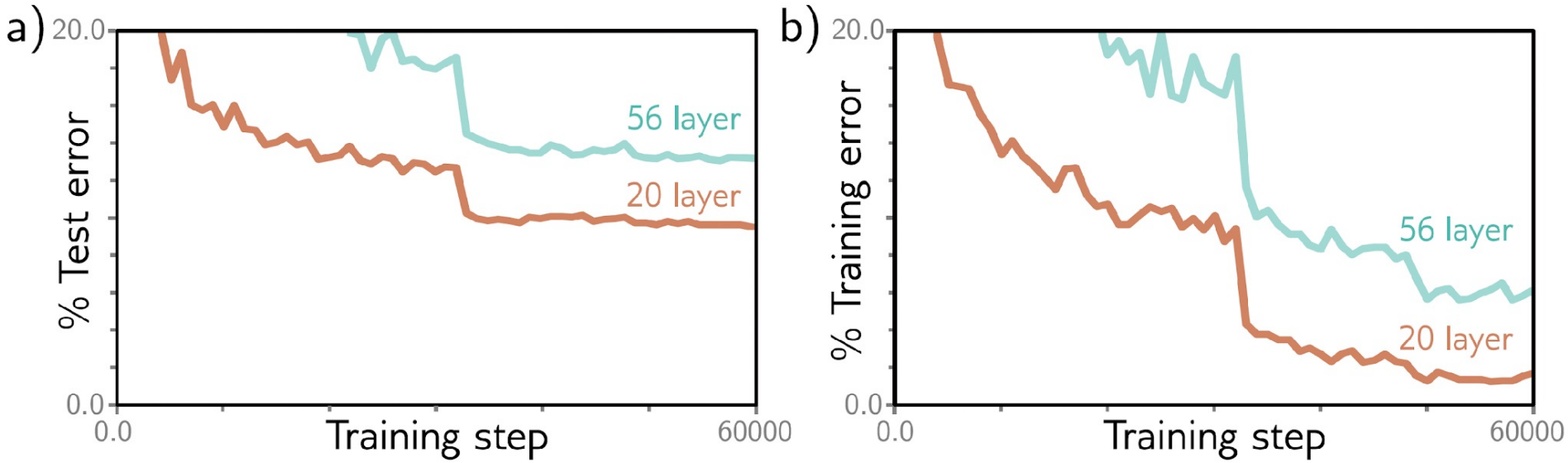



**Figure 11.2** Decrease in performance when adding more convolutional layers. a) A 20-layer convolutional network outperforms a 56-layer neural network for image classification on the test set of the CIFAR-10 dataset (Krizhevsky & Hinton, 2009). b) This is also true for the training set, which suggests that the problem relates to training the original network rather than a failure to generalize to new data. Adapted from He et al. (2016a).

Source: Understanding Deep Learning, S. Prince, 2023

# Residual networks

This can be addressed with skip connections.

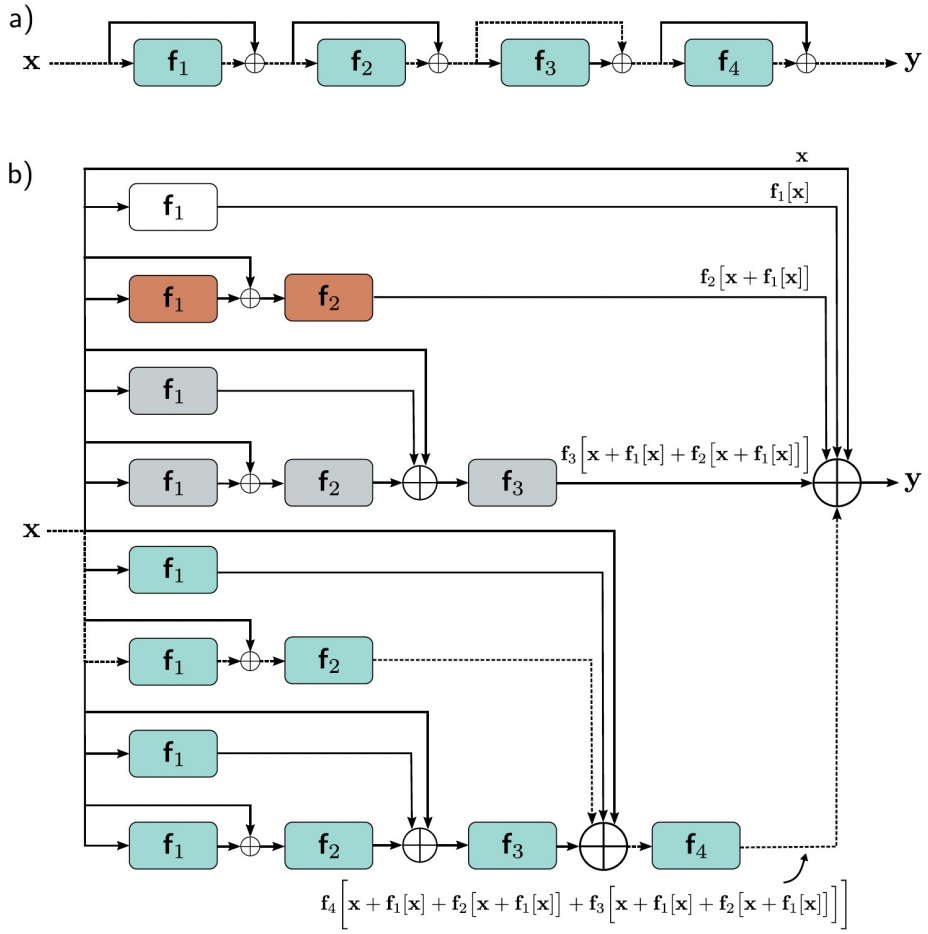



**Figure 11.4** Residual connections. a) The output of each function $\mathbf{f}_k[\mathbf{x}, \boldsymbol{\phi}_k]$ is added back to its input, which is passed via a parallel computational path called a residual or skip connection. Hence, the function computes an additive change to the representation. b) Upon expanding (unraveling) the network equations, we find that the output is the sum of the input plus four smaller networks (depicted in white, orange, gray, and cyan, respectively, and corresponding to terms in equation 11.5); we can think of this as an ensemble of networks. Moreover, the output from the cyan network is itself a transformation $\mathbf{f}_4[\bullet, \boldsymbol{\phi}_4]$ of another ensemble, and so on. Alternatively, we can consider the network as a combination of 16 different paths through the computational graph. One example is the dashed path from input $\mathbf{x}$ to output $\mathbf{y}$, which is the same in panels (a) and (b).

Source: Understanding Deep Learning, S. Prince, 2023

# Residual networks

## This can be addressed with skip connections.

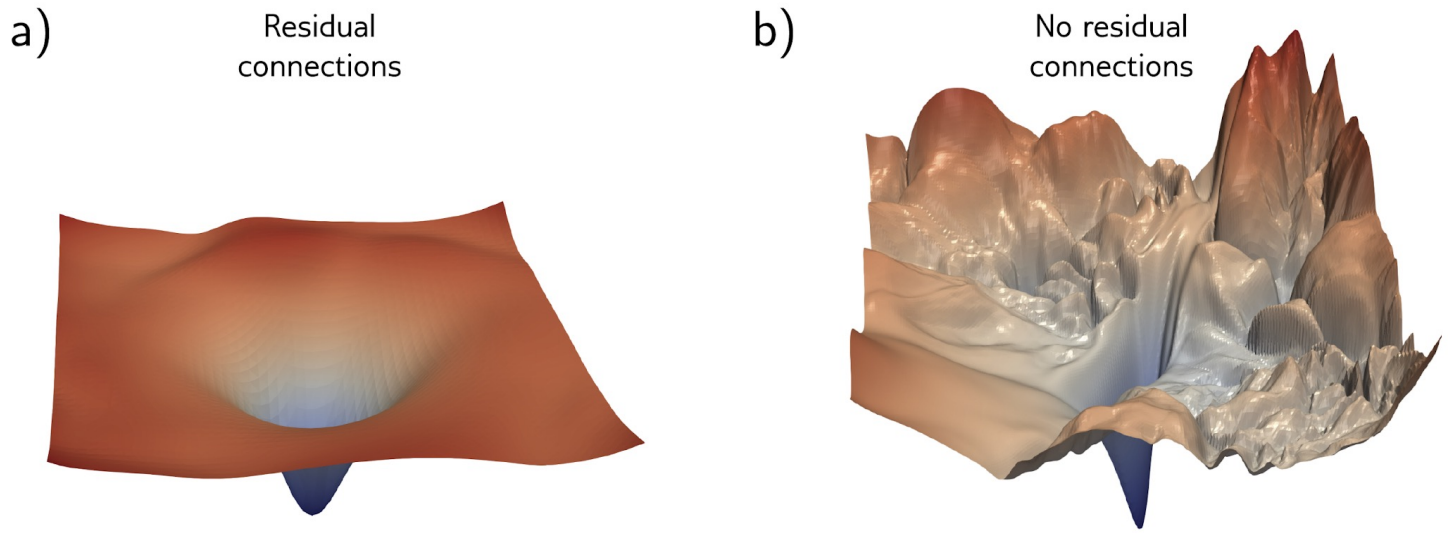



**Figure 11.13** Visualizing neural network loss surfaces. Each plot shows the loss surface in two random directions in parameter space around the minimum found by SGD for an image classification task on the CIFAR-10 dataset. These directions are normalized to facilitate side-by-side comparison. a) Residual net with 56 layers. b) Results from the same network without skip connections. The surface is smoother with the skip connections. This facilitates learning and makes the final network performance more robust to minor errors in the parameters, so it will likely generalize better. Adapted from Li et al. (2018b).

Source: Understanding Deep Learning, S. Prince, 2023

# Residual networks

But then we need to update the order of operations.

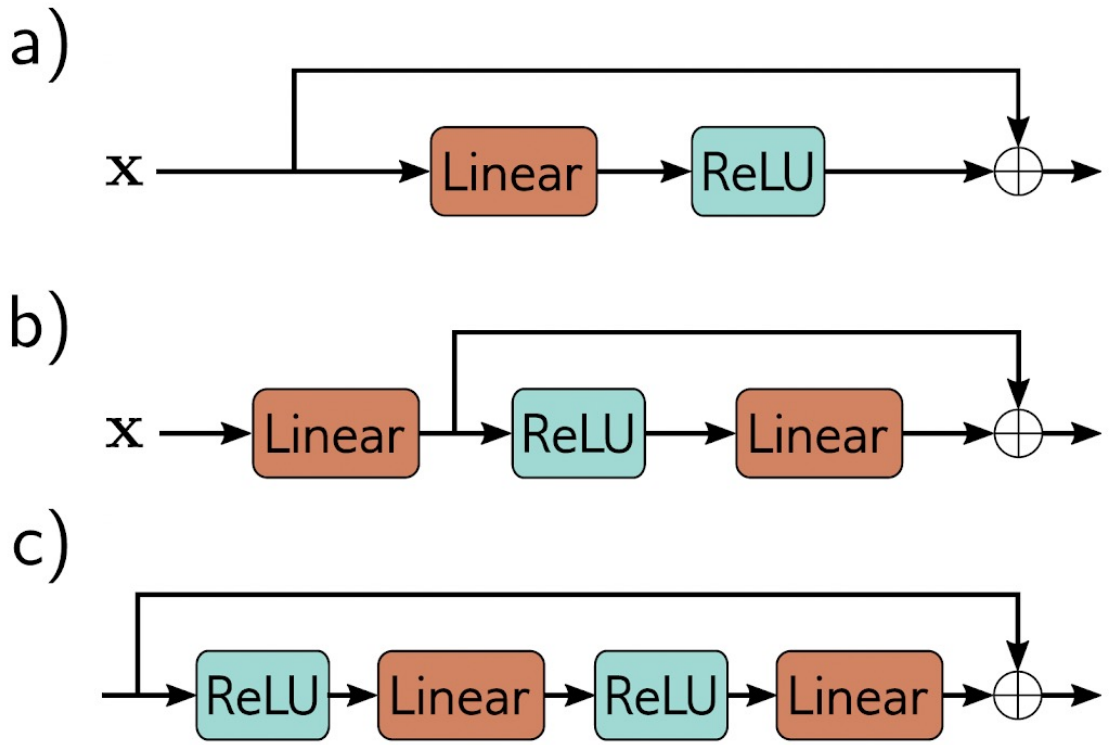



**Figure 11.5** Order of operations in residual blocks. a) The usual order of linear transformation or convolution followed by a ReLU nonlinearity means that each residual block can only add non-negative quantities. b) With the reverse order, both positive and negative quantities can be added. However, we must add a linear transformation at the start of the network in case the input is all negative. c) In practice, it's common for a residual block to contain several network layers.

Source: Understanding Deep Learning, S. Prince, 2023

*But then the variance doubles at every layer! -> Batch norm*

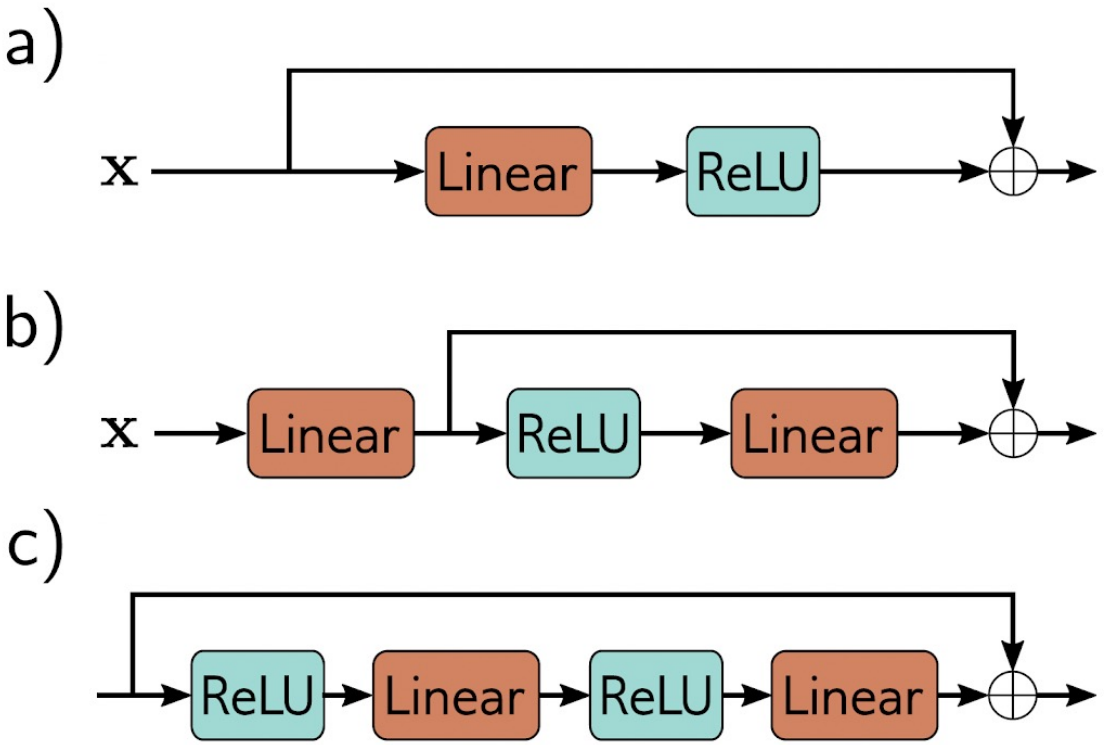



**Figure 11.5** Order of operations in residual blocks. a) The usual order of linear transformation or convolution followed by a ReLU nonlinearity means that each residual block can only add non-negative quantities. b) With the reverse order, both positive and negative quantities can be added. However, we must add a linear transformation at the start of the network in case the input is all negative. c) In practice, it's common for a residual block to contain several network layers.

Source: Understanding Deep Learning, S. Prince, 2023

# Let’s venture into the variations of a deep networks

Network architecture and inductive bias

Loss function

Activation function

Regularization

Initialization

Residual networks

Batch norm, layer norm

# Batch norm

Batch Norm addresses the exploding gradient problem.

Alternative intuition:

We initialize the weights so that they have a nice distribution.

Why don’t we do this at each pass then? ☺

That’s Batch Norm!

Introduced by Ioff & Szegedy (2015)

# Batch norm

Shift and scale each activation so that their mean and variance across the batch

become values that are learned during training.

Not constant values!

# Batch norm

Compute $m_B$ and $s_B$ (mean and variance) over the batch during training

Normalize activations $h_i = \frac{h_i - m_h}{s_h + \epsilon}$

Scale and shift: $h_i = \gamma \cdot h_i + \delta$

$\gamma$ and $\delta$ are learned during training, for each hidden unit (not each layer)

For $K$ layers containing each $D$ units, that's $2 * K * D$ extra parameters.

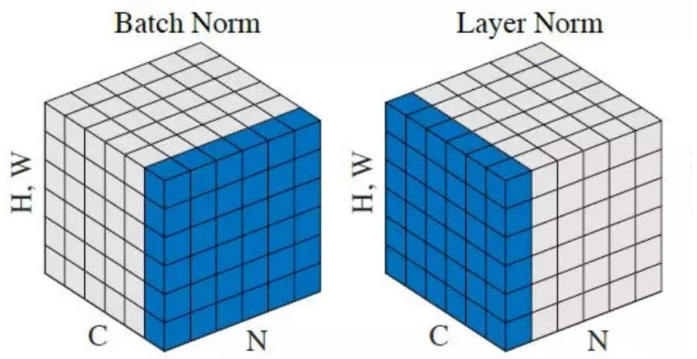
# Batch norm

Downsides of Batch Norm

- Prone to bugs (need to freeze during inference)
- More parameters to learn
- Introduces dependencies between the training samples
- Needs to recompute the statistics on the whole dataset at testing time

# Layer norm

Compute the layer normalization statistics over all the hidden units in the same layer.




All the hidden units in a layer share the same normalization terms
μ and σ, but different training cases have different normalization terms.

Introduced by Ba et al. in Layer Normalization

# Conclusion

The choice of the loss function is critical.

Deep learning adds inductive bias.

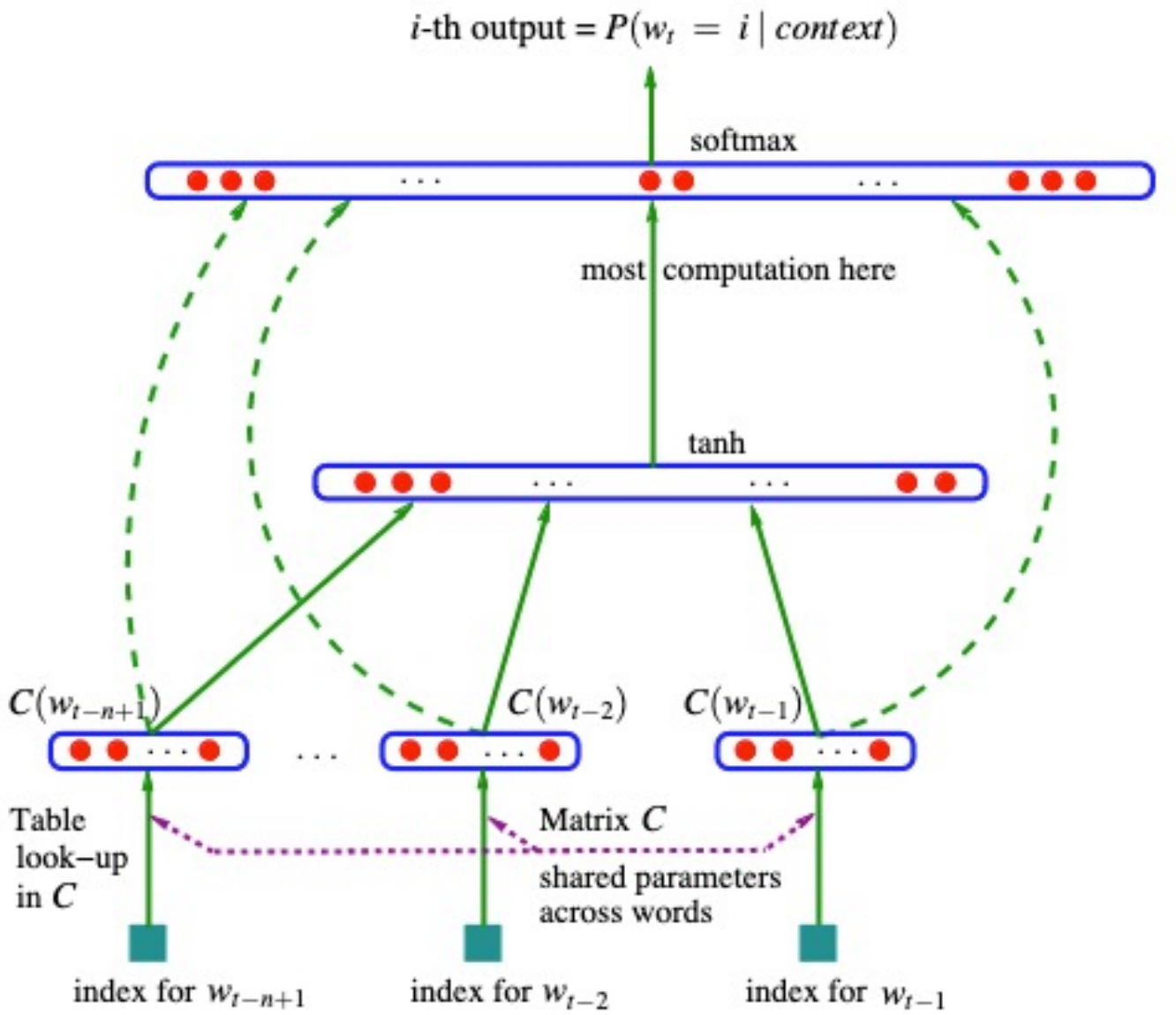
Deep learning is not supposed to work (in theory).

Yet, it works thanks to:

- Activation function
- Regularization
- Initialization
- Residual networks
- Normalization

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

Neural Probabilistic Langu

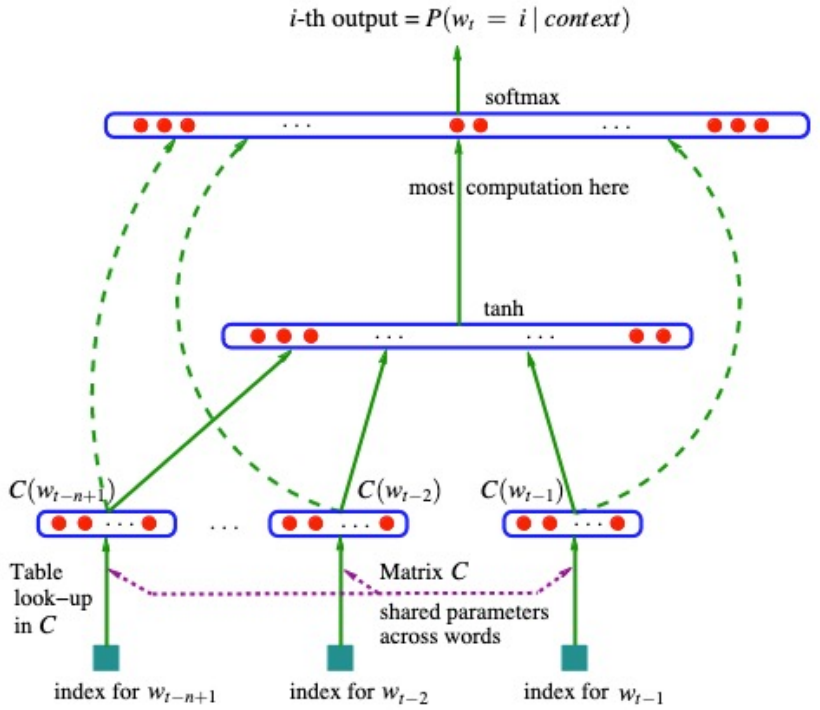



A Neural Probabilistic Language Model, I

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

## Neural Probabilistic Language Model

*Main limitation: fixed length context*




A Neural Probabilistic Language Model, Bengio et al, 2003

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

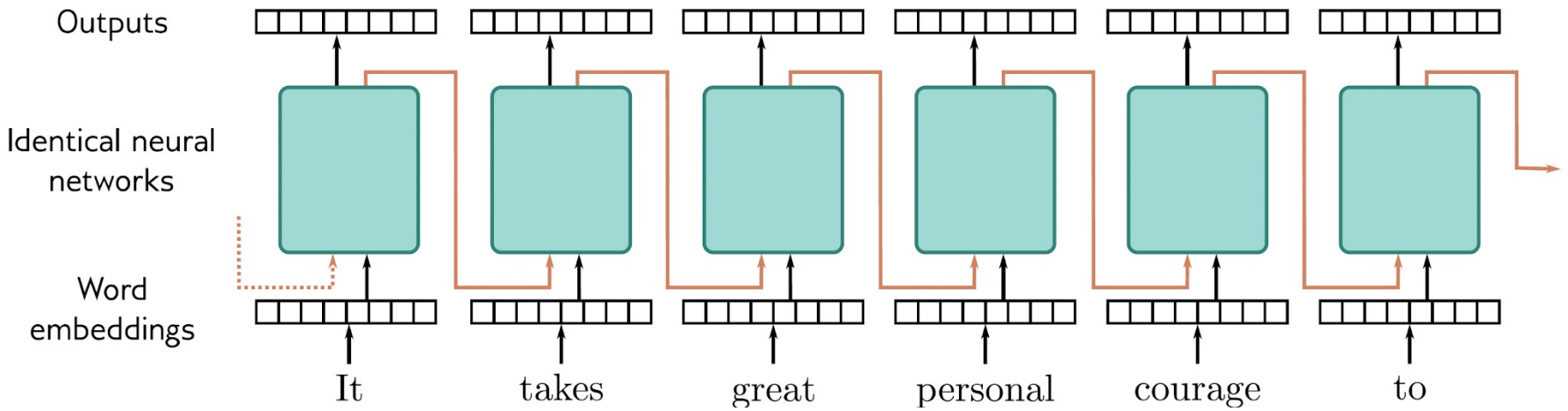



**Figure 12.19** Recurrent neural networks (RNNs). The word embeddings are passed sequentially through a series of identical neural networks. Each network has two outputs; one is the output embedding, and the other (orange arrows) feeds back into the next neural network, along with the next word embedding. Each output embedding contains information about the word itself and its context in the preceding sentence fragment. In principle, the final output contains information about the entire sentence and could be used to support classification tasks similarly to the <cls> token in a transformer encoder model. However, RNNs sometimes gradually "forget" about tokens that are further back in time.

Source: Understanding Deep Learning, S. Prince, 2023

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

RNNs

*Input layer*

$$x(t) = w(t) + s(t-1)$$

*Context / hidden state*

$$s_j(t) = f\left(\sum_i x_i(t) u_{ji}\right)$$

*Output layer*

$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right)$$

Recurrent neural network based language model, Mikolov et al, InterSpeech, 2010

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

Standard RNNs are unable to store information about past inputs for very long.

Learning Long-Term Dependencies with Gradient Descent is Difficult, Bengio et al, 1994.

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

LSTM

Long Short-term Memory is an RNN architecture designed to
be better at storing and accessing information than standard RNNs.

Generating Sequences With Recurrent Neural Networks, A. Graves, 2014

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

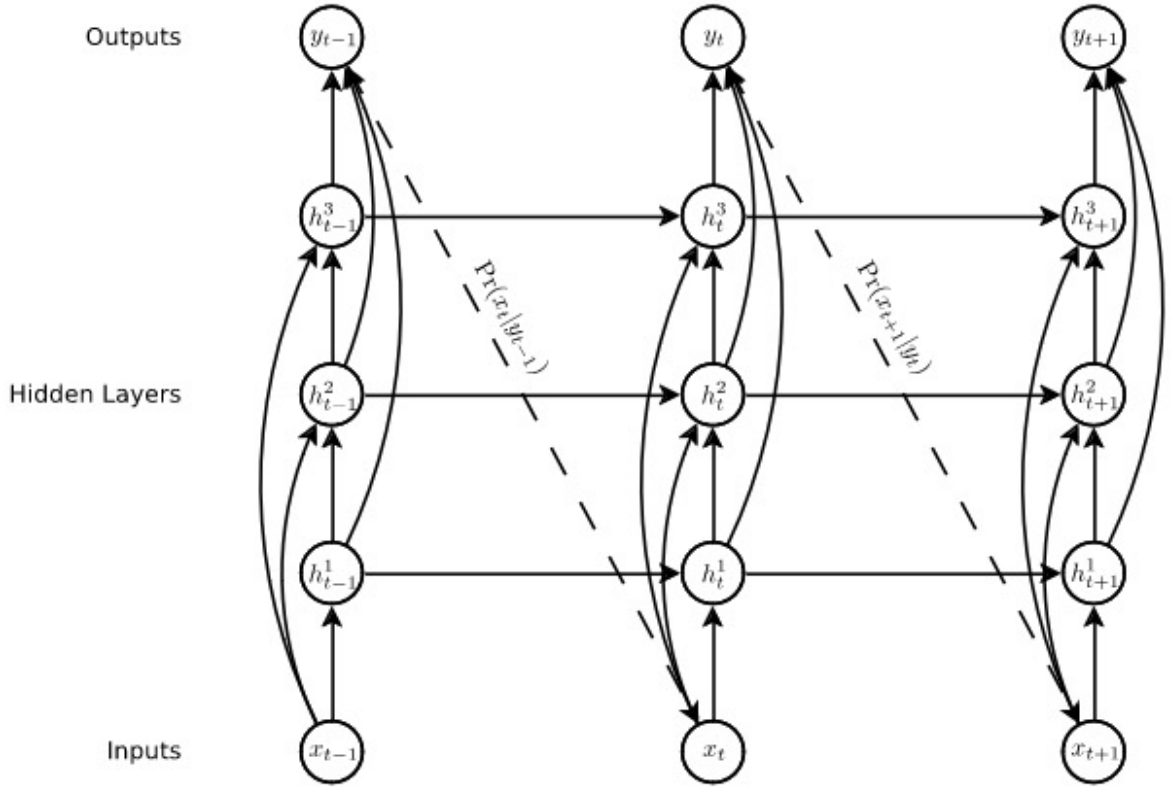



Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.

Generating Sequences With Recurrent Neural Networks, A. Graves, 2014

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

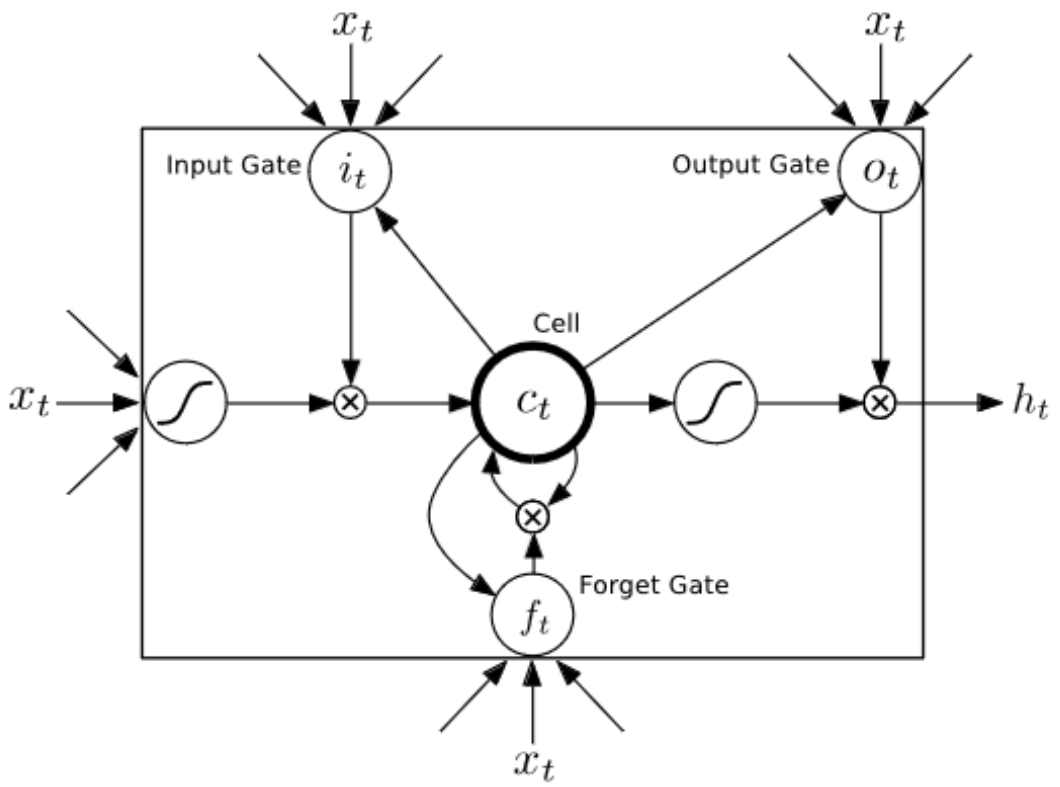



Figure 2: **Long Short-term Memory Cell**

Generating Sequences With Recurrent Neural Networks, A. Graves, 2014

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

## Better LSTMs

Sequence to sequence learning with neural networks, Sutskever, I., Vinyals, O., and Le, Q. , NIPS 2014.

Neural Machine Translation by Jointly Learning to Align and Translate, D. Bahdanau, K. Cho, Y. Bengio, 2015

## GRUs (Gated Recurrent Units)

On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, K. Cho et al, 2014

# Going beyond N-gram with Recurrent Neural Networks (RNNs)

GRU and LSTM seem comparable in performance.

They suffer from the same limitations:

Compressing the input sequence into a single vector.

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, J. Chung, C. Gulcehre, K. Cho, Y. Bengio, 2014.

The restaurant refused to serve me a ham sandwich because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and the service.

The restaurant refused to serve me a ham sandwich because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and the service.

Main constraints of NLP

1. NLP tasks often induce variable-length inputs.

2. Language is ambiguous: we need words/tokens to pay attention to each other.

3. Brute-force encoding the input generates large vectors: 37 words represented by 1024 embeddings -> 37,888 dimensions.

We need an efficient communication mechanism between words/tokens.