Chapter 10
Complexity of Machine Learning

Without doubt, the brain is the most complex adaptive system known to humanity, arguably also a complex system about which we know little. In both respects, the brain faces increasing competition from machine learning architectures.

We present an introduction to basic neural network and machine learning concepts, with a special focus on the connection to dynamical systems theory. Starting with point neurons and the XOR problem, the relation between the dynamics of recurrent networks and random matrix theory will be developed. The somewhat counter-intuitive notion of continuous numbers of network layers is shown next to lead to neural differential equations, respectively for information processing and error backpropagation. Approaches aimed at understanding learning processes in deep architectures make often use of the infinite-layer limit. As a result, machine learning can be described by Gaussian processes together with neural tangent kernels. Finally, the distinction between information processing and information routing will be discussed, with the latter being the task of the attention mechanism, the core component of transformer architectures.

10.1 Computation Units

Modern day computation devices and architectures are constructed using large numbers of local computation units. Examples are transistors and quantum gates for classical and quantum computers, or artificial neurons for deep learning algorithms.

Dimensionality Reduction Information processing by local computation units intrinsically involves dimensionality reduction, as illustrated in Fig. 10.1. Per output, only a single feature of the input data stream can be analyzed. Local units are hence feature extractors.

Commonly used computation units generate just a single output stream. An exception are quantum gates, which need to be unitary. Quantum gates have hence identical numbers of inputs and outputs, normally just two.

Perceptron Unit We start with the classical artificial neuron, denoted here "perceptron unit". Time is discrete, with individual in- and outputs corresponding to real numbers, often with restricted domains. Dimensionality reduction is achieved by taking the scalar product of the input vector $\mathbf{x}$ with a weight vector $\mathbf{w}$,

$$y = \sigma(a(x-b)), \qquad x = \mathbf{w}\cdot\mathbf{x}\,. \tag{10.1}$$

The output y is generated via a transfer function $\sigma(z)$. In machine learning, the threshold b is defined most of the time with an opposite sign. When the aim is to study a given set of weights $\mathbf{w}$, one keeps the gain a as a free parameter. If not, one can set it to unity, $a \to 1$, as done in machine learning. Geometrically, the isocline $x = b$,

$$\mathbf{w}\cdot\mathbf{x} = b\,, \tag{10.2}$$

corresponds to a plane in input space. Percetron units act hence as linear classifiers.

Planes divide space into two half-spaces, say S_A and S_B. As feature extractors, perceptron units output information about whether the input vector is part of either S_A or S_B.

Geometric Features Spaces When images or visual data are processed, input units may be associated with specific geometric locations $\mathbf{R}_i$, usually corresponding to pixels in a two-dimensional plane. After adaption, the weights w_i of a specific classifying neuron,

$$w_i(\mathbf{R}_i) \;\to\; w(\Delta\mathbf{R}_i), \qquad \Delta\mathbf{R}_i = \mathbf{R}_i - \mathbf{R}_y\,, \tag{10.3}$$

will be a function of the geometric location $\mathbf{R}_i$ of the afferent neuron, normally relative to the location $\mathbf{R}_y$ of the classifying neuron.

Geometric feature extraction is the prime task of the visual cortex. Examples are linear or center-surround contrasts and gratings, both in the black-white and in the color domain. Features can be highly non-linear geometric objects, classification is nevertheless linear in the space of input activities.

Transfer Functions The transfer function $\sigma(z)$ entering (10.1) comes in a range of varieties. Two classical variants are

$$\sigma_{\rm sig}(z) = \frac{1}{1+\mathrm{e}^{-z}}, \qquad \sigma_{\tanh}(z) = \tanh(z)\,, \tag{10.4}$$

where the first, the "sigmoidal", works in the range $\sigma_{\rm sig} \in [0,1]$. For the second we have $\sigma_{\tanh} \in [-1,1]$. Both transfer functions are monotonically

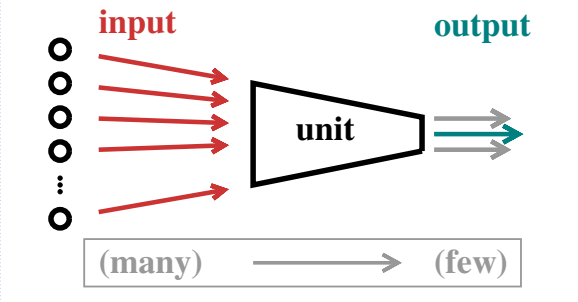

Fig. 10.1 Artificial neurons and other local computation units receive many inputs, with the subsequent processing reducing dimensionality. Only a few output streams are produced, in most cases only a single one. Computation units are hence equivalent to local feature extractors.

increasing and bounded, which makes the output a measure for the degree to which the input has been classified. The infinite-gain limit

$$\lim_{a\to\infty} \sigma_{\mathrm{sig}}(a(x-b)) \;=\; \begin{cases} 0.0 & x < b \\ 0.5 & x = b \\ 1.0 & x > b \end{cases} \tag{10.5}$$

leads to binary classification. The same holds for $\sigma_{\tanh}$.

ReLU Units The transfer function of "rectified linear units",

$$\sigma_{\mathrm{ReLU}}(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases} , \tag{10.6}$$

is piecewise linear.[1] Negative arguments are shunted. On the positive side, gradients will not vanish for large arguments, as they do for σ_{sig} and $\sigma_{\tanh}$, remaining finite for all $z > 0$. This helps to avoid the vanishing gradient problem when propagating errors backwards, a subject treated further below in Sect. 10.3.

Boltzmann Units So far, deterministic updating in discrete time has been used. Stochastic dynamics,[2]

$$\sigma_{\mathrm{Boltz}}(z) = \begin{cases} 1 & \text{with probability } \; p \\ 0 & \text{with probability } \; 1-p \end{cases} , \tag{10.7}$$

is however also an option. In the context of "Boltzmann machines" one relates the transition probability p to the difference between the two suitably defined energies, respectively of starting/final states. For

$$p = p(z) = \sigma_{\mathrm{sig}}(z) \tag{10.8}$$

[1] See Sect. ?? of Chap. ??, "??", for the general theory of piecewise linear dynamical systems.

[2] For the general theory of stochastic dynamical systems see Chap. ??, "??".

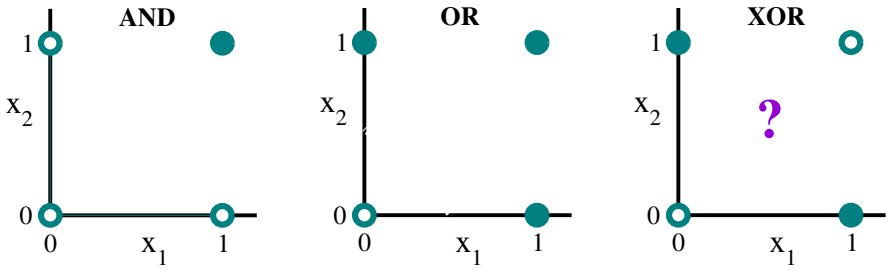

Fig. 10.2 The AND (left) and OR (middle) gates are examples of linearly separable logical gates (dashed diagonal line). The XOR gate is not linearly separable (right). Filled/open bullets denote logical true/false.

one recovers on average the sigmoidal transfer function.

10.1.1 Structured Units

The computation units discussed so far correspond to "point neurons", in the sense that the output depends exclusively on a single aggregate quantity, namely $x = \mathbf{w} \cdot \mathbf{x}$, respectively $z = a(x - b)$. A possible extension are detailed biological models that are characterized by a potentially large number of biophysical compartments. "Compartmental neurons" are of relevance in the neurosciences. Remaining on an abstract level, we discuss here several options for internally structured computation units.

Internal Memories In general there is a tradeoff between increasing either the number of units or the complexity of the constituent elements. The latter is a viable option in particular when qualitative new features are added. A prime example is the introduction of an internal variable acting as a local memory, viz a memory cell. An influential version in this respect is LSTM, a unit with "long short-term memory".

When an internal memory is present, the unit needs a mechanism deciding to which extend past activities influence the new output, together with the updated value of the memory cell. For a LSTM unit, the corresponding mechanisms are encoded as gates, respectively for input, output and forgetting. Gating parameters are adaptable, viz learned during training.

Gated Units Gating occurs when a certain data stream controls the processing of a second data stream. A basic example is the "gated linear unit", GLU,

$$\sigma_{\mathrm{GLU}}(\mathbf{x}) = (x_w - b_w)\, \sigma_{\mathrm{sig}}(x_v - b_v), \quad x_w = \mathbf{w} \cdot \mathbf{x}, \quad x_v = \mathbf{v} \cdot \mathbf{x}\,, \qquad (10.9)$$

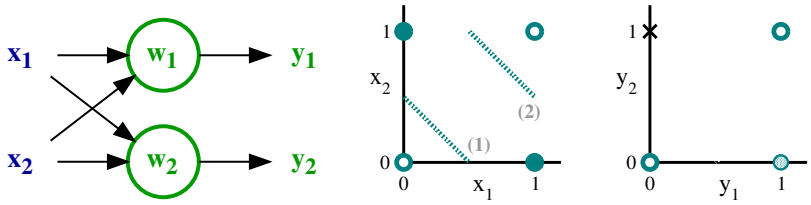

Fig. 10.3 Left: Receiving inputs x_1 and x_2, two hidden layer neurons produce outputs y_1 and y_2, the hidden-layer activity. Specific to the hidden layer are the weight vectors $\mathbf{w}_1$ and $\mathbf{w}_1$, together with the respective thresholds (not shown). **Middle:** In input space (x_1, x_2), the classifying planes (gray dashed lines) of the two hidden neurons; see (10.10). **Right:** The activity space of the hidden neurons, (y_1, y_2). Shown is the mapping (10.11) for binary neurons. The XOR configuration is now linearly separable (dashed diagonal line).

which contains two adaptable weight matrices and thresholds, $\mathbf{w}$, $\mathbf{v}$ and b_w, b_v. The output is a linear function of the input vector $\mathbf{x}$, which is however modulated by a sigmoidal, $\sigma_{\rm sig}(x_v - b_v) \in [0, 1]$. After training, $\mathbf{w}$ and $\mathbf{v}$ may encode distinct features. If this happens, the processing of a given feature may proceed only when the second feature is also present. This type of gating is hence equivalent to a coincidence detector for self-selected features. It holds as a corollary that a single GLU unit can express all logical gates, including the XOR gate.[3]

10.1.2 The XOR Problem

The term "artificial intelligence" was coined at a 1956 Dartmouth workshop. Hopes of rapidly achieving human-level AI were however crushed when it was realized 13 years later that linear classifiers,[4] like point neurons, may not perform non-linear classification tasks, such as representing an XOR gate; see Fig. 10.2. This was considered unfortunate, given that universal computation hinges on being able to express all logical gates, inclusively the XOR gate. The "XOR problem", as it was dubbed, is thought to have initiated an extended period of disillusionment with neuronal networks, the "AI winter".

Hidden Layers In machine learning jargon, whatever is not part of input or output is "hidden". A classical solution to the XOR problem is to add a hidden layer.

As an example consider that the weights and thresholds of the two hidden units are such that they define respectively the planes

[3] See exercise **??**.

[4] M. Minsky, S. Papert, "Perceptrons: An Introduction to Computational Geometry" (1969).

$$x_1 + x_2 = 0.5, \qquad x_1 + x_2 = 1.5\,, \tag{10.10}$$

as illustrated in Fig. 10.3. For binary neurons with $\sigma(z) = \theta(z)$, where $\theta(z)$ is the Heaviside step function, the mapping $(x_1, x_2) \to (y_1, y_2)$ reads

$$\begin{aligned} &(0,0) \to (0,0) \qquad (1,1) \to (1,1), \\ &(0,1) \to (1,0) \qquad (1,0) \to (1,0)\,, \end{aligned} \tag{10.11}$$

which implies that $(0,1)$ and $(1,0)$ are mapped to the same point in the activity space of the two hidden-layer units, (y_1, y_2). At this step, the XOR configuration becomes linearly separable.

Dimensional Embedding Classification becomes simpler when a given problem is embedded in an enlarged state space. We keep a single point neuron, extending however the input space via $(x_1, x_2) \to (x_1, x_2, x_3)$. As an example we set $x_3 = \pm\Delta$, together with

$$\begin{aligned} &(0,0) \to (0,0,+\Delta) \qquad (1,1) \to (1,1,+\Delta), \\ &(0,1) \to (1,0,-\Delta) \qquad (1,0) \to (1,0,-\Delta)\,. \end{aligned} \tag{10.12}$$

Consequently, the $x_3 = 0$ plane can be used for the classification of the XOR gate. Above example is constructed by hand. When using random embeddings, a common approach, larger embedding dimensions may be necessary. The brain makes extensive use of dimensional embedding,

Universal Function Approximators Solutions to the XOR problem abound. An option mentioned further above are gated units, see (10.9), which can do the job all alone. Neural networks are "Turing complete" when logical gates can be represented, acting as "universal function approximators". Given enough units, arbitrary complex functional dependencies can be modeled with increasing accuracy.

10.2 Recurrent Neural Networks

Point neurons connected via synaptic weights form a network of interconnected units.[5] Networks corresponding to "feedforward nets" are free of loops, "recurrent nets" not. In general, the "membrane potential" x_i of a point neuron is given by

$$x_i = e_i + \sum_j w_{ij} y_j, \qquad y_i = \tanh(a_i(x_i - b_i))\,, \tag{10.13}$$

[5] Generic network theory is developed in Chap. **??**, "**??**".

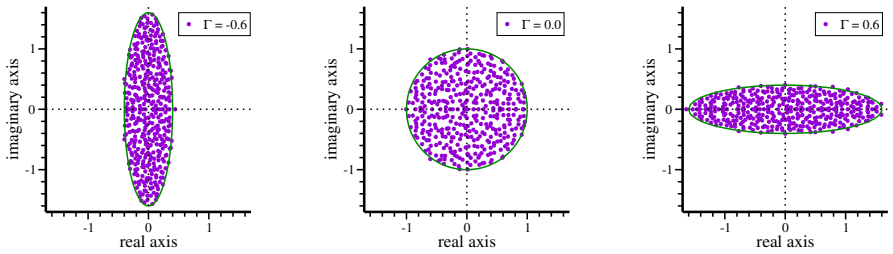

Fig. 10.4 The distribution of eigenvalues of an elliptic 400×400 matrix. The eigenvalues (filled dots) in the complex plane are shown for $\Gamma = -0.6/0.0/0.6$ (left/middle/right), as defined by (10.14). Included are the corresponding analytic results (10.15), which are valid in the thermodynamic limit (lines).

where e_i is an external driving and w_{ij} the entries of the synaptic weight matrix $\hat{w}$. A tanh-transfer function has been selected.

Lyapunov Exponents Recurrent nets feed back on themselves, which gives raise to a trailing memory. In the autonomous case, when $e_i \equiv 0$, the largest eigenvalue of $\hat{w}$, the largest "Lyapunov exponent" of the map defined by (10.13), determines if activities tend to increase or decrease in linear order.[6] This issue will be treated in detail in Sect. 10.2.2.

10.2.1 Random Matrix Theory

The statistics of the entries of a matrix can be used for estimating the corresponding spectrum of eigenvalues. When the entries are only weakly correlated, as defined below, random matrix theory applies, providing precise results. Often, matrix elements can be assumed to be independently distributed, in particular when the rank of the matrix is large, the typical case for both neuroscience and machine-learning applications.

Ensemble Average Random matrix theory deals with matrices with elements that are drawn from a given probability distribution, normally a Gaussian. Drawing all elements once generates a specific realization. "Ensemble average" denotes the average over realizations. In practice one is however interested in the spectrum of a specific matrix. For large matrices the ensemble average can be substituted by an average of all entries, the venue taken here.

Off-Diagonal Correlations In its basic form, random matrix theory can be applied only when the entries w_{ij} are statistically fully independent. It holds however also when the off-diagonal cross-correlation Γ,

[6] See Chap. **??**, "**??**".

$$\Gamma = \frac{\sum_{i,j} \left(w_{ij} - \mu_w\right)\left(w_{ji} - \mu_w\right)}{\sum_{i,j} \left(w_{ij} - \mu_w\right)^2}, \qquad \Gamma \in [-1, 1]\,, \tag{10.14}$$

is non-zero. For a $N \times N$ matrix, we defined the mean $\mu_w = \sum_{ij} w_{ij}/N^2$. The inequality $|\Gamma| \leq 1$ follows from $(a \pm b)^2 \geq 0$, or $a^2 + b^2 \geq \mp 2ab$. Three special cases are of interest.

- $\Gamma = \ \ 0$: Fully uncorrelated, complex eigenvalues.
- $\Gamma = \ \ 1$: Symmetric matrix with $w_{ij} = w_{ji}$; real spectrum.
- $\Gamma = -1$: Antisymmetric matrix with $w_{ij} = -w_{ji}$; imaginary eigenvalues.

One speaks of "elliptic matrices" when $\Gamma \neq 0$.

Elliptic Random Matrices We set $\mu_w = 0$, without loss of generality, and assume $\sigma_w = 1/\sqrt{N}$ for the standard deviation σ_w of the matrix elements. Random matrix theory states that the spectrum of the eigenvalues λ is uniformly distributed within an ellipse defined by

$$E_\Gamma = \left\{ \lambda = x + iy, \quad \frac{x^2}{(1+\Gamma)^2} + \frac{y^2}{(1-\Gamma)^2} \leq 1 \right\}. \tag{10.15}$$

The half widths are $1 \pm \Gamma$ respectively along the real and imaginary axis. All eigenvalues are real/imaginary for $\Gamma = \pm 1$. In Fig. 10.4 a comparison between numerical spectra and (10.15) is presented. The agreement would be perfect in the limit of infinite-large matrices.

Finite Mean For $\Gamma = 0$ the ellipse defined by (10.15) reduces to a circle and one recovers the "circular law of random matrix theory", namely that the eigenvalues are uniformly distributed within a circle. When the mean $\mu_w = \sum_{ij} w_{ij}/N^2$ is finite, an additional isolated eigenvalue λ_μ is generated. For $\Gamma = 0$ one can derive that

$$\lambda_\mu = \tilde{\mu}_w + \frac{\tilde{\sigma}_w^2}{\tilde{\mu}_w}, \qquad \mu_w = \frac{\tilde{\mu}_w}{N}, \qquad \sigma_w^2 = \frac{\tilde{\sigma}_w^2}{N}\,. \tag{10.16}$$

The additional contribution, $\tilde{\sigma}_w^2/\tilde{\mu}_w$, can be interpreted as an interaction between mean and continuum.

Spectral Radius When the eigenvalue spectrum is bounded, there exists a $R_w > 0$ such that

$$|\lambda_i| \leq R_w, \qquad \forall i \tag{10.17}$$

The "spectral radius" R_w determines the long-term evolution when applying the mapping in question repeatedly. For elliptic random matrices with zero mean $R_w = \tilde{\sigma}_w(1 + |\Gamma|)$.

Variance Control An interesting point is that the spectral radius R_w scales with $\tilde{\sigma}_w = \sigma_w\sqrt{N}$, which is directly proportional to $\sqrt{N}$ and to the standard deviation σ_w of the matrix elements. Standard learning algorithms, like Heb-

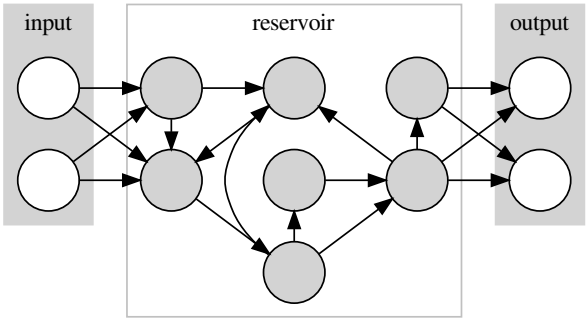

Fig. 10.5 Recurrent nets are the centerpiece of reservoir computing, mapping data input streams into a high-dimensional reservoir of non-linear features. Only the weights of linear output units are trained.

bian plasticity in biology or backpropagation in machine learning, treat each synaptic weight however as an independent adaptable parameter. The variance σ_w^2 resulting from the learning process is hence neither optimized nor controlled.

Reservoir computing Spectral radius regulation is central for "reservoir computing", as illustrated in Fig. 10.5. A recurrent net with otherwise fixed synaptic weights is used to transform the input into a multitude of random non-linear features. Only the weights of a subsequent linear output layer are adapted during training, typically for prediction tasks.

10.2.2 Criticality in Recurrent Networks

The brain is autonomously active, which implies that recurrent connections are functionally important. Does the level of internally produced activities impact the capability of the brain to process information? We have seen in Chap. **??**, "**??**", that critical dynamical systems exhibit non-trivial properties. Indeed, it can be shown that the long-range correlations characteristic of systems poised on a bifurcation point improve the processing of incoming data streams. This is captured by the "critical brain hypothesis". The quantitative impact of being close to criticality can be examined via a self-consistent theory for recurrent neural nets. We start with some preliminaries.

Self-Consistent Variance Theory We have seen that the variance of a random matrix determines its spectrum (10.15). It is hence clear that self-consistent approaches to recurrent nets will be on the level of variances.

Starting, we recall elementary relations for two statistically independent random variables z_1/z_2 with means μ_1/μ_2 and standard deviations σ_1/σ_2. The variances of the sum $z_1 + z_2$ and respectively of the product $z_1 z_2$ are given by

$$\sigma_{1+2}^2 = \sigma_1^2 + \sigma_2^2, \qquad \sigma_{1\cdot 2}^2 = \left(\sigma_1^2 + \mu_1^2\right)\left(\sigma_2^2 + \mu_2^2\right) - \mu_1^2\mu_2^2\,. \tag{10.18}$$

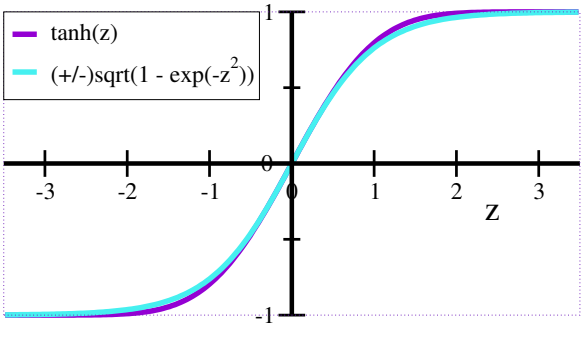

Fig. 10.6 Comparison between $\tanh(z)$ and the Gaussian transfer functions defined in (10.23).

The first relation is the standard sum rule for the variance,[7] the second relation is treated in exercise **??**.

Statistically Independent Units For a network with N units we define with

$$\sigma_w = \frac{\tilde{\sigma}_w}{\sqrt{N}}, \qquad \sigma_x, \qquad \sigma_y, \qquad \sigma_{\text{ext}}\,, \tag{10.19}$$

the standard deviations of respectively the synaptic weights, of the membrane potential x, of the neural activity y, and of the external input e, see (10.13). In reality, the respective distributions would be specific to the individual unit. The central approximation used here is that all units are statistically independent, and otherwise equivalent. It is then sufficient to work with (10.19).

Balanced Activities As a further simplification we consider networks with balanced activity and weight distributions, which implies vanishing means μ_w, μ_x, μ_y and μ_{ext}. The definition (10.13) of the membrane potential, $x_i = e_i + \sum_j w_{ij} y_j$, leads to

$$\sigma_x^2 = \sigma_{\text{ext}}^2 + N\sigma_w^2\sigma_y^2 = \sigma_{\text{ext}}^2 + \tilde{\sigma}_w^2\sigma_y^2\,, \tag{10.20}$$

where we used (10.18), (10.19) and that units receive N recurrent inputs. The variance of the neural activity is determined by

$$\sigma_y^2 + \mu_y^2 = \int dx \tanh^2(x-b)\, P(x)\,, \tag{10.21}$$

where $P(x)$ is the distribution of the membrane potential. We did set $a = 1$ and made use of $\langle (y-\mu_y)^2\rangle = \langle y^2\rangle - \mu_y^2$. For balanced activities the threshold vanishes, $b = 0$, together with $\mu_y = 0 = \mu_x$.

Central limit theorem The central limit theorem states that sums of large numbers of random variables converges to a normal distribution. We can hence assume that the probability distribution $P(x)$ of the membrane poten-

[7] More about random variables in Chap. **??**, "**??**".

tial entering (10.21) is normal distributed with mean μ_x and variance σ_x^2,

$$P(x) = \frac{1}{\sqrt{2\pi\sigma_x^2}}\, \mathrm{e}^{-(x-\mu_x)^2/(2\sigma_x^2)}, \qquad 1 = \int dx P(x)\,. \tag{10.22}$$

A last step is needed before the integral in (10.21) can be evaluated analytically.

Gaussian Transfer Function We define with

$$\sigma_{\text{Gauss}} = \pm\sqrt{\sigma^2_{\text{Gauss}}}, \qquad \sigma^2_{\text{Gauss}}(z) = 1 - \mathrm{e}^{-z^2} \tag{10.23}$$

a "Gaussian transfer function". The comparison between the tanh-transfer function and σ_{Gauss} presented in Fig. 10.6 shows that both transfer functions are functionally similar, with identical slopes at $z = 0$. One can therefore use one as an approximation for the other.

Variance of the Neural Activity We use the Gaussian transfer function for a balanced system, with b, μ_x, μ_y and μ_w vanishing, together with (10.22) in (10.21). The resulting expression,

$$\sigma_y^2 = \frac{1}{\sqrt{2\pi\sigma_x^2}} \int dx \left(1 - \mathrm{e}^{-x^2}\right) \mathrm{e}^{-x^2/(2\sigma_x^2)}\,, \tag{10.24}$$

leads via

$$1 - \sigma_y^2 = \frac{1}{\sqrt{2\pi\sigma_x^2}}\, \frac{\sigma_{\text{eff}}}{\sigma_{\text{eff}}} \int dx\, \mathrm{e}^{-x^2/(2\sigma^2_{\text{eff}})}, \qquad \frac{1}{\sigma^2_{\text{eff}}} = \frac{1 + 2\sigma_x^2}{\sigma_x^2}$$

to

$$1 - \sigma_y^2 = \frac{\sigma_{\text{eff}}}{\sigma_x} = \frac{1}{\sqrt{1 + 2\sigma_x^2}}, \qquad 2\sigma_x^2 = \frac{1}{(1-\sigma_y^2)^2} - 1\,. \tag{10.25}$$

Note that $y^2 < 1$ and hence $\sigma_y^2 < 1$.

Variance Self-Consistency For $\Gamma = 0$, the case considered here, one can substitute $\tilde{\sigma}_w$ by the spectral radius $R_w = \tilde{\sigma}_w$. Together, (10.20) and (10.25) lead to the self-consistency condition

$$2\big(\sigma^2_{\text{ext}} + R_w^2\sigma_y^2\big) = 2\sigma_x^2 = \frac{1}{(1-\sigma_y^2)^2} - 1\,,$$

or

$$2R_w^2\sigma_y^2\big(1-\sigma_y^2\big)^2 = 1 - \big(1 + 2\sigma^2_{\text{ext}}\big)\big(1-\sigma_y^2\big)^2\,, \tag{10.26}$$

which is quite remarkable. It states that the neural activity, in terms of the variance σ_y^2, is determined exclusively by the spectral radius R_w of the weight matrix, together with an eventually present external input, σ^2_{ext}.

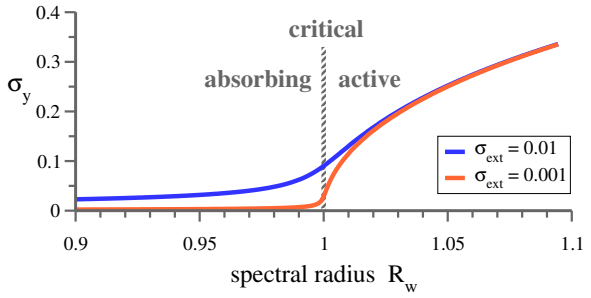

Fig. 10.7 The solution of the variance self-consistency condition (10.26). An absorbing phase transition occurs at $R_w = 1$ in the absence of external stimuli, viz when $\sigma_{\text{ext}} = 0$

Absorbing Phase Transition The system is autonomous when $\sigma_{\text{ext}} = 0$. In this case one obtains

$$2R_w^2\sigma_y^2 = 2\sigma_y^2 + \text{ higher powers}$$

when expanding both sides of (10.26) in powers of σ_y^2. Consequently, the system undergoes an "absorbing phase transition"[8] at $R_w = 1$.

- The activity dies out for $R_w < 1$.
- The activity steadily increases for $R_w > 1$, until limited by the non-linearity of the transfer function.

The state with vanishing neural activity is the final state of all orbits when $R_w < 1$, whatever the starting configuration. It is hence termed "absorbing". The active state is chaotic when performing simulations with a specific weight matrix. This is a consequence of $R_w > 1$, as discussed in Chap. **??**, "**??**".

Criticality vs. External Driving In Fig. 10.7 numerical solutions of (10.26) are presented. The phenomenology of a second-order phase transition is reproduced,[9] with the input variance σ_{ext}^2 acting as an external field.

Interestingly, Fig. 10.7 shows that the neural activity is substantial at criticality $R_w = 1$ even when external stimuli are comparatively weak. This result raises some doubts with regard to the critical brain hypothesis. Information processing will still benefit quantitatively from being close to criticality even when the variance σ_{ext}^2 of the data input stream is not negligible small. On a qualitative level, the working regime of the network does however not change as a function of the spectral radius in the presence of an external driving.

Feed-forward Networks Eq. (10.25) can be interpreted as a mapping of σ_x to σ_y across a non-linearity. Adding (10.20) one obtains a mapping from $\sigma_{y,n-1}$ to $\sigma_{y,n}$, when adding layer indices, which holds also across a feed-forward layer,

[8] Absorbing phase transitions are treated in depth in Sect. **??** of Chap. **??**, "**??**".

[9] See Sect. **??** of Chap. **??**, "**??**" for an introduction to the Landau theory of phase transitions. The connection is made in exercise **??**.

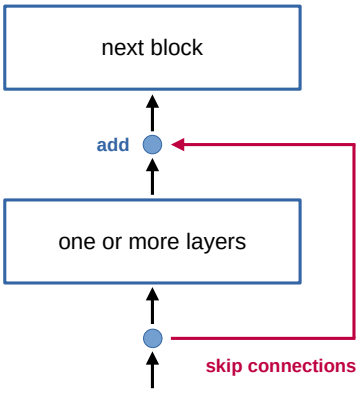

Fig. 10.8 Skip connections short-circuit the information processing by one or more layers. The input $\mathbf{x}_l$ is added to the output $\mathbf{y}_l$ of a given block, $\mathbf{x}_{l+1} = \mathbf{y}_l + \mathbf{x}_l$.

$$\sigma^2_{y,n} = 1 - \frac{1}{\sqrt{1+2\sigma^2_{x,n}}} = 1 - \frac{1}{\sqrt{1+2\big(\sigma^2_{\text{ext},n} + \tilde{\sigma}^2_w \sigma^2_{y,n-1}\big)}}\,, \tag{10.27}$$

where $\sigma^2_{\text{ext},n}$ is an additional input to layer n, if existing. Above expression is valid when all averages are zero, which would not be the case in a practical application. The generalization to non-zero μ_y, μ_x, μ_w is straightfoward, but a bit cumbersome, see exercise **??**.

10.3 Neural Differential Equations

Deep networks correspond to a mapping

$$\mathbf{f} = \mathbf{f}(\mathbf{x}, \boldsymbol{\vartheta})$$

of an input $\mathbf{x} = (x_1, x_2, ..)$ to an output $\mathbf{f} = (f_1, f_2, ..)$. The parameters $\boldsymbol{\vartheta} = (\vartheta_1, \vartheta_2, ..)$ are adaptable, viz determined during training.

Training Data Training data consists of a set of N_α pairs of inputs $\mathbf{x}^\alpha$ and target outputs $\mathbf{F}^\alpha$. The overall goal is to minimize a loss function L, e.g.

$$L = \sum_\alpha \big(\mathbf{F}^\alpha - \mathbf{f}^\alpha\big)^2, \qquad \mathbf{f}^\alpha = \mathbf{f}(\mathbf{x}^\alpha, \boldsymbol{\vartheta})\,, \tag{10.28}$$

which can be done by following the gradient, $\dot{\boldsymbol{\vartheta}} \sim -\nabla_{\boldsymbol{\vartheta}} L$. For layered systems, as the one illustrated in Fig. 10.3, the gradient is evaluated recursively.

Error Backpropagation As a basic deep learning architecture we consider a system made up of N_L identical layers,

$$\mathbf{y}_l = \mathbf{f}(\mathbf{x}_l, \boldsymbol{\vartheta}_l), \qquad \mathbf{x}_l = \mathbf{y}_{l-1}\,, \tag{10.29}$$

where $\boldsymbol{\vartheta}_l$ denotes the adaptable parameters of the lth layer, like thresholds and synaptic weights. In order to simplify the notation, we set $N_L = 3$. For a given training pair $(\mathbf{x}^\alpha, \mathbf{F}^\alpha)$, the gradient of the loss function with respect to the parameters $\boldsymbol{\vartheta}_3$ of the last layer is

$$\nabla_{\boldsymbol{\vartheta}_3} L = \nabla_{\boldsymbol{\vartheta}_3} \mathbf{f}(\mathbf{x}_3, \boldsymbol{\vartheta}_3) \cdot \mathbf{E}_3, \qquad \mathbf{E}_3 = (-2)\big(\mathbf{F}^\alpha - \mathbf{f}^\alpha\big)\,.$$

We used here the compressed notation $\nabla \mathbf{A} \cdot \mathbf{B} \equiv \sum_i (\nabla A_i) B_i$. With the chain rule, we have

$$\nabla_{\boldsymbol{\vartheta}_2} L = \nabla_{\boldsymbol{\vartheta}_2} \mathbf{f}(\mathbf{x}_2, \boldsymbol{\vartheta}_2) \cdot \mathbf{E}_2, \qquad \mathbf{E}_2 = \nabla_{\mathbf{x}_3} \mathbf{f}(\mathbf{x}_3, \boldsymbol{\vartheta}_3) \mathbf{E}_3 \tag{10.30}$$

for the gradient of the loss function with respect to $\boldsymbol{\vartheta}_2$. The error $\mathbf{E}_2$ is a linear function of $\mathbf{E}_3$, the error signal of the next layer. Errors are hence propagated from top to down, which can be continued recursively. This is denoted "backpropagation".

10.3.1 Residual Nets

Layers process their inputs typically via (10.1), which involves a non-linear transfer function $\sigma(\cdot)$. The gradient of $\sigma(\cdot)$ vanishes for large inputs when the transfer function is limited, as is it the case, e.g., for the sigmoidal. The magnitude of the error backpropagated via (10.30) decreases hence in magnitude layer by layer, a phenomenon denoted "vanishing gradient problem".

Skip Connections A popular workaround for the vanishing gradient problem are ReLU units, see (10.6), for which the gradient is constant for positive arguments. An alternative are "skip connections",

$$\mathbf{y}_l = \mathbf{x}_l + \mathbf{f}(\mathbf{x}_l, \boldsymbol{\vartheta}_l), \qquad \mathbf{x}_l = \mathbf{y}_{l-1}\,, \tag{10.31}$$

which corresponds to add the identity to the forward pass, see Fig. 10.8. Learning by adapting the parameters $\boldsymbol{\vartheta}_l$ has now the task to generate the correct 'residual' $\mathbf{y}_l - \mathbf{x}_l$, hence the name ResNet, "residual net".

Neural Differential Equation It is tempting, with $\mathbf{x}_{l+1} = \mathbf{y}_l$, to rewrite (10.31) as

$$\mathbf{x}_{l+1} - \mathbf{x}_l = \mathbf{f}(\mathbf{x}_l, \boldsymbol{\vartheta}_l), \qquad \frac{d\mathbf{x}}{dt} \approx \mathbf{f}(\mathbf{x}, \boldsymbol{\vartheta})\,, \tag{10.32}$$

where the layer index l was substituted by a pro forma time t, using $\mathbf{x}_l \to \mathbf{x}(t)$ and $\boldsymbol{\vartheta}_l \to \boldsymbol{\vartheta}(t)$. Layer time $t \in [0, T_L]$ is continuous, whereas the layer index $l = 1, 2 \ldots$ was discrete. The "neural differential equation" (10.32) corresponds hence to a continuous layer representation of a residual network.

Continuous Backpropagation As such, the neural differential equation (10.32) contains an unknown function, $\boldsymbol{\vartheta}(t)$, which specifies the value of the parameters for any layer time t. With skip connections, the recursive relation (10.30) for the error becomes

$$\mathbf{E}_{l-1} = \big(\nabla_{\mathbf{x}_l}\mathbf{f}(\mathbf{x}_l, \boldsymbol{\vartheta}_l) + \mathbf{1}\big)\mathbf{E}_l, \qquad \frac{d\mathbf{E}}{dt} \approx -\nabla_{\mathbf{x}}\mathbf{f}(\mathbf{x}, \boldsymbol{\vartheta}) \cdot \mathbf{E}\,. \tag{10.33}$$

Backpropagated errors tend to increase, given that $\nabla_{\mathbf{x}}\mathbf{f}$ is normally positive. In the continuous formulation, the error function $\mathbf{E} = \mathbf{E}(t)$ is obtained by integrating back in layer time. When adapting parameters via gradient minimization, $\Delta\boldsymbol{\vartheta} = -\eta\nabla_{\boldsymbol{\vartheta}}L$, one can use (10.30) to obtain

$$\Delta\boldsymbol{\vartheta} = -\eta\nabla_{\boldsymbol{\vartheta}}\mathbf{f}(\mathbf{x}, \boldsymbol{\vartheta}) \cdot \mathbf{E}\,, \tag{10.34}$$

which leads in turn to two interdependent differential equations with respect to layer time,

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \boldsymbol{\vartheta}), \qquad \dot{\mathbf{E}} = -\nabla_{\mathbf{x}}\mathbf{f}(\mathbf{x}, \boldsymbol{\vartheta}) \cdot \mathbf{E}\,. \tag{10.35}$$

In the adiabatic approximation, when the learning rate η is small, one takes $\boldsymbol{\vartheta}$ as constant when integrating $\mathbf{x}$ up from $\mathbf{x}(0) = \mathbf{x}^\alpha$. Then the error function $\mathbf{E}$ is evaluated downwards from $\mathbf{E}(T_L) = (-2)(\mathbf{F}^\alpha - \mathbf{x}(T_L))$. In the end, the parameters are updated.

It may seem a bit unusual, at first sight, to work with continuously stacked layers, which is however a viable option. Which one to use, continuous or discrete layers, is in the end a question of performance.

10.4 Gaussian Processes

In Sect. 10.2.2 we made extensive use of normal distributed random variables. Gaussian processes also involve normal distributed objects, however in the form of functions, and not variables. Starting with the basics, we recapitulate "multivariate" normal distributions, viz the case that more than one degree of freedom is relevant.

10.4.1 Multivariate Gaussians

The probability density for a set of N independent normal distributed random variables,

$$P_0(\mathbf{x}) = \prod_{i=1}^{N} \frac{\mathrm{e}^{-(x_i-\mu_i)^2/(2\sigma_i^2)}}{\sqrt{2\pi\sigma_i^2}} \tag{10.36}$$

is a special case of

$$P(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^N \sqrt{\det(\Sigma)}} \, \mathrm{e}^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})} \tag{10.37}$$

where Σ is a $N \times N$ symmetric matrix denoted "covariance matrix". The vector of means is $\boldsymbol{\mu}$, with Σ^{-1} denoting the inverse, $\Sigma\Sigma^{-1} = \Sigma^{-1}\Sigma = \mathbf{1}$.

Diagonalization We assume that the covariance matrix is positive definite, with eigenvalues σ_i^2. There exists hence an orthogonal transformation U such that

$$U^T \Sigma U = \begin{pmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_N^2 \end{pmatrix} \equiv \Sigma_0, \qquad U^{-1} = U^T \,, \tag{10.38}$$

with $\det(\Sigma_0) = \prod_i \sigma_i^2 = \det(\Sigma)$. The columns of U are the eigenvectors of Σ. We recall that the eigenvectors $\mathbf{e}_i$ of a non-singular matrix are also eigenvectors of the inverse,

$$\Sigma \, \mathbf{e}_i = \sigma_i^2 \, \mathbf{e}_i \qquad \Leftrightarrow \qquad \frac{1}{\sigma_i^2} \, \mathbf{e}_i = \Sigma^{-1} \mathbf{e}_i \,,$$

with the precondition that none of the eigenvalues vanish, viz that Σ is non-singular. Equivalently, the relation

$$U^T \Sigma^{-1} U = \begin{pmatrix} \frac{1}{\sigma_1^2} & & 0 \\ & \ddots & \\ 0 & & \frac{1}{\sigma_N^2} \end{pmatrix} \tag{10.39}$$

holds. For a quick check one multiplies both sides of (10.38) and (10.39).

Normalization The properties of a multivariate Gaussian distribution can be evaluated using U for transforming the variables,

$$\mathbf{x} = U\mathbf{y}, \qquad \det(U) = 1 \,. \tag{10.40}$$

The normalization of $P(x)$ follows directly,

$$\int d\mathbf{x} \, P(\mathbf{x}) = \int d\mathbf{y} \prod_i \frac{1}{\sqrt{2\pi\sigma_i^2}} \, \mathrm{e}^{-y_i/(2\sigma_i^2)} = 1 \,,$$

where we eliminated $\boldsymbol{\mu}$ via an appropriate shift of the origin. It also follows that (10.36) and (10.37) are identical, modulo an orthogonal variable transformation.

Covariance Matrix We have to show that Σ honors its name, namely that the general definition of the covariance matrix,

$$\big\langle (x_i - \mu_i)(x_j - \mu_j) \big\rangle = \int d\mathbf{x}\, (x_i - \mu_i)(x_j - \mu_j)\, P(\mathbf{x})\,, \tag{10.41}$$

coincides with Σ_{ij}. For statistically independent variables, distributed according to P_0, see (10.36), the covariance matrix is diagonal,

$$\big\langle (x_i - \mu_i)(x_j - \mu_j) \big\rangle_0 = \Sigma_0\,,$$

with Σ_0 defined in (10.38). Applying the orthogonal transformation U^T to both sides maps $P_0(\mathbf{x})$ to $P(\mathbf{x})$, and Σ_0 to Σ, the preposition we wanted to prove.

Regression A central machine learning task is regression, namely to extrapolate or interpolate training data. The goal is to make an educated guess about something not yet seen. This can be quantified in particular when assuming that values are drawn from multivariate Gaussians. We enlarge the set of variables,

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}, \qquad \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \qquad \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}, \tag{10.42}$$

where the Σ_{ij} are matrices, not matrix elements. The probability of observing $\mathbf{x}_1$, the prediction we want to make after having observed $\mathbf{x}_2$, is denoted as

$$P(\mathbf{x}_1|\mathbf{x}_2)\,,$$

reading, "the probability of observing $\mathbf{x}_1$ given $\mathbf{x}_2$".

Bayes Theorem Per definition, the relation

$$P(\mathbf{x}_1|\mathbf{x}_2)P(\mathbf{x}_2) = P(\mathbf{x}_1, \mathbf{x}_2), \qquad P(\mathbf{x}_1|\mathbf{x}_2) = \frac{P(\mathbf{x}_1, \mathbf{x}_2)}{P(\mathbf{x}_2)}\,, \tag{10.43}$$

holds. It is also denoted "Bayes theorem", in particular together with the recursive substitution $P(\mathbf{x}_1, \mathbf{x}_2) = P(\mathbf{x}_2|\mathbf{x}_1)P(\mathbf{x}_1)$. Given the relations,

$$1 = \int d\mathbf{x}_1 P(\mathbf{x}_1|\mathbf{x}_2), \qquad P(\mathbf{x}_1) = \int d\mathbf{x}_2 P(\mathbf{x}_1, \mathbf{x}_2)\,,$$

one has that $P(\mathbf{x}_1|\mathbf{x}_2) \geq 0$ is a properly normalized probability distribution, for any $\mathbf{x}_2$, denoted "posterior". Idem for the "marginal", $P(\mathbf{x}_1)$. Posterior distributions are the object of desire when the information gained from an observation is to be expressed quantitatively, a process denoted "inference".[10]

Variational Inference For practical calculations, the distributions involved in Bayesian inference need to be parameterized. One speaks of "variational inferences" when a certain functional form is stipulated together with appro-

[10] More about Bayesian statistics in Sect. **??** of Chap. **??**, "**??**".

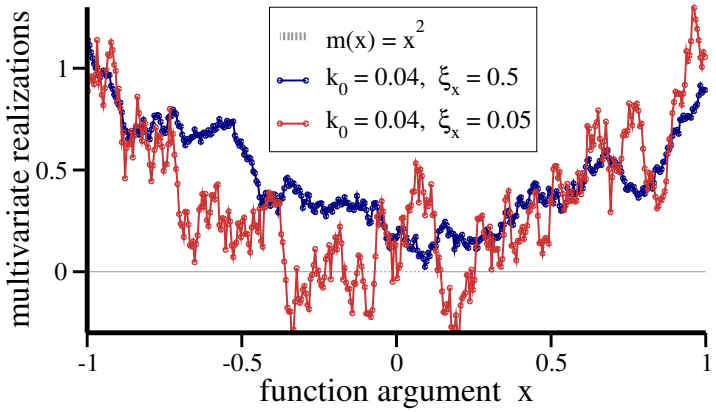

Fig. 10.9 The Gaussian process defined by (10.48), with a diagonal variance $k_0 = 0.04$. Shown is the mean function $m(x) = x^2$ (shaded line) and two realizations of the corresponding multivariate Gaussian distribution, respectively for a correlation length $\xi_x = 0.5/0.05$ (blue/red). The function estimate is smoother for longer-ranged correlations between the $N = 400$ arguments.

priate variational parameters. The approach is "variational" when the parameters are determined by minimizing an information-theoretical objective function.

Multivariate Gaussians are prime candidates for variational inference. One can show that the posterior $P(\mathbf{x}_1|\mathbf{x}_2)$ is also a multivariate Gaussian, with mean vector and covariance matrix, $\boldsymbol{\mu}_{1|2}$ and $\Sigma_{1|2}$, given by

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \Sigma_{12}\,\Sigma_{22}^{-1}(\mathbf{x}_2 - \boldsymbol{\mu}_2), \qquad \Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\,\Sigma_{22}^{-1}\Sigma_{21}\ . \quad (10.44)$$

The derivation involves skilled manipulations of Gaussian distributions. A motivation using neural tangent kernels is given in Sect. 10.4.3 One can generalize (10.44) to the case that observations are noisy and not accurate.

10.4.2 Correlated Stochastic Functions

In machine learning, a key objective is to generate increasingly accurate function approximations. Say we have a first guess, denoted "prior", which we want to refine during training. Being a guess, the prior cannot be accurately defined. Therefore, an algorithm for modeling stochastically broadened functional dependencies is needed. That's what stochastic processes are about.

Smooth Stochastic Functions We denote with f the stochastic function to be modeled. For every argument x from the support, the function is stochastically distributed around a mean value, $m(x)$, e.g. $m(x) = x^2$. The

notation

$$f \sim G_p(m, k), \qquad k = k(x, x')\,, \tag{10.45}$$

defines a Gaussian process for f. The covariance matrix $k(x, x')$ has two tasks.

- $k(x, x)$ is the variance of the probability distribution $f(x)$, for a given argument x.
- $k(x, x')$ makes sure that strong discontinuities are rare, when drawing function values for $x \neq x'$.

Estimates are independent for each argument x when the covariance matrix is diagonal.

Discrete Supports In practice, function estimates are evaluated on a discrete set of arguments, x_i, as in the example presented in Fig. 10.9. On a given set of N arguments, $\mathbf{x} = (x_1, \ldots, x_N)$, a multivariate Gaussian distribution,

$$\mu_i = m(x_i), \qquad \Sigma_{i,j} = k(x_i, x_j)\,, \tag{10.46}$$

is recovered, where $i, j \in [1, N]$. For every argument x_i, the function values f_i are distributed according to (10.37),

$$P(\mathbf{f}) = \frac{1}{\sqrt{2\pi}^N \sqrt{\det(\Sigma)}}\, \mathrm{e}^{-\frac{1}{2}(\mathbf{f}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{f}-\boldsymbol{\mu})} \tag{10.47}$$

where $\mathbf{f} = (f_1, \ldots, f_N)$ and with $\boldsymbol{\mu}$ and Σ given by (10.46). In Fig. 10.9 we illustrate a quadratic mean function,

$$m(x) = x^2, \qquad k(x, x') = k_0\, \mathrm{e}^{-|x-x'|/\xi_x}\,, \tag{10.48}$$

together with an exponentially decaying covariance matrix. The stochastic function approximation becomes smoother with increasing correlation length ξ_x.

Inference The expressions (10.44) for the mean and the covariance matrix of the posterior hold in analogy for Gaussian processes. We enlarge the number of arguments and function values as

$$\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2), \qquad \mathbf{f} = (\mathbf{f}_1, \mathbf{f}_2)\,,$$

where $\mathbf{x}_1$ is the N-dimensional vector of arguments covering the entire support. The data vector $\mathbf{x}_2$, of length $N_d < N$, includes all arguments for which training data is available, with corresponding function values $\mathbf{f}_2$. Inference consists of predicting $\mathbf{f}_1$.

In analogy to (10.44), the parameters of the posterior $P(\mathbf{f}_1|\mathbf{f}_2)$ are

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \Sigma_{12}\, \Sigma_{22}^{-1} (\mathbf{f}_2 - \boldsymbol{\mu}_2), \qquad \Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\, \Sigma_{22}^{-1} \Sigma_{21}\,. \tag{10.49}$$

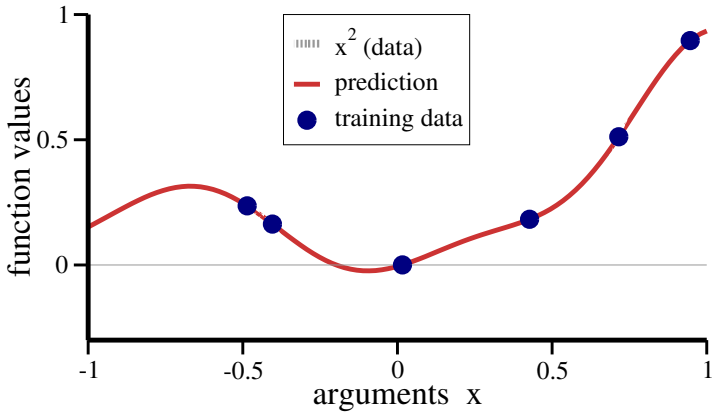

Fig. 10.10 Predictions (red line) inferred via (10.44) from a randomly drawn set of data points (blue dots). The covariance matrix is $k(x, x') = \exp(-(x - x')^2/0.2)$. The function values for the training data (blue bullets) follow $f_2(x) = x^2$. The mean function $m(x) = 0$ vanishes, in contrast to Fig. 10.9. Also included are 2σ confidence estimators (checkerborad red lines).

On a formal level, there is a difference regarding the term $\mathbf{f}_2 - \boldsymbol{\mu}_2$, as we are describing the distribution of function values $f_i = m(x_i)$, and not of the arguments.

Radial Basis Functions Correlations decay exponentially for non-critical processes, the reason we used an exponentially decaying covariance matrix in (10.48). Quadratic exponents,

$$m(x) \to 0, \qquad k(x, x') = k_0 \, \mathrm{e}^{-(x-x')^2/(2\sigma_k^2)} , \tag{10.50}$$

are however more covenient for frameworks based on multivariate Gaussians. Covariance matrices of this type are denoted "radial basis functions". Also included in (10.50) is a standard selection, namely a vanishing mean function $m(x)$. An alternative would be the average of the training data. When inference is performed, the mean of the posterior will be updated via (10.49), viz based on the training data. A non-trivial prior may bias this process.

In Fig. 10.10 an example of inference with Gaussian processes is presented. The variance $[\sigma_{1|2}(x)]^2$ of the prediction is given by $\Sigma_{1|2}(x, x)$, which has been included in terms of a confidence interval of two standard deviations. It is evident that inference with Gaussian processes is much better at interpolating data than extrapolating. The latter because predictions revert to the prior away from the data, here to $m(x) = 0$. The estimated confidence intervals depend sensibly on the positioning of the training data.

Intra-Domain Inference As a matter of principle, inference can be performed between different domains. Out-of domain inference requires however

knowledge about semantic cross-correlations, which cannot be modeled with radial basis functions, or another generic ansatz for the covariance matrix.

Here we work exclusively with intra-domain inference, such as predicting a function value when observing another, as in Fig. 10.10. The corollary is that all components of the covariance matrix have identical dependencies on their arguments.

Intra-domain inference includes the case of predicting the value for an observed position, viz that $\mathbf{x}_1 \to \mathbf{x}_2$. When this happens, one would expect that the prediction concurs with the observation, as it happens for the example shown in Fig. 10.10. This is easily confirmed analytically when both $\mathbf{x}_1$ and $\mathbf{x}_2$ are scalars, which implies $\Sigma_{11} = \Sigma_{12} = \Sigma_{21} = \Sigma_{22} \equiv k(0)$ when $x_1 = x_2$. The predicted mean is then

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\, \Sigma_{22}^{-1}(x_2 - \mu_2) \;\to\; x_2\,,$$

given that $\mu_1 = m(x_1)$ coincides with $\mu_2 = m(x_2)$ when $x_1 \to x_2$. The inferred mean function value, $\mu_{1|2}$, coincides hence with the training value x_2. This will not be the case when the data is noisy, which is described by adding $\sigma_D^2 \delta_{ij}$ to the covariance matrix Σ_{ij} in (10.46).

10.4.3 Machine Learning with Neural Tangent Kernels

Deep learning architectures typically involve large numbers of computational units. It is hence natural to assume that Gaussian distributions abound. This is indeed the case.

Infinite Layer Limit A basic setup is that of a single layer neural network,

$$f = \sum_i v_i y_i, \qquad y_i = \sigma(x_i - b_i), \qquad x_i = \sum_j w_{ij} x_j^\alpha\,, \tag{10.51}$$

where f is a classifying unit, the function to be computed, and $\mathbf{y} = (y_1, y_2, ..)$ the activity of the hidden layer. The training samples $\mathbf{x}^\alpha = (x_1^\alpha, x_2^\alpha, ..)$ are specific to the problem at hand. All internal parameters are taken to be random but frozen, viz corresponding to a specific realization.

The components of the training vectors, like x_i^α and x_j^α, are not necessarily uncorrelated, which implies that the membrane potentials x_i are independent random variables, but possibly not normal distributed. The same holds for the y_i. As a sum of independent distributions, f becomes however Gaussian for large hidden layers. This line of arguments, denoted the "infinite layer limit", can be extended to deep architectures.

Learning Correlations Learning is all about extracting information from data samples. Inputs can be differentiated, viz classified, only when differences can be quantified in terms of correlations. In leading order this regards

two-point correlations. Given that the output is Gaussian distributed in the infinite-layer limit, the output f of a deep learning architecture can be modeled in leading order by a Gaussian process over input space,

$$f \sim G_p(m,k), \qquad m = m(\mathbf{x}), \qquad k = k(\mathbf{x},\mathbf{x}')\,, \tag{10.52}$$

compare (10.45). Mean function and covariance matrix are averages over random sets $\boldsymbol{\vartheta}$ of parameters,

$$\begin{aligned} m(\mathbf{x}) &= \big\langle f(\mathbf{x};\boldsymbol{\vartheta})\big\rangle_{\boldsymbol{\vartheta}} \\ k(\mathbf{x},\mathbf{x}') &= \big\langle (f(\mathbf{x};\boldsymbol{\vartheta}) - m(\mathbf{x})(f(\mathbf{x}';\boldsymbol{\vartheta}) - m(\mathbf{x}')\big\rangle_{\boldsymbol{\vartheta}}\,. \end{aligned} \tag{10.53}$$

The output of the network is Gaussian because the parameters $\boldsymbol{\vartheta}$ act as random variables, hence the average over $\boldsymbol{\vartheta}$. The functional dependencies of the covariance matrix were central for the application of Gaussian processes discussed before. This is however not a relevant point for the "neural network gaussian process" defined by (10.53), which serves mainly as a conceptual background.

Supervised Learning During training, the output of the network,

$$f^\alpha = f(\mathbf{x}^\alpha;\boldsymbol{\vartheta})$$

is compared to the target F^α associated with the specific training sample $\mathbf{x}^\alpha$. The set of network parameters $\boldsymbol{\vartheta}$ are adapted by minimizing the loss function $L = \sum_\alpha (F^\alpha - f^\alpha)^2$ via gradient descent,

$$\frac{d}{dt}\,\boldsymbol{\vartheta} = \eta \sum_\alpha (\nabla_{\boldsymbol{\vartheta}} f^\alpha)\left(F^\alpha - f^\alpha\right), \tag{10.54}$$

with η being the learning rate. The right-hand-side includes the contribution of all N_α training samples. Training is "supervised" because the target F^α is explicitly provided. For a general input $\mathbf{x}$, the output $f = f(\mathbf{x};\boldsymbol{\vartheta})$ evolves as

$$\begin{aligned} \frac{d}{dt} f = \left(\nabla_{\boldsymbol{\vartheta}} f\right)\cdot\left(\frac{d}{dt}\boldsymbol{\vartheta}\right) \;&\equiv\; \eta \sum_\alpha \Theta(\mathbf{x},\mathbf{x}^\alpha)\big(F^\alpha - f^\alpha\big) \\ \Theta(\mathbf{x},\mathbf{x}^\alpha) &= \nabla_{\boldsymbol{\vartheta}} f(\mathbf{x};\boldsymbol{\vartheta})\cdot\nabla_{\boldsymbol{\vartheta}} f(\mathbf{x}^\alpha;\boldsymbol{\vartheta}) \end{aligned} \tag{10.55}$$

when using the gradient update rule (10.54). So far we did not make use of the infinite layer limit. Eqs. (10.54) and (10.55) are valid for any least-square minimization via gradient descent.

Tangent Kernels As a function of its arguments, $\mathbf{x}^\alpha$ and $\mathbf{x}^\beta$, the "neural tangent kernel" $\Theta(\mathbf{x}^\alpha,\mathbf{x}^\beta)$ is a symmetric matrix. With the matrix elements

being given by a scalar product, the neural tangent kernel is also positive definite.[11] For large networks the tangent kernel is

– constant during training and
– independent of the initial distribution of network parameters.

One says that $\Theta(\mathbf{x}^\alpha, \mathbf{x}^\beta)$ is "deterministic". We do not delve into the proof of these two prepositions, which involve inductive argumentation tailored to the architecture at hand. Systems become increasingly over-parametrized when network sizes diverge, which leads to the emergence of an exponentially large number of optimal parameter configurations. The probability for the starting parameter configuration to be close to an optimum will therefore increase steadily with layer width. Individual parameters need to change just a little when their number diverges.

Training Space Given that the tangent kernels are constant during training, one can solve the time evolution of $f = f(\mathbf{x}; \boldsymbol{\vartheta})$ explicitly via (10.55). First one needs to determine the time-dependency of the $f^\alpha = f(\mathbf{x}^\alpha; \boldsymbol{\vartheta})$, which is done in the space of training data,

$$\mathbf{F} = (F^1, \ldots, F^{N_\alpha}), \qquad \mathbf{f}_2 = (f^1, \ldots, f^{N_\alpha}),$$

where we used the subscript '2' to indicate training inputs, in equivalence to (10.49). Specifying (10.55) to the training space yields

$$\frac{d}{dt} f^\beta = \eta \sum_\alpha \Theta(\mathbf{x}^\beta, \mathbf{x}^\alpha)\big(F^\alpha - f^\alpha\big), \qquad \frac{d}{dt}\mathbf{f}_2 = \eta\, \Theta_{22}\big(\mathbf{F} - \mathbf{f}_2\big), \tag{10.56}$$

which constitutes a closed set of equations for N_α dynamical variables, $\mathbf{f}_2$. The projection of the tangent kernel to the training space, Θ_{22}, is a symmetric $N_\alpha \times N_\alpha$ matrix. The solution,

$$\mathbf{f}_2(t) = \mathbf{e}^{-\eta\,\Theta_{22} t}\, \mathbf{f}_2(0) + \big(\mathbf{1} - \mathbf{e}^{-\eta\,\Theta_{22} t}\big)\, \mathbf{F}\,, \tag{10.57}$$

holds when Θ_{22} is constant. One verifies (10.57) by direct differentiation. The constant of integration has been selected such that $\mathbf{f}_2(t=0) = \mathbf{f}_2(0)$. Perfect learning, $\mathbf{f}_2(t) \to \mathbf{F}$, is recovered in the limit $t \to \infty$.

Tangent Kernel Inference We define with $\mathbf{f}_1$ the vector of network outputs to be predicted and use (10.57) to rewrite (10.55) as

$$\frac{d}{dt}\mathbf{f}_1 = \eta\, \Theta_{12}\big(\mathbf{F} - \mathbf{f}_2\big) = \eta\, \Theta_{12}\, \mathbf{e}^{-\eta \Theta_{22} t}\big(\mathbf{F} - \mathbf{f}_2(0)\big)\,,$$

where the first and second arguments of Θ_{12} are respectively generic inputs and training data. Starting from $\mathbf{f}_1(0)$ and $\mathbf{f}_2(0)$, direct integration yields

[11] See exercise **??**.

$$\mathbf{f}_1(t) = \mathbf{f}_1(0) + \Theta_{12}\,\Theta_{22}^{-1}\left(\mathbf{1} - \mathbf{e}^{-\eta\Theta_{22}t}\right)\left(\mathbf{F} - \mathbf{f}_2(0)\right), \tag{10.58}$$

which is a quite remarkable result.

- The output of the network is fully specified by (10.58), including the entire training procedure.
- When training is complete, viz for $t \to \infty$, the predictions obtained using the infinite layer limit coincide formally with the corresponding expression for Bayesian inference, Eq. (10.49).

As for Bayesian inference, one has that $\mathbf{f}_1 \to \mathbf{F}$ when predictions are made for training data. This is seen easily for the case that $\mathbf{f}_1$ spans the entire training space, which leads to $\Theta_{12} \to \Theta_{22}$ and $\mathbf{f}_1(0) \to \mathbf{f}_2(0)$.

Note, that the formal equivalence of (10.58) and (10.49) does not imply that the tangent kernel $\Theta(\mathbf{x}, \mathbf{x}')$ and the covariance matrix $k(\mathbf{x}, \mathbf{x}')$ are identical. They are not. The covariance matrix is defined via (10.41) independently in terms of two-point correlations of the output $f = f(\mathbf{x}, \boldsymbol{\vartheta})$, in contrast to the definition of the neural tangent kernel, as given by (10.55). It is however possible, though rather cumbersome, to express $k(\mathbf{x}, \mathbf{x}')$ in terms of $\Theta(\mathbf{x}, \mathbf{x}')$.

Confidence Intervals The tangent kernel approach generates precisely defined predictions for inputs $\mathbf{x}$ not seen during training. Given that tangent kernels are deterministic in the infinite layer limit, this result may seem at odds with the statement that the output is Gaussian distributed, which holds for the same limit. Indeed, there is no contradiction.

The tangent kernel predictions (10.58) depend on $\mathbf{f}_1(0)$ and $\mathbf{f}_2(0)$, namely on the outputs generated before training did start, which are determined in turn by the starting parameter configuration. Given that $\boldsymbol{\vartheta}$ is drawn randomly, confidence intervals are obtained by sampling (10.58) with respect to $\boldsymbol{\vartheta}$, viz by sampling $\mathbf{f}_1(0)$ and $\mathbf{f}_2(0)$ with respect to the initial set of parameters.

The Linear Paradox Training is "lazy" when parameters hardly change, as in the infinite layer limit. The first order Taylor expansion,

$$f(\mathbf{x}, \boldsymbol{\vartheta}) \approx f(\mathbf{x}, \boldsymbol{\vartheta}_0) + \big(\nabla_{\boldsymbol{\vartheta}} f(\mathbf{x}, \boldsymbol{\vartheta}_0)\big)\Delta\boldsymbol{\vartheta}, \qquad \Delta\boldsymbol{\vartheta} = \boldsymbol{\vartheta} - \boldsymbol{\vartheta}_0\,, \tag{10.59}$$

becomes then exact. The "linear model" (10.59) is a bit paradoxical. It is still non-linear in $\mathbf{x}$, but linear in parameter space. This is possible as one works in a regime of a diverging number of parameters, one is hence dealing with overfitting.

10.5 Attention Induced Information Routing

Classical deep learning is about information processing. When data sets become complex, as for language processing, it may become important to focus processing resources to subsets of the input stream, the very task of

"attention". As such, attention has long been studied both in the neurosciences and in psychology. The first self-contained computational model was however developed from a machine learning perspective.

Top-Down Attention Signals Classically, attention is mediated in the neurosciences by a top-down signal. An example would be the task to concentrate on red objects in the visual stream. This can be done f.i. by gain control, viz by modulating the gain a of the neural transfer mapping $\sigma(a(x-b))$, see (10.1). The gain of neurons processing red/non-red objects would be increased/decreased.

Another example for neural filtering is biased competition, which involves mutually inhibiting clusters of neurons. Clusters encoding red objects will dominate the competition when receiving an additional boost, e.g. in the form of additional top-down inputs.

The Homunculus Problem From a computational perspective, neuroscience models like gain control or biased competition have two core shortcomings.

Generally, attention signals are taken to be generated by some unknown process in higher brain areas, which corresponds on a functional level to presume the existence of an identity, an "homunculus", deciding whether or not to pay attention to a certain type of objects or data. This is a problem, since theories not including the generation of attention signals are incomplete. Related is the question of how higher brain areas know about lower-area specializations, f.i. with regard to the color of objects within the input stream.

Training Attention The homunculus problem is related conceptional to an assumption often made implicitly in the neurosciences, as well as in psychology. The view is that advanced cognitive systems first develop an understanding of the world, via data acquisition and world modeling, with attention being an add-on. In this view, a cognitive system can pay attention to red objects only once the system understands concepts like 'red' and 'object'.

The approach taken by machine learning is radically different. Here, all parts of the system are jointly trained. This includes attention, when incorporated, as in transformer architectures. The view is that attention can develop in a meaningful manner only together with representation extractions and world modeling.

10.5.1 Transformer

Functionally, attention corresponds to information routing. Only part of the incoming data is processed further. We have seen that gated units, as described by (10.9), are capable to regulate the processing on a local level.

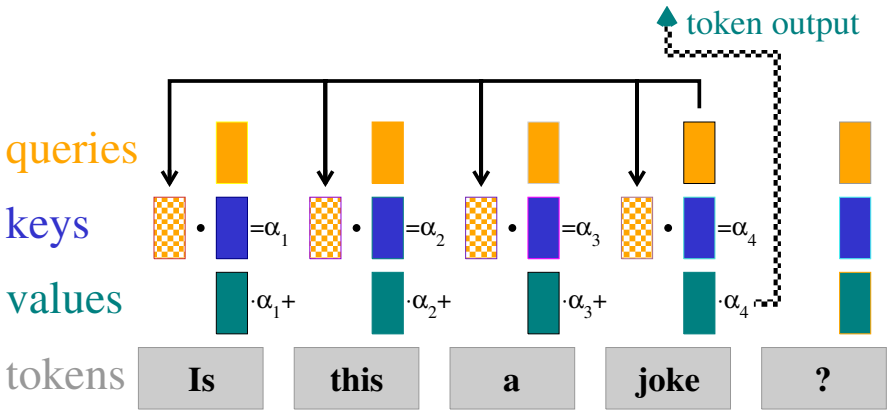

Fig. 10.11 Attention in machine learning. Tokens with activities $\mathbf{x}_i$ generate query/key/-value vectors as $\mathbf{Q}_i = \hat{Q}_i \cdot \mathbf{x}_i$, $\mathbf{K}_i = \hat{K}_i \cdot \mathbf{x}_i$ and $\mathbf{V}_i = \hat{V}_i \cdot \mathbf{x}_i$. Causality implies to attend only to previous words in a sentence. For a given token, here 'joke', the scalar product of its query vector with all previous key vectors generates weights α_i. The sum of the weighted values is the output of the attention layer. See (10.61) and (10.62).

Attention works in contrast on higher level, deciding whether somewhat substantial chunks of data are of interest or not.

Tokenization We have seen previously that it may not be desirable to work directly with raw data.[12] For time series', this amounts to extract meaningful symbols, a procedure denoted "tokenization" in the context of machine learning. For text processing, the most widely used tokens are word equivalents, with token vocabularies having typically about $5 \cdot 10^4$ entries. Tokens are mapped to high-dimensional activity states $\mathbf{x}_i$ in $\boldsymbol{R}^D$, e.g. with $D = 2^9$. Further processing will benefit if embedding schemes respect semantic relations between tokens.

Pairwise Attention Attention involves pairs of tokens where one token needs to decide if another token carries relevant information. This decision cannot be carried out directly on the level of the respective token activities, say $\mathbf{x}_i$ and $\mathbf{x}_j$, with the reason being that activity encodings are constrained by the nature of the overall task. For a working attention mechanism, tokens therefore need to generate associated objects. Given that attention is inherently asymmetric, at least two additional objects are necessary, "query" and "key". In the end, a given token will receive relevant information not just from a single other token, but from a larger number. A third object is hence needed for the alignment of the respective representations, "value".

Query, Key, Value The constituent units of attention-based machine learning models, such as transformers, are tokens, with high-dimensional activity

[12] See the discussion on time series characterization, Sect. ?? of Chap. ??, "??".

vectors $\mathbf{x}_i$. Tokens come with three matrices,

$$\hat{Q}_i, \qquad \hat{K}_i, \qquad \hat{V}_i\,,$$

denoted query, key and value. The entries of these matrices are adapted via backpropagation during training. The three associated query, key and value vectors,

$$\mathbf{Q}_i = \hat{Q}_i \cdot \mathbf{x}_i, \qquad \mathbf{K}_i = \hat{K}_i \cdot \mathbf{x}_i, \qquad \mathbf{V}_i = \hat{V}_i \cdot \mathbf{x}_i\,, \tag{10.60}$$

are activity dependent. In order to decide whether token j carries information that is relevant to i, the respective queries and keys are compared, viz $\mathbf{Q}_i$ with $\mathbf{K}_j$.

Dot Product Attention As a matter of principle, many methods would be suitable for comparing query and key vectors, say $\mathbf{Q}_i$ with $\mathbf{K}_j$. Given that the involved query and key matrices are anyhow adapted during training, the respective scalar product,

$$e_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j\,,$$

is sufficient. For a measure of how relevant j is to i, we need to map e_{ij} to a value $\alpha_{ij} \in [0, 1]$, e.g. via

$$\alpha_{ij} = \frac{1}{Z_i}\,\mathrm{e}^{-\beta e_{ij}}, \qquad Z_i = \sum_{j\le i} \mathrm{e}^{-\beta e_{ij}}, \qquad \sum_{j\le i} \alpha_{ij} = 1\,, \tag{10.61}$$

where β is an effective inverse temperature. The Boltzmann distribution (10.61) is denoted "softmax" in machine learning jargon.

Causal Attention In (10.61) attention is paid only to past events $j \le i$, the causality condition. Attention is not an all-or-nothing affair, but a graded process, with weights that are given by the entries α_{ij} of the attention matrix. The output activity $\mathbf{y}_i$ of token i is determined via

$$\mathbf{y}_i = \sum_{j\le i} \alpha_{ij} \mathbf{V}_j \tag{10.62}$$

by the amount of attention paid to what happened before. An illustration is given in Fig. 10.11. The past becomes irrelevant in the limiting case $\alpha_{ij} = \delta_{ij}$, which is included in (10.62).

Transformer The primary task of attention is information routing. Once selected, information needs also to be processed. For transformer models this is done on a token-per-token level, by feeding $\mathbf{y}_i$ into a perceptron with a single hidden layer. Transformer layers consist hence of two sublayers, attention and a classical token-wise neural net.

Linearized Attention Attention is a non-linear process scaling quadratically with the number of tokens per layer, N_T, the context length. There

are $N_\mathrm{T}^2/2$ scalar products $\mathbf{Q}_i \cdot \mathbf{K}_j$ to be computed, which are individually quadratic in the respective token activities, $\mathbf{x}_i$ and $\mathbf{x}_j$. Linearized versions of attention forfeit the mapping (10.61) to a normalized distribution, using instead

$$\mathbf{y}_i\big|_\mathrm{lin} = \sum_{j\le i} (\mathbf{Q}_i \cdot \mathbf{K}_j)\mathbf{V}_j = \mathbf{Q}_i \cdot \sum_{j\le i} \mathbf{K}_j \otimes \mathbf{V}_j \tag{10.63}$$

for the new token activity, modulo a normalization factor, where $\otimes$ denotes the outer product.

Recurrent Linear Attention We rewrite (10.63) as

$$\mathbf{y}_i\big|_\mathrm{lin} = \mathbf{Q}_i \cdot \hat{S}_i, \qquad \hat{S}_i = \sum_{j\le i} \gamma^{i-j}\, \mathbf{K}_j \otimes \mathbf{V}_j$$

where we introduced with $\gamma \in [0,1]$ an additional decay factor. It plays a role equivalent to the spectral radius of recurrent networks, see Sect. 10.2.1. Past events are explicitly discounted when $\gamma < 1$. The update relation

$$\hat{S}_{i+1} = \mathbf{K}_{i+1} \otimes \mathbf{V}_{i+1} + \gamma \hat{S}_i \tag{10.64}$$

shows that the inference effort scales linearly with context length N_T. This formulation is also denoted "retentive network".

One can interpret (10.64) as an abstract recurrent network unfolding in 'context time', with the entries of the key-value matrix $\hat{S}_i$ corresponding to neuronal activities. The new network activity $\hat{S}_{i+1}$ is given by the sum of two terms, the input $\mathbf{K}_{i+1} \otimes \mathbf{V}_{i+1}$ and the old activity $\hat{S}_i$, with the latter being discounted by the spectral radius γ of the network.