

Transformed data

Lecture 22

Consider a transform with a single parameter: $\vec{s}(\vec{x}, \xi)$ [$\vec{s}(\vec{x}, 0) = \vec{x}$]

In the infinite data limit, the sum-of-squares error function is given by:

$$E = \frac{1}{2} \iint d\vec{x} dt (y(\vec{x}) - t)^2 p(t|\vec{x}) p(\vec{x})$$

1D output (single output node) for simplicity

Imagine that each \vec{x} is perturbed many times: $\vec{x} \rightarrow \vec{s}(\vec{x}, \xi)$, where ξ is drawn from $p(\xi)$.

Then

$$\tilde{E} = \frac{1}{2} \iiint d\vec{x} dt d\xi (y(\vec{s}(\vec{x}, \xi)) - t)^2 p(t|\vec{x}) p(\vec{x}) p(\xi)$$

↑
over expanded data set

Now, assume that $\int d\xi \xi p(\xi) = E(\xi) = 0$,

$$E(\xi^2) = \int d\xi \xi^2 p(\xi) = \lambda \ll \text{small variance}$$

s.t. we only consider "small" transformations of \vec{x} .

Then

$$\begin{aligned} \bar{S}(\vec{x}, \xi) &= \bar{S}(\vec{x}, 0) + \xi \underbrace{\frac{\partial}{\partial \xi} \bar{S}(\vec{x}, \xi)}_{\bar{\tau}} \Big|_{\xi=0} + \\ &+ \frac{\xi^2}{2} \underbrace{\frac{\partial^2}{\partial \xi^2} \bar{S}(\vec{x}, \xi)}_{\bar{\tau}'} \Big|_{\xi=0} + \mathcal{O}(\xi^3) \end{aligned}$$

Next,

$$\begin{aligned} \bar{y}(\bar{S}(\vec{x}, \xi)) &= \bar{y}(\vec{x} + \xi \bar{\tau} + \frac{\xi^2}{2} \bar{\tau}') = \\ &\approx \bar{y}(\vec{x}) + \xi \bar{\tau}_i \frac{\partial \bar{y}(\vec{x})}{\partial x_i} + \frac{\xi^2}{2} \bar{\tau}'_i \frac{\partial \bar{y}(\vec{x})}{\partial x_i} + \\ &+ \frac{\xi^2}{2} \bar{\tau}_i \bar{\tau}_j \frac{\partial^2 \bar{y}(\vec{x})}{\partial x_i \partial x_j} \quad \text{sums over } i, j \text{ implied} \end{aligned}$$

Then

$$\begin{aligned} \bar{E} &= \frac{1}{2} \iiint d\vec{x} dt d\xi p(t|\vec{x}) p(\vec{x}) p(\xi) \times \\ &\times \left[\bar{y}(\vec{x}) + \xi \bar{\tau}_i \frac{\partial \bar{y}}{\partial x_i} + \frac{\xi^2}{2} \bar{\tau}'_i \frac{\partial \bar{y}}{\partial x_i} + \frac{\xi^2}{2} \bar{\tau}_i \bar{\tau}_j \frac{\partial^2 \bar{y}}{\partial x_i \partial x_j} - \right. \\ &\quad \left. - t \right]^2 \approx \\ &\approx \frac{1}{2} \iiint d\vec{x} dt \underbrace{[\bar{y}(\vec{x}) - t]^2}_{E} p(t|\vec{x}) p(\vec{x}) \quad \text{⊕} \end{aligned}$$

$\int d\xi p(\xi) = 1$

$$\oplus \underbrace{E(\xi)}_{\text{"0"}} \frac{1}{2} \int \int d\vec{x} dt \dots + \underbrace{E(\xi^2)}_{\lambda} \frac{1}{2} \int d\vec{x} dt p(t|\vec{x}) p(\vec{x}) \times$$

$$\times \left[(y(\vec{x}) - t) \left[\tau_i \frac{\partial y}{\partial x_i} + \tau_j \tau_j \frac{\partial^2 y}{\partial x_i \partial x_j} \right] + \tau_i \frac{\partial y}{\partial x_i} \tau_j \frac{\partial y}{\partial x_j} \right]$$

So, $\tilde{E} = E + \lambda \Omega$, where

$$\Omega = \frac{1}{2} \int d\vec{x} p(\vec{x}) \left[(y(\vec{x}) - \underbrace{E[t|\vec{x}]}_{\int dt p(t|\vec{x}) t}) \left[\dots \right] + \tau_i \frac{\partial y}{\partial x_i} \tau_j \frac{\partial y}{\partial x_j} \right]$$

↑ $\int dt p(t|\vec{x}) = 1$

Now, note that $y(\vec{x}) = E[t|\vec{x}]$ minimizes E and "almost" minimizes

$$\tilde{E}: y(\vec{x}) = E[t|\vec{x}] + \theta(\xi) \quad \text{minimizes } \tilde{E} \text{ in fact}$$

Thus, the 1st term in Ω is $\theta(\xi)$ while the 2nd is $\theta(\xi^0)$, so that

$$\Omega \approx \frac{1}{2} \int d\vec{x} p(\vec{x}) \left(\underbrace{\tau_i \frac{\partial y}{\partial x_i}}_{\text{1D Jacobian}} \right)^2 \ll \text{same as before!}$$

$$E(x_i x_j) = \begin{cases} \int \int dx_i dx_j x_i x_j p(x_i) p(x_j) = 0, & i \neq j \\ \int dx_i x_i^2 p(x_i) \equiv \lambda, & i = j \end{cases}$$

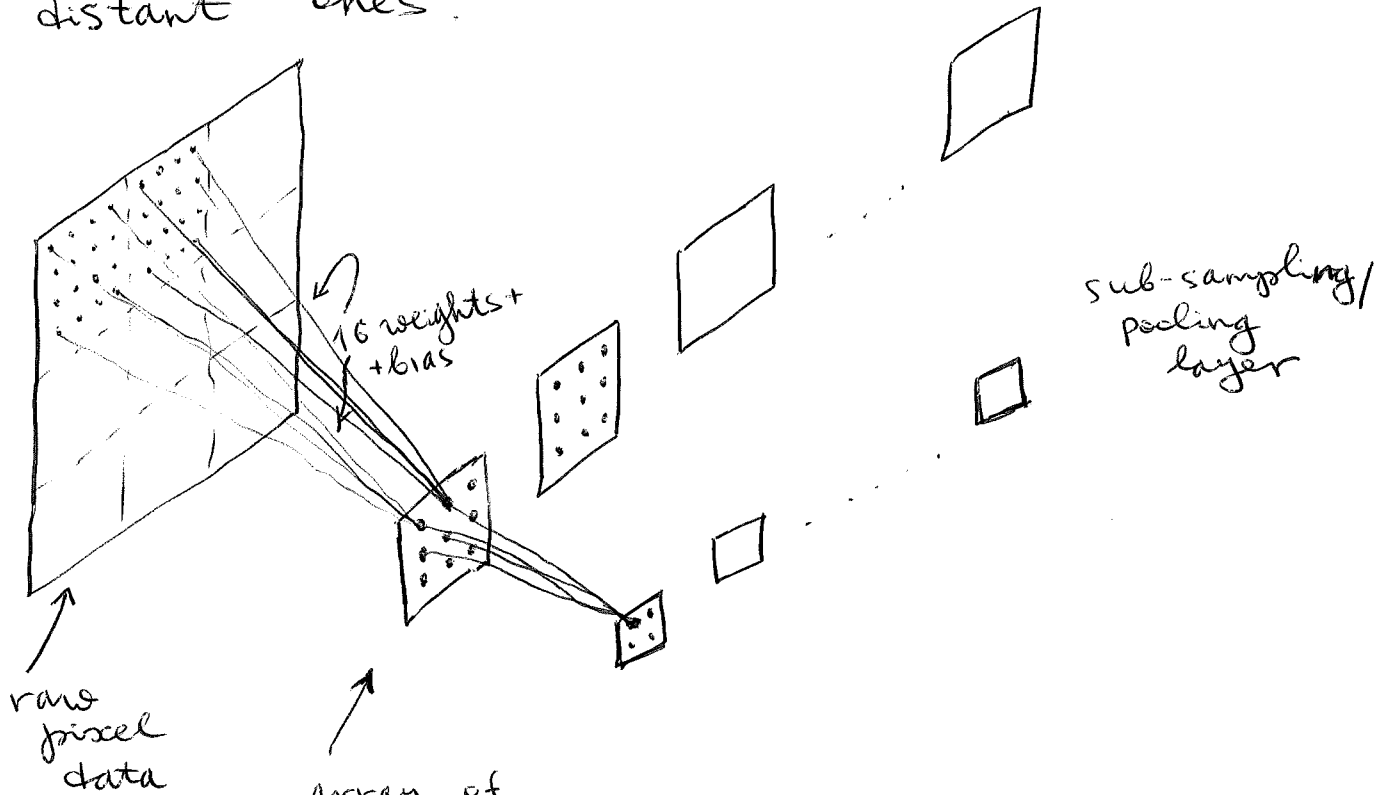
$$\text{Here, } \Omega = \frac{1}{2} \int d\vec{x} p(\vec{x}) \left[\underbrace{(y(\vec{x}) - E[t|\vec{x}])}_{\mathcal{O}(\|\vec{x}\|), \text{ discard}} \underbrace{\nabla^2 y(\vec{x})}_{\sum_i \nabla_i^2} + \underbrace{\nabla_i y(\vec{x}) \nabla_i y(\vec{x})}_{\|\nabla y\|^2} \right] =$$

$$= \frac{1}{2} \int d\vec{x} p(\vec{x}) \|\nabla y(\vec{x})\|^2.$$

Tikhonov regularization

Convolutional networks

Idea: avoid manual feature extraction by extracting higher-and-higher level invariant features from raw pixel data. Use the fact that nearby pixels are more strongly correlated than distant ones.



array of feature maps; each node in a given feature map has the same weights + bias \Rightarrow acts as a kernel transform : $h(\sum_j W_{kj} x_j)$ (or convolution)

these weights are called a filter bank

Idea: a given feature map detects the presence of a given feature anywhere within a map \Rightarrow weights must be shared since the feature to be detected is the same.

Each pooling node combines data from several nodes in a given feature map (or several feature maps), to reduce dimension of the representation and decrease sensitivity to small shifts & distortions.

Implementation: 1. $\sigma(\underbrace{\tilde{w} \langle z \rangle}_{\text{average of all inputs}} + \tilde{w}_0)$ [older]
2. $\max\{z_j\}$ max-pooling

Finally, 2-3 stages of feature extraction (convolution) & pooling are stacked, with the ~~no~~ number of feature maps going up in a given layer as features become higher-level (but lower-dimensional).

The final two layers are a convolutional layer which is fully connected to an output layer, typically with soft-max activation functions for $K > 2$ classification.

Modern implementation:

1. Use $\text{ReLU}(\cdot)$ as $h(\cdot)$ in convolutional layers, where $\text{ReLU}(z) = \max(z, 0)$, rather than $\underbrace{\text{rectified linear unit}}_{\text{unit}}$ $\sigma(z)$ or $\tanh(z)$.

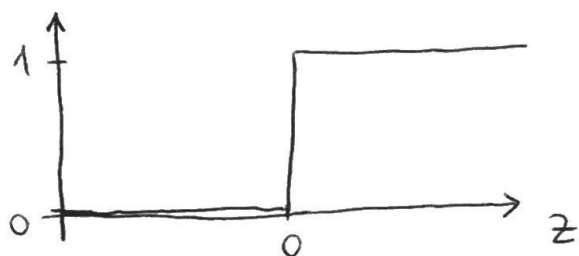
Typically learns much faster with $\text{ReLU}(\cdot)$.

2. Use stochastic gradient descent for training \Rightarrow steepest descent informed by a few datapoints at a time rather than all of them.
3. Use backpropagation to compute the gradients
4. In deep NNs, pretrain intermediate convolutional layers using Boltzmann machines (refine afterwards, pre-trained weights by backpropagation)

Non-linear activation functions

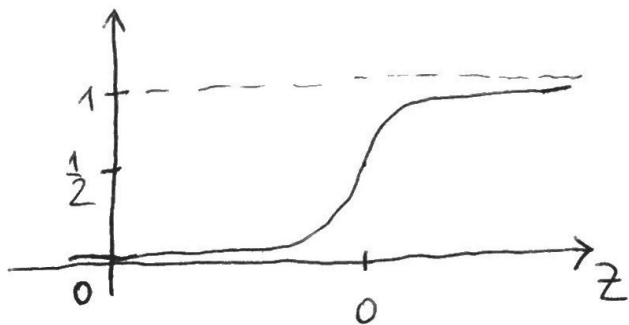
"Classical" activation functions are prone to saturation on the tails, where the gradients become small or vanish completely:

Perceptron



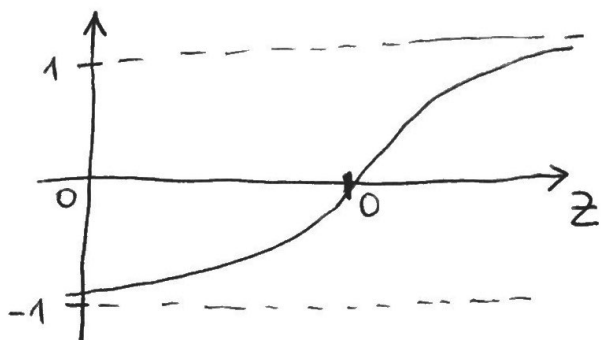
$$\theta(z)$$

Sigmoid



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

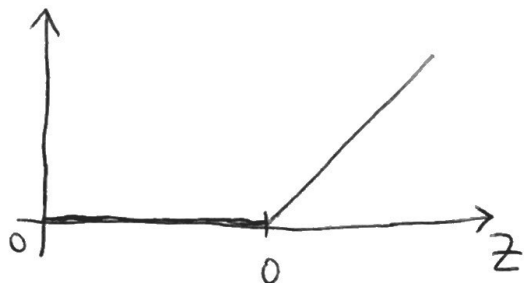
Tanh



$$\tanh(z)$$

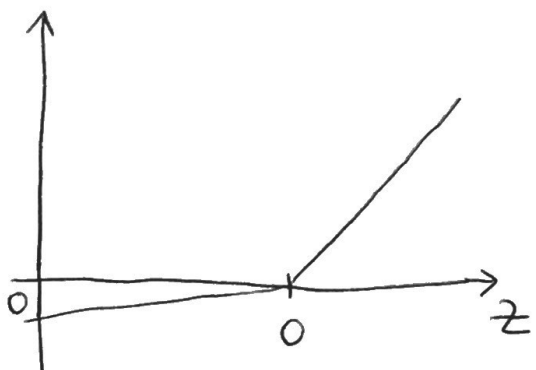
Gradient saturation is widely believed to reduce convergence and/or predictive power. Therefore, other activation functions have been proposed:
(adopted)

ReLU



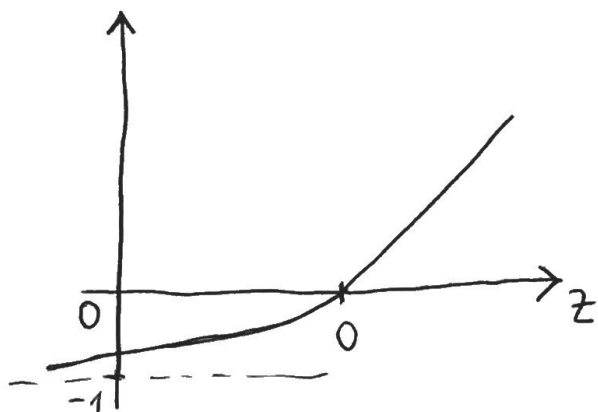
$$\max(0, z)$$

Leaky ReLU



$$\begin{cases} z, z \geq 0 \\ 0.1z, z < 0 \end{cases}$$

ELU



$$\begin{cases} z, z \geq 0 \\ e^z - 1, z < 0 \end{cases}$$