Is Kernel Prediction More Powerful than Gating in Convolutional Neural Networks?

Lorenz K. Müller [1]

Abstract

Neural networks whose weights are the output of a predictor (HyperNetworks) achieve excellent performance on many tasks. In ConvNets, kernel prediction layers are a popular type of HyperNetwork. Previous theoretical work has argued that a hierarchy of multiplicative interactions exists in which gating is at the bottom and full weight prediction, as in HyperNetworks, is at the top. In this paper, we constructively demonstrate an equivalence between gating combined with fixed weight layers and weight prediction, relativizing the notion of a hierarchy of multiplicative interactions. We further derive an equivalence between a restricted type of HyperNetwork and factorization machines. Finally, we find empirically that gating layers can learn to imitate weight prediction layers with an SGD variant and show a novel practical application in image denoising using kernel prediction networks. Our reformulation of predicted kernels, combining fixed layers and gating, reduces memory requirements.

1. Introduction

A powerful generalization of neural networks is to transform their weights from static parameters to dynamic predictions from the network's current input. Such networks have been termed HyperNetworks (Ha et al., 2016) and Fast Weight Networks (Schmidhuber, 1992). While not always viewed under this perspective, models of this kind are part of widely used methods in recommender systems (Rendle, 2010) (Factorization Machines (FM)), natural language processing (NLP) (Vaswani et al., 2017) (Transformers) and various image processing tasks (Howard et al., 2019; Bako et al., 2017) (Attention, Kernel Prediction).

[1]Computing Systems Lab, Huawei Technologies, Zurich, Switzerland. Correspondence to: Lorenz K. Müller <lorenz.muller@huawei.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

Gating mechanisms in neural networks are commonly used in recurrent neural network (RNN) architectures (Hochreiter & Schmidhuber, 1997; Chung et al., 2014), where they have enabled successes in tasks like speech recognition and machine translation (Lipton et al., 2015). The gating mechanism dynamically controls information flow in these architectures, particularly in the sequence dimension. Additionally, such mechanisms can prevent the vanishing gradient problem of RNNs (Hochreiter & Schmidhuber, 1997).

In several applications for which RNNs have been the state-of-the-art approach for some time, now transformer architectures (Vaswani et al., 2017) achieve even better results. One key aspect in which transformers differ from gating architectures is how they handle data dependency across the sequence dimension: Recurrence and gating are supplanted by dynamic prediction of attention weight matrices (Bahdanau et al., 2016), which express relationships between positions in the sequence dimension. Transformers have now also spread to domains like image processing (Dosovitskiy et al., 2021) and recommender systems (Sun et al., 2019).

Also, many architectures exist in convolutional networks that use weight prediction (or kernel prediction). In this context predicting weights can be especially effective when it enables the 'feed-forward' network (for which the weights are predicted) to be limited to a constrained form: E.g., a single color transformation matrix per pixel (Gharbi et al., 2017), a denoising kernel per pixel (Mildenhall et al., 2018) or an explainable, linear model (Bohle et al., 2021). Additionally, predicted kernels in convolutional networks allow decoupling parameter count from required FLOPs, which can benefit lightweight compute settings (Muller, 2021).

Gating layers have recently been adopted outside their original application sphere in RNNs. In the NLP domain, gMLPs (Liu et al., 2021) propose to use a gating layer as a stand-in for the standard multi-headed attention in a transformer-like architecture. In image enhancement, NAFNet (Chen et al., 2022) uses multiple forms of linear gating for highly effective image denoising. In implicit representations, element-wise multiplication has also been used as a lightweight alternative to full HyperNetworks (Mehta et al., 2021).

Here, we extend the prior perspective (Jayakumar et al.,

2020) on the relationship of weight prediction and gating by taking into account the impact of fixed-weight layers that are virtually always present adjacent to gating layers. When not considering fixed-weight layers, gating *misleadingly* appears less general than weight prediction. In this paper, we show that they are on equal footing from this more realistic perspective.

1.1. Contributions

The main contributions of our paper are:

- Showing that the hierarchy of multiplicative interactions of (Jayakumar et al., 2020) misses an important piece of the puzzle: The hierarchy is flat if fixed layers are taken into account.
- Theoretically motivating works like (Liu et al., 2021; Mehta et al., 2021; Chen et al., 2022) by deriving a new relationship between gating layers and HyperNetworks (Theorem 3.1) also for the case spatial filters (Corollary 3.2).
- Empirically studying the learnability of given weight prediction layers by gating layers (Sec. 4) for random and trained networks.
- Applying gating layers to reduce the memory footprint of a real-world image burst denoising network (Sec. 4.3) based on kernel prediction.
- Deriving a new relationship between FMs / low-rank matrix factorization and HyperNetworks (Theorem 3.3)

2. Background

This section summarizes informal descriptions of the models we will examine in the rest of the paper and provides the formal definitions we will use later.

2.1. HyperNetworks

Motivated by neuroscientific observations (Kupfermann, 1979), researchers in natural language processing have argued (long before practical demonstrations) that having weights in a neural network with different time constants ('fast' and 'slow' weights), allows for more powerful models (Hinton, 1987).

Specifically, (Schmidhuber, 1992) describes Fast Weight Memories as using a slowly learning feed-forward network that predicts weight changes for a second 'fast-weight' Network. Several following papers propose the idea of predicting weights for neural networks with neural networks independently, e.g., as HyperNetworks (Ha et al., 2016) and (in the convolutional case) as Dynamic Convolutions (Klein et al., 2015; Riegler et al., 2015).

The transformer (Vaswani et al., 2017) is a particularly widespread variant of HyperNetworks (Schlag et al., 2021) (in the sense of the definition for HyperNetworks we use here). The multi-headed attention of transformers predicts a weight matrix that is to be applied; the distinguishing features compared to a general HyperNetwork are that the predicted matrix is applied to the 'sequence' dimension of the input (not e.g., the 'channel' dimension) and that the weight prediction takes a particular form (multi-headed attention). Some further explanations are given in the appendix, Sec. A.5.3.

Formally, let a HyperNetwork layer be a neural network layer, whose weights have been predicted by a 'weight-prediction' neural network $\text{NN}^W(\cdot)$ (whose output is two dimensional, a matrix)

$$x_j^l = f\left(\sum_i [\text{NN}^W(\vec{x}^0)]_{ij} \cdot x_i^{l-1}\right) \tag{1}$$

where $\vec{x}^0$ is the input, $\vec{x}^l$ is the lth layer's activation, and $f(\cdot)$ is the non-linearity. The collection of such layers, excluding the weight-prediction, will be referred to as the 'feed-forward' network. This definition corresponds closely to the ideas of (Ha et al., 2016; Schmidhuber, 1992).

A common special case is that the weight prediction network NN^W and the feed-forward network share some parameters. This is, for example, the case when NN^W takes as its inputs the output of the directly preceding feed-forward layer rather than the input of layer 0.

2.2. Factorization Machines

A factorization machine (FM) (Rendle, 2010) generalizes low-rank matrix factorization. The two are equivalent when the input to the FM is binary. The key feature of an FM is that the quadratic interactions are formulated efficiently, such that it computes a matrix entry in $O(kn)$ for an $n \times n$ matrix approximated at rank k.

Formally, (Rendle, 2010) defines the factorization machine (FM) by the equation

$$y_j = w_j + \sum_{i=0}^{n-1} w_{ji} x_i + \sum_{i=0}^{n-1} \sum_{k=i+1}^{n-1} \left(\sum_{l=0}^{n_l-1} v_{jil} v_{jkl}\right) x_i x_k. \tag{2}$$

Here $\vec{x}$ are input features, $\vec{y}$ is the regression target, $\vec{w}$ are biases for $\vec{y}$, $\mathbf{W}$ are the learned weights of a linear predictor and the learned weight tensor $\mathbb{V}$ is used to construct a low-rank 3-tensor that weighs feature-crosses of $\vec{x}$; j indexes entries of $\vec{y}$, i, k index $\vec{x}$, and finally l indexes (low-rank) dimension of the V.

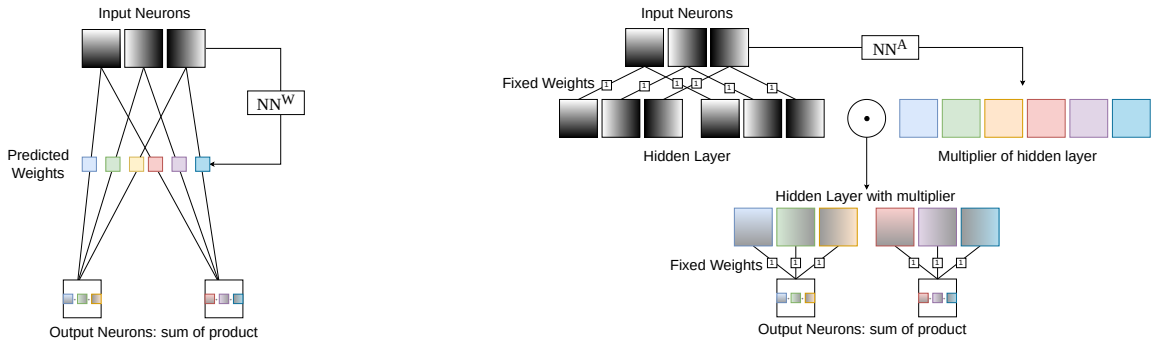

Figure 1. Predicting weights with a neural network NN^W is equivalent to predicting an element-wise multiplier (with NN^A) combined with two fixed layers of appropriate size (without non-linearity). The fixed layers are used to project into a high enough dimension in which the element-wise multipliers can take the role of weights. See Theorem 3.1 for details. Best viewed on a screen.

2.3. Gating Layers

In a gating layer, two sets of activations (or, for convolutions, feature maps) are multiplied element-wise. The output of such an operation is sometimes referred to as a dynamic feature (Han et al., 2021). Some key examples where such an operation is used in practice are recurrent neural networks (e.g., in LSTMs (Hochreiter & Schmidhuber, 1997)), MLP variants (e.g. gMLP (Liu et al., 2021)), convolutional neural networks (e.g., NAFNet (Chen et al., 2022)) or implicit neural networks (e.g., modulated periodic activations (Mehta et al., 2021)). Some authors label such operations as a form of attention (e.g., squeeze-and-excitation layers (Hu et al., 2018) or simplified channel attention (Chen et al., 2022)).

Formally, given two input vectors $\vec{x}$, $\vec{a}$ a gating layer computes an output $\vec{y}$ by

$$y_i = x_i \cdot \sigma(a_i) \tag{3}$$

where $\sigma(\cdot)$ is some activation function (and may also be the identity).

Given two collections of feature maps X, A, a gating layer computes an output collection of feature map Y by

$$Y_{cij} = X_{cij} \cdot \sigma(A_{cij}) \tag{4}$$

where c indexes the channel and i, j the two spatial dimensions.

3. Theory

3.1. HyperNetwork Layers and Gating Layers

Let a gated double layer (GDL) be

$$x_j^l = f\left(\sum_{k=1}^{n_k} u_{kj} \cdot [\text{NN}^a(\vec{x}_0)]_k \cdot \left(\sum_{i=1}^{n_i} v_{ik} \cdot x_i^{l-1}\right)\right) \tag{5}$$

where $\vec{x}^0$ is the network's input, $\vec{x}^l$ is the lth layer's activation, $f(\cdot)$ is the non-linearity, u_{kj} and v_{ik} are fixed weight matrices, and $[\text{NN}^a(\vec{x}_0)]_k$ is a neural network, whose output is vectorial (with n_k entries). Note that the only non-linearities occur inside NN^a and at the end with $f(\cdot)$. Finally, n_i is the input, n_j is the output, and n_k is the latent dimensionality.

Theorem 3.1. *For any HyperNetwork layer H (Eq. 1), there exists a gated double layer G (Eq. 5) that produces the same output $\vec{y}$ for any given input $\vec{x}$ if $n_k \geq n_j \cdot n_i$.*

Proof. A visual example of the central idea of this proof is given in Fig. 1. In summary, we can use the matrices u_{kj} and v_{ik} as indexes such that each entry of $[\text{NN}^a(\vec{x}_0)]_k$ maps to exactly one entry of $[\text{NN}^W(\vec{x}^0)]_{ij}$. This is possible when there are sufficiently many entries in $[\text{NN}^a(\vec{x}_0)]_k$, namely when $n_k \geq n_j \cdot n_i$. The detailed proof is in the appendix in Sec. A.1. □

Note that this further implies that HyperNetworks and networks consisting of GDLs have equal representational power and relativizes the notion of a hierarchy of multiplicative interactions in the presence of fixed weight layers.

3.1.1. Kernel Prediction

An interesting variant of this construction concerns Kernel Prediction Networks (Mildenhall et al., 2018), where a deep network predicts denoising kernels *per pixel* of an input image sequence. A convolutional variant of the gated double layer can emulate the application of these *dynamic, local* kernels.

Let a kernel prediction layer acting on input feature maps $X \in \mathbb{R}^{C \times H \times W}$ (where C, H, W are the input's number

of channels, height and width respectively) to output $Y \in \mathbb{R}^{1\times H\times W}$ be a layer that applies an individual predicted kernel $[\text{NN}^K(X)]$ of size $f \times f$ centered on each pixel; as a formula

$$Y_{ij} = \sum_{c,f_i,f_j} X_{(i+f_i)(j+f_j)c} \cdot [\text{NN}^K(X)]_{ijc\,f_i\,f_j} \quad (6)$$

Let a convolutional gated double layer acting on X to output Y (as above) be a fixed convolution layer, followed by an element-wise multiplier (that is predicted from input) and another fixed convolution layer; as a formula (assuming that the second fixed filter convolution is 1x1 for better legibility)

$$Y_{ij} = \sum_{c,f_i,f_j,h} X_{(i+f_i)(j+f_j)c} \cdot F_{hc\,f_i\,f_j} \cdot [\text{NN}^A(X)]_{ijh} \cdot G_h \quad (7)$$

Here, indices i, j run over horizontal and vertical pixel positions, f_i, f_j filter positions, and c, h index the channel dimension. G and F are filter banks and X, Y and $[\text{NN}^A(X)]$ are sets of feature maps.

Corollary 3.2. *For any kernel prediction layer K with filter size f, there exists a convolutional gated double layer L that produces the same output for any given input if $C_k \geq C_X \cdot C_Y \cdot f^2$.*

Proof. We can reformulate K as a HyperNetwork layer H by appropriately reshaping the input tensor X by flattening it. The number of entries in the resulting predicted matrix is $C_X \cdot C_Y \cdot H \cdot W \cdot f^2$, so by Theorem 3.1 we need to make sure that 1) the element-wise multiplier we construct has at least this many entries (and again use the fixed layers to do indexing) and that 2) the filter size of at least one of the fixed filters in the gated double layer has sufficient spatial extent (i.e. $f \times f$). □

In Sec. 4, we show empirically that this substitution works in practice (even for a memory-saving low-rank approximation) for the same setup as the one by (Mildenhall et al., 2018) on the task of image burst denoising.

3.2. Linear One-layer HyperNetworks are FMs

Let a single (low-rank) linear HyperNetwork layer with linear weight prediction Linear^W be

$$\begin{aligned} x^1_j &= \sum_i [\text{Linear}^W(\vec{x}^0)]_{ij} \cdot x^0_i + w_j \quad (8) \\ &= \sum_{i=0}^{n-1} \underbrace{\left(w_{ji} + \sum_{k=0}^{n-1} u^{\text{rank-}n_l}_{jik} x_k \right)}_{w^{\text{eff}}_{ji}} x_i + w_j \end{aligned}$$

where $\vec{x}^0$ are input features (indexed by j), $\vec{x}^1$ is the regression target (indexed by i, k), $\vec{w}$ is the bias for each target (indexed by j), $\mathbf{W}$ are the biases of the linear weight prediction (indexed by j, i), and $\mathbb{U}$ is the (low-rank) tensor that linearly maps input features to predicted weights (indexed by j, i, k).

Theorem 3.3. *Low-rank linear HyperNetwork layers with linear weight prediction (Eq. 8) are Factorization Machines (Eq. 2) for binary input vectors.*

The proof is in the appendix in Sec. A.2.

Note that binary inputs are not an artificial but practically relevant case for FMs: Binary input FMs correspond to low-rank matrix factorization (Rendle, 2010). In the non-binary case, the low-rank linear HyperNetwork layer is an FM with self-interactions, i.e. non-zero coefficients in front of square terms.

4. Experiments

In this section, we demonstrate empirically that for a given HyperNetwork layer H, an equivalent gated double layer G does not only exist but can also be learned (for definitions, see Sec.3). We tackle this question in three different settings.

In the first two settings, the task for the gated double layer G is to reproduce the output of a fixed HyperNetwork layer H. The construction of H differs between the two: First, the weights of H are drawn at random and second, H is trained on CIFAR-10.

The third experiment takes a 'real-world' network, the Kernel Prediction Network (KPN) of (Mildenhall et al., 2018), trained for image burst denoising. Instead of learning to reproduce the output of a pre-trained network, we train a KPN variant from scratch, where a convolutional gated double layer replaces the kernel prediction.

We do not perform experiments relating to Theorem 3.3, because in this case there is a one-to-one correspondence of parameters between the FM and the linear HyperNetwork, such that the learning dynamics are also identical.

For better reproducibility, we include code for the two following subsections in the supplemental materials[1]. The total compute budget for this experimental section amounts to a few GPU days.

4.1. Random Networks

Here we examine the question: Given a fixed, random HyperNetwork layer H, will a gated double layer G be able to emulate H when trained with an SGD variant? This

[1]Found here: https://drive.proton.me/urls/W8S6343JZ4#gpGZdMJM4eTv

question is relevant because while Theorem 3.1 guarantees the existence of such a G and gives a way to construct it by hand, in practice, neural networks are trained by SGD variants. Whether or not SGD can find the given (or a different) assignment of variables in G such that it matches H can give us an idea of whether GDLs can replace predicted weights in practice.

We are also interested in how low-rank or over-complete variants of G affect trainability and vary the hidden layer's size in G. We also train an MLP M to emulate H as an illustrative baseline. M is dimensioned such that it is at least as large as G in terms of FLOPs. See Fig. 2 for details of H, G and M.

To train G and M we feed 5'000'000 random normal vectors of length 128 into H and observe its output. G and M are trained with Adagrad[2] (Lydia & Francis, 2019) and a cosine learning rate schedule (Loshchilov & Hutter, 2017) on the L2 loss to match this output. After each epoch, we test the L2 loss of G and M's predictions on 1'000 previously unseen random normal vectors.

The architectures of H, G and M are described schematically in Fig. 2. We introduce a scale parameter s that scales the hidden layer of G and M. When $s = 1$, the hidden layer of G has the minimum size required that there exists a weight assignment for G that matches any given H in general (this corresponds to the equality of the condition $n_k \geq n_j \cdot n_i$ in Theorem 3.1).

Fig. 3 shows the training curve of a single run and the final relative error (MSE divided by the mean of the square of the output of H) as a function of the ratio 'scale' between the number of predicted weights in H and number of the predicted element-wise multiplier in G (for different sizes 'dim' of the predicted weight in H).

We observe that sufficiently large G approximate H well, while M plateau at a higher error. Neither of the networks overfits the training data (which is unsurprising given the high number of training samples). We also see that larger networks converge to a smaller error within the same number of epochs; in principle even at scale 1 a perfect solution exists (as shown in Theorem 3.1), but it appears Adagrad does not recover it perfectly with the given hyperparameters and training time.

One might at first think that this indicates that the decomposition is not useful when training from scratch. However, we will see in the following experiments that for practical applications, even a low-rank approximation (scale < 1) can be sufficient to match the output quality in a practical task (image burst denoising). Related work shows further such cases, e.g., (Liu et al., 2021; Mehta et al., 2021). This is important because the GDL *at full rank* is potentially *less* efficient than a HyperNetwok.

[2]We tested Adam (Kingma & Ba, 2017), SGD and Adagrad at scale 1 and used the last as it worked best for both G and M.

4.2. Trained Networks

Here, we examine the question: Given a HyperNetwork layer H trained on CIFAR-10, will a gated double layer G be able to emulate H when trained with SGD? In contrast to the previous section with random networks, there is a finite probability that the predicted weight in H is low-rank, making it easier to predict for smaller G.

Again, we are also interested in how low-rank or over-complete variants of G affect trainability and vary the hidden layer's size in G. We also train an MLP M to emulate H as an illustrative baseline. M is dimensioned such that it is at least as large as G in terms of FLOPs.

CIFAR-10 (Krizhevsky et al., 2009) is a dataset of 60'000 colour images of size 32×32 of 10 different classes. As customary, we train on 50'000 images and use 10'000 for testing in the standard split. A fixed weight ConvNet (architecture details are given in the appendix) first extracts feature vectors of length $l \in \{16, 32, 64, 128\}$ from each image. On these feature vectors (and corresponding target weights), we train H and then train G and M to reproduce the output of H.

Similar to the section on random networks Fig. 3 shows training curves and final, relative L2 loss as a function of the hidden layer size a function of the ratio s between predicted weights in H and size of the predicted element-wise multiplier in G. The results look similar to the previous section: Sufficiently large G approximate H well, also for trained layers H. One notable difference is that the error drops off at a lower scale for the trained network. This is consistent with the possibility that the predicted weight in H is low-rank.

4.3. Kernel Prediction Network

Here we examine the question: Given a practical network that contains a HyperNetwork layer H, can we replace this layer H with a gated double layer L and train the model to equal quality as the original? Can we gain a computational benefit using a low-rank approximation of the H?

The standard KPN We consider the kernel prediction network (KPN) of (Mildenhall et al., 2018), an effective image burst denoiser. The KPN is a U-Net-like (Ronneberger et al., 2015) CNN that predicts an individual (spatial) denoising kernel centered on each pixel in the input image burst and then combines all pixels weighted by their kernels into a single denoised image. For an illustration of the network

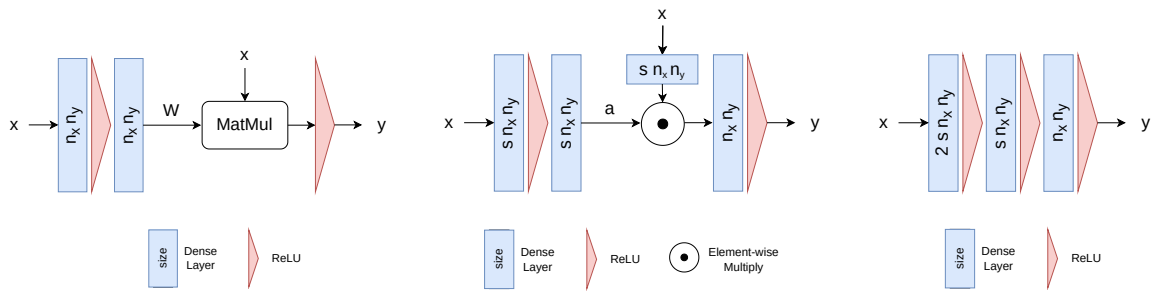

Figure 2. The networks used in the fully connected network experiments. (Left): The HyperNetwork H. (Middle): The gated double layer G. (Right) The MLP M, dimensioned to at least as many FLOPs as G. x is the input vector, y the output and $n_i, i \in \{x, y\}$ their respective sizes. s is an additional parameter that we vary to examine whether the predicted weight can be approximated at lower rank.

(a) Random: results (b) Random: learning (c) Trained: results (d) Trained: learning

Figure 3. Results for learning to emulate a random HyperNetwork layer. (a): Relative L2 error as a function of the hidden layer scale in the GDL G / in the MLP M. 'dim' indicates the size of the predicted weight matrix in H. (b): The training curve for scale $s = 1$. The GDL G is better than an equally sized MLP M at emulating a HyperNetwork H at all sizes. (c) and (d) show the same data for a CIFAR-10 pretrained HyperNetwork layer. (a) and (c) are averaged over 10 random initializations.

layout, see the appendix. Formally it reads

$$Y_{ijc} = \sum_{t,f_i,f_j} X_{(i+f_i)(j+f_j)ct} \cdot [\mathrm{NN}^K(X)]_{ijt,f_i,f_j} \tag{9}$$

note the additional time dimension (indexed by t) compared to Eq. 6 that is summed over (instead of over the channels). For an input image burst of height H, width W, length T and channel number C, the aim is to predict a single image of equal height, width and channel count. The tensor $[\mathrm{NN}^K(X)]_{ijct,f_i,f_j}$ that contains the collection of denoising kernels then has $H \times W \times T \times C \times f \times f$ entries, where f is the edge-length of one individual kernel. We consider the grey-scale case $C = 1$.

Gated Double Layer Decomposition We can replace the kernel prediction stage with the gated double layer as in Corollary 3.2. An illustration of the resulting network is given in Fig. 5 in the appendix, alongside an illustration of the original KPN.

As a formula, the double-gated decomposition reads (assuming that the second fixed filter convolution is 1×1 for better legibility; in practice, we use a $f \times f$ convolution for both G and F)

$$Y_{ijc} = \sum_{t,f_i,f_j} X_{(i+f_i)(j+f_j)ct} \cdot F_{ijht\,f_i\,f_j} \cdot [\mathrm{NN}^A(X)]_{ijh} \cdot G_{ijh} \tag{10}$$

note again the additional time dimension (indexed by t) compared to Eq. 7 that is summed over. F and G are fixed convolution filter banks. G takes as input all feature maps of the hidden layer that F projects into, and F takes as input $n_c \cdot n_t$ feature maps (one map per input channel per time-step in the burst). $\mathrm{NN}^A(X)$ is computed by a network that looks exactly like NN^K except for the last layer, which now has a different size, namely $H \times W \times C \times M$ (where M is the number of channels in the new hidden layer), such that it can be cast into the required shape. Note that the ratio between the sizes of the last layer in the original network and ours is $\frac{T \times f \times f}{M}$.

By choosing the hidden layer's channel count, M, we can either approximate the kernels NN^K at full rank or lower rank. Intuitively, it seems unlikely that full rank is required because some kernels are implausible for image denoising (e.g., a kernel that combines the four corners and nothing else; a similar observation is made by (Xia et al., 2020)). Here, we opt for a channel-count $M \in \{64, 128\}$. With this choice, the largest tensor computed in our network variant

is about $\{2.1, 1.5\}\times$ smaller than the largest tensor in the original KPN.

Setup details Apart from the last layer of the network, we use the identical setup as (Mildenhall et al., 2018) with 5×5 kernels based on the code[3] released by the authors. Hyperparameters are not tuned to our variant but are left as given in the code.

One modification we make concerns the validation data. Originally, the authors create a synthetic test set of validation image bursts from a set of non-public validation images; instead, we create a synthetic test set of validation image bursts from the publicly available McMaster set of images (Zhang et al., 2011). However, we use the identical code for the image to burst transformation. This transformation adds independently drawn Gaussian read and shot noise with the following standard deviations to each image I in the burst: $\sigma_{\text{read}} = 10^r$, where $r \sim U(-3, -1.5)$, $\sigma_{\text{shot}} = sqrt(I) \cdot 10^r$, where $r \sim U(-2, -1)$. Additionally, the transformation shifts the input image by up to 2 pixels with high probability and up to 16 pixels with low probability. Please consider the code linked in the footnote under the gamma corrected setting and the paper (Mildenhall et al., 2018) for further details.

Results Table 1 summarizes our results. The best-performing network uses our double-gated layer rather than the standard predicted kernel formulation. While the output quality improvement is insignificant, this network reduces memory requirements substantially (its widest layer has $1.5\times$ fewer channels) compared to the original network. Shrinking the hidden layer in the double-gated construction too far (Gated-64) deteriorates the output quality significantly.

5. Discussion

Theorem 3.1 gives a new perspective on why e.g. the NAFNet architecture (Chen et al., 2022) is so effective: The use of gating as an activation function effectively allows the network to implement a rank-constrained kernel prediction. Note that compared to GDL, the NAFNet layer is 'missing' the second fixed-weight layer, but in practice, this second set of weights can be omitted/absorbed into the first set of weights of the subsequent layer (if the two layers have the same size). Generally, gating layers are virtually always used together with fixed-weight layers. These fixed-weight layers can 'absorb' the fixed weights of the GDL in practice.

A potential reason for the recent popularity of gating in place of weight prediction could lie in the ease of initialization: As representative examples, both dealing with a variant of image enhancement for their HyperNetwork (Muller, 2021) devote a subsection to initialization methods where NAFNet (Chen et al., 2022) works 'out-of-the-box'.

Similarly, the effectiveness of 'modulated periodic activations' by (Mehta et al., 2021) as an alternative to HyperNet-SIRENs in implicit image representation networks (Sitzmann et al., 2020) is unsurprising given Theorem 3.1. Here, the prediction of weights of a layer is supplanted by the prediction of an activation vector that is introduced into the network by element-wise multiplication.

Theorem 3.3 gives us a different point of view on what HyperNetworks are. Because factorization machines are, on the one hand, a generalization of rank-k matrix factorization and a special case of a single HyperNetwork layer, we can now see (multi-layer) HyperNetworks as a kind of (iterated) generalized rank-k matrix factorization.

5.1. Related Work

5.1.1. Networks with gating layers

Gating layers are a common architectural component of neural networks, probably popularized by LSTMs (Hochreiter & Schmidhuber, 1997). Some recent papers have suggested variants of gating layers as an alternative to predicting full weight or attention matrices, e.g. gMLP (Liu et al., 2021) to replace multi-headed attention (Vaswani et al., 2017) and modulated periodic activations (Mehta et al., 2021) to replace HyperNet-SIRENs (but without theoretical motivation).

Our results provide a firm theoretical foundation for such architectures: In an appropriate dimension, gating and weight prediction are equivalent; in a lower dimension, gating is a low-rank weight prediction.

5.1.2. HyperNetworks in Computer Vision

Many recent architectures contain a weight (or kernel) prediction step similar to the KPN we studied in detail in this paper. A brief overview of related works is given in Table 2, and a more complete explanation of these architectures is given in the Appendix A.5. Note that many further examples exist. The empirical success and popularity of predicting weights evident from this table underlines the relevance of our theoretical findings.

5.1.3. Relating HyperNetworks to other architectures

(Schlag et al., 2021) show that linearized multi-headed attention is a special kind of HyperNetwork but do not relate this to gating mechanisms. In our definition of HyperNetwork layers, we allow non-linearities (such as the softmax of multi-headed attention); notably, this means that the con-

[3] https://github.com/google/burst-denoising

Table 1. 8-frame-combining KPN compared with the same setup, where a gated double layer replaces the predicted kernel layer as in Corollary 3.2. Max. Size indicates the total size of all feature maps in the largest layer.

Model	Max. Size	Test PSNR
Single Frame	$1 \times H \times W$	20.0
Original KPN (Mildenhall et al., 2018)	$200 \times H \times W$	**32.6**
Gated-128 (ours)	$128 \times H \times W$	**32.8**
Gated-64 (ours)	$64 \times H \times W$	31.9

Table 2. Different HyperNetworks (with a focus on Computer Vision) and their properties. We include the FM as the simplest possible HyperNetwork. The 'Conv.' column marks networks containing convolution layers, and 'Local' networks apply spatially varying kernels locally at each pixel. 'FFL' abbreviates 'feed-forward layer'. See the appendix for further details on these architectures.

Model	Conv.	Local	1 FFL	specific NN^W	Initial Application
HyperNetwork (Schmidhuber, 1992; Ha et al., 2016)					time series
Dynamic Convolution, e.g. (Klein et al., 2015)	×				img. enhancement
Dynamic Alignment Net (Bohle et al., 2021)	×				img. classification
Dynamic Filter Net (Jia et al., 2016)	×	×			img. enhancement
Adaptive Convolution, e.g. (Niklaus et al., 2017)	×	×	×		video enhancement
HDR-Net (Gharbi et al., 2017)	×	×	×	×	img. enhancement
CondConv (Yang et al., 2019)	×			×	img. classification
Squeeze-And-Excitation (Hu et al., 2018)	×			×	img. classification
FM (Rendle, 2010)			×	×	recommendation
MHA (Vaswani et al., 2017)				×	NLP

struction of Theorem 3.1 applies to multi-headed attention mechanisms as well (in as far as a suitable low-rank basis can be constructed, see also Sec. 5.2).

(Littwin et al., 2020) characterize infinite-width HyperNetworks and compute their GP and NTK kernels.

(Jayakumar et al., 2020) propose a hierarchy of *multiplicative interactions*. Our results show how the simplest layer (gating) of this hierarchy can emulate the most general (HyperNetwork) with a simple construction (the GDL). This means that this hierarchy is flat in practice and that weight prediction is *not* more powerful than gating, as long as the gating is combined with fixed weight layers (which in practice always is the case). In the appendix, we also highlight the relationship between the Multiplicative Interaction layer of (Jayakumar et al., 2020) and Factorization Machines.

5.2. Limitations

In the categorization of (Smith et al., 2022), we see two limitations of our work. Firstly, relating to generalizability: The idea of low-rank kernel prediction applies to all ConvNets and is simple to generalize to further ConvNets in a useful way, but e.g., generalization to MHA-based architectures is not trivial because it is not straightforward to construct a useful low-rank basis for the MHA-layer (efficient MHA alternatives are a field of research in their own right, see e.g., Sec. 2.5.1 of (Wan et al., 2023)).

Secondly, regarding robustness: Our proof of Theorem 3.1 shows how to construct by hand a GDL that matches a HN, and we show a practically useful application with KPNs; we do not have a proof that SGD will *always* be able to learn a good GDL when a good HN can be learned with SGD (though this limitation is very wide-spread in deep learning).

6. Conclusion

This paper introduces two new equivalences between seemingly distinct machine learning models: Firstly, gating layers combined with linear, fixed-weight layers can implement dynamically predicted weights, and further, FMs (a generalization of low-rank matrix factorization) are a restricted kind of HyperNetwork. We show empirically that gating layers can learn to emulate a given set of predicted weights with gradient descent to greater accuracy than comparably sized neural networks with fully connected layers without gating. A practical, memory-saving application of this to Kernel Prediction Networks using only static weights and element-wise multiplication (rather than a dynamic local filter) is also given.

Importantly, beyond this practical application, our results lend theoretical support to the empirical success of some recently popular architecture choices (Liu et al., 2021; Mehta et al., 2021; Chen et al., 2022) and our work revises the hierarchy of multiplicative interactions (Jayakumar et al.,

2020) as flat, when fixed weight layers are involved.

Impact Statement

This paper presents work that aims to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Bahdanau, D., Cho, K., and Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, May 2016.

Bako, S., Vogels, T., Mcwilliams, B., Meyer, M., NováK, J., Harvill, A., Sen, P., Derose, T., and Rousselle, F. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *ACM Transactions on Graphics*, 36 (4):1–14, July 2017. ISSN 0730-0301, 1557-7368. doi: 10.1145/3072959.3073708.

Bohle, M., Fritz, M., and Schiele, B. Convolutional Dynamic Alignment Networks for Interpretable Classifications. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10029–10038. IEEE, 2021.

Chen, L., Chu, X., Zhang, X., and Sun, J. Simple Baselines for Image Restoration. *arxiv:2204.04676*, August 2022. doi: 10.48550/arXiv.2204.04676.

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

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021.

Gharbi, M., Chen, J., Barron, J. T., Hasinoff, S. W., and Durand, F. Deep bilateral learning for real-time image enhancement. *ACM Transactions on Graphics*, 36(4): 118:1–118:12, July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073592.

Ha, D., Dai, A., and Le, Q. V. HyperNetworks. *arXiv:1609.09106 [cs]*, December 2016.

Han, Y., Huang, G., Song, S., Yang, L., Wang, H., and Wang, Y. Dynamic Neural Networks: A Survey, December 2021.

Hinton, G. E. Using fast weights to deblur old memories. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 177–186. Erlbaum, 1987.

Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324. IEEE, 2019.

Hu, J., Shen, L., and Sun, G. Squeeze-and-Excitation Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141. IEEE, 2018.

Jayakumar, S. M., Czarnecki, W. M., Menick, J., Schwarz, J., Rae, J., Osindero, S., Teh, Y. W., Harley, T., and Pascanu, R. Multiplicative Interactions and Where to Find Them. In *International Conference on Learning Representations*, March 2020.

Jia, X., De Brabandere, B., Tuytelaars, T., and Gool, L. V. Dynamic Filter Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization, January 2017.

Klein, B., Wolf, L., and Afek, Y. A Dynamic Convolutional Layer for Short Range Weather Prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4840–4848. IEEE, 2015.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kupfermann, I. Modulatory Actions of Neurotransmitters. *Annual Review of Neuroscience*, 2(1):447–465, 1979. doi: 10.1146/annurev.ne.02.030179.002311.

Lipton, Z. C., Berkowitz, J., and Elkan, C. A Critical Review of Recurrent Neural Networks for Sequence Learning, October 2015.

Littwin, E., Galanti, T., Wolf, L., and Yang, G. On Infinite-Width Hypernetworks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13226–13237. Curran Associates, Inc., 2020.

Liu, H., Dai, Z., So, D., and Le, Q. V. Pay Attention to MLPs. In *Advances in Neural Information Processing Systems*, volume 34, pp. 9204–9215. Curran Associates, Inc., 2021.

Loshchilov, I. and Hutter, F. SGDR: Stochastic Gradient Descent with Warm Restarts, May 2017.

Lydia, A. A. and Francis, F. S. Adagrad - An Optimizer for Stochastic Gradient Descent. *Int. J. Inf. Comput. Sci.*, 6 (0972):3, 2019.

Mehta, I., Gharbi, M., Barnes, C., Shechtman, E., Ramamoorthi, R., and Chandraker, M. Modulated Periodic Activations for Generalizable Local Functional Representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14214–14223. IEEE, 2021.

Mildenhall, B., Barron, J. T., Chen, J., Sharlet, D., Ng, R., and Carroll, R. Burst Denoising With Kernel Prediction Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2502–2510. IEEE, 2018.

Muller, L. K. Overparametrization of HyperNetworks at Fixed FLOP-Count Enables Fast Neural Image Enhancement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 284–293. IEEE, 2021.

Niklaus, S., Mai, L., and Liu, F. Video Frame Interpolation via Adaptive Separable Convolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 261–270. IEEE, 2017.

Parikh, A. P., Täckström, O., Das, D., and Uszkoreit, J. A Decomposable Attention Model for Natural Language Inference. *arXiv:1606.01933 [cs]*, September 2016.

Rendle, S. Factorization Machines. In *2010 IEEE International Conference on Data Mining*, pp. 995–1000. IEEE, December 2010. doi: 10.1109/ICDM.2010.127.

Riegler, G., Schulter, S., Ruther, M., and Bischof, H. Conditioned Regression Models for Non-Blind Single Image Super-Resolution. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 522–530, 2015.

Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015.

Schlag, I., Irie, K., and Schmidhuber, J. Linear Transformers Are Secretly Fast Weight Programmers. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 9355–9366. PMLR, July 2021.

Schmidhuber, J. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, 4(1):131–139, January 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.1.131.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit Neural Representations with Periodic Activation Functions. In *Advances in Neural Information Processing Systems*, volume 33, pp. 7462–7473. Curran Associates, Inc., 2020.

Smith, J. J., Amershi, S., Barocas, S., Wallach, H., and Wortman Vaughan, J. Real ml: Recognizing, exploring, and articulating limitations of machine learning research. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pp. 587–597, 2022.

Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., and Jiang, P. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, pp. 1441–1450, New York, NY, USA, November 2019. Association for Computing Machinery. ISBN 978-1-4503-6976-3. doi: 10.1145/3357384.3357895.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Wan, Z., Wang, X., Liu, C., Alam, S., Zheng, Y., Qu, Z., Yan, S., Zhu, Y., Zhang, Q., Chowdhury, M., et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 2023.

Xia, Z., Perazzi, F., Gharbi, M., Sunkavalli, K., and Chakrabarti, A. Basis Prediction Networks for Effective Burst Denoising With Large Kernels. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11841–11850, Seattle, WA, USA, June 2020. IEEE. ISBN 978-1-72817-168-5. doi: 10.1109/CVPR42600.2020.01186.

Yang, B., Bender, G., Le, Q. V., and Ngiam, J. CondConv: Conditionally Parameterized Convolutions for Efficient Inference. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Zhang, L., Wu, X., Buades, A., and Li, X. Color demosaicking by local directional interpolation and nonlocal adaptive thresholding. *Journal of Electronic Imaging*, 20 (2):023016, April 2011. ISSN 1017-9909, 1560-229X. doi: 10.1117/1.3600632.

Zhang, Y., Li, K., Li, K., Wang, L., Zhong, B., and Fu, Y. Image Super-Resolution Using Very Deep Residual Channel Attention Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 286–301, 2018.

A. Appendix

A.1. Proof Theorem 1

Proof. In detail, we set

$$v_{ik} = \begin{cases} 1 & \text{if } k \bmod n_i = i \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

This will copy the input neurons i exactly j times if $n_k = n_j \cdot n_i$. Further, we set

$$u_{kj} = \begin{cases} 1 & \text{if } k//n_i = j \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

where $//$ denotes integer division. This will sum n_j neuron groups of size n_i (in the hidden layer) into a single output neuron. When $n_k > n_j \cdot n_i$ we set terms v_{ik}, u_{kj} where $k > n_j \cdot n_i$ to zero.

In between the application of these two indexing matrices, the element-wise multiplication with a_k occurs. We set

$$a_k = w_{ij} \text{ where } k = n_i \cdot j + i \tag{13}$$

In programming parlance, a_k is a reshaped w_{ij}.

Inserting the given values for v_{ik}, u_{kj} and a_k into Eq. 5 recovers Eq. 1 and verifies the theorem. □

This proof is very simple; the key contribution here is considering the combination of gating and fixed-weight layers.

A.2. Proof Theorem 2

Proof. The proof can be given by rearranging the terms in the corresponding definitions.

$$\begin{aligned} y_j &= w_j + \sum_{i=0}^{n-1} w_{ji} x_i + \sum_{i=0}^{n-1} \sum_{k=i+1}^{n-1} \left(\sum_{l=0}^{n_l - 1} v_{jil} v_{jkl} \right) x_i x_k \\ &= w_j + \sum_{i=0}^{n-1} \left(w_{ji} + \sum_{k=i+1}^{n-1} \left(\sum_{l=0}^{n_l - 1} v_{jil} v_{jkl} \right) x_k \right) x_i \\ &= \sum_{i=0}^{n-1} x_i \underbrace{\left(w_{ji} + \sum_{k=i+1}^{n-1} u_{jik}^{\text{rank-}n_l} x_k \right)}_{w_{ji}^{\text{eff}}} + w_j \end{aligned} \tag{14}$$

When we compare this to the Eq. 8 we see that the summation indices in the inner sum do not match. This means 1) each term where $i \neq k$ occurs twice and 2) terms where $i = k$ do occur. Point 1) can be taken care of by re-scaling weights by a factor of two. Point 2) is the reason to require binary inputs; for binary inputs, the element-wise square is equal to the input itself, and these additional terms can be canceled by adjusting w_{ji}. □

A.3. Layout KPNs

Figs. 4, 5 show the layouts of our variant and the original KPN network.

A.4. Layout of the feature extraction ConvNet for CIFAR experiments

See Fig. 6

A.5. Formalized Comparison of some HyperNetwork Variants

This section formally compares different HyperNetwork variants with a focus on computer vision applications (in the NLP domain, many variants of attention layers exist that lie outside the scope of this paper). The motivation for this section is to simplify and unify published definitions. Because HyperNetworks are developed somewhat independently in various communities, the disparate notations used by the different authors obfuscate similarities that we want to highlight.

For simplicity, we omit the sample or 'batch' dimension – in most practical implementations, there is such a dimension to make use of data parallelism. Note that contrary to fixed weight matrices, predicted weights cannot be shared across a mini-batch of samples because a different weight may be predicted for each input.

For completeness, we repeat the definitions given in the main text.

A.5.1. Definition HyperNetwork layer

Let a HyperNetwork (Ha et al., 2016; Schmidhuber, 1992) layer be a neural network layer whose weights have been predicted by a 'weight-prediction' neural network $\text{NN}^W(\cdot)$

$$x_j^l = f\left(\sum_i [\text{NN}^W(\vec{x}^0)]_{ij} \cdot x_i^{l-1} \right) \tag{15}$$

where $\vec{x}^0$ is the input, $\vec{x}^l$ is the lth layer's activation, and $f(\cdot)$ is the non-linearity.

A.5.2. Factorization Machines, the simplest HyperNetwork

For the case of binary inputs $\vec{x}^0$, a single, (low-rank) linear HyperNetwork layer with linear NN^W is equivalent to a factorization machine (Rendle, 2010) (and thereby to a low-

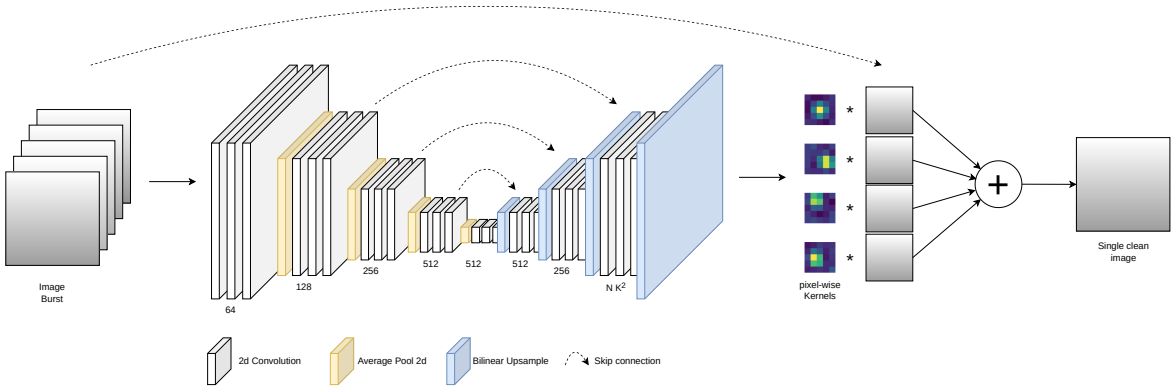

Figure 4. The original KPN for comparison to Fig. 5.

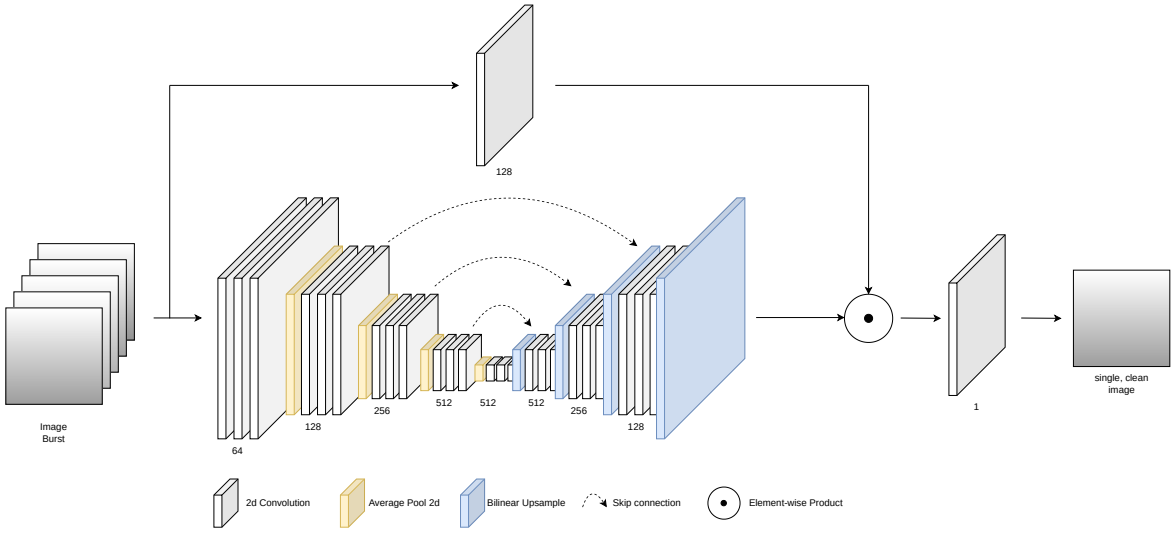

Figure 5. Example of a practical application of Theorem 3.1. A kernel prediction network, where the kernel prediction is replaced with a gated double layer. Here, we assume a single output color channel and treat the time dimension of the input burst as channels.

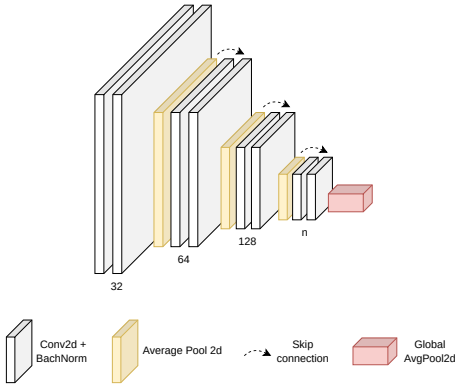

Figure 6. Layout of the CNN used for feature extraction in the CIFAR experiments in Sec. 4. The number of output features n varies in the different experiments.

rank matrix factorization).

$$x_j^1 = \sum_i [\text{Linear}^W(\vec{x}^0)]_{ij} \cdot x_i^0 \tag{16}$$

(Jayakumar et al., 2020) propose a multiplicative interaction (MI) layer, that is, a full-rank, linearized HyperNetwork layer. In contrast to the factorization machines the MI layer 1) does not concatenate its inputs, but keeps them distinct 2) expresses the full 3d interaction tensor (i.e. some product terms occur twice and self-interactions are included) and 3) models the interactions at full-rank.

A.5.3. Sequence-based HyperNetworks

In sequence-based networks (e.g. in NLP), the input operand X is two-dimensional with entries $x_{i,s}$, where i is a neuron ID or channel dimension and s is a position in a sequence.

Fast-weight attention In this notation, the attention mechanism of (Bahdanau et al., 2016) can be written as

$$y_{i,t} = \sum_s [\text{NN}^W(X^0)]_{st} \cdot x_{i,s} \tag{17}$$

The output of $[\text{NN}^W(X^0)]_{st}$ is called the attention matrix. The difference to the HyperNetwork layer in Eq. 1 is that here the attention matrix mediates between sequence dimensions rather than channel dimensions.

Outer-product attention In dot-product attention mechanisms, like in (Parikh et al., 2016), the attention matrix is predicted at rank 1 (as also suggested by (Schmidhuber, 1992)):

$$y_{i,t} = \sum_s f_{\text{NL}}\left([\text{NN}_a^W(X^0)]_t \cdot [\text{NN}_b^W(X^0)]_s\right) \cdot x_{i,s} \tag{18}$$

where NN_a^W and NN_b^W are two (potentially overlapping) weight prediction networks and $f_{\text{NL}}(\cdot)$ is a simple function (e.g. the identity or a softmax over one of the sequence dimensions). When $f_{\text{NL}}(\cdot)$ is not the identity, the matrix it outputs can have a rank greater than 1.

Multi-headed attention In multi-headed attention, several outer-product attention layers are concatenated. A compact way of writing this is as above, with an additional 'head' dimension indexed by h

$$y_{i,t,h} = \sum_s f_{\text{NL}}\left([\text{NN}_a^W(X^0)]_{t,h} \cdot [\text{NN}_b^W(X^0)]_{s,h}\right) \cdot x_{i,s} \tag{19}$$

This head dimension is finally transposed into the channel dimension by concatenation, i.e.

$$y_{i+h\cdot n_i,t} = \sum_s f_{\text{NL}}\left([\text{NN}_a^W(X^0)]_{t,h} \cdot [\text{NN}_b^W(X^0)]_{s,h}\right) \cdot x_{i,s} \tag{20}$$

Multi-headed attention is commonly used in transformers (Vaswani et al., 2017) and seems particularly effective at balancing low-rank constraints (due to memory limitations) and expressiveness. Some definitions of MHA include linear layers acting on $x_{i,s}$; here, we treat them separately. Further, some definitions of MHA require specific forms of NN_a, NN_b, e.g. an affine transformation of $x_{i,s}$.

A.5.4. Convolutional HyperNetworks

In (2d) convolutional HyperNetworks, the input operand X is three dimensional with entries $x_{j,rs}$, where j is the channel dimension and r, s are two spatial dimensions.

Dynamic convolution The convolutional HyperNetwork layer then reads

$$x_{j,rs}^l = f\left(\sum_{i,d_r,d_s} [\text{NN}^W(X^0)]_{ij,d_r d_s} \cdot x_{i,(r+d_r)(s+d_s)}^{l-1}\right) \tag{21}$$

where a multi-channel image / feature map X has entries $x_{j,rs}$. The indexes i, j index channels, the r, s are spatial coordinates and dr, ds are offsets in these spatial coordinates. The sum now goes over all input channels and *offsets* (d_r, d_s) in the spatial dimension (i.e. it is a convolution with kernel-size (d_r, d_s) over the spatial dimensions). The weight prediction network $[\text{NN}^W(X^0)]_{ij,d_r d_s}$ predicts full convolution kernels whose dimensions are input channels i, output channels j, spatial dim. 1 offset d_r and spatial dim. 2 offset d_s.

This definition corresponds to (Klein et al., 2015) and is known as a dynamic convolution.

The dynamic alignment net (Bohle et al., 2021) finds an interesting application of such layers: By using many predicted weights after each other, *without non-linearity in between*, a more interpretable network can be constructed

(because for any input sample, the feed-forward network is linear, though this linear network may be different for each sample).

Dynamic filter network The convolutional HyperNetwork layer with *local* filters reads

$$x^l_{j,rs} = f\left(\sum_{i,d_r,d_s} [\text{NN}^W(X^0)]_{ij,sr,d_r d_s} \cdot x^{l-1}_{i,(r+d_r)(s+d_s)}\right) \tag{22}$$

This differs from the above in that the weight prediction network now outputs different filter for each spatial location, such that its output is six-dimensional: i input channels, j output channels, s and r spatial dimensions and d_s, d_r spatial dimension offsets.

This definition corresponds to (Zhang et al., 2018) and is known as a dynamic filter network.

Adaptive Convolution and Kernel Prediction Networks This (Niklaus et al., 2017; Bako et al., 2017) is the single feed-forward layer variant of the previous

$$x^{\text{out}}_{j,rs} = f\left(\sum_{i,d_r,d_s} [\text{NN}^W(X^0)]_{ij,sr,d_r d_s} \cdot x^{0}_{i,(r+d_r)(s+d_s)}\right) \tag{23}$$

Having only a single layer in the feed-forward network can be desirable to constrain the complexity of the transformation that is applied to any given input.

Deep Bilateral Learning This (Gharbi et al., 2017) is a simplified special case of the previous, namely

$$x^{\text{out}}_{j,rs} = \sum_i [\text{NN}^W(X^0)]_{ij,sr} \cdot x^0_{i,rs} \tag{24}$$

where the spatial offsets d_s, d_r are always zero (i.e. a 1x1 convolution), and there is no non-linearity in the feed-forward net. Importantly, NN^W has a special form including a bilateral grid (see (Gharbi et al., 2017)). This NN^W can be computed efficiently and yet is expressive.

Squeeze-And-Excitation Layer This (Hu et al., 2018) is a special case of the general dynamic convolution, where the predicted weight is decomposed into the product of fixed weight and a channel-wise dynamic component. The predicted weight reads

$$W^{\text{eff}}_{ij,d_r d_s} = \text{MLP}[\text{Pool}(\text{NN}^W(X^0))]_i \cdot W_{ij,d_r d_s} \tag{25}$$

and the resulting restricted dynamic convolution is

$$x^l_{j,rs} = f\left(\sum_{i,d_r,d_s} W^{\text{eff}}_{ij,d_r d_s} \cdot x^{l-1}_{i,(r+d_r)(s+d_s)}\right) \tag{26}$$

CondConv The approach of Yang and colleagues (Yang et al., 2019) is comparable to the squeeze-and-excitation layer, where the predicted weight for the CondConv reads

$$W^{\text{eff}}_{ij,d_r d_s} = \sum_k \text{MLP}[\text{Pool}(\text{NN}^W(X^0))]_k \cdot W_{kij,d_r d_s} \tag{27}$$

i.e., it is computed as a weighted sum of several weights.